# Enterprise PL/I for z/OS 6.1

# 言語解説書

SC31-5716-00

# 目次


**表 ..... xxiii**

**図 ..... xxvii**

**Enterprise PL/I for z/OS 言語解説書 ..... xxix**
第 1 版 (2022 年 5 月) ..... xxix

**本書について ..... xxxi**
本書で用いられる表記規則 ..... xxxi
セマンティクス ..... xxxiii
工業規格 ..... xxxiv
変更の要約 ..... xxxiv
本リリースにおける機能強化 ..... xxxiv
5.3 からの機能拡張 ..... xxxv
5.2 からの機能拡張 ..... xxxviii
5.1 からの機能拡張 ..... xl
4.5 からの機能拡張 ..... xlii
4.4 からの機能拡張 ..... xlii
4.3 からの機能拡張 ..... xliii
4.2 からの機能拡張 ..... xliii
4.1 からの機能拡張 ..... xliv
3.9 からの機能拡張 ..... xliv
3.8 からの機能拡張 ..... xliv
3.7 からの機能拡張 ..... xlv
3.6 からの機能拡張 ..... xlvi
3.5 からの機能拡張 ..... xlvi
3.4 からの機能拡張 ..... xlvi
3.3 からの機能拡張 ..... xlvi
3.2 からの機能拡張 ..... xlvi
3.1 からの機能拡張 ..... xlvii
ご意見の送付方法 ..... xlix
アクセシビリティー ..... xlix

**第 1 章プログラムのエレメント ..... 1**
1 バイト文字セット ..... 1
10 進数 ..... 2
2 進数 ..... 3
16 進数 ..... 3
特殊文字 ..... 3
複合記号 ..... 4
大文字小文字の区別 ..... 5
SBCS ステートメント・エレメント ..... 5
ステートメント ..... 8
単純ステートメント ..... 9
複合ステートメント ..... 10
グループ ..... 10
2 バイト文字セット ..... 11
DBCS ID ..... 11
DBCS ステートメント・エレメント ..... 12

DBCS 継続規則 13

**第 2 章データ・エレメント 15**
データ項目 15
変数 15
定数 16
引用符の使用 16
句読点のついた定数 16
データ・タイプとその属性 16
データ属性 17
非データ属性 17
計算データ・タイプとその属性 22
コード化算術データとその属性 22
ストリング・データとその属性 29
日付属性 40
プログラム制御のデータ・タイプとその属性 43
ラベル・データと LABEL 属性 43
フォーマット・データと FORMAT 属性 44
VARIABLE 属性 45
値の属性 45
VALUE 属性 45
名前付き定数 45
VALUELIST 属性 47
VALUELISTFROM 属性 47
VALUERANGE 属性 48

**第 3 章式および参照 51**
計算の順序 53
ターゲット 53
変数 53
疑似変数 53
中間結果 54
演算式 54
ハンドル演算 55
ポインター演算 55
算術演算 56
ビット演算子 63
比較演算 64
連結演算 66
各種演算の組み合わせ 68
演算子の優先順位 68
配列式 70
接頭演算子と配列 70
挿入演算子と配列 70
構造式 71
制限付き式 71

**第 4 章データ変換 73**
計算データ変換の組み込み関数 74
ストリングの長さの変換 74
算術値の精度の変換 75
モードの変換 75
その他のデータ属性の変換 76
ソースからターゲットへの変換規則 77
例 85
例: DECIMAL FIXED から小数部がある BINARY FIXED への変換 85
例: 算術からビット・ストリングへの変換 85

例: 算術から文字への変換 86
例: 変換エラー 86

**第 5 章プログラムの編成 87**
プログラム 87
プログラム構造 87
プログラムの活動化 88
プログラムの終了 88
ブロック 88
ブロックの活動化 89
ブロックの終了 89
パッケージ 90
プロシージャー 92
PROCEDURE ステートメント 93
ENTRY ステートメント 94
パラメーター属性 95
プロシージャーの活動化 97
プロシージャーの終了 98
再帰的プロシージャー 99
外部プロシージャーの動的ロード 100
サブルーチン 102
例 1 103
例 2 103
組み込みサブルーチン 104
関数 104
例 105
組み込み関数 106
プロシージャーへの引数の引き渡し 106
BYVALUE および BYADDR の使用 106
INONLY、INOUT、および OUTONLY の使用 107
仮引数 108
MAIN プロシージャーへの引数の引き渡し 109
開始ブロック 109
BEGIN ステートメント 109
開始ブロックの活動化 110
開始ブロックの終了 110
入り口データ 110
入り口定数 111
入り口変数 111
ENTRY 属性 112
OPTIONAL 属性 114
LIST 属性 116
LIMITED 属性 118
総称入り口 119
GENERIC 属性 119
入り口呼び出しまたは入り口値 121
CALL ステートメント 121
RETURN ステートメント 122
サブルーチンからの戻り 122
関数からの戻り 122
OPTIONS オプションとその属性 123
RETURNS オプションとその属性 132

**第 6 章タイプ定義 135**
ユーザー定義のタイプ (別名) 135
DEFINE ALIAS ステートメント 135
序数の定義 136

DEFINE ORDINAL ステートメント....136
タイプ付き構造体と共用体の定義....137
HANDLE 属性....139
タイプ変数の宣言....140
TYPE 属性....140
ORDINAL 属性....141
タイプ付き構造体修飾....141
ピリオド演算子 (.) の使用....142
配列およびタイプ付き構造体または共用体の組み合わせ....142
ハンドルの使用....143
序数の使用....143
タイプ付き関数....146

**第 7 章データ宣言....147**
明示宣言....147
DECLARE ステートメント....148
属性分配....149
暗黙宣言....149
宣言の有効範囲....150
INTERNAL 属性と EXTERNAL 属性....152
RESERVED 属性....155
FORCE 属性....156
SUPPRESS 属性....157
データの位置合わせ....158
ALIGNED 属性と UNALIGNED 属性....158
属性のデフォルト....165
言語に固有のデフォルト....165
DEFAULT ステートメント....166
言語に固有のデフォルトの復元....170
配列....170
次元属性....171
DIMACROSS 属性....172
配列の例....172
添え字....173
配列のクロスセクション....174
構造体および共用体....174
構造体....174
共用体....175
構造体および共用体の修飾....176
UNION への割り当て....178
LIKE 属性....178
INDFOR 属性....180
NOINIT 属性....181
NULLINIT 属性....181
XML 関連属性....181
JSON 関連属性....183
集合体の組み合わせおよびマッピング....183
配列、構造体、および共用体の組み合わせ....183
構造体または共用体の配列のクロスセクション....184
構造体および共用体の操作....184
構造体および共用体のマッピング....184

**第 8 章ステートメントとディレクティブ....193**
ALLOCATE ステートメント....193
ASSERT ステートメント....193
代入ステートメントと複合代入ステートメント....197
代入ステートメント....197

BY DIMACROSS オプションを使用する代入ステートメント ... 198
複合代入ステートメント ... 198
ターゲット変数 ... 199
割り当ての実行方法 ... 201
複数割り当て ... 203
例 ... 203
ATTACH ステートメント ... 205
BEGIN ステートメント ... 205
CALL ステートメント ... 205
CANCEL THREAD ステートメント ... 205
CLOSE ステートメント ... 205
DECLARE ステートメント ... 205
DEFAULT ステートメント ... 205
DEFINE ALIAS ステートメント ... 205
DEFINE ORDINAL ステートメント ... 206
DEFINE STRUCTURE ステートメント ... 206
DELAY ステートメント ... 206
DELETE ステートメント ... 206
DETACH ステートメント ... 206
DISPLAY ステートメント ... 207
DO ステートメント ... 207
タイプ 1 ... 208
タイプ 2 およびタイプ 3 ... 208
タイプ 4 ... 214
基本的な繰り返しの例 ... 215
WHILE、UNTIL を指定した DO ステートメントの例 ... 216
UPTHRU と DOWNTHRU を指定した DO の例 ... 217
REPEAT の例 ... 218
END ステートメント ... 218
ENTRY ステートメント ... 219
EXIT ステートメント ... 219
FETCH ステートメント ... 219
FLUSH ステートメント ... 219
FORMAT ステートメント ... 219
FREE ステートメント ... 220
GET ステートメント ... 220
GO TO ステートメント ... 220
IF ステートメント ... 221
例 ... 222
短絡計算 ... 222
%INCLUDE ディレクティブ ... 223
ITERATE ステートメント ... 223
LEAVE ステートメント ... 224
%LINE ディレクティブ ... 225
LOCATE ステートメント ... 225
%NOPRINT ディレクティブ ... 225
%NOTE ディレクティブ ... 225
ヌル・ステートメント (null statement) ... 226
ON ステートメント ... 226
OPEN ステートメント ... 226
OTHERWISE ステートメント ... 226
PACKAGE ステートメント ... 226
%PAGE ディレクティブ ... 227
%POP ディレクティブ ... 227
%PRINT ディレクティブ ... 227
PROCEDURE ステートメント ... 227
%PROCESS ディレクティブ ... 227
*PROCESS ディレクティブ ... 228

%PROCINC ディレクティブ....228
*PROCINC ディレクティブ....228
%PUSH ディレクティブ....228
PUT ステートメント....229
QUALIFY ステートメント....229
READ ステートメント....230
REINIT ステートメント....230
RELEASE ステートメント....230
RESIGNAL ステートメント....230
RETURN ステートメント....230
REVERT ステートメント....230
REWRITE ステートメント....230
SELECT ステートメント....231
SIGNAL ステートメント....232
%SKIP ディレクティブ....232
STOP ステートメント....232
WAIT ステートメント....233
WHEN ステートメント....233
WRITE ステートメント....233
%XINCLUDE ステートメント....233
XDECLARE ステートメント....233
XDEFINE ALIAS ステートメント....233
XDEFINE ORDINAL ステートメント....234
XDEFINE STRUCTURE ステートメント....234
XPROCEDURE ステートメント....234

**第 9 章ストレージ制御....235**
ストレージのクラス、割り振り、および割り振り解除....235
静的ストレージとその属性....236
自動ストレージとその属性....237
被制御ストレージとその属性....238
被制御変数のための ALLOCATE ステートメント....238
被制御変数のための FREE ステートメント....240
被制御変数の複数世代....240
アスタリスクの表記法....241
調節可能エクステント....241
被制御変数のための組み込み関数....241
基底付きストレージとその属性....242
BASED 宣言でのエクステント指定....242
BASED VARYING ストリング....243
BASED 変数のストレージ割り振り....243
ロケーター変数....244
DEFINED 属性および UNION 属性....244
INITIAL 属性....244
ロケーター・データ....244
POINTER 変数とその属性....247
基底付き変数のための組み込み関数....247
基底付き変数のための ALLOCATE ステートメント....248
基底付き変数のための FREE ステートメント....249
REFER オプション (自己定義データ)....249
区域データとその属性....251
オフセット・データとその属性....252
LOCATES 属性....253
区域変数のための組み込み関数....255
区域の割り当て....255
区域の入出力....256
リスト処理....256

ASSIGNABLE 属性および NONASSIGNABLE 属性....257
NORMAL 属性および ABNORMAL 属性....258
BIGENDIAN 属性および LITTLEENDIAN 属性....258
HEXADEC 属性と IEEE 属性....259
CONNECTED 属性と NONCONNECTED 属性....260
DEFINED 属性および POSITION 属性....261
非連結ストレージ....262
単純定義....262
iSUB 定義....263
ストリング・オーバーレイ定義....264
POSITION 属性....265
INITIAL 属性....265
配列変数の初期設定....269
共用体の初期設定....270
静的変数の初期設定....270
自動変数の初期設定....271
基底付き変数と被制御変数の初期設定....271
例....271

**第 10 章入出力....273**
データ・セット....274
連続....274
索引....274
Relative....274
領域内....274
ファイル....274
FILE 属性....275
RECORD 属性と STREAM 属性....278
INPUT、OUTPUT、および UPDATE 属性....278
SEQUENTIAL 属性と DIRECT 属性....279
BUFFERED 属性および UNBUFFERED 属性....279
ENVIRONMENT 属性....280
KEYED 属性....280
PRINT 属性....280
ファイルのオープンとクローズ....280
OPEN ステートメント....281
暗黙オープン....282
CLOSE ステートメント....284
FLUSH ステートメント....285
SYSPRINT および SYSIN....285

**第 11 章レコード単位データ伝送....287**
データ伝送....287
位置合わせされていないビット・ストリング....287
可変長ストリング....287
区域変数....288
データ伝送ステートメント....288
READ ステートメント....288
WRITE ステートメント....288
REWRITE ステートメント....289
LOCATE ステートメント....289
DELETE ステートメント....289
データ伝送ステートメントのオプション....290
FILE オプション....290
FROM オプション....290
IGNORE オプション....291
INTO オプション....291

KEY オプション... 291
KEYFROM オプション... 291
KEYTO オプション...292
SET オプション... 292
処理モード... 293
移動モード...293
位置指定モード... 293

**第 12 章ストリーム指向データ伝送...295**
データ伝送ステートメント...295
GET ステートメント...295
PUT ステートメント...296
データ伝送ステートメントのオプション... 296
COPY オプション...296
データ指定オプション... 297
FILE オプション... 299
LINE オプション...299
PAGE オプション...299
SKIP オプション...299
STRING オプション...300
データ・リスト項目の伝送...301
データ・ディレクティブのデータ指定... 302
データ・ディレクティブ・データの制限...302
データ・ディレクティブ・データの構文...303
GET データ・ディレクティブ...303
PUT データ・ディレクティブ...305
編集ディレクティブ・データ指定... 306
GET 編集ディレクティブ... 308
PUT 編集ディレクティブ... 308
FORMAT ステートメント... 309
リスト・ディレクティブ・データ指定...310
リスト・ディレクティブ・データの構文...310
GET リスト・ディレクティブ...311
PUT リスト・ディレクティブ...312
PRINT 属性...312
ストリーム入出力での DBCS データ...313

**第 13 章編集ディレクティブのフォーマット項目... 315**
A フォーマット項目...315
B フォーマット項目...315
C フォーマット項目...316
COLUMN フォーマット項目...316
E フォーマット項目... 317
F フォーマット項目... 319
G フォーマット項目...320
L フォーマット項目...321
LINE フォーマット項目... 321
P フォーマット項目... 321
PAGE フォーマット項目... 322
R フォーマット項目...322
SKIP フォーマット項目... 322
V フォーマット項目...323
X フォーマット項目...323

**第 14 章ピクチャー指定文字... 325**
ピクチャー反復因数...325
文字データ用のピクチャー文字... 325

数字データ用のピクチャー文字.....326
数字と小数点.....327
ゼロ消去.....328
挿入文字.....329
通貨記号の定義.....331
符号および通貨記号の使用.....332
貸方記号、借方記号、オーバーパンチ、およびゼロ置き換え文字.....334
指数文字.....336
位取り係数.....337

**第 15 章条件処理.....339**
条件接頭語.....339
条件接頭語の有効範囲.....341
OPTIMIZATION によって起こる条件.....341
ON ユニット.....341
ON ステートメント.....341
ヌル ON ユニット.....342
ON ユニットの有効範囲.....343
動的に派生した ON ユニット.....343
ファイル変数のための ON ユニット.....343
REVERT ステートメント.....344
SIGNAL ステートメント.....345
RESIGNAL ステートメント.....345
複数条件.....345
CONDITION 属性.....346

**第 16 章条件.....347**
ANYCONDITION 条件.....347
AREA 条件.....348
ASSERTION 条件.....349
ATTENTION 条件.....349
CONDITION 条件.....350
CONFORMANCE 条件.....351
CONVERSION 条件.....351
ENDFILE 条件.....353
ENDPAGE 条件.....353
ERROR 条件.....354
FINISH 条件.....355
FIXEDOVERFLOW 条件.....355
INVALIDOP 条件.....356
KEY 条件.....357
NAME 条件.....357
OVERFLOW 条件.....358
RECORD 条件.....358
SIZE 条件.....359
STORAGE 条件.....360
STRINGRANGE 条件.....361
STRINGSIZE 条件.....361
SUBSCRIPTRANGE 条件.....362
TRANSMIT 条件.....362
UNDEFINEDFILE 条件.....363
UNDERFLOW 条件.....364
ZERODIVIDE 条件.....365

**第 17 章マルチスレッド化機能.....367**
スレッドの作成.....367
ATTACH ステートメント.....368

スレッドの終了....369
スレッドの取り消し....369
スレッドの完了の待機....369
スレッドの切り離し....370
条件処理....370
タスク・データとタスク属性....370
THREADID 組み込み関数....370
スレッド間のデータの共用....371
スレッド間のファイルの共用....371

**第 18 章組み込み関数、疑似変数、およびサブルーチン....373**
組み込み関数、疑似変数、および組み込みサブルーチンの宣言および呼び出し....373
BUILTIN 属性....373
組み込み関数および疑似変数の呼び出し....373
組み込みサブルーチンの呼び出し....374
組み込み関数、疑似変数、および組み込みサブルーチンの引数の指定....374
引数の集合....374
ヌル引数およびオプション引数....375
数学関数の正確度....375
組み込み関数のカテゴリー....375
算術組み込み関数....375
配列処理組み込み関数....376
バッファー管理組み込み関数....377
条件処理組み込み関数....379
日付/時刻組み込み関数....380
浮動小数点の照会組み込み関数....383
浮動小数点演算組み込み関数....384
入出力組み込み関数....384
整数演算組み込み関数....385
JSON 組み込み関数....385
数学組み込み関数....386
その他の組み込み関数....387
序数処理組み込み関数....389
精度処理組み込み関数....389
疑似変数....390
ストレージ制御組み込み関数....391
ストリング処理組み込み関数....393
サブルーチン....395
個々の組み込み関数、疑似変数、およびサブルーチンの説明....397
ABS....397
ACOS....398
追加....398
ADDR....398
ADDRDATA....399
すべて....399
ALLCOMPARE....399
ALLOC31....400
ALLOCATE....400
ALLOCATION....401
ALLOCNEXT....401
ALLOCSIZE....401
ANY....402
ASIN....402
ATAN....402
ATAND....402
ATANH....403
AUTOMATIC....403

AVAILABLEAREA..... 403
BASE64DECODE.....404
BASE64DECODE8..... 404
BASE64DECODE16..... 405
BASE64ENCODE..... 405
BASE64ENCODE8..... 406
BASE64ENCODE16..... 407
BETWEEN..... 408
BETWEENEXCLUSIVE..... 408
BETWEENLEFTEXCLUSIVE..... 409
BETWEENRIGHTEXCLUSIVE..... 409
BINARY.....409
BINARYVALUE..... 410
BINSEARCH.....410
BINSEARCHX.....411
ビット..... 411
BITLOCATION.....412
BOOL.....412
バイト..... 412
BYTELENGTH.....413
CDS..... 413
CEIL..... 414
CENTERLEFT..... 414
CENTRELEFT..... 414
CENTERRIGHT..... 415
CENTRERIGHT..... 415
CHARACTER..... 415
CHARGRAPHIC.....416
CHARVAL..... 417
CHECKSTG.....417
CHECKSUM.....418
CODEPAGE.....418
COLLATE..... 418
COLLAPSE.....418
COMPARE..... 419
COMPLEX.....419
CONJG..... 420
COPY.....420
COS..... 421
COSD.....421
COSH.....421
COUNT..... 421
CS.....421
CURRENTSIZE.....423
CURRENTSTORAGE.....423
DATAFIELD..... 424
DATE..... 424
DATETIME.....424
DAYS..... 424
DAYSTODATE.....425
DAYSTOMICROSECS..... 426
DAYSTOSECS.....426
DECIMAL.....426
DIMENSION.....427
DIVIDE.....427
EDIT.....427
EMPTY..... 428
ENDFILE..... 428

ENTRYADDR..... 429
ENTRYADDR 疑似変数..... 429
EPSILON..... 429
ERF..... 429
ERFC..... 429
EXP..... 430
EXPONENT..... 430
FILEDDINT..... 430
FILEDDTEST..... 431
FILEDDWORD..... 431
FILEID..... 432
FILENEW..... 432
FILEOPEN..... 433
FILEREAD..... 433
FILESEEK..... 433
FILETELL..... 434
FILEWRITE..... 434
固定..... 434
FIXEDBIN..... 435
FIXEDDEC..... 435
FLOAT..... 436
FLOATBIN..... 436
FLOATDEC..... 437
FLOOR..... 437
FOLDEDFULLMATCH..... 438
FOLDEDSIMPLEMATCH..... 438
GAMMA..... 438
GETENV..... 439
GETJCLSYMBOL..... 439
GETSYSINT..... 439
GETSYSWORD..... 439
GTCA..... 441
GRAPHIC..... 441
HANDLE..... 442
HBOUND..... 442
HBOUNDACROSS..... 442
HEX..... 443
HEX8..... 444
HEXDECODE..... 445
HEXDECODE8..... 445
HEXIMAGE..... 446
HEXIMAGE8..... 446
HIGH..... 447
HUGE..... 447
IAND..... 447
ICLZ..... 447
IEOR..... 448
IFTHENELSE..... 448
IMAG..... 449
IMAG 疑似変数..... 449
INARRAY..... 449
INDEX..... 450
INDEXR..... 450
INDICATORS..... 451
INLIST..... 451
INOT..... 451
IOR..... 452
ISFINITE..... 452

ISIGNED... 452
ISINF... 453
ISLL... 453
ISJCLSYMBOL... 453
ISMAIN... 454
ISNAN... 454
ISNORMAL... 454
ISRL... 454
ISZERO... 455
IUNSIGNED... 455
JSONGETARRAYEND... 455
JSONGETARRAYSTART... 456
JSONGETCOLON... 456
JSONGETCOMMA... 456
JSONGETMEMBER... 457
JSONGETOBJECTEND... 458
JSONGETOBJECTSTART... 458
JSONGETVALUE... 459
JSONPUTARRAYEND... 461
JSONPUTARRAYSTART... 461
JSONPUTCOLON... 461
JSONPUTCOMMA... 461
JSONPUTMEMBER... 462
JSONPUTOBJECTEND... 463
JSONPUTOBJECTSTART... 463
JSONPUTVALUE... 464
JSONVALID... 465
JULIANTOSMF... 465
LBOUND... 465
LBOUNDACROSS... 465
LEFT... 466
LENGTH... 466
LINENO... 466
LOCATION... 467
LOCNEWSPACE... 467
LOCNEWVALUE... 468
LOCSTG... 469
LOCVAL... 469
LOG... 470
LOGGAMMA... 470
LOG2... 470
LOG10... 470
LOW... 471
LOWERASCII... 471
LOWERCASE... 471
LOWERLATIN1... 472
LOWER2... 472
MAINNAME... 472
最大... 472
MAXDATE... 473
MAXEXP... 473
MAXVAL... 474
MAXLENGTH... 475
MEMCONVERT... 475
MEMCOLLAPSE... 476
MEMCU12... 476
MEMCU14... 477
MEMCU21... 477

MEMCU24....478
MEMCU41....478
MEMCU42....478
MEMINDEX....479
MEMREPLACE....480
MEMSEARCH....481
MEMSEARCHR....482
MEMSQUEEZE....482
MEMVERIFY....483
MEMVERIFYR....484
MICROSECS....484
MICROSECSTODATE....485
MICROSECSTODAYS....485
MIN....486
MINDATE....486
MINEXP....486
MINVAL....487
MOD....487
MPSTR....488
MULTIPLY....489
NULL....489
NULLENTRY....490
OFFSET....490
OFFSETADD....490
OFFSETDIFF....490
OFFSETSUBTRACT....491
OFFSETVALUE....491
OMITTED....491
ONACTUAL....491
ONAREA....491
ONCHAR....492
ONCHAR 疑似変数....492
ONCODE....492
ONCONDCOND....492
ONCONDID....493
ONCOUNT....493
ONEXPECTED....494
ONFILE....494
ONGSOURCE....494
ONGSOURCE 疑似変数....494
ONHBOUND....495
ONKEY....495
ONLBOUND....495
ONLINE....496
ONLOC....496
ONOFFSET....496
ONOPERATOR....496
ONPACKAGE....497
ONPROCEDURE....497
ONSOURCE....497
ONSOURCE 疑似変数....497
ONSUBCODE....498
ONSUBCODE2....498
ONSUBSCRIPT....498
ONTEXT....498
ONUCHAR....499
ONUCHAR 疑似変数....499
ONUSOURCE....499

ONUSOURCE 疑似変数....499
ONWCHAR....500
ONWCHAR 疑似変数....500
ONWSOURCE....500
ONWSOURCE 疑似変数....500
ORDINALNAME....501
ORDINALPRED....501
ORDINALSUCC....501
PACKAGENAME....501
PAGENO....502
PICSPEC....502
PLACES....502
PLIASCII....503
PLIATTN....503
PLICANC....504
PLICKPT....504
PLIDELETE....504
PLIDUMP....504
PLIEBCDIC....504
PLIFILL....505
PLIFREE....505
PLIMOVE....505
PLIOVER....506
PLIPARSE....506
PLIREST....508
PLIRETC....508
PLIRETV....508
PLISAXA....509
PLISAXB....509
PLISAXC....509
PLISAXD....510
PLISRTA....510
PLISRTB....511
PLISRTC....511
PLISRTD....511
PLISTCK....511
PLISTCKE....512
PLISTCKELOCAL....512
PLISTCKEUTC....512
PLISTCKF....512
PLISTCKLOCAL....512
PLISTCKP....513
PLISTCKPLOCAL....513
PLISTCKPUCT....513
PLISTCKUTC....513
PLITRAN11....514
PLITRAN12....514
PLITRAN21....515
PLITRAN22....515
POINTER....516
POINTERADD....516
POINTERDIFF....517
POINTERSUBTRACT....517
POINTERVALUE....517
POLY....517
POPCNT....518
PRECVAL....518
PRECISION....518

PRED....519
PRESENT....519
PROCEDURENAME....519
PROD....519
PUTENV....520
QUICKSORT....520
QUICKSORTX....520
RADIX....521
RAISE2....521
RANDOM....522
RANK....522
REAL....522
REAL 疑似変数....522
REGEX....523
REM....525
REPATTERN....525
REPEAT....526
replace....526
REPLACEBY2....527
REVERSE....527
RIGHT....528
ROUND....528
ROUNDAWAYFROMZERO....530
ROUNDTOEVEN....530
SAMEKEY....531
SCRUBOUT....531
SCALE....532
SCALEVAL....532
SEARCH....532
SEARCHR....533
SECS....534
SECSTODATE....535
SECSTODAYS....535
SIGN....535
SIGNED....536
SIN....536
SIND....536
SINH....536
SIZE....537
SMFTOJULIAN....538
SOURCEFILE....538
SOURCELINE....538
SQRT....538
SQRTF....538
SQUEEZE....539
STACKADDR....539
STCKETODATE....539
STCKTODATE....540
STORAGE....540
STRING....540
STRING 疑似変数....541
SUBSTR....541
SUBSTR 疑似変数....542
SUBTO....542
SUBTO 疑似変数....543
SUBTRACT....543
SUCC....543
SUM....544

SYSNULL..... 544
SYSTEM..... 544
TALLY..... 545
TAN..... 545
TAND..... 545
TANH..... 545
THREADID..... 545
TIME..... 546
TIMESTAMP..... 546
TINY..... 546
TRANSLATE..... 546
TRIM..... 547
TRUNC..... 548
TYPE..... 548
TYPE 疑似変数..... 548
UHIGH..... 549
ULENGTH..... 549
ULENGTH8..... 549
ULENGTH16..... 549
ULOW..... 550
UNALLOCATED..... 550
UNHEX..... 550
UNSIGNED..... 551
UNSPEC..... 551
UNSPEC 疑似変数..... 553
UPOS..... 554
UPPERASCII..... 554
UPPERCASE..... 554
UPPERLATIN1..... 555
USUBSTR..... 555
USUPPLEMENTARY..... 556
UTCDATETIME..... 556
UTCMICROSECS..... 556
UTCSECS..... 557
UTF8..... 557
UTF8STG..... 558
UTF8TOCHAR..... 558
UTF8TOWCHAR..... 558
UUID..... 558
UUID4..... 559
UVALID..... 559
UWIDTH..... 560
VALID..... 560
VALIDDATE..... 561
VALIDVALUE..... 562
VARGLIST..... 562
VARGSIZE..... 562
VERIFY..... 563
VERIFYR..... 564
WCHARVAL..... 564
WEEKDAY..... 564
WHIGH..... 565
WHEREDIFF..... 565
WIDECHAR..... 565
WLOW..... 566
WSCOLLAPSE..... 566
WSCOLLAPSE16..... 567
WSREPLACE..... 567

WSREPLACE16 568
XMLCHAR 568
XMLSCRUB 569
XMLSCRUB16 570
XMLUCHAR 571
Y4DATE 571
Y4JULIAN 572
Y4YEAR 572

**第 19 章タイプ付き関数 575**
タイプ付き関数の呼び出し 575
タイプ付き関数の引数の指定 575
タイプ付き関数の要旨 575
BIND 576
CAST 576
FIRST 576
LAST 577
NEW 577
RESPEC 577
SIZE 578
VALUE 578

**第 20 章プリプロセッサーの機能 579**
プリプロセッサー・オプション 580
プリプロセッサー走査 581
プリプロセッサー・ステートメントの実行 581
リスト制御ステートメントの実行 582
入力テキストの実行 582
プリプロセッサー変数およびデータ要素 583
プリプロセッサー参照とプリプロセッサー式 583
プリプロセッサー名の有効範囲 584
プリプロセッサー・プロシージャー 584
プリプロセッサー・プロシージャーの引数およびパラメーター 585
%PROCEDURE ステートメント 586
プリプロセッサー RETURN ステートメント 587
プリプロセッサー ANSWER ステートメント 587
プリプロセッサー CALL ステートメント 589
プリプロセッサー組み込み関数 589
COLLATE 590
COMMENT 590
COMPILEDATE 590
COMPILETIME 591
COPY 592
COUNTER 592
DIMENSION 592
HBOUND 593
INDEX 593
LBOUND 593
LENGTH 594
LOWERCASE 594
MACCOL 594
MACLMAR 595
MACNAME 595
MACRMAR 595
MAX 595
MIN 595
PARMSET 595

QUOTE... 596
REPEAT... 596
SUBSTR... 596
SYSDIMSIZE... 597
SYSOFFSETSIZE... 597
シスパーム... 597
SYSPOINTERSIZE... 598
SYSTEM... 598
SYSVERSION... 598
TRANSLATE... 598
TRIM... 599
UPPERCASE... 599
VERIFY... 599
プリプロセッサー・ステートメント... 600
%ACTIVATE ステートメント... 600
%assignment ステートメント... 601
%DEACTIVATE ステートメント... 601
%DECLARE ステートメント... 601
%DO ステートメント... 603
%END ステートメント... 604
%GO TO ステートメント... 604
%IF ステートメント... 604
%INCLUDE ステートメント... 605
%INSCAN ステートメント... 606
%ITERATE ステートメント... 606
%LEAVE ステートメント... 607
%NOTE ステートメント... 607
%null ステートメント... 608
%REPLACE ステートメント... 608
%SELECT ステートメント... 608
%XINCLUDE ステートメント... 609
%XINSCAN ステートメント... 609
プリプロセッサーの例... 609

**付録 A 制限値... 615**

**特記事項... 625**
商標... 625
**参照文献... 627**
PL/I 資料... 627
関連資料... 627
**用語集... 629**

**索引... 651**

# 表

1. 英字と同等の値.....1
2. 10 進数と同等の値..... 3
3. 特殊文字と同等の値..... 3
4. 複合記号の説明.....4
5. 区切り文字..... 6
6. 演算子..... 7
7. 定数タイプによる属性の分類..... 19
8. 変数タイプによる属性の分類..... 21
9. コード化算術データ属性の省略形..... 23
10. FIXED BINARY SIGNED データ・ストレージ所要量..... 25
11. FIXED BINARY UNSIGNED データ・ストレージ所要量..... 25
12. 2 進浮動小数点定数の例..... 28
13. 10 進浮動小数点定数の例..... 28
14. ストリング・データ属性の省略形.....30
15. 文字定数の例.....33
16. X 定数の例..... 34
17. ビット定数の例..... 35
18. B4 定数の例.....35
19. B3 定数の例.....36
20. GX (16 進) グラフィック定数の例..... 36
21. 混合文字定数の例.....37
22. UX (16 進) UCHAR 定数の例.....38
23. WX (16 進) ワイド文字定数の例..... 39

24. 算術演算子....56
25. 1 つまたは複数の FLOAT オペランドの算術演算の結果....58
26. RULES(ANS) における、2 つのスケールのない FIXED オペランド間の算術演算の結果....59
27. RULES(ANS) における、2 つのスケールされた FIXED オペランド間の算術演算の結果....60
28. RULES(IBM) における、2 つの FIXED オペランド間の算術演算の結果....61
29. 固定小数点除算と定数式の比較....62
30. 累乗演算の特殊事例....62
31. ビット演算子用の論理演算子....63
32. ビット演算子....64
33. ビット演算の例....64
34. 演算の優先順位と変換の手引き....68
35. CEIL (n*3.32) の値と CEIL (n/3.32) の値の表....76
36. 序数処理組み込み関数....144
37. タイプ付き関数....146
38. ハーフワード、ワード、およびダブルワードの規定境界の位置合わせ....158
39. 位置合わせ要件....159
40. デフォルトの算術精度....166
41. DEFAULT 内の attribute-expression に指定する属性....169
42. 複合代入演算子....198
43. 代替ファイル属性....275
44. データ伝送のタイプによる属性....275
45. PL/I ファイル宣言の属性....276
46. 暗黙オープンによって暗黙に定義される属性....282
47. マージ後の属性と暗黙に定義される属性....283
48. PRINT ファイルのオプションとフォーマット項目....313

49. 文字ピクチャー指定の例..... 326
50. 数字と小数点文字の例..... 328
51. ゼロ抑制文字の例..... 329
52. 挿入文字の例..... 330
53. 符号と通貨記号の例..... 334
54. T、I、および R ピクチャー文字の解釈..... 335
55. 貸方記号、借方記号、オーバーパンチ、およびゼロ置き換え文字の例..... 336
56. 指数文字の例..... 337
57. 位取り係数文字の例..... 337
58. 条件のクラスと状況..... 340
59. 構造体または共用体の引数を受け入れることができる組み込み関数および疑似変数..... 374
60. 算術組み込み関数..... 376
61. 配列処理組み込み関数..... 376
62. バッファー管理組み込み関数..... 377
63. 条件処理組み込み関数..... 379
64. 日付/時刻組み込み関数..... 381
65. 日付/時刻のパターン..... 382
66. 時刻のみのパターン..... 383
67. 浮動小数点の照会組み込み関数..... 383
68. 浮動小数点演算組み込み関数..... 384
69. 入出力組み込み関数..... 384
70. 整数演算組み込み関数..... 385
71. JSON 組み込み関数..... 385
72. 数学組み込み関数..... 386
73. その他の組み込み関数..... 387

74. 序数処理組み込み関数 ..... 389

75. 精度処理組み込み関数 ..... 389

76. 組み込み疑似変数 ..... 390

77. ストレージ制御組み込み関数 ..... 391

78. ストリング処理組み込み関数 ..... 393

79. 組み込みサブルーチン ..... 395

80. ソース・バッファーを EBCDIC として Base64 にエンコードする例 ..... 406

81. ソース・バッファーを UTF-8 として Base64 にエンコードする例 ..... 407

82. ソース・バッファーを UTF-16 として Base64 にエンコードする例 ..... 408

83. UNSPEC によって戻されるビット・ストリングの長さ ..... 551

84. タイプ付き関数 ..... 575

85. 言語エレメントの制限 ..... 615

86. マクロ機能の制限 ..... 617

87. LOWERCASE 組み込み関数および UPPERCASE 組み込み関数に対してサポートされているコード・ページ値 ..... 618

# 図

1. PL/I アプリケーション構造.....87

2. データ宣言の有効範囲..... 151

3. 入り口およびラベル宣言の有効範囲..... 152

4. 構造体のマッピング例..... 188

5. 小構造体 G のマッピング..... 188

6. 小構造体 E のマッピング..... 189

7. 小構造体 N のマッピング..... 189

8. 小構造体 S のマッピング..... 189

9. 小構造体 C のマッピング..... 190

10. 小構造体 M のマッピング..... 190

11. 大構造体 A のマッピング..... 191

12. 構造 A の最終的なマッピングにおけるオフセット.....192

13. 一方向のチェーンの例..... 257

# Enterprise PL/I for z/OS 言語解説書

バージョン 6 リリース 1

SC31-5716-00

お願い

本書および本書で紹介する製品をご使用になる前に、625 ページの『特記事項』に記載されている情報をお読みください。

## 第 1 版 (2022 年 5 月)

本書は、Enterprise PL/I for z/OS バージョン 6 リリース 1 (5655-PL6)、および IBM Developer for z/OS PL/I for Windows (旧 Rational Developer for System z PL/I for Windows) バージョン 9.1、ならびに新しい版またはテクニカル・ニュースレターで明記されていない限り、以降のすべてのリリースに適用されます。製品のレベルに合った正しい版をご使用ください。

資料を注文する場合は、IBM® 担当者または最寄りの IBM 営業所にご連絡ください。 <!--Publications are not stocked at the address below.-->

A form for readers' comments is provided at the back of this publication. If the form has been removed, address your comments to:

IBM コーポレーション、部門 H150/090
555 Bailey Ave.
San Jose, CA, 95141-1099
United States of America

情報を IBM に送信する場合、提供者がいかなる義務も負うことのない適切な方法で、その情報の使用または配布する非独占的権利を IBM に提供することになります。

IBM Enterprise PL/I for z/OS は継続的デリバリー (CD) モデルをサポートしており、資料は CD モデルで配布されるフィーチャーを記述するために更新されるので、3 カ月ごとに更新の有無を確認することをお勧めします。

**Copyright International Business Machines Corporation 1999, 2022. All Rights Reserved.**

US Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

# 本書について

本書は、以下の IBM 製品で IBM PL/I コンパイラーを 使用するプログラマーを対象とした解説書です。

- Enterprise PL/I for z/OS, バージョン 6 リリース 1
- IBM Developer for z/OS PL/I for Windows (旧 Rational Developer for System z PL/I for Windows) バージョン 9.1

本書は初歩的な手引書ではなく、すでに PL/I 言語に関する知識を持ち、IBM PL/I コンパイラー用のプログラムの作成に参照情報を 必要とする読者を対象としています。 本書には、ガイダンス情報や汎用プログラミング・インターフェースも含まれています。

本書は解説書であり、最初から順に読まれることを意図していません。 したがって、事前に定義されていない用語が使用されている場合があります。 用語は定義される箇所で強調表示され、その定義は用語集に記載されています。

WRKSTN➡

ワークステーションのオープン・アイコンとクローズ・アイコンによって区別されるテキストは、PL/I ワークステーション製品 (AIX および Windows) でのみサポートされる機能を指定します。
⬅

# 本書で用いられる表記規則

本書での表記法を、以下に説明します。 例およびユーザーが提供する情報は、大文字と小文字の両方を使用して示されています。 本書で使用されている構文図には、以下の規則が適用されます。

**矢印記号**

構文図は、左から右、上から下へと線をたどって読んでください。

**►► & boxh; & boxh; & boxh;**
ステートメントはここから始まります。

**& boxh; & boxh; & boxh; ►**
ステートメントの構文は次の行へ続きます。

**► & boxh; & boxh; & boxh;**
ステートメントは前の行から続いています。

**& boxh; & boxh; & boxh; 講習**
ステートメントはここで終わります。

完全なステートメント以外の構文単位の図は、► & boxh; & boxh; & boxh; 記号で始まり、& boxh; & boxh; & boxh; ► 記号で終わります。

**規則**

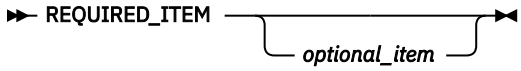
- キーワード、許容される同義語、および予約パラメーターは、MVS プラットフォームでは大文字で示され、UNIX プラットフォームでは小文字で示されます。 これらの項目は示されたとおりに入力する必要があります。
- 変数は、小文字のイタリック体で示します (例えば、*column-name*)。 これらはユーザー定義のパラメーターまたはサブオプションを表します。
- コマンドの入力において、パラメーターおよびキーワードを区切る 句読記号がない場合は、少なくとも 1 つのブランクで区切る必要があります。
- 句読記号 (スラッシュ、コンマ、ピリオド、括弧、引用符、等号) と 数字は、示されたとおりに入力する必要があります。
- 脚注は、(1) のように番号を括弧に入れて示します。
- ␣ 記号は 1 つのブランク位置を示します。

**必須項目**

必須項目は横線 (メインパス) に示します。

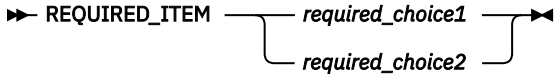
© Copyright IBM Corp. 1999, 2022

**オプション項目**
オプション項目は、メインパスの下に示します。

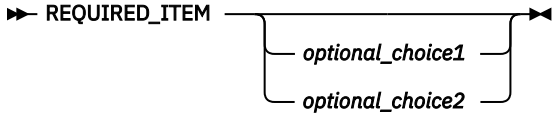



メインパスより上にオプション項目を示すこともあります。これは 読みやすくするためで、ステートメントの実行には影響を及ぼしません。

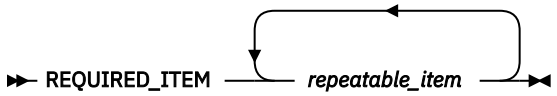
**複数の必須項目またはオプション項目**
複数の項目から選択する場合には、それらの項目が縦に重なって、 スタックを形成しています。 複数の項目からいずれか１つを選択しなければならない 場合は、 スタック上の項目のうち、１つがメインパス上に置かれます。



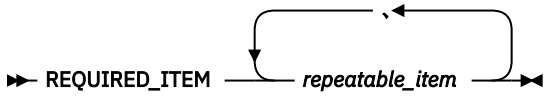

項目がオプショナルである場合、 メインパスの下にある支線上に縦に並んだ項目として示されます。




**反復可能項目**
メインパスの上方を通って左側へ戻る矢印は、項目が反復可能であることを 示します。

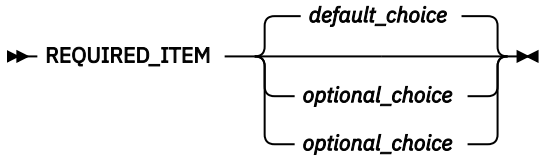



反復矢印の途中にコンマがある場合は、反復される項目をコンマで 区切らなければなりません。

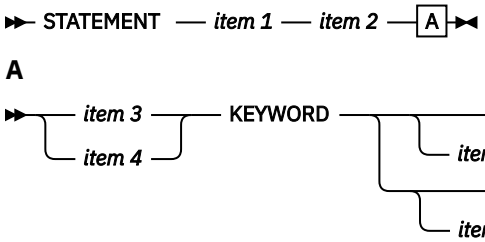



スタックの上の繰り返しを示す矢印は、 スタックから複数の選択項目を指定できることを示しています。

**デフォルト・キーワード**
IBM 提供のデフォルト・キーワードはメインパスより上に示され、 それ以外の選択項目はメインパスより下に示されます。 構文図の下にあるパラメーター・リストでは、デフォルト選択項目に 下線を付けてあります。

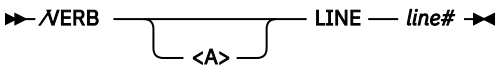



**フラグメント**
構文図は、フラグメント (部分) に分割する必要がある場合があります。 フラグメントは、文字またはフラグメント名で表され、次のように設定されます: | A |。 フラグメントは、メイン・ダイアグラムの終わりの後に続きます。 次の例は、フラグメントの使い方を示したものです。

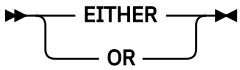



**置換ブロック**

場合によっては、いくつかのパラメーターのセットが、<A>などの置換ブロックによって表されることがあります。 例えば、虚数 /VERB コマンドでは、 ⁄'VERB LINE 1, ⁄'VERB EITHER LINE 1, または ⁄'VERB OR LINE 1 と入力できます。

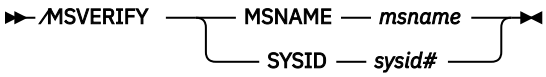



ここで、<A> は次のようになります。




**パラメーターの終わり**

数値を持つパラメーターは記号 '#' で終わり、名前であるパラメーターは 'name' で終わり、汎用とすることができるパラメーターは '*' で終わります。




この例の MSNAME キーワードは名前の値をサポートし、SYSID キーワードは数値をサポートします。

## セマンティクス

PL/I 言語の説明には、以下の規則が使用されます。

- 説明文は簡略化しています。 例えば、通常は、より正確な"「x は変数の名前でなければならない」"ではなく、"「x は変数でなければならない」"と記述します。 同様に、"「x の値が送信される」"の代わりに"「x が送信される」"と記述することもできます。 構文が"参照"を示す場合は、後で"「参照される変数」"の代わりに"「変数」"と記述できます。
- 異なる 2 つのソース構成が同等であるというときは、同じ結果をもたらすことを意味しますが、必ずしもその仕組みが同じであるとは限りません。
- 構文仕様の後のテキストで特に明記していない限り、修飾されていない用語"式" または"参照" はスカラー式を指します。 スカラー式以外の式の場合には、その式のタイプを明記してあります。 例えば、用語"「配列式」"は、スカラー式も構造式も有効でないことを示します。
- 結果または動作が「未定義」の場合、これは行っては"「ならない」"ものです。 未定義機能を使用すると、異なる PL/I プロダクトのインプリメンテーションまたはリリースで 異なる結果を生成することになります。 そのアプリケーション・プログラムはエラーであると見なされます。
- デフォルト という用語は、ユーザーが明示的に選択しなかったときにシステムで想定される代替値、属性、またはオプションを説明する場合に使われています。
- 暗黙 という用語は、明示的な指定のないときにプログラムによってとられる アクションを説明する場合に使用されます。
- (単語内以外で使用される) 小文字の b は、ブランク文字を示しています。

# 工業規格

PL/I コンパイラーは、 下記の工業規格 (1987 年 12 月の時点での IBM の解釈にもとづきます) の 仕様に従って設計されています。

- 情報交換用米国標準コード (ASCII)、X3.4 - 1977
- 情報交換用ポケット選択文字米国標準規格、レベル 1、X3.77 - 1980 (1979 年 3 月 1 日 ISO への提案事項)
- 情報交換用縦方向キャリッジ位置指定文字米国標準規格、レベル 1 の草案。 dpANS X3.78 (1979 年 3 月 1 日 ISO への提案事項)
- 米国標準規格 PL/I 汎用サブセットの選択機能 (ANSI X3.74-1987)

# 変更の要約

このセクションでは、Enterprise PL/I for z/OS バージョン 6 リリース 1 の本書に対して行われた主な変更をリストします。 読者の便宜のために、ここに記載されている変更には相互参照が関連付けられています。最新の技術変更は、HTML バージョンでは強調表示され、PDF バージョンでは左マージンに縦線 (|) が付けられます。

## 本リリースにおける機能強化

本リリースは、本書および他の IBM PL/I ブックに記載されている次の機能強化を提供します。

### ユーザビリティーにおける機能拡張

- ENTRY 宣言および PROCEDURE ステートメントは、CMPAT(V1|V2|V3)、AMODE31、および AMODE64 を選択項目として追加します。 123 ページの『OPTIONS オプションとその属性』を参照してください。
- ALIGNED 属性が更新され、パラメーター n が追加されました。 ALIGNED(n) は、データ・タイプのデフォルトの位置合わせをオーバーライドする位置合わせを指定します。 158 ページの『ALIGNED 属性と UNALIGNED 属性』を参照してください。
- 名前付き定数の使用は、JSONNAME 属性でサポートされています。 183 ページの『JSONNAME 属性』を参照してください。
- INITIAL 属性は、選択項目として INITACROSS を追加します。これにより、すべての構造体メンバーがスカラーである構造体の 1 次元配列へのエレメントの追加または削除を容易にすることで、それらの配列の初期化に役立ちます。 265 ページの『INITIAL 属性』を参照してください。

### 新規および変更された組み込み関数、疑似変数、およびサブルーチン

- WHEREDIFF 組み込み関数は、2 つのバッファーを比較し、異なる最初のバイトの索引を戻します。 565 ページの『WHEREDIFF』を参照してください。
- YYYYMMDDHHMISS999999 日付/時刻パターンがサポートされます。 380 ページの『日付/時刻組み込み関数』を参照してください。
- ONHBOUND 組み込み関数は、配列索引によって SUBSCRIPTRANGE が発生したときに、上限の値を戻します。 495 ページの『ONHBOUND』を参照してください。
- ONLBOUND 組み込み関数は、配列索引によって SUBSCRIPTRANGE が発生した場合に、下限の値を戻します。 495 ページの『ONLBOUND』を参照してください。
- ONSUBSCRIPT 組み込み関数は、SUBSCRIPTRANGE の発生の原因となった無効な配列索引の値を戻します。 498 ページの『ONSUBSCRIPT』を参照してください。
- GTCA 組み込み関数は、LE 制御ブロックへのポインターを返します。 441 ページの『GTCA』を参照してください。
- REG12 組み込み関数は除去されました。
- PLISTCKP 組み込み関数は、対応する Perform Timing Facility Function ハードウェア命令を生成します (z/OS のみ)。 513 ページの『PLISTCKP』を参照してください。

- PLISTCKPLOCAL 組み込み関数は、対応する Perform Timing Facility Function ハードウェア命令を生成し、現地時間を指定するように PTFF 値を調整します (z/OS のみ)。 513 ページの『PLISTCKPLOCAL』を参照してください。
- PLISTCKPUTC 組み込み関数は、対応する Perform Timing Facility Function ハードウェア命令を生成し、UTC 時間を指定するように PTFF 値を調整します (z/OS のみ)。 513 ページの『PLISTCKPUCT』を参照してください。
- PLIPARSE 組み込み関数は、文字ストリングをサブストリングに構文解析します。 506 ページの『PLIPARSE』を参照してください。
- GETSYSWORD 組み込み関数に、新規引数 SYSNAME、SYSNODE、および SYSPLEX が追加されました。 439 ページの『GETSYSWORD』を参照してください。
- 型付き構造体の使用は、これらの JSON 関数 JSONGETMEMBER、JSONGETVALUE、JSONPUTMEMBER、JSONPUTVALUE に対して許可されます。 457 ページの『JSONGETMEMBER』、459 ページの『JSONGETVALUE』、464 ページの『JSONPUTVALUE』、および 462 ページの『JSONPUTMEMBER』 を参照してください。
- パターンが MMM および Mmm を使用する場合の REPATTERN のインライン化。 525 ページの『REPATTERN』を参照してください。
- QUICKSORT 組み込み関数に、オプションの引数 m および n が追加されました。 520 ページの『QUICKSORT』を参照してください。
- QUICKSORTX 組み込み関数に、オプションの引数 m および n が追加されました。 520 ページの『QUICKSORTX』を参照してください。
- UUID および UUID4 組み込み関数は、小文字の結果を戻すことがあります。 558 ページの『UUID』および 559 ページの『UUID4』を参照してください。
- ORDINAL データのテストは、VALID 組み込み関数を介してサポートされます。 560 ページの『VALID』を参照してください。

### 新規または変更されたステートメント、ディレクティブ、および条件:

- XDECLARE ステートメントは、名前の属性の一部またはすべてを指定します。 233 ページの『XDECLARE ステートメント』を参照してください。
- %PROCINC ディレクティブは、コンパイラー・オプションをオーバーライドするために使用されます。 228 ページの『%PROCINC ディレクティブ』を参照してください。
- *PROCINC ディレクティブは、%PROCINC ディレクティブと同義です。 228 ページの『*PROCINC ディレクティブ』を参照してください。
- SIZE 条件は、より多くの動作をする可能性があります。 359 ページの『SIZE 条件』を参照してください。
- 明示的な CONFORMANCE 条件がサポートされます。 339 ページの『条件接頭語』を参照してください。

## 5.3 からの機能拡張

本リリースは、本書および他の IBM PL/I ブックに記載されている次の機能強化を提供します。

### ユーザビリティーにおける機能拡張

- JSONTRIMR 属性は、指定されたフィールドの末尾ブランクを JSON 関数が切り取ることを示します。 183 ページの『JSONTRIMR 属性』を参照してください。
- UCHAR 属性は、ストリングが UTF-8 データを保持すること、および INDEX、SUBSTR、VERIFY などの関数がこれを UTF-8 依存方式で処理することを指定します。 37 ページの『UCHAR データ』 及び 38 ページの『UX (16 進) UCHAR 定数』 を参照してください。
- 文字 // は、行の残り部分がコメントであることを指定します。 6 ページの『区切り文字と演算子』を参照してください。
- 47 ページの『VALUELIST 属性』は、変数、引数、または戻り値が持つことができる値のセットを制限します。

- 47 ページの『VALUELISTFROM 属性』は、その VALUELIST 属性が現在の宣言にも適用される添え字なし参照を指定します。
- 182 ページの『XMLNAME 属性』は、XMLCHAR 組み込み関数で変数に使用される名前を指定する機能を提供します。
- LIKE 属性は、ENTRY 記述およびパラメーター宣言で使用できます。 178 ページの『LIKE 属性』を参照してください。
- SUPPRESS(NOGLOBAL) 属性を使用すると、コンパイラーは、ネストされたプロシージャーでの変数の使用に対してフラグを立てません。 157 ページの『SUPPRESS 属性』を参照してください。
- コンパイラーでは、異なる DISPLAY ステートメントでの異なる ROUTCDE 値および DESC 値の指定がサポートされます。 207 ページの『DISPLAY ステートメント』を参照してください。
- コンパイラーでは、*YYYY/MM/DD*、*YY/MM/DD*、*YYYY-MM-DDTHH:MI:SS.999999*、*DD/MM/YYYY*、および *DD/MM/YY* の日付/時刻パターンがサポートされます。 380 ページの『日付/時刻組み込み関数』を参照してください。
- コンパイラーによって配列のエクステントがその INITIAL 属性から推論されます。 171 ページの『次元属性』を参照してください。
- コンパイラーでは、STATIC NONASGN 配列エレメントの定数としての使用がサポートされます。 71 ページの『制限付き式』を参照してください。
- コンパイラーでは、PUT DATA ステートメントでの名前付き定数の使用がサポートされます。 297 ページの『データ指定オプション』を参照してください。
- コンパイラーでは、ドットで区切った一連の ID からなるタイプ名参照 (`paint.color` など) が受け入れられます。
- コンパイラーは、HANDLE、OFFSET、ENTRY、および AREA 変数への (`''` または `''b` と指定された) ヌル・ストリングの割り当ての使用をサポートします。 199 ページの『非計算ターゲット』を参照してください。

## 新規および変更された組み込み関数、疑似変数、およびサブルーチン

- 組み込み関数 571 ページの『XMLUCHAR』は、構造体に対応する XML を UTF-8 としてバッファーに書き込みます。
- 組み込み関数 542 ページの『SUBTO』は、ストリングの終了位置で指定されたサブストリングを戻します。
- 543 ページの『SUBTO 疑似変数』は、ストリングの終了位置で指定されたサブストリングを割り当てます。
- 組み込み関数 401 ページの『ALLOCNEXT』では、AREA を使用した高速割り振りが提供されます。
- 組み込み関数 413 ページの『BYTELENGTH』では、UCHAR タイプ・ストリングで使用されるバイト数である FIXED BINARY(31) 値が返されます。
- 組み込み関数 431 ページの『FILEDDWORD』では、DSORG オプションを使用して、ファイル参照のデータ・セット編成を取得できます。
- 組み込み関数 438 ページの『FOLDEDFULLMATCH』では、Unicode の完全な 大/小文字変換規則に従って小文字に変換されたときに 2 つのストリングが同じかどうかを示す FIXED BINARY(31) 値が返されます。
- 組み込み関数 438 ページの『FOLDEDSIMPLEMATCH』では、Unicode の単純な 大/小文字変換規則に従って小文字に変換されたときに 2 つのストリングが同じかどうかを示す FIXED BINARY(31) 値が返されます。
- 組み込み関数 439 ページの『GETSYSINT』では、要求されたシステム情報の値である *size_t* 値が返されます。
- 組み込み関数 439 ページの『GETSYSWORD』では、ASID、ACTINFO、MSGCLASS、JESNODE、JOBCLASS、JOBNUMBER、および SMFID がキーワードとして受け入れられます。
- 組み込み関数 448 ページの『IFTHENELSE』では、C 条件式 (`x?y:z`) と等価な機能が提供されます。

- 組み込み関数 449 ページの『INARRAY』では、式が配列エレメントのいずれかと等しいかどうかを示す BIT(1) 値が返されます。
- 組み込み関数 453 ページの『ISJCLSYMBOL』では、入力引数名が、有効なエクスポート済み JCL シンボルであるかどうかを示す BIT(1) 値が返されます。
- 組み込み関数 462 ページの『JSONPUTMEMBER』および 464 ページの『JSONPUTVALUE』では、名前を小文字、大文字、またはそのままのいずれで書き込むかを指定するオプション・パラメーターが受け入れられます。
- 組み込み関数 471 ページの『LOWERASCII』では、ASCII 文字すべてを対応する小文字に変換した UCHAR ストリングが返されます。
- 組み込み関数 472 ページの『LOWERLATIN1』では、ASCII 文字および Latin-1 補足文字すべてを対応する小文字に変換した UCHAR ストリングが返されます。
- 組み込み関数 473 ページの『MAXDATE』では、指定された日付/時刻パターンに対応する最遅日付/時刻値を含む文字ストリングが返されます。
- 組み込み関数 475 ページの『MEMCONVERT』に、変換で使用される手法を指定するパラメーター *t* が追加されています。
- 組み込み関数 480 ページの『MEMREPLACE』では、指定された 3 番目のバッファーの 1 つ以上のオカレンスを 4 番目のバッファーで置き換えて、ソース・バッファーの内容がターゲット・バッファーに充てんされ、ターゲット・バッファーに書き込まれるバイト数を示す *size_t* 値が返されます。
- 組み込み関数 496 ページの『ONOPERATOR』では、ASSERTION 条件を発生させた ASSERT COMPARE ステートメントの演算子を値として持つ CHARACTER(2) ストリングが返されます。
- 組み込み関数 499 ページの『ONUCHAR』では、CONVERSION 条件の原因となった UTF-8 データを含む UCHAR(1) ストリングが返されます。
- 499 ページの『ONUCHAR 疑似変数』では、ONUCHAR 組み込み関数の現行値が設定されます。
- 組み込み関数 499 ページの『ONUSOURCE』では、その値が CONVERSION 条件の発生時に処理中であったフィールドの内容である UCHAR ストリングが返されます。
- 499 ページの『ONUSOURCE 疑似変数』では、ONUSOURCE 組み込み関数の現行値が設定されます。
- 組み込み関数 512 ページの『PLISTCKLOCAL』では、対応するストア・クロック・ハードウェア命令が生成され、STCK 値を調整して現地時間が提供されます。
- 組み込み関数 513 ページの『PLISTCKUTC』では、対応するストア・クロック・ハードウェア命令が生成され、STCK 値を調整して UTC 時間が提供されます。
- 組み込み関数 512 ページの『PLISTCKELOCAL』では、対応するストア・クロック・ハードウェア命令が生成され、STCKE 値を調整して現地時間が提供されます。
- 組み込み関数 512 ページの『PLISTCKEUTC』では、対応するストア・クロック・ハードウェア命令が生成され、STCKE 値を調整して UTC 時間が提供されます。
- 組み込み関数 518 ページの『PRECVAL』では、数値表現の精度が返されます。
- 組み込み関数 520 ページの『QUICKSORT』では、単純比較を使用した配列のクイック・ソートが実行されます。
- 組み込み関数 520 ページの『QUICKSORTX』では、指定された比較関数を使用した配列のクイック・ソートが実行されます。
- 組み込み関数 523 ページの『REGEX』では、あるストリングに対して指定された正規表現またはパターンとストリングと一致したことを示す FIXED BINARY(31) が返されます。
- 組み込み関数 526 ページの『replace』では、ターゲット・ストリング内のサブストリングの 1 つ以上のオカレンスが別のサブストリングで置換され、 置き換えられたサブストリングを含むターゲット・ストリングが返されます。
- 組み込み関数 539 ページの『STCKETODATE』では、STCKE 値が日付/時刻ストリングに変換されます。
- 組み込み関数 540 ページの『STCKTODATE』では、STCK 値が日付/時刻ストリングに変換されます。
- 組み込み関数 532 ページの『SCALEVAL』では、数値表現の位取り係数が返されます。
- 組み込み関数 531 ページの『SCRUBOUT』では、2 番目のストリングからすべての文字を除去したストリングが返されます。

- 組み込み関数 549 ページの『UHIGH』では、各 UTF-8 データ項目が最も大きい UCHAR 値 ('F48FBFBF'ux) を持つ、長さ *x* の UCHAR ストリングが返されます。
- 組み込み関数 550 ページの『ULOW』では、各 UTF-8 データ項目が最も小さい UCHAR 値 ('00'ux) を持つ、長さ *x* の UCHAR ストリングが返されます。
- 組み込み関数 550 ページの『UNHEX』では、HEX 組み込み関数の逆が実行されます。
- 組み込み関数 554 ページの『UPPERASCII』では、ASCII 文字すべてを対応する大文字に変換した UCHAR ストリングが返されます。
- 組み込み関数 555 ページの『UPPERLATIN1』では、ASCII 文字および Latin-1 補足文字すべてを対応する大文字に変換した UCHAR ストリングが返されます。
- 組み込み関数 559 ページの『UUID4』では、バージョン 4 の汎用固有 ID である CHARACTER(36) ストリングが返されます。
- JSON 組み込み関数が CMPAT(V1) でサポートされます。
- 組み込み関数 562 ページの『VALIDVALUE』では、式の値が変数の値セットのエレメントの 1 つと一致するかどうかを示す値が返されます。
- 233 ページの『XDEFINE ALIAS ステートメント』は DEFINE ALIAS ステートメントと同じです。ただし、指定された名前が前の (X)DEFINE ALIAS ステートメントに既に出現している場合、この XDEFINE ステートメントは無視されます。
- 234 ページの『XDEFINE ORDINAL ステートメント』は DEFINE ORDINAL ステートメントと同じです。ただし、指定された名前が前の (X)DEFINE ORDINAL ステートメントに既に出現している場合、この XDEFINE ステートメントは無視されます。
- 234 ページの『XDEFINE STRUCTURE ステートメント』は DEFINE STRUCTURE ステートメントと同じです。ただし、指定された名前が前の (X)DEFINE STRUCTURE ステートメントに既に出現している場合、この XDEFINE ステートメントは無視されます。

### 新規または変更されたステートメントおよび条件

- 229 ページの『QUALIFY ステートメント』と対応する END ステートメントは修飾ブロックを区切り、ORDINAL、その他のタイプ、および名前付き定数の名前空間を作成します。

## 5.2 からの機能拡張

本リリースは、本書および他の IBM PL/I ブックに記載されている次の機能強化を提供します。

### ユーザビリティーにおける機能拡張

- コンパイラーは、到達不能な ASSERT UNREACHABLE ステートメントに、他の到達不能なステートメントにフラグを立てるときとは異なるメッセージでフラグを立てます。
- コンパイラーは、他の構造体のメンバーである AGGREGATE リスト型付き構造体において展開します。
- 属性リストに、長さ 256 以下の定数 CHARACTER および BIT における VALUE 属性、また数値 PICTURE 定数における VALUE 属性の内容が表示されます。
- コンパイラーは、5 つの時間専用パターン *HHMISS*、*HH:MI:SS*、*HHMI*、*HH:MI*、および *HH* をサポートします。 380 ページの『日付/時刻組み込み関数』を参照してください。
- コンパイラーは、コードが含まれている PROC が本来は値を返す関数であるにもかかわらず、そのコードのプログラム・ロジックが END ステートメントに導く可能性がある場合に、コードにフラグを立てます。
- コンパイラーは、WHEN 文節内の式が、その文節を含む SELECT ステートメント内の WHEN 文節のいずれかに含まれている以前の式に一致する場合は対象の SELECT ステートメントにフラグを立てます。
- コンパイラーは、INIT を VALUE に置き換えできる、より多くのコードにフラグを立てます。
- コンパイラーは、AUTOMATIC 変数のアドレスを返す関数にフラグを立てます。その理由は、そのアドレスが、関数の呼び出し側によって使用された場合に信頼できなくなるためです。
- コンパイラーは、関数がインライン化されない場合、メッセージを出します。

- コンパイラーは、ランタイム・オプションを定義するための正しい属性を持っていない、PLIXOPT という名前の変数のすべての宣言にフラグを立てます。
- コンパイラーは、オペランドが同一の論理 AND 演算と論理 OR 演算にフラグを立てます。
- コンパイラーは、VALUE タイプ関数が、一部のみ初期化されている構造化タイプに適用されるコードにフラグを立てます。
- コンパイラーは、AUTOMATIC 変数でのストリングの長さが、その後に宣言される変数の SIZE に依存しているコードにフラグを立てます。

## 新規および変更された組み込み関数、疑似変数、およびサブルーチン

- 新しい 405 ページの『BASE64ENCODE』 組み込み関数は、ソース・バッファーを、DEFAULT コンパイラー・オプションの ASCII/EBCDIC サブオプションによって指定された文字セット内に Base 64 の値を保持するバッファーにエンコードし、新しい 404 ページの『BASE64DECODE』 組み込み関数は、DEFAULT コンパイラー・オプションの ASCII/EBCDIC サブオプションによって指定された文字セット内にエンコードされている Base 64 からソース・バッファーをデコードします。
- 新しい 410 ページの『BINSEARCH』 組み込み関数および 411 ページの『BINSEARCHX』 組み込み関数は、指定された値のバイナリー・サーチを配列内で実行するためのコードを生成します。バイナリー・サーチは、単純な比較またはユーザー指定関数の呼び出しによる必要な比較を実行することによって行います。
- 新しい 418 ページの『CODEPAGE』 組み込み関数は、コンパイラーの CODEPAGE オプションの値を条件としたコードの作成、およびそのコンパイラー・オプションの値の照会に役立ちます。
- 新しい 418 ページの『COLLAPSE』 組み込み関数および 539 ページの『SQUEEZE』 組み込み関数は、オプションで指定された位置から開始して、文字のすべての複数オカレンスを 1 つに削減するストリングを返します。
- 新しい 432 ページの『FILENEW』 組み込み関数は、自動ストレージ内の新しいファイル定数を指す FILE 変数を返します。
- 新しい 439 ページの『GETJCLSYMBOL』 組み込み関数および 439 ページの『GETSYSWORD』 組み込み関数は、要求されたエクスポート済み JCL 記号または要求されたシステム情報を示す、CHARACTER ストリング値を返します。
- 新しい 444 ページの『HEX8』 組み込み関数および 446 ページの『HEXIMAGE8』 組み込み関数は、式の 16 進数値を UTF-8 ストリングとして返します。
- 471 ページの『LOWERCASE』 組み込み関数および 554 ページの『UPPERCASE』 組み込み関数は、2 つ目のオプション引数を受け入れるようになりました。それにより、すべての文字が同等の小文字または同等の大文字に変換されるコード・ページを指定できるようになりました。 すべての A-Z/a-z の値および Ä ウムラウト/ä ウムラウトの値が同等の小文字または大文字の値に変換されます。
- 新しい 476 ページの『MEMCOLLAPSE』 組み込み関数は、ターゲット・バッファーを、ソース・バッファーの内容で埋めます。その際、指定された文字のすべての複数オカレンスは 1 つのオカレンスに置き換えられ、その文字の先行インスタンスおよび末尾インスタンスのトリムも行われます。 ターゲット・バッファーに書き込まれるバイト数を示す *size_t* 値が返されます。
- 新しい 482 ページの『MEMSQUEEZE』 組み込み関数は、ターゲット・バッファーをソース・バッファーの内容で埋めます。その際、指定された文字のすべての複数オカレンスは 1 つのオカレンスに置き換えられます。 ターゲット・バッファーに書き込まれるバイト数を示す *size_t* 値が返されます。
- 新しい組み込み関数 484 ページの『MICROSECS』、485 ページの『MICROSECSTODATE』、485 ページの『MICROSECSTODAYS』、426 ページの『DAYSTOMICROSECS』、および 556 ページの『UTCMICROSECS』 により、浮動小数点秒値に固有の丸めの問題なしに、8 バイト整数マイクロ秒値を使用して時間値を保持し、その値と日時ストリングとの間で変換を行うことができます。
- 新しい 487 ページの『MINVAL』 組み込み関数および 474 ページの『MAXVAL』 組み込み関数は、その数値オペランドが取ることのできる最小値または最大値を返します。
- 新しい 491 ページの『ONACTUAL』 組み込み関数および 494 ページの『ONEXPECTED』 組み込み関数は、ASSERTION 条件が発生した際に、ASSERT COMPARE ステートメント内の「実際の」式または「予期される」式の値である文字ストリングを返します。

- 新しい 498 ページの『ONTEXT』 組み込み関数は、ASSERTION 条件が発生した際に、ASSERT ステートメント内の TEXT ストリングの値である文字ストリングを返します。
- 新しい 497 ページの『ONPACKAGE』 組み込み関数は、ASSERTION 条件が発生した際に、PACKAGE の名前である文字ストリングを返します。
- 新しい 497 ページの『ONPROCEDURE』 組み込み関数は、ある条件が発生しているプロシージャーの名前を返します。 ONPROCEDURE および ONPROC は、ONLOC 組み込み関数の優先名としてサポートされるようになりました。
- 525 ページの『REPATTERN』 および 546 ページの『TIMESTAMP』 組み込み関数を使用して、*YYYY-MM-DD HH*:*MI*:*SS*.999999 の z システム・フォーマットで現在の日時を取得できるようになりました。
- 新しい 530 ページの『ROUNDTOEVEN』 組み込み関数は、偶数への丸めの規則に従って、*n* によって指定された桁で丸められた *x* の値を返します。
- 新しい 566 ページの『WSCOLLAPSE』 は、CHARACTER としてエンコードされているソース・バッファー内のすべての空白文字を省略します。 ターゲット・バッファーに書き込まれるバイト数を示す *size_t* 値が返されます。
- 新しい 567 ページの『WSREPLACE』 は、CHARACTER としてエンコードされたソース・バッファー内の \t、\f、\v、\n、および \r の各文字をブランクで置き換えます。 ターゲット・バッファーに書き込まれるバイト数を示す *size_t* 値が返されます。
- 新しい 569 ページの『XMLSCRUB』 組み込み関数は、CHARACTER ソース・バッファーをクリーニングします。 ターゲット・バッファーに書き込まれるバイト数を示す *size_t* 値が返されます。
- ROUNDDEC 組み込み関数は、530 ページの『ROUNDAWAYFROMZERO』 に名前変更されました。
- WHITESPACECOLLAPSE 組み込み関数は、567 ページの『WSCOLLAPSE16』 に名前変更されました。
- WHITESPACEREPLACE 組み込み関数は、568 ページの『WSREPLACE16』 に名前変更されました。
- XMLCLEAN 組み込み関数は、570 ページの『XMLSCRUB16』 に名前変更されました。

### 新規または変更されたステートメントおよび条件

- 193 ページの『ASSERT ステートメント』 は、新しい ASSERT COMPARE ステートメントをサポートします。このステートメントでは、より多くの情報を含んだアサーションのテスト方法が提供され、式の実際の値を予期された値と正しく比較できます。
- 新しい 349 ページの『ASSERTION 条件』 は、ASSERT ステートメントが失敗し、ASSERT(CONDITION) コンパイラー・オプションが有効な時に発生します。
- 93 ページの『PROCEDURE ステートメント』 および 94 ページの『ENTRY ステートメント』 は、オプション引数として EXTERNAL 属性をサポートします。

### パフォーマンス向上

- インライン化されていない場合、REPATTERN は、ソース日付をマイクロ秒値に変換し、次に、そのマイクロ秒値をターゲット日付形式に変換することにより、浮動小数点秒値が中間値として使用された際に精度が失われません。

### CICS の機能拡張

- CICS® プリプロセッサー出力に、プリプロセッサーを実行する時に有効なすべての CICS オプションのリストが含まれます。

## 5.1 からの機能拡張

本リリースは、本書および他の IBM PL/I ブックに記載されている次の機能強化を提供します。

### ユーザビリティーにおける機能拡張

- コンパイラーでは、LP(64) オプションで 64 ビット・アプリケーションがサポートされます。 これは、このオプションを使用すると、POINTER および HANLDE のサイズを 8 バイトにでき、構造体および配列

のサイズを 2G より大きくでき、また多くの組み込み関数 (PLIMOVE など) で 8 バイトの整数引数を使用し、8 バイトの整数結果を返すことができることを意味しています。

- INITIAL 属性がマクロ宣言ステートメントで使用できます。
- JSONNAME 属性は、jsonPut 関数または jsonGet 関数で変数に使用または予期される名前を指定する機能を提供します。 特に、この属性を使用して、PL/I の命名規則に準拠しない名前を指定することができます。
- JSONOMIT 属性は、JSON 関数によって生成される出力から特定データ項目のフィールドを除去する必要があることを指示します。 特に、変数に JSONOMIT 属性があると、jsonPut 関数は、変数がヌル・ストリング''に等しい場合に、生成されたテキストからこの変数を除去します。
- SUPPRESS 属性は、NOPADDING をサブ属性として受け入れます。これは変数に適用可能です。 レベル 1 構造体名に SUPPRESS(NOPADDING) を指定すると、その構造体に関しては RULES(NOPADDING) コンパイラー・オプションは無視されます。
- FIXED BIN(p,0) 以外の属性が REFER オブジェクトで使用された場合、コンパイラーは I レベル・メッセージではなく W レベル・メッセージを出します。
- UNION と、より一般的な UNION を含む構造体に対して、いくつかの割り当てがサポートされます。
- XMLCONTENT 属性は、XMLCHAR 組み込み関数で生成されたテキストに変数が含まれている場合に、その変数をタグなしテキストとして提示することを指定します。

## 新規および変更された組み込み関数、疑似変数、およびサブルーチン

- 新しい ALLOC31 組み込み関数は、2 GB 境界より下のヒープに、指定したサイズのストレージを割り振ります。
- 新しい組み込み関数 BETWEENEXCLUSIVE、BETWEENLEFTEXCLUSIVE、および BETWEENRIGHTEXCLUSIVE を使用すれば、変数が 2 つの指定値の間にあるかどうかをテストするコードを簡単に作成できます。 また、これらの組み込み関数を使用すると、そのようなテストのためのより良いコードをコンパイラーで生成することが容易になります。
- 新しい組み込み関数 CHECKSUM を使用すれば、データのバッファー用のチェックサム値を取得できます。
- 新しい組み込み関数 HEXDECODE および HEXDECODE8 を使用すれば、16 進表記データを含む文字データを、対応する値を持つデータに簡単に変換できます。
- 新しい組み込み関数 ICLZ は、整数値内の先行ゼロを素早くカウントする方法を提供します。
- 新しい組み込み関数 MAINNAME は、現行呼び出しチェーンでの MAIN 関数の名前を返します。
- 新しい組み込み関数 ONSUBCODE2 は、VSAM 障害に関するより詳細な情報を、ご使用のプログラムに提供します。
- 新しい組み込みサブルーチン PLIATTN は、コンパイラーがアテンション・ブレークポイントを挿入する場所に対する明示制御を提供します。
- TIMESTAMP 組み込み関数を使用して、*YYYY-MM-DD-HH.MI.SS*.999999 の z システム形式で現行日時を取得できます。
- 新しい組み込み関数 UTCDATETIME および UTCSECS を使用すれば、UTC 日時をストリングとして、またリリアン秒として取得できます。
- 新しい組み込み関数 UTF8STG を使用すれば、UTF-8 テキストの解析するコードを簡単に作成できます。
- 新しい組み込み関数 UUID を使用して、汎用固有 ID を取得することができます。

## パフォーマンス向上

- 桁数の多い固定小数点の除算のいくつかが、10 進浮動小数点 (DFP) 機能を使用して行われるようになりました。 これにより、一部の ZERODIVIDE 例外は INVALIDOP として報告される可能性があります。

## SQL 機能拡張

- INDFOR 属性を使用すると、標識変数の構造体を、別の PL/I 構造体に合わせて宣言することが容易になります。

## 4.5 からの機能拡張

本リリースは、本書および他の IBM PL/I ブックに記載されている次の機能強化を提供します。

- 新しい REINIT ステートメントを使用すれば、変数を INITIAL 値にリセットできます。
- 新しい BETWEEN 組み込み関数を使用すれば、変数が 2 つの指定値の間に入っているかどうかを検査するコードを簡単に記述できます。 この関数を使用すれば、コンパイラーによってそのような検査のための良質のコードを生成する作業も簡略化できます。
- 新しい INLIST 組み込み関数を使用すれば、変数の値が指定値のリストに入っているかどうかを検査するコードを簡単に記述できます。 この関数を使用すれば、コンパイラーによってそのような検査のための良質のコードを生成する作業も簡略化できます。
- 新しい NULLENTRY 組み込み関数を使用すれば、入り口変数にヌル値を割り当てたり、入り口変数の値がヌルかどうかを検査したりする処理を簡単に実行できます。 INITAUTO や INITBASED などのオプションを使用した場合は、入り口変数が初期化されます。
- 新しい組み込みサブルーチン PLISTCK、PLISTCKE、PLISTCKF を使用して、対応するストア・クロック・ハードウェア命令を生成できます。 日時組み込み関数よりも簡単にコード・セクションの時間を取得したり、高い精度でクロック値を取得したりすることができます。
- 一連の新しい組み込み関数では、JSON の構文解析、生成、妥当性検査が可能です。
- 新しい組み込み関数 SMFTOJULIAN と JULIANTOSMF を使用すれば、SMF 形式とユリウス形式の間で日付を変換できます。
- 新しい NULLINIT 属性を指定すると、INITIAL 属性を持たない変数の要素にデフォルトの初期値が設定されます。
- LOCATES 属性が拡張され、位置指定する型は、どの計算型でもよくなりました (例えば、ORDINAL 型も STRUCT 型も可能です)。
- BIT、CHARACTER、WIDECHAR の変数の宣言で新しい VARYING4 属性を使用して、その変数に 4 バイトの長さ接頭部が含まれることを指定できます。
- ユーザーは、新規 FORCE(NOLAXQUAL) 属性と、RULES(NOLAXQUAL) オプションの新規 FORCE サブオプションを使用すれば、 構造体ごとに NOLAXQUAL 規則を強制できます。
- ピクチャー・ストリングの中で、コンマや小数点やスラッシュと同じように、アポストロフィを挿入文字として使用できるようになりました。
- 新しいマクロ・プリプロセッサー組み込み関数 SYSDIMSIZE、SYSOFFSETSIZE、SYSPOINTERSIZE を使用すれば、さまざまなデータ項目に使用されるバイト数を変更するさまざまなコンパイラー・オプションの下で正しく動作するコードを作成できるようになります。
- 新しいマクロ XPROCEDURE ステートメントは、マクロ PROCEDURE ステートメントとほぼ同じですが、XPROCEDURE ステートメントの左端の名前が、定義済みのプリプロセッサー・プロシージャーの名前になっている場合に、プリプロセッサーは、XPROCEDURE ステートメントの後続の出現箇所にフラグを立てずに、その箇所を無視します。

## 4.4 からの機能拡張

本リリースは、本書および他の IBM PL/I ブックに記載されている次の機能強化を提供します。

- 新しい LOCATES 属性を使用すると、コンパイラーによってストリングの疎配列が作成されて渡されるときに、ストレージを大幅に削減できます。
- 新しい WIDEPIC 属性は、ピクチャー文字をデータ項目の各位置に関連付けることにより、WIDECHAR データ項目の特性を指定します。
- ALLOCATE 組み込み関数の新しいオプション引数として、AREA 参照が追加されました。
- 新しい組み込み関数 BASE64ENCODE8、BASE64ENCODE16、BASE64DECODE8、および BASE64DECODE16 が、Base64 のエンコードおよびデコードをサポートするようになりました。
- 新しい INDEXR 組み込み関数の機能は、INDEX 組み込み関数の機能と同じですが、検索が右から左へ行われます。
- 新しい LOCSTG 組み込み関数は、LOCATES を使用して間接的に保持できるすべての値を保持するために必要となるすべての割り振り済みストレージを保持するために必要なバイト数を返します。

- 新しい LOCVAL 組み込み関数は、LOCATES の description で指定されたタイプの区域内で、指定されたオフセットの値を返します。
- 新しい組み込み関数 XMLCLEAN、WHITESPACEREPLACE、および WHITESPACECOLLAPSE は、XML の処理を軽減します。
- LOCNEWSPACE 組み込みサブルーチンは、LOCATES 属性で記述された変数タイプのスペースを割り振ります。
- LOCNEWVALUE 組み込みサブルーチンは、LOCATES 属性で記述されたタイプの指定値のスペースを割り振り、その指定値をそのスペースの関連付けられたアドレスに割り当てます。
- 新しい CANCEL THREAD ステートメントは、指定されたスレッドを取り消します。
- DEFAULT ステートメントは、RANGE および属性キーワードが指定された論理式をサポートします。
- 新しいプリプロセッサー CALL ステートメントは、MACRO プロシージャーからの MACRO プロシージャーの呼び出しをサポートします。

## 4.3 からの機能拡張

本リリースは、本書および他の IBM PL/I ブックに記載されている次の機能強化を提供します。

- 新規の ASSERT ステートメントにより、プログラムの正確さの検証が向上します。 このステートメントは、条件が真であるか偽であるか、またはステートメントを実行してはならないかどうかを表明します。
- 以下のハンドル演算のサポートが拡張され、以下が可能になりました。
  - 同じ構造体タイプが関連付けられた複数のハンドルを比較する。
  - 関連する構造体タイプを考慮して、ハンドルへの加算およびハンドルからの減算を行う。
  - 関連する構造体タイプを考慮して、2 つのハンドルの相違を計算する。
- WIDECHAR ストリングの最大長が 32767 に拡張されました。
- 一部の状況で、LIKE 属性を使用した宣言がサポートされるようになりました。
- PROCEDURE ステートメントに対して SUPPRESS 属性を使用できます。
- OPTIONAL パラメーターのサポートにより、省略された OPTIONAL パラメーターを入り口の引数として渡すことができるようになりました (その入り口の宣言内の対応するパラメーターもまた OPTIONAL である場合に限ります)。
- INOUT および OUTONLY で BYADDR が暗黙指定されるようになりました。
- 新規の ALLCOMPARE 組み込み関数により、2 つの構造体の比較がサポートされます。
- USURROGATE は、 USUPPLEMENTARY 組み込み関数で置き換えられました。
- 新規の組み込み関数 UTF8、UTF8TOCHAR、および UTF8TOWCHAR により、CODEPAGE オプションを考慮した、CHAR と UTF-8 の間の変換が可能になりました。 また、これらにより、UTF-8 と UTF-16 の間の変換も簡素化されます。 さらに、ユーザーは UTF8 関数を使用して、UTF-8 リテラルを作成し、UTF-8 データで静的変数を初期設定できます。

## 4.2 からの機能拡張

本リリースは、本書および他の IBM PL/I ブックに記載されている次の機能強化を提供します。

- XMLCHAR 組み込み関数の使用による XML 生成は、XML 属性と、ヌル値の省略をサポートするようになりました。
- コンパイラーは、REFER を指定しない、調整可能な BASED のいくつかの用法を明示的にサポートするようになりました。
- コンパイラーは、POINTER とヌル・ストリング ('' および ''b) との比較をサポートするようになりました。
- コンパイラーは、単一のコンパイルで許可される個別のインクルード・ファイルの最大数を、2047 から 4095 に引き上げています。
- 新規の BY DIMACROSS 形式の代入により、SQL 複数行フェッチの結果を処理するコードの作成がさらに簡単になりました。

- コンパイラーとプリプロセッサー (EXEC SQL コードの構文解析時の単なる SQL プリプロセッサーではない) はすべて、<> を不等号記号としてサポートするようになりました。
- 新規の INDICATORS 組み込み関数により、配列を、構造を持つ SQL 標識変数として宣言することが容易になります。
- 新規の POPCNT 組み込み関数は、x の対応するバイトにおいて 1 になっているビットの数をバイトごとに保持する FIXED BIN 値を返します。

## 4.1 からの機能拡張

本リリースは、本書および他の IBM PL/I ブックに記載されている次の機能強化を提供します。

- 新規 PLISAXD 組み込みサブルーチンは、スキーマに対する妥当性検査により、XML 文書の解析機能を提供します。
- 新規 ONAREA 組み込み関数を使用すると、以前は実行時エラー・メッセージまたはメモリー・ダンプでのみ参照可能であった情報の別の部分、つまり AREA 条件が発生した AREA 参照の名前に簡単にアクセスすることができます。
- 新規 VALUE 型付き関数は、対応する構造体型を持つ変数の初期化または代入をサポートします。
- INITIAL 属性を DEFINE STRUCTURE ステートメントの基本名で使用できます。

## 3.9 からの機能拡張

本リリースは、本書および他の IBM PL/I ブックに記載されている次の機能強化を提供します。

- 新規の MEMCU12、MEMCU21、MEMCU14、MEMCU24、MEMCU41、および MEMCU42 組み込み関数は、UTF-8、UTF-16、および UTF-32 間の変換機能を提供します。z/OS では、対応するハードウェア命令を利用したインライン・コードでこの変換を行います。
- 新規の PLITRAN11、PLITRAN12、PLITRAN21、および PLITRAN22 組み込み関数は、1 バイトおよび 2 バイト・バッファーの変換機能を提供します。z/OS では、対応するハードウェア命令を利用したインライン・コードでこの変換を行います。
- 新規の USURROGATE 組み込み関数は、CHAR または WCHAR ストリングに UTF サロゲート・ペアが含まれているかどうかをテストする機能を提供します。
- 新規の ROUNDDEC 組み込み関数は、DFP 数を小数第 n 位で丸める (ROUND 組み込み関数の場合は、n 桁目での丸め) かどうかを指定する機能を提供します。

  **注 :** PL/I for z/OS V5.1 以降、この組み込み関数は ROUNDAWAYFROMZERO に名前変更されています。
- 新規の INONLY、INOUT、および OUTONLY 属性を使用すれば、コードをより自己記述的にし、コンパイラーで、より正確な診断を生成する (例えば、仮引数が INONLY と宣言されていれば仮引数にフラグを設定できなくなり、未初期化引数が OUTONLY と宣言されていれば未初期化引数にフラグを設定できなくなりました) ことが簡単にできます。
- 新規の %DO SKIP; ステートメントを利用すれば、簡単にコード・ブロックをコンパイルから除外し、コメントを「コメント化」できます。
- 追加で 6 つの日付/時刻パターンが、入出力に対してゼロ消去をサポートするようになりました。

## 3.8 からの機能拡張

本リリースは、本書および他の IBM PL/I ブックに記載されている次の機能強化を提供します。

- 新規 PLISAXC 組み込み関数によって、ユーザーは、z/OS XML System Services パーサーを、SAX パーサーのように活用できます。 このパーサーの基礎サポートのおかげで、PLISAXC で、名前空間および 2G を超える文書がサポートされます。
- 新規 ULENGTH、ULENGTH8、ULENGTH16、UPOS、USUBSTR、UVALID、および UWIDTH 組み込み関数によって、ユーザーは、UTF-8 および UTF-16 データが入ったストリングの照会および処理を行えます。
- 新規 FIXEDBIN、FIXEDDEC、FLOATBIN、および FLOATDEC 組み込み関数によって、ユーザーは、数値変換で (モード以外の) すべての結果属性を指定できるため、 より分かりやすいだけではなく、パフォーマンスも向上したコードを作成できます (特に一部の DFP 変換において)。

- 新規 ONLINE 組み込み関数を使用すると、以前は実行時のエラー・メッセージまたはダンプでしか使用できなかった情報のもう一方の部分、つまり、条件が発生したユーザー・コードの行番号に簡単にアクセスすることができます。
- 新規 REG12 組み込み関数によって、CAA のアドレスが返され、ユーザーは、一部の Language Environment サービスを使用したコードを作成しやすくなります。
- REPATTERN 組み込み関数がさらに 3 つの Db2 日時フォーマットをサポートします。
- 新規 DIMACROSS 属性によって、Db2 複数行フェッチの活用が容易になります。
- 新規 SUPPRESS 属性によって、未初期化および未参照の変数に対するコンパイラー警告メッセージを選択的に抑止するのが容易になります。
- 末尾の OPTIONAL 引数が内部プロシージャーの呼び出しでも省略可能になりました。
- USAGE コンパイラー・オプションの新規 HEX サブオプションによって、ユーザーは HEX 組み込み関数を VARYING および VARYINGZ ストリングに適用する際のデータの表示量を指定できます。

## 3.7 からの機能拡張

本リリースは、本書および他の IBM PL/I ブックに記載されている次の機能強化を提供します。

- IEEE 10 進浮動小数点 (DFP) がサポートされます。

  これには、次の新しい組み込み関数のサポートが含まれています。

  - ISFINITE
  - ISINF
  - ISNAN
  - ISNORMAL
  - ISZERO

  また、DFP サポートの一環として、次の古い組み込み関数が更新されました。

  - EPSILON
  - EXPONENT
  - HUGE
  - MAXEXP
  - MINEXP
  - PLACES
  - PRED
  - RADIX
  - ROUND
  - SCALE
  - SUCC
  - TINY
- 新規の MEMCONVERT 組み込み関数を使用すると、任意のコード・ページ間で任意の長さのデータを変換できます。
- 新規の ONOFFSET 組み込み関数を使用すると、以前は実行時のエラー・メッセージまたはダンプ、つまりある条件が発生したユーザー・プロシージャーのオフセットでのみ使用可能だった他の情報に簡単にアクセスすることができます。
- 新規 STACKADDR 組み込み関数は、現在の動的保存域 (z/OS 上のレジスター 13) のアドレスを戻し、ユーザーが独自の診断コードを作成するのを容易にします。
- 新規の QUOTE オプションを使用すると、引用符 (") 記号に大体コード・ポイントを指定することができます。これは、この記号がコード・ページ・インバリアントではないためです。
- 新規の XML コンパイラー・オプションを使用すると、XMLCHAR 組み込み関数の出力内のタグを、すべて大文字にするか、大/小文字を宣言で使用した とおりに指定できます。

## 3.6 からの機能拡張

このリリースでは、新しい言語機能が提供されます。 その他のプラットフォーム固有の機能拡張については、該当するプログラミング・ガイドに説明があります。

- PICSPEC 組み込み関数がサポートされるようになりました。そのため、CHARACTER データをすぐに PICTURE にキャストできます。
- THREADID 組み込み関数が z/OS で使用できるようになりました。 変更も加えられており、スレッド ID にポインターを戻すようになりました。また、すべての場合にパラメーターが必須となっています。

## 3.5 からの機能拡張

このリリースでは、新しい言語機能が提供されます。 その他のプラットフォーム固有の機能拡張については、該当するプログラミング・ガイドに説明があります。

- LOCATION 組み込み関数は、構造体が割り振られていなくても、REFER を使用して構造体の中の最初のエレメントを指定できるようになりました。
- Db2 の日付パターンである「YYYY-MM-DD」、「MM/DD/YYYY」および「DD.MM.YYYY」を日時処理の組み込み関数で使用できるようになりました。

## 3.4 からの機能拡張

このリリースでは、新しい言語機能が提供されます。 その他のプラットフォーム固有の機能拡張については、該当するプログラミング・ガイドに説明があります。

- DEFAULT ステートメントの意味が、ホスト・コンパイラーでの意味と一致するようになりました。
- ENTRY ステートメントが含まれている PROCEDURES 内の BEGIN ブロックの内部での RETURN ステートメントのサポート。
- REPLACEBY2 組み込み関数
- NOINIT 属性
- MACRO プリプロセッサーの中の以下の組み込み関数
  - LOWERCASE
  - MACNAME
  - TRIM
  - UPPERCASE

## 3.3 からの機能拡張

このリリースでは、以下の言語機能を含む、Enterprise PL/I V3R3 で提供されたすべての機能拡張 も提供しています。

- 以下の組み込み関数
  - MEMINDEX
  - MEMSEARCH
  - MEMSEARCHR
  - MEMVERIFY
  - MEMVERIFYR
  - XMLCHAR
- GET EDIT 内の V フォーマット項目

## 3.2 からの機能拡張

このリリースでは、以下の言語機能を含む、Enterprise PL/I V3R2 で提供されたすべての機能拡張 も提供しています。

- OPTIONS(COBOL) を指定した PROC および ENTRY に対する NOMAP、 NOMAPIN、および NOMAP 属性のサポート。
- 異なる RETURNS 属性を持つ ENTRY ステートメントを指定した PROC に対する、旧ホスト・コンパイラーで提供していたのと同じ方法でのサポート。
- OPTIONS(RETCODE) は、OPTIONS(COBOL) を指定した PROC および ENTRY を想定します。
- 未処理の場合、SIZE 条件は ERROR にプロモートされません。
- 新規の USAGE コンパイラー・オプションを使用すると、 RULES(IBM|ANS) オプションの他の影響を受けずに、ROUND および UNSPEC 組み込み関数 の IBM または ANS 動作を完全に制御できます。
- PUT LIST と PUT EDIT ステートメントで POINTER が使用できるように なりました。8 バイトの 16 進値が出力されます。
- ABNORMAL 属性を STATIC 変数で指定すると、STATIC 変数が 使用されていなくてもこの変数は保存されます。

## 3.1 からの機能拡張

本リリースは、本書および他の IBM PL/I ブックに記載されている次の機能強化を提供します。

本リリースでは、以下を含む Enterprise PL/I V3R1 で提供されたすべての機能強化も提供しています。

- z/OS でのマルチスレッド化のサポート
- z/OS での IEEE 浮動小数点のサポート
- マクロ・プリプロセッサーでの ANSWER ステートメントのサポート
- PLISAXA および PLISAXB 組み込みサブルーチンを介した SAX 形式 XML 構文解析
- 追加の組み込み関数
  - CS
  - CDS
  - ISMAIN
  - LOWERCASE
  - UPPERCASE

本リリースでは、以下を含む VisualAge PL/I V2R2 で提供されたすべての機能強化も提供しています。

- WIDECHAR 属性を介した初期 UTF-16 サポート

  以下については、まだサポートされていません。

  - ソース・ファイル内の WIDECHAR 文字
  - W ストリング定数
  - ストリーム入出力内の WIDECHAR 式の使用
  - レコード入出力での WIDECHAR からの暗黙の型変換または WIDECHAR への暗黙の型変換
  - レコード入出力での暗黙の endianness フラグ

    WIDECHAR ファイルを作成する場合は、 ファイルの最初の 2 バイトとして endianness フラグ ('fe_ff'wx) を書き込んでください。
- DEFAULT ステートメントでサポートされる DESCRIPTORS オプションと VALUE オプション
- PUT DATA 機能強化
  - POINTER、OFFSET、およびサポートされているその他の非計算変数
  - タイプ 3 DO 仕様が使用可能
  - 添え字が使用可能
- DEFINE ステートメントの機能強化
  - 指定されていない構造体の定義
  - CAST および RESPEC タイプ付き関数

- 追加の組み込み関数
  - CHARVAL
  - ISIGNED
  - IUNSIGNED
  - ONWCHAR
  - ONWSOURCE
  - WCHAR
  - WCHARVAL
  - WHIGH
  - WIDECHAR
  - WLOW
- プリプロセッサー機能強化
  - プリプロセッサー・プロシージャーでの配列のサポート
  - %DO ステートメントでの WHILE、UNTIL および LOOP キーワードのサポート
  - %ITERATE ステートメントのサポート
  - %LEAVE ステートメントのサポート
  - %REPLACE ステートメントのサポート
  - %SELECT ステートメントのサポート
  - 追加の組み込み関数
    - COLLATE
    - COMMENT
    - COMPILEDATE
    - COMPILETIME
    - COPY
    - カウンター
    - DIMENSION
    - HBOUND
    - INDEX
    - LBOUND
    - LENGTH
    - MACCOL
    - MACLMAR
    - MACRMAR
    - MAX
    - MIN
    - PARMSET
    - QUOTE
    - REPEAT
    - SUBSTR
    - SYSPARM
    - SYSTEM
    - SYSVERSION
    - TRANSLATE

- VERIFY

# ご意見の送付方法

本書または PL/I の他のマニュアルについてご意見がありましたら、IBM 発行のマニュアルに関する情報の Web ページ (http://www.ibm.com/jp/manuals/) よりお送りください。今後の参考にさせていただきます。(URL は、変更になる場合があります) この資料または他の PL/I 資料に関するご意見については、以下のいずれかの方法で IBM にご連絡ください。

- compinfo@cn.ibm.com に E メールを送信します。

  資料名、資料番号、PL/I のバージョン、および該当する場合はご意見の対象であるテキストの具体的な場所 (例: ページ番号) を必ず記載してください。
- この資料の巻末にある Readers' Comment Form に記入して、以下の住所に郵便で送付するか、IBM 担当員に提出する。 If the form has been removed, address your comments to:

  IBM
  Reader Comments
  H150/090
  IBM World Trade Asia Corporation
  San Jose, CA 95141-1003
  :NONE.
- 米国の FAX 番号 (800)426-7773 にご意見をファックスで送る

お客様が IBM に情報を送信する場合、お客様は、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、当該情報を使用または配布する非独占的権利を IBM に付与します。

# アクセシビリティー

アクセシビリティー機能は、運動や視覚などに障害を持つユーザーが情報技術製品を快適に使用できるように支援します。 z/OS® のアクセシビリティー機能には、Enterprise PL/I に対するアクセシビリティーが備わっています。

## アクセシビリティー機能

z/OS には、以下の主要アクセシビリティー機能が含まれています。

- スクリーン・リーダーおよび画面拡大ソフトウェアで一般的に使用されるインターフェース
- キーボードのみによるナビゲーション
- 色、コントラスト、フォント・サイズなどの表示属性をカスタマイズする機能

z/OS は、最新の W3C 標準である WAI-AL1.0 (http://www.w3.org/TR/wai-aria/)を使用して、 US セクション 508 (https://www.access-board.gov/ict/) および Web Content Accessibility Guidelines (WCAG) 2.0 (http://www.w3.org/TR/WCAG20/)に確実に準拠します。 アクセシビリティー機能を利用するには、最新リリースのスクリーン・リーダーを、この製品でサポートされる最新の Web ブラウザーと併用してください。

## キーボード・ナビゲーション

ユーザーは、TSO/E または ISPF を使用して z/OS user ユーザー・インターフェースにアクセスできます。

また、ユーザーは IBM Developer for z/OS を使用して z/OS サービスにアクセスすることもできます。

これらのインターフェースへのアクセスに関する情報については、以下の資料を参照してください。

- *z/OS TSO/E 入門*
- *z/OS TSO/E ユーザーズ・ガイド*
- *z/OS ISPF ユーザーズ・ガイド第 1 巻*
- *IBM Developer for z/OS の資料*

上記の資料には、 キーボード・ショートカットまたはファンクション・キー (PF キー) の使用方法を 含む TSO/E および ISPF の使用方法が記載されています。 それぞれの資料では、PF キーのデフォルトの設定値とそれらの機能の変更方法についても説明しています。

## インターフェースに関する情報

Enterprise PL/I オンライン製品資料は IBM Knowledge Center で利用でき、標準の Web ブラウザーで表示できます。

PDF ファイルでのアクセシビリティー・サポートは限定的です。 PDF 資料では、オプションのフォント拡大機能およびハイコントラスト表示設定を使用でき、キーボードのみでナビゲートできます。

ご使用のスクリーン・リーダーが構文図やソース・コード例を正確に読み上げたり、 ピリオドやコンマといった PICTURE 記号を含むテキストを正確に読み上げたりできるようにするには、ご使用のスクリーン・リーダーがすべての句読点を読み上げるように設定する必要があります。

支援技術製品は、z/OS にあるユーザー・インターフェースと連携して機能します。 ガイダンス情報については、z/OS インターフェースへのアクセスに使用する支援技術製品の資料を参照してください。

## アクセシビリティー関連情報

標準 IBM ヘルプ・デスクとサポート Web サイトに加え、IBM は、聴覚が不自由なお客様が営業やサポート・サービスにアクセスするために使用できる TTY 電話サービスを立ち上げました。

TTY サービス
800-IBM-3383 (800-426-3383)
(北米内)

## IBM のアクセシビリティーへの取り組み

IBM のアクセシビリティーへの取り組みについて詳しくは、IBM Accessibility (www.ibm.com/able) を参照してください。

# 第 1 章 プログラムのエレメント

この章では、PL/I プログラムを作成するときに使用する基本エレメントについて説明します。 エレメントには、文字セット、プログラマー定義の ID、キーワード、区切り文字、 およびステートメントが含まれます。

PL/I は、1 バイト文字セット (SBCS) および 2 バイト文字セット (DBCS) を サポートしています。

PL/I 言語要素の実装制限は 615 ページの『付録 A 制限値』に記載されています。

## 1 バイト文字セット

文字セット とは、文字 と呼ばれる固有表記の順序付きセットです。 例えば、モールス符号の記号セット、またはキリル・アルファベットの文字が文字セットです。

CODEPAGE コンパイラー・オプションは、コンパイラーが CHARACTER から UTF-8 と UTF-16 へ、および UTF-8 と UTF-16 から CHARACTER へとデータを変換する方法を決定します。 さらに、CODEPAGE コンパイラー・オプションは、UTF8TOCHAR 組み込み関数によって返される値も決定します。 例えば、16 進数でのユーロ記号の UTF-8 文字は E282AC です。 CODEPAGE(1140) では E282AC に対して返される値は '9F'x ですが、CODEPAGE(1142) では E282AC に対して返される値は '5A'x です。

PL/I は、CODEPAGE コンパイラー・オプションで許可される文字セットをサポートしています。 これらのコード・ページ内の多くの文字は インバリアント です。それらは、すべての EBCDIC コード・ページおよびすべての ASCII コード・ページで同じ 16 進数の値を持ちます。 インバリアント 文字のセットには、英語のアルファベット、10 桁の 10 進数、および PL/I によって使用されるその他の特殊文字が含まれます。 インバリアント でない特殊文字については、BRACKETS、CURRENCY、NAMES、NOT、OR、および QUOTE の各コンパイラー・オプションを使用して、これらの記号に対してソース内でどのような値が受け入れ可能かをコンパイラーに認識させることができます。

定数およびコメントでは、SBCS 文字のすべての値を使用することができます。

PL/I エレメント (ステートメント、キーワード、および区切り文字など) は、以下のセクションで説明する文字に限定されています。

### 英字および特別言語文字

PL/I のデフォルトの英字は、英字に特別言語文字が加わります。

#### 英字

英語アルファベットを含む 26 の基本英字があります。 それぞれの英字とそれに対応する ASCII および EBCDIC 値 (16 進表記) を 1 ページの表 1 に示します。

表 *1.* 英字と同等の値

| 文字 | EBCDIC 16 進値 (大文字) | EBCDIC 16 進値 (小文字) | ASCII 16 進値 (大文字) | ASCII 16 進値 (小文字) |
|---|---|---|---|---|
| A | C1 | 81 | 41 | 61 |
| B | C2 | 82 | 42 | 62 |
| C | C3 | 83 | 43 | 63 |
| D | C4 | 84 | 44 | 64 |
| E | C5 | 85 | 45 | 65 |
| F | C6 | 86 | 46 | 66 |
| G | C7 | 87 | 47 | 67 |

© Copyright IBM Corp. 1999, 2022

表 1. 英字と同等の値 (続き)

| 文字 | EBCDIC 16 進値 (大文字) | EBCDIC 16 進値 (小文字) | ASCII 16 進値 (大文字) | ASCII 16 進値 (小文字) |
|---|---|---|---|---|
| H | C8 | 88 | 48 | 68 |
| I | C9 | 89 | 49 | 69 |
| J | D1 | 91 | 4A | 6A |
| K | D2 | 92 | 4B | 6B |
| L | D3 | 93 | 4C | 6C |
| M | D4 | 94 | 4D | 6D |
| N | D5 | 95 | 4E | 6E |
| O | D6 | 96 | 4F | 6F |
| P | D7 | 97 | 50 | 70 |
| Q | D8 | 98 | 51 | 71 |
| R | D9 | 99 | 52 | 72 |
| S | E2 | A2 | 53 | 73 |
| T | E3 | A3 | 54 | 74 |
| U | E4 | A4 | 55 | 75 |
| V | E5 | A5 | 56 | 76 |
| W | E6 | A6 | 57 ヶ所 | 77 |
| X | E7 | A7 | 58 | 78 |
| Y | E8 | A8 | 59 | 79 |
| Z | E9 | A9 | 5A | 7A |

### 特別言語文字

デフォルトの特別言語文字は、番号記号 (#)、単価記号 (@)、および通貨記号 ($) です。これらの文字の 16 進値は、コード・ページに応じて変化します。 NAMES コンパイラー・オプションを使用すると、 独自の特別言語文字を定義することができます。 特別言語文字を定義する方法の詳細については、「プログラミング・ガイド」を参照してください。

### 英数字

英数字 は、英字または特別言語文字、あるいは数字のいずれかです。

## 10 進数

PL/I は、10 桁の 10 進数 (0 から 9) を認識します。 単に数字とも呼ばれ、10 進定数やその他の表現や値を記述するために使用されます。

以下の表では、数字とその 16 進値を表しています。

表 *2. 10* 進数と同等の値

| 文字 | EBCDIC 16 進値 | ASCII 16 進値 |
|---|---|---|
| 0 | F0 | 30 |
| 1 | F1 | 31 |
| 2 | F2 | 32 |
| 3 | F3 | 33 |
| 4 | F4 | 34 |
| 5 | F5 | 35 |
| 6 | F6 | 36 |
| 7 | F7 | 37 |
| 8 | F8 | 38 |
| 9 | F9 | 39 |

## 2 進数

PL/I は、0 と 1 の 2 進数字を認識します。 これらはビットとも呼ばれ、バイナリー定数およびビット定数の記述に使用されます。

## 16 進数

PL/I は、16 桁の 16 進数字 (0 から 9 および A から F) を認識します。 A から F は、それぞれ 10 から 15 までの 10 進値を表します。 16 進数は、16 進数字、または単に 16 進と呼ばれることもあり、 定数を 16 進表記で表す場合に使用されます。

## 特殊文字

このトピックでは、特殊文字、それらの PL/I での意味、 ならびに ASCII と EBCDIC 値 (16 進表記) を示します。

表 *3.* 特殊文字と同等の値

| 文字 | 意味 | デフォルトの EBCDIC 16 進値 | デフォルトの ASCII 16 進値 |
|---|---|---|---|
| b | ブランク | 40 | 20 |
| = | 等号記号または代入記号 | 7E | 3D |
| + | 正符号 | 4E | 2B |
| - | 負符号 | 60 | 2D |
| * | アスタリスクまたは乗算記号 | 5C | 2A |
| / | スラッシュまたは除算記号 | 61 | 2F |
| ( | 左括弧 | 4D | 28 |
| ) | 右括弧 | 5D | 29 |
| , | コンマ | 6B | 2C |
| . | 小数点またはピリオド | 4B | 2E |
| ' | 単一引用符 | 7D | 27 |

表 3. 特殊文字と同等の値 (続き)

| 文字 | 意味 | デフォルトの EBCDIC 16 進値 | デフォルトの ASCII 16 進値 |
|---|---|---|---|
| " | 二重引用符 [1] | 7F | 22 |
| % | パーセント | 6C | 25 |
| ; | セミコロン | 5E | 3B |
| : | コロン | 7A | 3A |
| ¬ | NOT 記号、排他的 OR 記号 [1] | 5F | 5E |
| & | AND 記号 | 50 | 26 |
| \| | OR 記号 [1] | 4F | 7C |
| > | より大記号 | 6E | 3E |
| < | より小記号 | 4C | 3C |
| _ | 区切り文字 (下線) | 6D | 5F |

**注:**

1. OR (|)、NOT (¬)、および引用符 (") 記号には、可変コード・ポイントがあります。 コンパイラー・オプションの OR、NOT、および QUOTE を使用して、これらの演算子を表す代替記号を定義することができます。 これらのオプションの詳細については、「プログラミング・ガイド」を参照してください。

## 複合記号

特殊文字を組み合わせて、複合記号を作成することができます。

下の表では、これらの記号とその意味を示しています。 複合記号には、ブランクを入れることはできません。

表 4. 複合記号の説明

| 複合記号 | 意味 |
|---|---|
| ∥ | 連結 |
| ** | 累乗演算 |
| ¬< | より小でない |
| ¬> | より大きくない |
| ¬= または <>[1] | 等しくない; 評価、排他 OR および代入 |
| <= | より小か等しい |
| >= | より大か等しい |
| /* | コメントの始まり |
| */ | コメントの終わり |
| // | 行末コメント標識 |
| –> | ロケーター (ポインターとオフセット) |
| => | ロケーター (ハンドル) |
| += | 式の計算、加算および代入 |
| -= | 式の計算、減算および代入 |

表 4. 複合記号の説明 (続き)

| 複合記号 | 意味 |
|---|---|
| *= | 式の計算、乗算および代入 |
| /= | 式の計算、除算および代入 |
| \|= | 式の計算、OR および代入 |
| &= | 式の計算、AND および代入 |
| ∥= | 式の計算、連結と代入 |
| **= | 式の計算、指数化および代入 |
| (: | タイプ付き関数仮パラメーター・リストの始まり |
| :) | タイプ付き関数仮パラメーター・リストの終わり |

**注:**

1. ¬= の代わりに <> を使用することができます。

# 大文字小文字の区別

PL/I プログラム内では、 大文字と小文字を組み合わせて使用することができます。

キーワードまたは ID の使用では、小文字はそれに対応する大文字と見なされます。 DBCS 文字として小文字を入力した場合でも、大文字に変換されます。

コメントまたは文字、混合または漢字ストリング定数内で大文字と小文字を 組み合わせて使用する場合には、小文字は小文字のままです。

# SBCS ステートメント・エレメント

このセクションでは、1 バイト文字セット (SBCS) を使用する PL/I プログラムを構成する要素について説明します。

PL/I ステートメントは、ID、区切り文字、演算子、および定数で構成されます。 定数は、15 ページの『第 2 章 データ・エレメント』で説明します。

## ID

*ID* は、コメントや定数の中には含まれない文字の集まりです。

P、PIC、および PICTURE 以外は、ID が先になり、 区切り文字がそのあとに続かなければなりません。 (P、PIC、および PICTURE の各 ID のあとは、文字ストリングが続きます。) ID の最初の文字は、英字、特別言語文字のいずれかにならなければなりません。 ID 名が INTERNAL シンボルである場合、区切り (_) 文字を最初の文字にすることもできます。 残りの文字としては、英字、特別言語文字、数字、または下線 (_) 文字が可能です。 外部ユーザー名は、IBM、PLI、CEE、_IBM、_PLI、 および _CEE 以外の文字で始まる名前ではいけません。

ID は、PL/I キーワードまたはプログラマーが定義する名前となります。 ID がキーワードである場合には、PL/I は構文から判別することができるので、 プログラマーが定義した名前として任意の ID を使用することができます。 PL/I では予約されている 語はありません。 ただし、IF や THEN などの一部のキーワードを変数名として使用すると、プログラムが不必要に難解なものになる場合があります。

### *PL/I* キーワード

キーワード は、PL/I 内で特定の意味を持つ ID です。

取られる処置またはデータの属性などを、キーワードで指定することができます。 例えば、READ、DECIMAL、および ENDFILE などはキーワードです。 キーワードによっては省略形を使うことができます。 キーワードとその省略形は、大文字で示されます。

### プログラマー定義の名前

PL/I プログラムでは、 変数やプログラム制御データに名前 を付けます。 そのほかに、組み込み名、条件名、および総称名があります。

任意の ID を名前として使用することができます。 名前はプログラム・ブロック内でそれぞれ固有でなければなりません。 同一ブロック内のファイルと浮動小数点変数の両方に同じ名前を付けることはできません。

読みやすくするために、Gross_Pay のように中間に 区切り文字 (_) を入れることができます。

以下に示すのは名前の例です。

```
A                   Rate_of_pay
Record              Loop_3
```

プログラマー定義の外部名に関するその他の要件については、152 ページの『INTERNAL 属性と EXTERNAL 属性』 を参照してください。

名前が必要であるにもかかわらず名前を使用しない場合は、アスタリスク (*) を ID として使用することができます。 詳しくは、ENTRY 属性の説明にある例を参照してください。

## 区切り文字と演算子

区切り文字 と演算子 は、ID と 定数を分離するのに使われます。

6 ページの表 5 は区切り文字を示します。

表 *5.* 区切り文字

| 名前 | 区切り文字 | 用途 |
|---|---|---|
| コンマ | , | リスト内の各エレメントを区切ります。BY NAME オプションの前でも使います。 |
| 期間 | . | 修飾名の各エレメントを接続します。10 進数または 2 進数の小数点としても使います。 |
| セミコロン | ; | ステートメントの終わりを表します。 |
| 等号 | = | 代入を表すほか、条件式では「等しい」ことを表します。 |
| コロン | : | ステートメントに接頭語をつなぐときに使います。次元属性で、下限と上限を連結するときや、DEFAULT ステートメントの RANGE 指定内でも使います。 |
| ブランク | b | エレメントを区切ります。 |
| 括弧 | ( ) | リスト、式、反復因数 (iteration factor)、および反復因数 (repetition factor) を囲みます。各種のキーワードに関連する情報を囲むときも使います。 |
| ロケーター | –> | ロケーター修飾 (ポインターとオフセット) を表します。 |
| | => | ロケーター修飾 (ハンドル) も表します。 |
| パーセント | % | % ステートメントおよび % ディレクティブを表します。 |

**注 :** 必要な記号を忘れると、トレース困難なエラーが起こることがあります。 よくあるエラーとしては、引用符が対になっていない、小括弧が対になっていない、コメント区切り文字が対になっていない、ならびにセミコロンが欠落しているなどがあります。

7 ページの表 6 は演算子を示します。

表 6. 演算子

| 演算子の タイプ | 文字 | 説明 |
|---|---|---|
| 算術 | + | 加算または前置加算 |
| | - | 減算または前置減算 |
| | * | 乗算 |
| | / | 除算 |
| | ** | 累乗演算 |
| 比較 | = | 次と等しい |
| | ¬= または <> | 等しくない |
| | < | より小 |
| | ¬< | より小でない |
| | > | より大 |
| | ¬> | より大きくない |
| | <= | より小か等しい |
| | >= | より大か等しい |
| 論理演算子 | ¬ | NOT、排他的 OR |
| | & | AND |
| | \| | または |
| ストリング | ∥ | 連結 |

区切り文字として使う文字をほかのコンテキストで使うことができます。 例えば、名前を修飾するときに使うピリオドは区切り文字 (例えば、 `Weather.Temperature`) ですが、 算術定数内で使用する場合は 10 進小数点です (例えば、`3.14`)。

## ブランク

各演算子または区切り文字の前後をブランク (b) で囲むことができます。

ブランク以外の区切り文字で区切られていない ID や定数は、1 つ以上のブランクで区切る必要があります。 ただし、この規則の唯一の例外として、ID P、PIC、 および PICTURE の後にはブランクなしで文字ストリングを続けることができます。 ブランクは、入れることが可能なところであれば、いくつでも入れることができます。

ID、複合記号、または定数 (文字定数、混合定数、ワイド文字、および漢字ストリング定数は除く) の中にブランクがあってはなりません。 以下の例を参照してください。

| | | |
|---|---|---|
| `ab+bc` | この宣言は、次の宣言と同等です。 | `Ab + Bc` |
| `Table(10)` | この宣言は、次の宣言と同等です。 | `TABLEb(b10bbb)` |
| `First,Second` | この宣言は、次の宣言と同等です。 | `first,bsecond` |
| `AtoB` | は次と等しくない | `AbtobB` |

以上のほかに、ブランクを必要また は許可する場合については、 それぞれの言語特性を説明 する際に言及します。

### コメント

ブランクを区切り文字として使用できるところでは、 コメントを使用することができます。 コメントは、ブランクと見なされ、区切り文字として使用されます。 コメントはプログラム内のロジックでは無視され、影響を及ぼしません。

以下の 2 種類のコメントがあります。

- /* (スラッシュ、アスタリスク) 文字の後に一連の文字 (改行を含む) が続き、さらにその後に */ 文字が続きます。 この場合、コメント化されるテキストは */ 文字で終わります。また、コメント化されるテキストは複数の行にまたがる場合があります。
- // (2 つのスラッシュ) 文字の後に一連の文字が続きます。 この場合、コメント化されるテキストは、// 文字を含む行の末尾で終わります。

1 番目の種類のコメントは、以下の例のように 1 つ以上の行に入力できます。

```
A = /* This comment is on one line */ 21;

    /* This comment spans
       two lines          */
```

2 番目の種類のコメントは、以下の例のように、コメントを開始した行で終わります。

```
A = 21 ; // This comment ends with the end of this line
```

以下の例では、コメントとして表示されるものは、 (引用符で囲まれているので) 実際には文字ストリング定数です。

```
A = '/* This is a constant, not a comment */' ;
```

コメントをネストすることはできません。 ただし、コメントを含むコードを「コメント化」する方法として %DO SKIP: ステートメントを使用することができます。

## ステートメント

ID、区切り文字、演算子、および定数を使用して、 PL/I ステートメントを作ることができます。

ソース・プログラムは一連のレコードまたは行から構成されますが、 PL/I は、プログラムを連続した文字のストリームと見なします。 PL/I ステートメントのフォーマットにはほとんど制約がないので、プログラムを作成する際に特別のコーディング規則に注意する必要はなく、 各ステートメントを特定の桁位置から始める必要もありません。 ステートメントは、前のステートメントのすぐあとの位置から開始できます。また、ブランク (いくつでもよい) で区切ることができます。

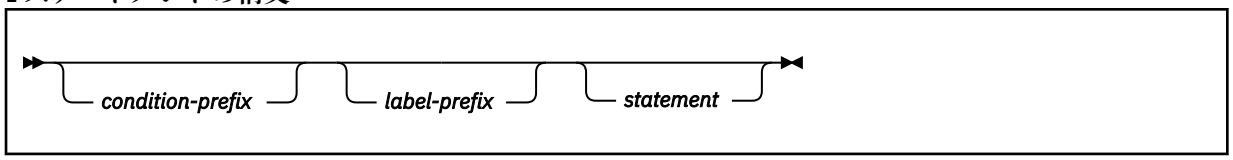
ステートメントの中には、% 記号で始まるものもあります。 これらのステートメントは、プリプロセッサーおよびコンパイラーの演算 (ライブラリーからのプログラム・ソース・テキストを始めとするリストの制御など) を指示する % ディレクティブであるか、または PL/I マクロ機能の % ステートメントであるかのいずれかです。 % ディレクティブは単独で 1 行を占めなければなりません。

プログラムの読みやすさと保守容易性を向上するため、またソース行に末尾ブランクがないことで予期しない結果が生じることを防ぐためには、以下のガイドラインに従ってください。

- 言語エレメントを分割して複数行に置くようなことがないようにします。 ストリング定数を複数行に記述する必要がある場合は、連結演算子 (∥) を使用してください。
- 1 行に複数のステートメントを記述することはできません。
- % ディレクティブを複数行に分割することはできません。

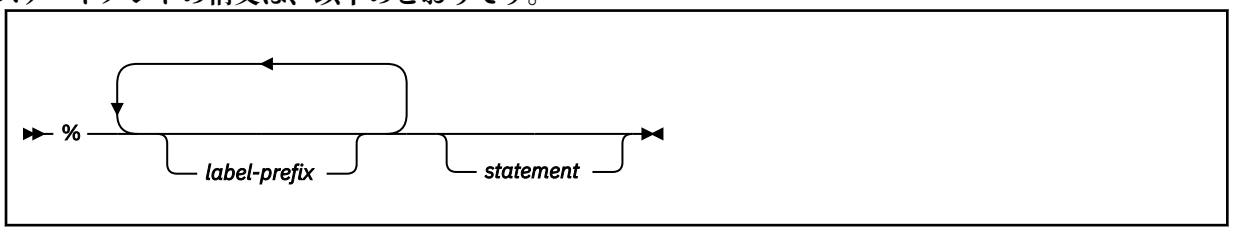
193 ページの『第 8 章 ステートメントとディレクティブ』では、PL/I ステートメント、マクロ機能 % ステートメント、および % ディレクティブを英字順にリストしています。

**PL/I ステートメントの構文**

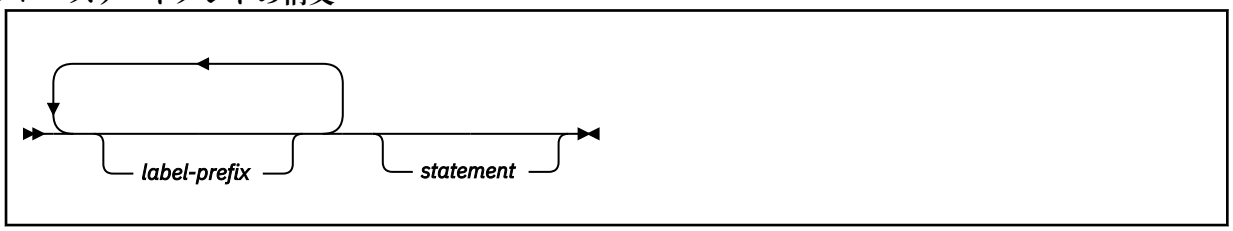



**% ディレクティブの構文**

**% ステートメントの構文は、以下のとおりです。**




**マクロ・ステートメントの構文**




各ステートメントは、いずれかのグループまたはブロックに含まれて いなければなりません。 マクロ・ステートメントは、なんらかのマクロ・グループ またはプロシージャーの中に囲まれていなければなりません。

**condition-prefix**
*condition-prefix* は、PL/I 条件の可能または禁止を指定します (339 ページの『第 15 章 条件処理』を参照)。

**label-prefix**
*label-prefix* は、1 つまたは複数のステートメント・ラベルです。 ステートメント・ラベルとは、 プログラム内のどこか別の場所からそのステートメントを参照できるように、 ステートメントを識別する名前です。 ステートメント・ラベルは、ラベル定数 (43 ページの『ラベル・データと LABEL 属性』を参照)、入り口定数 (110 ページの『入り口データ』を参照)、またはフォーマット定数 (44 ページの『フォーマット・データと FORMAT 属性』を参照) のいずれかになります。

DECLARE、DEFAULT、WHEN、OTHERWISE および ON ステートメント以外の任意のステートメントは、 ラベル接頭部を持つことができます。 ラベル接頭部の構文は以下のとおりです。

一般的にこの資料全体の独立ステートメントの構文は、条件接頭語または ラベル接頭部を示しません。

**ステートメント**
単純または複合ステートメントがあります。

# 単純ステートメント

単純ステートメントのタイプは、キーワード、代入、およびヌルです。

**キーワード・ステートメント**
キーワード・ステートメント は、キーワードで始まるステートメントです。 このキーワードは、ステートメントの機能を示します。

以下の例では、READ および DECLARE はキーワードです。

```
read file(In) into(Input);          /* keyword statement  */
%declare Text char;                 /* keyword %statement  */
```

**ASSIGNMENT ステートメント**
代入ステートメント には、 代入記号 (=) の左側にある 1 つまたは複数の ID、および右側の式が含まれます。 このステートメントはキーワードで始まりません。 以下の例を参照してください：

```
A = B + C;                          /*   assignment statement  */
%Size = 15;                         /* % assignment statement   */
```

**ヌル・ステートメント**
ヌル・ステートメント は、1 つのセミコロンだけで構成され、 演算できないステートメントです。 以下の例を参照してください：

```
      ;                             /* null statement         */
Label:;                             /* labeled null statement */
  % ;                               /* % null statement     */
```

## 複合ステートメント

複合ステートメントはすべてキーワード・ステートメントです。 各ステートメントは、そのステートメントの目的を示すキーワードで始まっています。 複合ステートメントには、1 つまたは複数の単純ステートメント または複合ステートメントが含まれます。

4 つの複合ステートメント、IF、ON、WHEN、および OTHERWISE があります。 複合ステートメントは、セミコロンで終了します。このセミコロンは、 複合ステートメントの最後のステートメントの終わりも示します。

**複合ステートメントの例**

```
on conversion
  onchar() = '0';

if Text = 'stmt' then
  do;
    select(Type);
      when('if') call If_stmt;
      when('do') call Do_stmt;
      when('') /* do nothing */ ;
      otherwise
        call Other_stmt;
    end;
    call Print;
  end;
end;

%if Type = 'AREA' %then
  %Size = Size + 16;
%else;
```

## グループ

ステートメントは、グループと呼ばれるさらに大きなプログラム単位内に入れることもできます。

グループ は、DO グループまたは SELECT グループのいずれかです。 DO グループとは、DO ステートメントとそれに対応する END ステートメントで区切られた一連のステートメントです。 SELECT グループは、SELECT ステートメントとそれに対応する END ステートメントで区切られた、一連の WHEN ステートメントとオプションの OTHERWISE ステートメントです。 区切りステートメントは、グループの一部です。

複合ステートメント内にグループがある場合、グループを単独ステートメントであるかのように処理して、そのグループに制御が渡されるか、 あるいはそのグループが迂回されることもあります。

グループ内での制御の流れについては、DO グループの場合は 207 ページの『DO ステートメント』の 項で説明し、SELECT グループの場合は 231 ページの『SELECT ステートメント』の項で説明します。

各グループは、いずれかのグループまたはブロック内に入れられなければなりません。 グループには、任意の数 (ゼロを含む) のステートメント、 グループ、またはグループが入ることができます。

# 2 バイト文字セット

2 バイト文字セット (DBCS) の各文字は、2 バイトで保存されます。 GRAPHIC コンパイラー・オプションが有効な場合は、 DBCS 文字や SBCS 文字を使用していくつかのソース言語エレメントを書くことができます。

特に、ソース・プログラムの以下の場所で DBCS 文字を使用できます。

- コメントの内部
- ステートメント・ラベルまたは ID の一部として
- G または M リテラル内

ただし、INCLUDE ファイル名と FETCH ステートメントの TITLE オプションは、SBCS でなければなりません。

z/OS プラットフォームでは、DBCS 文字の各ストリングをシフト・コードで直接囲む必要がありますが、コンパイラーでサポートされる他のプラットフォームでは、シフト・コードは不要であり、受け入れられません。 以降の例ではこれらのシフト・コードが含まれていますが、z/OS 以外のすべてのプラットフォームの場合にはそれらを省略する必要があります。

## DBCS ID

DBCS ID は、DBCS フォーマットの 1 バイト文字、2 バイト文字、 またはその両方を組み合わせたものでもかまいません。

### DBCS フォーマットの 1 バイト ID

1 バイト文字だけを含む DBCS ID は、 先頭文字規則も含めて通常の PL/I 命名規則に 従っていなければなりません。

DBCS と同等のものとして表された 1 バイト文字を含む DBCS ID は、 SBCS での同じ ID の同義語です。

**注記 :**

1. 本書では、記号 **.** (太字ピリオド) 使用して、SBCS としても表すことができる 2 バイト文字の最初のバイトを表します。
2. 本書では、「kk」を使用して、2 バイト文字を表します。
3. 本書では、「<」を使用してシフトアウト文字 ('0E'X) を表します。
4. 本書では、「>」を使用してシフトイン文字 ('0F'X) を表します。

**例**

```
<.I.B.M> = 3;    /* is the same as IBM=3; */
```

### 2 バイト文字を含む DBCS ID

DBCS 名に使用されるバイト数は、コンパイラーの LIMITS オプションで指定されている名前の最大長を値を超えてはなりません。

DBCS ID 内部の DBCS フォーマットで表された SBCS 文字は、SBCS と見なされます。 次の例を参照してください。

```
A<kk>B
A<kk.B>
<.Akk>B            /* are all A<kk>B  (SBCS-DBCS-SBCS) */
```

## 2 バイト文字 ID の使用

DBCS ID は、SBCS ID が許される個所ならどこでも、使用することができます。

DBCS ID を EXTERNAL 名および %INCLUDE ファイル名に使用する場合は、その ID がオペレーティング・システムで受け入れ可能であることを確認するか、環境名を持つ EXTERNAL 属性を使用して、または OPEN ステートメントの TITLE オプションを使用して受け入れ可能になっていることを確認する必要があります。

**関連情報**

EXTERNAL 属性
INTERNAL 属性と EXTERNAL 属性は、名前の有効範囲を指定します。

281 ページの『OPEN ステートメント』
OPEN ステートメントは、ファイルをデータ・セットに関連づけます。 これによって、OPEN ステートメントで指定された属性と DECLARE ステートメント で指定された属性が組み合わされます。 また、オープンされるファイルに必要な属性の一部がまだ宣言されていない場 合は、このステートメントでファイルの属性の指定を完成させることができます。

# DBCS ステートメント・エレメント

この節では、DBCS を使用しての PL/I 言語エレメントの作成方法に関する 補足情報を提供します。

この節における言語エレメントの定義および一般的な使用規則は、5 ページの『SBCS ステートメント・エレメント』の 対応する節に記載されています。

下のリストは、DBCS でコーディングできる言語要素を示しています。 規則の説明と使用例が含まれています。

**ID**

SBCS、DBCS またはその両方を使用します。

```
dcl Eof                 /* in SBCS, is the same as        */
dcl <.E.o.f>              /* this in DBCS.                  */

dcl <kkkk>X                /* these are all the same, where  */
dcl <kkkk.X>              /* kk is a valid                  */
dcl <kkkk>x               /* DBCS character                 */
dcl <kkkk.x>              /*                                */
```

**コメント**

SBCS、DBCS またはその両方を使用します。

```
/* comment */            /* all SBCS                      */
/* <.T.y.p.ekk> */       /* DBCS text */
```

コメント区切り文字を SBCS で指定しておく必要があります。

**混合文字ストリング定数**

SBCS、DBCS、またはその両方を使用し、SBCS または DBCS 引用符で囲みます。

ストリングが SBCS 引用符で囲まれている場合は、M 接尾部のあるストリングで終了する必要はありません。

引用符に囲まれた DBCS データは SBCS に変換されません。そのデータおよびシフト・コードは、ストリングの内部表記にそのままで保管されます。

```
'<.a.b.c>'M
'<.I.B.M.'.S>'M
```

```
'<.I.B.M>''<.S>'M
'IBM<kk>'M
```

**グラフィック定数**
DBCS のみを使用し、SBCS または DBCS 引用符で囲みます。

```
'<.a.b.c>'G                  /* 6 byte graphic constant  */
'<.I.B.M.'.S>'G            /* 10 byte graphic constant .I.B.M.'.S  */
```

G リテラルは、前後を DBCS 引用符で囲んで指定でき、またその場合は G 自体を DBCS で指定することもできます。 そのため、以下はすべて同等です。

```
'<.a.b.c>'G
<.'.a.b.c.'>G
<.'.a.b.c.'.G>
```

## DBCS 継続規則

ストリング・リテラルまたは ID に、単一の SBCS ブランクで 右マージンと区切られた DBCS 文字がある場合には、 そのブランクは無視され、ステートメントのエレメントは 次のレコードの左マージンに続きます。

# 第 2 章 データ・エレメント

この章では、PL/I プログラムで使用可能なデータの種類、およびデータを記述するために使用する属性について説明します。

この章では以下の情報を扱います。

- データ項目の検討
- 変数および定数の検討
- 使用可能なデータ・タイプおよびそれらを定義する属性

**関連情報**

147 ページの『データ宣言』
PL/I プログラムを実行すれば、特定のデータ・タイプのさまざまなデータ項目が処理されます。 名前が付いていない算術定数やストリング定数を除き、 プログラム内ではデータ項目はすべて名前によって参照されます。 各データ名には、(明示的または暗黙的な) 宣言によって、属性と意味が与えられます。 この章では、明示宣言および暗黙宣言、スカラー、配列、 構造体、および共用体の宣言、名前の有効範囲、データの位置合わせ、 デフォルトの属性を説明します。

## データ項目

データ項目 は、変数または定数のいずれかの値になります。 (これらの用語は、一般の数学的な使用法と必ずしも同じではありません。 用語については、次の節で述べています。)

データ項目は、 スカラー と呼ばれる単一の項目です。 あるいはデータ集合体 と呼ばれる項目の集まりであることもあります。

データ集合は、総体的にまたは個別に参照できるデータ項目の集まりです。 データ集合の種類は、配列、構造体、および共用体 です。 任意のタイプの計算データまたはプログラム制御データを使用して、データ集合を作成することができます。

**関連情報**

170 ページの『配列』
配列 は、同一の属性を持つ要素からなる n 次元の集合です。

174 ページの『構造体』
構造体 とは、構造体、共用体、要素変数、 および配列であるメンバー・エレメントの集合です。

175 ページの『共用体』
共用体 とは、相互にオーバーレイし合い、同じストレージを有するメンバー・エレメントの集合です。 メンバーは、構造体、共用体、要素変数、および配列であることができます。 それらは、同一の属性を持っていなくても構わない。

183 ページの『配列、構造体、および共用体の組み合わせ』
構造体または共用体に次元属性を指定すると、 それぞれ構造体の配列、 または共用体の配列 になります。 このような配列のエレメントは、同一の名前、レベル、 およびメンバーを持つ構造体または共用体です。

## 変数

変数 は、プログラムの実行中に変化し得る 1 つまたは複数の値を持ちます。

変数は宣言で使用され、そこで変数の名前および特定な属性を宣言します。 ただし、NONASSIGNABLE 属性の変数は、プログラムの実行中に変化しません。

変数参照 は、次のいずれかになります。

- 宣言されている変数名
- 以下の 1 つまたは複数から宣言された名前から派生した参照
  - ポインター修飾

© Copyright IBM Corp. 1999, 2022

- 構造体修飾
- 添え字付け

**関連情報**

51 ページの『式および参照』
この章では、式および参照のさまざまなタイプを説明しています。

257 ページの『ASSIGNABLE 属性および NONASSIGNABLE 属性』
ASSIGNABLE 属性と NONASSIGNABLE 属性は、関連づけられた変数が、割り当てのターゲットとなり得るかどうかを指定します。

## 定数

定数 は、変化しない値を持ちます。 計算データの定数は、DECLARE ステートメントで定数の値を宣言するか定数に名前を付けることによって参照されます。

名前のついた定数の宣言に関する詳細については、45 ページの『名前付き定数』を参照してください。

プログラム制御データの定数は、名前によって参照されます。

## 引用符の使用

ストリング定数、16 進定数、およびピクチャー指定は、 単一引用符または二重引用符のいずれかで囲まれます。

ストリング内の引用符には、以下の規則が適用されます。

- 組み込まれる引用符が、ストリングを囲むために使用されたものと同じタイプである場合には、組み込まれる 1 つの引用符ごとに 2 つの引用符を入力しなければなりません (つまり、'' または "")。
- 組み込まれる引用符が、ストリングを囲むのに使用されたものと異なるタイプである場合は、 組み込まれる各インスタンスごとに 1 つの引用符だけを入力します。 単一引用符は、データとして処理されます。

**例**

```
'Shakespeare''s "Hamlet"' is identical to
"Shakespeare's ""Hamlet"""

PICTURE "99V9" is identical to
PICTURE '99V9'
```

**注 :** 本書の構文図では、単一引用符を示しています。 二重引用符は、特に明記されていなければ、置き換えることができます。

## 句読点のついた定数

読みやすくするために、算術、ビット、および 16 進数の定数には、区切り文字 (_) を使用できます。

| | | |
|---|---|---|
| '1100_1010'B | は次と等しい | '11001010'B |
| 1100_1010B | は次と等しい | 11001010B |
| 'C_A'B4 | は次と等しい | 'ca'b4 |
| 'C_A'XN | は次と等しい | 'ca'XN |
| 16_777_216 | は次と等しい | 16777216 |

# データ・タイプとその属性

PL/I プログラムで使用されるデータは、 計算データまたはプログラム制御データに分類されます。

**計算データ**
求める結果を得るための計算に使用する値を表します。 計算データは算術データとストリング・データから構成されます。

算術データは、コード化算術データまたは数値ピクチャー・データのいずれかです。

コード化算術データ項目は、有理数です。 算術データ項目の属性は、基数 (BINARY または DECIMAL)、スケール (FLOAT または FIXED)、精度 (有効数字と小数点の配置)、 およびモード (REAL または COMPLEX) です。

39 ページの『数字データ』は、文字形式で保持されている数値データです。

ストリング は、単一データ項目として処理される一連の隣接する文字、ビット、グラフィック、uchar、またはワイド文字のことです。

**プログラム制御データ**
プログラムの実行を制御するために使用される値を表示します。 これは、区域、入り口、ラベル、ファイル、フォーマット、ポインター、およびオフセットのデータ・タイプで構成されます。

**例**

```
Area = (Radius**2) * 3.1416;
```

Area と Radius は、計算データのコード化算術変数です。 2 および 3.1416 は、計算データのコード化算術定数です。

数値 3.1416 がプログラム内の複数の場所で使用されている場合、または特定のデータまたは精度属性を必要とする場合は、名前付き定数として宣言できます。 名前付き定数の使用は、文書化の観点からも使用法の整合性の観点からも優れています。 したがって、上記のステートメントは以下のようにコーディングできます。

```
dcl Pi FIXED DECIMAL (5,4) VALUE(3.1416);
area = (radius**2) * Pi;
```

プログラム制御データの定数は、コンパイラーによって判別される値を持ちます。 下記の例では、loop という名前は プログラム制御データのラベル定数を表しています。 loop という値は、A=2*B; ステートメントのアドレスです。

```
loop:  A=2*B;
       C=B+6;
```

### 属性

データ項目を使用して作業するためには、 PL/I にデータ・タイプとその処理方法を知らせる必要があります。 属性 がこの情報を提供します。 属性の種類は、データ属性と非データ属性です。

## データ属性

データ属性は、計算データ、プログラム制御データ、およびプログラム特性を記述します。

| | | | |
|---|---|---|---|
| AREA | 固定 | ORDINAL | TYPE |
| BINARY | FLOAT | PICTURE | UCHAR |
| ビット | 形式 | POINTER | UNSIGNED |
| CHARACTER | GRAPHIC | PRECISION | UNION |
| 複合 | HANDLE | REAL | VARYING |
| DECIMAL | LABEL | RETURNS | VARYING4 |
| DIMENSION | LOCATES | SIGNED | VARYINGZ |
| ENTRY | NONVARYING | STRUCTURE | WIDECHAR |
| FILE | OFFSET | TASK | WIDEPIC |

## 非データ属性

非データ属性は、非データ要素 (組み込み関数など) を記述するか、他のデータ属性を持つ要素の追加説明を提供します。

| | | | |
|---|---|---|---|
| ABNORMAL | DEFINED | INTERNAL | RECORD |
| ALIGNED | DIMACROSS | KEYED | SEQUENTIAL |
| ASSIGNABLE | DIRECT | LIKE | STATIC |
| AUTOMATIC | ENVIRONMENT | LIST | STREAM |
| BASED | 排他的 | LITTLEENDIAN | UNALIGNED |
| BIGENDIAN | EXTERNAL | NONASSIGNABLE | UNBUFFERED |
| BUFFERED | GENERIC | NONCONNECTED | UPDATE |
| BUILTIN | HEXADEC | NORMAL | VALUE |
| BYADDR | IEEE | OPTIONAL | VALUELIST |
| BYVALUE | INDFOR | OPTIONS | VALUERANGE |
| CONDITION | INITIAL | OUTONLY | VARIABLE |
| connected | INONLY | OUTPUT | XMLATTR |
| CONTROLLED | INOUT | PARAMETER | XMLOMIT |
| | INPUT | POSITION | |
| | | PRINT | |

例えば、CHARACTER というキーワードは、 計算データのストリング・タイプのデータ属性です。 FILE というキーワードは、プログラム制御データのファイル・タイプのデータ属性です。 INTERNAL という有効範囲属性は、データ項目が宣言されるブロック内でのみ、そのデータ項目が認識されることを指定します。

キーワードおよび式を使用して属性を指定する方法について詳しくは、147 ページの『第 7 章 データ宣言』を参照してください。 手短に言えば、以下のいずれかの方法で属性を指定します。

- 明示的に DECLARE ステートメントを使用する方法
- コンテキストから PL/I が属性を判別する方法
- プログラマーが定義したデフォルトまたは言語によって定められたデフォルトを使用する方法

19 ページの表 7 と 21 ページの表 8 は、 PL/I の各種の属性と、計算データおよびプログラム制御データの 各タイプとの関連を示しています。これらの表は、定数と名前付き定数が、示されたデータ属性とスコープ属性 (19 ページの表 7) のみを持つことができることを示しています。変数は追加の属性 (21 ページの表 8) を指定できます。

次の例を考えてください。

```
Area = (Radius**2)*3.1416;
```

定数 3.1416 には、以下の属性が与えられます。

- DECIMAL

  これは、定数が明示的に 2 進定数ではないために与えられます。
- 固定

  これは、定数が固定小数点数であるために与えられます。
- PRECISION(5,4) (5 桁の有効数字で、そのうちの 4 桁は小数点の右側にあること)
- REAL

  これは、定数が虚数部を持たないために与えられます。
- INTERNAL および ALIGNED

(19 ページの表 7 の"「コード化算術式」"の行、および"「データ属性」"と"「有効範囲属性」"の列を参照してください。)

定数 1.0 (10 進固定小数点定数) は、定数 1 (もう 1 つの 10 進固定小数点定数)、'1'B (ビット定数)、'1' (文字定数)、1B (2 進固定小数点定数) または 1E0 (10 進浮動小数点定数) とは異なります。

以下の例では、変数 Pi は 5 桁の 10 進数で、そのうちの 4 桁は小数点の右側にあるという PRECISION のついた、FIXED および DECIMAL というプログラマー定義のデータ属性があります。

```
declare Pi fixed decimal(5,4) initial(3.1416);
```

この DECLARE ステートメントには Pi の他の属性が含まれていないため、PL/I は残りの属性のデフォルトを適用します。

- データ属性列からの REAL
- 位置合わせ属性列からの ALIGNED
- 有効範囲属性列からの INTERNAL
- ストレージ属性列からの AUTOMATIC
- データ属性列からの SIGNED

(21 ページの表 8 のコード化算術式の行を参照してください。)

表 7. 定数タイプによる属性の分類

| **定数タイプ** | **データ属性**[注 1 および 2] | **スコープ属性**[注 1 および 2] |
|---|---|---|
| コード化算術 | REAL \| 虚数<br>FLOAT \| FIXED<br>BINARY \| DECIMAL<br>PRECISION<br>SIGNED | 内部 |
| 名前付きのコード化算術式 | REAL \| COMPLEX<br>FLOAT \| FIXED<br>BINARY \| DECIMAL<br>PRECISION<br>VALUE<br>SIGNED \| UNSIGNED | 内部 |
| ストリング | BIT \| CHARACTER \| GRAPHIC \|<br>UCHAR \| WIDECHAR<br>(*length*) | 内部 |
| 名前の付いたストリング | BIT \| CHARACTER \|<br>GRAPHIC \|<br>UCHAR \| WIDECHAR<br>[(*length*)]<br>NONVARYING<br>VALUE | 内部 |
| 名前の付いたロケーター | POINTER \| OFFSET \| HANDLE<br>VALUE<br>LOCATES | 内部 |
| 名前の付いたピクチャー | PICTURE \| WIDEPIC<br>REAL \| COMPLEX<br>VALUE | 内部 |

表 7. 定数タイプによる属性の分類 (続き)

| 定数タイプ | データ属性[注 1 および 2] | スコープ属性[注 1 および 2] |
|---|---|---|
| ファイル[注 3] | FILE ENVIRONMENT<br>STREAM \| RECORD<br>INPUT \| OUTPUT \| UPDATE<br>SEQUENTIAL \| DIRECT<br>BUFFERED \| UNBUFFERED[Note 4]<br>KEYED<br>PRINT | INTERNAL \| EXTERNAL |
| 入り口[注 5] | ENTRY [RETURNS] | INTERNAL \| EXTERNAL |
| フォーマット[注 5] | FORMAT | 内部 |
| ラベル[注 5] | LABEL | 内部 |

**注記 :**

1. この表の中で大文字で表されている属性は、明示的に宣言されます。 小文字の属性は、暗黙的にデータ・タイプに指定されます。
2. データ属性のデフォルトには下線が引かれています。 リテラルのデータ属性はコンテキストに応じて異なるため、デフォルトは適用されません。 名前のついた定数およびファイル定数には選択可能な属性があるので、デフォルトが示されます。
3. ファイル属性は、273 ページの『第 10 章 入出力』に記載されています。
4. BUFFERED は、SEQUENTIAL ファイルのデフォルトです。 UNBUFFERED は、DIRECT ファイルのデフォルトです。
5. フォーマット定数およびラベル定数、ならびに INTERNAL 項目定数は、 DECLARE ステートメントでは宣言されません。

表 8. 変数タイプによる属性の分類

| 変数の型 | データ属性 | 位置合わせ属性 | 有効範囲属性 | ストレージ属性 |
|---|---|---|---|---|
| 領域 | AREA(*size*) | ALIGNED \| UNALIGNED | INTERNAL \| EXTERNAL<br><br>(INTERNAL は以下については必須: AUTOMATIC BASED DEFINED PARAMETER) | AUTOMATIC \| STATIC \| BASED \| CONTROLLED<br><br>(AUTOMATIC は INTERNAL の場合のデフォルト。STATIC は EXTERNAL のについてデフォルト。)<br><br>定義済み変数:<br>DEFINED<br>[POSITION]<br><br>パラメーター:<br>PARAMETER<br>[CONNECTED \| NONCONNECTED]<br>[CONTROLLED]<br><br>[INITIAL<br>[CALL]]<br><br>[VARIABLE]<br><br>[NORMAL \|<br>ABNORMAL]<br><br>ASSIGNABLE \| NONASSIGNABLE |
| コード化算術<br>注 1 | REAL \| COMPLEX<br>FLOAT \| FIXED<br>BINARY \| DECIMAL<br>PRECISION<br>[SIGNED \|<br>UNSIGNED] | ALIGNED \| UNALIGNED | | |
| 記入 | ENTRY [RETURNS]<br>[LIMITED] | | | |
| ファイル | FILE | | | |
| Format | FORMAT | | | |
| ラベル | ラベル | | | |
| ロケーター | POINTER \| HANDLE \|<br>{OFFSET<br>[(area-variable)]<br>LOCATES} | | | |
| 序数 | ORDINAL | | | |
| ピクチャー | PICTURE \| WIDEPIC<br>REAL \| COMPLEX | ALIGNED \| UNALIGNED | | |
| ストリング | BIT \| CHARACTER \|<br>GRAPHIC \| UCHAR \|<br>WIDECHAR<br>[(length)]<br>[ VARYING \|<br>VARYING4 \|<br>VARYINGZ \|<br>NONVARYING] | | | |
| タスク | TASK | ALIGNED \| UNALIGNED | | |

**配列:** DIMENSION は、いずれかの宣言に追加できます。変数。 詳細については、170 ページの『配列』を参照してください。

表 8. 変数タイプによる属性の分類 (続き)

| 変数の型 | データ属性 | 位置合わせ属性 | 有効範囲属性 | ストレージ属性 |
|---|---|---|---|---|
| **構造体および共用体**<br>• 主要な構造体および共同体の場合: 有効範囲、ストレージ (INITIAL 以外)、位置合わせ、 STRUCTURE または UNION、および LIKE 属性を指定することができます。<br>• 構造体あるいは共同体であるメンバーの場合: 位置合わせ、STRUCTURE または UNION、および LIKE 属性を指定することができます。<br>• メンバーは、常に INTERNAL 有効範囲属性を持ちます。<br>詳し くは、174 ページの『構造体』 および 175 ページの『共用体』 を参照 してください。 | | | | |
| **注記 :**<br>1. 宣言しなかった名前や、データ・タイプを指定せずに宣言した名前は、コード化算術変数と見なされます。 デフォルト属性は、165 ページの『属性のデフォルト』 に記載されています。 示されるデフォルトは、IBM デフォルトです。 ANS デフォルトは、FLOAT と DECIMAL ではなく FIXED と BINARY です。<br>2. POSITION は、ストリング・オーバーレイ定義を使用した場合だけ使用することができます。 | | | | |

# 計算データ・タイプとその属性

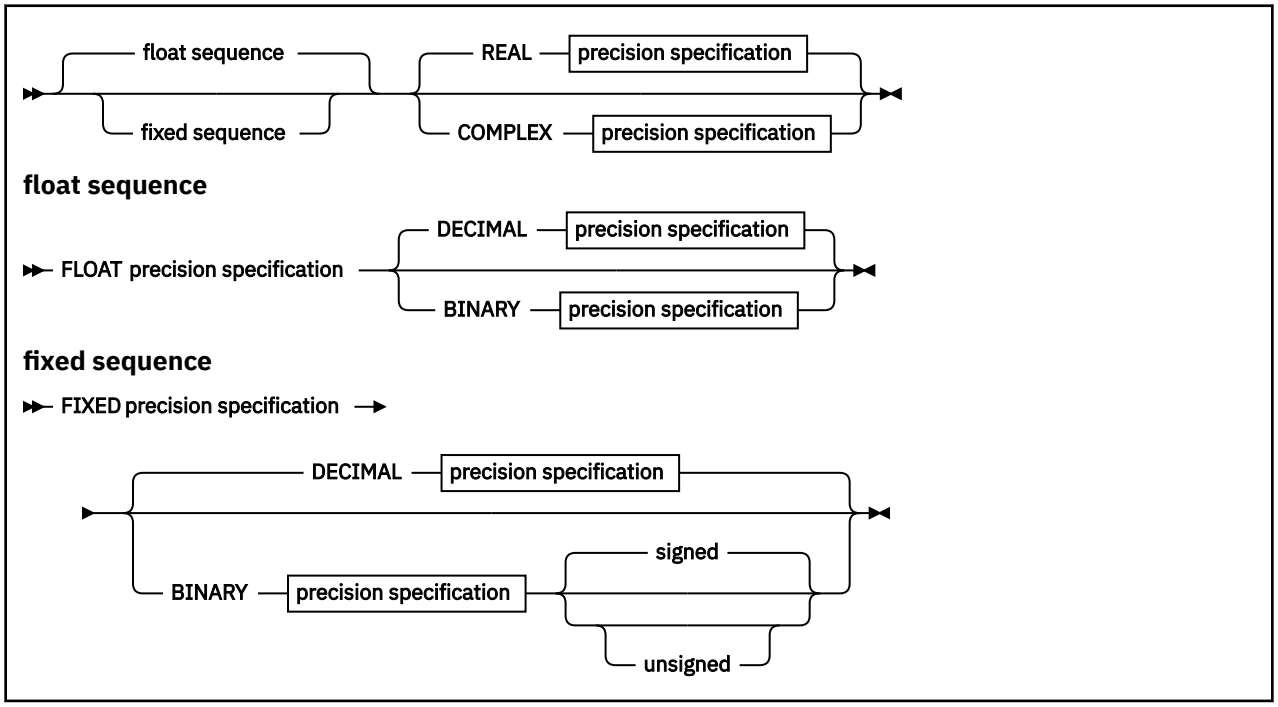
この節では、計算データとして分類されるデータ・タイプ、 およびそれに関連づけられる属性を説明します。

## コード化算術データとその属性

このトピックでは、コード化算術データの構文を示し、コード化算術データ属性とその省略形をリストします。

コード化算術データの一般情報については、16 ページの『データ・タイプとその属性』を参照してください。

### コード化算術データの構文



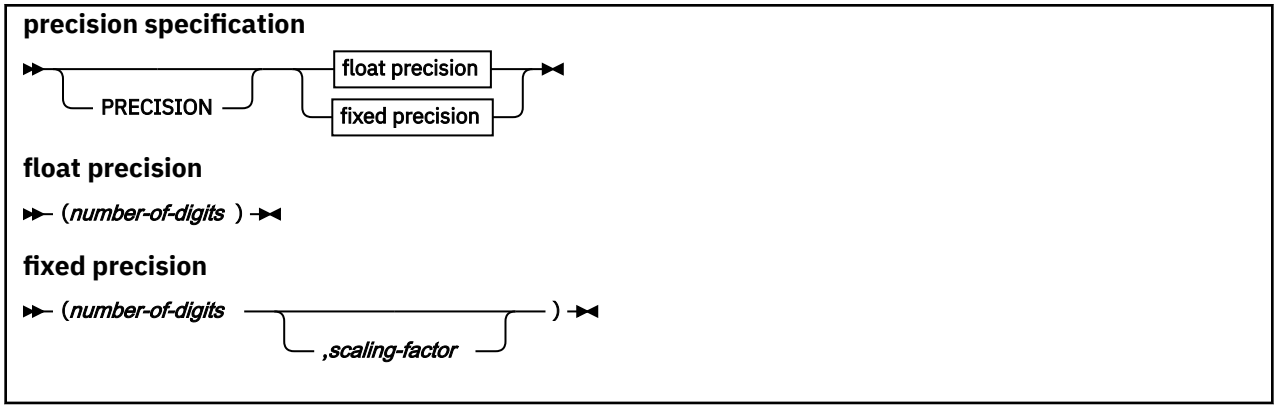



表 9. コード化算術データ属性の省略形

| 属性 | 省略形 |
|---|---|
| BINARY | BIN |
| COMPLEX | CPLX |
| DECIMAL | 12 月 |
| PRECISION | PREC |

## BINARY 属性と DECIMAL 属性

コード化算術データ項目の基数 は、10 進数または 2 進数のいずれかです。 DECIMAL はデフォルトです。

## FIXED 属性と FLOAT 属性

コード化算術データ項目のスケール は、 固定小数点または浮動小数点のいずれかです。

固定小数点データ項目は有理数であり、その 10 進小数点または 2 進小数点の位置は、 定数のときは小数点として示され、変数のときは宣言済みの位置取り係数で指定されます。

浮動小数点データ項目は、小数部分と指数部分のフォーマットの有理数です。

## PRECISION 属性

コード化算術データ項目の精度 には、桁数と位置取り係数が含まれます。 (位置取り係数は、固定小数点項目にのみ使用されます。)

**桁数**
何桁まで値を持つことができるかを指定する整数です。 固定小数点項目の場合、整数は有効数字の数になります。 浮動小数点の場合、整数は小数点を排除して保持される 有効数字の数になります (小数点の位置によって異なる)。

**位取り係数 (scaling factor)**
数値の最右端の桁に関して、10 進または 2 進小数点の 想定する位置を指定する整数です。符号は付けても付けなくてもかまいません。 位取り係数を指定しない場合、デフォルトは 0 になります。

精度属性の指定を表すとき、通常は (*p*,*q*) と書かれています。 ただし、*p* は桁数を表し、*q* は位取り係数を表します。

負の位取り係数 (-q) を指定すると、実際の最右端の桁から 右方向へ q 桁ずれた位置に小数点がある整数を表します。 桁数よりも大きい正の位取り係数 (q) を指定すると、実際の最右端の桁から 左方向に q 桁ずれた位置に小数点がある小数を表します。 いずれの場合も、間にゼロがあると想定されますが、それらは保管されません。実際に保管されるのは、指定された桁数だけです。

PRECISION を省略する場合は、精度属性を 同じ属性分配レベルのスケール (FIXED または FLOAT)、基数 (DECIMAL または BINARY)、またはモード (REAL または COMPLEX) の属性のすぐあとに書き、その間にほかの属性を指定してはなりません。

省略しない場合には、PRECISION は、宣言内の任意の場所に置くことができます。

整数値 は、位取り係数がゼロの固定小数点値を意味します。

## REAL 属性と COMPLEX 属性

算術データ項目 (コード化算術データ項目または数字データ項目) のモード が 実数か複素数のいずれかです。

実数データ項目は、実数の値を表す数です。

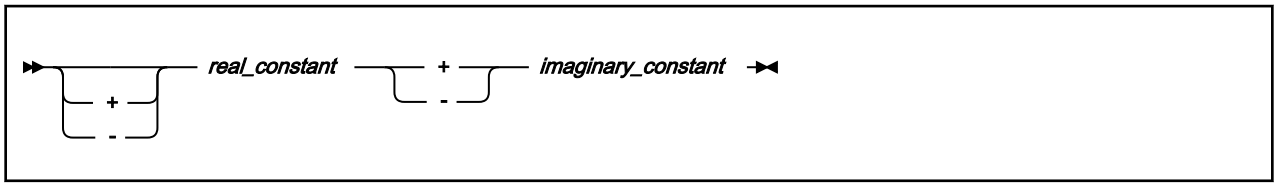
複素数データ項目は、2 つの部分 (実数部分と虚数部分) から構成されます。 複素数データ項目を表す変数の場合、実数部分と虚数部分の基数、スケール、および精度は同じです。

算術変数のデフォルトは REAL です。

虚数定数は、任意のタイプの実定数として書かれ、その直後に文字 I が続きます。 その例をいくつか紹介します。

```
27I
3.968E10I
11011.01BI
```

これらは、いずれも実数部分がゼロであると見なされます。 実数部分がゼロ以外の複合値は、次の構文を使った式で表します。




例えば、38+27I

下記の y および z という 2 つの複素数が与えられたとき、

```
y = complex(A,B);
z = complex(C,D);
```

x=y/z は以下のように計算されます。

```
real(x) = (A*C + B*D)/(C**2 + D**2);
imag(x) = (B*C - A*D)/(C**2 + D**2);
```

x=y*z は以下のように計算されます。

```
real(x) = A*C - B*D;
imag(x) = B*C + A*D;
```

これらの計算の途中で計算条件が起こることもあります。

## SIGNED 属性と UNSIGNED 属性

SIGNED 属性と UNSIGNED 属性は、FIXED BINARY 変数と ORDINAL 変数にのみ使用できます。

SIGNED は、変数が負の値と想定できることを示します。 UNSIGNED は、変数が非負の値とのみ想定できることを示します。

UNSIGNED は、固定小数点演算のセマンティクスに次のような影響を与えます。

- IAND、IEOR、INOT および IOR のすべてのオペランドが UNSIGNED だと、その結果は UNSIGNED です。
- ISLL と ISRL の第 1 オペランドが UNSIGNED だと、その結果は UNSIGNED です。
- REAL または IMAG のオペランドが UNSIGNED だと、その結果は UNSIGNED です。

RULES(ANS) コンパイラー・オプションを使用している場合、UNSIGNED は、固定小数点演算のセマンティクスに次のような影響を与えます。

- 加算、乗算または除算の両方のオペランドが UNSIGNED だと、その結果は UNSIGNED です。
- MAX または MIN のすべてのオペランドが UNSIGNED だと、その結果は UNSIGNED です。
- REM または MOD のすべてのオペランドが UNSIGNED だと、その結果は UNSIGNED です。

25 ページの表 10 および 25 ページの表 11 に示されているように、 SIGNED 属性と UNSIGNED 属性はストレージ所要量に影響を及ぼします。

表 *10. FIXED BINARY SIGNED* データ・ストレージ所要量

| 精度 | ストレージ占有量 (バイト) |
|---|---|
| 精度 <= 7 | 1 |
| 7 < 精度 <= 15 | 2 |
| 15 < 精度 <= 31 | 4 |
| 31 < 精度 <= 63 | 8 |

表 *11. FIXED BINARY UNSIGNED* データ・ストレージ所要量

| 精度 | ストレージ占有量 (バイト) |
|---|---|
| 精度 <= 8 | 1 |
| 8 < 精度 <= 16 | 2 |
| 16 < 精度 <= 32 | 4 |
| 32 < 精度 <= 64 | 8 |

## 2 進固定小数点データ

2 進固定小数点変数を宣言するデータ属性は、BINARY および FIXED です。

次の例では、Factor は、20 個のデータ・ビット (このうち 2 個は 2 進小数点の右側にあります) の 2 進固定小数点データを表すことができる変数として宣言されています。

```
declare Factor binary fixed (20,2);
```

符号付きおよび符号なしの固定 2 進データが占めるストレージについては、24 ページの『SIGNED 属性と UNSIGNED 属性』 を参照してください。

宣言した数のデータ・ビットは、低位桁に入りますが、高位桁の余分な桁位置も、 そのデータ項目を使って行われるすべての演算の対象となります。 算術オーバーフローがこの余分な高位の桁位置に及んだかどうかが検出されるのは、 SIZE 条件が割り込み可能になっている場合だけです。

## 2 進固定小数点定数

2 進固定小数点定数は、1 つ以上のビットで構成され (2 進小数点はオプション)、直後に文字 B が続きます。

2 進固定小数点定数は精度 (*p*,*q*) を持ちます。 *p* はその定数にあるデータ・ビットの合計数、*q* は 2 進小数点の右側にあるビット数です。

**例**

| 定数 | 精度 |
|---|---|
| 1011_0B | (5,0) |
| 1111_1B | (5,0) |

| 定数 | 精度 |
|---|---|
| 101B | (3,0) |
| 1011.111B | (7,3) |

## XN (16 進) 2 進固定小数点定数

XN 定数は、16 進表記の SIGNED REAL FIXED BINARY 定数を表します。

定数が 8 桁以下の場合、その精度は 31 です。9 桁以上の場合は精度は 63 です。

次の例について考えてみます。

```
'100'XN              /*  same as '00000100'XN with value 256      */
'8000'XN             /*  same as '00008000'XN with value 32,768   */
'FFFF'XN             /*  same as '0000FFFF'XN with value 65,535   */
"ffff_ffff"XN        /*  is the value -1                          */
```

必要に応じて、XN 定数の 16 進数値は、指定された値の左側に 16 進のゼロが埋め込まれます。

## XU (16 進) 2 進固定小数点定数

XU 定数は、16 進表記の UNSIGNED REAL FIXED BINARY 定数を表します。

定数が 8 桁以下の場合、その精度は 32 です。9 桁以上の場合は精度は 64 です。

次の例について考えてみます。

```
'100'XU              /*  same as '00000100'XU with value 256      */
'8000'XU             /*  same as '00008000'XU with value 32,768   */
'FFFF'XU             /*  same as '0000FFFF'XU with value 65,535   */
"ffff_ffff"XU        /*  is the value 2**32-1                     */
```

必要に応じて、XU 定数の 16 進数値は、指定された値の左側に 16 進のゼロが埋め込まれます。

## 10 進固定小数点データ

10 進固定小数点変数を宣言する場合のデータ属性は、DECIMAL および FIXED です。

例えば、次の DECLARE ステートメントは、A が 5 桁の 10 進固定小数点データを表すことを指定しています。 5 桁のうちの 4 桁は小数点の右側にあります。

```
declare A fixed decimal (5,4);
```

次の 2 つの例はいずれも、B が 7 桁の整数を表すことを指定しています。

```
declare B fixed (7,0) decimal;
declare B fixed decimal(7);
```

次の例は、C の位取り係数が -2 であることを指定しています。 これは、C が -9999999*100 から 9999999*100 の範囲内 (増分は 100) の 7 桁を保持することを意味します。

```
declare C fixed (7,-2) decimal;
```

次の例は、D が 3 桁 (うち 2 桁は小数部) の固定小数点データを表すことを指定しています。

```
declare D decimal fixed real(3,2);
```

10 進固定小数点データは、1 バイトに 2 桁ずつ保管され、最右端バイトの右 4 ビットに符号が保管されます。 したがって、10 進固定小数点データ項目は、変数の宣言で桁数 *p* を 偶数として指定しても、必ず奇数として保管されます。

偶数の桁数を宣言すると、最高位の桁数に余分な 1 桁分が生じます。 この桁位置は、そのデータ項目を使って行われるすべての演算 (例えば、比較演算) の 対象となります。 余分な最高位の桁位置がゼロ以外の場合は、算術演算または割り当てでデータを使用すると、例外が発生することがあります。 算術オーバーフローまたは割り当てが、この余分な最高位の桁位置にまで及んだかどうかが検出されるのは、SIZE 条件が使用可能になっている場合だけです。

## 10 進固定小数点定数

10 進固定小数点定数は、1 桁以上の 10 進数で構成されます。小数点はあってもなくても構いません。

10 進固定小数点定数の精度は (*p*,*q*) であり、*p* はその定数の全桁数で、*q* は、小数点の右側に指定された桁数です。

**例**

| 定数 | 精度 |
|---|---|
| 3.1416 | (5,4) |
| 455.3 | (4,1) |
| 732 | (3,0) |
| 1_200_300 | (7,0) |
| 003 | (3,0) |
| 5280 | (4,0) |
| .0012 | (4,4) |

## 2 進浮動小数点データ

2 進浮動小数点の変数を宣言するデータ属性は、BINARY および FLOAT です。

例えば、次の DECLARE ステートメントで、S は 2 進数 16 桁の精度を持つ 2 進浮動小数点データを表します。

```
declare S binary float (16);
```

指数は 5 桁の 10 進数を超えることはできません。 宣言した精度が (21) 以下であれば、短精度の浮動小数点フォーマットで保管されます。 宣言した精度が (21) より大きく、(53) 以下であれば、 長精度の浮動小数点フォーマットが使用されます。 宣言した精度が (53) より大きい場合には、拡張精度の浮動小数点フォーマットが使用されます。

## 2 進浮動小数点定数

2 進浮動小数点定数では、小数部の後に指数と文字 B が続きます。

小数部は 2 進固定小数点定数です。 指数は、英字の E、S、D、または Q のあとに 10 進整数 (符号はあってもなくてもよい) を書き、 これで 2 の累乗を表します。 E を使用する定数は、精度 (*p*) があります。*p* は小

数部の 2 進数の数字です。 S、D および Q を使用する定数には、必ず最大単精度、最大倍精度、および最大拡張精度があります。

表 *12. 2* 進浮動小数点定数の例

| 定数 | 精度 |
|---|---|
| 101101E5B | (6) |
| 101.101E2B | (6) |
| 11101E-28B | (5) |
| 11.01E+42B | (4) |
| 1S0b | (21) |
| 1D0b | (53) |
| 1Q0b | (64) (**Windows**) |
| 1Q0b | (106) (**AIX**) |
| 1Q0b | (109) (**z/OS**) |

## 10 進浮動小数点データ

10 進浮動小数点の変数を宣言するデータ属性は、DECIMAL および FLOAT です。

以下に例を示します。

```
declare Light_years decimal float(5);
```

Light_years の値は、 5 桁の 10 進数を持つ浮動小数点データを表しています。

IEEE 10 進浮動小数点データの場合、以下が適用されます。

- 宣言した精度が 7 以下であれば、短精度の浮動小数点フォーマットが使用されます。
- 宣言した精度が 7 より大きく、16 以下であれば、長精度の浮動小数点フォーマットが使用されます。
- 宣言した精度が 16 より大きい場合には、拡張精度の浮動小数点フォーマットが使用されます。

他のすべての 10 進浮動小数点データの場合、以下が適用されます。

- 宣言した精度が 6 以下であれば、短精度の浮動小数点フォーマットが使用されます。
- 宣言した精度が 6 より大きく、16 以下であれば、長精度の浮動小数点フォーマットが使用されます。
- 宣言した精度が 16 より大きい場合には、拡張精度の浮動小数点フォーマットが使用されます。

## 10 進浮動小数点定数

10 進浮動小数点定数は、小数部のあとに指数が続きます。

小数部は、10 進固定小数点定数になります。 指数は、E、S、D、または Q のあとに、4 桁以下の 10 進整数 (符号はあってもなくてもよい) を書き、 これで 10 の累乗を表します。 E を使用する定数には、精度 (*p*) があり、この *p* は、小数部の桁数になります。 S、D および Q を使用する定数は、常に単精度、倍精度、および拡張精度を表します。

表 *13. 10* 進浮動小数点定数の例

| 定数 | 精度 |
|---|---|
| 15E-23 | (2) |
| 15E23 | (2) |
| 4E-3 | (1) |
| 1.96E+07 | (3) |

表 *13. 10* 進浮動小数点定数の例 *(続き)*

| 定数 | 精度 |
|---|---|
| 438E0 | (3) |
| 3_141_593E-6 | (7) |
| .003_141_593E3 | (9) |
| 1s0 | (6) |
| 1d0 | (16) |
| 1q0 | (18) (**Windows**) |
| 1q0 | (32) (**AIX**) |
| 1q0 | (33) (**z/OS**) |

最後の 5 つの例は、(異なる精度を持っていますが) 同じ値を示しています。

IEEE 10 進浮動小数点 (DFP) の場合、10 進浮動小数点リテラルが、必要に応じてゼロ以外の数字が小数点の後に続かないように右側ユニット表示に変換された場合、つまり、指数が調整された場合に (例えば、3.1415E0 を 31415E-4 として表示する場合に行われるように)、リテラルによって与えられる精度の正常数の範囲にある指数を持っている必要があります。 これらの境界は、MINEXP-1 および MAXEXP-PLACES の値によって与えられます。 特に、次が保持される必要があります。

- 短精度の浮動小数点の場合、 -95 <= 指数 <= 90
- 長精度の浮動小数点の場合、 -383 <= 指数 <= 369
- 拡張精度の浮動小数点の場合、 -6143 <= 指数 <= 6111

したがって、IEEE 10 進浮動小数点 (DFP) の場合、最も大きい正の短精度の浮動小数点リテラルは 9999999E90 (または .9999999E97) で、最も小さい正のゼロでない短精度の 10 進浮動小数点は 1E-95 になります。

最も大きい正の浮動小数点値をリテラルとして指定したい場合は、代わりに HUGE 組み込み関数を使用するようにしてください。 同様に最も小さい非ゼロの正の値を指定する場合は、TINY 組み込み関数を使用するようにしてください。

# ストリング・データとその属性

このセクションでは、ストリング・データのタイプおよび属性について説明します。

ストリングに関する一般情報については、16 ページの『データ・タイプとその属性』を参照してください。

## BIT 属性、CHARACTER 属性、GRAPHIC 属性、UCHAR 属性、および WIDECHAR 属性

このトピックでは、BIT、CHARACTER、GRAPHIC、UCHAR、および WIDECHAR の各属性について説明し、属性の構文を示し、それらの省略形をリストします。

BIT 属性は、ビットの変数を指定します。

CHARACTER 属性は、文字変数を指定します。 ストリングは、PICTURE 属性を使用して宣言することができます。

GRAPHIC 属性は、GRAPHIC 変数を指定します。

UCHAR 属性は、UTF-8 データを保持する UCHAR 変数を指定します。

WIDECHAR 属性は、UTF-16 データを保持する WIDECHAR 変数を指定します。

## BIT 属性、CHARACTER 属性、GRAPHIC 属性、UCHAR 属性、および WIDECHAR 属性の構文

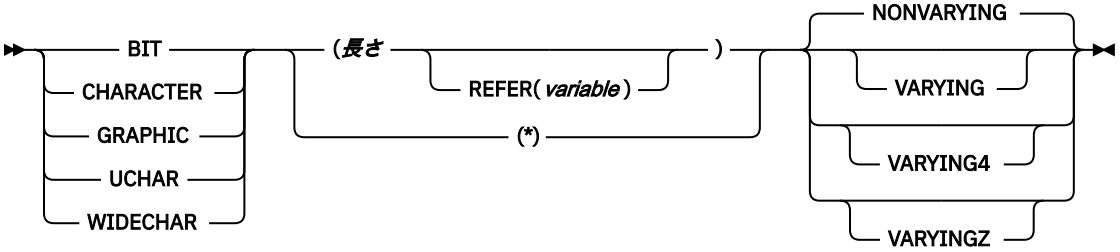



表 *14.* ストリング・データ属性の省略形

| 属性 | 省略形 |
|---|---|
| CHARACTER | CHAR |
| GRAPHIC | G |
| UCHAR | |
| WIDECHAR | WCHAR |
| NONVARYING | NONVAR |
| VARYING | VAR |
| VARYINGZ | VARZ ? |

**length**

NONVARYING ストリングの長さ、あるいは VARYING、VARYING4、または VARYINGZ ストリングの最大長を指定します。 長さは、必要に応じて、ビット、文字、uchar、ワイド文字、またはグラフィック (DBCS 文字) 単位になります。

長さは式またはアスタリスクとして指定できます。 長さが指定されていない場合、デフォルトは 1 です。 名前付き定数の場合、長さは値式の長さから決定されます。

パラメーターの場合は、パラメーターが CONTROLLED であるときだけ式は有効です。 パラメーターにアスタリスクで指定すると、渡される引数から長さが取られます。

長さ指定が式の場合は、変数にストレージが割り振られる時に、その式が評価され、a FIXED BINARY(31,0) 値に変換されます。 変換後の値は負であってはなりません。

STATIC データの場合には、式は、制限付きの式でなければなりません。

UCHAR データの場合、ストリングに割り振られるバイト数は、その長さの 4 倍です。

BASED データの長さの指定については、242 ページの『BASED 宣言でのエクステント指定』を参照してください。

**REFER**

REFER オプションに関しては、249 ページの『REFER オプション (自己定義データ)』を参照してください。

### 例

次のステートメントは、長さが 15 の文字データを表示できる変数として User を宣言します。

```
declare User character (15);
```

次の例は、ビット変数の宣言を示しています。

```
declare Symptoms bit (64);
```

## VARYING 属性、VARYING4 属性、VARYINGZ 属性、NONVARYING 属性

VARYING 属性、VARYING4 属性、VARYINGZ 属性は、変数の長さが 0 から宣言済み最大長までの範囲の可変長であることを指定します。 NONVARYING は、宣言した長さと等しい長さを常に持つ変数を指定します。

VARYING ストリングに割り振られるストレージには、ストリングの現在の長さを保持する 2 バイトが追加で含まれます。 現行値がストリング・タイプの単位であるため、BIT VARYING ストリングの現在の長さの値はストリング内の現在のビット数であり、UCHAR VARYING ストリングの値はストリング内の現在の UTF-8 文字の数です。

VARYING4 ストリングに割り振られるストレージには、ストリングの現在の長さを保持する 4 バイトが追加で含まれます。 現行値がストリング・タイプの単位であるため、BIT VARYING ストリングの現在の長さの値はストリング内の現在のビット数であり、UCHAR VARYING ストリングの値はストリング内の現在の UTF-8 文字の数です。

VARYINGZ CHARACTER ストリングに割り振られるストレージは、宣言した長さよりも 1 バイト長くなります。 ストリングの現在長は、それに割り振られたストレージの最初の '00'x の前にあるバイト数に等しくなります。

VARYINGZ GRAPHIC ストリングに割り振られるストレージは、宣言した長さの 2 倍よりも 2 バイト大きくなります。 ストリングの現在長は、それに割り振られたストレージの最初の '0000'gx の前にあるバイト数の半分に等しくなります。

VARYINGZ UCHAR ストリングに割り振られるストレージは、宣言した長さの 4 倍よりも 1 バイト大きくなります。 ストリングの現在長は、ストリングに割り振られたストレージの最初の '00'ux の前にある UTF-8 文字の数に等しくなります。

VARYINGZ WIDECHAR ストリングに割り振られるストレージは、宣言した長さの 2 倍よりも 2 バイト大きくなります。 ストリングの現在長は、それに割り振られたストレージの最初の '0000'wx の前にあるバイト数の半分に等しくなります。

VARYINGZ 属性は BIT ストリングには使用できません。

以下の場合、VARYING4 変数はサポートされていません。

- FILE の ENVIRONMENT 属性の PASSWORD サブオプション内
- レコード入出力ステートメントの KEYTO オプション内
- LOCATE ステートメントのサブジェクトとして
- レコード入出力ステートメントの FROM または INTO 文節のサブジェクトとして

下記の DECLARE ステートメントで、User と Zuser は、 最大長が 15 の可変長文字データを表しています。 ただし、User とは異なり、 Zuser は NULL 終止です。 割り振られたストレージは、User の場合は 17 バイトであり、 Zuser の場合は 16 バイトです。

```
declare User character (15) varying;
declare Zuser character (15) varyingz;
```

User と Zuser の長さは、 それぞれの時点でそれぞれに割り当てられるデータ項目の長さとなります。 MAXLENGTH 組み込み関数および LENGTH 組み込み関数を使用すると、 それぞれ宣言した長さと現在長を判別することができます。

VARYINGZ ストリング中の NULL 終止符は、ストリングの長さを判別する場合を除いては、 比較または割り当てには使用されません。 したがって、下記の宣言によるストリングの内部 16 進表示は同じですが、比較すれば等しくは**なりません**。

```
declare A char(4) nonvarying init( ('abc' || '00'x) );
declare B char(3) varyingz   init( 'abc' );
```

それとは反対に、次の例では、*Z* と *C* は等しいものとして比較されます。

```
dcl Z char(3) nonvarying init('abc');
dcl C char(3) varyingz init('abc');
```

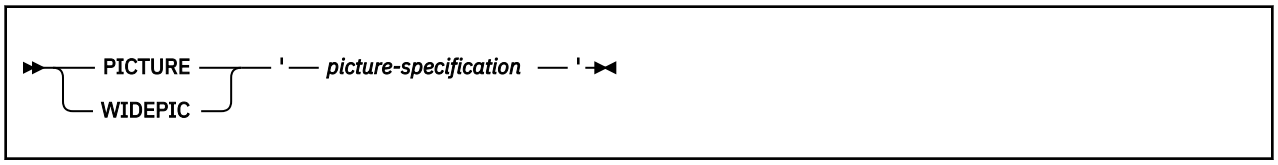
VARYING、VARYING4、VARYINGZ のストリングは、*の長さを持つパラメーターとして受け渡しできます。 それらが NONASSIGNABLE 属性を持っている場合、 記述子なしで渡すことができます。

## PICTURE 属性および WIDEPIC 属性

PICTURE 属性は、ピクチャー文字とデータ項目内の各文字位置に関連付けることにより、文字データ項目の特性を指定します。 WIDEPIC 属性は、ピクチャー文字とデータ項目内の各文字位置に関連付けることにより、WIDECHAR データ項目の特性を指定します。

ピクチャー文字は、その位置を占有できる文字のカテゴリーを指定します。

### 構文




**省略形** PIC

**picture-specification**
文字データ項目または数字データ項目のいずれかを記述します。 ピクチャー指定の後には、ピクチャー指定を続ける必要があります。 有効な文字については、325 ページの『文字データ用のピクチャー文字』 または 326 ページの『数字データ用のピクチャー文字』を参照してください。

PICTURE の *picture-specification* には、以下の規則が適用されます。

- 数字ピクチャー指定は、定数の表示によって指定する場合と同じように、数字データの算術属性を指定します。
- 数字データは、算術値を持ちますが、文字フォーマットで保管されます。 数字データは、算術演算が行われる前にコード化算術データに変換されます。
- 数字データ項目の基数は、10 進数です。 そのスケールは、固定小数点または浮動小数点のいずれかです (K または E というピクチャー文字は浮動小数点スケールを表します)。 数字データ項目の精度は、有効数字の桁数です (浮動小数点については指数を除きます)。 有効数字は、桁位置および条件式のピクチャー文字で指定されます。 数字データ項目の位取り係数は、V または F のピクチャー文字、もしくは V と F の組み合わせに基づいて決められます。
- 10 進データだけは、ピクチャー文字で表示することができます。 複素数データを宣言するには、固定小数点または浮動小数点データ項目を記述する 1 つのピクチャー指定と共に COMPLEX 属性を指定します。

WIDEPIC は、PICTURE を使用する場合と同じ方法で使用できますが、WIDEPIC には以下の規則が追加で適用されます。

- ピクチャー指定には算術ピクチャーを指定する必要があるため、指定にシンボル A または X が含まれていてはなりません。
- ピクチャー指定に、通貨記号またはオーバーパンチ・シンボルが含まれていてはなりません。

WIDEPIC'9V.99' は、値 3.14 を CHAR ではなく WIDECHAR として保持します。

**関連情報**
39 ページの『数字データ』
数字データ項目は、数字ピクチャー指定と PICTURE 属性を使用して宣言された変数の値です。 このデータ項目は、10 進固定小数点の値または 10 進浮動小数点の値を文字で表記したものです。

## 文字データ

CHARACTER 属性のついたデータには、文字セットによって サポートされる 256 個のどの文字も入れることができます。 PICTURE 属性のついたデータは、ピクチャー指定文字と一致する文字を持たなければなりません。

それぞれの文字は 1 バイトのストレージを占有します。

## 文字定数

文字定数は、連続した一連の文字を単一引用符または二重引用符で囲んだものです。

定数内に含まれる引用符は、16 ページの『引用符の使用』でリストされる規則に従います。 文字定数の長さは、開始引用符と閉じ引用符の間にある文字の個数です。 ただし、2 つの引用符は、1 文字として数えられます。

ヌル文字定数とは、間にブランクを入れずに引用符を 2 つ書いたものです。

### 構文

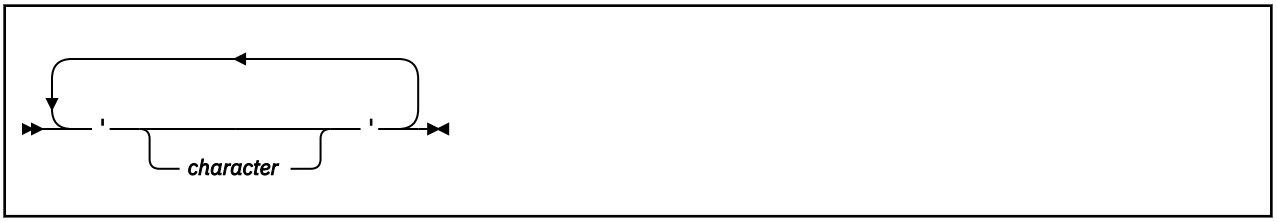



表 *15.* 文字定数の例

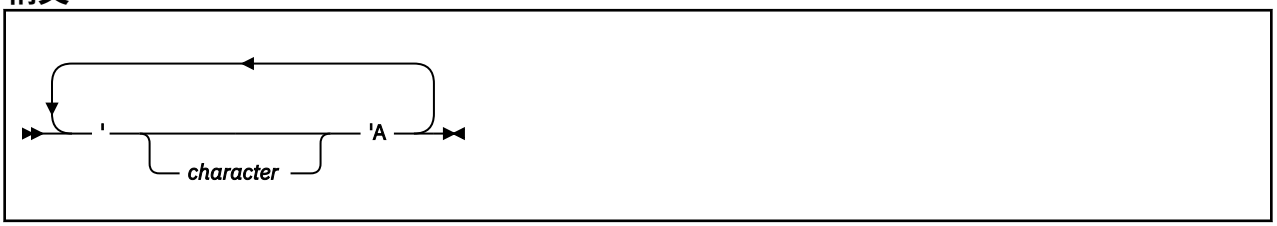
| 定数 | 長さ |
|---|---|
| `'Shakespeare''s "Hamlet"'` | 22 |
| `"Shakespeare's ""Hamlet"""` | 22 |
| `"Page 5"` | 6 |
| `'/* This is a comment */'` | 27 |
| `''` | 0 |
| `(2)'Walla '` | 12 |

最後の例の、括弧内の数はストリング反復因数 であり、 そのあとにある一連の文字を指定回数だけ反復することを表します。 この例は、定数 `"Walla Walla "` と同等です。 ストリング反復因数は、定数であり、括弧で囲まれなければなりません。

## A (ASCII) 文字定数

A (ASCII) 文字定数は、A で終わる文字定数です。 A 文字定数のデータは ASCII に変換されます。

### 構文




**文字**

A 文字定数内のすべての文字は、コード・ページによって変わることがない文字でなければならず、また、UTF-8 に変換された場合は 1 バイトのみを占める必要があります。

#### 例

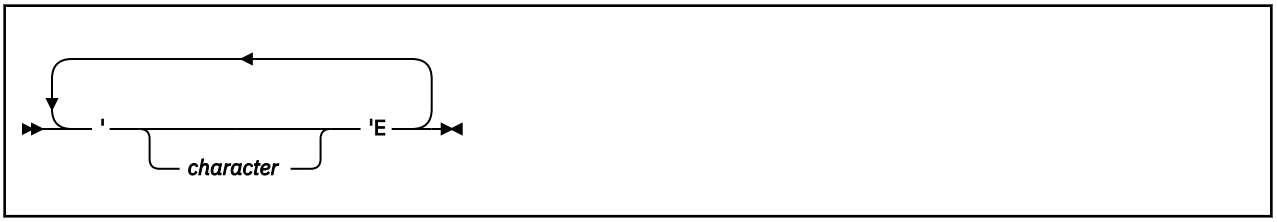
'123'A は、16 進値 '313233'X を表します。

この表現は、DEFAULT(ASCII | EBCDIC) オプションの設定に依存しません。

## E (EBCDIC) 文字定数

E (EBCDIC) 文字定数は、E で終わる文字定数です。 E 文字定数のデータは EBCDIC に変換されます。

### 構文




**文字**
E 文字定数内のすべての文字は、コード・ページによって変わることがない文字でなければならず、また、UTF-8 に変換された場合は 1 バイトのみを占める必要があります。

#### 例

'123'E は 16 進値 'F1F2F3'X を表します。

この表現は、DEFAULT(ASCII | EBCDIC) オプションの設定に依存しません。

## X (16 進数) 文字定数

X 文字定数は、単一引用符または二重引用符で囲まれた偶数個の 16 進数字の連続したシーケンスで、直後に文字 X が続きます。 16 進数字の各ペアは、1 文字を表します。

X 定数の長さは、指定された 16 進数の数の半分になります。

ヌルの X 定数は 2 つの引用符として書かれ、そのあとに接尾部 X が続きます。

### 構文

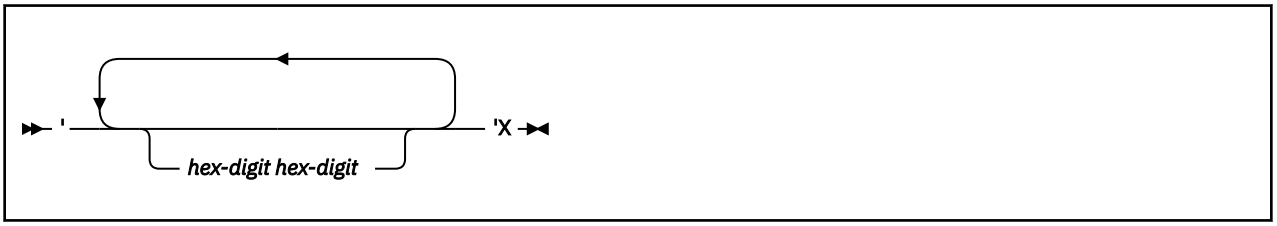



表 *16. X* 定数の例

| 定数 | 長さ |
|---|---|
| "0d0A"x | 2 |
| ''X | 0 |

**注 :** X 定数を使用すると、プログラムの移植性が制限されることがあります。

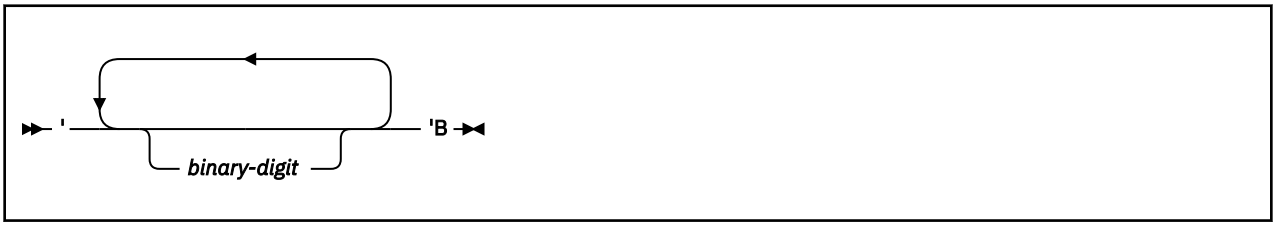
## ビット・データ

BIT 属性を持つデータでは、ビット単位でストレージの操作を行うことができます。 ストレージの 1 バイトは 8 ビットで構成されます。

## ビット定数

ビット定数は、単一または二重引用符で囲まれた一連の 2 進数の列であり、 そのすぐ後ろに B が続きます。

### 構文




ヌルのビット定数は、2 つの引用符で書かれ、そのあとに B が続きます。

表 *17.* ビット定数の例

| 定数 | 長さ |
|---|---|
| '1'B | 1 |
| "1100_1010_11"B | 10 |
| (64)'0'B | 64 |
| ''B | 0 |
| '0'B | 1 |

3 番目の例の括弧内の数字はストリング反復因数であり、 指定した回数だけ後続のビットを反復することを示します。 上記の例では、64 個のゼロのビットからなるストリングになります。

(ビット・データから文字データへの変換、 および文字データからビット・データへの変換に関する説明については、 77 ページの『ソースからターゲットへの変換規則』 を参照してください。)

## B4 (16 進) ビット定数

B4 ビット定数は、単一引用符または二重引用符で囲まれた 一連の 16 進数の列であり、そのすぐあとに B4 が続きます。 各 16 進数は、4 つのビットを表します。 BX は、B4 の同義語です。

### 構文

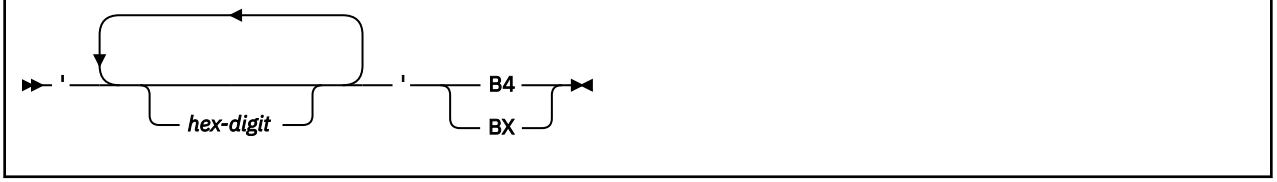



表 *18. B4* 定数の例

| | | |
|---|---|---|
| 'CA'B4 | は次と等しい | "1100_1010"B |
| '80'B4 | は次と等しい | '1000_0000'B |
| '1'B4 | は次と等しい | '0001'B |
| (2)'F'B4 | は次と等しい | '1111_1111'B |
| (2)'F'B4 | は次と等しい | 'FF'BX |
| ''B4 | は次と等しい | ""B |

## B3 (8 進) ビット定数

B3 ビット定数は、単一引用符または二重引用符で囲まれた一連の 8 進数の列であり、そのすぐ後に B3 が続きます。 各 8 進数は、3 つのビットを表します。

表 *19. B3* 定数の例

| | | |
|---|---|---|
| '22'B3 | は次と等しい | "010_010"B |
| '40'B3 | は次と等しい | '100_000'B |
| '1'B3 | は次と等しい | '001'B |
| (2)'7'B3 | は次と等しい | '111_111'B |
| ''B3 | は次と等しい | ""B |

## グラフィック・データ

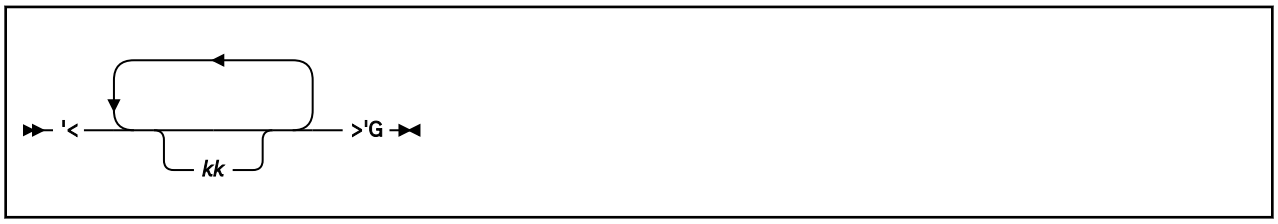
GRAPHIC データには、すべての DBCS 文字を入れることができます。 各 DBCS 文字は、2 バイトのストレージを占有します。

## グラフィック定数

グラフィック定数は、単一引用符または二重引用符で囲まれた一連の DBCS 文字 の列です。 グラフィック定数は、定数内の各 DBCS 文字ごとに最大 2 バイトのストレージを使用します。

G リテラルは、前後を DBCS 引用符で囲んで指定でき、またその場合は G 全体を DBCS で指定することもできます。

### 構文




GRAPHIC コンパイラー・オプションは、受け入れられるグラフィック定数に対して有効でなければなりません。 GRAPHIC ENVIRONMENT オプションがグラフィック定数を含む STREAM 入出力ファイルに 指定されていない場合には、CONVERSION 条件が生じます。

## GX (16 進) グラフィック定数

GX グラフィック定数は、単一引用符または二重引用符で囲まれた一連の 16 進数の 列 (4 の倍数) であり、そのすぐあとに GX が続きます。 4 桁の 16 進の各グループは、1 つの DBCS 文字を表します。

### 構文

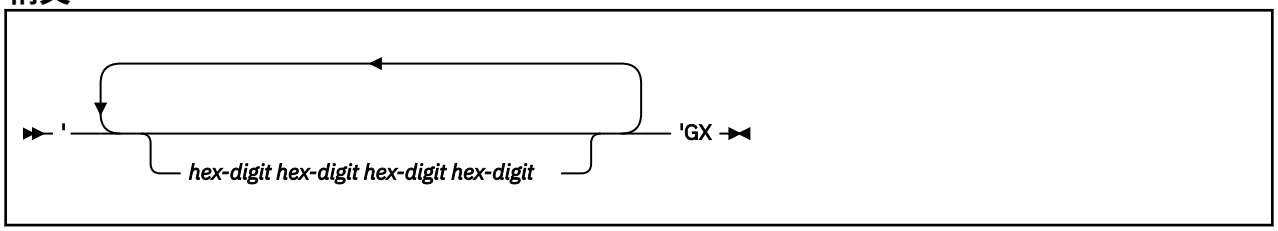



表 *20. GX (16* 進*)* グラフィック定数の例

| | |
|---|---|
| '81a1'GX | 1 個の DBCS 文字を表します。 |
| ""gX | ''g と同じです。 |

**注 :** GX を使用すると、プログラムの移植性が制限されることがあります。

## 混合文字データ

混合文字データには、SBCS および DBCS 文字を入れることができます。 混合データは、CHARACTER データ・タイプによって表され、CHARACTER データの処理規則にしたがいます。

OPTIONS 属性の CHARGRAPHIC オプション、および MPSTR 組み込み関数は、混合データ処理を援助するために使用することができます。 CHARGRAPHIC についての詳細は、123 ページの『OPTIONS オプションとその属性』を参照してください。 MPSTR については、488 ページの『MPSTR』を参照してください。

## M (混合) 文字定数

M 定数は、直後に文字 M が続く、引用符で囲まれた DBCS 文字、SBCS 文字、またはその両方の連続したシーケンスです。

定数内に含まれる引用符は、16 ページの『引用符の使用』でリストされる規則に従います。 定数 M の長さは、引用符の間にある SBCS 文字の数です。 ただし、2 つの単一引用符は 1 文字として計算され、 ストリング内の DBCS 文字の数を 2 倍にして計算します。

ヌルの M 定数は、2 つの引用符で書かれ、そのあとに M が書かれます。

### 構文

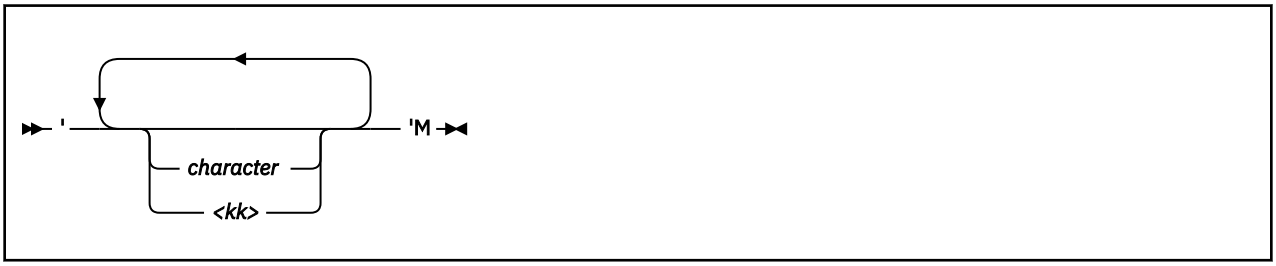



表 *21.* 混合文字定数の例

| 定数 | 長さ |
|---|---|
| 'IBM<kk>'M | z/OS では 7 バイト、その他のプラットフォームでは 5 バイト |
| '<.I.B.M>'M | z/OS では 8 バイト、その他のプラットフォームでは 6 バイト |
| ''M | 0 |

GRAPHIC コンパイラー・オプションは、受け入れられる混合定数に有効でなければなりません。 GRAPHIC ENVIRONMENT オプションは、混合定数を持つ STREAM 入出力ファイルに 指定されていない場合は、CONVERSION 条件が生じます。

z/OS では、以下の追加の規則が混合定数に適用されます。

- シフトアウト/シフトインのペアは合致しなければなりません。ペアをネストすることはできません。
- DBCS 部分には、どちらのバイトにも '0E'x または '0F'x が含まれていてはなりません。
- 文字部分には、シフト・コードとして特に意図されていない限り、値 '0E'x または '0F'x が含まれていてはなりません。

**注 :** シフト・コードは z/OS でのみ使用されるため、混合データや M 定数を使用するとプログラムの移植性が制限されることもあります。

## UCHAR データ

UCHAR データには、任意の UTF-8 ストリングを含めることができます。 各 UCHAR は、1 から 4 バイトのストレージを占有します。

UCHAR には以下の制約事項が適用されます。

- DEFINED では UCHAR はサポートされていない。
- UCHAR を含む割り当てでは、ソースとターゲットはオーバーラップすることができません。

- *x* が UCHAR の場合、TRANSLATE(*x, y*) は使用できません。 この場合、3 番目の引数を指定する必要があります。
- UCHAR は以下の組み込み関数ではサポートされません。
  - 414 ページの『CENTERLEFT』 と 415 ページの『CENTERRIGHT』
  - 418 ページの『COLLAPSE』 と 539 ページの『SQUEEZE』
  - 466 ページの『LEFT』 と 528 ページの『RIGHT』
  - 527 ページの『REPLACEBY2』
  - 523 ページの『REGEX』
  - 531 ページの『SCRUBOUT』

## UX (16 進) UCHAR 定数

UX UCHAR 定数は、単一引用符または二重引用符で囲まれた一連の 16 進数の列であり、そのすぐあとに UX が続きます。 ストリングには偶数個の 16 進数が含まれている必要があり、これらの 16 進数が有効な UTF-8 ストリングを表現している必要があります。

### 構文

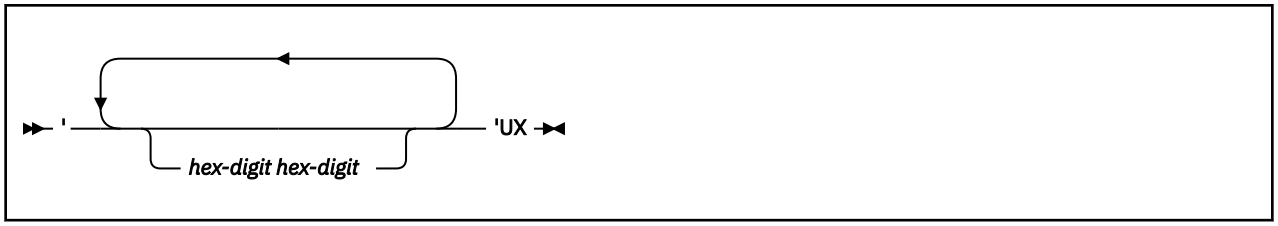



表 *22. UX (16* 進*) UCHAR* 定数の例

| | |
|---|---|
| '00'UX | 1 個の UCHAR を表します。 |
| ""uX | ''u と同じです。 |

**注記 :**

- UCHAR の最大値は 'F48FBFBF'ux です。
- UCHAR の最小値は '00'ux です。

## ワイド文字データ

WIDECHAR には、2 バイトに収まる任意の UTF-16 文字を入れることができます。

現在、以下のためのサポートはまだありません。

- ソース・ファイル内の WIDECHAR 文字
- W ストリング定数
- ストリーム入出力内の WIDECHAR 式の使用
- レコード入出力での WIDECHAR からの暗黙の型変換または WIDECHAR への暗黙の型変換
- レコード入出力での暗黙の endianness フラグ

  WIDECHAR ファイルを作成する場合は、ファイルの最初の 2 バイトとして endianness フラグ ('fe_ff'wx) を書き込んでください。

## WX (16 進) ワイド文字定数

WX ワイド文字定数は、単一引用符または二重引用符で囲まれた一連の 16 進数の 列 (4 の倍数) であり、そのすぐあとに WX が続きます。 4 桁の 16 進の各グループは、1 つの UTF-16 文字を表します。

### 構文

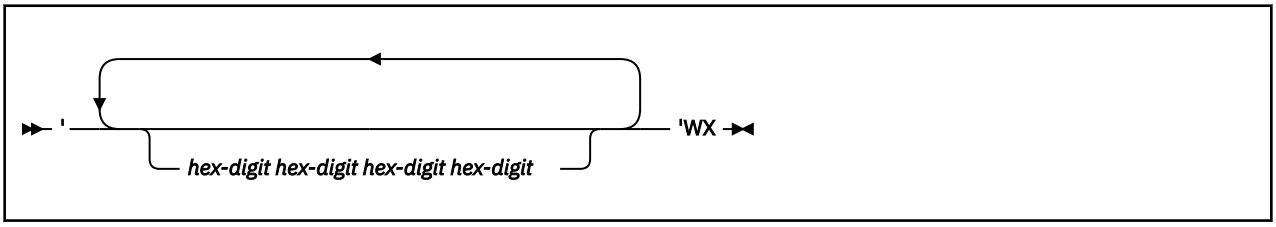



表 *23. WX (16* 進*)* ワイド文字定数の例

| | |
|---|---|
| '0031'wx | 1 個の UTF-16 文字を表します。 |
| ""wX | ''w と同じです。 |

**ノード:**

- WX 定数は、ビッグ・エンディアン・フォーマットで指定する必要が あります (プログラムをリトル・エンディアンフォーマットで実行する場合でも同様です)。 このため、例えば文字 '1' のワイド文字値は、常に '0031'wx ('3100'wx でなく) として指定する必要があります。
- WX を使用すると、プログラムの移植性が制限されることがあります。

## 数字データ

数字データ項目は、数字ピクチャー指定と PICTURE 属性を使用して宣言された変数の値です。 このデータ項目は、10 進固定小数点の値または 10 進浮動小数点の値を文字で表記したものです。

数字ピクチャー指定は、算術値に変換できるデータだけに割り当てられる文字ストリングを表します。

次の例を考えてください。

```
declare Price picture '999V99';
```

この宣言は、Price に割り当てられる値は、想定小数点が右端の 2 桁の前にある、5 桁の 10 進数の文字ストリングとして保持されることを指定します。 Price に割り当てられるデータは、この仮想小数点を基準にして、10 進固定小数点データの場合の小数点位置合わせと同じ方法で位置合わせされます。

数字データは算術属性を持ちますが、コード化算術フォーマットでは保管されません。 文字ストリングとして保管されます。 したがって、このデータは、10 進固定小数点フォーマットまたは 10 進浮動小数点フォーマットに変換されない限り、算術演算に使用できません。 このような変換は自動的に行われますが、余分な処理時間が必要になります。

数字データはストリングのように文字フォーマットで表され、コード化算術データのように小数点位置合わせが行われますが、その処理方法は、コード化算術項目またはストリングのいずれとも異なります。編集文字を数字データ項目内に挿入するように指定することができ、実際にデータ項目内に保存することができます。 したがって、その項目を文字ストリングとして印刷または処理する場合には、 代入演算に編集文字も含まれることになります。 ただし、数字データ項目を別の数字または算術変数に割り当てる場合は、編集文字は代入演算には含まれません。 割り当てられるのは、実際の数字、符号、および想定小数点の位置だけです。 次の例を考えてください。

```
declare Price picture '$99V.99',
        Cost character (6),
        Amount fixed decimal (6,2);
Price = 12.28;
Cost = '$12.28';
```

PRICE のピクチャー指定では、通貨記号 ($) と小数点 (.) が編集文字です。 これらの文字は、データ項目内の文字として保管されます。 これは、算術値の一部ではありません。 両方の代入ステートメントが実行されると、Price と Cost の実際の内部文字表記は 同じであると見なされることがあります。 印刷される場

合は、全く同じに印刷されます。 ただし、常に同じ方法で機能するわけではありません。 次の例を考えてください。

```
Amount = Price;
Cost   = Price;
Amount = Cost;
Price  = Cost;
```

最初の 2 つの代入ステートメントが実行されると、Amount の値は 0012.28 となり、Cost の値は '$12.28' となります。 Amount への Price の割り当てにおいて、通貨記号と小数点は編集文字であり、割り当ての一部ではありません。 Price の数値は、内部コード化算術フォーマットに変換されます。 ただし、Price を Cost に割り当てるときは、 文字ストリングへの割り当てなので、数字ピクチャー指定の編集文字も常に 割り当ての対象になります。 Price は文字フォーマットで保管されているので、変換は必要ありません。

3 番目と 4 番目の代入ステートメントを実行しようとすると、CONVERSION 条件が起こります。 ストリング内に通貨記号があれば算術定数としては有効でなくなるため、 Cost の値を Amount に割り当てることはできません。 これと同じ理由で、Cost の値を Price に 割り当てることもできません。 数字ピクチャー指定付きで宣言した変数に割り当てることができるのは、算術タイプの値か、算術タイプに変換できる値だけです。

小数点は編集文字でもあり、文字ストリング内の実際の文字にもなりますが、 算術フォーマットへの変換で CONVERSION 条件を引き起こしません。なぜなら、 算術定数内に小数点があってもよいからです。 正符号 (+) と負符号 (-) についても同様で、算術フォーマットに変換すると 算術定数の先頭に符号が付くからです。

数字ピクチャー指定では、その他の編集文字 (ゼロ抑制文字、浮動文字、挿入文字など) を使用することができます。

**関連情報**

325 ページの『ピクチャー指定文字』
ピクチャー指定とは、一連のピクチャー文字を一重または二重引用符で囲んだものです。 この文字は、文字データ項目または数字データ項目の各位置の内容、および出力の内容を記述します。

# 日付属性

2 つのオペランドが DATE 属性を持っている場合、 コンパイラーは、暗黙の日付比較と変換を行います。 DATE 属性は、変数 (または引数または戻り値) が 日付を指定したパターンで保持することを指定します。

## 構文

**パターン**
サポートされている日付のパターンの 1 つです。 パターンを指定しない場合、デフォルトの値 YYMMDD になります。

DATE 属性は、以下の属性の集合の 1 つを持つ変数がある場合のみ有効です。

- CHAR(n) NONVARYING
- PIC'(n)9' REAL
- FIXED DEC(n,0) REAL

*n* の長さまたは精度は、**pattern** の長さに一致しなければなりません。

RESPECT コンパイル時オプションを指定すると (詳細は「プログラミング・ガイド」を参照)、次の事柄が起こります。

- DATE 属性を重視することをコンパイラーは認識する。

- DATE 組み込み関数は、属性 DATE('YYMMDD') を持つ値を戻す。

これで、エラー・メッセージが生成されなくても、 属性 DATE('YYMMDD') を持つ変数に DATE() を割り当てることができます。 DATE() が DATE 属性を持たない変数に割り当てられる場合には、 必ずエラー・メッセージが生成されます。

## 暗黙の DATE 比較

DATE 属性を指定して宣言された 2 つの変数を比較する場合、 DATE 属性は暗黙の共通化 を引き起こします。 1 つの変数のみが DATE 属性を持っている場合の比較には、フラグが付けられ、 もう一方の被比較数は、通常同じ DATE 属性を持っている場合と同様に 処理されます (適用されるいくつかの例外については、後述します)。

暗黙の共通化とは、共通の比較可能な表現に日付を変換するために コンパイラーがコードを生成することを意味します。 このプロセスでは、WINDOW コンパイル時オプションで指定する *window* を使用して、2 桁の年を変換します。

次のコード・フラグメントでは、 DATE 属性を有効と見なした場合、 2 番目の DISPLAY ステートメントにある比較は時間帯が限定されます。 これは、window を 1900 で開始した場合、 この比較が偽を戻すということです。 しかし、window を 1950 で開始する場合、 この比較は真を戻します。

```
dcl a   pic'(6)9' date;
dcl b   pic'(6)9' def(a);
dcl c   pic'(6)9' date;
dcl d   pic'(6)9' def(c);

b = '670101';
d = '010101';

display( b || ' < ' || d || ' ?' );
display( a < c );
```

日付比較は、以下の位置で発生することがあります。

- IF ステートメントと SELECT ステートメント
- WHILE 文節または UNTIL 文節
- TO 文節が原因で起こる暗黙の比較

## 同様のパターンを持つ日付の比較

ある条件下では、コンパイラーは同一のパターンを持つ日付を比較する特別なコードを生成しません。

そのような条件は以下のとおりです。

- 比較演算子 =、¬=、または <> が使われている。
- パターンが、YYYY、YYYYMM、YYYYDDD、または YYYYMMDD と等価である。

## 異なるパターンを持つ日付の比較

異なるパターンの日付を含む比較の場合、 共通の比較可能な表現に日付を変換するために、 コンパイラーはコードを生成します。

変換が行われると、コンパイラーは 2 つの値を比較します。

## DATE 属性とリテラルを含む比較

1 つの被比較数が DATE 属性を持っており、 他方の被比較数がリテラルの場合に比較を行うと、 コンパイラーは W レベルのメッセージを発行します。

次のコンパイラーの動作は、 以下に示すようにリテラルの値によって異なります。

- リテラルが有効な日付で表される場合、DATE 属性を持つ被比較数と同じ 日付パターンと window を持っている場合と同様に処理されます。

- リテラルが有効な日付で表されない場合、 他方の被比較数の DATE 属性は無視されます。

```
dcl start_date char(6) date;
if start_date >= '' then /* no windowing */
…
if start_date >= '851003' then /* windowed */
…
```

## DATE 属性と非リテラルを含む比較

1 つの被比較数が DATE 属性を持っており、 他方の被比較数が日付でもリテラルでもない場合の比較では、コンパイラーは E レベルのメッセージを発行します。 非日付値は、他方の被比較数と同じ日付パターンと window を 持っている場合と同様に処理されます。

```
dcl start_date char(6) date;
dcl non_date char (6);

if start_date >= non_date then /* windowed */
…
```

## 暗黙の DATE 指定

日付パターンを指定して宣言された 2 つの変数を割り当てる際に、 DATE 属性は暗黙の変換の原因になることもあります。

- ソースとターゲットが、同じ DATE 属性とデータ属性を持っている場合、どちらも DATE 属性を持っていない場合と同様に、割り当てが進行します。
- ソースとターゲットが異なる DATE 属性を持っている場合、 コンパイラーはソースの日付を変換するためにコードを生成してから、 割り当てを行います。
- ソースは DATE 属性を持っており、ターゲットが DATE 属性を 持っていない場合の割り当てでは、コンパイラーは E レベルのメッセージを発行し、 DATE 属性を無視します。
- ターゲットは DATE 属性を持っており、ソースが DATE 属性を 持っていない (かつ、ソースがリテラルではない) 場合の割り当てでは、コンパイラーは E レベルのメッセージを発行し、DATE 属性を無視します。
- ターゲットは DATE 属性を持っており、ソースが DATE 属性を 持っていない (かつ、ソースがリテラルである) 場合の割り当てでは、 コンパイラーは W レベルのメッセージを発行し、DATE 属性を無視します。

  ```
  dcl start_date char(6) date;
  start_date = '';
  …
  ```

- ソースでは 4 桁の年を保持していて、 ターゲットでは 2 桁の年を保持している場合、 ソースはターゲットの window には存在しない年を保持することがあります。 この場合、ERROR 条件が発生します。

  ```
  dcl x char(6) date;
  dcl y char(8) date('YYYYMMDD');

  y = '20600101';

  x = y; /* raises error if window is <= 1960 */
  ```

- DATE 属性は以下の中では、無視されます。
  - デバッガー
  - レコード入出力ステートメント中で実行される割り当て
  - ストリーム入出力ステートメント (GET DATA など) 中で 実行される割り当てと変換

ウィンドウ操作ソリューションを選択しなくても、 2 桁または 4 桁の年の両方を操作しなければならないコードが いくつかあることがあります。 次の状況では、複数の日付パターンを使用できます。

```
dcl old_date char(6) date('YYMMDD');
dcl new_date char(8) date('YYYYMMDD');

new_date = old_date;
```

### 日付の診断

PL/I では、引数を記述した項目に式が引数として渡される場合、または式が RETURN ステートメントで使用される場合に、有効な 代入が行われます。

日付変数を以下のように使用すると、フラグが付けられます。

- 以下を含む (明示的または有効な) 代入
  - 日付から非日付
  - 非日付から日付
- 日付に適用される何らかの算術演算
- BY 文節中の日付の使用 (これは算術演算を暗黙指定する)
- 何らかの算術組み込み関数中での日付の使用
- BINARY、DECIMAL、FIXED、FLOAT、または PRECISION を除く、 任意の算術組み込み関数中での日付の使用
- 組み込み関数 SUM、PROD、または POLY 中での日付の使用

上記のすべての場合にコードが生成されますが、 ウィンドウ操作は起こりません。 結果では、DATE 属性は無視されます。

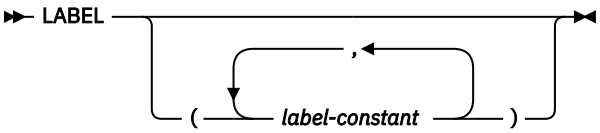
## プログラム制御のデータ・タイプとその属性

この節では、プログラム制御データとそれに関連する属性を説明しています。 プログラム制御データは、プログラムの実行を制御する値を指定するのに使います。

### ラベル・データと LABEL 属性

ラベル とは、ラベル定数またはラベル変数の値です。

**構文**




ラベル定数のリストが与えられる場合、変数はリストに含まれる定数を常にその値として持つ必要があります。そして、そのリスト内のラベル定数は、ラベル宣言を含むブロック内で既知である必要があります。括弧で囲まれたラベル定数のリストは、ラベル配列に関する LABEL 属性仕様で使用できます。

ラベル定数は、ステートメント (PROCEDURE、ENTRY、PACKAGE、または FORMAT を除く) のラベル接頭部として書き込まれる名前です。これにより、実行中に、そのステートメントへの参照を介してプログラム制御をそのステートメントに転送することができます。(ラベル接頭部の構文については、8 ページの『ステートメント』 を参照してください。)

例えば、次のコード行では Abcde がラベル定数です。

```
Abcde: Miles = Speed*Hours;
```

ラベルのついたステートメントは、通常の命令順序に従ってこの ステートメントを実行させることもできますし、プログラム内の別の場所から GO TO ステートメントによって このステートメントに制御権を移動して実行させることもできます。

ラベル変数には、別のラベル変数の値またはラベル定数を 割り当てることができます。 そのような割り当てが行われると、ソース・ラベルの環境が ターゲット・ラベル変数に割り当てられます。 ラベルの静的配列が初期値を持つことを宣言した場合、その配列は割り当て不可と見なされます。

GO TO ステートメントで使用されるラベル変数は、その GO TO ステートメントの実行時にアクティブであるブロック内で使われているラベル定数でなければなりません。 次の例を考えてください。

```
declare Lbl_x label;
Lbl_a:   statement;
          .
          .
          .
Lbl_b:   statement;
          .
          .
          .
       Lbl_x = Lbl_a;
          .
          .
          .
       go to Lbl_x;
```

Lbl_a と Lbl_b はラベル定数であり、 Lbl_x はラベル変数です。 Lbl_a を Lbl_x に割り当てることにより、 GO TO Lbl_x ステートメントは 制御を Lbl_a ステートメントに移します。 それ以外の場合には、 Lbl_b を Lbl_x に 割り当てるステートメントをプログラムに入れることができます。 この割り当てにより、Lbl_x への参照は、 Lbl_b への参照と同じになります。 この Lbl_x の値は、別の値が割り当てられるまで保持されます。

ラベル変数に無効な値が割り当てられても、エラーが検出されるとは限りません。 以下の例では、値 I が基数となっている 配列 Z の特定エレメントに伝送が行われます。

```
  go to Z(I);
    .
    .
    .
  Z(1):  if X = Y then return;
    .
    .
    .
  Z(2):  A = A + B + C * D;
    .
    .
    .
  Z(3):  A = A + 10;
```

Z(2) が省略されると、 I=2 の場合には GO TO Z(I) で ERROR 条件が生じます。 SUBSCRIPTRANGE 条件が使用不可の場合、I < LBOUND(Z) または I > HBOUND(Z) の場合には GO TO Z(I) が予測不能な結果を引き起こします。

# フォーマット・データと FORMAT 属性

フォーマット・データ項目とは、フォーマット定数またはフォーマット変数のことです。 フォーマット定数は、FORMAT ステートメントのラベル接頭部として書かれる名前です。 FORMAT 属性は、宣言される名前がフォーマット変数であることを指定します。

FORMAT 属性で宣言された名前は、別のフォーマット変数を持つか、 またはそれに割り当てられるフォーマット定数を持ちます。 そのような割り当てが行われると、ソース・ラベルの環境が ターゲット・ラベル変数に割り当てられます。

他の PL/I コンパイラーとの互換性を維持するために、フォーマット変数をラベル変数として宣言することもできます。

次の例を考えてください。

```
  Prntexe: format
              ( column(20),A(15), column(40),A(15), column(60),A(15) );
  Prntstf: format
              ( column(20),A(10), column(35),A(10), column(50),A(10) );
```

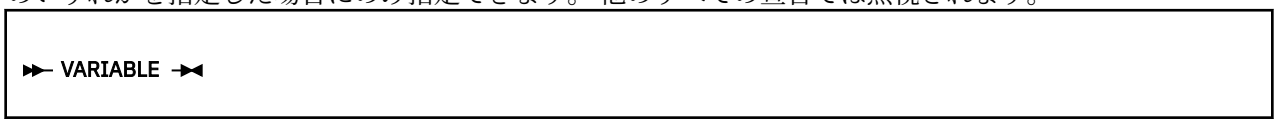
Prntexe と Prntstf はフォーマット定数です。

2 番目の例は、4 および 5 が次と同じ効果を持つことを示します 2 と同じ効果を持つことを示し、6 および 7 が次と同じ効果を持つことを示します 3.

```
1  dcl Print format;
2  put edit (X,Y,Z) (R(Prntexe) );
3  put edit (X,Y,Z) (R(Prntstf) );
4  Print = Prntexe;
5  put edit (X,Y,Z) (R(Print) );
6  Print = Prntstf;
7  put edit (X,Y,Z) (R(Print) );
```

## VARIABLE 属性

VARIABLE 属性は、名前を変数として設定します。また、この属性は、属性 ENTRY、FILE、または LABEL のいずれかを指定した場合にのみ指定できます。 他のすべての宣言では無視されます。




名前が構造体または共用体のメンバーである場合、 または以下のいずれかの属性が指定されている場合、VARIABLE 属性は暗黙に指定されています。

ストレージ・クラス属性
DIMENSION
PARAMETER
位置合わせ属性
INITIAL

以下の例では、Account1 と Account2 は、 ファイル変数であり、File1 と File2 はファイル定数です。

```
declare Account1 file variable,
        Account2 file automatic,
        File1 file,
        File2 file;
```

File1 と File2 は、 Account1 または Account2 に順番に割り振られます。

# 値の属性

このセクションでは、値の属性と名前付き定数について説明します。

## VALUE 属性

このトピックでは、VALUE 属性について説明し、その構文を示します。

**制限付き式 (restricted expression)**
式は、計算結果がスカラー値にならなければなりません。

**関連情報**
71 ページの『制限付き式』
PL/I に (おそらく、符号付きの) 定数が必要なときには、制限付き式 を使用することができます。

## 名前付き定数

名前付き定数は、スカラーまたは構造体用に宣言できます。

スカラー ID として、名前付き定数は、他のデータ属性と共に VALUE 属性を使用して宣言されます。 名前に対するすべての参照は、該当する定数への参照として 論理的に処理されますが、属性の完全セットは明示的に宣言されるか、 デフォルトが使用されるかのいずれかです。

VALUE 属性がリーフ要素で指定され、すべての構造体要素のメンバーに対する参照があいまいでない場合、構造体では名前付き定数の名前空間を定義できます。 VALUE 属性を持つ構造体が以下の条件をすべて満たしていれば、その構造体の要素を指定できます。

- 構造体のすべてのリーフ要素に VALUE 属性がある。
- 構造体に配列または共用体が含まれていない。
- 構造体に AUTOMATIC などのストレージ属性がない。

**注 :** 名前付き定数を使用すると、名前が付いていない場合とは 影響が異なってくることがあります。 名前付き定数についての属性は、明示およびデフォルト属性を含む宣言から採用されます。 名前の付いていない定数についての属性は、定数の形、書式、 およびサイズから推論されます。 ストリング・データについては、長さが指定されない場合、 またはアスタリスクを使用して指定されている場合、 その長さは制限付き式の長さから判別します。

名前付き定数は、アプリケーション・プログラム内で使用するために より正確でなければなりません。また、それによって、 より予測しやすい結果が生じることになります。 例えば、名前が付いた定数 Unit を FIXED BINARY VALUE(1) として定義すると、 FIXED BINARY(15) VALUE(1) という属性を持ちます。 単に数字の 1 を使用すると、その属性は FIXED DECIMAL(1,0) になります。 起こりうるその他の違いについては、46 ページの『#unique_273/unique_273_Connect_42_nmconst』 を参照してください。

また、名前付き定数により、アプリケーションの パラメーターを設定することができます。この結果、 デバッグと保守が容易になります。

名前付き定数は、算術データ、ストリング・データとして、 およびポインターやオフセットとして宣言することができます。 名前付き定数は、使用する前に宣言しなければなりません。

**関連情報**

29 ページの『ストリング・データとその属性』
このセクションでは、ストリング・データのタイプおよび属性について説明します。

22 ページの『コード化算術データとその属性』
このトピックでは、コード化算術データの構文を示し、コード化算術データ属性とその省略形をリストします。

## 名前付き定数の例

名前付き定数は、定数が必要なときはいつでも使用することができます。 名前付き定数を、プログラムのあとのほうに出てくる制限付き式の中で使用して、独立定数の計算を行うこともできます。

名前付き定数 では、名前付き定数、ならびに名前付き定数と名前の付いていない定数の間に生じる属性や精度の差を示しています。

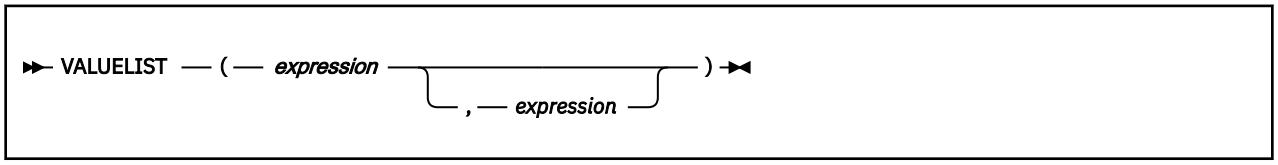
**名前付き定数**

```
Dcl A4 value(148) fixed bin,
    C4 value(261) fixed bin,
    Whole value(800) fixed bin;
Dcl Notes (4) static,
    init(a4,  (Whole/4),   /* 148, 200  */
         c4,  (Whole*2));  /* 261, 1600 */


/* note that "Head" gets length equal to length of VALUE   */


Dcl Head char VALUE('Feel the Power of PL/I'); /* char(22) */
Dcl Headsize fixed bin value(length(Head));     /* 22       */
Dcl 1 Head1 static,
      2 * char(Headsize) initial(Head),        /* char(22) */
      2 * char(20) init(''),
      2 * char(5) init('Page '),
      2 Page_number pic 'zz9',
      2 * char(0);
Dcl TwoHeads char(2*Headsize);                /* char(44) */
Dcl Page0 picture 'zz9' value(0);
Dcl MyNullPtr ptr value(ptrvalue('ffff_ffff'xn));


/* Differences in attributes/results of
```

```
   named and unnamed constants                                    */

Dcl Pi float bin value (3.1416);  /* is FLOAT BINARY(21) but ... */
3.1416                            /* is FIXED DECIMAL(5,4)       */

Dcl Unit fixed bin value(1);      /* is FIXED BINARY(15) but ... */
1                                 /* is FIXED DECIMAL(1,0)       */
1.0                               /* is FIXED DECIMAL(2,1)       */
1B                                /* is FIXED BINARY(1)          */
0000_0000_0000_001B               /* is FIXED BINARY(15)         */

Dcl Title char(20) value('SCIDS'); /* is CHAR(20)      but ... */
Dcl Title2 char     value('SCIDS');/* is CHAR(5)               */
'SCIDS'                            /* is CHAR(5)               */
```

# VALUELIST 属性

VALUELIST 属性は、変数、引数、または戻り値が持つことができる値のセットを制限する、値のリストを指定します。




VALUELIST 属性は、計算タイプおよび序数タイプでのみ有効です.

各式は次のようになっている必要があります。

1. 計算タイプを使用して指定されている場合は計算タイプを保持し、序数タイプを使用して指定されている場合は同じ序数タイプを保持している。
2. 定数値を保持している。
3. リスト内に 1 回のみ出現している (ただし順序は任意)。

48 ページの『例』 を参照してください。

**関連情報**

562 ページの『VALIDVALUE』
VALIDVALUE は、式の値が変数の値セットのエレメントの 1 つと一致するかどうかを示す値を返します。

47 ページの『VALUELISTFROM 属性』
VALUELISTFROM 属性は、すべてのエレメントが VALUE 属性を持ち、現在の宣言の VALUELIST を定義するために使用する必要がある、添え字なしの構造体参照を指定します。

48 ページの『VALUERANGE 属性』
VALUERANGE 属性は、変数、引数、または戻り値が持つことができる値のセットを制限する、値の包括的範囲を指定します。

# VALUELISTFROM 属性

VALUELISTFROM 属性は、すべてのエレメントが VALUE 属性を持ち、現在の宣言の VALUELIST を定義するために使用する必要がある、添え字なしの構造体参照を指定します。

次の場合、

```
dcl 1 a, 2 b fixed bin value(31), 2 c fixed bin value(28), 2 d fixed bin value(31);
dcl x fixed bin valuelistfrom a;
```

x は属性 VALUELIST(31, 28, 31) を持つことになります。

**関連情報**

178 ページの『LIKE 属性』

LIKE 属性は、宣言される名前が、参照される構造体または共用体 (LIKE 属性のオブジェクト) と論理的に同じ編成であることを指定します。

47 ページの『VALUELIST 属性』
VALUELIST 属性は、変数、引数、または戻り値が持つことができる値のセットを制限する、値のリストを指定します。

# VALUERANGE 属性

VALUERANGE 属性は、変数、引数、または戻り値が持つことができる値のセットを制限する、値の包括的範囲を指定します。

VALUERANGE 属性は、計算タイプおよび序数タイプを使用する場合にのみ有効です。 計算タイプが数値の場合、これは REAL である必要があります。

各式は次のようになっている必要があります。

1. 計算タイプを使用して指定されている場合は計算タイプを保持し、序数タイプを使用して指定されている場合は同じ序数タイプを保持している。
2. 定数値を保持している。
3. 前の式よりも大きい。

## 例

次のステートメントがあるとします。

```
define alias numeric_month fixed bin(7) valuerange(1,12);

dcl imonth type numeric_month;

dcl cmonth char(3)
          valuelist( 'Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
                     'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec' );
```

変数 *imonth* は 1 から 12 までの値のみを保持する必要があり、変数 *cmonth* は指定された 12 個の値のいずれか 1 つのみを保持する必要があります。

562 ページの『VALIDVALUE』 組み込み関数を使用すると、変数の値がこれらの属性に指定されている値の 1 つであるかどうかをテストできます。 例えば、上記のステートメントの場合、以下の式は同等です。

```
VALIDVALUE( imonth )

BETWEEN( imonth, 1, 12 )

( 1 <= imonth ) & ( imonth <= 12 )
```

コンパイラーでこれらの属性を使用して、コードを最適化し、制限された値セットの 1 つのみがこのような変数に割り当てられていることを検査することもできます。 例えば、上記のステートメントの場合、以下のステートメントはどちらも無効です。

```
 dcl month_due type numeric_month init(0);

 cmonth = '';
```

### 関連情報

562 ページの『VALIDVALUE』
VALIDVALUE は、式の値が変数の値セットのエレメントの 1 つと一致するかどうかを示す値を返します。

47 ページの『VALUELIST 属性』

VALUELIST 属性は、変数、引数、または戻り値が持つことができる値のセットを制限する、値のリストを指定します。

# 第 3 章 式および参照

この章では、式および参照のさまざまなタイプを説明しています。

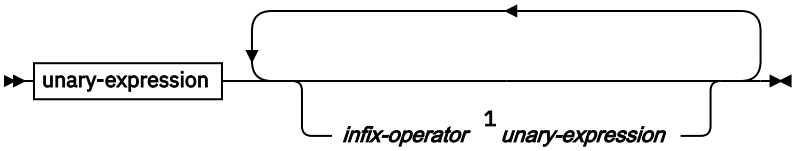
式 は、値を表すものです。 式は以下のいずれかになります。

- 単一定数、変数、または関数参照
- 定数、変数、または関数参照の任意の組み合わせ (組み合わせで使用される演算子および括弧を含む)

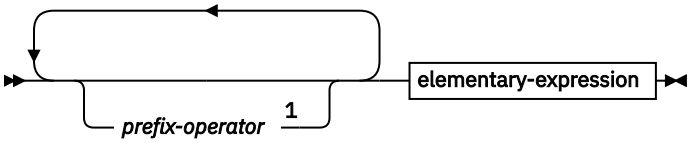
演算子が含まれている式を演算式 といいます。

演算式の定数および変数は、オペランド と呼ばれます。 詳細については、54 ページの『演算式』を参照してください。

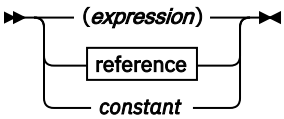
以下の図は、式および参照の構文を示しています。



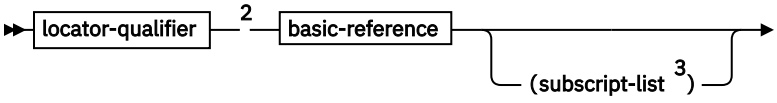

**unary-expression**

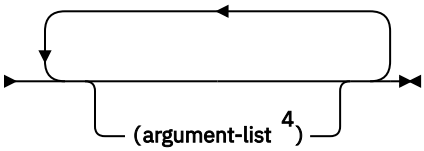



**elementary-expression**

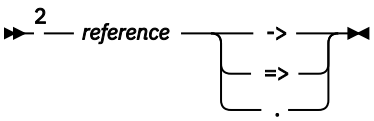



**reference**

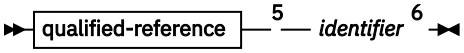



**locator-qualifier**

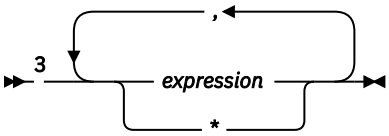



**basic-reference**

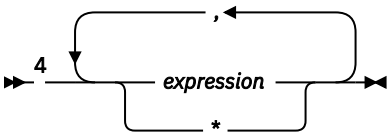



**subscript-list**

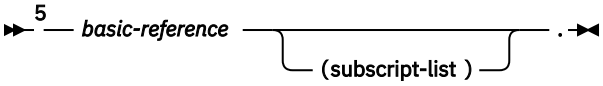
© Copyright IBM Corp. 1999, 2022




**argument-list**




**qualified-reference**




注:

[1] Operators are shown in 7 ページの表 6.

[2] Locator-qualifier is described under 245 ページの『ロケーター修飾』 and 141 ページの『タイプ付き構造体修飾』.

[3] Subscripts are described under 170 ページの『配列』.

[4] Arguments are described in 106 ページの『プロシージャーへの引数の引き渡し』.

[5] Qualified-reference is described under 176 ページの『構造体および共用体の修飾』.

[6] Identifiers are described under 5 ページの『ID』.

式は要素式 (スカラー式ともいう)、配列式、または構造式 のいずれかに分類されます。 要素変数および配列変数は、同じ式内に書くことができます。

**要素式**
単一の値を表します。 この定義には、構造体や共用体内の基本名や、 配列の単一エレメントを指定する添え字付きの名前も含まれます。

**配列式**
値の配列を表します。 この定義には、次元属性を持つ構造体や共用体のメンバーが含まれます。

**構造式**
構造化された値のセットを表します。

次の例を考えてください。

```
dcl A(10,10) bin fixed(31),
    B(10,10) bin fixed(31),
    1 Rate,
      2 Primary dec fixed(4,2),
      2 Secondary dec fixed(4,2),
    1 Cost(2),
      2 Primary dec fixed(4,2),
      2 Secondary dec fixed(4,2),
    C bin fixed(15),
    D bin fixed(15);
dcl Pi bin float value(3.1416);
```

以下は要素式です。

```
Pi
27
C
C * D
A(3,2) + B(4,8)
Rate.Primary - Cost.Primary(1)
A(4,4) * C
Rate.Secondary / 4
A(4,6) * Cost.Secondary(2)
sum(A)
addr(Rate)
```

以下は配列式です。

```
A
A + B
A * C - D
B / 10B
```

大半の PL/I ステートメントの構文では、 式の計算結果がステートメントの構文規則に合ってさえいれば、式を使うことができます。 構文仕様の後のテキストで特に明記していない限り、修飾されていない用語式または参照 はスカラー式を指します。 スカラー式以外の式については、式のタイプが明記してあります。例えば、配列式 という用語は、スカラー式が無効であることを示します。

構造式の例を以下に示します。

```
Rate = Rate*2
```

# 計算の順序

PL/I ステートメントには、複数の式または参照が含まれていることがしばしばあります。 特定インスタンス (例えば、代入ステートメントなど) についての説明があるときを除いて、計算の順序は任意であり、(概念的には) 同時に計算することも可能です。

次の例を考えてください。

```
dcl (X,Y,Z) entry returns(float), (F,G,H) float;
F = X( Y(G,H), Z(G,H) );
```

関数 Y と Z は、 それらに渡された引数の値を変更することがあります。 したがって、X が返す値は、 どちらの関数が最初に呼び出されるかに応じて異なります。 最初のパラメーターが最初に計算されると推定すべきではありません。 最後のパラメーターを最初に計算するほうが望ましい場合もあります。

INC 機能がそれに渡された引数の値を増やし、更新された値を戻すと仮定すると、以下の例では、B(1,2) または B(2,1) は、 どちらの添え字が最初に計算されるかによって、結果が決まります。 どちらの添え字が最初に計算されるかを推定すべきではありません。

```
dcl B(2,2);
I = 0;
put list ( B( INC(I), INC(I) ) );
```

# ターゲット

式または変換の計算結果は、ターゲット に割り当てられます。 ターゲットは、変数、疑似変数、または中間結果フィールドのいずれかです。

## 変数

変数は、式評価または式変換のターゲットとすることができます。

ステートメント A = B; のような代入の場合、ターゲットは代入記号の左側にある変数 (この場合は A) です。 変数に対する割り当ては、ストリーム入出力、DO、DISPLAY、およびレコード入出力ステートメント内でも行うことができます。

## 疑似変数

疑似変数はターゲット・フィールドを表します。

次の例を考えてください。

```
declare A character(10),
        B character(30);
substr(A,6,5) = substr(B,20,5);
```

この代入ステートメントでは、SUBSTR 組み込み関数は、ストリング B の 20 番目の文字から始まる長さ 5 のサブストリングを取り出します。 SUBSTR 疑似変数は、ストリング A の中の場所を表しますが、 この場所がターゲットです。 したがって、A の最後の 5 文字は B の 20 番目から 24 番目の文字で置き換えられます。 A の最初の 5 文字は変更されません。

疑似変数に関する情報については、373 ページの『第 18 章 組み込み関数、疑似変数、およびサブルーチン』を参照してください。

## 中間結果

式が計算されるとき、通常のターゲット属性は、ソース、実行される演算、および第 2 オペランドの属性から部分的に取り出されます。 デフォルトが使用されたり、インプリメンテーション制限事項および規則 (例えば、 最大の精度など) が存在していることもあります。 これ以降に演算が行われると、中間の結果が変換される場合があります。 式の値が計算されたあと、変数または疑似変数に割り当てるため に、その結果さらに変換されることもあります。 これらの変換は、プログラマー定義のデータが変換される場合と同じ規則に 従って行われます。

次の例を考えてください。

```
declare A character(8),
        B fixed decimal(3,2),
        C fixed binary(10);
A = B + C;
```

式 B + C の計算時 およびその結果の割り当て時に、4 つの異なる結果が生じます。

1. 中間結果。B が 2 進数に変換されて、ここに割り当てられます。
2. 中間結果。2 進数の加算結果がここに割り当てられます。
3. 中間結果。2 進数の結果が 10 進固定小数点フォーマットに変換されて、ここに割り当てられます。
4. A。最終的に結果を入れるためのフィールド。10 進固定小数点の値が 文字フォーマットに変換されて、ここに割り当てられます。

最初の結果の属性は、ソース B の属性、演算子、 および一方のオペランドの属性によって決まります。 算術挿入演算子の一方のオペランドが 2 進数であれば、もう一方のオペランドは、演算の前に 2 進数に変換されます。

2 番目の結果の属性は、ソース (C と変換された 表示の B) の属性から判別します。

3 番目の結果の属性は、ソース (2 番目の結果) の一部と、最終ターゲット A の属性の一部で決定されます。最終ターゲットから決定される属性は DECIMAL のみです (2 進数の算術表現は、文字値に変換する前に 10 進数表現に変換する必要があります)。

A の属性は、DECLARE ステートメントによって決まります。

# 演算式

演算式は、1 つまたは複数の単一演算で形成されます。 単一演算とは、接頭演算子演算 (単一オペランドの前に 演算子がある) または挿入演算子演算 (2 つのオペランドの間に 演算子がある) のどちらかです。 挿入演算子演算の 2 つのオペランドは、通常は、演算実行時には データ・タイプが同じでなければなりません。

PL/I では、式に含まれている演算のオペランドは、 演算が行われる前に、必要に応じて同じデータ・タイプに変換されます。 変換の詳細な規則については、73 ページの『第 4 章 データ変換』を参照してください。

1 つの式で異なるデータ・タイプを使用することに関しては、 ほとんど制約はありません。 ただし、各種のデータ・タイプが混在しているときは変換が行われます。 実行時に変換が行われると、プログラムの実行時間が長くなります。 さらに、変換によって精度が失われることもあります。 データ・タイプが混在している式を使用する場合は、関連する変換規則を理解しておく必要があります。

演算のクラスには、ハンドル、ポインター、算術、ビット、比較、および連結があります。

## ハンドル演算

以下のハンドル演算を PL/I プログラムで使用できます。

- 同じ構造体タイプが関連付けられた 2 つのハンドルを比較します。
- 関連する構造体タイプを考慮して、ハンドルに式を足したり、またはハンドルから式を引いたりします。

  例えば、*x* が構造体タイプ *t* のハンドルであり、*n* が整数値である場合、*x*+*n* および *x*–*n* は *t* のハンドルです。

  – *x*+*n* は、*x* の関連ポインター値を、構造体 *t* のサイズの *n* 倍だけ増やします。

  – *x*-*n* は、*x* の関連ポインター値を、構造体 *t* のサイズの *n* 倍だけ減らします。

  また、+= および –= 複合代入演算子を使用して、ハンドルを増加および減少させることもできます。
- 関連する構造体タイプを考慮して、2 つのハンドルの相違を計算します。

  例えば、*x* および *y* が構造体タイプ *t* のハンドルである場合、*x*-*y* の結果は、*x* と *y* の間の *t* のインスタンスの数です。 値 *x*-*y* は、(PTRVALUE(*x*)-PTRVALUE(*y*)) / SIZE(:t:) の結果と等しくなります。

  異なる構造体タイプへの 2 つのハンドルの相違を計算することはできません。

## ポインター演算

以下のポインター演算を PL/I プログラムで使用できます。

- ポインター式に式を追加したり、またはそこから式を引いたりします。

  式のタイプは、計算式でなければなりません。 必要に応じて、非ポインター・オペランドは *size_t*[1] に変換されます。以下の例を参照してください。

```
Ptr1 = Ptr1 - 16;
Ptr2 = Ptr1 + (I*J);
```

  これらの演算を行うために、POINTERADD 組み込み関数を使用することもできます。結果をロケーター参照として使用する場合には、POINTERADD を使用しなければ なりません。 例えば、(Ptr1 + 16) -> Based_ptr は無効で、pointeradd(Ptr1,16) -> Based_ptr は有効です。
- ポインターからポインターを引き、ポインター間でその論理的な差を計算します。 結果は *size_t*[1] 値です。

```
Diff = Ptr2 - Ptr1;
```

- 挿入演算子を使用して、ポインター式を比較します。

```
if Ptr2 > Ptr1 then
   Diff = Ptr2 - Ptr1;
```

- ポインター式をヌル・ストリング ('' または ''b) と比較します。

  DEFAULT コンパイラー・オプションの NULLSTRPTR サブオプションは、コンパイラーが、ヌル・ストリングのポインターへの割り当てと、ヌル・ストリングとポインターの比較を処理する方法を決定します。どちらのケースでも、このオプションは、ヌル・ストリングが、NULL 組み込み関数または SYSNULL 組み込み関数に対する参照として扱われるかどうかを決定します。

  例えば、NULLSTRPTR(SYSNULL) サブオプションが有効である場合、以下のコードの割り当てでは、SYSNULL() が header に割り当てられ、header が SYSNULL() に等しい場合には、比較は真の値を生成します。

```
dcl header pointer;

header = '';
```

[1] LP(32) コンパイラー・オプションが有効な場合、*size_t* は FIXED BIN(31) です。 LP(64) コンパイラー・オプションが有効な場合、*size_t* は FIXED BIN(63) です。

```
...

if header = '' then...
```

- BINARYVALUE 組み込み関数を使用して、算術コンテキストでポインター式を使用します。

```
Diff = Bin31 + binaryvalue(Ptr1);
```

- POINTERVALUE 組み込み関数を使用して、ポインター・コンテキストで計算による式を使用します。

```
dcl 1 Cvtprt pointer based(pointervalue(16));
dcl 1 Cvt based(Cvtptr),
      2 Cvt ...;
```

必要に応じて、式は *size_t*[1] に変換されます。

PL/I ブロックでは、ポインター算術を使用して、 構造体または配列変数内の任意のエレメントにアクセスすることができます。 ただし、ブロックは、構造体あるいは配列変数の入っているものに渡さなければならず、その名前の有効範囲内に参照できる集合を持っていなければなりません。

# 算術演算

算術演算は、1 つの算術演算子を持つ結合オペランドによって指定します。 ADD、SUBTRACT、DIVIDE、および MULTIPLY 組み込み関数を使用して、 算術演算を使用することもできます。

算術演算では以下の演算子を使用できます。

表 *24.* 算術演算子

| 演算子 | 演算子名 |
|---|---|
| + | 追加 |
| - | 減算 |
| * | 乗算 |
| / | 除算 |
| ** | 累乗演算 |

正符号 (+) と負符号 (-) は、接頭演算子として使うことも、挿入演算子として 使うこともできます。 これ以外のすべての算術演算子は、挿入演算子としてのみ使用することが できます

接頭演算子は、挿入演算子の任意のオペランドの前に付けることができ、 そのオペランドと関連があります。 例えば、A*-B という式の場合、負符号 (-) は A の値を B の値の -1 倍で乗じることを示しています。

1 つの変数に複数の接頭演算子を付けることができます。 正の接頭演算子を複数付けても、累積的な効果はありませんが、負の演算子を 2 つ続けると、正の接頭演算子を 1 つ使用するのと同じ結果になります。

## 算術演算におけるデータ変換

算術演算の 2 つのオペランドは、タイプ、基数、モード、精度、およびスケールの点で異なっていても構いません。 異なっているときは、変換が行われます。

コード化算術オペランドの場合は、58 ページの表 25 を 使用して変換方法を判別することができます。 各オペランドは、計算結果のタイプ、基数、およびモードに変換されます。 計算結果の精度とスケールに変換する必要はありません。

**注 :** スケールされる FIXED BINARY オペランドは、スケールされる FIXED DECIMAL に変換されてから、それについての演算が行われます。

### タイプ

文字オペランドは、FIXED DECIMAL(N,0) に変換されます。 ビット・オペランドは、FIXED BINARY(M,0) に変換されます。 数字オペランドは、ピクチャー指定によって決まる位取りおよび精度を持つ DECIMAL に変換されます。

最大値については、615 ページの『付録 A 制限値』を参照してください。

グラフィック、ワイド文字、および uchar の変数とストリングは、すべての計算コンテキストで使用できます。 変換する必要があるときは、従うべき規則は、文字に対しても同様になります。

算術演算の結果は、常に、コード化算術フォーマットになります。 算術挿入演算子演算で行われる可能性のある変換は、タイプ変換だけです。

### 基数
2 つのオペランドの基数が異なっている場合、10 進数オペランドは、同等の 2 進数に変換されます。

### モード
2 つのオペランドのモードが異なっている場合は、実数オペランドが複素数モードに変換されます。これは、実数部分と同じ基数、スケール、および精度を与えて、虚数部分ゼロを設けるという方法で行われます。 ただし、例外があります。

累乗演算では、第 2 オペランド (演算の指数) が位取り係数ゼロを持つ固定小数点実数である場合、変換は不要です。

### 精度
精度や位取り係数だけ (または両方) が異なっている場合は、タイプ変換は不要です。

### スケール
2 つのオペランドのスケールが異なっている場合は、固定小数点オペランドが 浮動小数点のスケールに変換されます。 ただし、例外があります。

累乗演算では、第 1 オペランドが浮動小数点の位取り、第 2 オペランド (演算の指数) が位取り係数ゼロを持つ固定小数点実数、つまり精度 (p,0) で宣言された整数または変数である場合、変換は不要ですが、結果は浮動小数点になります。

累乗演算の両オペランドが固定小数点数である場合は、 次の変換方法のうちの 1 つが行われます。

- 指数の精度が (p,0) 以外であれば、両オペランドが浮動小数点数に変換されます。
- 指数が符号なし整数でない場合、第 1 オペランドが浮動小数点数に変換されます。
- 固定小数点累乗演算を行うと、使用可能な最大桁数を超えることが両オペランドの 精度から明らかになる場合は、第 1 オペランドが浮動小数点数に変換されます。

## 算術演算の結果
式の中のオペランドが必要に応じて変換されたあと、算術演算が行われ、結果が出されます。 この結果は、式の値であることも、さらに後続の演算の対象となる中間結果であることもあります。あるいは、なんらかの条件が起こることもあります。

58 ページの表 25 と 59 ページの表 26 は、 各種の算術演算から帰結する属性と精度を示しています。

62 ページの表 30 は、58 ページの表 25 と 59 ページの表 26 の 右端の欄に注記している累乗演算の特殊事例における結果の属性を示しています。

FIXED(p1,q1) および FIXED(p2,q2)、ABS(q1-q2) の加算または減算は、DECIMAL の N と BINARY の M 以下でなければなりません。

FIXED(p1,q1) と FIXED(p2,q2) の乗算または除算では、結果のスケール因数は -128 から 127 の間でなければなりません。

z/OS プラットフォームで浮動小数点の計算にどのセットの命令を使用するかは、次の 2 つのコンパイラー・オプションによって決まります。

- FLOAT(DFP) を指定した場合
  - 結果が FLOAT DEC となる計算はすべて、IEEE 10 進浮動小数点命令を使用して行われます。
  - 結果が FLOAT BIN となる計算はすべて、DEFAULT コンパイラー・オプションの HEXADEC および IEEE サブオプションで指定されたフォーマット用の浮動小数点命令を使用して行われます。
- FLOAT(NODFP) を指定した場合

– 結果が FLOAT となる計算はすべて、DEFAULT コンパイラー・オプションの HEXADEC および IEEE サブオプションで指定されたフォーマット用の浮動小数点命令を使用して行われます。

したがって、FLOAT(NODFP) および DEFAULT(HEXADEC) オプションのもとでは、すべての計算は 16 進浮動小数点命令を使用して行われ、IEEE で宣言された変数は HEXADEC に変換されます。 一方、FLOAT(NODFP) および DEFAULT(IEEE) オプションのもとでは、すべての計算は、IEEE 2 進浮動小数点命令を使用して行われ、HEXADEC で宣言された変数は必要に応じて IEEE に変換されます。

その他すべてのプラットフォームでは、浮動小数点の計算は、そのプラットフォームに固有の IEEE 2 進浮動小数点命令を使用して行われます。

コンパイラー・オプション RULES(ANS) のもとでは、一方のオペランドが FIXED DECIMAL にスケールされており、他方のオペランドが FIXED BINARY にスケールされていると、FIXED BINARY 値は FIXED DECIMAL に変換されます。 60 ページの表 27 は、コンパイラー・オプション RULES(ANS) のもとでの この場合の結果の属性および精度を示しています。 RULES コンパイラー・オプションの詳細については、「プログラミング・ガイド」を参照してください。

表 *25. 1* つまたは複数の *FLOAT* オペランドの算術演算の結果

<table>
<tr><th>第 1<br>オペランド<br>(p1,q1)</th><th>第 2<br>オペランド<br>(p2,q2)</th><th>以下の<br>結果の属性<br>加算、<br>減算、<br>乗算、<br>または除算</th><th>追加<br>又は<br>減算<br>精度</th><th>乗算<br>精度</th><th>除算<br>精度</th><th>属性<br>以下の<br>結果<br>累乗演算</th></tr>
<tr><td>FLOAT<br>DECIMAL<br>(p1)</td><td>FLOAT<br>DECIMAL<br>(p2)</td><td rowspan="3">FLOAT<br>DECIMAL<br>(p)</td><td rowspan="6" colspan="3">p = MAX(p1,p2)</td><td rowspan="3">FLOAT<br>DECIMAL (p)<br>(特殊な場合以外は、<br>C が適用される)<br>p = MAX(p1,p2)</td></tr>
<tr><td>FLOAT<br>DECIMAL<br>(p1)</td><td>固定<br>DECIMAL<br>(p2,q2)</td></tr>
<tr><td>固定<br>DECIMAL<br>(p1,q1)</td><td>FLOAT<br>DECIMAL<br>(p2)</td></tr>
<tr><td>FLOAT<br>BINARY<br>(p1)</td><td>FLOAT<br>BINARY<br>(p2)</td><td rowspan="3">FLOAT<br>BINARY<br>(p)</td><td rowspan="3">FLOAT<br>BINARY (p)<br>(特殊な場合以外は、<br>C が適用される)<br>p = MAX(p1,p2)</td></tr>
<tr><td>FLOAT<br>BINARY<br>(p1)</td><td>固定<br>BINARY<br>(p2,q2)</td></tr>
<tr><td>固定<br>BINARY<br>(p1,q1)</td><td>FLOAT<br>BINARY<br>(p2)</td></tr>
<tr><td>固定<br>DECIMAL<br>(p1,q1)</td><td>FLOAT<br>BINARY<br>(p2)</td><td rowspan="3">FLOAT<br>BINARY<br>(p)</td><td rowspan="3" colspan="3">p = MAX(<br>CEIL(p1*3.32),p2)</td><td rowspan="3">FLOAT<br>BINARY (p)<br>(特殊な場合以外は、<br>A または C が<br>適用される)<br>p = MAX(<br>CEIL(p1*3.32),p2)</td></tr>
<tr><td>FLOAT<br>DECIMAL<br>(p1)</td><td>固定<br>BIN<br>(p2,q2)</td></tr>
<tr><td>FLOAT<br>DECIMAL<br>(p1)</td><td>FLOAT<br>BINARY<br>(p2)</td></tr>
</table>

表 *25. 1* つまたは複数の *FLOAT* オペランドの算術演算の結果 *(続き)*

| **第 1 オペランド (p1,q1)** | **第 2 オペランド (p2,q2)** | **以下の 結果の属性 加算、 減算、 乗算、 または除算** | **追加 又は 減算 精度** | **乗算 精度** | **除算 精度** | **属性 以下の 結果 累乗演算** |
|---|---|---|---|---|---|---|
| 固定 BINARY (p1,q1) | FLOAT DECIMAL (p2) | FLOAT BINARY (p) | p = MAX(p1,CEIL(p2*3.32)) | | | FLOAT BINARY (p) (特殊な場合以外は、 B または C が 適用される) p = MAX( p1,CEIL(p2*3.32)) |
| FLOAT BINARY (p1) | 固定 DECIMAL (p2,q2) | | | | | |
| FLOAT BINARY (p1) | FLOAT DECIMAL (p2) | | | | | |

表 *25. 1* つまたは複数の *FLOAT* オペランドの算術演算の結果

**注記 :**

1. 累乗演算の特殊事例は、62 ページの表 30 で説明しています。
2. CEIL(N*3.32) 値の表については、76 ページの表 35 を参照してください。

表 *26. RULES(ANS)* における、*2* つのスケールのない *FIXED* オペランド間の算術演算の結果

| **第 1 オペランド (p1,q1)** | **第 2 オペランド (p2,q2)** | **以下の 結果の属性 加算、 減算、 乗算、 または除算** | **追加 または 減算 精度** | **乗算 精度** | **除算 精度** | **属性 以下の 結果 累乗演算** |
|---|---|---|---|---|---|---|
| 固定 DECIMAL (p1,0) | 固定 DECIMAL (p2,0) | 固定 DECIMAL (p,q) | p = 1 +MAX(p1,p2) q = 0 | p = 1 +p1+p2 q = 0 | p = N q = N-p1 | FLOAT DECIMAL (p) (特殊な場合以外は、 A が適用される) p = MAX(p1,p2) |
| 固定 BINARY (p1,0) | 固定 BINARY (p2,0) | 固定 BINARY (p,0) | p = 1 +MAX(p1-q1, p2-q2) +q q = 0 | p = 1+p1 +p2 q = 0 | p = M q = 0 | FLOAT BINARY (p) (特殊な場合以外は、 B が適用される) p = MAX(p1,p2) |
| 固定 DECIMAL (p1,0) | 固定 BINARY (p2,0) | 固定 BINARY (p,0) | p = 1 +MAX(r,p2) q = 0 | p = 1 +r+p2 q = 0 | p = M q = 0 | FLOAT BINARY (p) (特殊な場合以外は、 A が適用される) p = MAX(CEIL (p1*3.32 ),p2) |
| 固定 BINARY (p1,0) | 固定 DECIMAL (p2,0) | 固定 BINARY (p,0) | p = 1 +MAX(p1,t) q = 0 | p = 1 +p1+t q = 0 | p = M q = 0 | FLOAT BINARY (p) (特殊な場合以外は、 B が適用される) p = MAX(CEIL (p1*3.32 ),p2) |
| *M* は FIXED BINARY の最大精度です。<br>*N* は FIXED DECIMAL の最大精度です。<br>r = 1 + CEIL(p1*3.32)<br>s = CEIL(ABS(q1*3.32)) * SIGN(q1) | | | | t = 1 + CEIL(p2*3.32)<br>u = CEIL(ABS(q2*3.32)) * SIGN(q2)<br>v = CEIL(p2/3.32)<br>w = CEIL(p1/3.32) | | |

表 *26. RULES(ANS)* における、*2* つのスケールのない *FIXED* オペランド間の算術演算の結果

**注記 :**

位取り係数は、-128 から +127 の範囲内になければなりません。

1. 累乗演算の特殊事例は、62 ページの表 30 で説明しています。
2. CEIL(N*3.32) 値の表については、76 ページの表 35 を参照してください。
3. RULES(ANS) においては、 スケールのない FIXED オペランドを持つ除算は、 両方のオペランドが FIXED DECIMAL の場合にのみ スケールのある結果を生じます。

表 *27. RULES(ANS)* における、*2* つのスケールされた *FIXED* オペランド間の算術演算の結果

| **第 1 オペランド (p1,q1)** | **第 2 オペランド (p2,q2)** | **以下の 結果の属性 加算、 減算、 乗算、 または除算** | **追加 または 減算 精度** | **乗算 精度** | **除算 精度** | **属性 以下の 結果 累乗演算** |
|---|---|---|---|---|---|---|
| 固定 DECIMAL (p1,q1) | 固定 DECIMAL (p2,q2) | 固定 DECIMAL (p,q) | p = 1 + MAX(p1-q1, p2-q2) +q<br>q = MAX(q1,q2) | p = 1 +p1+p2<br>q = q1+q2 | p = N<br>q = N-p1+q1-q2 | FLOAT DECIMAL (p)<br>(特殊な場合以外は、A が適用される)<br>p = MAX(p1,p2) |
| 固定 DECIMAL (p1,q1) | 固定 BINARY (p2,0) | 固定 DECIMAL (p,q) | p = 1 +MAX(p1-q1,v) +q<br>q = q1 | p = 1 +p2+v<br>q = q1 | p = N<br>q = N-p1+q1 | FLOAT BINARY (p)<br>(特殊な場合以外は、A が適用される)<br>p = MAX(CEIL (p1*3.32 ),p2) |
| 固定 BINARY (p1,0) | 固定 DECIMAL (p2,q2) | 固定 DECIMAL (p,q) | p = 1 +MAX(p2-q2,w) +q<br>q = q2 | p = 1 +p2+w<br>q = q1 | p = N<br>q = N-w-q2 | FLOAT BINARY (p)<br>(特殊な場合以外は、B が適用される)<br>p = MAX(CEIL (p1*3.32 ),p2) |
| *M* は FIXED BINARY の最大精度です。<br>*N* は FIXED DECIMAL の最大精度です。<br>r = 1 + CEIL(p1*3.32)<br>s = CEIL(ABS(q1*3.32)) * SIGN(q1) | | | | t = 1 + CEIL(p2*3.32)<br>u = CEIL(ABS(q2*3.32)) * SIGN(q2)<br>v = CEIL(p2/3.32)<br>w = CEIL(p1/3.32) | | |

表 *27. RULES(ANS)* における、*2* つのスケールされた *FIXED* オペランド間の算術演算の結果

**注記 :**

位取り係数は、-128 から +127 の範囲内になければなりません。 加算および減算の演算では、スケール因数の差の絶対値は、FIXED BIN 演算の場合は M、FIXED DEC 演算の場合は N を超えてはなりません。

1. 累乗演算の特殊事例は、62 ページの表 30 で説明しています。
2. CEIL(N*3.32) 値の表については、76 ページの表 35 を参照してください。
3. RULES(ANS) のもとでは、スケールされた FIXED BINARY は使用できません。

表 *28. RULES(IBM)* における、*2* つの *FIXED* オペランド間の算術演算の結果

| 第 1 オペランド (p1,q1) | 第 2 オペランド (p2,q2) | 以下の 結果の属性 加算、 減算、 乗算、 または除算 | 追加 又は 減算 精度 | 乗算 精度 | 除算 精度 | 属性 以下の 結果 累乗演算 |
|---|---|---|---|---|---|---|
| 固定 DECIMAL (p1,q1) | 固定 DECIMAL (p2,q2) | 固定 DECIMAL (p,q) | p = 1 +MAX(p1-q1, p2-q2) +q<br>q = MAX(q1,q2) | p = 1 +p1+p2<br>q = q1+q2 | p = N<br>q = N-p1+q1-q2 | FLOAT DECIMAL (p) (特殊な場合以外は、 A が適用される)<br>p = MAX(p1,p2) |
| 固定 BINARY (p1,q1) | 固定 BINARY (p2,q2) | 固定 BINARY (p,q) | p = 1 +MAX(p1-q1, p2-q2) +q<br>q = MAX(q1,q2) | p = 1 +p1+p2<br>q = q1+q2 | p = M<br>q = M-p1 +q1-q2 | FLOAT BINARY (p) (特殊な場合以外は、 B が適用される)<br>p = MAX(p1,p2) |
| 固定 DECIMAL (p1,q1) | 固定 BINARY (p2,q2) | 固定 BINARY (p,q) | p = 1 +MAX(r-s, p2-q2)+q<br>q = MAX(s,q2) | p = 1+r +p2<br>q = s+q2 | p = M<br>q = M-r +s-q2 | FLOAT BINARY (p) (特殊な場合以外は、 A が適用される)<br>p =MAX( CEIL((p1*3.32 ),p2) |
| 固定 BINARY (p1,q1) | 固定 DECIMAL (p2,q2) | 固定 BINARY (p,q) | p = 1 +MAX(p1-q1,t-u) +q<br>q = MAX(s,q1,u) | p = 1 +p1+t<br>q = q1+u | p = M<br>q = M-p1 +q1-u | FLOAT BINARY (p) (特殊な場合以外は、 B が適用される)<br>p = MAX(p1, CEIL(p2*3.32)) |
| *M* は FIXED BINARY の最大精度です。<br>*N* は FIXED DECIMAL の最大精度です。<br>r = 1 + CEIL(p1*3.32)<br>s = CEIL(ABS(q1*3.32)) * SIGN(q1) | | | | t = 1 + CEIL(p2*3.32)<br>u = CEIL(ABS(q2*3.32)) * SIGN(q2)<br>v = CEIL(p2/3.32)<br>w = CEIL(p1/3.32) | | |

表 *28. RULES(IBM)* における、*2* つの *FIXED* オペランド間の算術演算の結果

**注記 :**

位取り係数は、-128 から +127 の範囲内になければなりません。 加算および減算の演算では、スケール因数の差の絶対値は、FIXED BIN 演算の場合は M、FIXED DEC 演算の場合は N を超えてはなりません。

1. 累乗演算の特殊事例は、62 ページの表 30 で説明しています。
2. CEIL(N*3.32) 値の表については、76 ページの表 35 を参照してください。
3. FIXED BINARY の精度 p の境界は LIMITS コンパイラー・オプションの FIXEDBIN サブオプションによって決まり、FIXED DECIMAL の境界は FIXEDDEC サブオプションによって決まります。 LIMITS オプションは、位取り係数 q の境界では効力を持ちません。

次の式について考察します。

```
A * B + C
```

まず、演算 A * B が行われ、中間結果が生じます。 次に、(中間結果) + C という演算が行われて、式の値が求められます。

PL/I は、属性を別の変数に指定するのと同じ方法で 中間結果属性を指定します。 結果の属性は、2 つのオペランドの属性 (または接頭部演算の場合には 単一オペランド)、および関係する演算子から派生します。 結果の属性を求める方法については、53 ページの『ターゲット』 に説明されています。

組み込み関数 ADD、SUBTRACT、MULTIPLY、および DIVIDE を使用すると、加算、 減算、乗算、および除算演算に関してインプリメンテーションで定められている 精度の規則を無効にすることができます。

## *FIXED* 除算

固定小数点除算を行うと、オーバーフローや切り捨てが起こることがあります。

次の式について考察します。

```
25+1/3
```

この式の評価結果は未定義であり、固定小数点除算の結果が実装環境で定義されている最大精度の値になるために FIXEDOVERFLOW 条件が発生します。

続いて、次の式について考察します。

```
25+01/3
```

定数には作成される精度があるので、 式は 25.3333333333333 (最大精度は 15) という結果になります。

2 つの評価の結果を 62 ページの表 29 に示します。

表 *29.* 固定小数点除算と定数式の比較

| アイテム | 精度 | 結果 |
|---|---|---|
| 1<br>3<br>1/3<br>25<br>25+1/3 | (1,0)<br>(1,0)<br>(15,14)<br>(2,0)<br>(15,14) | 1<br>3<br>0.33333333333333<br>25<br>未定義 (undefined)<br>(左側が切り捨てられます。<br>FIXEDOVERFLOW が起こります。<br>最大値が<br>精度は 15) |
| 01<br>3<br>01/3<br>25<br>25+01/3 | (2,0)<br>(1,0)<br>(15,13)<br>(2,0)<br>(15,13) | 01<br>3<br>00.3333333333333<br>25<br>25.3333333333333 |

PRECISION 組み込み関数を使用することもできます。次の例を参照してください。

```
25+prec(1/3,15,13)
```

**注 :** 正確な精度が必要な状況では、名前のついた定数をお勧めします。

## 累乗演算の使用

このトピックでは、PL/I での指数の取扱方法について説明します。

表 *30.* 累乗演算の特殊事例

| ケース | 第 1 オペランド | 第 2 オペランド | 結果の属性 |
|---|---|---|---|
| A | FIXED DECIMAL (p1,q1) | 値が n の整数 | FIXED DECIMAL (p,q)<br>(p <= N である場合)<br>ここで<br>p = (p1 + 1)*n-1<br>q = q1*n |

表 *30.* 累乗演算の特殊事例 *(*続き*)*

| ケース | 第 1 オペランド | 第 2 オペランド | 結果の属性 |
|---|---|---|---|
| B | FIXED BINARY (p1,q1) | 値が n の整数 | FIXED BINARY (p,q)<br>(p <= M である場合)<br>ここで<br>p = (p1 + 1)*n-1<br>q = q1*n |
| C | FLOAT (p1) | FIXED (p2,0) | 第 1 オペランドと同じ基数の FLOAT (p1) |

**実モードまたは複素数モードでの x**y の特殊事例 :**

**実モード:**
**複素数モード:**

**x=0 および y>0 の場合、**
**結果は 0** になります。 x=0、および y の実数部>0 と y の虚数部=0 の場合、結果は 0 になります。

**x=0 かつ y<=0**
**エラー条件が発生します。** x=0 かつ y の実数部分 <=0 または y の虚数部分 ¬=0 ならば、ERROR 条件が発生します。

**x<0 かつ y が FIXED (p,0) でないとき**
**エラー条件が発生します。** x¬=0 かつ y の実数部分および虚数部分 =0 ならば、結果は 1。

### 単純計算での指数の使用に関する考慮事項

指数を使用する場合、単純計算は正しくないものになることがあります。 例えば、10**-3 の結果は 0.0010 ではなく誤って 0.0009 になります。 このような場合は、結果を丸めてください。 あるいは、指数を使用せずに、除算または乗算を使用してください。

## ビット演算子

ビット演算子は、論理演算子を持つ結合オペランドによって指定します。

下表に、ビット演算子に使用できる論理演算子と、各演算子が接頭演算子または挿入演算子として使用可能かどうかを示します。

表 *31.* ビット演算子用の論理演算子

| 演算子記号 | 演算子名 | 接頭演算子として | 挿入演算子として |
|---|---|---|---|
| ¬ | *NOT*排他 *OR* | Yes | Yes |
| & | と | No | Yes |
| \| | *or* | No | Yes |

これらの演算子は、ブール代数の場合と同じです。

ビット演算のオペランドは、演算の実行前に、必要に応じてビット・ストリングに変換されます。 挿入演算子の演算の両オペランドの長さが異なっている場合は、短い方のストリングの右側に '0'B が埋め込まれます。

ビット演算の演算結果は、両オペランドと同じ長さのビット・ストリングです。

ビット演算は、1 ビットずつ行われます。 64 ページの表 32 は、各演算子のビット位置の結果を説明しています。

表 *32.* ビット演算子

| A | B | ¬A | ¬B | A&B | A\|B | A¬B |
|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 |

64 ページの表 33 は、ビット演算子の例を示しています。

表 *33.* ビット演算の例

| オペランドと値 | 演算 | 結果 |
|---|---|---|
| A = '010111'B<br>B = '111111'B<br>C = '110'B<br>D = 5 | ¬ A | '101000'B |
| | ¬ C | '001'B |
| | C & B | '110000'B |
| | A \| B | '111111'B |
| | A ¬ B | ''101000''B |
| | A ¬ C | '100111'B |
| | C \| B | '111111'B |
| | A \| (¬C) | '011111'B |
| | ¬((¬C)\|(¬B)) | '110111'B |
| | SUBSTR(A,1,1)\|(D=5) | '1'B |

### BOOL 組み込み関数

演算子 ¬、&、および | を使用して指定する *not*、*exclusive-or*、*and*、および *or* の各演算に加えて、BOOL 組み込み関数を使用してブール演算を実行することもできます。

**関連情報**

BOOL 組み込み関数
BOOL は、*x* および *y* に対するブール演算 *z* の結果であるビット・ストリングを返します。 結果の長さは、*x* または *y* いずれか長い方のオペランドと等しくなります。

## 比較演算

比較演算は、1 つの挿入演算子を持つ結合オペランドによって指定します。

比較演算では以下の挿入演算子を使用できます。

- <
- ¬<

- <=
- =
- ¬= または <>
- >=
- >
- ¬>

比較演算の結果は、常に長さ 1 のビット・ストリングになります。 値は、関係が true の場合は '1'B、関係が false の場合は '0'B です。

比較は以下のように定義されます。

**代数**
コード化算術フォーマットの符号付き算術値を比較します。 2 つのオペランドが基数、スケール、精度、またはモードで異なっている場合は、 算術演算変換に似た方法で変換されます。 数字データは、比較される前にコード化算術フォーマットに変換されます。 複素数であるオペランドの比較には、演算子 =、¬=、および <> のみが有効です。

**文字**
左から右へ文字単位で文字 (バイト単位で 2 進数値) を比較します。

**ビット**
左から右へ 1 ビットずつ、2 進数を比較します。

**Graphic**
左から右へ DBCS 文字を 1 文字 (2 バイト) ずつ比較します。 比較は、DBCS 文字の 2 進数値に基づいて行われます。

**uchar**
左から右へ、UTF-8 文字の 2 進数値に基づいて uchar を 1 文字ずつ比較します。

**ワイド文字**
左から右へ、2 バイトの対の 2 進数値に基づいて ワイド文字を 1 文字ずつ比較します。

**序数データ**
関係演算子を使用して、同じタイプの序数を比較します。

**ポインターおよびオフセット・データ**
関連する任意の演算子が入っているポインター値とオフセット値を比較します。 ただし、変換を行うことができるのは、オフセットからポインターへの場合です。

**プログラム制御データ**
内部コード化フォーマットのオペランドを比較します。 比較演算子 =、¬=、および <> のみが使用できます。領域変数は比較できません。 タイプ変換は起こりません。プログラム制御データの比較では、オペランド間の タイプの相違はすべてエラーになります。

以下のオペランドについては、比較は等しいという結果になります。

**記入**
比較演算では、非アクティブ・ブロックのエントリー・ポイントを値として持つ入り口変数を指定しても、エラーにはなりません。 同じ PROCEDURE または ENTRY ステートメントの入り口名は、等しいものを比較することはできません。

**Format**
同じ FORMAT ステートメントに関するフォーマット・ラベルの比較が等しい場合。

**ファイル**
2 つのオペランドがファイル値を表しており、ファイル値の各部分ごとに 等しい場合。

**ラベル**
同じステートメントに付いている 2 つのラベルを比較すると、 等しいという結果になります。 比較演算では、これ以上活動しないブロック内のラベル定数を値として持つラベル変数を指定しても、エラーにはなりません。

複合ステートメントのラベルを、その複合ステートメントの本体に含まれているラベルと比較した場合、等しいという結果にはなりません。

計算データの比較で両オペランドのデータ・タイプが別々の比較タイプである 場合には、比較タイプに関して優先度の低い方のオペランドが、もう一方の オペランドの比較タイプに合うように変換されます。 比較タイプの優先順位は、(1) 代数比較 (最も高い)、(2) uchar 比較、(3) ワイド文字比較、(4) グラフィック比較、(5) 文字比較、(6) ビット比較の順です。 例えば、ビット・ストリングと固定小数点値を比較する場合には、10 進数値との代数比較のためにビット・ストリングが 2 進固定小数点フォーマットに変換されます。 さらに、10 進数値も 2 進固定小数点フォーマットに変換されます。

長さの異なる 2 つのストリングの比較では、短い方のストリングの右側に埋め込まれます。 この埋め込みは以下から構成されます。

- 文字比較のときのブランク
- ビット比較のときの '0'B
- グラフィック比較のときのグラフィック (DBCS) ブランク
- uchar 比較のときの uchar ブランク ('20'ux)
- ワイド文字比較のときのワイド文字ブランク ('0020'wx)

以下に、IF ステートメントでの比較演算の例を示します。

```
if A = B
    then action-if-true;
    else action-if-false;
```

A = B の式の演算結果は、これが真であれば '1'B となり、偽であれば '0'B となります。

次の代入ステートメントを考えてください。

```
X = A <= B;
```

A が B より小さければ、'1'B という値が X に割り当てられます。それ以外の場合は、'0'B という値が割り当てられます。

次の代入ステートメントを考えてください。

```
X = A = B;
```

最初の等号は代入記号です。2 番目の等号は比較演算子です。 A が B と等しければ、'1'B という値が X に割り当てられます。等しくない場合は、'0'B という値が割り当てられます。

演算式での比較の例を以下に示します。

```
(X<0)*A + (0<=X & X<=100)*B + (100<X)*C
```

この式の値は X の値に従って、A、B、 または C のいずれかになります。

## 連結演算

連結とは、左側のオペランドの最後の文字、ビット、グラフィック、uchar、またはワイド文字のすぐあとに (間に何もはさまずに)、右側のオペランドの最初の文字、ビット、グラフィック、uchar、またはワイド文字が続く形で、2 つのオペランドを結合することを表します。

連結演算を指定するには、連結挿入文字でオペランド同士を結び付けます。

```
||
```

連結はストリング (文字、ビット、グラフィック、uchar、またはワイド文字) にしか行えないので、連結演算子を使用すると、ストリング・タイプへの変換が行われることがあります。 結果は、RULES コンパイラー・オプションの設定値に応じて異なります。

## RULES(IBM) のもとでの結果

RULES(IBM) を指定する場合、連結演算子は以下のように振る舞います。

- いずれかのオペランドが uchar である場合、結果は uchar になります。
- いずれかのオペランドがワイド文字である場合は、 結果はワイド文字になります。
- いずれかのオペランドがグラフィックである場合は、結果はグラフィックになります。
- いずれかのオペランドがビットまたは 2 進数の場合は、結果はビットになります。
- それ以外の場合は、文字になります。

次の例を参照してください。

```
dcl B bin(4)  initial(4),
    C bit(1)  initial('1'b);
put skip list (B || C);

/*  Produces '01001'  not 'bbb41'  */
```

## RULES(ANS) のもとでの結果

RULES(ANS) を指定する場合、連結演算子は以下のように振る舞います。

- いずれかのオペランドが uchar である場合、結果は uchar になります。
- いずれかのオペランドがワイド文字である場合は、 結果はワイド文字になります。
- いずれかのオペランドがグラフィックである場合は、結果はグラフィックになります。
- 両方のオペランドがビットである場合には、結果はビットになります。
- それ以外の場合は、文字になります。

以下に例を示します。

```
dcl B bin(4)  initial(4),
    C bit(1)  initial('1'b);
put skip list (B || C);

/*  Produces 'bbb41', not '01001'  */
```

連結演算の結果は、2 つのオペランドの長さの合計に等しい長さを持つストリングであり、そのタイプ (文字、ビット、グラフィック、uchar、またはワイド文字) は、2 つのオペランドのタイプと同じです。

連結のためにオペランドの変換が必要であった場合は、オペランド変換後のストリングの長さによって結果の長さが決まります。

| オペランドと値 | 演算 | 結果 |
|---|---|---|
| A = '010111'B<br>B = '101'B<br>C = 'xy,Z'<br>D = 'aa/BB' | A \|\| B | '010111_101'B |
| | A \|\| A \|\| B | '010111_010111_101'B |
| | C \|\| D | 'xy,Zaa/BB' |
| | D \|\| C | 'aa/BBxy,Z' |
| | B \|\| D | '101aa/BB' |

上記の最後の例では、ビット・ストリング '101'B が文字ストリング '101' に変換されてから、連結が行われます。 結果は、文字ストリングになります。

## 各種演算の組み合わせ

1 つの演算式の中で、各種演算を組み合わせることができます。 どのような組み合わせでもかまいません。

次の例を考えてください。

```
declare Result bit(3),
  A fixed decimal(1),
  B fixed binary (3),
  C character(2), D bit(4);
Result = A + B < C & D;
```

この式の中の計算は、それぞれの計算の規則に従って実行されますが、 その前に、下記のように必要とするデータ変換が行われます。

- A の 10 進数値が 2 進数値に変換されます。
- A と B を加算する 2 進数加算が行われます。
- この 2 進数の結果が、C の変換された 2 進数値と比較されます。
- ビット・ストリングで表された比較結果が、ビット変数 D の 長さまで広げられてから、& 演算が行われます。
- & 演算の結果 (長さ 4 のビット・ストリング) は、 変換されずに、右側で切り捨てられて、Result に割り当てられます。

この例の式は、左から右へ順に計算を行うように記述されています。 ただし、計算の順序は、式の中に書かれている演算子の優先順位によって決まります。

**関連情報**

68 ページの『演算子の優先順位』
式の評価において、演算子はさまざまな優先順位を持ちます。

## 演算子の優先順位

式の評価において、演算子はさまざまな優先順位を持ちます。

68 ページの表 34 に、式の評価における演算子の優先順位を示します。

表 *34.* 演算の優先順位と変換の手引き

| 優先順位 | 演算子 | 演算のタイプ | 注釈 |
|---|---|---|---|
| 1 | ** | 算術 | 結果はコード化算術フォーマットです。 |
| | 接頭部 +、- | 算術 | オペランドがコード化算術フォーマットであれば、変換は不要です。 |
| | | | オペランドが CHARACTER ストリングまたは数字 (PICTURE) 表記の固定小数点 10 進数であれば、FIXED DECIMAL に変換されます。 |
| | | | オペランドが数字 (PICTURE) 表記の 10 進浮動小数点数であれば、FLOAT DECIMAL に変換されます。 |
| | | | オペランドが BIT ストリングであれば、FIXED BINARY に変換されます。 |
| | 接頭演算子 ¬ | ビット・ストリング | BIT 以外のデータはすべて BIT に変換されます。 |
| 2 | *,/ | 算術 | 結果はコード化算術フォーマットです。 |
| 3 | 挿入演算子 +、- | 算術 | 結果はコード化算術フォーマットです。 |

表 *34.* 演算の優先順位と変換の手引き *(*続き*)*

| 優先順位 | 演算子 | 演算のタイプ | 注釈 |
|---|---|---|---|
| 4 | ∥ | 連結 | 67 ページの『RULES(ANS) のもとでの結果』 及び 67 ページの『RULES(IBM) のもとでの結果』 を参照してください。 |
| 5 | <、¬<、<=、=、¬= または <>、>=、>、¬> | 比較 | 結果は常に '1'B または '0'B です。 |
| 6 | & | ビット・ストリング | BIT 以外のデータはすべて BIT に変換されます。 |
| 7 | │ | ビット・ストリング | BIT 以外のデータはすべて BIT に変換されます。 |
| | 挿入演算子 ¬ | ビット・ストリング | BIT 以外のデータはすべて BIT に変換されます。 |

表 *34.* 演算の優先順位と変換の手引き

**注 :**

1. 優先順位が高いものから順に演算子を並べてあります。グループ 1 の 優先順位が一番高く、グループ 7 が一番低くなっています。 同じ優先順位グループに属する演算子の優先順位はすべて同じです。 例えば、累乗演算子 ** の優先順位は、接頭演算子 + と -、 および *NOT* 演算子 ¬ と同じです。
2. 優先順位グループ 1 内の演算子が式の中で複数使われているときは、その式の中での優先順位は右から左への順になります。すなわち最右端の指数または接頭演算子が最高の優先順位を持ち、その隣が次の優先順位を持ち、というようになります。 その他の優先順位グループの場合は、同じ優先順位グループ内の 演算子が式の中で複数使われているときは、その式の中での計算順序 つまり優先順位は、左から右への順になります。

式 `A + B < C & D` の評価の順序は、式の要素が `(((A + B) < C) & D)` のように括弧で囲まれている場合と同じです。

式の計算順序 (したがって、その結果も) は、括弧を使用して変えることができます。 括弧で囲まれている式が最初に計算されて 1 つの値が求められ、次にその前後にある演算子に関係づけられています。

例えば、上記の式を次のように変えることができます。

```
(A + B) < (C & D)
```

A の値が固定小数点 2 進数に変換されてから、 加算が行われ、固定小数点 2 進数の結果 (result_1) が生成されます。 C の値がビット・ストリングに変換され、 (このように変換できる場合) *AND* 演算が行われます。 この時点で、式は `Result_1 < Result_2` に縮小されます。

`Result_2` は 2 進数に変換され、代数比較が実行され、式全体の長さ 1 のビット・ストリングが生成されます。

演算子の優先順位は、オペランド間 (またはサブオペランド間) の関係で決められます。 次の例を考えてください。

```
A + (B < C) & (D || E ** F)
```

この場合、PL/I では、 累乗演算を連結より先に実行するということだけが定められています。 `(D|| E ** F)` の計算と他方のオペランド `(A + (B < C))` の計算との順序関係は定められていません。

どのような演算式 (接頭演算子の式を除く) も、最終的には、単一の挿入演算子演算に変形されるものでなければなりません。 その最終的な演算のオペランドと演算子によって、式全体の計算結果の属性が決まります。 次の例では、& 演算子は、最終段階の挿入演算子演算の演算子です。

```
A + B < C & D
```

計算結果は長さ 4 のビット・ストリングになります。

次の例では、括弧で囲まれているため、 最終段階の挿入演算子演算の演算子が比較演算子になります。

```
(A + B) < (C & D)
```

計算結果は長さ 1 のビット・ストリングになります。

# 配列式

配列式では、演算子 (接頭演算子と挿入演算子の両方とも)、要素変数、および定数も使うことができます。 演算の組み合わせの規則やオペランドのデータ変換に関する規則は、エレメント演算の場合と同じです。

配列式は以下として使用できます。

- 割り当てまたは複数の割り当てにおけるソース
- ALL、ANY、POLY、PROD または SUM といった組み込み関数への引数
- (関連パラメーターが、長さが不明なストリングでない場合は) ユーザー・プロシージャーおよびユーザー関数への引数
- PUT LIST ステートメントおよび PUT EDIT ステートメントのデータ・リストの項目

配列式を計算すると、結果は配列になります。 配列を対象とする演算はすべて、行を主体としてエレメントごとに行われます。 したがって、1 つの配列式で参照するすべての配列は、次元の数が同じでなければならず、しかも各次元ごとに境界が同じでなければなりません。

## 接頭演算子と配列

配列に接頭演算子演算を行うと、同じ境界を持つ配列になります。 この配列の各エレメントは、もとの配列の各エレメントに行われた演算の結果です。

**例**

```
If A is the array      5   3  -9
                       1   2   7
                       6   3  -4

then -A is the array  -5  -3   9
                      -1  -2  -7
                      -6  -3   4
```

## 挿入演算子と配列

挿入演算子演算の一方のオペランドが配列変数のときは、 他方のオペランドはエレメント、別の配列でもかまいません。

### 配列とエレメントの演算

エレメントと配列が挿入演算子で結ばれている式の結果は、もとの配列と同じ境界を持つ配列になります。

この配列の各エレメントは、元の配列内の対応するエレメントと単一エレメントとの間で行われた演算の結果になります。 次の例を参照してください。

```
If A is the array         5  10   8
                         12  11   3

then A*3 is the array    15  30  24
                         36  33   9

and 9 > A is the array of   1   0   1
bit strings of length 1     0   0   1
```

配列とエレメントの間で演算を行う場合、そのエレメントは、 その配列内のエレメントであってもかまいません。 次の代入ステートメントを考えてください。

```
A = A * A(1,2);
```

再度 A に上記の値を使用すると、新たに A に代入される値は以下のようになります。

```
    50    100   800
  1200   1100   300
```

つまり、A(1,2) の値が再び取り出されます。

### 配列同士の演算

挿入演算子演算の 2 つのオペランドが共に配列である場合は、この 2 つの配列の次元の数は同じでなければならず、しかも対応する次元ごとに下限と上限が同じでなければなりません。

演算結果は、元の 2 つの配列と同じ限界を持つ配列になります。演算は、元の 2 つの配列の対応するエレメント間で行われます。

#### 例

```
If A is the array          2   4   3
                           6   1   7
                           4   8   2
```

```
and if B is the array      1   5   7
                           8   3   4
                           6   3   1
```

```
then A+B is the array      3   9  10
                          14   4  11
                          10  11   3
```

```
and A*B is the array       2  20  21
                          48   3  28
                          24  24   2
```

```
and A>B is the array of    1   0   0
bit strings of length 1    0   0   1
                           0   1   1
```

## 構造式

構造式は、構造体参照とは異なり、関連するパラメーターに定数エクステント (つまり、一定のストリング長、区域サイズ、および配列境界) がある限りは、割り当てを行うときと、プロシージャーまたは関数への引数としてのみ使用できます。

構造式に現れる構造体変数はすべて、以下のように同一の構造でなければなりません。

- 構造体は、小構造化がすべて同じで、中に含まれるエレメントと配列の数が同じでなければなりません。
- 構造体内 (および小構造があれば、その内部で) のエレメントや配列の位置指定は、 同じでなければなりません。
- 対応する位置にある配列は、同じ境界を持たなければなりません。

## 制限付き式

PL/I に (おそらく、符号付きの) 定数が必要なときには、制限付き式 を使用することができます。

制限付き式の値はコンパイル時に計算され、定数として使用されます。 例えば、式を使用して、以下に必要な定数を定義することができます。

- 静的、パラメーターおよび基本宣言のエクステント
- 入り口記述のエクステント
- 静的初期設定で使用される値および反復因数

制限付き式は通常の式と同等ですが、各オペランドは以下のいずれかでなければなりません。

- 定数または名前付き定数

  名前付き定数は、使用する前に宣言する必要があります
- 制限付き式に適用される (以下のカテゴリーからの) 組み込み関数
  - ストリング処理
  - 算術計算 (RANDOM を除く。すべての引数は REAL でなければなりません)
  - 数学 (すべての引数が REAL の場合)
  - 浮動小数点照会
  - 浮動小数点操作
  - 整数操作
  - 精度処理
  - 配列処理関数: DIMACROSS、 DIMENSION、 HBOUND、 HBOUNDACROSS、 LBOUND、 および LBOUNDACROSS
  - ストレージ制御関数: BINARYVALUE、 LENGTH、 NULL、 OFFSETVALUE、 POINTERVALUE、 SIZE、 STORAGE、 および SYSNULL
  - その他の関数: BYTE、 CHARVAL、 COLLATE、 INDICATORS、 PACKAGENAME、 POPCNT、 PROCEDURENAME、 RANK、 SOURCEFILE、 SOURCELINE、 および WCHARVAL
- タイプ付き関数: BIND、CAST、FIRST、LAST、RESPEC、SIZE、および VALUE

**例**

```
dcl Max_names fixed bin value (1000),
    Name_size fixed bin value (30),
    Addr_size fixed bin value (20),
    Addr_lines fixed bin value (4);
dcl 1 Name_addr(Max_names),
      2 Name char(Name_size),
      2 * union,
        3 Address char(Addr_lines*Addr_size), /* address    */
        3 addr(Addr_lines) char(Addr_size),
      2 * char(0);
dcl One_Name_addr char(size(Name_addr(1)));   /* 1 name/addr*/
dcl Two_Name_addr char(length(One_Name_addr)
                       *2);                /* 2 name/addrs  */
dcl Name_or_addr char(max(Name_size,Addr_size)) based;

dcl Ar(10) pointer;
dcl Ex     entry( dim(lbound(Ar):hbound(Ar)) pointer);
dcl Identical_to_Ar( lbound(Ar):hbound(Ar) ) pointer;
```

この例で、名前付き定数のいずれかの値を変更すると、従属宣言はすべて自動的に再計算されます。

# 第 4 章 データ変換

この章では、計算データのデータ変換について説明します。 PL/I は、特定の属性セットを持つデータ項目が、 別の属性セットを持つデータ項目に割り当てられると、 データ変換を行います。

計算データ項目の値が変換されると、その内部表記、精度 (算術値の場合)、 モード (算術値の場合)、または長さ (ストリング値の場合) が変わることがあります。 下の表は、どのような場合にほかの属性に変換が行われるかをまとめたものです。

| ケース | ターゲットの属性 |
|---|---|
| 割り当て | 代入記号の左側にある変数の属性 |
| 式のオペランド | 式を計算するときの規則によって決まる |
| ストリームの入力 (GET ステートメント) | 受け入れフィールドの属性 |
| ストリームの出力 (PUT ステートメント) | ストリームが編集ディレクティブであれば、フォーマット・リストによって決まり、 それ以外の場合は文字ストリング |
| PROCEDURE または ENTRY への引数 | 対応するパラメーターの属性 |
| 組み込み関数または疑似変数への引数 | 関数または疑似変数によって決まる |
| INITIAL 属性 | 初期設定される変数のほかの属性 |
| RETURN ステートメントの式 | PROCEDURE ステートメントで指定されている属性 |
| DO ステートメントのオプション BY、TO、または REPEAT | 制御変数の属性 |

以下のステートメントによって、CHARACTER 値への変換が行われる場合があります。

| ステートメント | オプション |
|---|---|
| DISPLAY | |
| レコード入出力 | KEYFROMKEY |
| OPEN | TITLE |

以下のステートメントによって、BINARY 値への変換が行われる場合があります。

| ステートメント | オプション/属性/参照 |
|---|---|
| DECLARE、ALLOCATE、DEFAULT | 長さ、サイズ、次元、境界、反復因数 |
| DELAY | ミリ秒 |
| FORMAT (および GET や PUT 内のフォーマット項目) | 反復因数、w、d、s、p |
| OPEN | LINESIZE、PAGESIZE |
| I/O | SKIP、LINE、IGNORE |
| ほとんどのステートメント | 添え字 |

ソース・データ項目とターゲット・データ項目のすべての属性 (ストリングの長さは除く) が、 コンパイル時に指定されていなければなりません。 変換は以下の条件の 1 つを起こすことがあります。 CONVERSION、OVERFLOW、SIZE、または STRINGSIZE。 (347 ページの『第 16 章 条件』 を参照してください。)

© Copyright IBM Corp. 1999, 2022

定数は実行時だけでなくコンパイル時に変換されることもあります。 いずれの場合も、変換はこのトピックで説明されているとおりに変換が行われます。

変換に関する説明では、*M* および *N* の意味に注意してください。

- *M* は FIXED BINARY の最大精度です。 これは、コンパイラー・オプション LIMITS(FIXEDBIN(M1,M2)) からの値 M2 です。
- *N* は FIXED DECIMAL の最大精度です。 これは、コンパイラー・オプション LIMITS(FIXEDDEC(N1,N2)) からの値 N2 です。

1 つの演算で変換が複数回必要になることがあります。 ただし、必ずしも複数回行われるとは限りません。 変換規則を理解するには、これらの演算を別々に考えることをお勧めします。 次の例を考えてください。

```
dcl A fixed dec(3,2) init(1.23);
dcl B fixed bin(15,5);
B = A;
```

この例では、1.23 という 10 進数表記は、まず BINARY (11,7) 値に変換されて 1.0011101B となります。 次に、精度の変換が行われて、BINARY(15,5) の 1.00111B という値になります。

**関連情報**

245 ページの『ロケーターの変換』
少数の例を除き、ロケーター・データは他のデータ・タイプに変換できません。

# 計算データ変換の組み込み関数

式の計算の途中、入出力 GET や PUT 演算、代入演算、および引数とパラメーター間で変換を行うことができます。

さらに、以下の組み込み関数を使用すれば、変換を開始させることもできます。

| | | |
|---|---|---|
| BINARY | 固定 | REAL |
| ビット | FLOAT | SIGNED |
| CHAR | GRAPHIC | UNSIGNED |
| COMPLEX | IMAG | WIDECHAR |
| DECIMAL | PRECISION | |

それぞれの関数は、必要な変換を行って、 その関数名で指定される属性を持つ値を戻します。

組み込み関数 COMPLEX、GRAPHIC、および IMAG によって行われる変換は 例外ですが、必要な属性を持つ PL/I 変数に割り当てを行っても、 上記の組み込み関数で行われる変換と同じ結果を得ることができます。 しかし、組み込み関数を使用するほうが、変換を実行するために変数を 作成するよりも容易であることは明らかです。

**関連情報**

373 ページの『組み込み関数、疑似変数、およびサブルーチン』
多くの共通タスクが組み込み関数、サブルーチン、および疑似変数の形で使用可能です。 これらの共通タスクを使用することにより、作成するコードの数が減り、信頼性の高い迅速な処理が できます。 この章では、PL/I プログラムで使用できる組み込み関数、サブルーチン、および疑似変数について説明します。

# ストリングの長さの変換

ソース・ストリングは、ターゲット・ストリングの左から右へ割り当てられます。 ソース・ストリングがターゲット・ストリングよりも長い場合、右側の超過した文字、ビット、グラフィック、uchar、またはワイド文字は無視され、STRINGSIZE 条件が起こります。 ターゲット・ストリングが固定長で、ソース・ストリングよりも長いときは、ターゲット・ストリングの右側が埋め込まれます。 STRINGSIZE が割り込み禁止になっており、 ソース・ストリングやターゲット・ストリング (またはこれらの両方) の長さが 実行時に判別され、しかもターゲット・ストリングが短すぎてソース・ストリングが入りきらない場合には、 予測できない結果が発生することがあります。

**注 :** 変数でパラメーターとして SUBTO/SUBSTR を使用し、その変数がターゲットに含まれていないストリングを指定すると、STRINGRANGE 条件が有効になっていない場合に予測不能な結果が生じる可能性があります。

文字ストリングにはブランクが埋め込まれ、ビット・ストリングには '0'B、漢字ストリングには DBCS ブランク、uchar ストリングには uchar ブランク、ワイド文字ストリングにはワイド文字ブランクがそれぞれ埋め込まれます。

```
declare Subject char(10);
Subject = 'Transformations';
```

「Transformations」は 15 文字であるため、PL/I がストリングを Subject に割り当てると、ストリングの右端から 5 文字が切り捨てられます。 これは以下のステートメントを実行することと同等です。

```
Subject = 'Transforma';
```

下に示す最初の 2 つのステートメントは、 同じ値を Subject に割り当てます。 また、最後の 2 つのステートメントは、 同じ値を Code に割り当てます。

```
Subject = 'Physics';
Subject = 'Physics   ';
declare Code bit(10);
Code = '110011'B;
Code = '1100110000'B;
```

次の 2 つのステートメントは、Subject に 同等の値を割り当てません。

```
Subject = '110011'B;
Subject = '1100110000'B;
```

最初のステートメントが実行されると、右側のビット定数がまず文字ストリングに 変換されてから、ビットのゼロではなくブランク文字が右側に埋め込まれます。 このステートメントは次のステートメントと同等です。

```
Subject = '110011bbbb';
```

2 番目のステートメントはビットから文字タイプへの変換のみを必要とし、次のステートメントと同等です。

```
Subject = '1100110000';
```

ストリング値が、VARYING 属性または VARYING4 属性を持つストリング変数に割り当てられるときは、ブランク文字またはゼロのビットによる埋め込みは行われません。 代わりに、ターゲット・ストリング変数の長さが、割り当てられたストリングの長さにセットされます。 ただし、割り当てられたストリングが、可変長ストリング変数で宣言されている 最大の長さより長いときは、切り捨てが行われます。

## 算術値の精度の変換

算術値のデータ属性がターゲットのデータ属性 (精度は除く) と同じであれば、 精度の変換が必要になります。

固定小数点データ項目の場合は、10 進数も 2 進数も小数点で位置合わせされて精度が変換されます。 したがって、左側または右側で埋め込みや切り捨てが行われることがあります。 左側 (最高位桁) でゼロ以外の 2 進数または 10 進数が失われると、SIZE 条件が起こります。

浮動小数点データ項目の場合は、右側 (最低位桁) で、切り捨てまたはゼロによる埋め込みが行われることがあります。

## モードの変換

複合値が実数値に変換される場合は、虚数部分が無視されます。 実数値が複合値に変換される場合は、虚数部分がゼロになります。

# その他のデータ属性の変換

10 進数表記と 2 進数表記の間で変換が行われると、 値が変わることがあります。

下記のデータ属性を持つデータ項目を ソースからターゲットへ変換する場合の規則について、以下に説明します。

- コード化算術フォーマット

  FIXED BINARY
  FIXED DECIMAL
  FLOAT BINARY
  FLOAT DECIMAL
- 算術文字 PICTURE
- CHARACTER
- ビット
- GRAPHIC
- UCHAR
- WIDECHAR

2 進数と 10 進数の間の変換では、係数 3.32 が次のように使用されます。

- n 桁の 10 進数は、CEIL (n*3.32) 桁の 2 進数に変換されます。
- n 桁の 2 進数は、CEIL (n/3.32) 桁の 10 進数に変換されます。

76 ページの表 35 に、これらの変換を計算する CEIL 値を示します。

表 *35. CEIL (n*3.32)* の値と *CEIL (n/3.32)* の値の表

| n | CEIL (n*3.32) | n | CEIL (n/3.32) |
|---|---|---|---|
| 1 | 4 | 1-3 | 1 |
| 2 | 7 | 4-6 | 2 |
| 3 | 10 | 7-9 | 3 |
| 4 | 14 | 10-13 | 4 |
| 5 | 17 | 14-16 | 5 |
| 6 | 20 | 17-19 | 6 |
| 7 | 24 | 20-23 | 7 |
| 8 | 27 | 24-26 | 8 |
| 9 | 30 | 27-29 | 9 |
| 10 | 34 | 30-33 | 10 |
| 11 | 37 | 34-36 | 11 |
| 12 | 40 | 37-39 | 12 |
| 13 | 44 | 40-43 | 13 |
| 14 | 47 | 44-46 | 14 |
| 15 | 50 | 47-49 | 15 |
| 16 | 53[1] | 50-53 | 16 |
| 17 | 57 ヶ所 | 54-56 | 17 |

表 35. CEIL (n*3.32) の値と CEIL (n/3.32) の値の表 (続き)

| n | CEIL (n*3.32) | n | CEIL (n/3.32) |
|---|---|---|---|
| 18 | 60 | 57-59 | 18 |
| 19 | 64 | 60-63 | 19 |
| 20 | 67 | 64-66 | 20 |
| 21 | 70 | 67-69 | 21 |
| 22 | 74 | 70-73 | 22 |
| 23 | 77 | 74-76 | 23 |
| 24 | 80 | 77-79 | 24 |
| 25 | 83 | 80-83 | 25 |
| 26 | 87 | 84-86 | 26 |
| 27 | 90 | 87-89 | 27 |
| 28 | 93 | 90-92 | 28 |
| 29 日 | 97 | 93-96 | 29 日 |
| 30 | 100 | 97-99 | 30 |
| 31 | 103 | 100-102 | 31 |
| 32 | 107 | 103-106 | 32 |
| 33 | 110 | 107-109 | 33 |
| | | 110-112 | 34 |
| | | 113-116 | 35 |

**注 1 :** ceil(16*3.32) = 54 のときは、53 という値が使用されます。 そうでない場合には、浮動小数点 10 進数 (16) が 2 進数に変換され、(浮動小数点 2 進数 (54) が拡張浮動小数点として表示されるので) 長精度の 浮動小数点から拡張精度の浮動小数点に変換されることになります。

固定小数点整数値の場合は、変換しても値は変わりません。 小数部分を持つ固定小数点の値の場合は、因数 3.32 によって必要なだけの桁数またはビット数が用意され、変換後の値ともとの値の違いは最低位の 1 桁 (1 ビット) 未満に抑えられます。

例えば、属性が FIXED DECIMAL (1,1) である 10 進定数 .1 は、.0001B という 2 進数値に 変換され、したがって 1/10 は 1/16 に変換されます。 属性が FIXED DECIMAL (2,2) である 10 進定数 .10 は、.0001100B という 2 進数値に 変換され、したがって 10/100 は 12/128 に変換されます。

## ソースからターゲットへの変換規則

以下のデータ属性を持つデータ項目には、以下のソースからターゲットへの変換規則が適用されます。

- コード化算術
  - FIXED BINARY
  - FIXED DECIMAL
  - FLOAT BINARY
  - FLOAT DECIMAL
- 算術文字 PICTURE
- CHARACTER

- ビット
- GRAPHIC
- UCHAR
- WIDECHAR

## ターゲット: コード化算術

**ソース:**

**FIXED BINARY、FIXED DECIMAL、**
**FLOAT BINARY、および FLOAT DECIMAL**
これらはすべてコード化算術データです。 コード化算術データ間での変換の規則については、ターゲットがこれらの各データ・タイプである場合に、その説明箇所で述べています。

**算術文字 PICTURE**
データは、まず 10 進数に変換されます。そのスケールと精度は、対応する PICTURE 指定によって決まります。 次に、この 10 進値がターゲットの基数、スケール、モード、および精度に変換されます。 FIXED DECIMAL または FLOAT DECIMAL をソースとして使用しているコード化算術データの特定のターゲット・タイプの説明を参照してください。

**CHARACTER**
ソース・ストリングは有効な算術定数または複素数式でなければなりません。それ以外の場合は、CONVERSION 条件が生じます。 定数の頭に符号が付いていてもよく、定数の前後にブランクがあってもかまいません。 符号と定数の間には、ブランクがあってはなりません。また、複素数式では、実数部分の終わりと虚数部分の先頭の符号との間にブランクがあってはなりません。

定数は、基数、スケール、モード、および精度の属性を持ちます。 割り当ての場合のように、ターゲットの属性がソースの属性と無関係であるときは、定数はターゲットの属性に変換されます。 ソースとして定数の属性を使用しているコード化算術データの特定のターゲット・タイプの説明を参照してください。

演算式の計算の場合のように中間結果が必要なときは、中間結果の属性は、もとのストリングの代わりに精度 (N,0) の 10 進固定小数点値が使われている場合の中間結果の属性と同じです。 (このため、コンパイラーは、定数の属性には関係なく、あらゆる場合に対処できるコードを生成することができます。) この結果、定数の小数部分が失われることがあります。 ソースとして FIXED DECIMAL を使用しているコード化算術データの特定のターゲット・タイプの説明を参照してください。

文字データ項目から中間の 10 進固定小数点への最初の変換では、 値が中間結果のデフォルト・サイズを超える可能性があります。 超える場合、使用可能になっていれば、SIZE 条件が起こります。

複素数を表す文字ストリングが実数のターゲットに割り当てられる場合、ストリングの実数部分だけがターゲットに割り当てられるので、そのストリングの虚数部分が正しい算術文字であるかどうかの検査は行われず、CONVERSION 条件も起こりません。

ソースがヌル・ストリングまたは 1 つ以上のブランクから成るストリングである場合、ターゲットにはゼロの値が割り当てられます。 CONVERSION 条件が not 引き起こされます。

**ビット**
演算式の計算時に変換が起こる場合は、 ソースのビット・ストリングは FIXED BINARY(M,0) の符号なし値に変換されます。 FIXED BINARY をソースとして使用しているコード化算術データの特定のターゲット・タイプの説明を参照してください。

ソース・ストリングが、使用可能な精度よりも長いときは、左側のビットが無視されます。 ゼロ以外のビットが失われると、SIZE 条件が起こります。

ヌル・ストリングはゼロの値になります。

**GRAPHIC**
グラフィック変数とストリングが CHARACTER に変換された 後は、<xref href="#str/charsc">『CHARACTER』</xref>で説明 されている文字ソースの規則に従ってください。

**UCHAR**
uchar 変数とストリングが CHARACTER に変換された後は、CHARACTER で説明されている文字ソースの規則に従ってください。

**WIDECHAR**
ワイド文字変数とストリングが CHARACTER に変換された後は、『CHARACTER』で説明されている文字ソースの規則に従ってください。

## ターゲット: FIXED BINARY (p2,q2)

**ソース:**

**FIXED DECIMAL (p1,q1)**
結果の精度は、p2 = min(M,1+CEIL(p1*3.32)) で、q2=CEIL(ABS(q1*3.32))*SIGN(q1) です。

**FLOAT BINARY (p1)**
75 ページの『算術値の精度の変換』で示したように、精度が変換されます。 p1 は宣言または指定されている値で、q1 は、2 進小数点の位置で決められ、 指数の値で変更されます。

**FLOAT DECIMAL (p1)**
FIXED DECIMAL から FIXED BINARY への変換の場合と同じように精度が 変換されます。p1 は宣言または指定されている値で、q1 は 10 進小数点の 位置で決められ、指数の値で変更されます。

**算術文字 PICTURE**
『ターゲット: コード化算術フォーマット』を参照してください。

**CHARACTER**
『ターゲット: コード化算術フォーマット』を参照してください。

**ビット**
『ターゲット: コード化算術フォーマット』を参照してください。

**GRAPHIC**
『ターゲット: コード化算術フォーマット』を参照してください。

**UCHAR**
『ターゲット: コード化算術フォーマット』を参照してください。

**WIDECHAR**
『ターゲット: コード化算術フォーマット』を参照してください。

## ターゲット: FIXED DECIMAL (p2,q2)

**ソース:**

**FIXED BINARY (p1,q1)**
結果の精度は、p2=1+CEIL(p1/3.32) で、q2=CEIL(ABS(q1/3.32))*SIGN(q1) です。

**FLOAT BINARY (p1)**
FIXED BINARY から FIXED DECIMAL への変換の場合と同じように精度が 変換されます。p1 は宣言または指定されている値で、q1 は 2 進小数点の位置で 決められ、指数の値で変更されます。

**FLOAT DECIMAL (p1)**
75 ページの『算術値の精度の変換』で説明したように、精度が変換されます。p1 は 宣言または指定されている値で、q1 は 10 進小数点の位置で決められ、 指数の値で変更されます。

**算術文字 PICTURE**
『ターゲット: コード化算術フォーマット』を参照してください。

**CHARACTER**
『ターゲット: コード化算術フォーマット』を参照してください。

**ビット**
『ターゲット: コード化算術フォーマット』を参照してください。

**GRAPHIC**
『ターゲット: コード化算術フォーマット』を参照してください。

**UCHAR**
『ターゲット: コード化算術フォーマット』を参照してください。

**WIDECHAR**
『ターゲット: コード化算術フォーマット』を参照してください。

## ターゲット: FLOAT BINARY (p2)

**ソース:**

**FIXED BINARY (p1,q1)**
結果の精度は、p2=p1 です。 指数は、値の小数部分を表します。

**FIXED DECIMAL (p1,q1)**
結果の精度は、p2=CEIL(p1*3.32) です。 指数は、値の小数部分を表します。

**FLOAT DECIMAL (p1)**
結果の精度は、p2=CEIL(p1*3.32) です。

**算術文字 PICTURE**
『ターゲット: コード化算術フォーマット』を参照してください。

**文字**
『ターゲット: コード化算術フォーマット』を参照してください。

**ビット**
『ターゲット: コード化算術フォーマット』を参照してください。

**GRAPHIC**
『ターゲット: コード化算術フォーマット』を参照してください。

**UCHAR**
『ターゲット: コード化算術フォーマット』を参照してください。

**WIDECHAR**
『ターゲット: コード化算術フォーマット』を参照してください。

## ターゲット: FLOAT DECIMAL (p2)

**ソース:**

**FIXED BINARY (p1,q1)**
結果の精度は、p2=CEIL(p1/3.32) です。 指数は、値の小数部分を表します。

**FIXED DECIMAL (p1,q1)**
結果の精度は、p2=p1 です。 指数は、値の小数部分を表します。

**FLOAT BINARY (p1)**
結果の精度は、p2=CEIL(p1/3.32) です。

**算術文字 PICTURE**
『ターゲット: コード化算術フォーマット』を参照してください。

**文字**
『ターゲット: コード化算術フォーマット』を参照してください。

**ビット**
『ターゲット: コード化算術フォーマット』を参照してください。

**GRAPHIC**
『ターゲット: コード化算術フォーマット』を参照してください。

**UCHAR**
『ターゲット: コード化算術フォーマット』を参照してください。

**WIDECHAR**
『ターゲット: コード化算術フォーマット』を参照してください。

## ターゲット: 算術文字 PICTURE

算術文字 PICTURE データ項目は、10 進固定小数点 または 10 進浮動小数点の値を文字表記で表したものです。 ソースから算術文字 PICTURE への変換に関する以下の説明では、 左端または右端の桁を失わずに割り当てを行うのに 必要なターゲットの属性を示してあります。

**ソース:**

**FIXED BINARY (p1,q1)**
ターゲットが次の条件を満たしている必要があります。

```
fixed decimal (1+x+q-y,q) or
float decimal (x)
```

ここで、x>=CEIL(p1/3.32)、y=CEIL(q1/3.32)、および q>=y です。

**FIXED DECIMAL (p1,q1)**
ターゲットが次の条件を満たしている必要があります。

```
fixed decimal (x+q-q1,q) or
float decimal (x)
```

ここで、x>=p1 および q>=q1 です。

**FLOAT BINARY (p1)**
ターゲットが次の条件を満たしている必要があります。

```
fixed decimal (p,q) or
float decimal (p)
```

ここで、p>=CEIL(p1/3.32) であり、p と q の値は、ソースの指数によって保持できる値の範囲を考慮します。

**FLOAT DECIMAL (p1)**
ターゲットが次の条件を満たしている必要があります。

```
fixed decimal (p,q) or
float decimal (p)
```

ここで、p>= p1 であり、p および q の値は、ソースの指数によって保持できる値の範囲を考慮します。

**算術文字 PICTURE**
ソースの暗黙定義される属性は、FIXED DECIMAL または FLOAT DECIMAL です。 該当ターゲットに関する項目を参照してください。

**文字**
『ターゲット: コード化算術フォーマット』を参照してください。

**BIT(n)**
ターゲットが次の条件を満たしている必要があります。

```
fixed decimal (1+x+q,q) or
float decimal (x)
```

ここで、x>=ceil(n/3.32) および q>=0 です。

**GRAPHIC**
『ターゲット: コード化算術フォーマット』を参照してください。

**UCHAR**
『ターゲット: コード化算術フォーマット』を参照してください。

**WIDECHAR**
『ターゲット: コード化算術フォーマット』を参照してください。

## ターゲット: CHARACTER

**ソース:**

**FIXED BINARY、FIXED DECIMAL、**
**FLOAT BINARY、および FLOAT DECIMAL**

以下に説明するように、コード化算術値は 10 進定数 (負数の場合は頭に負符号 (-) が付けられる) に変換されます。 その定数は、ソースの属性から長さが導出される中間の文字ストリングに入れられます。 次に、この中間のストリングが、ストリングの割り当ての規則に従ってターゲットに割り当てられます。

コード化算術値から文字ストリングへの変換の規則は、リスト・ディレクティブ出力、データ・ディレクティブ出力、およびキーの評価 (REGIONAL ファイルの場合も) でも使用されます。

**FIXED BINARY (p1,q1)**

まず、2 進数の精度 (p1,q1) が、同等の 10 進数の精度 (p,q) に変換されます。 ただし、p=1+CEIL(p1/3.32) および q=CEIL(ABS(q1/3.32))*SIGN(q1) です。 そのあとの規則は、FIXED DECIMAL から CHARACTER への変換の規則と同じです。

**FIXED DECIMAL (p1,q1)**

p1>=q1>=0 の場合:

- 定数は、幅が p1+3 のフィールドで右寄せされます。 (3 は、負符号、小数点または 2 進小数点、および小数点の前の先行ゼロの可能性を考慮するために必要です。)
- 小数部分の小数点のすぐ前にあるゼロ 1 つを除いて、先行ゼロがブランクで置き換えられます。ソースの値がゼロのときも、ゼロ 1 つが残されます。
- 負数であれば、最初の桁の前に負符号 (-) が付けられます。 正の値には符号は付けられません。
- q1=0 の場合、小数点は表示されません。q1>0 の場合は、小数点が表示され、定数の小数点以下の桁数は q になります。

p1<q1 または q1<0 の場合は、定数の右方に位取り係数が付加されます。 定数は任意指定の符号付き整数です。 項目の値がゼロであっても、位取り係数が付けられ、下記の構文になります。

```
F{+|-}nn
```

ただし、{+|-}nn の値は -q1 です。

中間結果のストリングの長さは p1+k+3 です。 ただし、k は、q1 の値が入りきるのに必要な桁数です (符号と 英字 F は桁数に数えられません)。

算術値が複素数の場合、中間結果のストリングは、虚数部分が実数部分に連結した形になります。左側の実数部分はソースの実数部分として生成され、 右側の虚数部分は、符号と英字 I が必ず付加されます。 このようにして生成されたストリングは、2 つのエレメントの間にブランクをはさんでいない複素数式となります。 この中間結果のストリングの長さは次のとおりです。

```
2*p1+7     for p1>=q1>=0
2*(p1+k)+7 for p1<q1 or q1<0
```

下の例は、実数および複素数の固定小数点 10 進数値から生成される、 中間結果のストリングを示します。

```
Precision     Value     String

(5,0)       2947        'bbbb2947'
(4,1)       -121.7      'b-121.7'
(4,-3)      -3279000    '-3279F+3'
(2,1)       1.2+0.3I    'bbb1.2+0.3I'
```

**FLOAT BINARY (p1)**

まず、浮動小数点 2 進数の精度 (p1) が同等の浮動小数点 10 進数の 精度 (p) に変換されます。ただし、p=CEIL(p1/3.32) です。 そのあとの規則は、FLOAT DECIMAL から CHARACTER への変換の規則と同じです。

**FLOAT DECIMAL (p1)**
ソースが 10 進浮動小数点数のときは、E(w,d,s) という形の E フォーマット項目で 伝送される場合と同様に変換されます。ただし、以下のような条件です。

```
w, the length of the intermediate string, is p1+8.

d, the number of fractional digits, is p1-1.

s, the number of significant digits, is p1.
```

算術値が複素数の場合、中間結果のストリングは、虚数部分が実数部分に連結した形になります。左側の実数部分はソースの実数部分として生成され、右側の虚数部分は、符号と英字 I が必ず付加されます。 このようにして生成されたストリングは、2 つのエレメントの間にブランクをはさんでいない複素数式となります。 中間結果のストリングの長さは 2*p+17 です。

下の例は、実数および複素数の浮動小数点 10 進数値から生成される、 中間結果のストリングを示します。

```
Precision    Value          String

(5)          1735*10**5     'b1.7350E+0008'
(5)          -.001663       '-1.6630E-0003'
(3)          1              'b1.00E+0000'
(5)          17.3+1.5I      'b1.7300E+0001+1.5000E+0000I'
```

**算術文字 PICTURE**
実数の算術文字フィールドは、文字ストリングと解釈され、 文字ストリングの長さを変換する際の規則に従ってターゲット・ストリングに割り当てられます。 複素数の算術文字フィールドの場合は、 実数部分と虚数部分が連結されてから、ターゲット・ストリングに割り当てられます。 そのターゲット・ストリングには挿入文字が含まれます。

**ビット**
ビット 0 は文字 0 になり、ビット 1 は文字 1 になります。 ヌル・ビット・ストリングはヌル文字ストリングになります。 生成された文字ストリングは、ストリングの長さを変換する際の規則に従ってターゲット・ストリングに割り当てられます。

**GRAPHIC**
DBCS が SBCS に変換されるのは、 対応する SBCS 文字がある場合のみです。 そうでない場合には、CONVERSION 条件が起こります。

**UCHAR**
uchar からの変換は、(コンパイラーの CODEPAGE オプションで指定した) ターゲット・コード・ページにすべての uchar の表現がある場合にのみ実行されます。 そうでない場合には、CONVERSION が起こります。

**WIDECHAR**
ワイド文字からの変換は、(コンパイラーの CODEPAGE オプションで指定した) ターゲット・コード・ページにすべてのワイド文字の表現がある場合にのみ実行されます。 そうでない場合には、CONVERSION が起こります。

## ターゲット: BIT

**ソース:**

**FIXED BINARY、FIXED DECIMAL、**
**FLOAT BINARY、および FLOAT DECIMAL**
必要に応じて、算術値は 2 進数に変換され、符号や小数部分は無視されます。 (複素数の算術値の場合は、虚数部分も無視されます。) このようにして生成された 2 進値は、ビット・ストリングと見なされ、 ストリングを割り当てる際の規則に従ってターゲット・ストリングに割り当てられます。

**FIXED BINARY (p1,q1)**
中間結果のビット・ストリングの長さは、次の式で求められます。

```
min(M,(p1-q1))
```

(p1-q1) が負数またはゼロのときは、ヌル・ビット・ストリングが生成されます。

下の例は、固定小数点 2 進数値から生成される、中間結果のストリングを示します。

```
Precision   Value    String

(1)           1      '1'B
(3)          -3      '011'B
(4,2)      1.25      '01'B
```

**FIXED DECIMAL (p1,q1)**
中間結果のビット・ストリングの長さは、次の式で求められます。

```
  min(M,CEIL((p1-q1)*3.32))
```

(p1-q1) が負数またはゼロのときは、ヌル・ビット・ストリングが生成されます。

下の例は、固定小数点 10 進数値から生成される、中間結果のストリングを示します。

```
Precision   Value    String

(1)           1        '0001'B
(2,1)      1.1        '0001'B
```

**FLOAT BINARY (p1)**
中間結果のビット・ストリングの長さは、次の式で求められます。

```
  min(M,p1)
```

**FLOAT DECIMAL (p1)**
中間結果のビット・ストリングの長さは、次の式で求められます。

```
  min(M,ceil(p1*3.32))
```

**算術文字 PICTURE**
まず、データは、算術 PICTURE 指定で決められたスケールと精度を持つ 10 進数として解釈されます。 次に、その項目は、FIXED DECIMAL または FLOAT DECIMAL から BIT への 変換規則に従って変換されます。

**CHARACTER**
文字 0 はビット 0 になり、文字 1 はビット 1 になります。 0 または 1 以外の文字を使用すると、CONVERSION 条件が発生します。 ヌル・ストリングは、ヌル・ビット・ストリングになります。 生成されたビット・ストリング (その長さはソースの文字ストリングの長さと同じ) は、ストリングを割り当てる際の規則に従ってターゲットに割り当てられます。

**GRAPHIC**
グラフィック 0 はビット 0 になり、グラフィック 1 はビット 1 になります。 0 または 1 以外のグラフィックでは、CONVERSION 条件が発生します。 ヌル・ストリングは、ヌル・ビット・ストリングになります。 生成されたビット・ストリング (その長さはソースの 漢字ストリングの長さと同じ) は、 ストリングを割り当てる際の規則に従ってターゲットに割り当てられます。

**UCHAR**
uchar の 0 ('30'ux) はビットの 0 になり、uchar の 1 ('31'ux) はビットの 1 になります。 0 または 1 以外の uchar があると、CONVERSION 条件が起こります。 ヌル・ストリングは、ヌル・ビット・ストリングになります。 生成されたビット・ストリング (その長さはソースの uchar ストリングの長さと同じ) は、ストリングを割り当てる際の規則に従ってターゲットに割り当てられます。

**WIDECHAR**
ワイド文字 0 ('0030'wx) はビット 0 になり、ワイド文字 1 ('0031'wx) はビット 1 になります。 0 または 1 以外のワイド文字があると、CONVERSION 条件が発生します。 ヌル・ストリングは、ヌル・ビット・ストリングになります。 生成されたビット・ストリング (その長さはソースの ワイド文字ストリングの長さと同じ) は、 ストリングを割り当てる際の規則に従ってターゲットに割り当てられます。

### ターゲット: GRAPHIC

非グラフィック・ソースは、最初に、『ターゲット: CHARACTER』で説明した規則に従って文字に変換されます。 そのあとで、その結果の文字ストリングが DBCS ストリングに変換されます。

### ターゲット: UCHAR

uchar およびワイド文字以外のソースは、最初に、ターゲット: CHARACTER で説明した規則に従って文字に変換されます。 そのあとで、その結果の文字ストリングが uchar ストリングに変換されます。 ワイド文字ソースは UTF-16 から UTF-8 に変換され、ターゲットに割り当てられます。

uchar への文字データの変換の結果は、コンパイラーの CODEPAGE オプションの設定値によって異なります。 例えば、CODEPAGE(1140) オプションでは、文字 '59'x と 'A1'x は記号の ß と ~ を表し、uchar の 'DF'ux と '7E'ux にそれぞれ変換されます。 一方、CODEPAGE(1141) オプションでは、これらの文字は ~ と ß を表し、uchar の '7E'ux と 'DF'ux にそれぞれ変換されます。

### ターゲット: WIDECHAR

ワイド文字および uchar 以外のソースは、最初に、ターゲット: CHARACTER で説明した規則に従って文字に変換されます。 そのあとで、その結果の文字ストリングがワイド文字ストリングに変換されます。 uchar ソースは UTF-8 から UTF-16 に変換され、ターゲットに割り当てられます。

ワイド文字への文字データの変換の結果は、コンパイラーの CODEPAGE オプションの設定値によって異なります。 例えば、CODEPAGE(1140) オプションでは、文字 '59'x と 'A1'x は記号の ß と ~ を表し、ワイド文字の '00DF'wx と '007E'wx にそれぞれ変換されます。 一方、CODEPAGE(1141) オプションでは、これらの文字は ~ と ß を表し、ワイド文字の '007E'wx と '00DF'wx にそれぞれ変換されます。

## 例

このセクションでは、追加データ変換の例を示します。

## 例: DECIMAL FIXED から小数部がある BINARY FIXED への変換

```
dcl I fixed bin(31,5) init(1);
    I = I+.1;
```

この結果、I の値は 1.0625 になります。 これは、.1 が FIXED BINARY (5,4) に変換され、2 進数の近似値が 0.0001B (端数は丸められません) となるためです。 この数の 10 進表記は .0625 となります。 .1 の代わりに .1000 を指定した場合の結果は、これと異なります。

## 例: 算術からビット・ストリングへの変換

```
dcl A bit(1),
    D bit(5);
A=1;          /* A has value '0'B     */
D=1;          /* D has value '00010'B */
D='1'B;       /* D has value '10000'B */
if A=1 then go to Y;
       else go to X;
```

これは X への分岐になります。 なぜなら、下記の一連の処置が取られ、A への割り当てが行われるからです。

1. 10 進定数の 1 の属性は FIXED DECIMAL(1,0) です。この定数は、 属性が FIXED BINARY(4,0) である一時記憶域に割り当てられ、0001B という値を取ります。
2. 次に、この値が長さ (4) のビット・ストリングに変換されて、'0001'B になります。

3. ビット・ストリングは A に割り当てられます。 A の宣言された長さは 1 で、割り当てられる値の長さが 4 であるため、右側で切り捨てが行われ、A の最終的な値は '0'B になります。

IF ステートメント内の比較演算では、'0'B と 1 が FIXED BINARY に変換され、算術値として比較されます。両者は等しくないので、A=1 の関係は偽 の結果になります。

D への最初の割り当てでは、A への割り当て時と同じような一連の処置が行われます。ただし、割り当てられる値の右側にゼロが埋め込まれる点が異なっています。これは、D の長さが、割り当てられる値の長さより 1 だけ大きい長さとして宣言されているからです。

## 例: 算術から文字への変換

次に示す例では、負符号 (-)、10 進または 2 進小数点、および小数点の前の 先行ゼロを入れる場合に備えて、3 桁のブランクを用意しておく必要があります。

```
dcl A char(4),
    B char(7);
A='0'; /*A has value '0bbb'*/
A=0;   /*A has value 'bbb0'*/
B=1234567; /*B has value 'bbb1234'*/
```

## 例: 変換エラー

```
dcl Ctlno char(8) init('0');
do I=1 to 100;
Ctlno=Ctlno+1;
⋮
end;
```

この例では、システムで許可される FIXED DECIMAL の精度の最大値に 15 が使用されました。 この例では、下記の一連の処理が行われるので、CONVERSION 条件が起こります。

1. CTLNO の初期値、すなわち '0bbbbbbb' は FIXED DECIMAL(15,0) に 変換されます。
2. 属性 FIXED DECIMAL(1,0) を持つ 10 進定数 1 が追加されます。追加の規則に従って、結果の精度は (16,0) です。
3. 次に、この値を再び CTLNO に割り当てるために、この値の長さ 18 の文字ストリングに変換されます。
4. CTLNO の長さは 8 なので、割り当てによって右側が切り捨てられ、CTLNO の最終的な値はブランクだけになります。 2 回目のループでは、この値を算術タイプに正しく変換できません。

# 第 5 章 プログラムの編成

この章では、いくつかのステートメントを各種のブロックに まとめて、PL/I プログラムを編成する方法、ブロック間で制御が移される方法、 および異なるブロックで同一データを使用する方法を説明します。

多数のプログラマーが手分けして 1 つのプログラムを作成する場合は特に、 プログラムを複数のブロックに適切に分割して作業を進めると、 プログラムのコーディングもテストも簡単になります。 プログラムを適切に分割すれば、ストレージの有効利用にもつながります。 なぜなら、自動ストレージは、ストレージを宣言しているブロックに制御権が移動した 時点で割り振られ、ブロックが終了するときに解放されるからです。

## プログラム

PL/I はブロックで編成される言語であり、パッケージ、プロシージャー、開始ブロック、ステートメント、式、 および組み込み関数から構成されます。 PL/I アプリケーション は、ロード・モジュール と呼ばれる 1 つ以上の個別にロード可能なエンティティーから構成されます。 ロード・モジュールは、1 つ以上の個別にコンパイルされたエンティティー (コンパイル単位 *(CU)* と呼ばれます) から構成されます。 特に明記されていない限り、プログラム とは、 PL/I アプリケーションまたはコンパイル単位のことを意味します。

### プログラム構造

プログラム とは PL/I アプリケーションまたはコンパイル単位のことを言います。 これらは、ロード・モジュール を構成する、個別にコンパイルされるエンティティーです。 PL/I アプリケーション は 1 つ以上のロード・モジュール から構成されます。

コンパイル単位は、PL/I PACKAGE または外部 PROCEDURE です。 各パッケージには、0 個かそれ以上のプロシージャーが含まれます。 プロシージャーのいくつかは搬出可能です。 PL/I 外部プロシージャーまたは内部プロシージャーには、0 個かそれ以上のブロックが入ります。 PL/I ブロックは、PROCEDURE または BEGIN ブロックのいずれかであり、 それぞれのブロックに、ステートメントまたはブロック (あるいはその両方) が含まれ ます。(設定されない場合もあります)

PL/I ブロックで、高位モジュラー・アプリケーションを処理することができます。 なぜなら、ブロックには、変数名やストレージ・クラスを定義する宣言を入れることができるからです。 そのため、単一ブロックまたはブロック・グループに対して変数の 有効範囲を制限することができます。あるいは、コンパイル単位 またはロード・モジュール全体を通してそれを指定することができます。

ブロックをそれ自体に入れる程度を自由に決めることができるので、PL/I は、コードを再使用できるよう、多数のコンパイル単位やアプリケーションが共用できるブロックを 処理することができます。

87 ページの図 1 で、アプリケーション構造を説明しています。

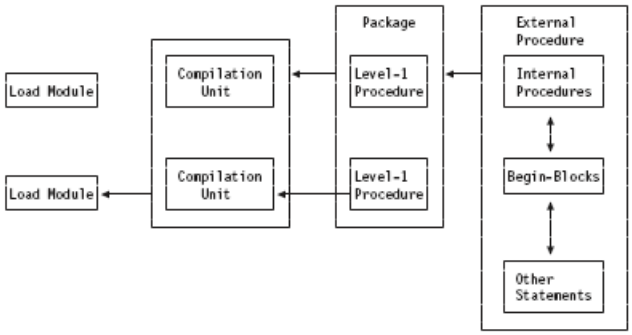



図 *1. PL/I* アプリケーション構造

© Copyright IBM Corp. 1999, 2022

**関連情報**

90 ページの『パッケージ』
パッケージとは、宣言、デフォルト・ステートメント、およびプロシージャー・ブロックだけ入れることができるブロックのことです。 名前が再び宣言されない限りは、パッケージは、パッケージ内に含まれるすべての宣言およびプロシージャーによって共用される名前の有効範囲を構成します。

92 ページの『プロシージャー』
プロシージャーとは、PROCEDURE ステートメントとそれに対応する END ステートメントとで区切られている一連のステートメントです。 プロシージャーは、主プロシージャー、サブルーチン、または関数などです。

109 ページの『開始ブロック』
開始ブロックは、BEGIN ステートメントとそれに対応する END ステートメントによって区切られている一連のステートメントです。

## プログラムの活動化

PL/I プログラムがアクティブになるのは、呼び出し側プログラムが主プロシージャー を呼び出したときです。

この呼び出し側プログラムは、通常はオペレーティング・システムですが、別の プログラムでもかまいません。主プロシージャーは、外部プロシージャーであり、ステートメントに OPTIONS(MAIN) の指定があります。

### 例: 外部プロシージャーを呼び出す主プロシージャー

この例では、Contrl が主プロシージャーであり、プログラム内の他の外部プロシージャーを呼び出します。 主プロシージャーは、プログラムが実行されている間ずっと アクティブになっています。

```
Contrl: procedure options(main);
  call A;
  call B;
  call C;
end Contrl;
```

## プログラムの終了

主プロシージャーが終了すると、プログラムは終了します。 プログラムが正常に終了したか異常終了したかに関係なく、終了すると、 呼び出し側プログラムに制御が戻されます。

88 ページの『例: 外部プロシージャーを呼び出す主プロシージャー』では、制御が C プロシージャーから Contrl プロシージャーに戻されると、Contrl は終了します。

**関連情報**

98 ページの『プロシージャーの終了』
プロシージャーが終了するのは、プロシージャー参照以外の方法で、 呼び出し側プログラム、呼び出し側ブロックに制御を戻したとき、 または別のアクティブ・ブロックに制御を渡したときです。

# ブロック

ブロック は、1 つの単位として処理され、その中で宣言された名前の有効範囲および名前用のストレージ割り振りを指定する、区切り文字で区切られた一連のステートメントです。 ブロックとしては、パッケージ、プロシージャー、または開始ブロックの いずれもありえる。

- ブロックは、その中で宣言された名前の有効範囲を設定します。
- ブロックは、自動変数の割り振りを制限します。
- ブロックは、DEFAULT ステートメントの有効範囲を決定します (165 ページの『属性のデフォルト』に説明があります)。

ブロックには、パッケージ、プロシージャー、および開始 の種類があります。

これらのブロックには、名前のローカル定義と見なされる宣言を入れることができます。 これは、名前の有効範囲を設定するため、 および自動変数の割り振りを制限するために行われます。 これらの宣言は、それ自身のブロックの外では知られることがなく、また、そのブロックを含んでいるブロック内でその名前を参照することはできません。 詳しくは、150 ページの『宣言の有効範囲』を参照してください。

ストレージは、ストレージが宣言されるブロックの入り口で自動変数に割り振られ、 ブロックから出るときに解放されます。 詳しくは、150 ページの『宣言の有効範囲』を参照してください。

## ブロックの活動化

割り振り内、ストレージの解放、名前の有効範囲の削除では、 各ブロックは同じ役割を果たします。 パッケージは、活動化されず、終了もしません。

活動化が行われる方法については、92 ページの『プロシージャー』および109 ページの『開始ブロック』を参照してください。

ブロックの活動化中には、以下の操作が行われます。

- 宣言ステートメントに書かれている式は、 エクステントおよび初期値 (反復因数も含めて) で計算されます。
- 自動変数用にストレージが割り振られ、 指定されていれば初期設定値がセットされます。
- ストレージは仮引数用に割り振られ、このブロックで作成される可能性がある コンパイラー作成の一時域用にも割り振られます。

自動変数の初期値およびエクステントは、同じブロック内で宣言されたほかの 自動変数の値やエクステントによって変化してはなりません。 例えば、以下の初期設定を行うと、 J と K についての無効な結果を生成する場合があります。

```
   dcl I init(10),J init(K),K init(I);
```

同様に、以下のコードでは、この構造体の初期化後に b には未定義の値 (値 10 でない可能性が最も高い) が入ります。

```
dcl
  1 a,
    2 b fixed bin init(c),
    2 c fixed bin init(10);
```

データ項目の宣言は、相互に依存し合うものであってはなりません。 例えば、次のような宣言は正しくありません。

```
   dcl A(B(1)), B(A(1));

   dcl D(E(1)), E(F(1)), F(D(1));
```

ブロックが活動化される途中でエラーが発生し、ERROR 条件 (またはその他の条件) が 起こることがあります。 そのような場合は、そのブロックの環境が不完全であることがあります。 特に、一部の自動変数が割り振られていないことがあります。 ERROR 条件の発生後に自動変数を参照するステートメントが実行されると、 割り振られていないストレージを参照する可能性があります。 割り振られていないストレージの参照の結果は未定義です。

## ブロックの終了

ブロックを終了する方法はいく通りもあります。 パッケージは、活動化されず、終了もしません。

終了方法については、92 ページの『プロシージャー』および109 ページの『開始ブロック』を参照してください。

ブロックの終了中には、以下の操作が行われます。

- ON ユニット環境が、ブロックの活動化以前の状態に再設定されます。
- そのブロック内のすべての自動変数に割り振られたストレージが解放されます。

# パッケージ

パッケージとは、宣言、デフォルト・ステートメント、およびプロシージャー・ブロックだけ入れることができるブロックのことです。 名前が再び宣言されない限りは、パッケージは、パッケージ内に含まれるすべての宣言およびプロシージャーによって共用される名前の有効範囲を構成します。

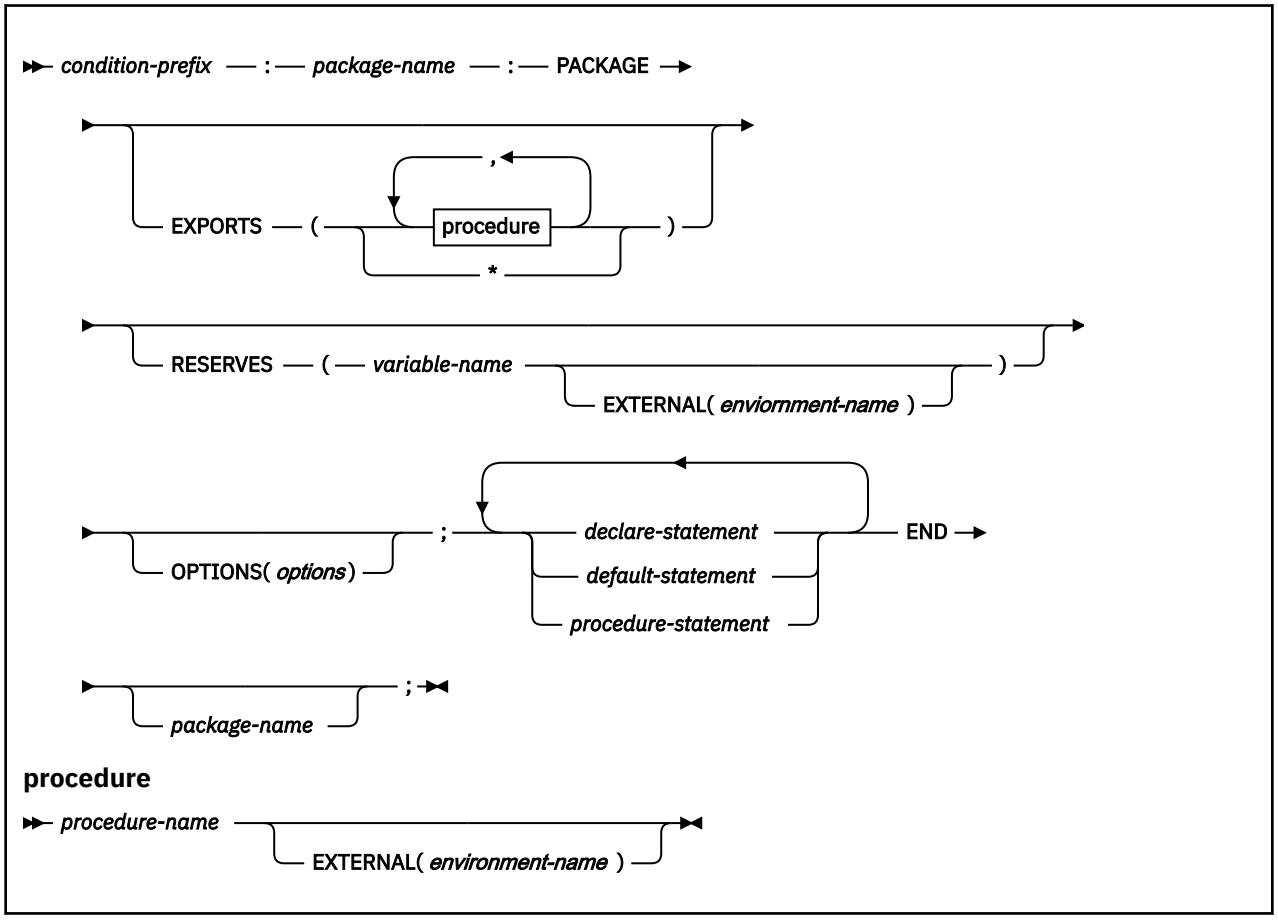
レベル 1 のプロシージャーの一部またはすべてが搬出され、 外部プロシージャーとしてパッケージの外で認識できるようになります。 パッケージは、複数のエントリー・ポイント・アプリケーションを実施する際に使用することができます。

MAIN プロシージャーを含んだパッケージは、FETCHABLE プロシージャーを含めてはいけません。 MAIN プロシージャーを含むパッケージは DLL にリンクしてはなりません。 このパッケージは、必要に応じて DLL 内のルーチンを呼び出すことができる基本実行可能モジュールの一部を形成する必要があります。 もちろん、このようなパッケージでは、静的にリンクされた他のルーチンから呼び出すことができる外部ルーチンも定義できます。 また、このパッケージでは、静的にリンクされた他のルーチンから参照可能な EXTERNAL STATIC データも定義できます。

MAIN ルーチンを含まないパッケージが DLL にリンクされた場合、DLL のそのパッケージからエクスポートされる EXTERNAL STATIC 変数は、RESERVED 属性を持つ変数のみです。

ソースに PACKAGE ステートメントが含まれる場合、最大でも 1 セットだけの *PROCESS ステートメントがなければならず、それはソースの最初のステートメントでなければなりません。 ソースに PACKAGE ステートメントが含まれない場合は、コンパイラーによって、最初の *PROCESS ステートメントのセットの後に有効にこのステートメントが挿入され、ソースには *PROCESS ステートメントのグループによって分離される複数の外部プロシージャーが含まれる場合があります。

2 つのパッケージが互いにリンクしている場合、名前の素集合をエクスポートする必要があります。



**condition-prefix**
PACKAGE ステートメントで指定される条件接頭語は、PROCEDURE ステートメントで指定変更されない限り、 パッケージ内に含まれているすべてのプロシージャーに適用されます。 詳しくは、339 ページの『条件接頭語』を参照してください。

**package-name**
パッケージの名前です。 リンク・モジュール内のすべてのパッケージ名は固有でなければなりません。

**EXPORTS**
すべての (EXPORTS(*)) または名前のついたプロシージャーは、搬出され、 パッケージ外で外部的に認識されることを指定します。 EXPORTS オプションを指定しないと、EXPORTS(*) が想定されます。

**プロシージャー名**
パッケージ内のレベル 1 のプロシージャーの名前です。

**EXTERNAL (environment name)**
150 ページの『宣言の有効範囲』に記載されている有効範囲属性です。

**RESERVES**
このパッケージがすべて (RESERVES(*)) に対してストレージを予約しているか、 あるいは RESERVED 属性を持つ名前付きの変数に対してだけ ストレージを予約しているかを指定します。 155 ページの『RESERVED 属性』を参照してください。

**variable name**
レベル 1 の外部静的変数の名前です。

**OPTIONS (option)**
PACKAGE ステートメントに適用可能な OPTIONS オプションについては、123 ページの『OPTIONS オプションとその属性』を参照してください。

**declare statement**
パッケージ内で宣言される変数は、その宣言がレベル 1 の プロシージャーの外部で行われる場合には、静的、基底付き、 被制御のストレージ・クラスを持たなければなりません。 自動変数を使用することはできません。 デフォルトのストレージ・クラスは、STATIC です。 147 ページの『第 7 章 データ宣言』を参照してください。

**default statement**
165 ページの『属性のデフォルト』を参照してください。

**procedure statement**
93 ページの『PROCEDURE ステートメント』を参照してください。

## PACKAGE ステートメントの例

### *PACKAGE* ステートメント

```
*Process S A(F)  LIMITS(EXTNAME(31)) NUMBER;
 Package_Demo: Package exports (Factorial);

 /*********************************************/
 /*                 Common Data               */
 /*********************************************/

 dcl N fixed bin(15);
 dcl Message char(*) value('The factorial of ');

 /*********************************************/
 /*                 Main Program              */
 /*********************************************/

 Factorial: proc options (main);
    dcl Result fixed bin(31);
    put skip list('Please enter a number whose factorial ' ||
                  'must be computed ');
    get list(N);
    Result = Compute_factorial(n);
    put list(Message || trim(N) || ' is ' || trim(Result));
 end Factorial;

 /*********************************************/
```

```
/*                  Subroutine                  */
/*********************************************/

Compute_factorial: proc (Input) recursive returns (fixed bin(31));
   dcl Input fixed bin(15);
   if Input <= 1 then
      return(1);
   else
      return( Input*Compute_factorial(Input-1) );
end Compute_factorial;

end Package_Demo;
```

# プロシージャー

プロシージャーとは、PROCEDURE ステートメントとそれに対応する END ステートメントとで区切られている一連のステートメントです。 プロシージャーは、主プロシージャー、サブルーチン、または関数などです。

アプリケーションは、OPTIONS(MAIN) を持つ 1 つの外部プロシージャーを 持たなければなりません。以下の例では、プロシージャーの名前は Name であり、プロシージャーのエントリー・ポイント を表しています。

```
Name:
   procedure;
end Name;
```

ENTRY ステートメントは、プロシージャーへの 2 次エントリー・ポイントを定義することができます。 次の例を考えてください。

```
Name: procedure;
B: entry;
end Name;
```

B は、Name プロシージャーへの 2 次エントリー・ポイントを定義します。 ENTRY ステートメントについては、112 ページの『ENTRY 属性』を参照してください。

プロシージャーは、1 つの名前を持たなければなりません。 別のプロシージャー内または開始ブロック内でネストされたプロシージャー・ブロックは、内部プロシージャー と呼ばれます。別のプロシージャーまたは開始ブロック内にネストされていない プロシージャー・ブロックは、外部プロシージャー と呼ばれます。 パッケージから搬出されたレベル 1 のプロシージャーも、外部プロシージャーになります。 外部プロシージャーは、ほかのコンパイル単位のほかのプロシージャーで 呼び出すことができます。 プロシージャーは、ほかのプロシージャーを呼び出すことができます。

プロシージャーを再帰的にすることができます。つまり、その中から再活動化したり、 すでに活動している別の活動プロシージャー内から再活動化したりすることができます。 プロシージャーを呼び出すときに引数を渡すことができます。

**関連情報**

150 ページの『宣言の有効範囲』
名前の適用対象となるプログラム部分は、その名前についての宣言の有効範囲 と呼ばれます。 ほとんどの場合、名前の宣言の有効範囲は、プログラム内で 名前が宣言される位置によって完全に決定されます。

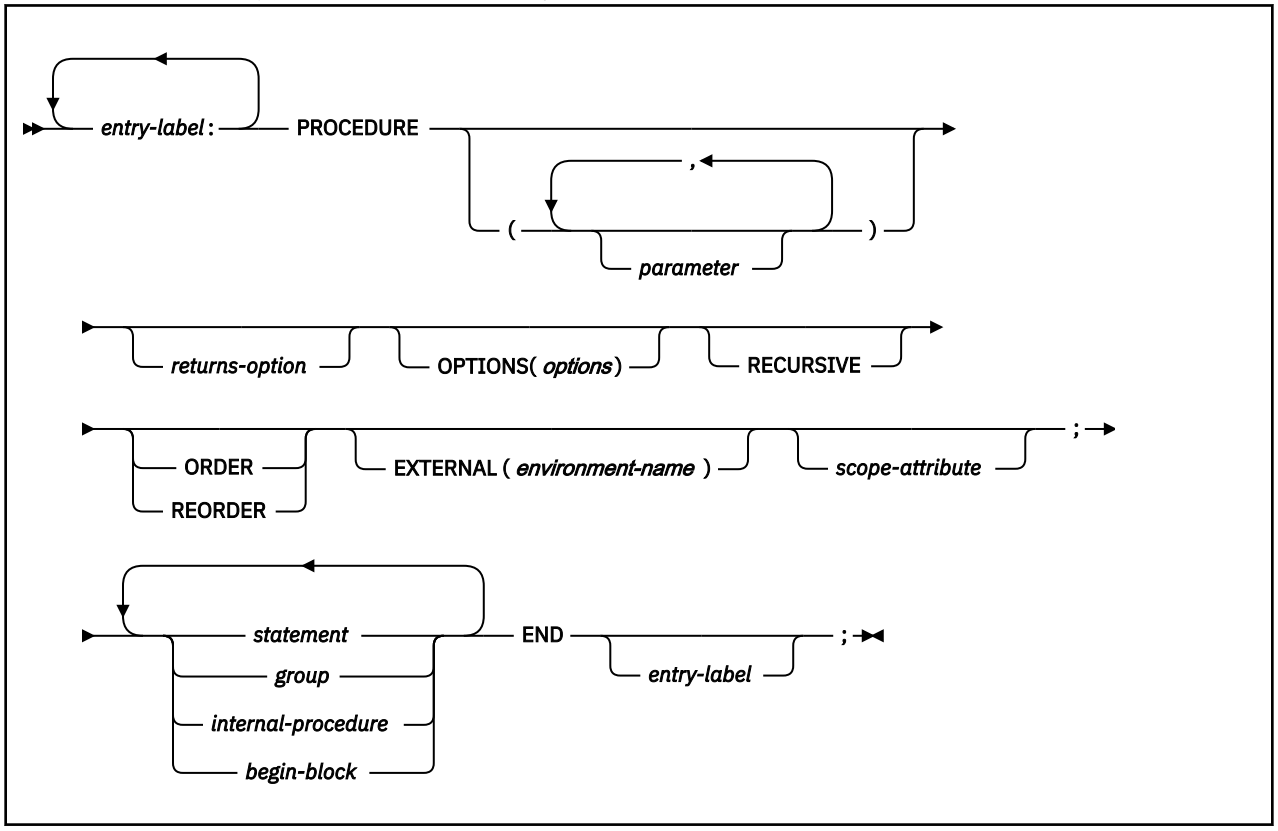
102 ページの『サブルーチン』
サブルーチン は、CALL ステートメントで呼び出される 内部または外部プロシージャーです。

104 ページの『関数』
関数 とは、ゼロまたはそれ以上の引数を持つプロシージャーであり、式で関数参照 を使用して呼び出されます。

106 ページの『プロシージャーへの引数の引き渡し』
関数またはサブルーチンが呼び出されると、パラメーターは、 左から右の順に、渡される引数と関連付けられます。

# PROCEDURE ステートメント

PROCEDURE ステートメントは、 プロシージャーを主プロシージャー、サブルーチン、 または関数として識別します。 プロシージャーおよびその他の特性によって予期されるパラメーターも、 プロシージャー・ステートメント上で指定することができます。




**省略形**: PROCEDURE の場合は PROC

**entry-label**
プロシージャーへのエントリー・ポイントです。 外部入り口は、呼び出されるプロシージャー内で明示的に宣言されます。 複数の入り口ラベルを指定すると、左端の名前が 1 次エントリー・ポイントとなり、 それは PROCNAME と ONLOC 組み込み関数が戻す名前です。 詳しくは、110 ページの『入り口データ』を参照してください。

**パラメーター**
95 ページの『パラメーター属性』 及び 106 ページの『プロシージャーへの引数の引き渡し』 を参照してください。

**returns-option**
関数プロシージャーについてのみ適用します。 104 ページの『関数』 及び 132 ページの『RETURNS オプションとその属性』 を参照してください。

**OPTIONS (option)**
123 ページの『OPTIONS オプションとその属性』を参照してください。

**RECURSIVE**
99 ページの『再帰的プロシージャー』を参照してください。

**ORDER または REORDER**
ORDER と REORDER は、プロシージャーまたは開始ブロックに指定される最適化オプションです。 ORDER と REORDER の使用の詳細については、123 ページの『OPTIONS オプションとその属性』の"ORDER または REORDER"を参照してください。

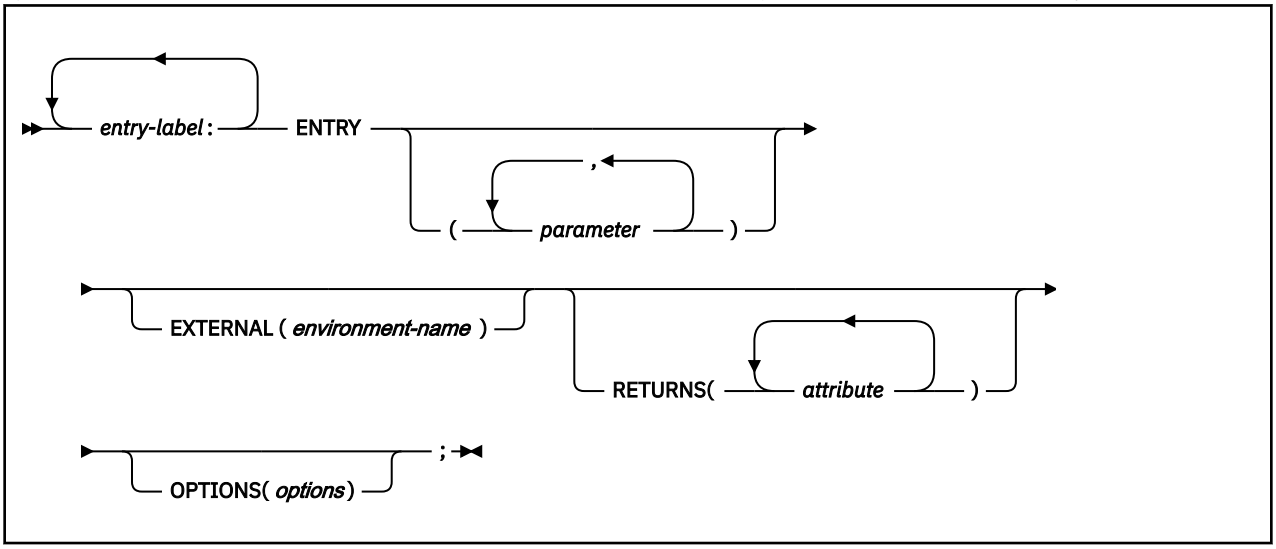
**EXTERNAL (environment name)**
150 ページの『宣言の有効範囲』に記載されている有効範囲属性です。

**scope-attribute**
150 ページの『宣言の有効範囲』を参照してください。

プロシージャー (サブルーチンまたは関数) は、1 つまたは複数のエントリー・ポイントを持つことができます。 プロシージャーへの 1 次エントリー・ポイントは、プロシージャー・ステートメントの 左端のラベルによって確立されます。 プロシージャーへの 2 次エントリー・ポイントは、 PROCEDURE ステートメントの追加のラベルおよび ENTRY ステートメントによって確立されます。 各エントリー・ポイントには 1 つの入り口名があります。 外部名の作成規則についての説明は、152 ページの『INTERNAL 属性と EXTERNAL 属性』を参照してください。

## ENTRY ステートメント

ENTRY ステートメントは、プロシージャーの 2 次エントリー・ポイントを指定します。 ENTRY ステートメントは、2 次エントリー・ポイントを定義するプロシージャーの内部になければなりません。 反復実行を指定する DO グループ内に置くこと、または ON ユニットの内部で あることはできません。




**entry-label**
プロシージャーへの 2 次エントリー・ポイントです。

**パラメーター**
95 ページの『パラメーター属性』 及び 106 ページの『プロシージャーへの引数の引き渡し』 を参照してください。

**EXTERNAL (environment name)**
150 ページの『宣言の有効範囲』に記載されている有効範囲属性です。

**RETURNS オプション**
132 ページの『RETURNS オプションとその属性』を参照してください。

**OPTIONS (option)**
123 ページの『OPTIONS オプションとその属性』を参照してください。

ENTRY ステートメントのパラメーターはすべて BYADDR でなければ ならず、またプロシージャーが ENTRY ステートメントを含んでいる場合は、 そのプロシージャーに対する非ポインター・パラメーターはすべて BYADDR でなければなりません。

ENTRY ステートメントを含むプロシージャーに RETURNS オプションがある場合、または含まれている ENTRY ステートメントのいずれかに RETURNS オプションがある場合は、以下の条件が適用されます。

- そのプロシージャーと ENTRY ステートメントの RETURNS オプションすべてに、 BYADDR 属性が指定 (または、コンパイル時オプション DEFAULT(RETURNS(BYADDR)) に よって暗黙指定) されている。
- さらに、これらのエントリー・ポイントのいずれかを呼び出すルーチンすべてが、 RETURNS(BYADDR) を指定して入り口を宣言しているか、 DEFAULT(RETURNS(BYADDR)) コンパイラー・オプションを指定して コンパイルされている。

プロシージャーに ENTRY ステートメントがあり、あるエントリー・ポイントには RETURNS 属性が 指定されているが、全エントリー・ポイントでは指定されていないときは、次の状 況で ERROR 条件が検出されます。

- プロシージャーが RETURNS 属性がないエントリー・ポイントに入力されたときに、 コードが式を使用して RETURN ステートメントを実行する場合。
- プロシージャーが、RETURNS 属性を持つエントリー・ポイントに入力されたときに、コードが式を使わずに RETURN ステートメントを実行する場合。

# パラメーター属性

パラメーターは、PROCEDURE または ENTRY ステートメント内に指定したパラメーター属性でコンテキストにもとづいて宣言されます。

パラメーターは、適切な属性を使用して明示的に宣言されなければなりません。 パラメーター属性は宣言内に指定することもできます。 属性が DECLARE ステートメント内に指定されていない場合には、 デフォルトの属性が適用されます。 パラメーター名は、添え字を付けたり、修飾することはできません。

21 ページの表 8 と以下では、パラメーターに対して宣言できる属性を説明します。

パラメーターには、常に INTERNAL 属性があります。

パラメーターが構造体または共用体である場合には、レベル 1 の 名前が指定されなければなりません。

パラメーターは、CONTROLLED ストレージ・クラス属性しか持つことができません。 被制御パラメーターは、被制御引数を持つ必要があり、 INITIAL 属性も持つことができます。

レコード単位の入出力で使用されるパラメーターまたは DEFINED 項目の基本変数としてのパラメーターは、連結ストレージ内になければなりません。 CONNECTED 属性は、プロシージャー内の宣言、 およびプロシージャー入り口宣言の記述子リスト内の両方で 指定されなければなりません。

## 単純パラメーターの境界、長さ、およびサイズ

単純パラメーターの境界、長さ、およびサイズは、 アスタリスクまたは制限付き式のいずれかによって指定されなければなりません。

実際の長さ、境界、またはサイズが起動のたびに異なるときには、 それぞれをアスタリスクで DECLARE ステートメントに指定できます。 アスタリスクが使用されている場合は、関連した引数の現行世代から長さ、 境界、またはサイズが取得されます。

関連した引数が仮引数である場合、 集合体のエレメントであるストリングの長さ指定として アスタリスクを使用することはできません。 ストリングの長さは整数として指定する必要があります。

## 被制御パラメーターの境界、長さ、およびサイズ

被制御パラメーターの境界、長さ、またはサイズは、 アスタリスクや要素式を使用して DECLARE ステートメントに 指定することができます。

### アスタリスクの表記法

アスタリスクを使用するとき、被制御パラメーターの長さ、境界、 またはサイズは、関連した引数の現行世代から取得されます。 異なる長さ、境界、またはサイズを ALLOCATE ステートメントに指定しないか ぎり、以後の被制御パラメーターの割り振りは、同じ境界、長さ、または サイズを使用して行われます。

引数の現行世代が存在しない場合、アスタリスクはパラメーターの 次元数だけを決定します。そのパラメーターに対してその他の参照を行うには、呼び込まれたプロシージャーの ALLOCATE ステートメント によって、事前に被制御パラメーターの境界、長さ、またはサイズが 指定されている必要があります。

### 式の表記法

パラメーターが割り振られるたびに、式が計算されて、 新しい割り振りの現行境界、長さ、またはサイズを指定します。 ただし、その DECLARE ステートメントの式は、 ALLOCATE ステートメント自体の境界、長さ、またはサイズの 指定によって指定変更されます。

## パラメーターの付いた配列引数の例

この例では、配列はパラメーター付きで宣言されています。

パラメーター付きの配列引数では、Sub1 が呼び出されると、割り振られている A および B が渡されます。

**パラメーターの付いた配列引数**

```
%process or('|') num  margins(1,72);
Package:package exports(*);


Main: procedure options(main);
declare (A(NA), B(NB), C(NC), D(ND) ) controlled;
declare (NA init(20), NB init(30), NC init(100),
ND init(100) ) fixed bin(31);
declare Sub1 entry((*) controlled, (*) controlled);
declare Sub2 entry ((*) ctl, (*) ctl, fixed bin);


allocate A,B;  /* A(20), B(30) */
display ('Gen1: DIM(A)=' || dim(A) || ', ' || "DIM(B)=" || dim(B));
call Sub1(A,B);


display ('Gen2: Allocn(A)=' || allocn(a) || ', ' ||
'Allocn(B)=' || allocn(B) );
display ('Gen2: DIM(A)=' || dim(A) || ', ' || "DIM(B)=" || dim(B));
free A,B;
display ('Gen1: Allocn(A)=' || allocn(A) || ', ' ||
'Allocn(B)=' || allocn(B) );
display ('Gen1: DIM(A)=' || dim(A) || ', ' || "DIM(B)=" || dim(B));
free A,B;
display ('Gen0: Allocn(A)=' || allocn(A) || ', ' ||
'Allocn(B)=' || allocn(B) );
call Sub2 (C,D,10);


display ('Gen1: Allocn(C)=' || allocn(C) || ', ' ||
'Allocn(D)=' || allocn(D) );
display ('Gen1: DIM(C)=' || dim(C) || ', ' || "DIM(D)=" || dim(D));
free C,D;
display ('Gen0: Allocn(C)=' || allocn(c) || ', ' ||
'Allocn(D)=' || allocn(D) );
end Main;


Sub1: procedure (U,V);
dcl (U(UB), V(*)) controlled,
UB fixed bin(31);
display ('Gen1: Allocn(U)=' || allocn(U) || ', ' ||
'Allocn(V)=' || allocn(V) );
display ('Gen1: DIM(U)=' || dim(U) || ', ' || "DIM(V)=" || dim(V));
UB=200;
allocate U,V; /* U(200), V(30) */
display ('Gen2: Allocn(U)=' || allocn(U) || ', ' ||
'Allocn(V)=' || allocn(V) );
display ('Gen2: DIM(U)=' || dim(U) || ', ' || "DIM(V)=" || dim(V));
end Sub1;


Sub2: procedure (X,Y,N);
dcl (X(N),Y(N)) controlled,
N fixed bin;
display ('Gen0: Allocn(X)=' || allocn(X) || ', ' ||
'Allocn(Y)=' || allocn(Y) );
allocate X,Y;  /* X(10), Y(10) */
display ('Gen1: Allocn(X)=' || allocn(X) || ', ' ||
'Allocn(Y)=' || allocn(Y) );
display ('Gen1: DIM(X)=' || dim(X) || ', ' || "DIM(Y)=" || dim(Y));
end Sub2;


end Package;
```

Sub1 の ALLOCATE ステートメントは、A および B の第 2 世代を割り振ります。 B には両方の世代で同じ境界がありますが、A には 2 番目の世代で異なる境界があります。

Main へ戻る際、最初の FREE ステートメントは (Sub1 で 割り振られている) A と B の 2 番目の世代を解放します。 2 番目の FREE ステートメントは、(Main で 割り振られている) A と B の最初の世代を解放します。

Sub2 では、X と Y は、N の値によって 異なる境界を使用して宣言されています。 X と Y が割り振られたら、それらの値を使用して、 割り振られた配列の境界を判別します。

Sub2 から Main へ戻る際、FREE ステートメントは、 (Sub2 で割り振られている) C と D の 世代のみを解放します。

## プロシージャーの活動化

プログラムの流れではプロシージャーは迂回され、PROCEDURE ステートメントの 前にあるステートメントが実行されたあと、そのプロシージャーの END ステートメントの あとにあるステートメントが実行されます。 プロシージャーはプロシージャー参照 によってのみ活動化できます。

メイン・プロシージャーの活動化方法については、88 ページの『プログラムの活動化』を参照してください。

呼び出しプロシージャーの実行は、呼び出されたプロシージャーが 呼び出すプロシージャーに制御を戻すまで遅らされます。

下記のいずれかのコンテキストの中に入り口式が書かれていることを、プロシージャー参照といいます。

- 121 ページの『CALL ステートメント』で説明されているように、CALL ステートメントを使用してサブルーチンを呼び出すコンテキスト
- 104 ページの『関数』で説明されているように、関数を呼び出すコンテキスト

この節の情報は、これらのコンテキストと関係があります。 ただしこの章の例では、CALL ステートメントを使用しています。

プロシージャー参照が実行されると、指定されたエントリー・ポイントを含む プロシージャーは呼び出される ことになります。プロシージャー参照が書かれている場所を呼び出し点 といい、参照が行われるブロックを呼び込み側ブロック といいます。呼び出し側ブロックは、呼び出されたプロシージャーに制御が移されても、アクティブのままです。

1 次エントリー・ポイントからプロシージャーが呼び出されると、 引数およびパラメーターは関連付けられ、 呼び出されたプロシージャーの最初のステートメントから実行が始まります。 プロシージャーが ENTRY ステートメントで 2 次エントリー・ポイントから呼び出されると、ENTRY ステートメントに続く最初のステートメントから、実行が始まります。 1 次エントリー・ポイントで、あるブロックの入り口について確立された環境は、 同じブロックが 2 次エントリー・ポイントで呼び出される時に確立される環境と同じです。

2 つのプロシージャー間の通信は、呼び出しプロシージャーから渡される引数、呼び込まれたプロシージャーから戻される値、および双方のプロシージャーにおいて既知の名前のいずれかによって行われます。 したがって、同じプロシージャーでも、呼び出し点が異なれば異なるデータを処理することができます。 次の例を考えてください。

```
Readin: procedure;
statement-1
statement-2
Errt: entry;
statement-3
statement-4
end Readin;
```

プロシージャーは、以下のいずれかの入り口参照によって活動化できます。

```
   call Readin;
   call Errt;
```

call Readin ステートメントは、その 1 次エントリー・ポイントで Readin を呼び出し、statement-1 で実行が開始されます。 call Errt ステートメントは 2 次エントリー・ポイント Errt で、Readin プロ

シージャーを呼び出し、実行は statement-3 で開始されます。 入り口定数 (Readin) を、以下の例のように、プロシージャー参照で使用される入り口変数に割り当てることもできます。

```
declare Readin entry,
        Ent1 entry variable;
Ent1 = Readin;
call Ent1;
call Readin;
```

2 つの CALL ステートメントは、同じ効果があります。

# プロシージャーの終了

プロシージャーが終了するのは、プロシージャー参照以外の方法で、 呼び出し側プログラム、呼び出し側ブロックに制御を戻したとき、 または別のアクティブ・ブロックに制御を渡したときです。

プロシージャーは、以下の状況下では正常 終了します。

- プロシージャー内の RETURN ステートメントに制御が達したとき。RETURN ステートメントが実行されると、呼び出し側プロシージャー内の呼び出し点に制御が戻されます。 呼び出し点が CALL ステートメントであれば、CALL の次のステートメントで、呼び出し側プロシージャー内の実行が再開します。 呼び出し点が関数参照であれば、その参照を含んでいるステートメントの実行が再開します。
- プロシージャー内の END ステートメントに制御が達したとき。 この場合、RETURN ステートメントが実行されたときと同じ結果になります。

プロシージャーは、以下の状況下では異常 終了します。

- プロシージャーの外へ制御権を移動する GO TO ステートメントに制御が達したとき。 GO TO ステートメントを含んでいるブロック内のラベルか、 またはプロシージャーに渡されたラベル引数に関連付けられている パラメーターを指定することができます。 単一スレッド・プログラムの現行スレッド内、またはマルチスレッド・プログラムの任意のスレッド内で、STOP ステートメントが実行されたとき。
- EXIT ステートメントが実行されたとき。
- ERROR 条件が起こり、ERROR または FINISH の ON ユニットが確立されていないとき。 さらに、一方または両方の条件の ON ユニットが確立されているが、 ON ユニットから出る方法が GO TO ステートメントではなく 正常な戻りによって出るようになっているとき。
- プロシージャーが、異常終了する別のプロシージャーを呼び出したとき。

GO TO ステートメントを使用してプロシージャーの外へ制御権を移動すると、 場合によっては、いくつかのプロシージャーや開始ブロックが終了することが あります。具体的に言えば、GO TO ステートメントで指定した移動先の場所が、終了しようとしているブロックを直接活動化したのではないブロックに含まれている場合には、活動化の順序においてその 2 つのブロックの間にあるすべてのブロックが終了します。 以下の例について考察します。

```
A: procedure options(main);
   statement-1
   statement-2
   B: begin;
      statement-b1
      statement-b2
      call C;
      statement-b3
   end B;
   statement-3
   statement-4
   C: procedure;
      statement-c1
      statement-c2
      statement-c3
      D: begin;
         statement-d1
         statement-d2
         go to Lab;
         statement-d3
      end D;
      statement-c4
   end C;
   statement-5
Lab: statement-6
```

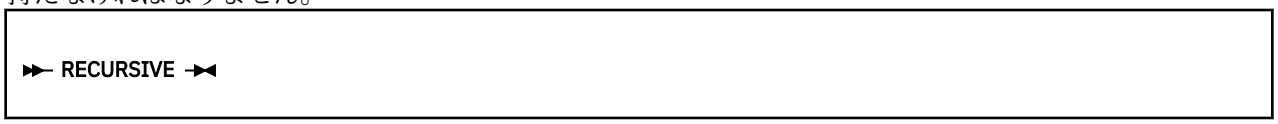
```
   statement-7
end A;
```

A は B をアクティブ化し、これによって C がアクティブ化され、さらに D がアクティブ化されます。 D では、ステートメント go to Lab は、制御を A 内の statement-6 に移します。 このステートメントは、D、C、または B には含まれていないので、この 3 つのブロックがすべて終了します。 A は活動化されたままです。 したがって、D の外へ制御権を移動することによって、 ブロック D が終了するだけでなく、 介在している B と C も終了します。

## 再帰的プロシージャー

それ自体から呼び出される、または別のアクティブのプロシージャーから 呼び出されるアクティブのプロシージャーは、 再帰的 プロシージャーです。 そのような呼び出しは、再帰と呼ばれます。

再帰的に呼び出されたプロシージャーは、PROCEDURE ステートメントで 指定された RECURSIVE 属性を持たなければなりません。




再帰プロシージャーの呼び出しごとの環境 (自動変数の値など) は、 被制御変数の割り振りのスタッキングと類似した方法で 保存されます (238 ページの『被制御ストレージとその属性』を参照)。 環境は、再帰的呼び出し時に、最後に記憶された環境がスタックに入れられ、 その呼び出しの終了時にスタックから除去される と考えてください。 現行ブロック内のラベル定数は、常に、そのラベルが入っているブロックの現行呼び出しへの参照になります。

ラベル定数が特定呼び出しのラベル変数に割り当てられ、 ラベル変数が再帰的プロシージャー内部で宣言されていない場合には、別の呼び出しでその変数を指定する GO TO ステートメントが、現行および介在するプロシージャーや開始ブロックを終了し、 割り当てが行われたときの環境を復元します。

再帰的プロシージャー内から入り口変数を通して呼び出された別の プロシージャーの環境は、その入り口定数がその変数に 割り当てられたときに存在していた環境です。 次の例を考えてください。

```
I=1;
call A;                                /* First invocation of A      */

A: proc recursive;
   declare Ev entry variable static;
   if I=1 then
      do;
        I=2;
        Ev=B;
        call A;                        /* 2nd invocation of A        */
      end;
   else call Ev;                       /* Invokes B with environment */
                                       /* of first invocation of A   */
B: proc;
   go to Out;
end B;
Out: end A;
```

プロシージャー B の中の GO TO ステートメントは、 1 回目に呼び出された A 内の END A ステートメントに 制御を移し、B を終了させるほかに、2 つの A を終了させます。

### 再帰が自動変数に及ぼす影響

再帰的プロシージャーの 1 回の活動化時に割り振られた変数の値は、次にそのプロシージャーが活動化されたときに変更されないようにしなければなりません。 これは、変数のスタッキングによって解決されます。スタックは、後入れ先出し法で動作します。 参照できるのは、自動変数の最新の世代のみです。 静的変数は、再帰の影響を受けないので、再帰的に呼び出されたプロシージャー相互間の連絡に使用できます。これは、再帰的プロシージャーを含んでいるプロシージャー内で宣言された自動変数にも、被制御変数と基底付き変数にも適用されます。

次の例を考えてください。

```
A: proc;
   dcl X;
   .
   .
   .
   B: proc recursive;
       dcl Z,Y static;
       call B;
      .
      .
      .
   end B;
 end A;
```

プロシージャー B の呼び出し全体にわたって、変数 X の単一の世代が存在します。 変数 Z は、プロシージャー B の呼び出しごとに異なる世代を持ちます。 変数 Y は、プロシージャー B の中でだけ 参照することができ、呼び出しのつど再割り振りされることは ありません (変数のスタッキングという概念は、238 ページの『被制御ストレージとその属性』の被制御変数の説明でも重要です)。

## 外部プロシージャーの動的ロード

モジュールは、FETCH および RELEASE ステートメントを使用して PL/I プログラムによって動的に取り出されたり、(ロードされたり)、解放されたり (削除されたり) します。

プロシージャー参照によって呼び出されたプロシージャーは、通常、プログラムの実行中ずっと主記憶域に常駐します。 ただし、必要とされるときにのみ、 プロシージャーを主記憶域にロードしておくこともできます。 つまり、呼び出しプロシージャーの実行中に、呼び出されたプロシージャーを 主記憶域に動的にロードし、あとで主記憶域から動的に削除します。

プロシージャーの動的ロードと動的削除が特に効果的なのは、呼び出しプロシージャーが実行されるつど、呼び出されたプロシージャーが必ず呼び出されるとは限らないときや、実行時間の短縮よりも主記憶域の節減のほうが重要であるときです。

FETCH ステートメントで入り口定数を指定すると、その入り口定数を含んでいる プロシージャーのコピーがすでに主記憶域に存在しているのでない限り、 そのプロシージャーをまず主記憶域にロードしなければ実行できないことを表します。 名前が FETCH ステートメントで参照される場合、プロシージャーは以下のようにディスクからロードすることもできます。

- CALL ステートメントまたは INITIAL 属性の CALL オプションを実行する
- 関数参照を実行する

CALL または関数参照の実行前にも実行後にも、FETCH ステートメントまたは RELEASE ステートメントに制御を渡す必要はありません。

どちらのステートメントがプロシージャーをロードした場合も、CALL ステートメント、CALL オプション、または関数参照が実行されると、普通の方法でプロシージャーが呼び出されます。

プロシージャーがすでに主記憶域にロードされていても、 エラーにはなりません。 取り出されたプロシージャーは、プログラム全体の実行が完了するまで 主記憶域に残しておくことができます。 RELEASE ステートメントを使用すれば、そのプロシージャーが占有していたストレージをいつでも解放することができ、別の目的で使用することができます。

### 規則と機能

FETCH および RELEASE には、以下の規則と機能があります。

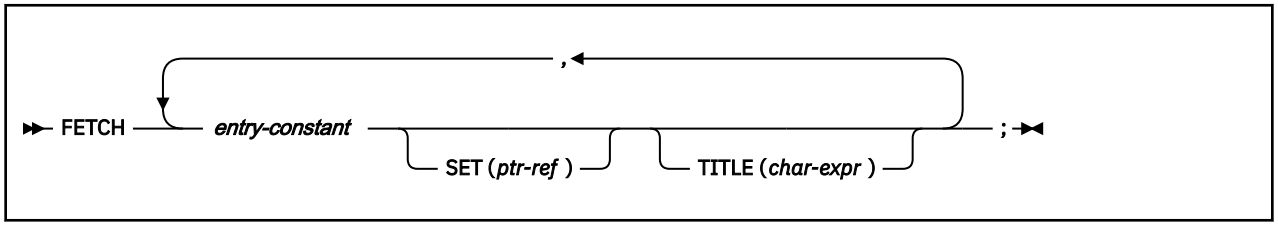
- 外部プロシージャーだけを取り出すことができます。
- EXTERNAL ファイルと CONDITION 条件は、アプリケーション全体 (主モジュールおよび取り出されたモジュール) で共有されます。 それ以外の外部変数は、単一モジュール内でのみ共用されます。
- 取り出されたプロシージャー中の STATIC 変数のストレージは、そのプロシージャーが入っているロード・モジュールがメモリーにロードされるときに割り振られます。 ロード・モジュールがメモリーにロードされるたびに、STATIC 変数には、それらの宣言が示す初期値が指定されます。

- FETCH ステートメントおよび RELEASE ステートメントは、 入り口定数を指定しなければなりません。 取り出されるプロシージャーの入り口定数は、 プロシージャーが取り出されている限り、 入り口変数に割り当てることができます。

## FETCH ステートメント

FETCH ステートメントは、指定されたプロシージャーが主記憶域に存在しているかを調べます。

主記憶域内に存在していないプロシージャーは、ディスクからロードされます。




***entry-constant***
取り出されるプロシージャーの名前で、オペレーティング・システムに 知られているものを指定します。 取り出し可能なプロシージャーに関連する考慮事項の詳細については、 「プログラミング・ガイド」を参照してください。

entry-constant (入り口定数) は、対応する CALL ステートメント、 CALL オプション、または関数参照内で使用する入り口定数と同じでなければなりません。

**SET**
フェッチするモジュールとフェッチされるモジュールの AMODE が両方とも AMODE(31) の場合は、ロードされるモジュールのエントリー・ポイントのアドレスに設定されるポインター参照 (*ptr-ref*) を指定します。それ以外の場合は、適切な AMODE でフェッチされるモジュールに制御を移すために使用されるグルー・コードのアドレスを指定します。 このオプションは、テーブル (実行できないロード・モジュール) を ロードするために使用することができます。 また、非 PL/I プロシージャーへ渡さなければならないアドレスを持つ、 取り出された項目に対しても使用することができます。

ロード・モジュールを後に RELEASE ステートメントによって解放して、さらにそのロード・モジュールに (ポインターを使用して) アクセスすると、予期しない結果が生じます。

**TITLE**
TITLE の場合、char-expr は任意の文字式、または文字式に変換可能な任意の式です。 TITLE が指定されると、指定されたロード・モジュール名が探索され、ロードされます。 指定されていない場合には、使用されるロード・モジュール名は、(表示されて いれば) 変数の EXTERNAL 属性で指定された環境名、または入り口定数名自体になります。

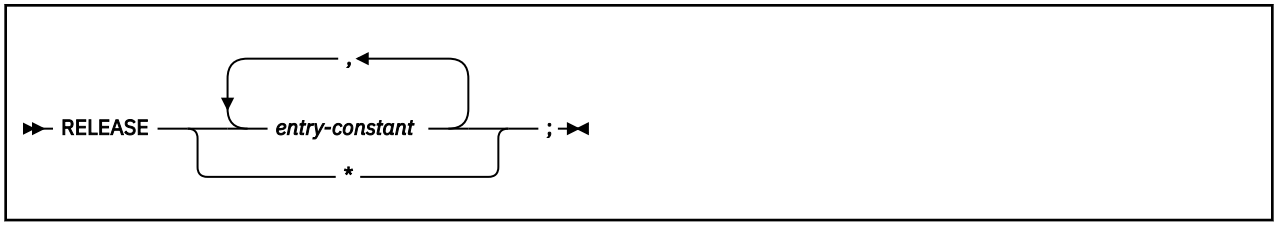
次の例を参照してください。

```
dcl A entry;
dcl B entry ext('C');
dcl T char(20) varying;
T = 'Y';

fetch A title('X');           /* X is loaded */
fetch A;                      /* A is loaded */
fetch B title('Y');           /* Y is loaded */
fetch B;                      /* C is loaded */
fetch B title(T);             /* Y is loaded */
```

タイトル・ストリングの詳細については、「プログラミング・ガイド」を参照してください。

## RELEASE ステートメント

RELEASE ステートメントは、指定した entry constant (入り口定数) が識別する プロシージャーによって占められた主記憶域を解放します。




**入り口定数 (entry constant)**
対応する CALL ステートメント、CALL オプション、関数参照、および FETCH ステートメントに使用する入り口定数と同じでなければなりません。 RELEASE * は、それ以前にフェッチされた すべての PL/I モジュールを解放します。 RELEASE ステートメントは、フェッチされた モジュール内部から実行してはなりません。

モジュールが同じ外部プロシージャー内で以前にフェッチされなかった場合、または既に解放されている場合、入り口定数は NULL ポインターになります。 解放が実行されない場合、メッセージは出されず、RELEASE ステートメントに続くステートメントの実行が続行されます。

以下の例を考えてください。ProgA と ProgB は、 ディスクに常駐しているプロシージャーの入り口名です。

```
        Prog:  procedure;

1          fetch ProgA;
2          call ProgA;
3          release ProgA;

4          call ProgB;
           go to Fin;

           fetch ProgB;
        Fin:  end Prog;
```

**1**
ProgA は、最初の FETCH ステートメントによって主記憶域にロードされます。

**2**
ProgA は、最初の CALL ステートメントに達したときに実行されます。

**3**
ProgA のストレージは、RELEASE ステートメントが実行されると解放されます。

**4**
ProgB は、このプロシージャーを参照する FETCH ステートメントが 実行されなくても、2 番目の CALL ステートメントに達すると、ロードされ、 実行されます。

FETCH ProgA ステートメントが省略された場合でも同じ結果になります。 RELEASE ステートメント内に ProgA を指定すると、 それを呼び出したときと同様に、CALL ProgA ステートメントが プロシージャーをロードします。

取り出されたプロシージャーは、 呼び出しプロシージャーとは別々にコンパイルされ、リンクされます。 FETCH ステートメント、RELEASE ステートメント、CALL ステートメント、 CALL オプション、および関数参照で指定される入り口定数は、ディスク上で 認識される名前でなければなりません。 これについては「プログラミング・ガイド」に説明があります。

**注 :** モジュールが解放される前に、モジュールは獲得したシステム・リソースをすべて解放する必要があります。 これには以下のアクションが含まれます (これらに限られません)。

- フェッチしたモジュールをすべて解放する。
- 開いたファイルをすべて閉じる。
- 割り振った制御ストレージをすべて解放する。

# サブルーチン

サブルーチン は、CALL ステートメントで呼び出される 内部または外部プロシージャーです。

サブルーチンの構文については、92 ページの『プロシージャー』を参照してください。

CALL ステートメントの引数は、 呼び出されたプロシージャーのパラメーターと関連付けられます。 サブルーチンは活動化されると、実行が開始します。 引数 (ゼロまたはそれ以上) は、入力専用、出力専用、 またはその両方が可能です。

サブルーチンは、RETURN ステートメントまたは END ステートメントで正常に終了します。 それから、制御が呼び込み側ブロックに戻ります。 サブルーチンは、98 ページの『プロシージャーの終了』で説明しているように、異常終了することもあります。

サブルーチン・プロシージャーは以下の条件を満たしていなければなりません。

- プロシージャー・ステートメントに RETURNS オプションを持っていないもの。
- 外部プロシージャーである場合には、RETURNS 属性の付いた入り口として宣言されないもの。
- 関数参照ではなく、CALL ステートメントを使用して呼び出されるもの。
- RETURN ステートメントを使用して結果の値を戻さないもの。

以下の例では、呼び込み側ブロックにとって外部および内部になる サブルーチンの呼び出しを示しています。

## 例 1

以下の例は、呼び込み側ブロックに対して外部であるサブルーチンの呼び出しを示しています。

```
        Prmain:  procedure;
                 declare Name character (20),
                 Item bit(5),
 4               Outsub entry;
 1               call Outsub (Name, Item);
        end Prmain;

 2     Outsub:  procedure (A,B);
                 declare A character (20),
                 B bit(5);
 3               put list (A,B);
        end Outsub;
```

**1**

Prmain の CALL ステートメントは、以下の場所でプロシージャー Outsub を呼び出し、**2**引数 Name および Item を使用します。

**2**

Outsub は、Prmain から 渡される Name と Item に そのパラメーターの A と B を関連付けます。 Outsub が実行されると、A への各参照は Name への参照として扱われます。 B への各参照は、Item への参照として扱われます。

**3**

put list (A,B) ステートメントは、 デフォルトの出力ファイル SYSPRINT へ Name と Item の値を送ります。

**4**

入り口定数として Outsub の宣言では、引数およびパラメーターの 属性が突き合わせられるので、パラメーター記述子は ENTRY 属性で指定する 必要はありません。 112 ページの『ENTRY 属性』も参照してください。

## 例 2

以下の例は、呼び込み側ブロックに対して内部であるサブルーチンの呼び出しを示しています。

```
        A:  procedure;
            declare Rate float (10),
                    Time float(5),
                    Distance float(15),
                    Sample file;
 1          call Readcm (Rate, Time, Distance, Sample);
```

```
3       Readcm:
2         procedure (W,X,Y,Z);
          declare W float (10),
                  X float(5),
                  Y float(15), Z file;
          get File (Z) list (W,X,Y);
          Y = W * X;
          if Y > 0 then
            return;
          else
            put list('ERROR READCM');
          end Readcm;

        end A;
```

**1**

引数 Rate、Time、Distance、および Sample がプロシージャー Readcm に渡されます。**3**また、パラメーター W、X、Y、および Z に関連付けられます。

**2**

W への参照は Rate への参照と同じ、X は Time と同じ、Y は Distance と同じ、Z は Sample と同じになります。

**3**

Readcm は A で明示的に 宣言されていないことに注意してください。 PROCEDURE ステートメントの指定に ENTRY 属性が暗黙に宣言されています。

## 組み込みサブルーチン

作成済みのプログラミング・タスクを備えた、組み込みサブルーチン を 使用することができます。 それらの組み込み名 は、BUILTIN 属性で明示的に宣言することができます。

BUILTIN 属性の情報または組み込み関数の説明について詳しくは、373 ページの『第 18 章 組み込み関数、疑似変数、およびサブルーチン』を参照してください。

## 関数

関数 とは、ゼロまたはそれ以上の引数を持つプロシージャーであり、 式で関数参照 を使用して呼び出されます。

関数参照は制御を関数プロシージャーに移します。関数プロシージャーは 制御と値を戻します。この値が式の計算で関数参照に置き換わることになります。 集合を戻すことはできません。 また、ENTRY 変数も、LIMITED 属性を持っていない限り戻すことはできません。 式の計算は引き続き行われます。

関数プロシージャーは以下の条件を満たしていなければなりません。

- プロシージャー・ステートメントに RETURNS オプションがあるもの。
- 外部プロシージャーである場合には、RETURNS 属性の付いた入り口として宣言されるもの。
- 関数参照を使用して呼び出されるもの。 CALL ステートメントは、戻り値に OPTIONAL 属性がある場合だけ、 それを呼び出すために使用することができます。 この場合には、戻り値は、戻りによって破棄されます。 RETURN の代わりに END を使用すると、予測しない結果になることがあります。
- RETURNS オプション内の属性と RETURNS 属性が合致するもの。
- RETURN ステートメントを使用して、制御と結果の値を戻すもの。

関数が呼び出されるときはいつでも、呼び出された式の引数は エントリー・ポイントのパラメーターと関連付けられます。 それから、制御はエントリー・ポイントに戻されます。 関数が活動化され、実行が開始します。

RETURN ステートメントは、関数を終了して、 その式に指定された値を呼び出し式に戻します。

98 ページの『プロシージャーの終了』で説明されているように、関数は異常終了することもあります。 異常終了した場合は、その関数を呼び出した式の計算は完了されずに、指定のステートメントに制御が移ります。

関数が定義されるので、引数リストを必要としない場合もあります。 そのような場合、式の中に書かれている外部関数名が関数参照として認識されるのは、その関数名が入り口名であると明示的に宣言されている場合だけです。 詳しくは、121 ページの『入り口呼び出しまたは入り口値』を参照してください。

**関連情報**

122 ページの『RETURN ステートメント』
RETURN ステートメントは、RETURN ステートメントが入っている サブルーチンまたは関数プロシージャーの実行を終了し、 呼び出しプロシージャーに制御を戻します。 制御は、呼び出し参照の直後の場所に戻されます。

# 例

以下の例は、呼び込み側ブロックに対して内部または外部である関数の呼び出しを示しています。

### 例 1

以下の例では、代入ステートメントに Sprod 関数への 参照が含まれています。

```
        Mainp: procedure;
               get list (A, B, C, Y);
 1             X = Y**3+Sprod(A,B,C);

 2      Sprod: procedure (U,V,W)
               returns (bin float(21));
               dcl (U,V,W) bin float(53);
               if U > V + W then
 3              return (0);
               else
 3              return (U*V*W);
         end Sprod;
```

**1**
Sprod が呼び出されると、引数 A、B、および C は、パラメーター U、V、および W に関連付けられます。 **2**、それぞれ。

**2**
RETURNS がプロシージャー・ステートメントに 指定されているので、Sprod は関数です。 これは、内部関数であるため、明示的な入り口宣言は必要ありません。 Sprod が外部関数であれば、Mainp は RETURNS を指定した入り口宣言を含みます。

**3**
Sprod は、ゼロを戻すか、 または U*V*W が表す値を戻します。 制御は Mainp 内の式に戻されます。 戻り値は、関数参照の値として用いられ、式の計算は続きます。

### 例 2

```
      Mainp:  procedure;
              dcl Tprod entry (bin float(53),
                               bin float(53),
                               bin float(53),
                               label) external
                returns (bin float(21));
              get list (A,B,C,Y);
 1             X = Y**3+Tprod(A,B,C,Lab1);
      Lab1:  call Errt;
      end Mainp;
```

```
 1      Tprod:  procedure (U,V,W,Z)
                returns (bin float(21));
                dcl (U,V,W) bin float(53);
                declare Z label;

 2              if U > V + W then
                    go to Z;
 3              else
                    return (U*V*W);
          end Tprod;
```

**1**
Tprod が呼び出されると、 Lab1 はパラメーター Z と関連付けられます。

**2**
U が V + W よりも大きい場合には、Lab1 というラベルの付いた ステートメントで Mainp に制御を戻します。 割り当て先の評価**1**は廃止されます。

**3**
U が V + W より大きくない場合には、 U*V*W は計算されて、通常の方法で Mainp に戻されます。 割り当て先の評価**1**は続行されます。

Tprod は外部プロシージャーであることに注意してください。 RETURNS が含まれている Mainp には明示的な入り口宣言があります。

## 組み込み関数

プログラマーが作成した関数プロシージャーを 使用できるようにするため、PL/I では、組み込み関数 のセットを提供しています。

組み込み関数には、共通して使用される算術関数のほかに、 ストレージなどを使用してストリングおよび配列を処理するための関数があります。 組み込み関数の呼び出し方法は、プログラマー定義の関数の呼び出しと同じです。 しかし、プログラマー定義の関数は 1 つのエレメント値しか返せませんが、多くの組み込み関数は値の配列を返すことができます。 組み込み関数の組み込み名は、BUILTIN 属性で明示的に宣言することができます。

**関連情報**

373 ページの『組み込み関数、疑似変数、およびサブルーチン』
多くの共通タスクが組み込み関数、サブルーチン、および疑似変数の形で使用可能です。 これらの共通タスクを使用することにより、作成するコードの数が減り、信頼性の高い迅速な処理が できます。 この章では、PL/I プログラムで使用できる組み込み関数、サブルーチン、および疑似変数について説明します。

# プロシージャーへの引数の引き渡し

関数またはサブルーチンが呼び出されると、パラメーターは、 左から右の順に、渡される引数と関連付けられます。

一般的に、以下の規則が適用されます。

- 計算データの引数を、計算データ・タイプのパラメーターに渡すことができます。
- プログラム制御データの引数は、同じタイプのパラメーターに 渡さなければなりません。ただし、次の場合はこの例外となります。
  - ポインターとオフセットは、相互に渡されます。
  - LIMITED ENTRY は ENTRY に渡されますが、ENTRY は LIMITED ENTRY に渡されません。
  - ラベル定数の配列は、引数として使用することはできません。

構造体引数が定数エクステントとともに宣言されていないかぎり、構造体から派生する一時的な集合体を必要とする引数は、使用できません。

引数リストの式は、サブルーチンまたは関数が呼び出される前に 呼び込み側ブロックで計算されます。 パラメーターには、それと関連付けられるストレージはありません。 パラメーターは、呼び出されたプロシージャーが、呼び出し側のプロシージャーに割り振られているストレージにアクセスできるようにするための手段を提供するためだけのものです。

## BYVALUE および BYADDR の使用

引数が値 (BYVALUE) で渡されない限り、引数への参照 (値ではない) は、一般的にサブルーチンまたは関数に渡されます。 これは、参照または BYADDR によって引数を渡すことによって認識されます。

プロシージャー内のパラメーターへの参照は、対応する引数への参照になります。 パラメーターの値への変更は、実際には、対応する引数の値への変更になります。 ただし、これは常に可能ではなく、好ましいものではありません。 例えば、定数は呼び出されたプロシージャーによって 変更されるべきではありませ

ん。 変更すべきではない引数には、 元の引数の値を含む仮引数 が渡されます。 パラメーターへの参照は、仮引数への参照であり、元の引数に対するものではありません。

BYADDR を指定すると、コンパイラーによって、パラメーター・リストに対応する 引数のアドレスが書き込まれます。 BYVALUE を指定すると、パラメーター・リストに 引数の値が書き込まれます。

BYVALUE を指定すると、仮引数は作成されませんが、呼び出された ルーチンで対応するパラメーターに行われたすべての変更は、呼び出しルーチンでは 不可視になります。これは、仮引数の場合も同様です。

BYVALUE は、AREA、PICTURE、WIDEPIC、VARYING、および VARYINGZ では無効です。

BYVALUE 引数は、正当にレジスターで受け渡しできるものである必要がある。 したがって、そのタイプは次のリスト内のいずれかでなければなりません。

- REAL FIXED BIN
- REAL FLOAT
- POINTER
- OFFSET
- HANDLE
- LIMITED ENTRY
- FILE
- ORDINAL
- CHAR(1)
- WCHAR(1)
- ALIGNED BIT(*n*) (*n* は 8 以下)

## INONLY、INOUT、および OUTONLY の使用

引数が、属性 INONLY または OUTONLY で宣言されていない限り、引数は INOUT となり、その引数は、渡される前に値を持ち、呼び出されたコードによって (おそらく) 変更されるものと想定されます。

引数を INONLY として宣言した場合、引数が渡される前に引数は値を持ち、呼び出されたコードによっては変更されないものと想定されます。 したがって、仮引数をそのような引数に対して作成する必要はありません。

引数を OUTONLY として宣言した場合、引数が渡される前に値を持たないが、呼び出されたコードによって設定されるものと想定されます。

BYVALUE 属性は、INONLY 属性を暗黙指定します。 したがって、属性 OUTONLY と BYVALUE は競合するため、同じ引数に両方を指定することはできません。 ただし、ASSIGNABLE 属性を BYVALUE 属性と同時に使用することは可能です。

INONLY 属性を持つが、明示的な ASSIGNABLE 属性を持たないパラメーターには、NONASSIGNABLE 属性が与えられます。 ASSIGNABLE 属性を BYADDR INONLY 属性と同時に使用するのは非常に危険です。

これらの属性を明示的に使用すると、コードがより自己記述的になります。 さらに、そうすることによって、コンパイラーがより良いコードを生成でき、未初期化のおそれのある変数のレポートでより正確になるようにすることができます。

対応するパラメーターに INOUT または OUTONLY 属性があるときに、INONLY 属性を持つ変数を引数としてエントリーに渡すと、その変数を変更することができ、これによってコードが無効になります。 例えば、以下のコードは保護例外を引き起こす可能性があります。

```
call test( 17 );

test: proc( x );
  dcl x fixed bin(31) INONLY;
  dcl e ext entry( INOUT fixed bin(31) );
  call e(x);
end;
```

# 仮引数

仮引数は、引数の値を保持するために自動的に作成される一時ストレージの一部です。

仮引数は、引数が以下のいずれかの場合に作成されます。

- 定数 (パラメーターが INONLY 属性を持つ場合を除く)
- 演算子、括弧、または関数参照を使用する式
- データ属性、位置合わせ属性、または接続属性が パラメーターに宣言された属性と異なる変数。

  境界、長さ、またはサイズだけが異なり、 しかもこれらがアスタリスクを使用して宣言されている場合、および定数以外の式を使用して被制御パラメーターの エクステントを定義している場合には、 上記は制御されていないパラメーターには適用されません。 定数以外の式を使用して被制御パラメーターのエクステントを定義している場合、 引数とパラメーターのエクステントは一致すると見なされます。

  PICTURE 属性を持つ引数とパラメーターの場合、反復因数が適用されたあと、 ピクチャー指定が正確に一致しない限り、仮引数が作成されます。 唯一の例外は、位取り係数に正符号が付いている引数またはパラメーターは、正符号の付いていないパラメーターまたは引数と一致することです。
- 調整可能な長さやサイズを持つストリングまたは区域 (長さやサイズが一定の 制御されないパラメーターに関連付けられるもの) なお、RULES(LAXCTL) コンパイラー・オプションの場合は、CONTROLLED ストリングまたは AREA のエクステントは常に変更可能です。 一方、RULES(NOLAXCTL) コンパイラー・オプションの場合は、定数として宣言されていない限り、CONTROLLED ストリングまたは AREA のエクステントは変更可能です。

## 仮引数属性の派生

PL/I は、仮引数の属性を以下のように派生させます。

- 内部プロシージャー内の関連パラメーターに関して宣言された属性から
- 外部入り口の宣言にある関連パラメーターのパラメーター記述子に指定された属性から

  このパラメーターの記述子がなかった場合は、定数または式の属性が使用されます。
- 配列境界の引数、ストリングの長さ、または区域のサイズのエクステント (宣言でアスタリスクによって指定されている場合) から

## 仮引数の規則

以下の規則が、仮引数に適用されます。

- パラメーターがエレメント (つまり、 構造体または配列のいずれでもない変数) である場合には、 引数は要素式でなければなりません。
- VARYING、VARYING4、VARYINGZ のストリング・エレメントを、長さが未定義である (つまり長さがアスタリスクで指定されている) NONVARYING パラメーターに渡した場合、元の引数の長さを持つ仮引数が作成されます。
- 引数として渡される入り口変数は位置合わせされたものと見なされるため、 引数とパラメーターの位置合わせだけが異なる場合は、 仮引数は作成されません。 入り口パラメーターの総称名引数の説明については、119 ページの『総称入り口』 を参照してください。
- パラメーターがプログラム制御タイプ (ロケーターを除く) である場合、 引数は同じデータ・タイプの参照でなければなりません。
- パラメーターがロケーター (ポインターまたはオフセット) である場合には、 引数はロケーターでなければなりません。 タイプが異なる場合には、仮引数が作成されます。 オフセット・パラメーターのパラメーター記述子で、関連区域を指定してはなりません。
- 制御されていないパラメーターを、 任意のストレージ・クラスの引数と関連付けることができます。 ただし、複数の引数の世代が存在するときは、パラメーターは呼び出し時に存在する世代とだけ関連付けられます。

- パラメーターが被制御状態のときは、このことを ENTRY 宣言の パラメーター記述子に明示的に指定しなければなりません。 さらに、被制御パラメーターは常に、対応する被制御引数 (添え字がなく、構造体の要素ではなく、仮引数を作成しない) を持っている必要があります。

  複数の引数の世代が呼び出し時に存在している場合には、 存在している世代のすべてのスタックにパラメーターは対応します。 その結果として呼び出し時に、被制御パラメーターは、 対応する引数の現行世代を表します。 関連する引数の割り振りスタックの操作を行うために、 被制御パラメーターは割り振られ、呼び出されたプロシージャー内で解放されます。

  被制御パラメーターのエクステントがアスタリスクまたは制限なしの式として 指定されている場合には、もとの宣言は、制限なしの式として宣言された エクステントを持たなければなりません。

## MAIN プロシージャーへの引数の引き渡し

主プロシージャーの PROCEDURE ステートメントは、パラメーター・リストを持つことができます。 PL/I では、そのようなパラメーターに特定考慮事項は必要ありません。 ただし、呼び出し側プログラムの引数の要件 (例えば、割り当てのターゲットとしてパラメーター を使用してはならないなど) に注意する必要があります。

呼び出し側プログラムがオペレーティング・システムであり、呼び出し対象プログラムが SYSTEM(MVS) コンパイラー・オプションを指定してコンパイルされている場合は、以下の規則が適用されます。

- 単一引数がメイン・プロシージャーに渡され、そのパラメーターは CHARACTER VARYING と宣言される必要があります。
- このパラメーターの現行長は、実行時の引数の長さと等しくセットされます。 そのため、次の例では、ストレージは引数の現行長に対してのみ割り振られます。

```
Tom:  proc (Param) options (main);
   dcl Param char(*) varying;
```

- このパラメーターの内容は、OPTIONS(MAIN) とともに指定した 2 番目のオプションによって異なります。
  - OPTIONS(MAIN NOEXECOPS) を指定する場合、 オペレーティング・システムによって PL/I に渡されるストリングは、 そのままプログラムに渡されます。 NOEXECOPS を指定することをお勧めします。
  - OPTIONS(MAIN) のみを指定すると、 オペレーティング・システムによって PL/I に渡されるストリングは、 最初に '/' を含むストリングまで、すべての文字を削除したものになります。 つまり、ストリングに '/' が含まれていない場合、 プログラムはヌル・ストリングを受け取ることになります。

# 開始ブロック

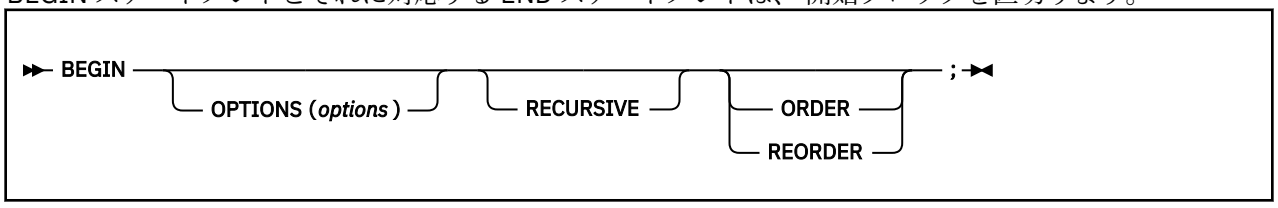
開始ブロックは、BEGIN ステートメントとそれに対応する END ステートメントによって区切られている一連のステートメントです。

**例**

```
B:  begin;
statement-1
statement-2
⋮
statement-n
end B;
```

## BEGIN ステートメント

BEGIN ステートメントとそれに対応する END ステートメントは、 開始ブロックを区切ります。



**OPTIONS (option)**
開始ブロック・オプションについては、123 ページの『OPTIONS オプションとその属性』を参照してください。

**RECURSIVE**
99 ページの『再帰的プロシージャー』を参照してください。

**ORDER または REORDER**
ORDER と REORDER は、プロシージャーまたは開始ブロックに指定される最適化オプションです。ORDER と REORDER の使用の詳細については、123 ページの『OPTIONS オプションとその属性』の"ORDER または REORDER"を参照してください。

## 開始ブロックの活動化

開始ブロックは、通常の流れによって、あるいはステートメント IF、ON、 WHEN、または OTHERWISE のユニットとして活動化されます。

GO TO ステートメントを使用して、ラベルのついた BEGIN ステートメントに 制御権を移動することができます。

## 開始ブロックの終了

開始ブロックは、プロシージャー参照以外のなんらかの方法で、 制御が別のアクティブのブロックに渡されたときに終了します。

これらの方法については、下のリストに説明があります。

- 開始ブロックの END ステートメントを実行すること。 ブロックが ON ユニットであるとき以外は、物理的に END のあとのステートメントで制御は継続します。
- 開始ブロック内 (あるいは開始ブロックにとって内部である 任意のブロック内) で GO TO ステートメントを実行すること。 これにより、制御権はブロックの外部に移動します。
- STOP または EXIT ステートメントを実行すること。
- 制御権が RETURN ステートメントに到達すること。これにより、 制御権は開始ブロックの外部および開始ブロックを含むプロシージャーの外部に移動します。

GO TO ステートメントで指定した移動先の場所が、終了しようとしているブロックを直接に活動化したのではないブロックに含まれている場合には、GO TO ステートメントが実行されると他のブロックも終了します。 つまり、活動化の順序においてその 2 つのブロックの間にあるすべてのブロックが終了します。 この例については、98 ページの『プロシージャーの終了』を参照してください。

# 入り口データ

入り口データは、入り口定数または入り口変数の値です。

入り口定数は、 PROCEDURE または ENTRY ステートメントに接頭部として付けられた名前、 または VARIABLE 属性ではなく ENTRY 属性で宣言された名前です。 これは、入り口変数に割り当てられます。 以下の例では、P、E1、および E2 は入り口定数、Ev は入り口変数です。

```
P:  procedure;
    declare Ev entry variable,
     (E1,E2) entry;


Ev = E1;
call Ev;
Ev = E2;
call Ev;
```

最初の CALL ステートメントは、エントリー・ポイント E1 を呼び出します。 2 番目の CALL はエントリー・ポイント E2 を呼び出します。

以下の例では、添え字のついた入り口変数の F(5) を宣言しています。

A、B、C、D、 および E という 5 つの入り口は X、Y、 および Z というパラメーターを使用してそれぞれ呼び出されます。

```
declare (A,B,C,D,E) entry,
declare F(5) entry variable initial (A,B,C,D,E);
do I = 1 to 5;
  call F(I) (X,Y,Z);
end;
```

内部プロシージャーのエントリー・ポイントである入り口定数が 入り口変数に割り当てられるときに、 入り口定数が内部にあったブロックがアクティブのままである間は、割り当てられた値は有効なままです (再帰的プロシージャーの場合は、現行のままです)。

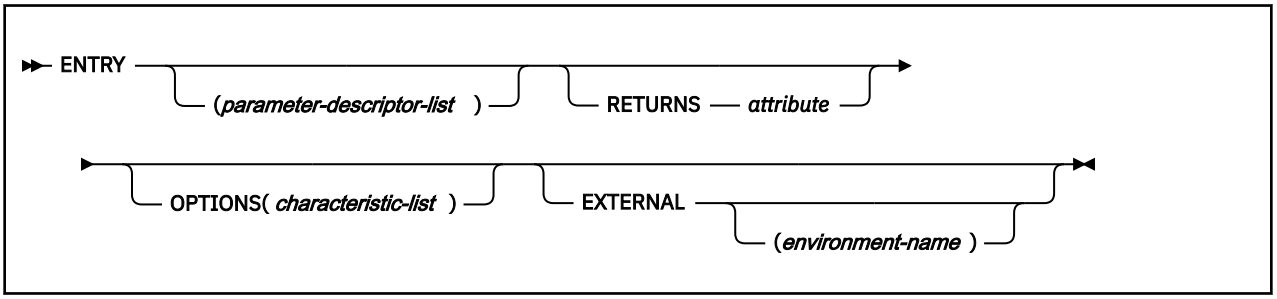
# 入り口定数

PROCEDURE または ENTRY ステートメントにラベル接頭部を指定すると、入り口定数を明示的に宣言することになります。

パラメーター記述子リストは、パラメーター宣言、 および (ある場合) デフォルトから得られます。

外部入り口定数は、明示的に宣言する必要があります。

- この宣言は、外部プロシージャーへの入り口点を定義します。
- この宣言はオプションで、入り口点のパラメーター記述子リスト (パラメーターの数とその属性) があればそれを指定します。
- この宣言は、入り口が関数である場合にプロシージャーによって返される値の属性を指定します。




属性は、任意の順序で示すことができます。

**ENTRY 属性**
ENTRY 属性の完全な構文については、112 ページの『ENTRY 属性』を参照してください。

**OPTIONS 属性**
OPTIONS 属性の完全な構文については、123 ページの『OPTIONS オプションとその属性』を参照してください。

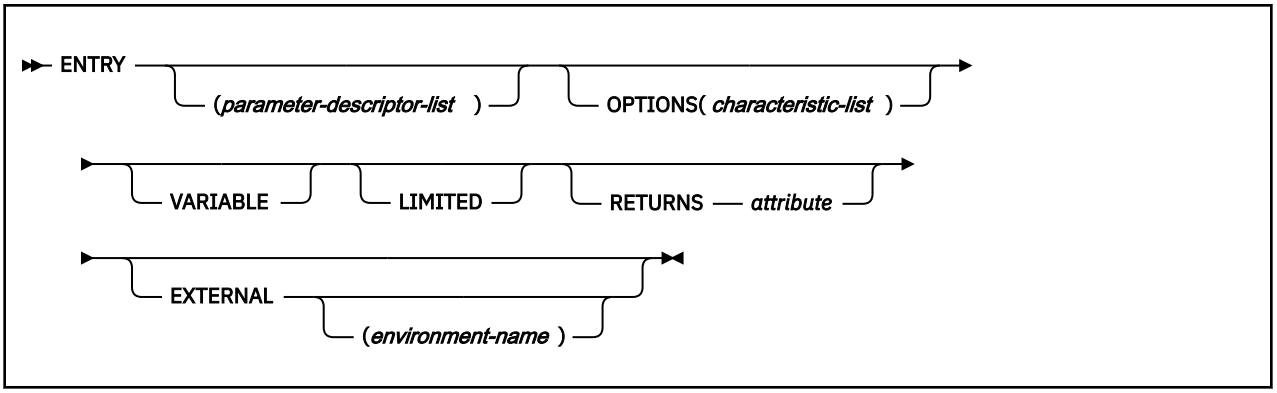
**RETURNS 属性**
RETURNS 属性の完全な構文については、132 ページの『RETURNS オプションとその属性』を参照してください。

**EXTERNAL 属性**
*environment-name* (環境名) を指定しないと、その名前は宣言と同じものになります。 EXTERNAL 属性の詳細については、152 ページの『INTERNAL 属性と EXTERNAL 属性』を参照してください。

# 入り口変数

入り口変数には、内部および外部入り口値の両方が含まれます。 入り口変数は、集合の一部になることができます。

構造化および配列の次元属性については、170 ページの『配列』および 174 ページの『構造体』を参照してください。




オプションは、任意の順序で示すことができます。

**ENTRY 属性**
112 ページの『ENTRY 属性』を参照してください。

**OPTIONS 属性**
123 ページの『OPTIONS オプションとその属性』を参照してください。

**VARIABLE 属性**
VARIABLE 属性は、入り口変数として名前を設定します。 この変数には、入り口定数および変数が含まれます。 構文情報については、45 ページの『VARIABLE 属性』を参照してください。

**LIMITED 属性**
118 ページの『LIMITED 属性』を参照してください。

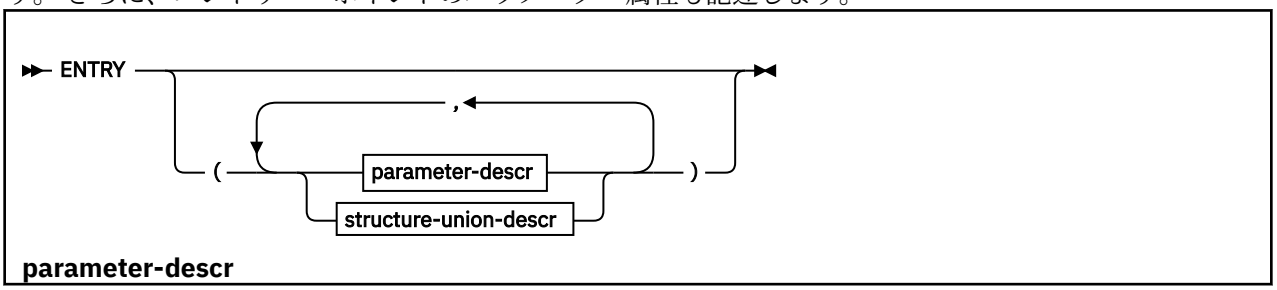
**RETURNS 属性**
132 ページの『RETURNS オプションとその属性』を参照してください。

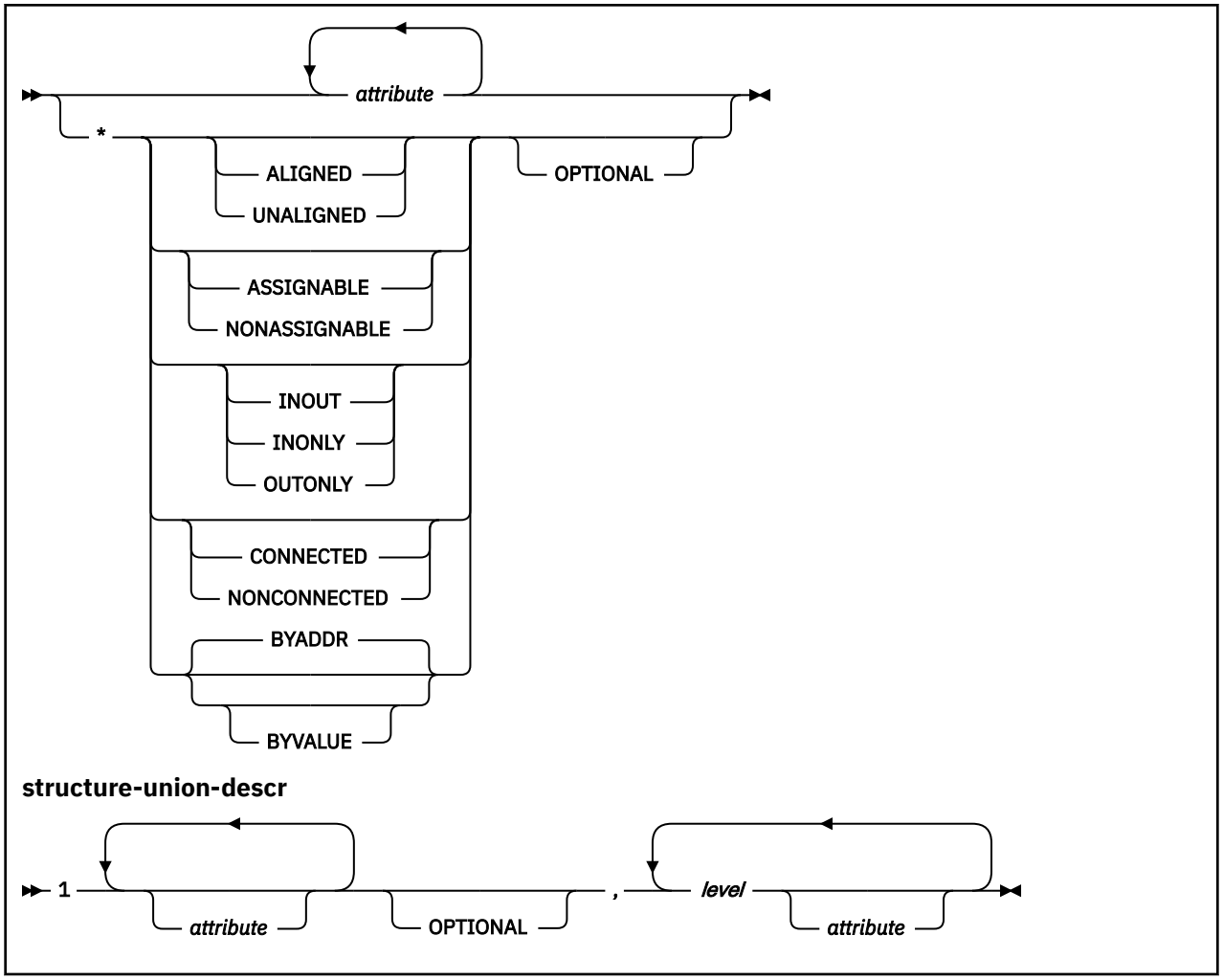
**EXTERNAL 属性**
150 ページの『宣言の有効範囲』を参照してください。

64 ビットでは、ENTRY 変数はすべて、LIMITED 属性を持つかどうかに関わらず、サイズが 8 バイトであり、デフォルトでダブルワード位置合わせされます。

## ENTRY 属性

ENTRY 属性は、宣言される名前が、 外部入り口定数または入り口変数のいずれかであることを指定します。 さらに、エントリー・ポイントのパラメーター属性も記述します。






**ENTRY**

パラメーター記述子リストを使用しない ENTRY 属性は、 RETURNS 属性によって暗黙指定されます。

**parameter-descr (parameter-descriptor)**

パラメーター記述子リストは、関連する外部入り口定数または入り口変数の パラメーターの属性を説明するために指定することができます。 引数とパラメーター属性の突き合わせ、および仮引数の作成用に使用されます。

パラメーター記述子リストが指定されない場合には、引数属性のデフォルトと パラメーター属性を突き合わせなければなりません。 そのため、引数属性とパラメーター属性を突き合わせない場合には、パラメーター記述子リストを指定しなければなりません。

それぞれのパラメーター記述子は、呼び出されたエントリー・ポイントの 1 つの パラメーターと対応し、(もし指定されていれば) そのパラメーターの属性が指定されます。

パラメーター記述子は、説明されているパラメーターと同じ順序で 示さなければなりません。 記述子がなければ、引数のデフォルトとパラメーターが突き合わされます。

パラメーターの記述子が不要な場合には、 記述子がないということをアスタリスクで示さなければなりません。 次の例を参照してください。

| | |
|---|---|
| `entry(character(10),*,*,fixed dec)` | 4 つの引数を示します。 |
| `entry(*)` | 1 つの引数を示します。 |
| `entry( )` | 入り口名は引数を持ってはならないということを指定します。 |
| `entry` | 任意の数の引数を持つことができるということを指定します。 |

| | |
|---|---|
| `entry(float binary,*)` | 2 つの引数を示します。 |

**属性**
使用可能な属性は、17 ページの『データ属性』にリストされた任意のデータ属性です。 属性は、パラメーター記述子内に任意の順に表示することができます。 配列パラメーター記述子の場合は、 次元属性が最初に指定されなければなりません。

*****
アスタリスクは、そのパラメーターにどのデータ・タイプでも 使用できることを指定します。 以下の属性のみが、アスタリスクの後に有効な属性です。

- ALIGNED または UNALIGNED
- ASSIGNABLE または NONASSIGNABLE
- BYADDR または BYVALUE
- CONNECTED または NONCONNECTED
- INONLY、INOUT、または OUTONLY
- OPTIONAL

変換は行われません。

**OPTIONAL**
114 ページの『OPTIONAL 属性』を参照してください。

**structure-union-descr (structure-union-descriptor)**
構造体共用体記述子の場合は、 記述子レベル番号がパラメーターのものと同じである 必要はありませんが、構造化は同じでなければなりません。 特定レベルの属性は、任意の順序で示すことができます。

アスタリスクが指定されている場合には、デフォルトは適用されません。 例えば、以下の宣言のデフォルトは、2 番目のパラメーターにだけしか適用されません。

```
dcl X entry(* optional, aligned); /* defaults applied for 2nd parm */
```

パラメーター記述子のエクステント (長さ、サイズ、および境界) は、 定数またはアスタリスクとして指定されなければなりません。 被制御パラメーターは、アスタリスクを持たなければなりません。

RETURNS 属性は、ENTRY 属性を暗黙指定します。 次の例を参照してください。

**パラメーター記述子の例**

```
Test:  procedure (A,B,C,D,E,F);


 declare A fixed decimal (5),
         B float binary (21),
         C pointer,
         1 D,
           2 P,
           2 Q,
             3 R fixed decimal,
         1 E,
           2 X,
           2 Y,
             3 Z,
         F(4) character (10);
 end Test;
```

**記述子の宣言の例**

```
declare Test entry
            (decimal fixed (5),
            binary float (21),
           *,
            1,
             2,
             2,
              3 decimal fixed,
             *,
            (4) char(10));
```

上の例では、パラメーター C および構造体パラメーター E には記述子がありません。

# OPTIONAL 属性

OPTIONAL は、パラメーター記述子リストの一部として、またはパラメーター宣言内の属性として指定できます。

引数にアスタリスクを指定することによって、 OPTIONAL 引数を呼び出しおよび関数参照で省略することができます。 引数リストのどの位置にある項目でも (最後の項目でも構わない) 省くことができます。 ただし、省略される項目は引数として数えられます。 入り口にその項目が含まれていても、 引数の数は入り口で許可されている最大数を超えてはなりません。

項目が LIMITED ENTRY でない限り、同じ項目に OPTIONAL と BYVALUE を使用することはできません。

受け取りプロシージャーは、OMITTED または PRESENT 組み込み関数を使用して、 OPTIONAL パラメーター/引数が入り口の呼び出しで省略されたかどうかを判別することができます。

省略された OPTIONAL パラメーターを入り口の引数として渡すことができますが、これは、その入り口の宣言内の対応するパラメーターもまた OPTIONAL である場合に限られます。

ENTRY 宣言内の最終パラメーターが OPTIONAL として宣言された場合、その ENTRY はそれらのパラメーターを完全に省略して呼び出すことができます。 該当する数のアスタリスクを指定する必要もありません。 例えば、ENTRY が 5 つのパラメーターを持つものとして宣言され、その最後の 2 つに OPTIONAL 属性がある場合、その ENTRY は 3 つ、4 つ、または 5 つの引数を指定して呼び出すことができます。

呼び出された ENTRY が明示的に宣言されている場合、および呼び出された ENTRY がネストされたサブプロシージャーである場合には、そのような末尾の OPTIONAL パラメーターを省略できます。 なお、ENTRY に OPTIONS(ASSEMBLER) 属性が指定されていない限り、生成されたコードでは、省略したパラメーターに NULL ポインターが入ります。

有効な呼び出しステートメントおよび無効な呼び出しステートメント では、 プロシージャー Vrtn に対して有効な CALL ステートメントと無効な CALL ステートメントの両方を示します。

**有効な呼び出しステートメントおよび無効な呼び出しステートメント**

```
   Caller: proc;
     dcl Vrtn entry (
              fixed bin,
              ptr optional,
              float,
              * optional);

/*  The following calls are valid:  */

     call Vrtn(10, *, 15.5, 'abcd');
     call Vrtn(10, *, 15.5, *);
     call Vrtn(10, addr(x), 15.5, *);
     call Vrtn(10, *, 15.5);
     call Vrtn(10, addr(x), 15.5);

/*  The following calls are invalid:  */

     call Vrtn(*, addr(x));
     call Vrtn(10,addr(x));
     call Vrtn(10);
     call Vrtn;
   end Caller;

   Vrtn: proc (Fb, P, Fl, C1);
     dcl Fb fixed bin,
         P ptr optional,
         Fl float,
         C1 char(8) optional;

     if ¬omitted(C1) then display (C1);
     if ¬omitted(P) then P=P+10;
end;
```

Vrtn は、OPTIONAL パラメーターが省略されているかどうかを判別し、該当する処置を行います。

**関連情報**

491 ページの『OMITTED』
OMITTED が含まれているプロシージャーの呼び出しで、*x* という名前のパラメーターが省略される場合、OMITTED は '1'B である BIT(1) 値を返します。

# LIST 属性

LIST は、パラメーター記述子リストの最後のパラメーターで、またはプロシージャーに対する最後のパラメーターの属性として指定できます。

LIST 属性が入り口宣言で指定された場合、 それは、ゼロまたはさらに追加の引数がそのエントリー・ポイントに渡される可能性があることを示します。 例えば、次の宣言では、 vararg が 1 文字の varyingz パラメーターを指定して 呼び出されなければならないこと、 および任意の数のほかのパラメーターを指定して呼び出されなければならないことを指定します。

```
dcl vararg  external
            entry( list byaddr char(*) varz nonasgn )
            options( nodescriptor byvalue );
```

LIST 属性がプロシージャーの最後のパラメーターの宣言中で指定される場合、 それは、ゼロまたはさらに追加の引数がそのプロシージャーに 渡された可能性があることを示します。

LIST 属性が指定された場合、記述子は許可されず、OPTIONS(NODESCRIPTOR) をその PROCEDURE ステートメントおよびその対応する ENTRY 宣言に対して指定する必要があります。

これらの追加パラメーターの最初のパラメーターのアドレスは、VARGLIST 組み込み関数によって取得できます。 以下のように、このアドレスは、追加パラメーターのアドレスを 取得するのに使用されることがあります。

- このプロシージャーに対する追加パラメーターが値によって (BYVALUE) 渡された場合、この初期アドレスを VARGSIZE 組み込み関数によって返された値ずつ連続的に増やすことで、追加パラメーターのアドレスが返されます。
- このプロシージャーに対する追加パラメーターが参照によって (BYADDR) 渡された場合、この初期アドレスをポインターのサイズずつ連続的に増やすことで、追加パラメーターのアドレスが返されます。

### 例

以下の (printf の簡単なバージョンを実現する) サンプル・プログラムは、 LIST 属性の使用法を示します。 ルーチン varg1 は、BYVALUE パラメーターを持つ変数引数リストの処理方法を示しています。 varg2 は、BYADDR パラメーターを持つこのようなリストの処理方法を示しています。

***LIST* 属性を説明するサンプル・プログラム**

```
*process rules(ans) dft(ans) gn;
 vararg: proc options(main);

   dcl i1      fixed bin(31) init(1729);
   dcl i2      fixed bin(31) init(6);
   dcl d1      float bin(53) init(17.29);

   dcl varg1   ext entry( char(*) varz byaddr list )
               options(byvalue nodescriptor );
   dcl varg2   ext entry( char(*) varz byaddr list )
               options(byaddr nodescriptor );

   call varg1( 'test byvalue' );
   call varg1( 'test1 parm1=%i', i1 );
   call varg1( 'test2 parm1=%i parm2=%i', i1, i2 );
   call varg1( 'test3 parm1=%d', d1 );

   call varg2( 'test byaddr' );
   call varg2( 'test1 parm1=%i', i1 );
   call varg2( 'test2 parm1=%i parm2=%i', i1, i2 );
   call varg2( 'test3 parm1=%d', d1 );
 end;

*process ;
  varg1:
    proc( text )
    options( nodescriptor byvalue );

    dcl text    list byaddr nonasgn varz char(*);

    dcl jx      fixed bin;
    dcl iz      fixed bin;
    dcl ltext   fixed bin;
    dcl ptext   pointer;
    dcl p       pointer;
    dcl i       fixed bin(31) based;
    dcl d       float bin(53) based;
    dcl q       float bin(64) based;
    dcl chars   char(32767) based;
    dcl ch      char(1) based;
```

```
    ptext = addr(text);
    ltext = length(text);
    iz = index( substr(ptext->chars,1,ltext), '%' );
    p = varglist();
    do while( iz > 0 );
      if iz = 1 then;
      else
        put edit( substr(ptext->chars,1,iz-1) )(a);
      ptext += iz;
      ltext -= iz;
      select( ptext->ch );
        when( 'i' )
          do;
            put edit( trim(p->i) )(a);
            p += vargsize( p->i );
          end;
        when( 'd' )
          do;
            put edit( trim(p->d) )(a);
            p += vargsize( p->d );
          end;
      end;
      ptext += 1;
      ltext -= 1;
      if ltext <= 0 then leave;
      iz = index( substr(ptext->chars,1,ltext), '%' );
    end;
    if ltext = 0 then;
    else
      put edit( substr(ptext->chars,1,ltext) )(a);
    put skip;
  end;
```

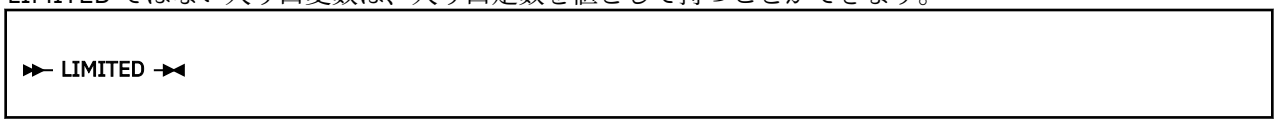
***LIST* 属性を説明するサンプル・プログラム**

```
*process ;
  varg2:
    proc( text )
    options( nodescriptor byaddr );

    dcl text    list byaddr nonasgn varz char(*);

    dcl jx      fixed bin;
    dcl iz      fixed bin;
    dcl ltext   fixed bin;
    dcl ptext   pointer;
    dcl p       pointer;
    dcl p2      pointer based;
    dcl i       fixed bin(31) based;
    dcl d       float bin(53) based;
    dcl q       float bin(64) based;
    dcl chars   char(32767) based;
    dcl ch      char(1) based;

    ptext = addr(text);
    ltext = length(text);
    iz = index( substr(ptext->chars,1,ltext), '%' );
    p = varglist();
    do while( iz > 0 );
      if iz = 1 then;
      else
        put edit( substr(ptext->chars,1,iz-1) )(a);
      ptext += iz;
      ltext -= iz;
      select( ptext->ch );
        when( 'i' )
          do;
            put edit( trim(p->p2->i) )(a);
            p += size( p );
          end;
        when( 'd' )
          do;
            put edit( trim(p->p2->d) )(a);
            p += size( p );
          end;
      end;
      ptext += 1;
      ltext -= 1;
      if ltext <= 0 then leave;
      iz = index( substr(ptext->chars,1,ltext), '%' );
    end;
    if ltext = 0 then;
    else
      put edit( substr(ptext->chars,1,ltext) )(a);
    put skip;
  end;
```

***LIST* 属性を説明するサンプル・プログラム**

## LIMITED 属性

LIMITED 属性は、入り口変数がネストされていない入り口定数だけを値として持つことを表します。
LIMITED ではない入り口変数は、入り口定数を値として持つことができます。




### 例

LIMITED 静的入り口変数は、ネストされていない入り口定数の値を使用して 初期設定できます。こうすれば、より効率的なコードを生成できます。 また、LIMITED 静的入り口変数は、LIMITED ではない入り口変数よりも小さいストレージを使用します。

```
Example: proc options(reorder reentrant);
dcl (Read, Write) entry;
dcl FuncRtn(2) entry limited
static init (Read, Write);
```

```
dcl (Prt1) entry;
dcl PrtRtn(2) entry variable limited
static init (Prt1,       /*  legal    */
             Prt2);      /*  illegal  */
Prt2: proc;
⋮
end Prt2;
end Example;
```

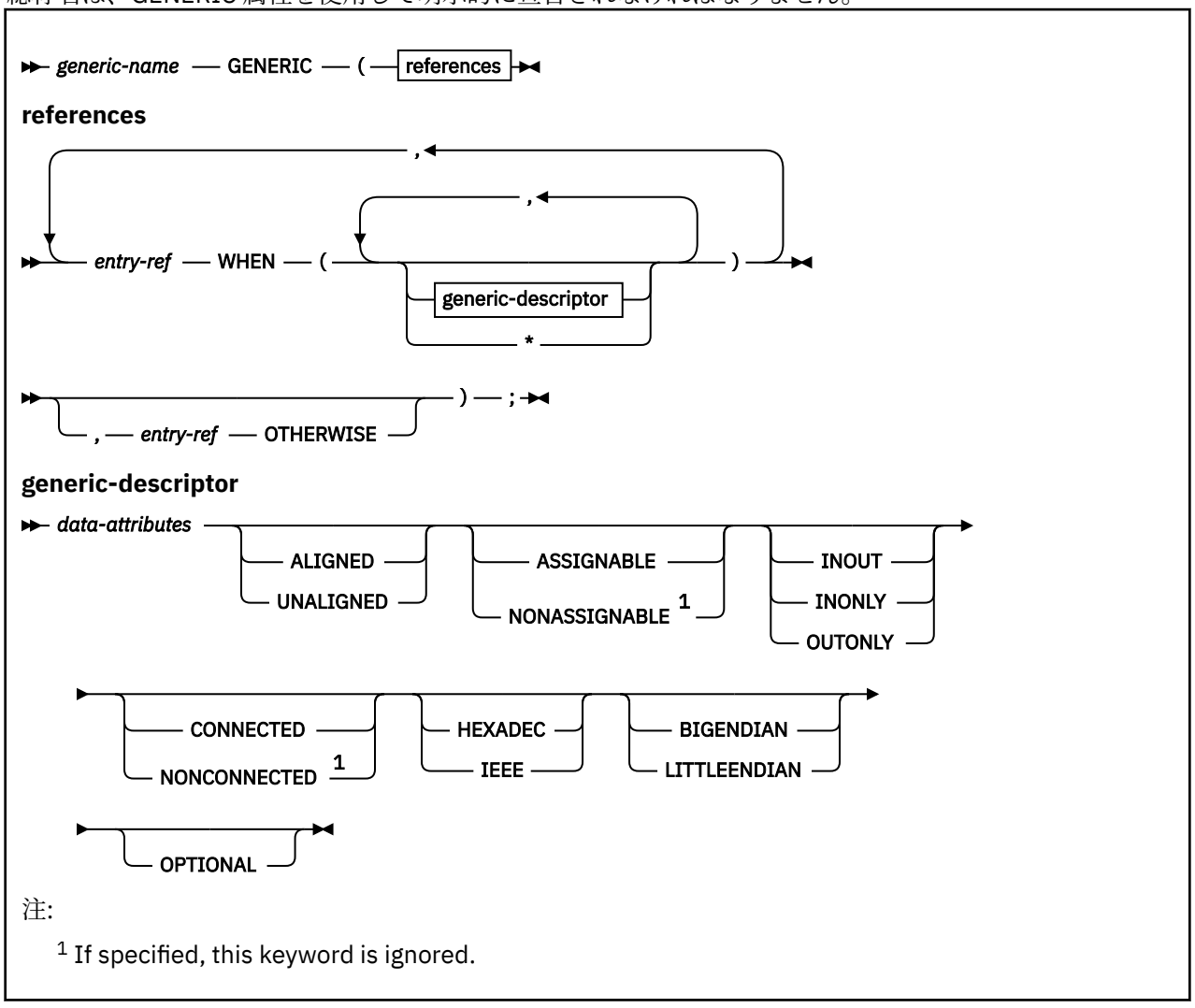
## 総称入り口

総称入り口宣言で、入り口参照とそれらの記述子のセットに対して総称名を指定します。

コンパイル時に、総称名の呼び出しがセット内の 1 つの入り口に置き換えられます。

## GENERIC 属性

総称名は、GENERIC 属性を使用して明示的に宣言されなければなりません。




注:

[1] If specified, this keyword is ignored.

**省略形:** OTHERWISE の場合には OTHER

総称宣言の構文については、148 ページの『DECLARE ステートメント』を参照してください。

**entry-ref**
添え字を付けたり、定義したりする必要はありません。 別の記述子のリストを利用して、同じ入り口参照を単一の GENERIC 宣言内で 複数回使用することができます。

**generic-descriptor**
単一引数と対応します。 対応する引数を持たなければならないので、 関連する入り口参照を置換用に選択できる属性を指定します。

構造体または共用体を指定することはできません。

記述子が不要な場所では、不在の場所をアスタリスクで示さなければなりません。

呼び出されるステートメント内のすべての引数が不在であることを表す 記述子は、入り口の WHEN 文節内で総称記述子を省略することによって表されます。 フォーマットは次のとおりです。

```
generic (... entry1 when( ) ...)
```

**data-attributes**
16 ページの『データ・タイプとその属性』にリストされています。

**ALIGNED および UNALIGNED**
158 ページの『ALIGNED 属性と UNALIGNED 属性』を参照してください。

**ASSIGNABLE および NONASSIGNABLE**
257 ページの『ASSIGNABLE 属性および NONASSIGNABLE 属性』を参照してください。

**CONNECTED および NONCONNECTED**
260 ページの『CONNECTED 属性と NONCONNECTED 属性』を参照してください。

**HEXADEC および IEEE**
259 ページの『HEXADEC 属性と IEEE 属性』を参照してください。

**BIGENDIAN および LITTLEENDIAN**
258 ページの『BIGENDIAN 属性および LITTLEENDIAN 属性』を参照してください。

**OPTIONAL**
114 ページの『OPTIONAL 属性』を参照してください。

総称名の呼び出しに達すると、呼び出しに指定されている引数の数 およびその属性は、セット内の各入り口の記述子リストと比較されます。 数および属性の両方で引数と記述子リストが一致する最初の入り口参照が、 総称名と置き換わります。

以下の例では、DECIMAL または FLOAT、および BINARY または FIXED という属性の 付いた記述子をちょうど 2 つだけ持つ入り口参照が、探索されます。

```
declare Calc generic (
             Fxdcal when (fixed,fixed),
             Flocal when (float,float),
             Mixed  when (float,fixed),
             Error otherwise);
Dcl     X decimal float (6),
        Y binary fixed (15,0);


        Z = X+Calc(X,Y);
```

正確な属性のついた記述子の正確な数のついた入り口が検出されない場合には、 (ある場合) OTHERWISE 文節のついた入り口が選択されます。 前の例では、Mixed が選択されて置き換えられています。

同じような方法で、引数の次元数に基づいて、入り口を選択することができます。

```
dcl  D generic (D1 when ((*))),
                D2 when((*,*))),
     A(2),
     B(3,5);
call D(A);      /* D1 selected because A has one dimension  */
call D(B);      /* D2 selected because B has two dimensions */
```

すべての記述子が省略されるか、またはアスタリスクで構成される場合には、 正しい数の記述子を持つ最初の入り口参照が選択されます。

総称値への参照で引数として使う入り口式は、ENTRY タイプの記述子とだけ一致します。 そのような記述子がないときには、プログラムはエラー状態になります。

# 入り口呼び出しまたは入り口値

入り口値自体が使用されるか、または入り口呼び出しによって戻される値が 使用されるか判然としない場合があります。 このトピックは、以下の場合にどちらが行われるかを理解するために役立ちます。

下表は、入り口参照が呼び出される (または呼び出されない) 条件についての説明です。

| 入口参照が . . . | それは . . . |
|---|---|
| 組み込み関数の場合 | 呼び出されます |
| ヌルであっても、引数リストを持っている場合 | 呼び出されます |
| CALL ステートメントで参照される場合 | 呼び出されます |
| 引数リストがなく、CALL ステートメントで参照されない場合 | 呼び出されません |

**例 1**

この例では、A が呼び出され、B(C) は入り口値として C を渡し、D( C() ) は C を呼び出します。

```
dcl ( A, B, C returns (fixed bin), D) entry;

call A;                        /* A is invoked                        */
call B(C);                     /* C is passed as an entry value       */
call D( C() );                 /* C is invoked                        */
```

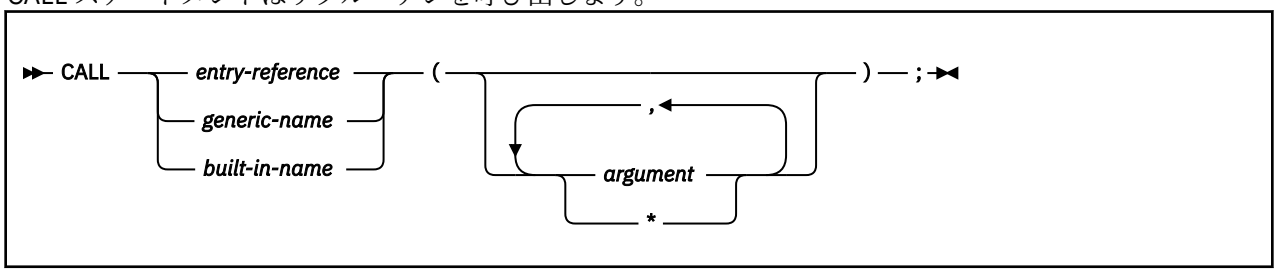
**例 2**

この例では、最初の割り当ては、入り口定数の整数への割り当て試行を表しているため、無効です。 2 番目の割り当ては有効です。

```
dcl A fixed bin,
    B entry returns ( fixed bin );

A = B;
A = B();
```

# CALL ステートメント

CALL ステートメントはサブルーチンを呼び出します。




**entry-reference**
呼び出されるサブルーチンの名前が ENTRY 属性で 宣言されることを指定します。 110 ページの『入り口データ』を参照してください。

**generic-name**
呼び出されるサブルーチンの名前が GENERIC 属性で 宣言されることを指定します。 119 ページの『総称入り口』を参照してください。

**組み込み名 (built-in name)**
呼び出されるサブルーチンの名前が BUILTIN 属性で 宣言されることを指定します。 373 ページの『BUILTIN 属性』を参照してください。

**argument**
呼び出されたサブルーチンに渡されるエレメント、要素式、または集合です。 106 ページの『プロシージャーへの引数の引き渡し』を参照してください。

CALL ステートメントの参照または式は、 呼び出しが実行されるブロック内で計算されます。 この中には、計算結果として入力される ON ユニットの実行も含まれます。

# RETURN ステートメント

RETURN ステートメントは、RETURN ステートメントが入っている サブルーチンまたは関数プロシージャーの実行を終了し、 呼び出しプロシージャーに制御を戻します。 制御は、呼び出し参照の直後の場所に戻されます。

式を伴う RETURN ステートメントを OPTIONS(MAIN) によるプロシージャー内で 使用することはできません。

式を伴わない RETURN ステートメントは、RETURNS オプションを指定したプロシージャー内では無効です。 反対に、式を伴う RETURN ステートメントは、RETURNS オプションを指定していない プロシージャー内では無効です。

RETURNS オプションを指定したプロシージャーでは、少なくとも 1 つの RETURN ステートメント (もちろん式を伴うもの) が含まれて いなければなりません。

## サブルーチンからの戻り

サブルーチンから戻るための RETURN ステートメント構文は以下のとおりです。

RETURN ステートメントが主プロシージャーを終了すると、プログラムが終了する前に、FINISH 条件が起こります。

## 関数からの戻り

関数から戻るための RETURN ステートメント構文は以下のとおりです。

関数参照に戻される値は、指定された式の値であり、 関数が入力された ENTRY ステートメントまたは PROCEDURE ステートメントの RETURNS オプションで指定されている属性と一致させるために変換されます。 次の例を考えてください。

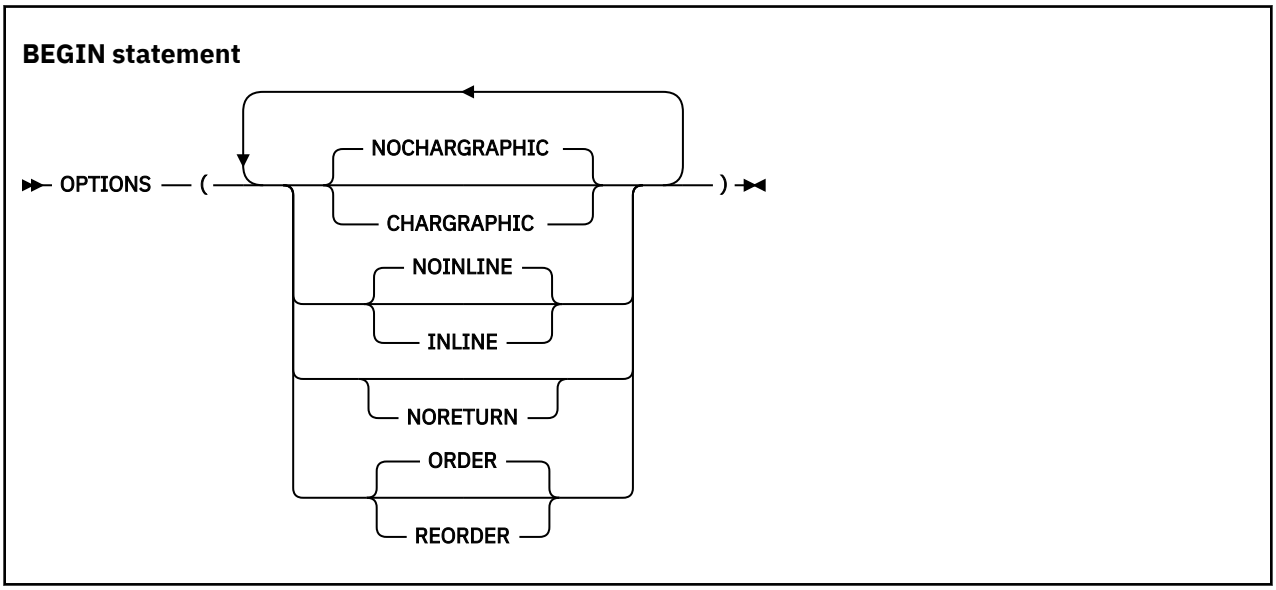
```
F: procedure returns(fixed bin(15));
   .
   .
   .
G: entry returns(fixed dec(7,2));
   .
   .
   .
   dcl D fixed bin(31);
   .
   .
   .
   return (D);
```

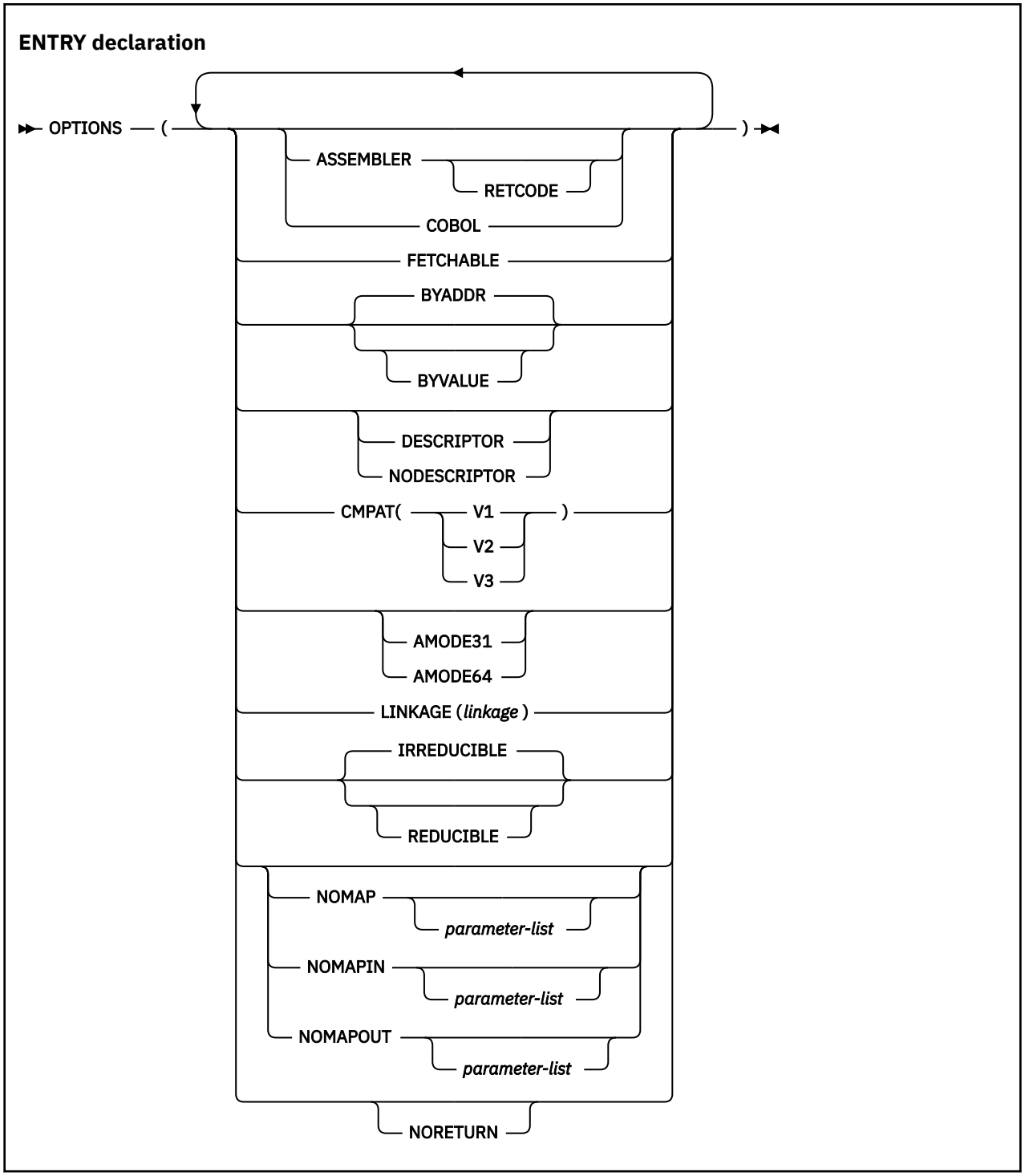
この関数が F で入力されると、D は、プロシージャー F (FIXED BIN(15)) の RETURNS オプションで指定された属性に変換されます。 しかし、この関数が G で入力されると、D は、エントリー G(FIXED DEC(7,2)) の RETURNS オプションで指定された属性に変換されます。

開始ブロック内の RETURN ステートメントに式を指定することはできません。

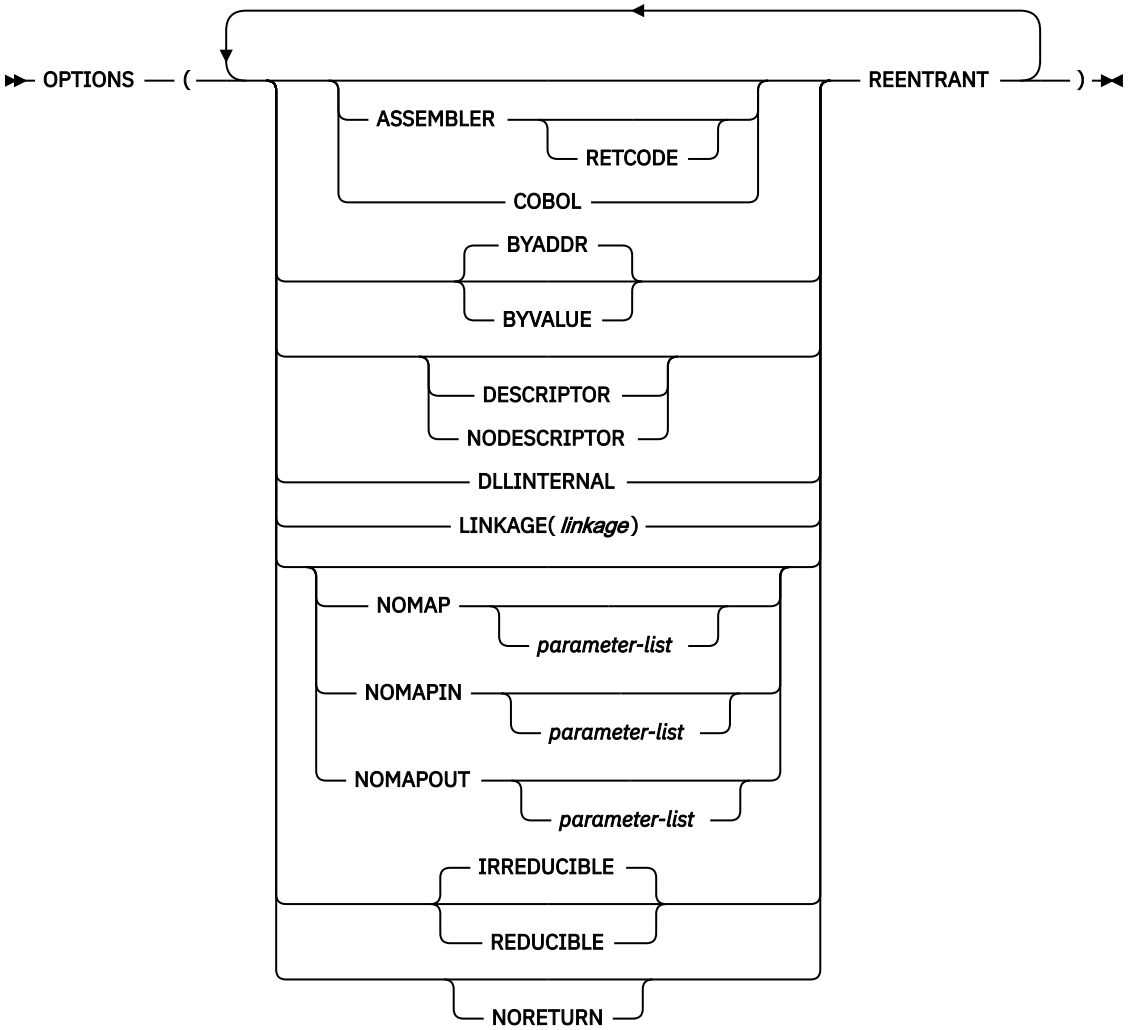
# OPTIONS オプションとその属性

OPTIONS オプションは、PACKAGE、PROCEDURE、ENTRY、および BEGIN の各ステートメントで指定できます。 OPTIONS 属性は ENTRY 宣言で指定できます。 OPTIONS は、ブロックおよびプロシージャー呼び出しに適用する処理特性を指定します。

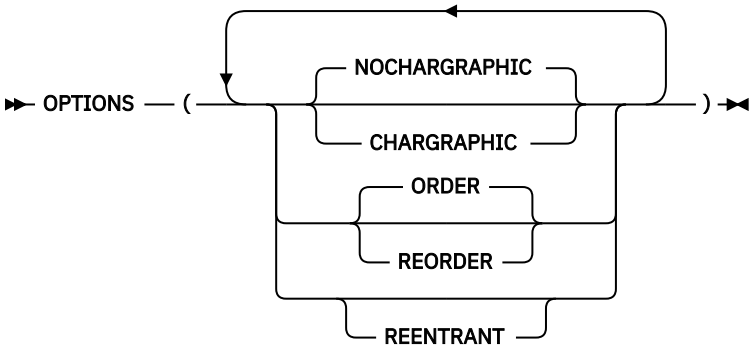



ENTRY declaration
OPTIONS
(
)
ASSEMBLER
RETCODE
COBOL
FETCHABLE
BYADDR
BYVALUE
DESCRIPTOR
NODESCRIPTOR
CMPAT(
V1
V2
V3
)
AMODE31
AMODE64
LINKAGE (linkage)
IRREDUCIBLE
REDUCIBLE
NOMAP
parameter-list
NOMAPIN
parameter-list
NOMAPOUT
parameter-list
NORETURN

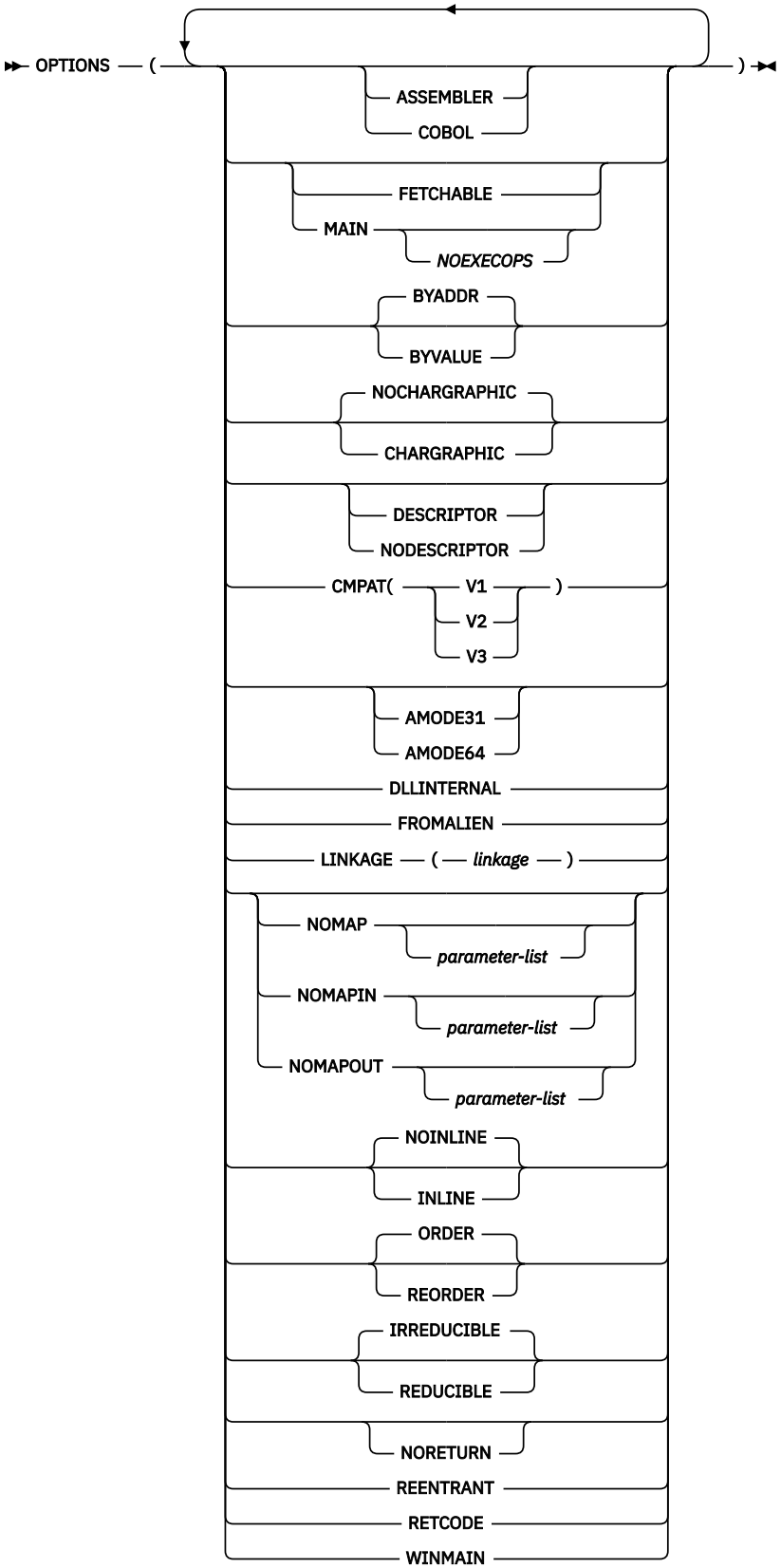
**ENTRY statement**




**PACKAGE statement**



**PROCEDURE ステートメント**




オプションは、ブランクまたはコンマで区切られます。 これらは、任意の順番に示すことができます。

**AMODE31 または AMODE64**

PROCEDURE ステートメントでは、AMODE31 は、コンパイラー・オプション LP (32) が有効である場合にのみ有効で、AMODE64 は、コンパイラー・オプション LP (64) が有効である場合にのみ有効です。

ENTRY 宣言では、AMODE31 または AMODE64 はルーチンのアドレッシング・モード (「amode」) を指定します。

AMODE31 または AMODE64 を指定するには、FETCHABLE、NODESCRIPTOR、および LINKAGE オプションを指定する必要があります。

OPTIONS (AMODE31)または OPTIONS (AMODE64) を指定した ENTRY または PROCEDURE に BYVALUE パラメーターがある場合、そのパラメーターのタイプは、POINTER、HANDLE、ORDINAL、または REAL FIXED BIN のいずれかでなければなりません。

OPTIONS (AMODE31) または OPTIONS (AMODE64)を指定した ENTRY または PROCEDURE に RETURNS (BYVALUE ...) が RETURNS 属性は、ORDINAL タイプか、4 バイト以下のスケールなし REAL FIXED BIN のいずれかを指定しなければなりません。

OPTIONS（AMODE31）または OPTIONS（AMODE64）のエントリ定数は、64 個以下のパラメータの（空の場合もある）括弧に囲まれたリストを指定しなければなりません。 引数がない場合は、ENTRY () として宣言する必要がある。

OPTIONS（AMODE31）または OPTIONS（AMODE64）のエントリ定数は、暗黙的または明示的に別のエントリに割り当てることはできません。

OPTIONS(AMODE31) を指定して ENTRY に渡される引数は、構造式または配列式であってはなりません。

**アセンブラー**
**省略形:** ASM

このオプションは NODESCRIPTOR と同じ効果があります。

プロシージャーに ASSEMBLER オプションがある場合、そのプロシージャーの終了によって、PLIRETV() 値が、そのプロシージャーの戻り値として使用されます。

OPTIONS(ASSEMBLER) を指定する PROCEDURE または ENTRY ステートメントは、別のリンケージが明示的に指定されない限り、LINKAGE(SYSTEM) を持ちます。

詳細については、プログラミング・ガイド を参照してください。

**BYADDR または BYVALUE**
引数とパラメーターがどのように受け渡しされるのかを指定します。 BYADDR はデフォルトです。

BYVALUE は、AREA、PICTURE、WIDEPIC、VARYING、および VARYINGZ では無効です。

BYVALUE および BYADDR 属性は、入り口宣言の記述子リスト、およびパラメーター宣言の属性リスト内にも指定することができます。 入り口またはパラメーター宣言に BYVALUE または BYADDR を指定すると、OPTIONS ステートメントに指定されたオプションが無効になります。

以下の例では、入り口宣言と OPTIONS オプションの 両方に BYVALUE および BYADDR を表しています。 例では、コンパイラー・オプション DEFAULT (BYADDR) が有効であると想定しています。

**例 1 :**

```
MAINPR: proc options(main);


  dcl D entry (fixed bin byaddr,
               ptr,
               char(4) byvalue)         /*  byvalue not needed      */
        options(byvalue);
  dcl E2 entry;                         /* default(byaddr) in effect */
  dcl Length fixed bin,
      P      pointer,
      Name   char(4);


  call D(Length, P, Name);             /* Length is passed byaddr */
```

```
                                        /* P is passed by value */
                                        /* Name is passed by value */


  call E2(P);                           /* P is passed by address */


D: proc(I, Q, C)
   options(byvalue);
  dcl I fixed bin byaddr,
      Q ptr,
      C char(4) byvalue;


E2: proc(Q);
   dcl Q ptr;
```

**例 2 :**

```
dcl F entry (fixed bin byaddr,        /*  byaddr not needed       */
             ptr,
             char(4) byvalue)
     options(byaddr);
dcl E3 entry;
dcl E4 entry (fixed bin byvalue);


call F(Length, P, Name);             /* Length is passed byaddr */
                                     /* P is passed byaddr      */
                                     /* Name is passed by value */


call E3(Name);                       /* Name is passed byaddr   */
call E4(Length);                     /* Length is passed by value */


F: proc(I,P,C) options(byaddr);
  dcl I fixed bin byaddr;             /*  byaddr not needed       */
  dcl P ptr byaddr;                   /*  byaddr not needed       */
  dcl C char(4) byvalue;              /*  byvalue needed          */


E3: proc(L);
   dcl L char(4);


E4: proc(N);
   dcl N fixed bin byvalue;
```

### CHARGRAPHIC または NOCHARGRAPHIC

**省略形:** CHARG、NOCHARG

外部プロシージャーの省略形は、NOCHARG です。 内部プロシージャーと開始ブロックは、それらが含まれているプロシージャーからデフォルトを継承します。

CHARG が有効な場合には、以下のようなセマンティックの変更が起こります。

- すべての文字ストリング割り当てでは、文字の割り当てを混合すると考えられます。
- STRINGSIZE 条件によって、MPSTR 組み込み関数が使用されます。 STRINGSIZE は、切り捨てを起こすことができ、かつインテリジェントな DBCS の切り捨てが必要とされる文字の割り当てが生じた場合に 使用可能にならなければなりません。 次の例を参照してください。

```
 Name: procedure options(chargraphic);
       dcl A char(5);
       dcl B char(8);

/* the following statement...                          */

       (stringsize): A=B;

/*...is logically transformed into...                  */

       A=mpstr(B,'vs',length(A));
```

CHARGRAPHIC オプションと NOCHARGRAPHIC オプションも OPTIONS オプションの外部で指定できますが、この使用法は推奨されておらず、どの言語標準にも準拠していません。

NOCHARG が有効な場合は、セマンティックの変更は起こりません。

**CMPAT**

CMPAT オプションは、ルーチンのコンパイルに使用する CMPAT コンパイラー・オプション設定を指定します。

PROCEDURE ステートメントでは、OPTIONS 属性の CMPAT 値は、コンパイラー CMPAT オプションの CMPAT 値と一致していなければなりません。

ENTRY 宣言では、AMODE31 も指定されていないと、CMPAT オプションは無視されます。 CMPAT と AMODE31 の両方を指定する場合、CMPAT 値は、ルーチンのコンパイルに使用する CMPAT コンパイラー・オプション設定と一致する必要があります。

**COBOL**

このオプションは NODESCRIPTOR と同じ効果を持ちますが、さらに OPTIONS(COBOL) には以下の効果があります。

- 入力宣言またはプロシージャー・ステートメントで他のリンケージが指定されていない限り、LINKAGE(SYSTEM) を暗黙指定します。
- NOMAP、NOMAPIN、または NOMAPOUT オプションの使用を許可します。
- プロシージャー・ステートメントで指定されている場合、 そのプロシージャーの終了によって、PLIRETV() 値が、そのプロシージャーの戻り値として 使用されることを暗黙指定します。

COBOL および MAIN を一緒に指定してはなりません。

**DESCRIPTOR または NODESCRIPTOR**

入り口宣言またはプロシージャー・ステートメントで指定したプロシージャーが呼び出されたときに、記述子リストを渡すかどうかを示します。

DESCRIPTOR が示される場合、必要に応じてコンパイラーは記述子を渡します。

NODESCRIPTOR が示される場合には、コンパイラーは記述子を渡しません。

上記のどちらも表示されない場合には、呼び出されたプロシージャーの いずれかのパラメーターがストリング、配列、区域、構造体、または共用体であるときだけ、DESCRIPTOR が想定されます。

いずれかのパラメーターまたはプロシージャーでエクステント、長さ、またはサイズを示すためにアスタリスク (*) を使用している場合、あるいはいずれかのパラメーターが NONCONNECTED である場合、それらを含むプロシージャー・ステートメントまたは入り口宣言に NODESCRIPTOR があると、エラーになります。

ただし、範囲が指定されていないパラメーターが INONLY VARYING、VARYING4、VARYINGZ のいずれかのストリングの場合は、NODESCRIPTOR を利用できます。

**DLLEXTERNAL または DLLINTERNAL**

このオプションは、プロシージャーまたは入り口が DLL の外部または内部にあることが意図されており、その結果、その名前はコンパイラーによって生成された定義サイド・ファイルに含まれる必要がある、または含まれていてはならないことを示します。

DLLEXTERNAL 属性および DLLINTERNAL 属性は、EXTERNAL プロシージャーまたは ENTRY でのみ有効です。 DLLINTERNAL 属性は FETCHABLE 属性と対立します。

**FETCHABLE**

このオプションは、必要に応じて、プロシージャーを呼び出す前に動的に取り出すかどうかを示します。

FETCHABLE 属性は、INTERNAL プロシージャーでは無効です。

FETCHABLE プロシージャーは、MAIN プロシージャーが含まれるロード・モジュール にはリンクしないでください。

**FROMALIEN**
このオプションは、プロシージャーを非 PL/I ルーチンから呼び出すことができるかどうかを示します。FROMALIEN は任意のプロシージャーで指定できます。ただし、その場合は不必要なオーバーヘッドが生じることになります。

**INLINE または NOINLINE**
INLINE と NOINLINE のオプションは最適化オプションであり、パッケージ内の開始ブロックおよびネストされないレベル 1 のプロシージャー用に指定できます。

INLINE は、開始ブロックまたはプロシージャーがパッケージ内で 呼び出されたとき、その開始ブロックまたはプロシージャーのコードが 呼び出し点でインライン で実行されることを示します。 INLINE が指定されていても、コンパイラーは、開始ブロックまたは プロシージャーをインラインとしないように選択することができます。

NOINLINE は、開始ブロックまたはプロシージャーがいかなる場合にも インラインで実行されないように指示します。

OPTIONS(INLINE) を使用すれば、 適切に構造化された読みやすいコードを書くのが容易になります。例えば、プログラムをプロシージャーのセットへの一連の呼び出しとして 作成した場合、OPTIONS(INLINE) を使えば、これらのプロシージャーを 1 つ 1 つ呼び出す必要がなくなり、オーバーヘッドが少なくなります。

プロシージャーまたは開始ブロックをインラインで実行すると、ONLOC などの組み込み関数によって戻される値は、 その関数がインライン化されているプロシージャーの名前を戻します。 同様に、トレースバックされた情報には、 呼び出されたプロシージャーは含まれません。

インライン化できないプロシージャーと開始ブロックがあります。 これらには下のリストに示すものがありますが、それら以外のものもあります。

- 条件の割り込み可能化が可変であるパッケージ内のプロシージャーと開始ブロック
- ON ステートメントまたは REVERT ステートメントを 含むプロシージャーと開始ブロック
- データ・ディレクティブの入出力ステートメントを含むプロシージャーと開始ブロック
- ENTRY、FORMAT、または LABEL 定数の割り当て または比較を含むプロシージャーと開始ブロック

INLINE オプションを指定したネストされないプロシージャーが 外部ではなく参照もされない場合、このプロシージャーのためのコードは作成されません。 INLINE も NOINLINE も指定されていないプロシージャーについては、 DEFAULT コンパイラー・オプションがオプションをセットします。

INLINE と NOINLINE の詳しい使用法については、「プログラミング・ガイド」を参照してください。

**LINKAGE**
このオプションは、使用する呼び出し規則を指定します。 このオプションは PROCEDURE ステートメントおよび ENTRY 宣言で指定できます。

**CDECL (INTEL のみ)**
このオプションは、32 ビット C コンパイラーが 使用する CDECL リンケージ規約を指定します。

**OPTLINK**
このオプションは最高速リンケージ規約であり、デフォルト設定されています。 これは、ほとんどのコンパイラーの場合は標準のリンケージではありません。

**STDCALL (Windows のみ)**
このオプションは、STDCALL (すべての Windows API が 使用する標準のリンケージ規約) を指定します。

**SYSTEM**
このオプションは、オペレーティング・システムを呼び出すときに 使用する呼び出し規則を指定します。 このオプションは OPTLINK よりも低速ですが、すべての MVS および AIX アプリケーションの標準オプションです。

呼び出し規則の詳細については、「プログラミング・ガイド」を参照してください。

**MAIN**
このオプションは、この外部プロシージャーが PL/I プログラムの 初期プロシージャーであることを示します。 1 プログラムにつき 1 つの外部プロシージャーにだけ、 MAIN が有効であり、必要とされま

す。 オペレーティング・システムの制御プログラムは、そのプログラムの実行の最初のステップとしてこのプロシージャーを呼び出します。

OPTIONS(MAIN) を指定した、 複数のプロシージャーを含んでいる PL/I プログラムは、 予測できない結果を生じることがあります。

MAIN が指定されている場合、COBOL は無効です。

**NOEXECOPS**
NOEXECOPS オプションは、MAIN オプションを使用する場合だけ有効です。このオプションは、ランタイム・オプションが、プログラムを呼び出すコマンドや ステートメントでは指定できないこと指定します。 主プロシージャーのパラメーターだけが指定されます。

**NOMAP、NOMAPIN、NOMAPOUT**
これらのオプションは、COBOL と PL/I の間のインターフェースにおけるデータ集合の自動操作を防止します。

各オプション引数リストでは、オプションが適用されるパラメーターを指定できます。 パラメーターは、どのような順番でも表示でき、コンマまたはブランクによって分離することができます。 オプション用の引数リストがない場合、項目名のすべてのパラメーターはデフォルト・リストになります。

NOMAP、NOMAPIN、または NOMAPOUT は、同一の OPTION 指定ですべてを使用することができます。 この指定は、複数の指定またはデフォルト引数リストの同一パ ラメーターに含まれてはなりません。

これらのオプションは受け入れられますが、COBOL オプションが適用されない限りは無視されます。

**NORETURN**

この属性は、この属性が指定されているルーチンが呼び出された場合、ルーチンは戻らず、その呼び出し後のコードは実行されないことを示しています (これは、例えば、ルーチンが STOP ステートメントまたは SIGNAL ERROR ステートメントで終了している場合に真となります)。

NORETURN の指定は、不要コード、および RETURN ステートメントがない可能性のあるコードを検出するために、コンパイラーが、最適化されたコードを生成し、より的確に動作するうえで役立ちます。

**ORDER または REORDER**
ORDER と REORDER は、プロシージャーまたは開始ブロックに指定される最適化オプションです。

ORDER は、ブロック内で修正された変数の最も新しく 割り当てられた値だけが入力される ON ユニットで 利用できることを示します。これは、ステートメントの 実行時およびブロック内の式で計算条件が生じたためです。

REORDER オプションを指定すると、コンパイラーが最適化コードを生成し、エラー・フリーの実行があった場合に、ソース・プログラムに指定された結果を生成することができます。

ORDER オプションと REORDER オプションも OPTIONS オプションの外部で指定できますが、この使用法は推奨されておらず、どの言語標準にも準拠していません。

ORDER と REORDER オプションの詳細については、「プログラミング・ガイド」を参照してください。

上記のオプションが両方とも外部プロシージャーに指定されない場合には、 デフォルトとして DEFAULT コンパイラー・オプションがセットされます。 内部ブロックは、含まれているブロックから ORDER または REORDER を継承します。

**REDUCIBLE または IRREDUCIBLE**
**省略形:** RED、IRRED

REDUCIBLE は、引数 (1 つまたは複数) が変更されない限り、 プロシージャーまたは入り口を複数回呼び出す必要がないこと、 およびプロシージャーの呼び出しに副次作用がないことを示します。

例えば、変更されないデータに基づいて結果を計算するユーザー作成の 関数には、REDUCIBLE が宣言されなければなりません。 乱数や時刻などの、変更されるデータに基づいて結果を計算する 関数は、IRREDUCIBLE として宣言する必要があります。

REDUCIBLE オプションと IRREDUCIBLE オプションも OPTIONS オプションの外部で指定できますが、この使用法は推奨されておらず、どの言語標準にも準拠していません。

**REENTRANT**
このオプションは無視されます。 Intel および AIX プラットフォームでは すべての PL/I プログラムが再入可能です。 z/OS プラットフォームでは、RENT コンパイラー・オプションによって、コンパイルされたすべてのプログラムが 再入可能であり、静的変数 (NOWRITABLE コンパイラー・オプションの利用が必要になる場合もあります。) を 変更しないものであれば、その他のプログラムも再入可能です。

**RETCODE**
このオプションを指定すると、ENTRY 点に ASM または COBOL オプション も指定されている場合、ENTRY は呼び出された後、PL/I 戻りコードとして保管さ れる値を戻します。 基本的に、このような ENTRY が呼び出された場合、その戻り値は PLIRETC サブルーチンに渡されます。

**WINMAIN (Windows のみ)**

WIN

このオプションは、自動的に LINKAGE(STDCALL) および EXT('WinMain') を暗黙指定します。 関連ルーチンには、以下の 4 つのパラメーターを含める必要があります。

1. インスタンス操作点
2. 直前の操作点
3. コマンド行へのポインター
4. ShowWindow に渡される整数

これらは、C WinMain が予期する 4 つのパラメーターと同じであり、 このルーチンからの呼び出しは、C ルーチンが予期するものと同じです。

## RETURNS オプションとその属性

プロシージャーが関数プロシージャーである場合には、 プロシージャー・ステートメントに RETURNS オプションを 指定しなければなりません。 さらに、呼び出しプロシージャーまたはパッケージで、RETURNS 属性のついた入り口としてプロシージャーを宣言しなければなりません。

RETURNS オプションおよび RETURNS 属性は、 戻される値の属性を指定するために使用されます。 RETURNS オプションの属性は、RETURNS 属性の属性と 一致しなければなりません。

サブルーチンである (そのため、CALL ステートメントによって呼び出される) プロシージャーでは、プロシージャー・ステートメントに RETURNS オプションがあってはなりません。 また、このようなプロシージャーの入り口宣言に RETURNS 属性があってはなりません。

複数の属性を指定する場合は、属性をブランクで区切る 必要があります (括弧で囲まれる精度などの属性以外の場合)。

属性は、それが宣言ステートメント内にあるのと同じ方法で指定されます。 デフォルトは通常の方法で適用されます。

指定できる属性は、次のとおりです。

- すべてのデータ属性
- 位置合わせ属性 ALIGNED および UNALIGNED
- データ以外の属性 BYVALUE/BYADDR、DATE、VALUELIST、および VALUERANGE

ENTRY 変数は、LIMITED 属性を持たなければなりません。 また、TYPE 属性を指定して、ユーザー定義タイプ、序数、 およびタイプ付き構造体および共用体を指定することができます。

ストリングの長さおよび区域サイズは、定数で指定されなければなりません。 戻り値は、指定される長さまたはサイズを持ちます。

RETURNS(BYVALUE) は、AREA、PICTURE、VARYING、VARYINGZ 属性では無効です。

プロシージャーに ENTRY ステートメントが含まれ、そのプロシージャーまたは 2 次エントリー・ポイントのいずれかが値も集合値も返さない場合、BYADDR 属性が有効でなければなりません。

z/OS では、BYADDR が RETURNS のデフォルトです。 C 関数を呼び出す場合、RETURNS の属性リスト内に BYVALUE が指定されていなければなりません。

**関連情報**

106 ページの『BYVALUE および BYADDR の使用』
引数が値 (BYVALUE) で渡されない限り、引数への参照 (値ではない) は、一般的にサブルーチンまたは関数に渡されます。 これは、参照または BYADDR によって引数を渡すことによって認識されます。

# 第 6 章 タイプ定義

プログラミング言語では、タイプ は値のセットの記述とこれらの値に対して実行できる操作のセットです。 PL/I には多くの組み込みデータ・タイプがあります。 それぞれのタイプは、多くのエレメントの属性を指定できます。 PL/I では、組み込みデータ・タイプを使用して、 ユーザー独自のタイプを定義することができます。 この章では、ユーザー定義のタイプ (別名、序数、 構造体、および共用体)、これらのタイプ、ハンドル、 およびタイプ付き関数をとる変数の宣言について説明します。

これらの組み込みデータ・タイプについては、15 ページの『第 2 章 データ・エレメント』を参照してください。

DEFINE ステートメントによって作成されるすべてのタイプは、DECLARE ステートメントで名前に適用されるものと同じスコーピング規則に従います。 例えば、あるプロシージャー内で定義された ORDINAL は、そのプロシージャーのすべての子プロシージャーで認識されますが、その姉妹または親プロシージャーでは認識されません。 このため、プロシージャーがタイプを返す場合、このタイプを親プロシージャーまたはパッケージ・レベルで定義する必要があります。

タイプを使用する前に、まずそのタイプの DEFINE ステートメントを記述する必要があります。

DEFINE ステートメント内のタイプ名は、先頭を __ (2 つの下線) にしてはなりません。

タイプ参照は、`paint.color` など、ドットで区切られた一連の ID で構成されます。

DEFINE ステートメントのバリアントごとに、対応する XDEFINE ステートメントがあり、 その構文は (最初のキーワードを除いて) 同じです。 XDEFINE ステートメントで指定された名前が同じタイプの前の (X)DEFINE ステートメント内に既に出現している場合、 その XDEFINE ステートメントは無視されます。 XDEFINE ステートメントで、 既に別のタイプとして定義されている名前を指定することはできません (例えば、XDEFINE ALIAS ステートメントで、XDEFINE ORDINAL ステートメント内に既に出現している名前を指定することはできません)。

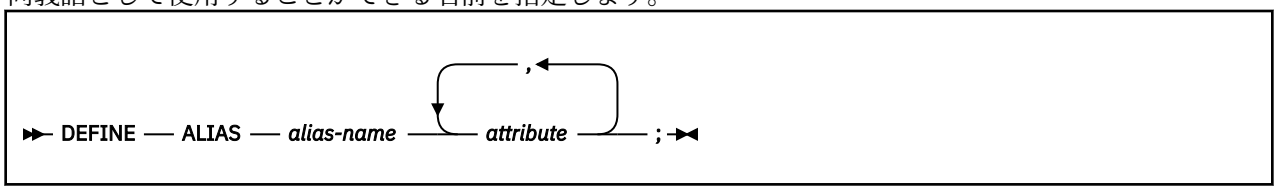
## ユーザー定義のタイプ (別名)

別名 は、明示のデータ・タイプが使用できる所では 必ず使用することができるタイプ名です。 DEFINE ALIAS ステートメントを使用して、 データ属性の収集について別名を定義することができます。

この方法で、データ・タイプに意味のある名前を割り当てて、 プログラムの理解度を高めることができます。 別名を定義することによって、データ属性のセットについての 短い注記を提供し、タイプミスを減らすこともできます。

### DEFINE ALIAS ステートメント

DEFINE ALIAS ステートメントは、 ユーザーが別名に割り当てたデータ・タイプ属性のセットのために、同義語として使用することができる名前を指定します。




**alias-name**
指定された属性によって定義された明示のデータ・タイプが 許可される場合はいつでも使用することができる名前を指定します。

**属性**
指定できる属性は、RETURNS オプションで有効なデータ属性、21 ページの表 8 にリストされている属性と、属性 DATE、VALUELIST、および VALUERANGE のみです。 RETURNS オプションと属性内の場合と同様に、ストリングの長さや区域サイズはいずれも制限付き式でなければなりません。

(UN)ALIGNED、(NON)NATIVE、BYVALUE/BYADDR などの非データ属性は、RETURNS オプションでは有効ですが、DEFINE ALIAS ステートメントでは無効です。

© Copyright IBM Corp. 1999, 2022

配列または構造体をとる属性リストに別名を指定することはできません。 ただし、DEFINE ORDINAL ステートメント、DEFINE STRUCTURE ステートメント、または他の DEFINE ALIAS ステートメントで定義されたタイプには、別名を指定できます。

欠落データ属性は PL/I デフォルトを使用して提供されます。

コンパイラーは、以下の 2 つのステートメントをすべてのコンパイルの PACKAGE レベルに追加します。

```
Define alias __SIGNED_INT signed fixed bin(p1,0);

Define alias __UNSIGNED_INT unsigned fixed bin(p2,0);
```

ここで、LP(32) では p1 = 31 および p2 = 32、LP(64) では p1 = 63 および p2 = 64 です。

これにより、32 ビット (後に 64 ビット) 用の正しいコードを作成することがはるかに容易になります。 配列の索引付けに使用されるすべての変数は、FIXED BIN(31) としてではなく、属性 TYPE __SIGNED_INT を使用して宣言する必要があります。

### 例

この例では、Name が DECLARE ステートメントで使用されるたびに、属性 char(31) varying を持ちます。

```
define alias Name   char(31) varying;
define alias Salary fixed dec(7);     /* real by default       */
define alias Zip    char(5)           /* nonvarying by default */
```

### 関連情報

136 ページの『DEFINE ORDINAL ステートメント』
DEFINE ORDINAL ステートメントは、順序付けられた (名前付きの) 値の セットを表すタイプを指定します。

137 ページの『タイプ付き構造体と共用体の定義』
DEFINE STRUCTURE ステートメントは名前付き構造体または 共用体タイプを指定します。

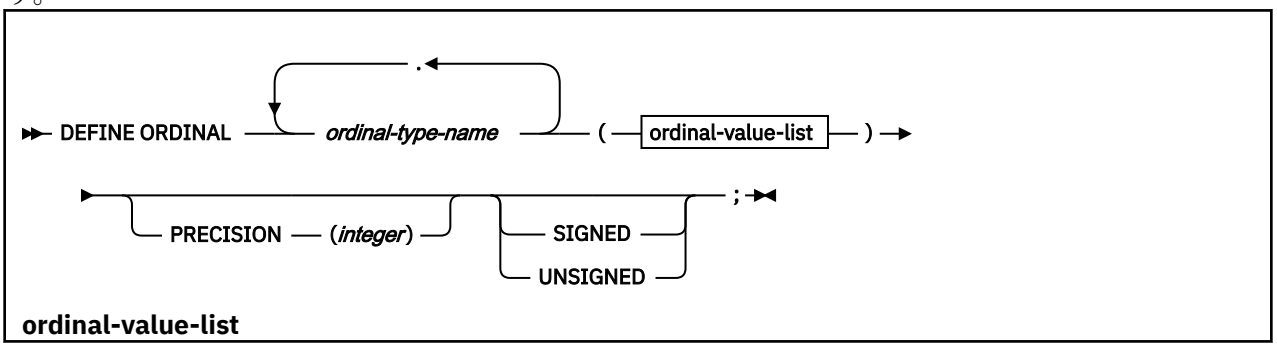
## 序数の定義

序数 は順序付けられた値の (名前付き) セットです。 DEFINE ORDINAL ステートメントを使用すれば、序数を定義して、 そのセットを参照するための有意義な名前を割り当てることができます。

例えば、color という序数を定義できます。 color 序数は、red、yellow、blue などのメンバーを含むことができます。 color セットのメンバーは、関連付けられている固定 2 進値ではなく、名前で参照することができ、コードがより自己説明的なものになります。 また、序数タイプとして宣言された変数は、同じ 序数タイプ (またはそのタイプのメンバー) に対してのみ 割り当てたり比較することができます。 この自動チェック機能により、プログラムの信頼性が向上します。

## DEFINE ORDINAL ステートメント

DEFINE ORDINAL ステートメントは、順序付けられた (名前付きの) 値の セットを表すタイプを指定します。



**ordinal-type-name**
序数値のセット名を指定します。 この名前は、ORDINAL 属性を持つ DECLARE ステートメントにおいてのみ使用できます。 この名前を別の場所で使用すると、ほかの任意の非序数名として処理されます。

**member**
セット内のメンバーの名前を指定します。

**VALUE**
セット内の特定メンバーの値を指定します。 最初のメンバーの場合、VALUE 属性を省略すると、ゼロの値が使用されます。 ほかのメンバーの場合、VALUE 属性を省略すると、 次に大きい整数値が使用されます。

VALUE 属性の値は整数でなければなりません。 この整数値は正負の符号を持つことができますが、 確実に増加していかなければなりません。 指定された (または想定された) VALUE 属性の値を、XN 定数として指定することもできます。

**PRECISION**
**省略形:** PREC

特定の序数値の精度を指定します。 この属性を省略すると、序数値の範囲が精度を決定することになります。

最大精度は、FIXED BINARY として宣言されたデータ項目の精度と同じです。

**SIGNED または UNSIGNED**
序数値が負の値を取ることができることを示しています。 これらを指定しないと、負の値を取るかどうかは 序数値の範囲によって決定されることになります。 例えば、負の値が存在している場合は、SIGNED 属性が適用されます。

### 例

以下の例では、Red の値は 0、Orange の値は 1 であり、以下同様に続きます。 一方、Low の値は 2、Medium の値は 3 です。

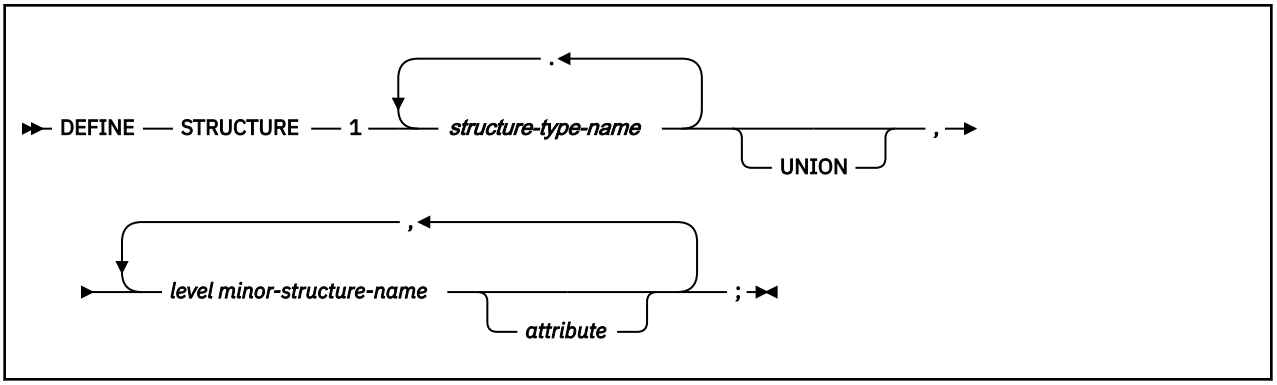
```
define ordinal Color ( Red,          /* is 0, since VALUE is omitted */
                       Orange,
                       Yellow,
                       Green,
                       Blue,
                       Indigo,
                       Violet );

define ordinal Intensity ( Low    value(2),
                           Medium,
                           High   value(5));
```

### 関連情報

24 ページの『SIGNED 属性と UNSIGNED 属性』
SIGNED 属性と UNSIGNED 属性は、FIXED BINARY 変数と ORDINAL 変数にのみ使用できます。

## タイプ付き構造体と共用体の定義

DEFINE STRUCTURE ステートメントは名前付き構造体または 共用体タイプを指定します。




**省略形:** STRUCT

**structure-type-name**
この構造化タイプに与えられる名前を指定します。 この名前は次元を持つことはできませんが、副構造体を持つことはできます。

**UNION**
176 ページの『UNION 属性』を参照してください。

**minor-structure-name**
下のレベルに与えられた名前を指定します。

**属性**
小構造体名についての属性を指定します。 以下の属性のみが許可されています。

- データ属性
- INITIAL 非データ属性

DEFINE STRUCT ステートメントに指定された任意のストリングの長さ、領域サイズ、または配列次元が制限された式でなければなりません。

DEFINE STRUCT ステートメント内の INITIAL 式は、アドレス値に依存しない、制限された式でなければなりません。 このため、これらの INITIAL 式で ENTRY、FILE、および LABEL 定数を使用することはできません。

欠落データ属性は PL/I デフォルトを使用して提供されます。 変数がタイプ付き構造体として宣言された場合は、以下のどの属性も、タイプ付き構造体のメンバーに影響を与えません。 これらの属性が影響するのは、変数が非タイプ付き構造体として宣言されている場合のみです。

- ALIGNED|UNALIGNED
- ASSIGNABLE|NONASSIGNABLE
- DIMACROSS
- NATIVE|NONNATIVE
- NORMAL|ABNORMAL
- SUPPRESS

**制約事項 :**

- 定義済み構造体は、その構造体の位置合わせの倍数となるバイト数でなければなりません。
- 定義済み構造体では、最も厳しい位置合わせを持つエレメントの前にあるバイト数が、 そのエレメントの位置合わせの倍数でなければなりません。

例えば、最も厳しく位置合わせされた項目として位置合わせされた fixed bin(31) フィールドが構造体に含まれている場合は、以下が適用されます。

- 構造体は、4 バイトの倍数でなければならない。
- 最初の位置合わせされた fixed bin(31) フィールドの前には、4 バイトの倍数のバイト数 がなければならない。

DEFINE STRUCTURE ステートメントは「強い」タイプを定義します。 言いかえると、そのタイプで宣言された変数は 同じタイプを持つ変数 (またはパラメーター) にのみ割り当てることができます。 タイプ付き構造体をデータ・ディレクティブ入出力ステートメントに使用することはできません。

定義する構造体名だけを指定し、メンバーを指定しない DEFINE STRUCTURE ステートメントは、「指定されていない構造体」を定義します。

- 指定されていない構造体は参照解除できませんが、HANDLE を宣言するために 使用できます (HANDLE はどちらにしても参照解除できません)。
- 指定されていない構造体は、メンバーを指定する後続の DEFINE STRUCTURE ステートメントの対象にすることもできます。

指定されていない構造体の定義は、構造体の定義に 2 つ目の構造体へのハンドルがあり、2 つ目の構造体にも 1 つ目の構造体へのハンドルがある場合に役立ちます。 例えば次の例では、親構造体に子構造体へのハンドルがあり、 一方で子構造体にも親構造体へのハンドルがあります。

```
define structure 1 child;

define structure
 1 parent,
   2 first_child   handle child,
   2 parent_data   fixed bin(31);

define structure
 1 child,
   2 parent        handle parent,
   2 next_child    handle child,
   2 child_data    fixed bin(31);
```

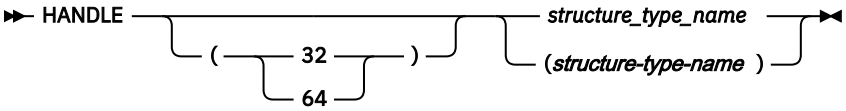
**関連情報**

174 ページの『構造体』
構造体 とは、構造体、共用体、要素変数、 および配列であるメンバー・エレメントの集合です。

265 ページの『INITIAL 属性』
INITIAL 属性は、変数にストレージが割り振られるときにその変数に 割り当てられる初期値を指定するものです。

## HANDLE 属性

HANDLE 属性を使用して、変数を構造体タイプへの ポインターとして宣言することができます。 そのような変数はハンドル と呼ばれます。




**32**
HANDLE(32) はサイズが 4 バイトであり、デフォルトでフルワード位置合わせされます。

**64**
HANDLE(64) はサイズが 8 バイトであり、デフォルトでダブルワード位置合わせされます。

***structure-type-name***
このハンドルがポイントするタイプ付き構造体を指定します。

LP(32) コンパイラー・オプションが有効な場合、デフォルトは HANDLE(32) です。 LP(64) コンパイラー・オプションが有効な場合、デフォルトは HANDLE(64) です。 HANDLE(64) は LP(64) でのみ有効です。

HANDLE(32) の HANDLE(64) への割り当ては常に有効です。 その逆は、HANDLE(64) の先頭 4 バイトがゼロの場合にのみ有効です。

定義された構造体と同様に、ハンドルはタイプが強く付きます。 ハンドルは同じ構造体タイプのハンドルについてのみ、 割り当てられ、比較することができます。 ハンドルについての算術演算はできません。

ADDR 組み込み関数を使用して、タイプ付き構造体のアドレスを ハンドルに割り当てることはできません。 ADDR 組み込み関数はポインターを戻し、そのポインターを ハンドルに割り当てることができないからで

す。 ただし、HANDLE 組み込み関数は、その引数としてタイプ付き 構造体を取り、そのタイプにハンドルを戻します。 次の例を考えてください。

次の例は、141 ページの『例 2』 で定義されている tm 構造体タイプに基づいています。 次のコードでは、tm タイプを見つけるハンドルが宣言され、Daterec のアドレスがそのハンドルに割り当てられます。

```
dcl P_Daterec handle tm;
dcl Daterec type tm;

P_Daterec = handle(Daterec);
```

ハンドルは POINTERVALUE 組み込み関数を使用して、 ポインターに変換することができます。

**関連情報**

POINTERVALUE 組み込み関数
POINTERVALUE は、変換された *x* の値であるポインター値を戻します。

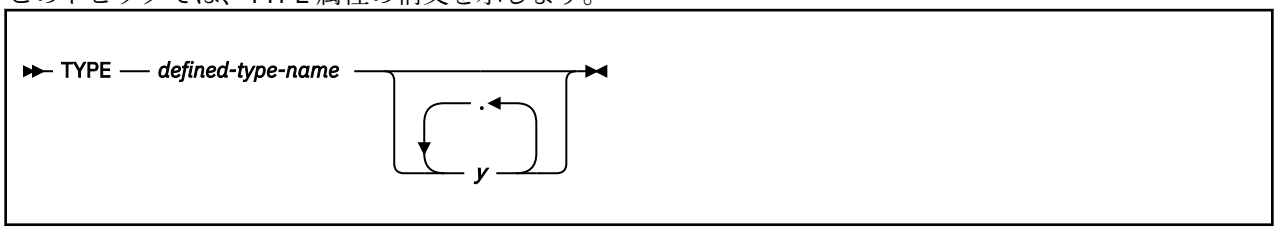
HANDLE 組み込み関数
HANDLE は、タイプ付き構造体 *x* にハンドルを戻します。

# タイプ変数の宣言

TYPE 属性を使用すれば、DEFINE ALIAS、DEFINE STRUCTURE、または DEFINE ORDINAL のいずれかのステートメントに指定されたタイプで変数を宣言できます。

## TYPE 属性

このトピックでは、TYPE 属性の構文を示します。




**defined-type-name**
直前の定義済み別名、 定義済み構造体、または序数タイプの名前を指定します。

***Y***
QUALIFY ブロックの名前を指定します。

### 例 1

以下のコードについて考察します。

```
define alias Name char(31) varying;
               /* Name has attributes char(31) varying */
dcl Employee_Name type Name;
               /* Employee_Name type char(31) varying */
define alias Rate fixed dec(3,2);
               /* Rate has attributes fixed dec real */
```

```
define structure
1 Payroll,
2 Name,
3 Last  type Name,
3 First type Name,
2 Hours,
3 Regular  fixed dec(5,2),
3 Overtime fixed dec(5,2),
2 Rate,
3 Regular  type Rate,
3 Overtime type Rate;

dcl Non_Exempt type Payroll;     /* Has Payroll structure type */
dcl Exempt     type Payroll;     /* Has Payroll structure type */
```

TYPE 属性を DEFINE ALIAS ステートメントで指定して、直前の DEFINE ALIAS ステートメントで定義されたタイプのための 別名を指定することができます。 次の例を参照してください。

```
define alias Word fixed bin(31);
define alias Short type word;
```

**例 2**

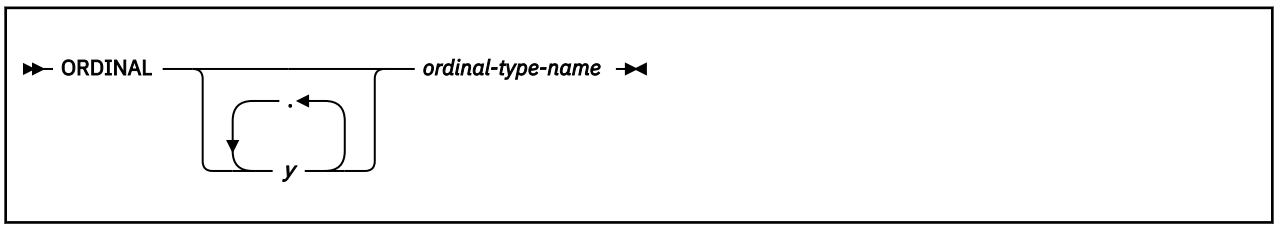
以下の例は、いくつかの名前付きタイプ、構造体タイプ (tm) を定義し、このタイプ付き構造体への ハンドルを取得する C 関数を宣言しています。

```
define alias int     fixed bin(31);
define alias time_t  fixed bin(31);
define structure
  1 tm
   ,2 tm_sec   type int    /* seconds after the minute (0-61)     */
   ,2 tm_min   type int    /* minutes after the hour (0-59)       */
   ,2 tm_hour  type int    /* hours since midnight (0-23)         */
   ,2 tm_mday  type int    /* day of the month (1-31)             */
   ,2 tm_mon   type int    /* months since January (0-11)         */
   ,2 tm_year  type int    /* years since 1900                    */
   ,2 tm_wday  type int    /* days since Sunday (0-6)             */
   ,2 tm_yday  type int    /* days since January 1 (0-365)        */
   ,2 tm_isdst type int    /* Daylight Saving Time flag           */
   ;

dcl localtime    ext('localtime')
            entry( nonasgn byaddr type time_t )
            returns( byvalue handle tm );

dcl time    ext('time')
            entry( byvalue pointer )
            returns( byvalue type time_t );
```

## ORDINAL 属性

TYPE 属性または ORDINAL 属性を使用して、序数タイプを持つ変数を宣言することができます。

TYPE 属性の構文については、140 ページの『TYPE 属性』を参照してください。




**ordinal-type-name**
定義済みの序数タイプ値のセット名を指定します。

***Y***
QUALIFY ブロックの名前を指定します。

ORDINAL 属性は FIXED や SIGNED などのほかの属性とは対立しますが、 BASED や DIMENSION などとは併用できます。

**例**

```
dcl Wall_color ordinal Color;
```

# タイプ付き構造体修飾

タイプ付き構造体のメンバーを参照するには、以下を使用します。 演算子、または => 演算子を持つハンドル。 典型的な非タイプ付き構造体内の名前の場合とは異なり、タイプ付き構造体内の名前は 独自の「ネーム・スペース」を形成し、 それら自体から参照することはできません。

例えば、以下の宣言および定義の場合、B は有効な参照ですが、Y は無効です。

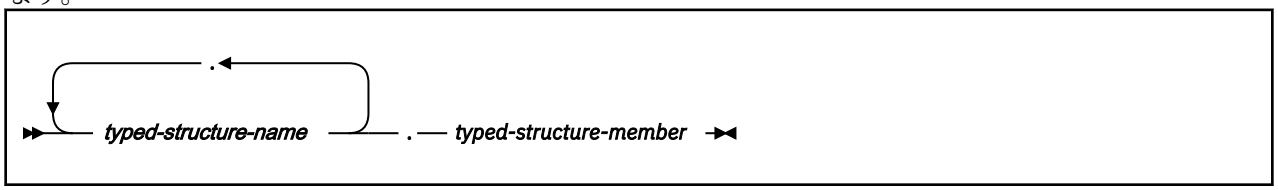
```
dcl 1 A,
2 B fixed bin,
2 C fixed bin;

define structure
1 X,
2 Y fixed bin,
2 Z fixed bin;
dcl S type X;
```

タイプ名も宣言された名前からは、別のネーム・スペース内にあります。 そのため、タイプの名前を変数名としても使用できます。

```
   define alias Hps pointer;
   declare Hps type Hps;
```

## ピリオド演算子 (.) の使用

このトピックでは、ピリオド演算子 (.) を使用してタイプ付き構造体メンバーを参照するための構文を示します。




**typed-structure-reference**
宣言されたタイプ付き構造体の名前です。

**typed-structure-member**
構造体タイプの、参照された大構造体メンバーまたは小構造体メンバーの名前です。

例えば、140 ページの『TYPE 属性』 の例で定義された構造体タイプ tm および関数 localtime の場合、以下のコードはシステム日付を取得し、時刻を表示します。

```
     dcl Daterec type tm;

     dcl ltime   type time_t;
     dcl ptime   handle tm;

     ltime = time( sysnull() );
     ptime = localtime( ltime );

     Daterec = ptime => tm;

display ( edit(Daterec.Hours,'99') || ':' ||
edit(Daterec.Minutes,'99') || ':' ||
edit(Daterec.Seconds,'99'));
```

## 配列およびタイプ付き構造体または共用体の組み合わせ

タイプ付き構造体または共用体では、次元属性を指定できます。 結果としての配列には、同じ名前、レベル、およびメンバーを持つ構造体または共用体が含まれます。

詳しくは、『183 ページの『配列、構造体、および共用体の組み合わせ』』を参照してください。

### 例 1

次の例を考えてください。

```
   dcl 1 a(3),
         2 b(4) fixed bin,
         2 c(5) fixed bin;
```

このタイプなし構造体の場合、a(1).b(2)、a.b(1,2)、および a(1,2).b の持つ意味は同一です。

**例 2**

ただし、以下のタイプ付き構造体の場合は、x(1).b(2) のみが有効です。

```
define structure
  1 t,
    2 b(4) fixed bin,
    2 c(5) fixed bin;

dcl x(3) type t;
```

また、代入ステートメント x.b = 0 は無効ですが、x(1).b = 0; は有効です。

**例 3**

この例は、143 ページの『例 2』 で定義されている構造体タイプ t に基づいています。

関数 f が以下のように宣言されているとします。

```
dcl f entry returns( type t );
```

display( f().b(2) ) は有効です。

## ハンドルの使用

=> 演算子を使用して、型付き構造体のアクセス・メンバーを処理します。

以下の例で、141 ページの『例 2』 で定義された tm タイプを指定すると、そのタイプのハンドルを使用して時刻が表示されます。

```
dcl P_Daterec handle tm;
P_Daterec = handle(Daterec);

display ( edit(P_Daterec=>tm_hours,'99') || ':' ||
edit(P_Daterec=>tm_min,'99') || ':' ||
edit(P_Daterec=>tm_sec,'99') );
```

ハンドルは、レベル 1 名 (タイプ名自体) を含む タイプ付き構造体内のすべてのメンバーを見付けることができます。 そのタイプ名についての、ハンドルによる参照は、 そのハンドルによってポイントされた構造体への参照を構成することかできます。 これで、そのハンドルによってこの集合データについての参照ができるようになります。 例えば、H1 および H2 が 2 つの割り振り済み構造体を指している場合は、以下のように 2 つの構造体を交換できます。

```
define structure 1 T, 2 U, 2 V, 2 W;
dcl (H1, H2) handle T;
dcl Temp     type T;

Temp = H1=>T;
H1=>T = H2=>T;
H2=>T = Temp;
```

# 序数の使用

序数を使用する場合は、以下の使用規則に留意してください。

- 序数は強いタイプ です。つまり、序数は同じタイプの別の序数に対してのみ割り当てたり 比較することができます。 序数は DECLARE ステートメントで宣言されていなければなりません。
- DEFINE ORDINAL ステートメントの ordinal-type-name は、比較にも割り当てにも使用できません。
- 序数はほかのデータ・タイプと同様に引数/パラメーターとして 受け渡しできます。
- 計算タイプの引数を必要とする組み込み関数では、 序数を引数として使用することはできません。 ただし、序数をサポートするための組み込み関数が定義されており、 BINARYVALUE が拡張されています。 これらの組み込み関数については、144 ページの表 36 にまとめてあります。 これらの関数の説明につい

ては、373 ページの『第 18 章 組み込み関数、疑似変数、およびサブルーチン』を参照してください。 これらの組み込み関数は 1 つの引数だけを取ります。 この引数は ORDINAL タイプの参照でなければなりません。

表 *36.* 序数処理組み込み関数

| 機能 | 説明 |
|---|---|
| BINARYVALUE | 序数を 2 進値に変換します。 |
| ORDINALPRED | 序数が次に低い値を戻します。 |
| ORDINALSUCC | 次に高い序数値を戻します。 |
| ORDINALNAME | 序数の名前を指定する文字ストリングを戻します。 |

### 例 1: DO ループで序数定義からの値をリストする

以下のサンプル・コードでは、最初の DO ループが Color セットのメンバー を昇順にリストし、2 番目の DO ループがそれらを降順にリストします。 この例では、137 ページの『例』からの序数定義が使用されています。

```
dcl Next_color ordinal Color;

do Next_color = first (:Color:)
                repeat ordinalsucc( Next_color )
                until (Next_color = last (:Color:));

  display( ordinalname( Next_color ) );
end;

do Next_color = last (:Color:)
                repeat ordinalpred( Next_color)
                until (Next_color = first(:Color:);

  display( ordinalname( Next_color));
end;
```

最初のループのサンプル出力は以下のようになります。

```
RED
ORANGE
YELLOW
GREEN
BLUE
INDIGO
VIOLET
```

### 例 2: 序数を配列と併用する

序数は、配列の下位境界または上位境界を含めて、 配列への索引としては使用できず、変数のエクステントとして定義することもできません。 ただし、BINARYVALUE 組み込み関数によって序数を 2 進値に変換することは可能です。 この関数が返す値を使用して、 配列の索引を作成したり、エクステントを定義することができます。

例えば、以下のパッケージは、各カラーの使用回数を保持するために配列 usage_count を定義し、この配列を更新するためにプロシージャー Record_usage を定義し、この配列内の値を表示するためにプロシージャー Show_usage を定義します。

```
Usage: package exports(*);

  define ordinal Color ( Red,
                         Orange,
                         Yellow,
                         Green,
```

```
                          Blue,
                          Indigo,
                          Violet );

  dcl Usage_count(   binvalue( first(:Color:))
                   : binvalue( last(:Color:)) )
         static fixed bin(31) init( (*) 0 );
              /* first(:Color:)  = Red    */
              /* last(:Color:) = Violet */

Record_usage: proc (Wall_color );
  dcl Wall_color type Color parm byvalue;

  Usage_count( binvalue(Wall_color) )
      = 1 + Usage_count( binvalue(Wall_color) );
end Record_usage;

Show_usage: proc;
  dcl Next_color type Color;

  do Next_color = Red upthru Violet;
    put skip list( ordinalname( Next_color) );
    put list( Usage_count( binvalue(Next_color) ));
  end;
end Show_usage;

end Usage;
```

### 例 3: 序数を使用して関数を作成する

序数を使用すれば、保守および強化が容易で、テーブル・ルックアップと同じくらい効率的な関数を作成することができます。

下記の例では、関数 Is_mellow により、 カラーが「柔らかい」かどうかを示すビットが返されます。 もっと多くのカラーが定義される場合は、 SELECT グループに「柔らかい」カラーを追加できます。 SELECT グループでは、自作したテーブルとは異なり、カラーの順序は DEFINE ステートメントでの順序と同じである必要はなく、まったく特定の順序になっていなくてもかまいません。

SELECT グループ内部のステートメントは すべて定数値を返す RETURN ステートメントですから、 コンパイラーが SELECT グループ全体をテーブル参照に変換します。

```
Is_mellow: proc( Test_color ) returns( bit(1) aligned );

  dcl Test_color type Color parm byvalue;

  select (Test_color);
    when( Yellow, Indigo)
        return( '1'b );
    otherwise
        return( '0'b );
  end;

end;
```

この機能を使用して、ユーザー独自の ORDINALNAME 組み込み関数を作成することも可能です。 この独自の関数は、各序数値について、ユーザーが表示させたい名前を返すことができます。 例えば、下記の関数 Color_name は、 最初の文字を大文字とする名前に関連付けられたカラー名を返します。

```
Color_name: proc( Test_color ) returns( char(8) varying );

  dcl Test_color type Color parm byvalue;

  select (Test_color);
    when ( Blue   ) return( 'Blue');
    when ( Green  ) return( 'Green');
    when ( Orange ) return( 'Orange');
    when ( Red    ) return( 'Red');
    when ( Yellow ) return( 'Yellow');
    otherwise return ('');
  end;

end;
```

# タイプ付き関数

タイプ名は宣言名とは別のネーム・スペースにあるので、 それらを変数参照が必要な所で使用することはできません。 特に組み込み関数の引数として使用することはできません。 ただし、タイプ名はタイプ付き関数 への 引数として使用することはできます。 (ANSI 用語では、これらのタイプ付き関数は照会関数 として 知られています。)

これらのタイプ付き関数は、146 ページの表 37 にまとめられています。

表 *37.* タイプ付き関数

| 関数 | 説明 |
|---|---|
| BIND | ポインターをタイプのハンドルに変換します。 |
| CAST | C の変換規則を使用して、式を指定のタイプに変換します。 |
| FIRST | 最初の値を序数セットに戻します。 |
| LAST | 最後の値を序数セットに戻します。 |
| NEW | 構造体タイプのためにストレージを獲得し、 獲得されたストレージにハンドルを戻します。 |
| RESPEC | 式のビット・パターンを変更せずに、式の属性を指定のタイプに変更します。 |
| SIZE | タイプを表すために必要なストレージの量を戻します。 |
| VALUE | 対応する構造体型を持つ変数の初期化または代入を行います。 |

**関連情報**

575 ページの『タイプ付き関数』
タイプ付き関数を使用して、定義されたタイプを操作できます。 この章では、タイプ付き関数について説明します。

# 第 7 章 データ宣言

PL/I プログラムを実行すれば、特定のデータ・タイプのさまざまなデータ項目が処理されます。 名前が付いていない算術定数やストリング定数を除き、プログラム内ではデータ項目はすべて名前によって参照されます。 各データ名には、(明示的または暗黙的な) 宣言によって、属性と意味が与えられます。 この章では、明示宣言および暗黙宣言、スカラー、配列、 構造体、および共用体の宣言、名前の有効範囲、データの位置合わせ、 デフォルトの属性を説明します。

データ項目のほとんどの属性は、プログラムがコンパイルされるときに判別されます。 非静的項目の場合は、属性の値 (配列の次元の境界、ストリングの長さ、 区域のサイズ、初期値) と一部のファイル属性は、プログラムの実行時に判定されます。 詳細については、89 ページの『ブロックの活動化』を参照してください。

データ項目、タイプ、および属性については、15 ページの『第 2 章 データ・エレメント』で紹介しています。

## 明示宣言

明示宣言 は、DECLARE ステートメントで、ラベル接頭部として、またはパラメーター・リストに、ID (名前) が示されることを言います。

以下のように名前が示されている場合、名前は明示的に宣言されています。

- DECLARE ステートメントに示されている場合

  DECLARE ステートメントが明示的に名前の属性を宣言します。
- 入り口定数として示されている場合

  PROCEDURE ステートメントおよび ENTRY ステートメントのラベルは、含まれるプロシージャー内の入り口定数の宣言を制定します。
- ラベル定数として示されている場合

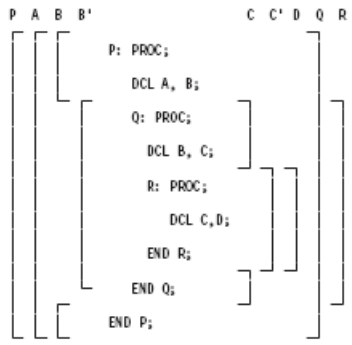
  ラベル定数は、明示的にラベルを宣言します。
- フォーマット定数として示されている場合

  FORMAT ステートメントのラベルは、ラベルの明示宣言を制定します。

**注記 :**

1. DECLARE ステートメントでの内部項目定数、ラベル定数、またはフォーマット定数の命名は無効です。
2. ラベル定数またはフォーマット定数に境界がある場合、その境界は、ソース・コードでラベルとして使用する際に指定された最小値および最大値によって判別されます。

名前の明示宣言の有効範囲は、宣言を含むブロックです。 つまり、同じ名前のもう 1 つの明示宣言が内部にあるブロック (および、 それらに含まれるブロック) を除いた、名前が含まれるすべてのブロックです。 次の図では、ラインで名前の宣言の有効範囲を示します。




© Copyright IBM Corp. 1999, 2022

B と B' は、 名前 B の 2 つの別個の使用法について示しています。 C と C' は、 名前 C の 2 つの使用法について示しています。

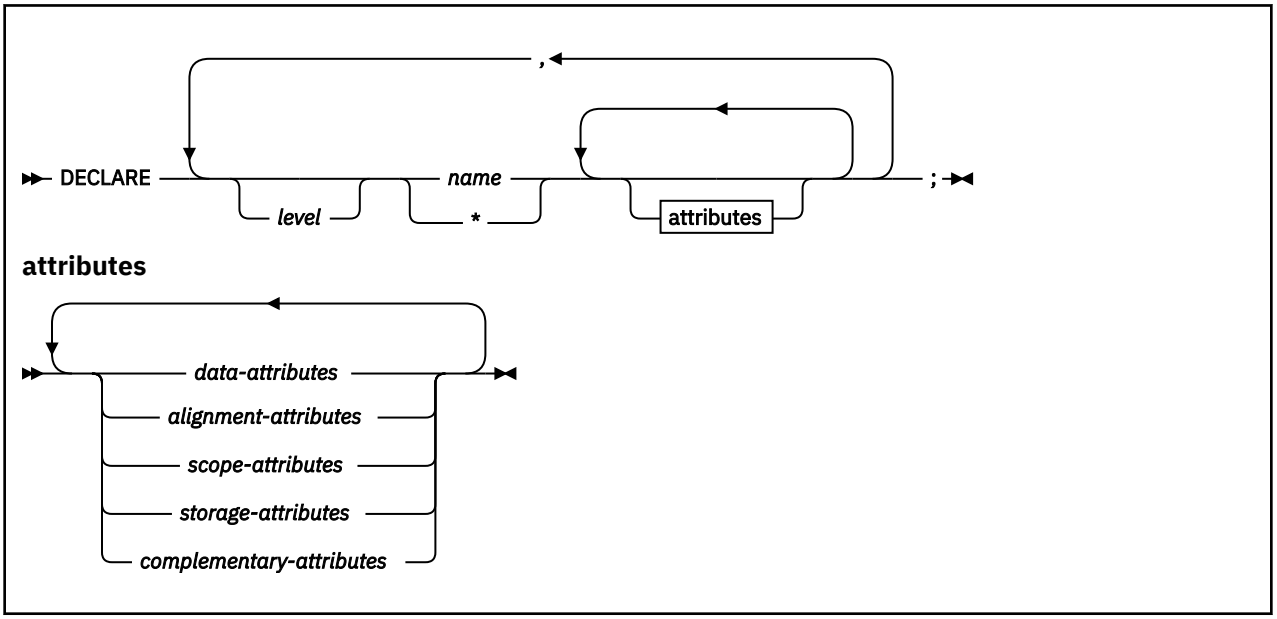
**関連情報**

150 ページの『宣言の有効範囲』
名前の適用対象となるプログラム部分は、その名前についての宣言の有効範囲 と呼ばれます。 ほとんどの場合、名前の宣言の有効範囲は、プログラム内で 名前が宣言される位置によって完全に決定されます。

# DECLARE ステートメント

DECLARE ステートメントは、 名前の一部の属性またはすべての属性を指定します。 属性が明示的に宣言されず、コンテキストによって判別できない場合は、デフォルト属性が適用されます。

DECLARE ステートメントは、プログラムの文書の重要な部分になることがあります。 同様に、デフォルト属性で十分なとき、または暗黙宣言が可能なときであっても、 多数の宣言を使用することができます。 DECLARE ステートメントの数に関しては制限がないので、さまざまな名前のグループに異なる DECLARE ステートメントを使用することができます。 どんなに多くの名前でも、1 つの DECLARE ステートメント内で 宣言することができます。




**省略形**: DCL

配列、構造体、および共用体の詳細については、170 ページの『配列』、174 ページの『構造体』、および 175 ページの『共用体』を参照してください。

*****

EXTERNAL 属性に環境名を指定しない限り、INTERNAL または EXTERNAL スカラーの *name* として、あるいはレベル 1 の EXTERNAL 構造体または共用体として使用することはできません (152 ページの『INTERNAL 属性と EXTERNAL 属性』 を参照)。

**属性**

属性は、任意の順序で示すことができます。

名前に明示的に与える属性はすべて、DECLARE ステートメント内に一緒に 宣言されていなければなりません。ただし、以下はこの例外となります。

FILE 属性を持つ名前には、OPEN ステートメントで属性を与える (あるいは 暗黙のオープンによって属性を暗黙に与える) こともできます。
パラメーター・リスト内の名前には、パラメーター属性がコンテキストによって与えられます。 そのブロックの内部にある DECLARE ステートメントで、その他の属性を指定することができます。

それぞれ別個のブロックとコンパイルの外部名の属性は整合性を必要とします。

グループの属性とメンバーについては、16 ページの『データ・タイプとその属性』を参照してください。

**level**
ゼロ以外の整数値です。 レベル番号を指定しないと、レベル *1* が 要素変数と配列変数のためのデフォルトとなります。 レベル *1* は、大構造体と共用体の名前について指定しなければなりません。

**名前**
レベル 1 のそれぞれの名前は、 ブロック内で固有でなければなりません。 レベル 1 の名前の詳細については、174 ページの『構造体』を参照してください。

条件接頭語とラベルは、DECLARE ステートメント上には指定できません。

**関連情報**
281 ページの『OPEN ステートメント』
OPEN ステートメントは、ファイルをデータ・セットに関連づけます。 これによって、OPEN ステートメントで指定された属性と DECLARE ステートメント で指定された属性が組み合わされます。 また、オープンされるファイルに必要な属性の一部がまだ宣言されていない場 合は、このステートメントでファイルの属性の指定を完成させることができます。

## 属性分配

いくかの名前に共通する属性を分配して、 同じ属性を繰り返し指定する手間を省くことができます。 属性分配を行うには、名前を括弧で囲み、それらすべての名前に適用される属性のセットをその後に続けます。

属性分配はネストできます。 また、次元属性を分配することができます。 属性分配は、構造体および共用体の基本名でも使用することができます。 分配されるレベル番号は、括弧で囲まれたリストの前に付けます。

括弧で囲まれたリスト内の名前は、コンマで区切られます。 分配される属性によって名前を指定変更することはできませんが、リスト内の任意の名前には、分配される属性と対立しない限り、他の属性を指定することができます。

**例**

下記の例は属性分配を示しています。 最後の宣言は、ネストされた属性分配を示しています。

```
declare (A,B,C,D) binary fixed (31);

declare (E decimal(6,5), F character(10)) static;

declare 1 A, 2(B,C,D) (3,2) binary fixed (15);

declare ((A,B) fixed(10),C float(5)) external;
```

# 暗黙宣言

プログラム内で使用する名前を明示的に宣言しなかった場合、その名前は暗黙のうちに宣言されます。 暗黙宣言の有効範囲は、その名前が使われている外部プロシージャーの PROCEDURE ステートメントのすぐあとで DECLARE ステートメントによってその名前を宣言した場合と同様にして決まります。

ファイル、入り口、組み込み関数を除き、暗黙宣言は、名前が最外部の プロシージャーで宣言された場合と同様の効果を持ちます。 ファイルと組み込み関数については、暗黙宣言は、名前が任意の プロシージャーの外側の論理パッケージに宣言された場合と同様の効果を持ちます。

**注 :** 組み込み関数およびファイル SYSIN と SYSPRINT 以外のものに 暗黙宣言を使用するのは、1987 ANSI 規格に対する違反であり、 避けなければなりません。

暗黙のうちに宣言された名前の属性の中には、その名前が使われているコンテキスト上から判定することができます。 コンテキスト宣言 と呼ばれるこれらの事例は、以下のようにリストされます。

- 組み込み関数の名前。
- CALL ステートメント、または INITIAL の CALL オプション内に示される名前、 あるいは引数リストの前にくる名前には、 ENTRY 属性と EXTERNAL 属性が与えられます。

- PROCEDURE ステートメントまたは ENTRY ステートメントの パラメーター・リストに示される名前には、 パラメーター属性が与えられます。
- FILE または COPY オプションにおいて示される名前、もしくはファイル名が 必要な条件に対するステートメント ON、SIGNAL または REVERT に示される 名前には、FILE 属性が与えられます。
- ステートメント ON CONDITION、SIGNAL CONDITION、または REVERT CONDITION に示される名前には、CONDITION 属性が与えられます。
- BASED 属性、SET オプションに示される名前、またはロケーター修飾記号の左側に 現れる名前には、POINTER 属性が与えられます。
- IN オプションまたは OFFSET 属性に示される名前には、AREA 属性が与えられます。

コンテキスト宣言ではない暗黙宣言は、165 ページの『属性のデフォルト』に示すように、 すべてデフォルトによって属性を取得します。 コンテキスト宣言は明示宣言の有効範囲内には存在することができないため、 明示宣言内の名前について確立された属性に、 名前のコンテキストが別の属性を追加することはできません。

### コンテキスト宣言の例

次の例では、PREQ に FILE 属性が、また S に AREA 属性が提供されます。

```
read file (PREQ) into (Q);

allocate X in (S);
```

## 宣言の有効範囲

名前の適用対象となるプログラム部分は、その名前についての宣言の有効範囲 と呼ばれます。 ほとんどの場合、名前の宣言の有効範囲は、プログラム内で 名前が宣言される位置によって完全に決定されます。

暗黙宣言は、名前が外部プロシージャーの PROCEDURE ステートメントの 直後にある DECLARE ステートメントで宣言されたものと見なされます。

名前がプログラム全体に対して同じ意味を持つ必要はありません。 ブロック内で明示的に宣言された名前は、そのブロック内だけで意味を持ちます。 そのブロックの外部では、同じ名前がそこでも宣言されていない限りその名前は認識されません。 名前を宣言するたびに、その有効範囲が確立されます。この場合、その外側ブロック内の名前は、別のデータ項目を参照することになります。 このため、ローカル定義を指定することにより、プログラムのほかの部分で使用されているすべての名前を知らなくても、プロシージャーまたは開始ブロックを作成することが可能になります。

以下の例では、A の出力は実際には C.A (2) です。 プロシージャー X で宣言されているように、B の出力は 1 です。

```
X: proc options(main);
  dcl (A,B) char(1) init('1');
  call Y;
  return;

  Y: proc;
    dcl 1 C,
        3 A char(1) init('2');
    put data(A,B);
    return;
  end Y;
end X;
```

したがって、ネストされたプロシージャーでは、PL/I は現行ブロック内に宣言された変数を使用してから、包含ブロックで宣言された変数を使用することになります。

名前の宣言の有効範囲を理解するためには、 包含された および内部にある という 2 つの 用語を理解しなければなりません。

PACKAGE、PROCEDURE、または BEGIN ステートメントから、 それに対応する END ステートメント (BEGIN、PACKAGE、 および PROCEDURE ステートメントの条件接頭語も含む) までの 間にあるブロックの全テキストは、そのブロックに包含されると 見なされます。ただし、ブロックに適用する任意の ENTRY

ステートメントのラベルと同様に、ブロックの先頭にある BEGIN または PROCEDURE ステートメントのラベルは、そのブロックに含まれません。 ネストされたブロックは、それらを包むブロックに包含されます。

ブロックに包含されるテキストのうち、そのブロック内の ネストされているほかのブロックに包含されていないものは、 そのブロックの内部にあると見なされます。プロシージャーの入り口名 (および BEGIN ステートメントのラベル) は、そのブロックに含まれません。 したがって、プロシージャーの入り口名は、包含ブロックの内部にあることになります。 外部プロシージャーの入り口名は、その外部プロシージャーの外部にあると見なされます。

151 ページの図 2 は、データ宣言の有効範囲を示しています。

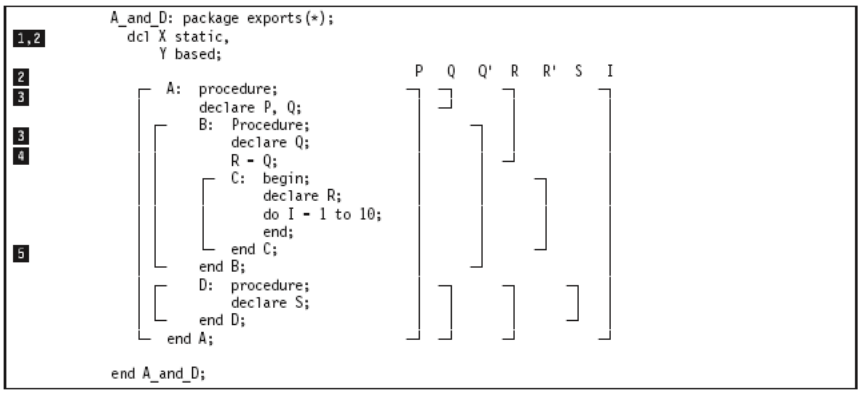



図 *2.* データ宣言の有効範囲

左方の大括弧はブロック構造体を示しています。右方の大括弧は名前の各宣言の有効範囲を示しています。 Q と R の 2 つの宣言の有効範囲は、 Q と Q'、 および R と R' となっています。

X と Y は、 パッケージに包含されるすべてのプロシージャーにとって可視となっています。

**1**

P はブロック A で宣言され、 再宣言されていないため、A 全体で認識されます。

**2**

Q はブロック A で宣言され、ブロック B で再宣言されます。 Q の最初の宣言の有効範囲は、B を除くすべての A です。Q の 2 番目の宣言の有効範囲は、ブロック B のみです。

**3**

R はブロック C で宣言されていますが、 R への参照がブロック B で行われています。 ブロック B での R への参照により、 外部プロシージャーの A で R の暗黙宣言が行われます。 したがって、異なる有効範囲を持つ 2 つの別個の名前 (151 ページの図 2 内の R および R') が存在することになります。 明示的に宣言された R の有効範囲は、ブロック C です。 ブロック B 内の暗黙的に 宣言された R の有効範囲は、 ブロック C を除く A のすべてです。

**4**

I はブロック C で参照されています。 このため、外部プロシージャー A で 暗黙宣言されていることになります。 A この結果、包含されたプロシージャー B、C、 および D を含むすべての A に宣言が適用されます。

**5**

S はプロシージャー D で明示的に宣言され、 D の内部でのみ認識されます。

152 ページの図 3 は、入り口定数とステートメント・ラベルの宣言の有効範囲を示しています。

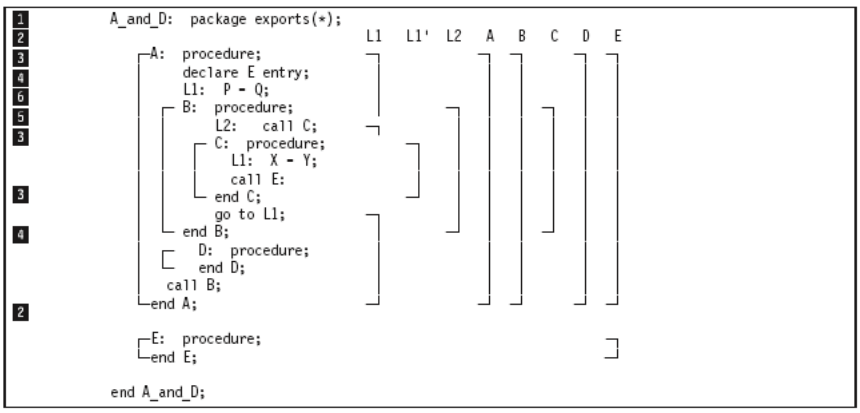



図 *3.* 入り口およびラベル宣言の有効範囲

152 ページの図 3 は、外部プロシージャーの A と E を 示しています。

**1**

名前 A の宣言の有効範囲は、ブロック A のすべてに 限られており、E は有効範囲ではありません。

**2**

E は A で外部入り口定数として明示的に宣言されています。 E の明示宣言は、ブロック A 全体に適用されます。 ブロック E 全体に適用される E の明示宣言にはリンクされ t ていません。 名前 E の宣言の有効範囲は、ブロック A の 全体とブロック E の全体です。

**3**

ラベル L1 は、A および C の内部ステートメントに示されています。 その結果、2 つの別々の宣言が確立されます。 最初の宣言は、ブロック C を除くブロック A の すべてに適用され、2 番目の宣言はブロック C にのみ適用されます。 したがって、ブロック B の GO TO ステートメントが実行されると、制御はブロック A の L1 に移され、 ブロック B は終了します。

**4**

D と B は、明示的にブロック A 内で宣言され、A 内の任意の場所で参照することができます。 ただし、それらは INTERNAL であり、 ブロック E 内で参照することはできません。

**5**

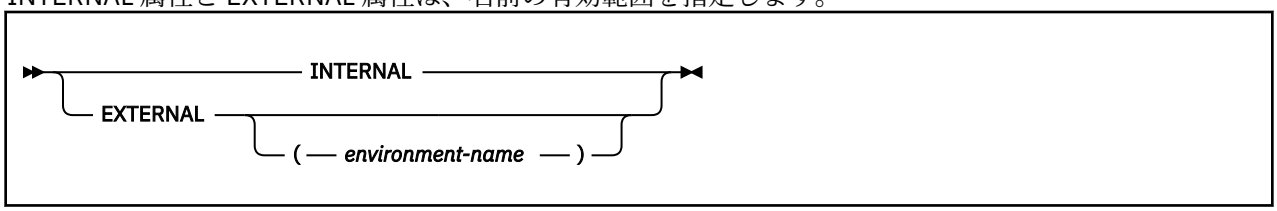
C は B で明示的に宣言され、 B の内部から参照できますが、B の外側からは参照できません。

**6**

L2 は B で宣言され、 ブロック B、および B に 包含される C で参照できますが、 B の外側からは参照できません。

## INTERNAL 属性と EXTERNAL 属性

INTERNAL 属性と EXTERNAL 属性は、名前の有効範囲を指定します。




**省略形:** INTERNAL は INT、EXTERNAL は EXT

**environment-name**
コンパイル単位の外側で認識されるプロシージャー名または変数名を指定します。

この指定により、宣言中の名前は実質的に内部となり、コンパイル単位の外側では認識されなくなります。 宣言中の名前の代わりに環境名が知られるようになります。

環境名は文字ストリング定数でなければならず、 大文字への変換なしにそのまま使用されます。

次の例を参照してください。

```
dcl X entry external ('koala');
```

環境名の先頭に区切り文字 (_) を使用することはできません。この文字で始まる名前は、ライブラリー用に予約されています。

リンカーが環境名を修飾するプラットフォームでは、外部属性を持つ 環境名を指定しても、その名前が変数名と大文字/ 小文字の違いしかない場合は、その環境名は修飾されます。 次の宣言について考えてみましょう。

```
dcl abc ext('kLm'), xyz ext('xYz' );
```

xyz の名前が修飾されます。 環境名の装飾について詳しくは、「*PL/I for Windows* プログラミング・ガイド」の『"呼び出し規則"』の章の『"リンケージについての考慮事項の理解"』トピックを参照してください。

INTERNAL は、内部プロシージャーの入り口名および他のすべての変数 (入り口定数、ファイル定数、およびプログラマー定義の条件を除く) のデフォルトです。 INTERNAL は、その名前が、それを宣言しているブロック内でだけ認識されることを指定します。 別の場所で同じ名前を明示的に宣言すると、別の有効範囲を持つ新しい オブジェクトを参照することになります。これらの 2 つの有効範囲は 相互に重なることはありません。

**注 :** INTERNAL は、パッケージ内のレベル 1 プロシージャー上で指定することができます。 パッケージが EXPORTS(*) を指定して宣言してある場合、INTERNAL プロシージャーはパッケージの外部から不可視になります。

EXTERNAL は、ファイル定数、(内部プロシージャー以外の) 入り口定数、およびプログラマー定義の条件のデフォルトです。 EXTERNAL 属性を持つ名前は、別々の外部プロシージャー内で、または外部プロシージャーに含まれているブロック内で複数回宣言することができます。 EXTERNAL 属性を持つ同じ名前を 2 回以上宣言した場合、それらの宣言はすべて同じデータを参照します。 EXTERNAL 属性を持つ名前の各宣言の有効範囲は、そのアプリケーション内でのその名前 (EXTERNAL 属性を持つもの) の すべての宣言の有効範囲を包含します。

大構造体または共用体の名前を複数のブロックで EXTERNAL として 宣言する場合、対応するメンバー名は異なっていてもかまいませんが、 メンバーの属性は同じでなければなりません。

次の例を考えてください。

```
ProcA: procedure;
  declare 1 A external,
            2 B,
            2 C;
            .
            .
            .
   end ProcA;


%process;
ProcB: procedure;
  declare 1 A external,
          2 B,
          2 D;
          .
          .
          .
  end ProcB;
```

A.B が ProcA 内で変換されると、 それは ProcB についても変換され、逆の場合も同様です。 A.C が ProcA 内で変換されると、 A.D は ProcB について変換され、逆の場合も同様です。

構造体と共用体のメンバーは、常に INTERNAL 属性を持ちます。

同じ名前の外部宣言はすべて同じデータを共用するため、それらの宣言の結果はすべて同じセットの属性にならなければなりません。 EXTERNAL の名前が別の外部プロシージャーで宣言されるときは、ユーザーは属性が確実に一致するようにしなければなりません。 各種宣言の有効範囲の例 は、さまざまな宣言とその有効範囲を説明しています。

### 各種宣言の有効範囲の例

```
          Scope_Example:  package exports(*);
 1          A: procedure;
 2             declare S character (20);
 7             dcl Set entry(fixed decimal(1)),
 7                 Out entry(label);
               call Set (3);
 9          E: get list (S,M,N);
 8          B: begin;
 4,5             declare X(M,N), Y(M);
                 get list (X,Y);
                 call C(X,Y);
```

```
 9,5        C:  procedure (P,Q);
                   declare
                     P(*,*),
                     Q(*),
 12,2                S binary fixed external;
                   S = 0;
 6                 do I = 1 to M;
                     if sum (P(I,*)) = Q(I) then
 8                      go to B;
                     S = S+1;
                     if S = 3 then
 9                      call Out (E);
                     Call D(I);
 8          B:  end;
             end C;
```

```
 9          D:  procedure (N);
                  put list ('Error in row ',
 2,3                    N, 'Table Name ', S);
             end D;
             end B;
             go to E;
             end A;
 9         Out:  procedure (R);
                 Declare
                   R Label,
 11                (K static internal,
 11,7                 L static external) init (0),
 12                S binary fixed external,
                   Z fixed decimal(1);
                 K = K+1; S=0;
                 if K<L then
                   stop;
 10              else go to R;
           end;
           Set:  procedure (Z);
                 declare Z fixed dec(1);
 7               L=Z;
                 declare L external init(0);
                 return;
                 end;
            end Scope_Example;
```

**1**

A は外部プロシージャー名です。 この有効範囲は、ブロック A の全体と、 A が外部として宣言されているほかの任意のブロックです。

**2**

S は、ブロック A およびブロック C で明示的に宣言されます。 文字変数宣言は、ブロック A (ブロック C を除く) のすべてに適用されます。 固定 2 進値宣言は、ブロック C の内部でのみ適用されます。 D はブロック C から呼び出されますが、D 内の PUT ステートメントでの S への参照は文字変数 S への参照であり、ブロック C 内で 宣言されている S への参照ではありません。

**3**

N はブロック D 内のパラメーターとして 書かれていますが、このブロックの外でも使用されます。 パラメーターとして書かれているため、D 内で N の明示宣言が確立されます。 D の外側で参照すると、ブロック A 内で N の暗黙宣言が行われます。 名前 N のこれらの 2 つの宣言は異なるオブジェクトを参照しますが、この場合、オブジェクトは同じデータ属性 (デフォルトでは FIXED BINARY(15,0) と INTERNAL) を持ちます。 DEFAULT(ANS) の場合の精度は (31,0) です。

**4**

X と Y は B 全体で認識され、B 内部のブロック C または D から 参照できます。ただし、B の外側の A 部分からは 参照できません。

**5**

P と Q はパラメーターです。 したがって、ブロック内でこれらの名前の宣言がほかになければ、 パラメーター・リストに指定されているだけで コンテキスト宣言がされていることになります。 しかし、P と Q が配列であることを 指定するには、別個の明示宣言が必要です。 引数 X と Y は配列として宣言されており、ブロック C 内で認識されていますが、P と Q も 配列であることを指定するには、DECLARE ステートメントでそれらを宣言しなければなりません。 (アスタリスク表記は、パラメーターの境界が引数の境界と 同じであることを示します。)

**6**

I および M は、外部プロシージャー A で明示的に宣言されません。 したがって、I がブロック C 内にしか出現しなくても、それらは暗黙的に宣言され、A 全体で認識されます。

**7**

例の中の外部プロシージャー Out と Set には、これら両方に共通の L という外部宣言があります。 また、プロシージャー A で ENTRY 属性を使用して明示的に宣言する必要があります。 ENTRY は EXTERNAL を暗黙指定するため、2 つの入り口定数 Set および Out は、2 つの外部プロシージャー全体で認識されます。

**8**

ラベル B はこのプログラムに 2 回現れます。 最初は A 内で開始ブロックのラベルとしてであり、 明示宣言となります。次にブロック C 内のラベルとして宣言され、END ステートメントの接頭部として使用されます。 したがって、ブロック C 内の go to B ステートメントは、ブロック C 内の END ステートメントのラベルを参照します。 ブロック C の外側では、B への参照は開始ブロックのラベルへの参照となります。

**9**

B ロック C および D は、B 内の任意のポイントから呼び出すことができますが、A 外部の B のその部分から呼び出すことも、別の外部プロシージャーから呼び出すこともできません。 同様に、ラベル E は外部プロシージャー A 全体で 認識されているため、E への制御の伝送は A 内部の どこからでも行うことができます。 ただし、ブロック C 内のラベル B は、C 内からのみ参照できます。 GO TO ステートメントによってブロック外に転送することはできますが、一般的にネストされたブロックに転送することはできません。 この例外となるケースは外部プロシージャー Out に 示されています。ここでは、ラベル E が ブロック C から引数として ラベル・パラメーター R に渡されています。

GET ステートメントと PUT ステートメントにはファイルが 指定されていないため、SYSIN と SYSPRINT は 暗黙宣言されていることに注意してください。

**10**

ステートメント else go to R; は、制御をラベル E に伝送します。E が A 内で宣言 されており、Out 内では知られていない場合でも同様です。

**11**

変数 K (INTERNAL) と L (EXTERNAL) は、Out プロシージャー・ブロック内で STATIC として宣言されています。 それらの値は Out への呼び出しの間で保護されています。

**12**

プロシージャー Out の中の S を プロシージャー C の中の S と同じものとして識別するために、どちらも属性 EXTERNAL を付けて宣言されています。

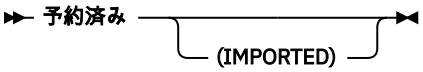
## RESERVED 属性

RESERVED 属性は、STATIC EXTERNAL を暗黙指定します。

さらに、変数が RESERVED 属性を持つ場合、アプリケーションは以下の条件に従います。

- 変数のすべての宣言は、RESERVED を指定しなければなりません。
- 変数名は、必ず 1 つのパッケージの RESERVES オプション中に 現れなければなりません。

変数が RESERVED 属性を持つ場合、 その変数がパッケージ内に保存されていない限り、INITIAL 値は無視されます。




コンパイル単位に RESERVED 属性を持つ変数があり、そのコンパイル単位がその変数の予約パッケージではない場合、そのコンパイル単位は、予約パッケージを含むロード・モジュールの一部であるか、予約パッケージを含む別のロード・モジュールから変数をインポートする必要があります。 後者の場合は、以下の条件が適用されます。

- 変数の宣言は、RESERVED(IMPORTED) 属性を指定しなければなりません。
- 変数は、DLL からエクスポートされなければなりません。
- DLL に関連付けられたサイド・ファイルは、インポートするモジュールのリンク中に組み込まれる必要があります。

### 例

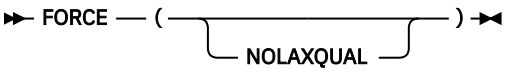
次の例では、パッケージ owns_x が変数 x 用のストレージを予約して初期化します。 パッケージ owns_y と同じロード・モジュールにリンクする必要があります。 このロード・モジュールは、パッケージ owns_z がリンクされているロード・モジュールから変数 z をインポートする必要があります。

```
owns_x:
  package
  exports(*)
  reserves(x);

  dcl x char(256) reserved init( ... );
  dcl y char(256) reserved init( ... );
  dcl z char(256) reserved(imported) init( ... );

end;

owns_y:
  package
  exports(*)
  reserves(y);

  dcl x char(256) reserved init( ... );
  dcl y char(256) reserved init( ... );
  dcl z char(256) reserved(imported) init( ... );

end;

owns_z:
  package
  exports(*)
  reserves(z);

  dcl z char(256) reserved(imported) init( ... );

end;
```

# FORCE 属性

FORCE 属性を使用して、各種メッセージを発行するようコンパイラーに指示できます。

FORCE 属性の構文図を以下に示します。



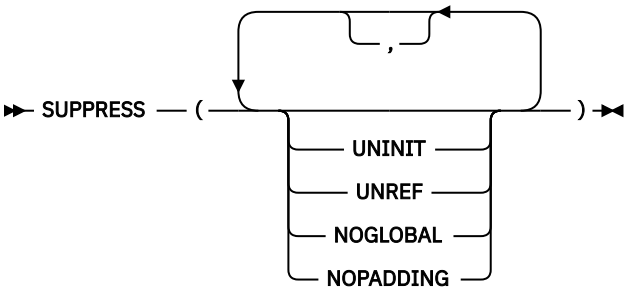

**NOLAXQUAL**
変数で FORCE(NOLAXQUAL) を指定すると、コンパイラーは、正しく修飾されていない変数の参照にフラグを立てます。

構造体または共用体で FORCE 属性を指定すると、その構造体または共用体のすべての要素にも FORCE 属性が適用されます。

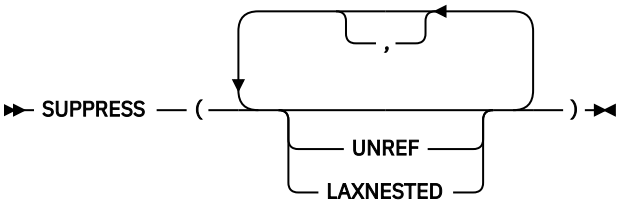
## SUPPRESS 属性

SUPPRESS 属性を使用して、各種メッセージを発行しないようにコンパイラーに指示することができます。

変数に対して SUPPRESS 属性を使用する構文図を次に示します。




プロシージャー・ステートメントに対して SUPPRESS 属性を使用する構文図を次に示します。




**UNINIT**
変数の宣言内で属性として SUPPRESS(UNINIT) が指定されている場合、この変数が未初期化のまま使用されている可能性があることを示すためにコンパイラーがフラグを立てることはありません。

**UNREF**
変数の宣言内で属性として SUPPRESS(UNREF) が指定されている場合、 この変数への参照がコンパイル単位に含まれていなくても、コンパイラーが変数に未使用のフラグを立てることはありません。

PROCEDURE ステートメントで SUPPRESS(UNREF) が指定されている場合、 このプロシージャーへの参照がコンパイル単位に含まれていなくても、コンパイラーがプロシージャーに未使用のフラグを立てることはありません。

**NOGLOBAL**
変数の宣言内で属性として SUPPRESS(NOGLOBAL) を指定することによって、 コンパイラーが、ネストされたプロシージャー内の変数の使用にフラグを立てることはありません。

構造体または副構造体で SUPPRESS(NOGLOBAL) 属性が指定された場合、そのすべての子に伝搬されます。

**NOPADDING**
レベル 1 構造体名に SUPPRESS(NOPADDING) を指定すると、その構造体に関しては RULES(NOPADDING) コンパイラー・オプションは無視されます。

構造体内のメンバーで指定された場合、SUPPRESS(NOPADDING) には効果がありません。

**LAXNESTED**
プロシージャーで SUPPRESS(LAXNESTED) が指定されている場合、プロシージャーの後ろに実行可能ステートメントがあっても、コンパイラーがプロシージャーにフラグを立てることはありません。

構造体または共用体で SUPPRESS 属性が指定されている場合、その構造体 (または共用体) のすべてのエレメントにも SUPPRESS 属性が適用されます。

# データの位置合わせ

コンピューターは情報を 8 ビットの倍数で保持します。 この 8 ビット単位の情報はバイト と呼ばれます。 コンピューターによるアクセスは、1 バイト、ハーフワード、ワード、またはダブルワードを単位として行われます。 ストレージのバイト位置は 0 から開始される連続した番号が付けられます。各番号は対応するバイトのアドレスです。

ハーフワード は連続している 2 バイト、フルワード は連続している 4 バイト、 ダブルワード は連続している 8 バイトをそれぞれ指します。ハーフワード、ワード、およびダブルワードは、それぞれの一番左のアドレスでアドレス指定されます。

ハーフワード、ワード、およびダブルワードが、主記憶域でそれぞれの 情報単位の規定境界に置かれていると、プログラムの実行が高速になります。つまり、158 ページの表 38 に示すように、情報単位のアドレスがその情報のバイト数の倍数となる場合、プログラムの実行が高速になります。

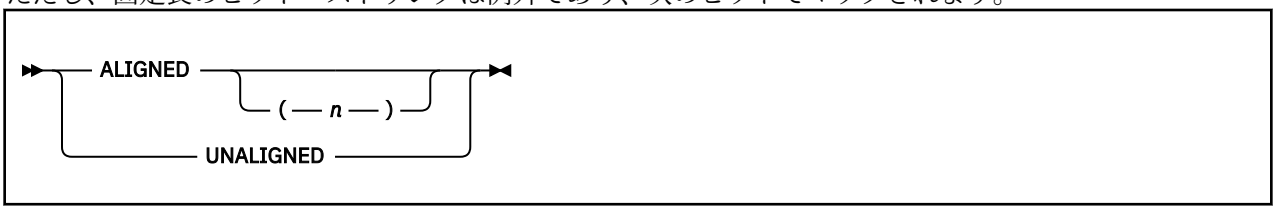
表 *38.* ハーフワード、ワード、およびダブルワードの規定境界の位置合わせ

| ストレージ・セクションのアドレス | | | | | | | |
|---|---|---|---|---|---|---|---|
| 5000 | 5001 | 5002 | 5003 | 5004 | 5005 | 5006 | 5007 |
| バイト | バイト | バイト | バイト | バイト | バイト | バイト | バイト |
| ハーフワード | | ハーフワード | | ハーフワード | | ハーフワード | |
| フルワード | | | | フルワード | | | |
| ダブルワード | | | | | | | |

PL/I は規定境界でのデータ位置合わせが可能です。 ただし、連続するデータ・エレメント間の未使用バイトがあると、 ストレージの使用が増加します。 例えば、データ項目がデータ・セットを作成するために 使用される集合のメンバーである場合、未使用バイトによって補助記憶域の必要量が増えることになります。 ALIGNED 属性と UNALIGNED 属性を使用すれば、適切な規定境界にデータを位置合わせ するかどうかを選択できます。

## ALIGNED 属性と UNALIGNED 属性

ALIGNED を指定すると、データ・エレメントがそのデータ・タイプ要件に合うストレージ境界で位置合わせされます。 UNALIGNED は、各データ・エレメントが次のバイト境界でマップされることを指定します。 ただし、固定長のビット・ストリングは例外であり、次のビットでマップされます。




デフォルトはエレメント・レベルで適用されます。 UNALIGNED は、 ビット・データ、文字データ、グラフィック・データ、uchar データ、 ワイド文字データ、および数字データのデフォルトです。 ALIGNED は、その他のすべてのデータ・タイプのデフォルトです。

ALIGNED(*n*) は、データ・タイプのデフォルトの位置合わせをオーバーライドする位置合わせを指定します。 ALIGNED(*n*) の n の有効な値は 1、2、4、および 8 のみです。

159 ページの表 39 に、ALIGNED 属性の要件をリストします。

**注記 :**

- プログラム制御データの位置合わせおよびストレージ所要量で、 サポートされるシステムを通して変更することができます。
- 複素数データは実数データの 2 倍のストレージを必要としますが、 位置合わせ要件は同じです。

表 39. 位置合わせ要件

| 変数の型 | 内部での格納 | ストレージ所要量 (バイト) | 位置合わせ要件 | |
|---|---|---|---|---|
| | | | ALIGNED データ | UNALIGNED データ |
| BIT(n) | ALIGNED: 8 ビットのグループ (またはその一部) ごとに 1 バイト<br>UNALIGNED: バイト境界とは無関係に、必要なだけのビット数 | ALIGNED: CEIL(n/8)<br>UNALIGNED: n ビット | バイト (データは 0 から 7 の任意のバイトで開始できます) | ビット (データはバイト内の 0 から 7 の任意のビットで開始できます) |

表 *39.* 位置合わせ要件 *(*続き*)*

| 変数の型 | 内部での格納 | ストレージ所要量 (バイト) | 位置合わせ要件 | |
|---|---|---|---|---|
| | | | **ALIGNED データ** | **UNALIGNED データ** |
| CHARACTER(n) | 1 文字につき 1 バイト | n | バイト (データは 0 から 7 の任意のバイトで開始できます) | バイト (データは 0 から 7 の任意のバイトで開始できます) |
| CHARACTER (n)VARYINGZ | 1 文字につき 1 バイト、および、 ヌル終了文字の 1 バイト | n+1 | | |
| GRAPHIC(n) | 1 グラフィックにつき 2 バイト | 2n | | |
| GRAPHIC (n)VARYINGZ | 1 図形文字につき 2 バイト、および ヌル終了文字の 2 バイト | 2n+2 | | |
| UCHAR (n) | 1 uchar につき 4 バイト | 4n | | |
| UCHAR (n) VARYINGZ | 1 uchar につき 4 バイト、および、ヌル終了文字の 1 バイト | 4n+1 | | |
| WIDECHAR (n) | 1 ワイド文字につき 2 バイト | 2n | | |
| WIDECHAR (n) VARYINGZ | 1 ワイド文字につき 2 バイト、 およびヌル終了文字の 2 バイト | 2n+2 | | |
| PICTURE | PICTURE 文字 (V、K、および F 位取り係数を除く) につき 1 バイト | V、K、および F 指定以外の PICTURE 文字の数 | | |
| DECIMAL FIXED (p,q) | パック 10 進フォーマット (1 桁につき 1/2 バイト、および、符号の 1/2 バイ ト) | CEIL((p+1)/2 | | |
| BINARY FIXED(p,q)<br>SIGNED<br>1 <= p <= 7<br>UNSIGNED<br>1 <= p <= 8 | 1 バイト | 1 | | |
| ORDINAL<br>SIGNED<br>1 <= p <= 7<br>UNSIGNED<br>1 <= p <= 8 | | | | |

表 39. 位置合わせ要件 (続き)

| 変数の型 | 内部での格納 | ストレージ所要量 (バイト) | 位置合わせ要件 | |
|---|---|---|---|---|
| | | | ALIGNED データ | UNALIGNED データ |
| BIT (n) VARYING | 2 バイトの接頭部、および、宣言される最大長の各 8 ビット・グループ (またはその一部) のための 1 バイト | ALIGNED:2+CEIL(n/8)<br>UNALIGNED: 2 バイト +n ビット | ハーフワード (データは 0、2、4、または 6 バイトから開始できます) | バイト (データは 0 から 7 の任意のバイトで開始できます) |
| CHARACTER (n) VARYING | 2 バイトの接頭部、および、宣言される最大長の各文字につき 1 バイト | n+2 | | |
| GRAPHIC(n) VARYING | 2 バイトの接頭部、および、宣言される最大長の各グラフィックにつき 2 バイト | 2n+2 | | |
| UCHAR (n) VARYING | 2 バイトの接頭部、および、宣言される最大長の各 uchar につき 4 バイト | 4n+2 | | |
| WIDECHAR (n) VARYING | 2 バイトの接頭部、および、宣言される最大長の各ワイド文字につき 2 バイト | 2n+2 | | |
| BINARY FIXED(p,q)<br>SIGNED<br>8 <= p <= 15<br>UNSIGNED<br>9 <= p <= 16 | ハーフワード | 2 | | |
| ORDINAL<br>SIGNED<br>8 <= p <= 15<br>UNSIGNED<br>9 <= p <= 16 | | | | |

表 *39.* 位置合わせ要件 *(*続き*)*

| 変数の型 | 内部での格納 | ストレージ所要量 (バイト) | 位置合わせ要件 | |
|---|---|---|---|---|
| | | | **ALIGNED データ** | **UNALIGNED データ** |
| BINARY FIXED(p,q)<br>SIGNED<br>16 <= p <= 31<br>UNSIGNED<br>17 <= p <= 32 | フルワード | 4 | フルワード (データは 0 または 4 バイトから開始できます) | バイト (データは 0 から 7 の任意のバイトで開始できます) |
| ORDINAL<br>SIGNED<br>16 <= p <= 31<br>UNSIGNED<br>17 <= p <= 32 | フルワード | | | |
| BINARY FLOAT(p)1<=p<=21 | 短浮動小数点 | | | |
| DECIMAL FLOAT(p)<br>1<=p<=6 if not DFP | 短浮動小数点 | | | |
| DECIMAL FLOAT(p)<br>1<=p<=7 if DFP | 短浮動小数点 | | | |

表 39. 位置合わせ要件 (続き)

| 変数の型 | 内部での格納 | ストレージ所要量 (バイト) | 位置合わせ要件 | |
|---|---|---|---|---|
| | | | ALIGNED データ | UNALIGNED データ |
| BIT(n) VARYING4 | 4 バイトの接頭部、および、宣言される最大長の各 8 ビット・グループ (またはその一部) のための 1 バイト | ALIGNED: 4+CEIL(n/8)<br>UNALIGNED: 4 バイト + n ビット | フルワード (データは 0 または 4 バイトから開始できます) | バイト (データは 0 から 7 の任意のバイトで開始できます) |
| CHARACTER(n) VARYING4 | 4 バイトの接頭部、および、宣言される最大長の各文字につき 1 バイト | n+4 | | |
| GRAPHIC(n) VARYING4 | 4 バイトの接頭部、および、宣言される最大長の各グラフィックにつき 2 バイト | 2n+4 | | |
| UCHAR (n) VARYING4 | 4 バイトの接頭部、および、宣言される最大長の各 uchar につき 4 バイト | 4n+4 | | |
| WIDECHAR(n) VARYING4 | 4 バイトの接頭部、および、宣言される最大長の各ワイド文字につき 2 バイト | 2n+4 | | |
| POINTER(32) | – | 4 | | |
| HANDLE(32) | – | | | |
| OFFSETSIZE(4) での OFFSET | – | | | |
| LP(32) での FILE | – | | | |
| LP(32) での ENTRY LIMITED | – | | | |
| ENTRY | – | 8 | | |
| LABEL または FORMAT | – | | | |
| TASK | – | 16 | | |

表 *39.* 位置合わせ要件 *(*続き*)*

| 変数の型 | 内部での格納 | ストレージ所要量 (バイト) | 位置合わせ要件 | |
|---|---|---|---|---|
| | | | **ALIGNED データ** | **UNALIGNED データ** |
| OFFSETSIZE(4) での AREA | – | 16+size | ダブルワード (データは 0 バイトから開始できます) | AREA データは、位置合わせなしにすることはできません。 |
| OFFSETSIZE(8) での AREA | – | 32+size | | |
| LP(64) での ENTRY LIMITED | – | 8 | | バイト (データは 0 から 7 の任意のバイトで開始できます) |
| POINTER(64) | – | | | |
| HANDLE(64) | – | | | |
| OFFSETSIZE(8) での OFFSET | – | | | |
| LP(64) での FILE | – | | | |
| BINARY FIXED(p,q)<br>SIGNED<br>32 <= p <= 63<br>UNSIGNED<br>33 <= p <= 64 | – | | | |
| BINARY FLOAT (p)<br>22 <= p <= 53 | 長浮動小数点 | | | |
| DECIMAL FLOAT(p)<br>7<=p<=16 if not DFP | | | | |
| DECIMAL FLOAT(p)<br>8<=p<=16 if DFP | | | | |
| BINARY FLOAT (p)<br>54 <= p | 拡張精度の浮動小数点 | 16 | | |
| DECIMAL FLOAT(p)<br>17<=p | | | | |

ALIGNED または UNALIGNED は、エレメント、配列、構造体、 または共用体変数について指定できます。構造体または共用体にいずれかの属性を適用すれば、 明示的には ALIGNED とも UNALIGNED とも宣言されていない すべての被包含エレメントに対してその属性を適用することになります。

以下に、構造体とそのエレメントについての、ALIGNED と UNALIGNED 宣言の効果を図示した例を示します。

```
declare 1 S,
          2 X bit(2),      /* unaligned by default  */
          2 A aligned,     /* aligned explicitly    */
            3 B,           /* aligned from A        */
            3 C unaligned, /* unaligned explicitly  */
              4 D,         /* unaligned from C      */
              4 E aligned, /* aligned explicitly    */
              4 F,         /* unaligned from C      */
            3 G,           /* aligned from A        */
          2 H;             /* aligned by default    */
```

**関連情報**

174 ページの『構造体』
構造体 とは、構造体、共用体、要素変数、 および配列であるメンバー・エレメントの集合です。

175 ページの『共用体』
共用体 とは、相互にオーバーレイし合い、同じストレージを有するメンバー・エレメントの集合です。 メンバーは、構造体、共用体、要素変数、および配列であることができます。 それらは、同一の属性を持っていなくても構わない。

# 属性のデフォルト

PL/I プログラムで使用する名前には、それぞれ完全な属性のセットが必須になります。 プロシージャーに渡された引数は、プロシージャーのパラメーターと一致する属性を持っていなければなりません。 関数から返される値は、予期される属性を持っていなければなりません。 ただし、ユーザーが属性の完全なセットを指定するのはまれです。 属性のデフォルトを使用することができます。

以下の属性には、言語に固有のデフォルト、または DEFAULT ステートメントを使用して定義するデフォルトを使用できます。

- 明示的に宣言された名前の属性
- 暗黙的に宣言された (コンテキストによる宣言を含む) 名前の属性
- パラメーター記述子に含まれる属性
- 関数プロシージャーから返される値の属性

DEFAULT ステートメントを使用して属性のデフォルトを指定し、言語に固有のデフォルトを変更するか、まったく新しいデフォルト・セットを作成することができます。

デフォルトによって適用される属性が、明示宣言またはコンテキスト宣言によって名前に適用された属性に優先することはありません。

## 言語に固有のデフォルト

データ属性を指定しないで宣言された変数には、 デフォルトにより算術属性が与えられます。

モード、スケール、基数が DECLARE ステートメントまたは DEFAULT ステートメントで指定されていない場合は、DEFAULT コンパイラー・オプションが変数属性を以下のように決定します。

- DEFAULT(IBM) が有効な場合は、 文字 I から N で開始される名前を持つ変数は 属性 REAL FIXED BINARY(15,0) が与えられます。 ほかのすべての変数は属性 REAL FLOAT DECIMAL(6) が与えられます。
- DEFAULT(ANS) が有効な場合、どの変数にも属性 REAL FIXED BINARY(31,0) が与えられます。

位取り係数が精度属性に指定されていない場合、 ほかのすべての属性に先行して、FIXED 属性が適用されます。 したがって、BINARY (p,q) 属性を指定した宣言は、 常に FIXED BINARY (p,q) 属性を指定した宣言と同じになります。

算術宣言で精度が指定されていない場合、DEFAULT コンパイラー・オプションにより、166 ページの表 40 に従って精度が決定されます。 有効範囲、ストレージ、および位置合わせ属性に関する言語に固有のデフォルトは、 21 ページの表 8 と 19 ページの表 7 に示されています。

ENTRY 宣言で記述リストが指定されていない場合、引数の属性は、 呼び出されたプロシージャー内のパラメーターの属性と一致しなければなりません。 例えば、以下の宣言を想定します。

```
dcl X entry;
call X( 1 );
```

引数は属性 REAL FIXED DECIMAL(1,0) を持っています。 次の例に示すように、プロシージャー x がそのパラメーターを別の属性で宣言した場合には、 エラーが発生します。

```
X: proc( Y );
  dcl Y fixed bin(15);
```

入り口宣言によってすべてのパラメーターの属性を指定すれば、 こうしたエラーの可能性を回避することができます。

表 *40.* デフォルトの算術精度

| 属性 | DEFAULT(IBM) | DEFAULT(ANS) |
|---|---|---|
| DECIMAL FIXED | (5,0) | (10,0) |
| BINARY FIXED | (15,0) | (31,0) |
| DECIMAL FLOAT | (6) | (6) |
| BINARY FLOAT | (21) | (21) |

## DEFAULT ステートメント

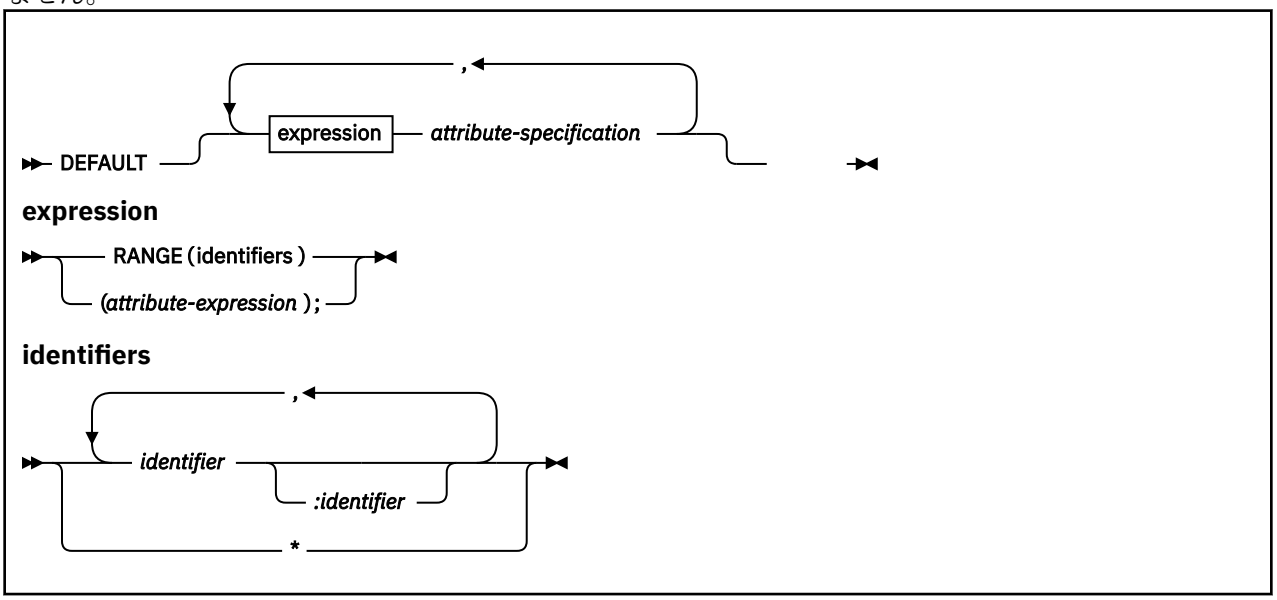
DEFAULT ステートメントは、(属性セットが完全ではない場合は) データ属性のデフォルトを指定します。 部分的に完成された明示宣言やコンテキスト宣言、あるいは暗黙宣言の場合、 DEFAULT ステートメントが適用されない属性については、 言語に固有のデフォルトが適用されます。

DEFAULT ステートメントでは、RANGE 属性およびその他の属性キーワードとともに、論理式を使用できます。 これらの属性は、169 ページの表 41 にリストされています。

DEFAULT ステートメントはほかのすべての属性指定を無効にします。 ただし、ENTRY 属性または FILE 属性を付けて、かつ VARIABLE 属性を暗黙指定する属性は付けずに宣言された名前については、DEFAULT ステートメントの適用に先立ち、PL/I が暗黙の CONSTANT 属性を指定します。 したがって、下記の例の場合は、PL/I によって Xtrn に CONSTANT 属性が 指定され、STATIC 属性は指定されません。

```
Sample: proc;

  default range(*) static;
  dcl Xtrn entry;

end;
```

構造体および共用体のエレメントには、修飾されたエレメント名ではなく、 エレメントの名前に応じてデフォルト属性が指定されます。 DEFAULT ステートメントは、構造体や共用体を作成するためには使用できません。




**省略形**: DFT

**RANGE( identifier )**
指定した ID と同じ文字で始まる名前に デフォルトが適用されるように指定します。 例えば、RANGE (ABC) は以下の名前に適用されます。

```
ABC
ABCD
ABCDE
```

以下の名前には適用されません。

```
ABD
ACB
AB
A
```

したがって、範囲指定に 1 文字の ID を指定した場合は、 その文字で始まるすべての名前に ID が適用されます。 RANGE ID は DBCS で指定できます。

**RANGE( identifier : identifier )**
名前の最初の文字が指定した 2 つの ID と一致するか、 またはアルファベット順で 2 つの ID の中間にある名前に、デフォルトが適用されるように指定します。 文字は DBCS で指定できますが、RANGE 指定が名前に適用されるかどうかの決定において、比較はすべて、含まれる文字の 16 進値のみに基づいています。 指定に使用する文字は、昇順のアルファベット順に並べる必要があります。 次の例を参照してください。

```
RANGE(A:G,I:M,T:Z)
```

**RANGE(*)**
DEFAULT ステートメントの有効範囲内のすべての 名前を指定します。 次の例を考えてください。

```
DFT RANGE (*) PIC '99999';
```

このステートメントは、すべての名前について、 デフォルト属性 REAL PICTURE '99999' を指定します。

**DESCRIPTORS**
以下の条件が真である場合に、明示入り口宣言のパラメーター記述子リストにある任意のパラメーター記述子に属性を組み込むことを指定します。

- 同じクラスの代替属性が存在していても、 この属性の組み込みは禁止されません。
- 少なくとも 1 つの属性がすでに存在している必要があります (DESCRIPTORS の デフォルト属性は、ヌル記述子には適用されません)。

次の例を考えてください。

```
DEFAULT DESCRIPTORS BINARY;
DCL X ENTRY (FIXED, FLOAT);
```

リスト内のそれぞれのパラメーター記述子に属性 BINARY が追加され、 次のように同等なリストが作成されます。

```
(FIXED BINARY, FLOAT BINARY)
```

**attribute-list**
特定の範囲内の名前に適用する属性を選択する際に使う、属性リストを指定します。 リスト内の属性はどの順序で指定してもかまいませんが、 相互にブランクで区切るようにしてください。

この属性リストからは、データ項目の宣言を完結させるために必要な属性だけが取り出されます。

FILE を使用した場合は、属性 VARIABLE と INTERNAL が暗黙指定されます。

次元属性は指定可能ですが、属性指定の最初の項目としてだけ 指定できます。 境界は算術定数または式として指定でき、REFER オプションを 組み込むことができます。 次の例を参照してください。

```
DFT RANGE(J) (5);
DFT RANGE(J) (5,5) FIXED;
```

DEFAULT ステートメントは、明示的に宣言されていない 名前の次元属性を指定できますが、属性 BUILTIN を指定すると、 添え字付きの名前がコンテキスト的に宣言されます。 したがって、デフォルトでは次元属性は明示的に宣言された名前にだけ 適用できます。

INITIAL 属性を指定できます。

データ項目に適用すると相互に対立する属性でも、属性指定では対立するとは限りません。 次の例を考えてください。

```
DEFAULT RANGE(S) BINARY VARYING;
```

この場合、文字 S で始まり、BIT、CHARACTER、または GRAPHIC 属性 を指定して明示的に宣言された名前には VARYING 属性が指定され、 その他のすべての名前 (算術データ以外のものとして明示的にまたはコンテキストで 宣言されていない) には BINARY 属性が指定されます。

**VALUE**

次元属性の前を除いて、attribute-specification の中のどこに 書いてもかまいません。

VALUE は、区域のサイズ、ストリングの長さ、および数値精度に関する デフォルトの規則を確立します。

DEFAULT ステートメントで、VALUE オプションは、区域のサイズ、ストリングの長さ、または数値精度 を指定できる場所にのみ置きます。

VALUE 文節の中のこれらのサイズ、長さ、および精度の指定は、システム・デフォルト属性 の後で、サイズ、長さ、および精度に関するシステム・デフォルトより前に適用されます。 したがって、例えば DCL I; および DEFAULT RANGE(*) VALUE( FIXED BIN(31) ); があるとした場合、変数 *I* はシステム・デフォルト属性である FIXED BINART を受け取りますが、精度 31 は (システム・デフォルトである 15 ではなく) VALUE オプションから受け取ります。

AREA データのサイズや、BIT、CHARACTER、または GRAPHIC データの長さは、 式または整数で表すことができ、REFER オプションを使用することもできます。 また、アスタリスクとして指定することもできます。

次の例を考えてください。

```
DEFAULT RANGE(A:C)
        VALUE (FIXED DEC(10),
               FLOAT DEC(14),
               AREA(2000));
DECLARE B FIXED DECIMAL,
        C FLOAT DECIMAL,
        A AREA;
```

これらのステートメントは、以下の宣言と同等です。

```
DECLARE B FIXED DECIMAL(10),
        C FLOAT DECIMAL(14),
        A AREA(2000);
```

value-specification では、特定の属性における精度指定であることが わかるように、基数属性とスケール属性を記入する必要があります。 基数属性とスケール属性は属性配分することができます (149 ページの『属性分配』を参照のこと)。

VALUE オプションの影響を受ける属性は、区域のサイズ、ストリングの 長さ、および精度だけです。上記の例での CHARACTER や FIXED BINARY など、オプションのその他の属性は、単に値を関連付ける属性を示すに過ぎません。 次の例を考えてください。

```
DEFAULT RANGE(I) VALUE(FIXED DECIMAL(8,3));
I = 1;
```

I が明示的に宣言されていない場合、I には、言語で定められている デフォルトの属性 FIXED BINARY (15,0) が与えられます。 I は DEFAULT ステートメントの影響を受けません。なぜなら、 このステート

メントは、FIXED DECIMAL である名前のデフォルトの精度が (8,3) であることだけを指定しているからです。

例えば、次のステートメントは、FIXED BINARY であることがすでに分かっている ID の精度を指定します。

```
DFT RANGE(*) VALUE(FIXED BINARY(31));
```

一方、次のステートメントは、デフォルトとしての FIXED BINARY 属性と精度の両方を指定します。

```
DFT RANGE(*) FIXED BINARY VALUE(FIXED BINARY(31));
```

***attribute-expression***

オペランドとして属性を指定し、演算子 and、or、および not を使用する式です。 この式が真である場合は、指定された属性が適用されます。 例えば、精度が指定されなかった場合、DEFAULT(FIXED & BIN) PREC(31) は、すべての FIXED BINARY 変数の精度を 31 に設定します。

RANGE 属性を除くすべての属性は、個別のキーワードです。 RANGE 属性は、例えば単純な DEFAULT ステートメント内では、*、*identifier*、または *identifier : identifier* を使用して指定する必要があります。

DEFAULT 内の *attribute-expression* では、次の表にリストした属性を使用できます。

表 *41. DEFAULT* 内の *attribute-expression* に指定する属性

| | | | |
|---|---|---|---|
| ABNORMAL | ENTRY | LIST | RESERVED |
| ALIGNED | ENVIRONMENT | MEMBER[1] | RETURNS |
| AREA | EVENT | NATIVE | SEQUENTIAL |
| ASSIGNABLE | 排他的 | NONASSIGNABLE | SIGNED |
| AUTOMATIC | EXTERNAL | NONCONNECTED | STATIC |
| BACKWARDS | FILE | NONNATIVE | STREAM |
| BASED | 固定 | NONVARYING | STRUCTURE |
| BINARY | FLOAT | NORMAL | TASK |
| ビット | FORMAT | OFFSET | TRANSIENT |
| BUFFERED | GENERIC | OPTIONAL | UCHAR |
| BUILTIN | GRAPHIC | OPTIONS | UNALIGNED |
| BYADDR | HEX | OUTONLY | UNBUFFERED |
| BYVALUE | HEXADEC | OUTPUT | UNION |
| CHARACTER | IEEE | PARAMETER | UNSIGNED |
| COMPLEX | INITIAL | PICTURE | UPDATE |
| CONDITION | INONLY | POINTER | VARIABLE |
| connected | INOUT | POSITION | VARYING |
| CONSTANT | INPUT | PRECISION | VARYING4 |
| CONTROLLED | INTERNAL | PRINT | VARYINGZ |
| DECIMAL | IRREDUCIBLE | RANGE | WIDECHAR |
| DEFINED | KEYED | REAL | WIDEPIC |
| DIMENSION | LABEL | 記録 | |
| DIRECT | LIMITED | REDUCIBLE | |

| 表 41. *DEFAULT* 内の *attribute-expression* に指定する属性 (続き) |
| --- |
| [1] 注: ここでは MEMBER を属性として使用できますが、PL/I DECLARE ステートメントでは属性として使用できません。 |

1 つのブロック内に複数の DEFAULT ステートメントを指定することができます。 DEFAULT ステートメントの有効範囲は、そのステートメントが指定されているブロック、および、そのブロック内で、同じ範囲の別の DEFAULT ステートメントを含まず、なおかつ同じ範囲の DEFAULT ステートメントを持つブロックに包含されていないすべてのブロックです。

内部ブロックの DEFAULT ステートメントは、明示的に宣言された名前に対してのみ効果があります。 これは、暗黙宣言の場合、その名前が使われている外部プロシージャーの PROCEDURE ステートメントのすぐ後にある DECLARE ステートメントでその名前が宣言されているかのように、宣言の有効範囲が決められるからです。

あるブロック内の DEFAULT ステートメントで指定した範囲が、そのブロックに 含まれているブロック内の DEFAULT ステートメントで指定した範囲と部分的に 重なり合ってもかまいません。 このような場合、包含ブロック内の DEFAULT ステートメントの範囲は、被包含 ブロック内の DEFAULT ステートメントの範囲分だけ減少することになります。 次の例を考えてください。

```
 P:  PROCEDURE;
L1:  DEFAULT RANGE (XY) FIXED;
 Q:  BEGIN;
L2:  DEFAULT RANGE (XYZ) FLOAT;
     END P;
```

DEFAULT ステートメント L1 の有効範囲は、プロシージャー P および含まれているブロック Q です。 DEFAULT ステートメント L1 の範囲は、文字 XY で始まるプロシージャー P 内のすべての名前と、文字 XY で始まる開始ブロック Q 内のすべての名前です。ただし、文字 XYZ で始まる名前は除きます。

DEFAULT ステートメントの接頭部としてラベルを指定できます。 このようなラベルへの分岐は、ヌル・ステートメントへの分岐として 扱われます。 条件接頭語は、DEFAULT ステートメントに付加できません。

## 言語に固有のデフォルトの復元

ステートメント `dft range(*) system;` はすべての名前に対して、包含ブロック内で確立されたプログラマー定義のデフォルト規則を指定変更します。 このステートメントを使用して、被包含ブロックの言語に固有のデフォルトを復元することができます。

# 配列

配列 は、同一の属性を持つ要素からなる n 次元の集合です。

配列そのものにだけ名前を付けます。 配列の個々の項目を参照するときは、配列内でのその項目の位置を使用します。 次元属性を提供することによって、名前が配列変数 であることを 指定できます。

REFER で指定しない限り、すべての配列のすべての次元に、少なくとも 1 つの要素が必要です。 配列の境界を REFER で指定する際に、以下の条件を満たしている場合は、要素数がゼロの配列を定義できます。

- 配列はアクセスされることも割り当てられることもない。
- 配列に 1 つの次元しかない (継承した次元を除く)。
- その次元の下限は 1 でなければならない。
- 構造に含まれるすべてのエレメントが、UNALIGNED または NONVARYING BIT でなければならない。

そのため、例えば以下のコードでは、ab3、abc1、および abc2 がアクセスも割り当てもされない場合、n1 がゼロのときに配列 a を割り振ることは有効です。

```
   dcl n1          fixed bin(31);
   dcl p           pointer;
   dcl
     1 a based(p),
       2 ab1       fixed bin(31),
       2 ab2       fixed bin(31),
       2 ab3( n1 refer(ab2) ),
```

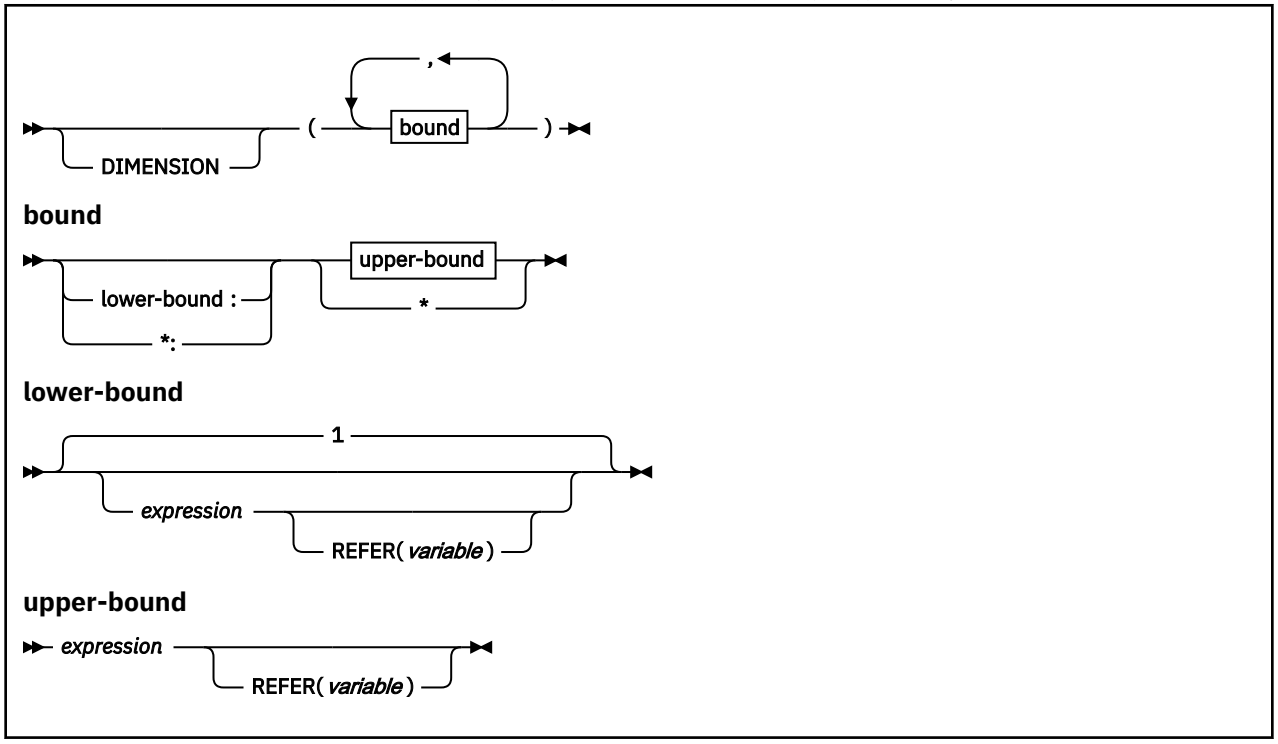
```
          3 abc1    char(40) var,
          3 abc2    char(40) var,
        2 ab4       char(40) var;
```

# 次元属性

DIMENSION 属性は、配列の次元数と、各次元の上限および下限を指定します。

制限なしの式で表される境界 は、配列にストレージが割り振られるときに 評価され、(CMPAT コンパイラー・オプションに対応した精度で) FIXED BINARY に変換されます。

次元のエクステント は、次元の上限と下限を含め、それらの間の整数の数です。




**省略形**: DIM

DIMENSION キーワードが省略される場合には、次元は宣言内の名前 (もしくは括弧で囲まれた名前のリスト) のすぐあとになければなりません。

境界 (bound) 指定の数が、その配列の次元の数を表します。 ただし、宣言される変数が構造体または共用体の配列に 含まれている場合は例外で、 そのときはその変数を含んでいる構造体または共用体から次元の数を 継承します。

境界の指定は、境界を次のようにして示します。

- 上限だけを指定すると、下限にはデフォルトの 1 が設定されます。
- 下限は上限以下でなければなりません。
- アスタリスク (*) は、パラメーターの場合はその記述子から、CONTROLLED 変数の場合はその現在の割り振りから (またはその直前の生成からの割り振り時に)、それ以外の変数の場合はその INITIAL 属性から、上限または下限 (またはその両方) が取得されることを指定します。

CONTROLLED でも PARAMETER でもない変数にアスタリスク (*) を使用して境界を指定できるのは、以下の場合のみです。

- 1 つの次元しかない (かつ、その次元が親から継承されていない) 場合。
- 単一のアスタリスクでのみその境界を定義している場合。
- INITIAL 値の一定のセットを指定する INITIAL 属性が含まれる場合。

したがって、これらの宣言の上限は以下のようになります。

```
3    for dcl a(*)  fixed bin init( 2,3,5 );
7    for    dcl b(*) fixed bin init( 2,3,(5)7 );
17    for    dcl c(*) fixed bin init( 2,3,(2)(5,(3) (7,11) ), 13 );
```

ただし、以下の宣言はすべて無効になります。

```
dcl x(*,*)  fixed bin init(1,2,3,4,5,6);
dcl y( 0 : * )  fixed bin init(1,2,3);
dcl z( * ) fixed bin init( 2, 3, (n) 7 );
```

## DIMACROSS 属性

DIMACROSS 属性は、構造体に DIMENSION 属性を指定しますが、それは構造体からは除去されてそのメンバーに伝搬します。

DIMACROSS 属性は、DIMENSION 属性と同じ構文です。ただし、当然のことながら DIMACROSS キーワードはオプションではありません。

DIMACROSS 属性は構造体にのみ有効であり、直接の子のいずれかに次元属性が既に存在する場合には無効です。

DIMENSION 属性を指定して宣言された変数とは異なり、DIMACROSS 属性を指定して宣言された変数は配列ではありません。 この変数の子は配列です。 ただし、この変数は、BY DIMACROSS 代入では配列として、または LBOUNDACROSS あるいは HBOUNDACROSS 組み込み関数では引数として使用することができます。

例えば、以下の宣言は同等です。

| 宣言 1 | 宣言 2 |
|---|---|
| `Dcl`<br>`  1 a(10) dimacross,`<br>`  2 b,`<br>`  2 c,`<br>`  3 d,`<br>`  3 e;` | `Dcl`<br>`  1 a,`<br>`  2 b(10),`<br>`  2 c(10),`<br>`  3 d,`<br>`  3 e;` |

## 配列の例

以下の例は、配列および配列次元の宣言を理解するために役立ちます。

次の宣言について考えてみましょう。

```
declare List fixed decimal(3) dimension(8);
```

List は、8 つのエレメントからなる 1 次元の配列であり、 それぞれ 3 桁の 10 進固定小数点エレメントであると宣言されています。 List の 1 つの次元は、境界値として 1 と 8 を持ち、 エクステントは 8 です。

次の例を考えてください。

```
declare Table (4,2) fixed dec (3);
```

Table は、8 つの 10 進固定小数点エレメントからなる 2 次元の 配列であると宣言されています。 Table の 2 つの次元は、1 と 4 および 1 と 2 の境界値を持ち、 エクステントは 4 と 2 です。

以下の例について考察します。

```
declare List_A dimension(4:11);
declare List_B (-4:3);
```

最初の例では、境界は 4 と 11 で、2 番目の例では -4 と 3 です。 どちらの場合も下限から上限までに整数が 8 つあるので、両者のエクステントは同じです。

複数の配列を使って配列データを処理する場合は (70 ページの『配列式』を参照)、単にエクステントだけでなく、両者の境界が同じでなければなりません。 前述の List、 List_A、 List_B は、すべてエクステントは同じですが、 境界が異なっています。

## 添え字

添え字 は、配列の次元内の位置を指定する要素式です。 添え字がアスタリスクであれば、次元のすべてのエレメントを 指定する。

配列エレメントの参照方法は、配列の上限と下限によって決まります。 例えば、次のようなデータ項目があるとします。

```
   20 5 10 30 630 150 310 70
```

上記データ項目を、前述で宣言した List という配列に割り当てたとします。 さまざまなエレメントが、次のように参照されます。

| 参照資料 | エレメント |
|---|---|
| LIST (1) | 20 |
| LIST (2) | 5 |
| LIST (3) | 10 |
| LIST (4) | 30 |
| LIST (5) | 630 |
| LIST (6) | 150 |
| LIST (7) | 310 |
| LIST (8) | 70 |

LIST のあとに書かれている括弧内の数字を添え字 といいます。添え字を括弧で囲んで配列名のあとに書くと、その配列内の特定のデータ項目を指すことになります。 LIST(4) のように、添え字付きの名前を参照すると、1 つのエレメントを参照することになります。これは要素変数です。 添え字が付いていない配列名 (例えば、LIST) を書けば、 配列全体を参照することができます。

前述で宣言した List_A と List_B に 同じデータを割り当てることができます。 この場合は、次のように参照します。

| 参照 | エレメント | 参照 |
|---|---|---|
| LIST_A (4) | 20 | LIST_B (-4) |
| LIST_A (5) | 5 | LIST_B (-3) |
| LIST_A (6) | 10 | LIST_B (-2) |
| LIST_A (7) | 30 | LIST_B (-1) |
| LIST_A (8) | 630 | LIST_B (0) |
| LIST_A (9) | 150 | LIST_B (1) |
| LIST_A (10) | 310 | LIST_B (2) |
| LIST_A (11) | 70 | LIST_B (3) |

同じデータを TABLE (2 次元の配列として宣言されている) に割り当てたとします。 TABLE は、次のように 4 行 2 列の行列として表せます。

| TABLE(m,n) | (m,1) | (m,2) |
|---|---|---|
| (1,n) | 20 | 5 |

| TABLE(m,n) | (m,1) | (m,2) |
|---|---|---|
| (2,n) | 10 | 30 |
| (3,n) | 630 | 150 |
| (4,n) | 310 | 70 |

TABLE のエレメントを参照するときは、コンマで区切った 2 つの添え字を括弧で囲んで、配列名のあとに書きます。 例えば、TABLE(2,1) は、2 行目の最初の項目 (この例では 10 というデータ項目) を指します。

TABLE を説明するためのマトリックスの使用は、純粋に概念的なものです。 これは、項目が実際にストレージに編成される方法とは関係ありません。 データ項目は、行を主体とした順序で割り当てられます。 つまり列を表す添え字がまず変化するということを意味します。 例えば、TABLE への割り当て順序は、TABLE(1,1)、TABLE(1,2)、TABLE(2,1)、TABLE(2,2) というようになります。

添え字を使用して配列を参照するときは、添え字の数は配列の次元の数と同じでなければなりません。

有効な算術値が得られる式であれば、添え字として使うことができます。 必要に応じて、値は (CMPAT コンパイラー・オプションに対応した精度で) FIXED BINARY に変換されます。 例えば、TABLE(I,J*K) を使用し、I、J、K の値を変えていけば、TABLE の個々のエレメントを参照できます。

## 配列のクロスセクション

添え字の代わりにアスタリスクを使えば、配列のクロスセクションを参照できます。

アスタリスクは、そのエクステント全体を使用することを意味します。 例えば、TABLE(*,1) は TABLE の第 1 列のすべてのエレメントを指します。 つまり、TABLE(1,1)、TABLE(2,1)、TABLE(3,1)、および TABLE(4,1) からなるクロスセクションを表します。 添え字付きの名前 TABLE(2,*) は TABLE の第 2 行のすべてのデータ項目を指します。 TABLE(*,*) は、TABLE と同様に、配列全体を指します。

添え字としてアスタリスクが付いている名前は、1 つのデータ・エレメントを表すのではなく、アスタリスクと同数の次元を持つ配列を表します。 したがって、そのような名前は要素式ではなく配列式です。

配列のクロスセクションを参照すると、ストレージで隣接していない複数のエレメントを参照することがあります。 そのようなクロスセクションによって表されるストレージを、 非連結 ストレージといいます。 次のような規則があります。 左端の境界をアスタリスクにし、 その右側の境界をアスタリスク以外のものにすると、配列のクロスセクションは非連結ストレージになります。 例えば、A(4,*,*) は連結ストレージになりますが、A(*,2,*) はなりません。

**関連情報**

260 ページの『CONNECTED 属性と NONCONNECTED 属性』
CONNECTED 属性は、パラメーターが連結ストレージのみに対する参照であることを指定します。
NONCONNECTED 属性は、パラメーターが不連続ストレージおよび連続ストレージを占有することを許可します。

# 構造体および共用体

このセクションでは、構造体および共用体について説明します。

## 構造体

構造体 とは、構造体、共用体、要素変数、 および配列であるメンバー・エレメントの集合です。

構造変数 は、データの集合全体を参照するときに 使用される名前です。 ただし配列とは異なり、構造体内の各メンバーにも名前を付けられます。 また、各メンバーの属性を別々にすることもできます。 参照されることがなければ、構造体やメンバーの名前として アスタリスクを使用できます。 例えば、予約項目や埋め込み文字項目を、 アスタリスクで指定することができます。

構造体は、いくつかのレベル を持ちます。レベル 1 の名前は、大構造体 と呼ばれます。それより下のレベルの名前は、小構造体または共用体 になります。最下位のレベルの名前は、基本 名と呼ばれ、 要素変数または配列変数を表すことができます。

構造体は、DECLARE ステートメントで、それぞれの名前の前にレベル番号を付けるという方法によって指定します。 レベル番号は、整数でなければなりません。

大構造体名は、レベル番号 1 を付けて宣言します。 小構造体名、共用体、および基本名は、2 以上のレベル番号を付けて宣言します。 レベル番号とそれに関連する名前は、区切り文字で区切る必要があります。 例えば、給与計算レコードの項目を次のように宣言することができます。

```
declare 1 Payroll,                    /*  major structure name */
          2 Name,                     /*  minor structure name */
            3 Last char(20),          /*  elementary name      */
            3 First char(15),
          2 Hours,
            3 Regular fixed dec(5,2),
            3 Overtime fixed dec(5,2),
          2 Rate,
            3 Regular fixed dec(3,2),
            3 Overtime fixed dec(3,2);
```

この例では、Payroll は大構造体で、それ以外の名前はこの構造体のメンバーです。Name、Hours、および Rate は小構造体で、それ以外のメンバーはすべて基本変数です。 構造体全体は、名前 Payroll を使用すれば参照することができ、 構造体内の特定の部分は、その小構造体名を使用すれば参照できます。 メンバーは、メンバー名の参照によって参照することができます。

上の例では、読みやすくするために字下げをしています。 このステートメントは、次のように連続したストリング内に書くこともできます。

```
Declare 1 Payroll, 2 Name, 3 Last char(20), . . .
```

後続の下位レベルに付けるレベル番号は、連続している必要はありません。 レベル n の小構造体には、その小構造体名と n 以下の レベル番号を持つ次の名前との間にある、n より大きい レベル番号を持つすべての名前が含まれます。

例えば、次の宣言では、先の例の宣言とまったく同じ構造体になります。

```
Declare 1 Payroll,
          4 Name,
            5 Last char(20),
            5 First char(15),
          3 Hours,
            6 Regular fixed dec(5,2),
            5 Overtime fixed dec(5,2),
          2 Rate,
            9 Regular fixed dec(3,2),
            9 Overtime fixed dec(3,2);
```

大構造体の記述は、通常、DECLARE ステートメントの終わりのセミコロンで終了します。 また、コンマで終了する場合もあり、この場合はあとに別の項目の宣言が続きます。

**関連情報**

175 ページの『共用体』
共用体 とは、相互にオーバーレイし合い、 同じストレージを有するメンバー・エレメントの集合です。 メンバーは、構造体、共用体、要素変数、および配列であることができます。 それらは、同一の属性を持っていなくても構わない。

178 ページの『UNION への割り当て』
UNION または UNION を含む構造体への割り当てが可能です。

# 共用体

共用体 とは、相互にオーバーレイし合い、 同じストレージを有するメンバー・エレメントの集合です。 メンバーは、構造体、共用体、要素変数、および配列であることができます。 それらは、同一の属性を持っていなくても構わない。

共用体全体には、データの集合全体で参照できる名前が付けられます。 構造体と同様、共用体の各エレメントもそれぞれ名前が付けられます。 アスタリスクは、共用体の名前、または共用体が参照されない場合

は メンバーの名前として使用できます。 例えば、予約項目や充てん文字項目をアスタリスクで指定することができます。

構造体と同様に、共用体はレベル 1 を含む任意のレベルにすることができます。 次に深いレベルにある共用体のすべてのエレメントは共用体のメンバーであり、同じストレージを占有します。 共用体によって占められるストレージは、 最大メンバーに必要なストレージと等しくなります。 通常、どのような場合でも使用されるメンバーは 1 つだけで、 プログラマーが使用されるメンバーを決定します。

共用体は、構造体と同様、それぞれの名前の前にレベル番号を 付けるという方法によって指定します。

共用体は、一般的に共通部、選択部、および可変部からなる 可変レコードの宣言に使用されます。 例えば、クライアント・ファイルのレコードは、次のように宣言されます。

```
Declare 1 Client,
          2 Number pic '999999',
          2 Type bit(1),
          2 * bit(7),
          2 Name union,
            3 Individual,
              5 Last_Name char(20),
              5 First_Name union,
                7 First   char(15),
                7 Initial char(1),
            3 Company char(35),
          2 * char(0);
```

この例では、Client が大構造体です。 構造体 Individual とエレメント Company は、 共用体 Name のメンバーです。 これらメンバーのうちの 1 つが、Type に応じてアクティブになります。 構造体 Individual には、 共用体 First_name とエレメント Last_name が含まれています。 共用体 First_name は、 First と Initial をメンバーとして持ち、 両方ともアクティブです。 またこの例では、アスタリスクを名前として使用しています。共用体の記述は、DECLARE ステートメントを終了するセミコロンまでで 終了するか、コンマによって終了します (コンマの場合、 別の項目の宣言があとに続きます)。

### UNION 属性

UNION 属性を使用すると、ある変数が共用体であり、そのメンバーが変数のあとに続き、論理的に次に高位のレベルであることを指定することができます。CELL は、UNION の同義語として受け入れられます。

**関連情報**

178 ページの『UNION への割り当て』
UNION または UNION を含む構造体への割り当てが可能です。

## 構造体および共用体の修飾

構造体のメンバーまたは共用体のメンバーは、その名前が固有であれば、 名前だけで参照できます。 別のメンバーが同じ名前を持つ場合には、そのレベルが同じか、異なるのか、 またはあいまいになります。 あいまいになる場合、正しいメンバーを固有に識別するには修飾された参照を 行わなければなりません。

修飾された参照 とは、ピリオドによって接続された 親メンバーにあたる 1 つ以上の名前で修飾されているメンバー名のことです。 (修飾された 参照の構文については、51 ページの『第 3 章 式および参照』を参照)。 ピリオドの前後には、ブランクがあってもかまいません。

構造体の要素への参照があいまいでないと見なされるのは、以下のいずれかの条件が当てはまる場合です。

- 参照が完全に修飾されている場合。つまり、要素への参照にすべての親の名前が含まれている場合です。
- 対象の要素が宣言されているブロック内の他の変数への部分修飾または完全修飾の参照になっていない場合。

修飾は、レベルの順番に従わなければなりません。 つまり、最高位レベルの名前は最初に、最低位レベルの名前は最後になければなりません。

レベル 1 の構造体名または共用体名は、ブロックの有効範囲内で 固有でなければなりませんが、メンバー名は、直接の親のもとで同じ 論理レベルに現れない限り、固有である必要はありません。 修飾名は、それが現れるブロック内で、同じ構造体で固有に 参照される必要がある場合にしか使用することができません。 次の例では、x.y (19) の値は表示されますが、 値 (17) は表示されません。

```
dcl Y fixed init(17);


begin;
  dcl
   1 X,
     2 Y fixed init(19);
  display( Y );
end;
```

参照の実行は、通常、参照を含む一番内側にあるブロック内で宣言された名前に適用されます。

以下の例では、あいまいな参照とあいまいでない参照の両方を説明しています。 以下の例では、 A.C は内側のブロックの C を参照しています。 D.E は外側のブロックの E を参照しています。

```
declare 1 A, 2 C, 2 D, 3 E;
   begin;
   declare 1 A, 2 B, 3 C, 3 E;
A.C = D.E;
```

次の例では、D は 2 回宣言されています。 A.D への参照は、2 番目の D を参照します。これは、A.D が 2 番目の D のみの完全な修飾であるためです。 最初の D は A.C.D として参照されます。

```
declare 1 A,
          2 B,
          2 C,
            3 D,
          2 D;
```

次の例では、A.C の参照はあいまいです。 これは、C がこの参照によって完全に修飾されないためです。

```
declare 1 A,
          2 B,
            3 C,
          2 D,
            3 C;
```

次の例では、A の参照を行うと最初の A が参照され、 A.A の参照を行うと 2 番目の A が参照され、 A.A.A の参照を行うと 3 番目の A が参照されます。

```
declare 1 A,
          2 A,
            3 A;
```

次の例では、X の参照を行うと 最初の DECLARE ステートメントが参照されます。 Y.Z の参照はあいまいなものとなります。 Y.Y.Z は 2 番目の Z を参照し、 Y.X.Z は最初の Z を参照します。

```
declare X;
declare 1 Y,
          2 X,
            3 Z,
            3 A,
          2 Y,
            3 Z,
            3 A;
```

名前の修飾については、150 ページの『宣言の有効範囲』を参照してください。

## UNION への割り当て

UNION または UNION を含む構造体への割り当てが可能です。

以下の条件に合致する場合、UNION を含む構造体は、UNION を含む別の構造体に割り当てることができます。

- ソース構造体およびターゲット構造体に、コンパイル時に認識されたエクステントがある。
- ソース構造体およびターゲット構造体が他の変数上の DEFINED ではない。
- UNION 属性が構造体から除去された場合にその構造体が割り当て可能である (例えば、それらの構造体は同じ構造を持ち、次元が含まれていなければならない)。
- ターゲット内の各 UNION で、ソース内の対応する要素が同じ属性を持つ UNION でなければならない。
- ソースおよびターゲット内の各 UNION の直接の子はすべて、整数バイトを占有しなければならない。 これは子自体が UNION である場合を除く。 この場合、これはその子のすべてのサブ要素に再帰的に該当しなければならない。

UNION を含む構造体は、以下の割り当てではサポートされません。

- 複数割り当て
- 複合割り当て
- BY NAME 割り当て
- BY DIMACROSS 割り当て

UNION の割り当て用に生成されたコードでは、割り当ては UNION 全体の単純バイト・コピーによって実行されます。

## LIKE 属性

LIKE 属性は、宣言される名前が、参照される構造体または共用体 (LIKE 属性のオブジェクト) と論理的に同じ編成であることを指定します。

オブジェクト変数のメンバー名およびその属性 (次元属性を含む) は、 有効にコピーされ、宣言される名前のメンバーになります。 必要であれば、コピーされたメンバーのレベル番号は自動的に調整されます。 オブジェクト変数名およびその属性 (次元属性を含む) は無視されます。

**object-variable**
大構造体、小構造体、または共用体のいずれでもかまいません。 LIKE 属性の指定を含むブロック内になければなりません。 修飾をすることはできますが、添え字を付けることはできません。オブジェクトまたはそのメンバーは、事前に宣言されている場合は LIKE 属性を持つこともできます。

すべての LIKE 属性のオブジェクトは、 その LIKE 属性が拡張される前に宣言された名前に対応しています。

新しいメンバーを、作成された構造または共用体に追加することはできません。 LIKE 属性のオブジェクト変数のすぐあとのレベル番号は、 LIKE 属性を持つ名前のレベル番号以下でなければなりません。

LIKE 属性は、ENTRY 記述内およびパラメーター宣言内でサポートされます。 ENTRY 記述内で使用される場合、メンバー名はコピーされません。 例えば、次の宣言は有効です。

```
dcl
    1 name,
    2 first   char(20) var,
    2 middle  char(10) var,
    2 last    char(30) var;

dcl func entry( like name  );
```

エントリー "func" の宣言は、この長い宣言と同じになります。

```
dcl func entry( 1, 2 char(20) var, 2 char(10) var, 2 char(30) var );
```

次の宣言では、X に対し同じ構造体が生成されます。

```
  dcl
    1 A(10) aligned static,
      2 B    bit(4),
      2 C    bit(4),
    1 X like A;


  dcl
    1 X,
      2 B bit(4),
      2 C bit(4);
```

次元 (DIM(10))、ALIGNED、および STATIC 属性は、 LIKE 拡張の一部としてコピーされないことに注意してください。

LIKE 属性は、デフォルトが適用される前、もしくは ALIGNED 属性 および UNALIGNED 属性が LIKE オブジェクト変数に含まれるエレメントに 適用される前に拡張されます。 ただし、LIKE 属性の拡張は、すべての LIKE 属性が解決された後にのみ発生します。

## 例

以下の宣言について考察します。

```
  declare 1 A,
            2 C,
              3 E(3) union,
                5 E1,
                5 E2,
              3 F;
  declare 1 B(10) union,
            2 C, 3 G, 3 H,
            2 D;
  begin;
  declare 1 C like B;
  declare 1 D(2),
            5 BB like A.C;
  end;
```

宣言 C および D の結果を、次の例に示します。

```
  dcl
      1 C,               /* DIM and UNION not copied. */
        2 C, 3 G, 3 H,
        2 D;


  dcl 1 D(2),
        5 BB,
          6 E(3) union,  /* DIM(3) and UNION copied.      */
            7 E1,        /* Note adjusted level-numbers.  */
            7 E2,
          6 F;
```

次の宣言は有効ですが、これは、B が C の前に宣言され、E が F の前に宣言されているためです。

```
 dcl 1 a, 2 a1 fixed bin;
 dcl 1 b, 2 b1 like a;
 dcl 1 c, 2 c1 like b;

 dcl 1 d, 2 d1 fixed bin;
 dcl 1 e  like d;
```

次の例は有効ですが、これは、すべての LIKE 参照の解決後に LIKE 参照が拡張されているためです。LIKE 参照が解決前に拡張されると、参照 aa3_array はあいまいになります。

```
  dcl 1 aa(30)
     ,5 aa1               char( 5)
     ,5 aa2               fixed bin(31)
     ,5 aa3_array(30)
       ,7 aa3_1           fixed dec(15,2)
       ,7 aa3_2           fixed dec(15,2)
       ,7 aa3_3           fixed dec(11,4)
       ,7 aa3_4           fixed dec(7,3)
    ;

  dcl bb                  like aa;
  dcl cc                  like
aa3_array;
```

以下の例は、C.E に LIKE 属性があるため、また、B が A の後に宣言されているため、無効です。 A および B の宣言の順序が逆になっている場合、コードは有効です。

```
  declare 1 A like C,
          1 B,
            2 C,
              3 D,
              3 E like X,
            2 F,
          1 X,
            2 Y,
            2 Z;
```

G.C を解決できないため、次の例は無効です。 G.C は解決されません。これは、A の LIKE 属性の解決を試みた後で、G の LIKE の拡張が行われているためです。

```
  declare 1 B,
            2 C,
              3 D,
              3 E,
            2 F,
          1 G like B;
          1 A like G.C,
```

## INDFOR 属性

INDFOR 属性は、宣言される名前が、参照される構造体または共用体 (INDFOR 属性のオブジェクト) と論理的に同じ編成であることを指定します。 INDFOR 属性は LIKE 属性に似ています。

オブジェクト変数のメンバー名とその属性 (次元属性を除く) は、実際には FIXED BIN(15) に変換され、宣言された名前のメンバーになります。 これは、属性をコピーする LIKE 属性とは異なります。

**object-variable**
大構造体、小構造体、または共用体のいずれでもかまいません。 これは、INDFOR 属性のあるブロック内で認識されていなければなりません。 修飾をすることはできますが、添え字を付けることはできません。 オブジェクトまたはそのメンバーが事前に宣言されている場合は、これらのオブジェクトまたはメンバーも INDFOR 属性を持つことがあります。

すべての INDFOR 属性のオブジェクトは、 その INDFOR 属性が拡張される前に宣言された名前に対応しています。

INDFOR 属性は、デフォルトが適用される前、もしくは ALIGNED 属性 および UNALIGNED 属性が INDFOR オブジェクト変数に含まれるエレメントに 適用される前に拡張されます。 ただし、INDFOR 属性の拡張は、すべての INDFOR 属性が解決された後にのみ発生します。

**例**

この例は、INDFOR 属性と LIKE 属性の相違を示しています。

この例は以下の宣言に基づいています。

```
dcl 1 a, 2 b char(8), 2 c fixed dec(5,0);
```

宣言がどのように展開されるかに注目してください。

| 指定する宣言ステートメント | 展開結果 |
|---|---|
| dcl 1 alike like a; | dcl 1 alike, 2 b char(8), 2 c fixed dec(5,0); |
| dcl 1 aindfor indfor a; | dcl 1 a, 2 b fixed bin(15), 2 c fixed bin(15); |

**関連情報**

178 ページの『LIKE 属性』
LIKE 属性は、宣言される名前が、参照される構造体または共用体 (LIKE 属性のオブジェクト) と論理的に同じ編成であることを指定します。

## NOINIT 属性

NOINIT 属性は、どの INITIAL 属性も無視されることを指定します。

NOINIT 属性はレベル 1 構造体で最も役立つと思われますが、任意の副構造でも指定できます。

NOINIT 属性は特に LIKE 属性に便利です。これは、新しい変数が古い変数の LIKE (類似) として宣言されたが NOINIT 属性も持つ場合、新しい変数は、古い変数からすべてのサブ構造を継承しますが、その INITIAL 値は継承しないからです。

## NULLINIT 属性

NULLINIT 属性を指定すると、INITIAL 属性を持たない変数にそれぞれのデータ属性に応じて INITIAL 属性が設定されます。

- INIT( (*) 0 ) - データ属性が FIXED または FLOAT の場合
- INIT( (*) '') - データ属性が PICTURE、CHAR、BIT、GRAPHIC、UCHAR、または WIDECHAR の場合
- INIT( (*) SYSNULL() ) - データ属性が POINTER の場合。ただし、DFT(NULLSTRPTR(NULL)) オプションが有効の場合を除く。これが友好の場合、INITIAL 属性は INIT( (*) NULL() ) になる
- INIT( (*) SYSNULL() ) - データ属性が OFFSET の場合
- INIT( (*) NULLENTRY() ) - データ属性が ENTRY の場合

上記のリストに挙げられていない属性の変数は、NULLINIT 属性によって変更されません。

NULLINIT 属性は、レベル 1 の名前でのみ指定できます。NOINIT 属性や INITIAL 属性と一緒に指定することはできません。 level-1 名には DEFINE STRUCTURE タイプを指定できます。

## XML 関連属性

XML 関連属性は、XML 属性の使用および XMLCHAR または XMLUCHAR 組み込み関数によって生成される制御出力を指定します。

**注 :** コンパイラーは、これらの属性を使用するフィールドを含む構造体が XMLCHAR または XMLUCHAR に渡されない限り、これらの属性を無視します。

### XMLATTR 属性

XMLATTR 属性は、XMLCHAR または XMLUCHAR によって生成される XML 出力で、フィールドがそれが持つ構造の属性として提示されることを示します。

XMLATTR は、以下のいずれかのエレメントと共に指定した場合は無効です。

- 配列。 ただし、構造体が配列の場合、XMLATTR は構造体のメンバーに対して許可されます。
- 構造体または共用体
- 名前が付いていない構造体エレメント
- 先行して、同じ親で XMLATTR 属性を指定せずに使用されている構造体エレメント

## XMLCONTENT 属性

XMLCONTENT 属性は、XMLCHAR または XMLUCHAR 組み込み関数によって生成されるテキストに変数が含まれている場合に、その変数がタグなしテキストとして表示されることを指定します。

## XMLIGNORE 属性

XMLIGNORE 属性は、XMLCHAR または XMLUCHAR がエレメントを無視する必要があることを示します。この属性が副構造体で指定されている場合、構造体のすべてのエレメントが無視されます。

## XMLNAME 属性

XMLNAME 属性は、XMLCHAR または XMLUCHAR 関数の変数に使用される名前を指定する機能を提供します。 特に、この属性を使用して、PL/I の命名規則に準拠しない名前を指定することができます。

これは、JSONNAME が JSONPUT 組み込み関数を提供するのと同じ機能を XMLCHAR および XMLUCHAR 組み込み関数に提供します。

## XMLOMIT 属性

XMLOMIT 属性は、フィールドがヌル・ストリング (' ') であるか 0 に等しい数値である場合に、XMLCHAR または XMLUCHAR によって生成される XML 出力から特定のデータ項目のフィールドを省略する必要があることを示します。

XMLOMIT 属性は、構造体に対して許可されます。この場合、構造体内のすべてのフィールドがヌルであれば、構造体とそのすべてのメンバーは省略されます。

XMLOMIT は、以下のいずれかのエレメントと共に指定した場合は無効です。

- 共用体
- 名前が付いていない構造体エレメント
- ネイティブではない浮動小数点を使用するエレメント (Windows 上の hex または dfp)

### *XMLATTR* および *XMLOMIT* の使用例

この例は、XMLATTR および XMLOMIT 属性を持つ構造体の宣言と、XMLCHAR または XMLUCHAR をその構造体で使用することによって得られる出力を示しています。

```
dcl
   1 order
     2 orderNr           char(20) init('1729'),
     2 customer,
       3 id              xmlattr fixed bin(31) init('2917'),
       3 name            xmlattr char(32) init('jakob'),
       3 firstname       xmlattr char(24) init('michael'),
       3 partno                  fixed bin(31) init(1367),
       3 special         xmlomit char(35) init('');
```

出力:

```
<order>
  <orderNr>1729</orderNr>
    <customer id='2917' name='jakob' firstname='michael'>
    <partno>1367</partno>
  </customer>
</order>
```

## JSON 関連属性

JSON 関連属性は、JSON 関数での変数属性の使用を指定し、JSON 関数によって生成される出力を制御します。

### JSONNAME 属性

JSONNAME 属性は、jsonPut 関数または jsonGet 関数で変数に使用または予期される名前を指定します。特に、JSONNAME 属性を使用して、PL/I の命名規則に準拠しない名前を指定することができます。

JSONNAME 属性は、CHARACTER タイプを持つスカラー制限式を指定する必要があります。

例えば、ハイフンのある名前を指定したい場合は、次の属性を変数に使用できます。

```
JSONNAME('comment-text')
```

### JSONOMIT 属性

JSONOMIT 属性は、特定データ項目のフィールドがヌル・ストリング ('') に等しいストリングまたは 0 に等しい数値である場合に、JSON 関数によって生成される出力からそのフィールドを除去する必要があることを指示します。

JSONOMIT は、以下のいずれかのエレメントと共に指定した場合は無効です。

- 構造体または共用体
- 名前が付いていない構造体エレメント
- ネイティブではない浮動小数点を使用するエレメント (Windows 上の hex または dfp)

### JSONTRIMR 属性

JSONTRIMR 属性は、指定されたフィールド (あるいはフィールドが構造体の場合は構造体のメンバー) の末尾ブランクを JSON 関数が切り取ることを示します。

# 集合体の組み合わせおよびマッピング

集合体とは、他のデータ項目の集合であるデータ項目です。 このセクションでは、構造体の配列、共用体の配列、構造体または共用体への参照、および構造体と共用体間のマッピングについて説明します。

## 配列、構造体、および共用体の組み合わせ

構造体または共用体に次元属性を指定すると、 それぞれ構造体の配列、 または共用体の配列 になります。このような配列のエレメントは、同一の名前、レベル、 およびメンバーを持つ構造体または共用体です。

例えば、ある構造体が 20 世紀から 21 世紀までの毎月の気象データを 保管するために使用されている場合、これは次のように宣言されます。

```
Declare 1 Year(1901:2100),
          3 Month(12),
            5 Temperature,
              7 High decimal fixed(4,1),
              7 Low decimal fixed(4,1),
            5 Wind_velocity,
              7 High decimal fixed(3),
              7 Low decimal fixed(3),
           5 Precipitation,
             7 Total decimal fixed(3,1),
             7 Average decimal fixed(3,1),
         3 * char(0);
```

1991 年 7 月の気象データは、Year(1991,7) によって参照できます。 このデータの各部分は、Temperature(1991,7) および Wind_Velocity(1991,7) によって参照できます。Precipitation.Total(1991,7) および Total(1991,7) はいずれも、1991 年 7 月の総雨量を指します。

Temperature.High(1991,3) は、添え字付きの修飾された参照であり、1991 年 3 月の最高気温を指します。

構造体または共用体の配列の中に、配列であるメンバーが含まれていると、 添え字付きの修飾された参照の必要性が明確になります。 次の例では、A および B は共に構造体です。

```
declare
  1 A (2,2),
    2 B (2),
      3 C fixed bin,
      3 D fixed bin,
    2 E fixed bin;
```

データ項目を参照するために、3 つの名前と 3 つの添え字が必要になることがあります。 次の例を参照してください。

A(1,1).B は、構造体の配列である B を参照します。
A(1,1) は構造体を参照します。
A(1,1).B(1) は構造体を参照します。
A(1,1).B(2).C はエレメントを参照します。

このような参照における添え字は、添え字の順序さえ変えなければ、 高位レベルの名前に付けても低位レベルの名前に付けてもかまいません。 上の例では、A.B.C(1,1,2) と A(1,1,2).B.C は、上記の構造体の配列については A(1,1).B(2).C と同じ意味になります。 すべての添え字を最下位レベルに移したのでなければ、 その参照には介在添え字 があるといいます。 そこで、例えば A.B(1,1,2).C には介在添え字があります。

構造体または共用体の配列の中で宣言されている項目は、親のもとで宣言されている次元を継承します。 構造体 A の配列の前の宣言では、配列 B は、A に対して宣言された 2 つの次元を継承するため、3 次元構造体です。 B が固有で、修飾を必要としない場合、特定の B を参照するには 3 つの添え字が必要です。2 つは特定の A を識別するためのもので、もう 1 つはその A 内の特定の B を識別するためのものです。

## 構造体または共用体の配列のクロスセクション

構造体または共用体の配列のクロスセクションを参照することはできません。 つまり、すべての添え字がアスタリスクでない限り、 参照するときにアスタリスク表記を使用することはできません。

## 構造体および共用体の操作

構造体は、すべての要素変数を参照可能なほとんどのコンテキストで参照できます。 ただし、共用体または共用体を含む構造体に対する参照には制限があります。

例えば、割り当て、入出力ステートメント、その他で構造体参照を行うことができます。

ただし、共用体または共用体を含む構造体に対する参照は以下のコンテキストに限られます。

- パラメーターおよび引数
- 組み込み関数とサブルーチンのうちストレージ制御や構造体を使用することができるもの

## 構造体および共用体のマッピング

共用体の各メンバーは、構造体のメンバーと同じ方法でマップされます。

それぞれのメンバー (共用体でない場合) は、構造体のメンバーの場合と同様にマップされます。 これによって、共用体の各メンバーの最初の記憶場所は、 各メンバーがそれぞれ異なる位置合わせを必要とし、 さらに (それによって) メンバーの開始位置より 前にそれぞれ異なる埋め込みを必要とする場合は、互いにオーバーレイすることはありません。

次の共用体について考えてみましょう。

```
dcl
  1 A union,
    2 B,
```

```
        3 C char(1),
        3 D fixed bin(31),
      2 E,
        3 F char(2),
        3 G fixed bin(31),
         2 H char(8);
```

3 バイトの埋め込みが、A および B の間に追加されます。 A と E の間には 2 バイト追加されます。 A と H の間には埋め込みバイトはありません。 これはつまり、C は H の 4 番目のバイトで開始し、F は H の 3 番目のバイトで開始するということです。

前述の例のように、どの UNION に対しても、ADDR、BITLOCATION、または LOCATION 組み込み関数を使用することはできません。 これらの関数は、共用体のメンバーの先頭の記憶場所が同じである場合にのみ使用してください。

共用体の各メンバーの先頭の記憶位置を同じにするには、各先頭メンバーがそれぞれ同じ位置合わせ要件を持ち、さらにメンバー (またはそのメンバーのメンバー) 内での最高位の位置合わせと同じになるようにしてください。

以降の説明は、小構造体または要素変数のいずれかである構造体または共用体の メンバーに適用されます。

どの構造 (大構造体または小構造体) でも、その長さ、位置合わせの要件、 および 8 バイト境界からの相対的な位置は、構造体のメンバーの長さ、 位置合わせの要件、および相対位置によって決まります。 各レベルおよび構造体全体でこれらの要件を決定するプロセスを、 構造体のマッピング といいます。

レコード単位の入出力を使用するとき、ある構造体にどのくらいの長さのレコードが必要であるかを判別します。 また、位置指定モードの入出力では、構造体の位置合わせを正しく行うために必要な埋め込みまたは再配置の量を 判別します。このような場合、構造体のマッピングを通して判別することができます。

構造体のマッピングのプロセスは、構造体のメンバー相互間に生じる使用されない ストレージ (埋め込み) の量を最小にします。 これは、以下に説明する規則に (事実上) 従ってこの処理全体を完了させてから、構造体にストレージを割り振ります。

構造体のマッピングは、物理的な処理ではありません。 「シフトする」や「相対位置変更する」などの用語は、 単に説明の便宜で使用するもので、実際にストレージ内で移動が行われるわけではありません。 構造体にストレージが割り振られるとき、マッピング処理の結果として相対位置が すでに知られています。

構造体全体のマッピングでは、項目 (エレメントまたはすでに個々のマッピングが判別されている小構造体) が順に対として組み合わされていきます。 1 つの対が作られると、それは 1 つの単位となり、別の単位と組み合わされます。構造体全体のマッピングが終わるまで、この操作が続けられます。 したがって、このプロセスの規則は、次の 2 種類に分類されます。

- 対を作る順序を決めるための規則
- 1 つの対のマッピングを行うための規則

以下にこれらの規則を説明し、これらの規則の適用例を示して詳しく説明します。 構造体エレメントの論理レベル とレベル番号 の違いを理解する必要があります。構造体の宣言が一定のレベル番号で行われたり適切な字下げをして 行われていれば (規則の説明のあとに詳しい例示があります)、 論理レベルについてはすぐに明らかになります。 いずれにしても、構造体宣言の始まりから順番に各項目に下記の規則を 適用していけば、その構造体の各項目の論理レベルを判別することができます。

**注 :** ある項目の論理レベルは、その項目を直接に含んでいる構造体の 論理レベルよりも、常に 1 単位だけ低くなるという規則です。

次の例では、下の行の数字は、宣言内の各項目の論理レベルを示しています。

```
dcl 1 A, 4 B, 5 C, 5 D, 3 E, 8 F, 7 G;

    1    2    3    3    2    3    3
```

## 対を作る順序の規則

対を作る順序を決める際の手順は、次のとおりです。

1. 最も低い論理レベル (論理レベル n と呼ぶことにする) にある小構造体を見つけます。
2. 論理レベル n の小構造体が複数あるときは、宣言内で最初に書いたものをとります。

3. この小構造体内の最初の 2 つのエレメントを対にして、1 つの単位を形成します。 このとき、1 つの対をマッピングするための規則を使用します (186 ページの『1 つの対のマッピングを行うための規則』を参照してください。)
4. この単位と、この小構造体で宣言されている次のエレメント (ある場合) とを対にして、より大きい単位を形成します。
5. この小構造体の全エレメントが 1 つの単位に組み合わされるまで、ステップ 4 を繰り返します。 これで、この小構造体のマッピングは完了です。 あらゆる埋め込みを含め、その位置合わせの要求と長さが決定され、構造体宣言を変更しなければ変更されません。 ダブルワード境界からのオフセットも、ここで決められます。 このオフセットは、含まれている構造体をマッピングするときに有効です。 また、このようなマッピングの結果として変わることがあります。
6. 宣言内の論理レベル n にある次の小構造体 (ある場合) について、 ステップ 3 からステップ 5 を繰り返します。
7. 論理レベル n のすべての小構造体のマッピングが終わるまで、 ステップ 6 を繰り返します。 これらの小構造体のおのおのは、構造体のマッピングを行う上で 1 つのエレメントと考えることができます。
8. 次の高い論理レベルで小構造体のためにペア化処理を繰り返します。 すなわち n を (n-1) に等しくし、ステップ 2 から 7 まで繰り返します。
9. n = 1 までステップ 8 を繰り返します。 その後ステップ 3 から 5 を大構造体について繰り返します。

## 1 つの対のマッピングを行うための規則

ここでは説明の便宜上、ストレージをいくつかの連続したダブルワードと考えます。 各ダブルワードは 8 バイトで、0 から 7 までの番号が付いているものとし、 この番号でダブルワード境界からのオフセットを表します。 さらに、構造体の始まりからの長さやオフセットを算出できるように、 各バイトには、任意のバイトから始めて 0 から順に連続した番号が 付いているものとします。

1. ダブルワード境界の対の最初のエレメントを開始するか、またはエレメントがすでにマップされている小構造体の場合は、示された量によってダブルワード境界からその分を 相対位置変更します。
2. この 1 対の 2 番目のエレメントを、最初のエレメントの終わりの後に続く最初の有効な位置から始まるように合わせます。 この位置は、2 番目のエレメントの位置合わせ要件に左右されます。 (2 番目のエレメントが小構造体である場合、 その位置合わせ要件はすでに決定されています。)
3. 最初のエレメントの位置合わせ要件が満たされる限り、最初のエレメントを 2 番目のエレメントの方向にシフトします。 どれだけシフトしたかによって、この 1 対がダブルワード境界からどれだけ離れているか (オフセット) が決まります。

この作業が完了すると、 この 2 つのエレメントの間の埋め込みは最少量になっており、後続の作業中に変わることはありません。 対は固定長の単位および位置合わせ要件となります。その長さは 2 つの長さの合計と埋め込みであり、その位置合わせ要件は 2 つの位置合わせ要件の高い方 (それらが異なる場合) です。

## UNALIGNED 属性による影響

187 ページの『構造体のマッピング例』にある例は、ALIGNED と宣言された構造体に適用される規則を示しています。 ALIGNED と宣言された構造体のマッピングは、多くの位置合わせ要件があるため、より複雑です。 UNALIGNED 属性を指定すると、ハーフワード、フルワード、 およびダブルワードの位置合わせ要件が 1 バイトに引き下げられ、 ビット・ストリングの位置合わせ要件が 1 ビットに引き下げられます。 構造体のマッピングについては同じ規則が適用されますが、引き下げられた位置合わせ要件が使われます。 使われないストレージが生じるのは、 構造体にビット・ストリングが含まれていて、 バイト内でビットの埋め込みが行われたときだけです。

AREA データは、位置合わせなしにすることはできません。

構造体に UNALIGNED 属性があり、位置合わせできないエレメントがある場合は、 UNALIGNED はそのエレメントについては無視されます。 エレメントが位置合わせされ、エラー・メッセージが出されます。 例えば、プログラムで次のように宣言した場合、 C には ALIGNED 属性が与えられます (継承される 属性 UNALIGNED は AREA と矛盾するからです)。

```
declare 1 A unaligned,
          2 B,
          2 C area(100);
```

## 構造体のマッピング例

次の例では、例のように宣言された構造体に、 構造体のマッピングの規則がどのように適用されるかを示します。

```
declare 1 A aligned,
          2 B fixed bin(31),
          2 C,
            3 D float decimal(14),
            3 E,
              4 F entry,
              4 G,
                5 H character(2),
                5 I float decimal(13),
              4 J fixed binary(31,0),
            3 K character(2),
            3 L fixed binary(20,0),
          2 M,
            3 N,
              4 P fixed binary(15),
              4 Q character(5),
              4 R float decimal(2),
            3 S,
              4 T float decimal(15),
              4 U bit(3),
              4 V char(1),
            3 W fixed bin(31),
          2 X picture '$9V99';
```

最も低い論理レベルにある小構造体は G です。 したがって、G が最初にマッピングされます。 次に、E がマッピングされ、N、S、 C、M がこの順序でマッピングされます。

それぞれの小構造体ごとに、プロセスのステップを表 188 ページの図 4 に示し、そのプロセスを視覚的に捕えた図を 188 ページの図 5 に示します。 最後に、191 ページの図 11 に示すように、大構造体 A がマッピングされます。 この例の終わりには、各メンバーが A の始まりからどのくらい離れているか (オフセット) を示す表 (192 ページの図 12) の形で、A の構造体マップを示しています。

| | Name of Element | Alignment Requirement | Length | Offset from Doubleword Begin | Offset from Doubleword End | Length of Padding | Offset from Minor Structure |
|---|---|---|---|---|---|---|---|
| Step 1 | H | Byte | 2 | 0 | 1 | | |
| | I | Doubleword | 8 | 0 | 7 | | |
| Step 2 | *H | Byte | 2 | 6 | 7 | | 0 |
| | I | Doubleword | 8 | 0 | 7 | 0 | 2 |
| Minor Structure | G | Doubleword | 10 | 6 | 7 | | |
| Step 1 | F | Fullword | 8 | 0 | 7 | | |
| | G | Doubleword | 10 | 6 | 7 | | |
| Step 2 | *F | Fullword | 8 | 4 | 3 | | 0 |
| | G | Doubleword | 10 | 6 | 7 | 2 | 10 |
| Step 3 | F & G | Doubleword | 20 | 4 | 7 | | |
| | J | Fullword | 4 | 0 | 3 | 0 | 20 |
| Minor Structure | E | Doubleword | 24 | 4 | 3 | | |
| Step 1 | P | Halfword | 2 | 0 | 1 | | 0 |
| | Q | Byte | 5 | 2 | 6 | | 2 |
| Step 2 | P & Q | Halfword | 7 | 0 | 6 | | |
| | R | Fullword | 4 | 0 | 3 | 1 | 8 |
| Minor Structure | N | Fullword | 12 | 0 | 3 | | |
| Step 1 | T | Doubleword | 8 | 0 | 7 | | 0 |
| | U | Byte | 1 | 0 | 0 | 0 | 8 |
| Step 2 | T & U | Doubleword | 9 | 0 | 0 | | |
| | V | Byte | 1 | 1 | 1 | 0 | 9 |
| Minor Structure | S | Doubleword | 10 | 0 | 1 | | |
| Step 1 | D | Doubleword | 8 | 0 | 7 | | 0 |
| | E | Doubleword | 24 | 4 | 3 | 4 | 12 |
| Step 2 | D . E | Doubleword | 36 | 0 | 3 | | |
| | K | Byte | 2 | 4 | 5 | 0 | 36 |
| Step 3 | D, E, & K | Doubleword | 38 | 0 | 5 | | |
| | L | Fullword | 4 | 0 | 3 | 2 | 40 |
| Minor Structure | C | Doubleword | 44 | 0 | 3 | | |
| Step 1 | N | Fullword | 12 | 0 | 3 | | |
| | S | Doubleword | 10 | 0 | 1 | | |
| Step 2 | *N | Fullword | 12 | 4 | 7 | | 0 |
| | S | Doubleword | 10 | 0 | 1 | 0 | 12 |
| Step 3 | N & S | Doubleword | 22 | 4 | 1 | | |
| | W | Fullword | 4 | 4 | 7 | 2 | 24 |
| Minor Structure | M | Doubleword | 28 | 4 | 7 | | |

*First item shifted right

図 *4.* 構造体のマッピング例

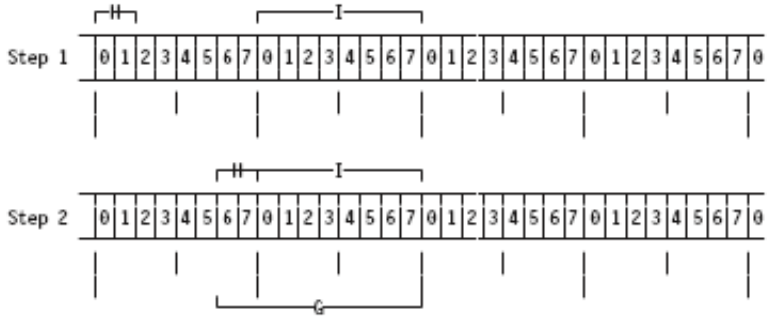



図 *5.* 小構造体 *G* のマッピング

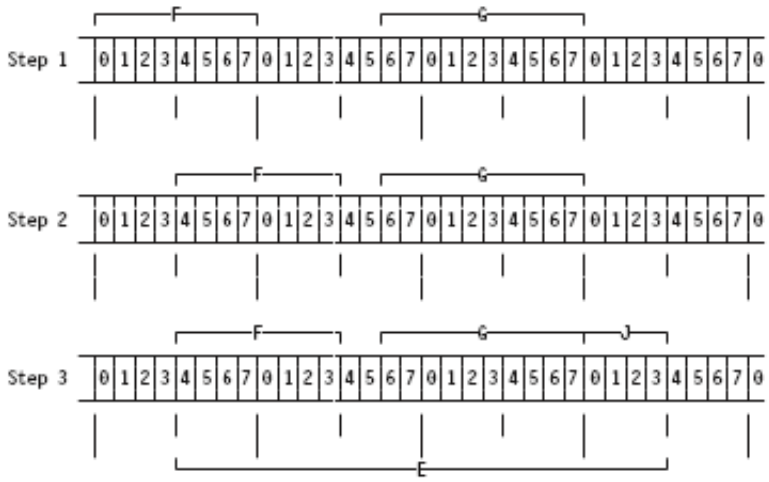



図 *6.* 小構造体 *E* のマッピング

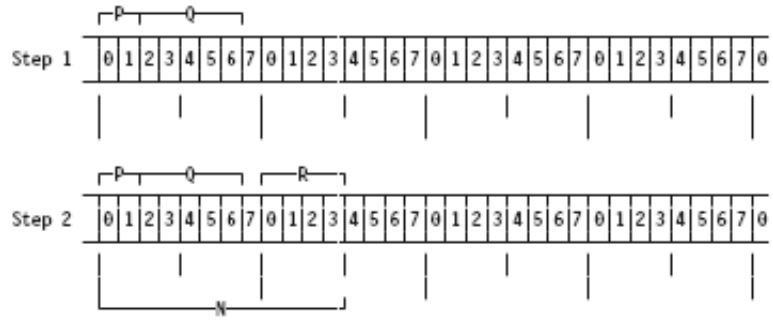



図 *7.* 小構造体 *N* のマッピング

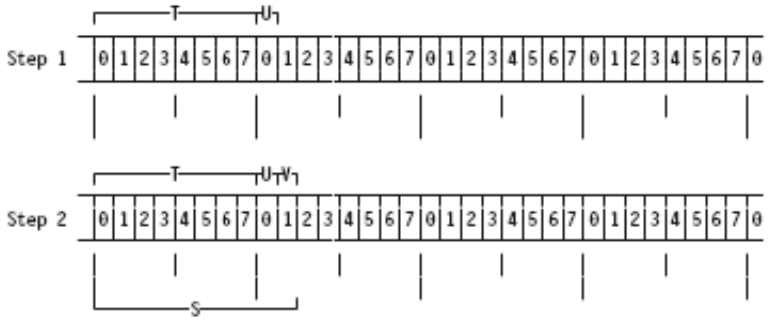



図 *8.* 小構造体 *S* のマッピング

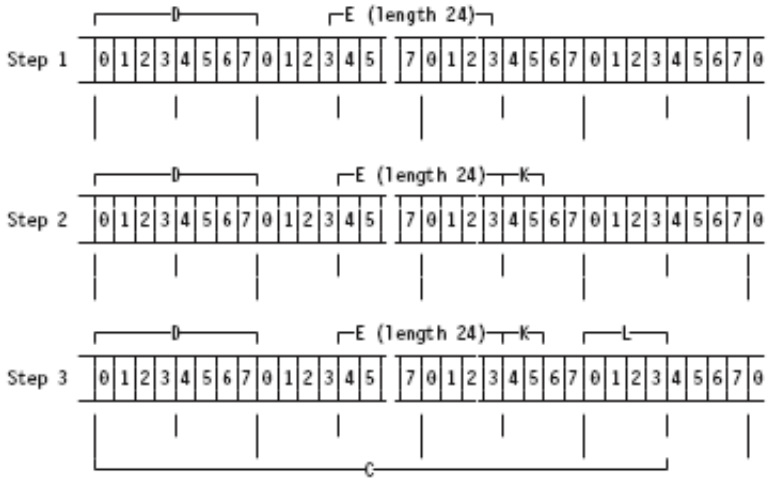



図 *9.* 小構造体 *C* のマッピング

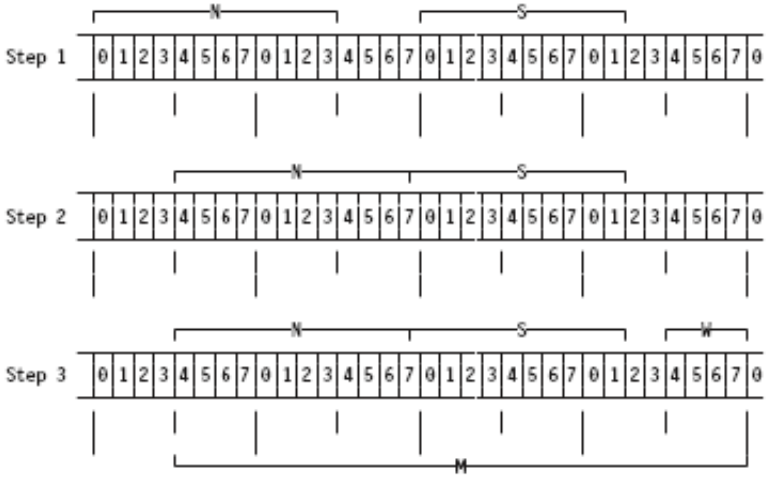



図 *10.* 小構造体 *M* のマッピング

| | Name of Item | Alignment Required | Length | Offset from Doubleword Begin | Offset from Doubleword End | Length of Padding | Offset from A |
|---|---|---|---|---|---|---|---|
| Step 1 | B | Fullword | 4 | 0 | 3 | | |
| | C | Doubleword | 44 | 0 | 3 | | |
| Step 2 | B* | Fullword | 4 | 4 | 7 | | 0 |
| | C | Doubleword | 44 | 0 | 3 | 0 | 4 |
| Step 3 | B & C | Doubleword | 48 | 4 | 3 | | |
| | M | Doubleword | 28 | 4 | 7 | 0 | 48 |
| Step 4 | B, C, & M | Doubleword | 76 | 4 | 7 | | |
| | X | Byte | 4 | 0 | 3 | 0 | 76 |
| | A | Doubleword | 80 | 4 | 3 | | |

* First item shifted right

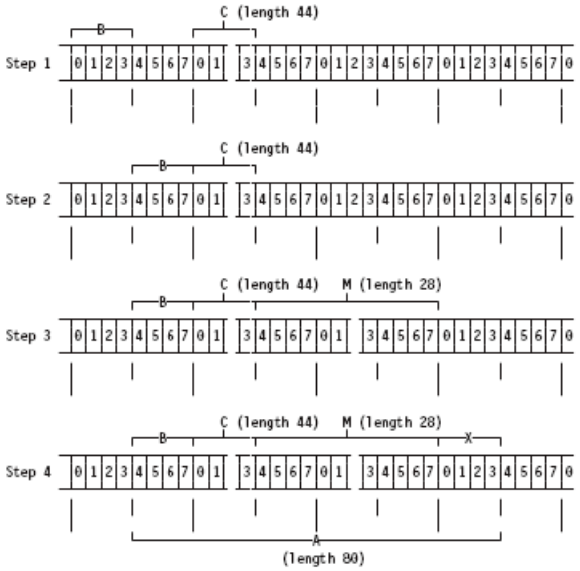



図 *11.* 大構造体 *A* のマッピング

| | | | | |
|---|---|---|---|---|
| A | | | | From A |
| B | | | | 0 |
| C | | | From C | 4 |
| D | | | 0 | 4 |
| padding (4) | | | 8 | 12 |
| E | | From E | 12 | 16 |
| F | | 0 | 12 | 16 |
| padding (2) | | 8 | 20 | 24 |
| G | From G | 10 | 22 | 26 |
| H | 0 | 10 | 22 | 26 |
| I | 2 | 12 | 24 | 28 |
| J | | 20 | 32 | 36 |
| K | | | 36 | 40 |
| padding (2) | | | 38 | 42 |
| L | | | 40 | 44 |
| M | | | From M | 48 |
| N | | From N | 0 | 48 |
| P | | 0 | 0 | 48 |
| Q | | 2 | 2 | 50 |
| padding (1) | | 7 | 7 | 55 |
| R | | 8 | 8 | 56 |
| S | | From S | 12 | 60 |
| T | | 0 | 12 | 60 |
| U | | 8 | 20 | 68 |
| V | | 9 | 21 | 69 |
| padding (2) | | | 22 | 70 |
| W | | | 24 | 72 |
| X | | | | 76 |

図 *12.* 構造 *A* の最終的なマッピングにおけるオフセット

# 第 8 章 ステートメントとディレクティブ

この章には、すべての PL/I ステートメントと % ディレクティブがリストされています。

% ステートメントおよびマクロ・ステートメントについては、579 ページの『第 20 章 プリプロセッサーの機能』に説明があります。

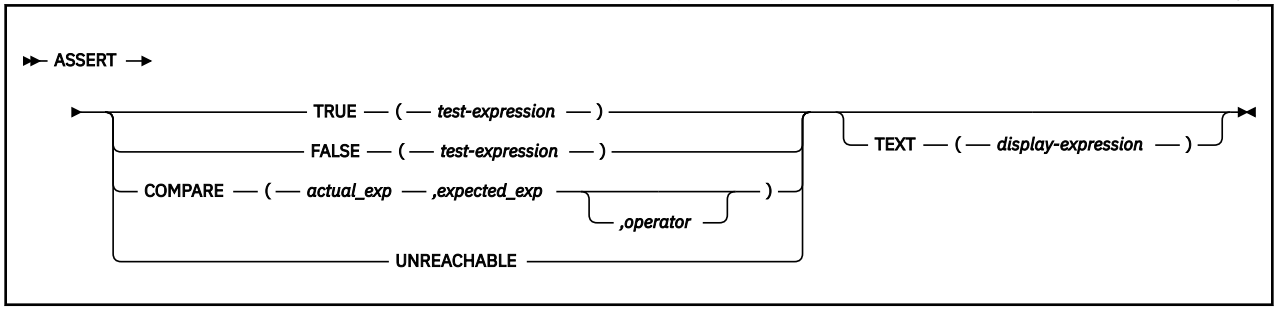
## ALLOCATE ステートメント

ALLOCATE ステートメントは変数用のストレージを割り振ります。

ALLOCATE ステートメントの詳細については、235 ページの『第 9 章 ストレージ制御』の 238 ページの『被制御変数のための ALLOCATE ステートメント』および 248 ページの『基底付き変数のための ALLOCATE ステートメント』を参照してください。

## ASSERT ステートメント

ASSERT ステートメントは、条件が真であるか偽であるかを表明したり、2 つの値を比較してそれらが等しいかどうかを判別したり、あるいはステートメントを実行する必要があるかどうかを表明したりします。




**TRUE(*test-expression*)**
*test-expression* 内の 1 つ以上のビットの値が '1'B である場合、*test-expression* が真であることを表明します。

**FALSE(*test-expression*)**
*test-expression* 内のすべてのビットの値が '0'B である場合、*test-expression* が偽であることを表明します。

**COMPARE(*actual_exp,expected_exp,z*)**
*actual_exp* の値が *expected_exp* の値と等しいことを表明します。 *z* は、演算子を指名するオプションの引数です。

**TEXT(*display-expression*)**
表明が失敗した場合、*display-expression* を表明のルーチンに渡します。

**UNREACHABLE**
GOTO、RETURN、または SIGNAL ステートメントなどの進行中のステートメントによってこのステートメントがバイパスされたため、ステートメントに到達できないことを表明します。

***display-expression***
スカラー CHARACTER 式。

***test-expression***
計算スカラー式。必要に応じて BIT に変換されます。

***actual_exp***
計算され、変換される可能性がある計算式。 これはスカラー式で、*expected_exp* と同じタイプでなければなりません。

***expected_exp***
計算され、変換される可能性がある計算式。 これはスカラー式で、*actual_exp* と同じタイプでなければなりません。

© Copyright IBM Corp. 1999, 2022

**オペレーター**
CHAR(2) 定数。 大文字の場合、この定数の値は、EQ、LE、LT、GT、GE、または NE のいずれかでなければなりません。 *operator* を指定しない場合、EQ がデフォルト値です。

**EQ**
次と等しい

**LE**
より小か等しい

**LT**
より小

**GT**
より大

**GE**
より大か等しい

**NE**
等しくない

表明が失敗した場合、コンパイルされたコードは、ASSERT UNREACHABLE ステートメントに対してルーチン IBMPASU を呼び出し、ASSERT TRUE と ASSERT FALSE の各ステートメントに対してルーチン IBMPAST を呼び出し、さらに ASSERT COMPARE ステートメントに対してルーチン IBMPASC を呼び出します。 これらのルーチンは OPTLINK リンケージを使用する必要があります。

**注 :**

- ASSERT(CONDITION) コンパイラー・オプションでは、ASSERTION 条件が発生します。
- ASSERT(ENTRY) コンパイラー・オプションが使用されている場合、コンパイルされたコードはルーチン IBMPASU/IBMPAST/IBMPASC を呼び出します。

コンパイルされたコードは、以下の BYVALUE パラメーターを使用して、ルーチン IBMPASU および IBMPAST を呼び出します。

- 可変文字ストリングとして PACKAGENAME 値を含むバッファーのアドレスを保持する POINTER。
- 可変文字ストリングとして PROCNAME 値を含むバッファーのアドレスを保持する POINTER。
- SOURCELINE 値を保持する FIXED BINARY(31)。
- TEXT 値の ADDRDATA を保持する POINTER。 TEXT 節を省略した場合は、値 SYSNULL が渡されます。
- TEXT 値の LENGTH を保持する FIXED BINARY(31)。 TEXT 節を省略した場合は、値 0 が渡されます。

コンパイルされたコードは、以下の BYVALUE パラメーターを使用して、ASSERT COMPARE ステートメントに対してルーチン IBMPASC を呼び出します。

- 可変文字ストリングとして PACKAGENAME 値を含むバッファーのアドレスを保持する POINTER。
- 可変文字ストリングとして PROCNAME 値を含むバッファーのアドレスを保持する POINTER。
- SOURCELINE 値を保持する FIXED BINARY(31)。
- *actual_exp* 値の ADDRDATA を保持する POINTER。 この節は省略できません。
- *actual_exp* 値の LENGTH を保持する FIXED BINARY(31)。
- *expected_exp* 値の ADDRDATA を保持する POINTER。 この節は省略できません。
- *expected_exp* 値の LENGTH を保持する FIXED BINARY(31)。
- TEXT 値の ADDRDATA を保持する POINTER。 TEXT 節を省略した場合は、値 SYSNULL が渡されます。
- TEXT 値の LENGTH を保持する FIXED BINARY(31)。 TEXT 節を省略した場合は、値 0 が渡されます。

実際の式および予期される式を表すストリングは、これらの式のタイプによって決まります。 式に含まれるものに応じて、以下のようになります。

- GRAPHIC または WIDECHAR 以外の計算タイプの場合、ストリングは CHARACTER に変換された式の値になります。
- POINTER または HANDLE の場合、ストリングはその HEX 値になります。

- ORDINAL の場合、ストリングはその ORDINALNAME 値になります。
- その他のタイプはすべて、ストリングはヌル・ストリングになります。

### 例: ASSERT TRUE、ASSERT FALSE、および ASSERT UNREACHABLE の各ステートメントの使用法

以下の例は、ASSERT TRUE、ASSERT FALSE、および ASSERT UNREACHABLE の各ステートメントの使用法を示しています。 この例で使用されるルーチンをコーディングする必要があります。

```
asserts: package;

    main: proc options(main);

        dcl n fixed bin;

        n = 1;
        assert true( n> 0 );
        assert true( n= 2 ) text('n not equal to 2');
        assert unreachable;

    end;

    ibmpasu:
        proc( packagename_ptr, procname_ptr, assert_sourceline,
                text_addr, text_length )
        ext( '_IBMPASU')
        options( byvalue linkage(optlink) );

        dcl packagename_ptr   pointer;
        dcl procname_ptr      pointer;
        dcl assert_sourceline fixed BINARY(31);
        dcl text_addr         pointer;
        dcl text_length       fixed BINARY(31);

        dcl assert_packagename char(100) var based(packagename_ptr);
        dcl assert_procname char(100) var based(procname_ptr);
        dcl assert_text char(text_length) based(text_addr);

        put skip edit( 'unreachable code hit on line ',
                           trim(assert_sourceline),
                           ' in ',
                           assert_packagename,
                           ':', assert_procname )
                       ( a );
        if text_length = 0 then;
        else
           put skip list( assert_text );
    end;

  ibmpast:
        proc( packagename_ptr, procname_ptr, assert_sourceline,
              text_addr, text_length )
        ext( '_IBMPAST')
        options( byvalue linkage(optlink) );

        dcl packagename_ptr   pointer;
        dcl procname_ptr      pointer;
        dcl assert_sourceline fixed BINARY(31);
        dcl text_addr         pointer;
        dcl text_length       fixed BINARY(31);

        dcl assert_packagename char(100) var based(packagename_ptr);
        dcl assert_procname char(100) var based(procname_ptr);
        dcl assert_text char(text_length) based(text_addr);


        put skip edit( 'conditional assertion failed on line ',
                           trim(assert_sourceline),
                           ' in ',
                            assert_packagename,
                            ':', assert_procname )
                          ( a );
        if text_length = 0 then;
        else
           put skip list( assert_text );
    end;
```

次に、ASSERT COMPARE ステートメントの使用例を示します。 この例で使用されるルーチンをコーディングする必要があります。

### 例: ASSERT COMPARE ステートメントの使用法

```
asserts: package;

   main: proc options(main);

     dcl n fixed bin;

     n = 1;
     assert compare(n,1);
     assert compare(n,2) text("n not equal to 2");
     assert unreachable;
   end;

   ibmpasc:
    proc( packagename_ptr, procname_ptr, assert_sourceline,
    actual_addr, actual_length,
    expected_addr, expected_length,
    text_addr, text_length )
    ext( '_IBMPASC')
    options( byvalue linkage(optlink) );

    dcl packagename_ptr   pointer;
    dcl procname_ptr      pointer;
    dcl assert_sourceline fixed BINARY(31);
    dcl actual_addr       pointer;
    dcl actual_length     fixed BINARY(31);
    dcl expected_addr     pointer;
    dcl expected_length   fixed BINARY(31);
    dcl text_addr         pointer;
    dcl text_length       fixed BINARY(31);

    dcl assert_packagename char(100) var based(packagename_ptr);
    dcl assert_procname char(100) var based(procname_ptr);
    dcl assert_text char(text_length) based(text_addr);
    dcl actual_text char(actual_length) based(actual_addr);
    dcl expected_text char(expected_length)
                                       based(expected_addr);

    put skip edit( 'compare code hit on line ',
                   trim(assert_sourceline),
                   ' in ',
                   assert_packagename,
                   ':', assert_procname )
              ( a );

    if text_length = 0 then;
    else
       put skip list( assert_text );

    if actual_length = 0 then;
    else
       put skip list( actual_text );

    if expected_length = 0 then;
    else
       put skip list( expected_text );

  end;
```

表明が失敗し、ASSERT(CONDITION) コンパイラー・オプションが有効な場合、349 ページの『ASSERTION 条件』が発生し、適切なメッセージと新規の 498 ページの『ONTEXT』、497 ページの『ONPACKAGE』、491 ページの『ONACTUAL』、および 494 ページの『ONEXPECTED』 組み込み関数の適切な値が返されます。

- ONPACKAGE 組み込み関数は、ASSERTION 条件が発生した PACKAGE の名前を提供します。
- ONTEXT 組み込み関数は、ASSERT ステートメントが失敗した場合に TEXT ストリングの値を提供します (ステートメントに TEXT オプションがない場合は、ヌル・ストリングを返します)。
- ONACTUAL と ONEXPECTED の各組み込み関数は、ASSERT COMPARE ステートメントが失敗した場合に、それぞれ実際の値と予期される値が指定されたストリングを提供します。

**注 :**

- ASSERTION 条件は常に割り込み可能です。
- ASSERTION 条件の暗黙アクションは、メッセージが出力され、ERROR 条件が発生することです。
- ASSERTION 条件の正常な戻りは、その次のステートメントを使用して実行を続けることです。
- ASSERTION 条件に関連付けられた ONCODE は、以下のとおりです。

| 条件コード | 意味 |
|---|---|
| **430** | SIGNAL ASSERTION |
| **431** | TEXT 節を含まない ASSERT TRUE/FALSE ステートメントが失敗した |
| **432** | TEXT 節を含む ASSERT TRUE/FALSE ステートメントが失敗した |
| **433** | TEXT 節を含まない ASSERT UNREACHABLE ステートメントが失敗した |
| **434** | TEXT 節を含む ASSERT UNREACHABLE ステートメントが失敗した |
| **435** | TEXT 節を含まない ASSERT COMPARE ステートメントが失敗した |
| **436** | TEXT 節を含む ASSERT COMPARE ステートメントが失敗した |

# 代入ステートメントと複合代入ステートメント

代入ステートメントは、式を計算し、その値を 1 つまたは複数のターゲット変数に割り当てます。 データを内部で移動させたり、 計算を指定したりするときにこれらのステートメントを使用します。

データを内部で移動させるときは、STRING オプションが指定された GET または PUT ステートメントを使用することもできます。 さらに、PUT ステートメントでは実行する必要のある計算を 指定することもできます。 295 ページの『第 12 章 ストリーム指向データ伝送』を参照してください。

ターゲット変数または疑似変数の属性は、ソース (変数、定数、 または式の計算結果) の属性と異なっている可能性があるので、 代入ステートメントで変換が必要になる 場合があります。

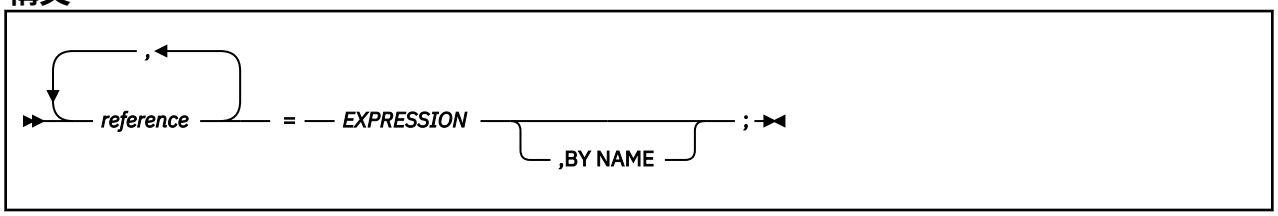
**関連情報**

73 ページの『データ変換』
この章では、計算データのデータ変換について説明します。 PL/I は、特定の属性セットを持つデータ項目が、 別の属性セットを持つデータ項目に割り当てられると、 データ変換を行います。

## 代入ステートメント

このトピックでは、代入ステートメントの構文およびパラメーターについて説明します。

### 構文




### パラメーター

**参照**
割り当てを指定するターゲットを指定します。

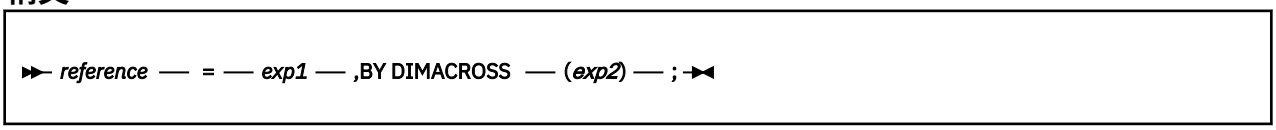
**expression**
計算され、変換される可能性がある式を指定します。

**BY NAME**
構造体割り当ての場合、BY NAME オプションは、202 ページの『BY NAME オプションを使用した構造体割り当て』で説明されている手順に従って割り当てを行うことを指定します。

# BY DIMACROSS オプションを使用する代入ステートメント

このトピックでは、BY DIMACROSS オプションを使用する代入ステートメントの構文およびパラメーターについて説明します。

## 構文




## パラメーター

**参照**
割り当てを指定するターゲットを指定します。

**exp1**
計算され、変換される可能性がある式を指定します。

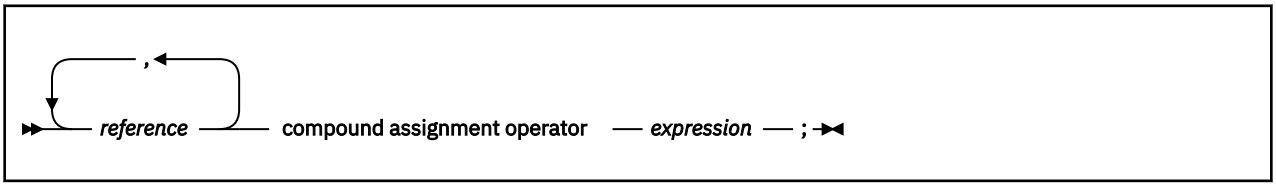
**BY DIMACROSS**
構造体割り当ての場合、BY DIMACROSS オプションは、203 ページの『BY DIMACROSS オプションを使用した構造体割り当て』で説明されている手順に従って割り当てを行うことを指定します。

**exp2**
関連する DIMACROSS 構造体エレメントに追加する索引として使用する式を指定します。

# 複合代入ステートメント

このトピックでは、複合代入ステートメントの構文およびパラメーターについて説明します。

## 構文




## パラメーター

**参照**
割り当てを指定するターゲットを指定します。

**compound assignment operator**
割り当てが行われる前に、参照および計算された式に適用する演算子を指定します。 198 ページの表 42 に、複合割り当てで使用できる 複合代入演算子をリストしています。

**expression**
計算され、変換される可能性がある式を指定します。

区域割り当てについては、251 ページの『区域データとその属性』を参照してください。

表 *42.* 複合代入演算子

| 複合代入演算子 | 意味 |
|---|---|
| += | 式の計算、加算および代入 |
| -= | 式の計算、減算および代入 |

表 42. 複合代入演算子 (続き)

| 複合代入演算子 | 意味 |
|---|---|
| *= | 式の計算、乗算および代入 |
| /= | 式の計算、除算および代入 |
| \|= | 式の計算、OR および代入 |
| &= | 式の計算、AND および代入 |
| ∥= | 式の計算、連結と代入 |
| **= | 式の計算、指数化および代入 |
| ¬= または <> | 式の計算、排他 OR および代入 |

最初に演算子がターゲットおよびソースに適用され、 次にその結果がターゲットに割り当てられます。

次の例を参照してください。

```
X += 1          は次と等しい      X = X+(1)
X *= Y+Z        は次と等しい      X = X*(Y+Z)
```

ただし、以下のステートメントは等しくありません。

```
X *= Y+Z        は次と等しくない  X = X*Y+Z
```

複合割り当てでは、ターゲット変数に指定された任意の添え字またはロケーター式が、 一度だけ計算されます。

f が関数であり、X が配列の場合、以下のステートメントは等しくありません。

```
X(f()) += 1     は次と等しくない  X(f()) = X(f())+1
```

関数 f は一度だけ呼び出されます。

**関連情報**

203 ページの『内部データ移動の例』
この代入ステートメント例は、内部データ移動に使用できます。 代入記号の右側にある式の値は、左側の変数に代入されます。

# ターゲット変数

ターゲット変数は、要素変数、配列変数、構造体変数、または疑似変数にすることができます。

## 非計算ターゲット

非計算ターゲットには、AREA、ENTRY、FILE、FORMAT、HANDLE、LABEL、OFFSET、ORDINAL、POINTER、およびタイプ付き STRUCTURE があります。

ターゲットが非計算である場合、ソースの属性は次のように制限されます。

| ターゲット | ソース | 考慮事項 |
|---|---|---|
| AREA | 次のいずれかである必要があります。<br>1. 区域<br>2. empty()<br>3. ヌル・ビットまたは文字ストリング | 区域にヌル・ストリングを割り当てることは、区域に empty() を割り当てることと同等です。 |

| ターゲット | ソース | 考慮事項 |
|---|---|---|
| ENTRY | 次のいずれかである必要があります。<br>1. 入り口 (ターゲットが LIMITED である場合、ソースも同じである必要がある)<br>2. nullentry()<br>3. ヌル・ビットまたは文字ストリング | 入り口にヌル・ストリングを割り当てることは、入り口に nullentry() を割り当てることと同等です。 |
| FILE | ファイル変数またはファイル定数である必要があります。 | |
| FORMAT | フォーマット変数またはフォーマット定数である必要があります。 | |
| HANDLE | 次のいずれかである必要があります。<br>1. 同じ構造体タイプへのハンドル<br>2. null()<br>3. sysnull()<br>4. ヌル・ビットまたは文字ストリング | DFT(NULSTRPTR) コンパイラー・オプションは、ハンドルにヌル・ストリングを割り当てることが、ターゲットに null() を割り当てることと同等であるのか、ターゲットに sysnull() を割り当てることと同等であるのかを指定します。 |
| LABEL | ラベル変数またはラベル定数である必要があります。 | |
| OFFSET | 次のいずれかである必要があります。<br>1. オフセット<br>2. ポインター (ターゲットに修飾する AREA がある場合)<br>3. null()<br>4. sysnull()<br>5. ヌル・ビットまたは文字ストリング | DFT(NULSTRPTR) コンパイラー・オプションは、オフセットにヌル・ストリングを割り当てることが、ターゲットに null() を割り当てることと同等であるのか、ターゲットに sysnull() を割り当てることと同等であるのかを指定します。 |
| ORDINAL | 同じタイプの序数である必要があります。 | |
| POINTER | 次のいずれかである必要があります。<br>1. ポインター<br>2. 修飾の AREA 付きのオフセット<br>3. null()<br>4. sysnull()<br>5. ヌル・ビットまたは文字ストリング | DFT(NULSTRPTR) コンパイラー・オプションは、ポインターにヌル・ストリングを割り当てることが、ターゲットに null() を割り当てることと同等であるのか、ターゲットに sysnull() を割り当てることと同等であるのかを指定します。 |
| タイプ付き STRUCTURE | 同じタイプの構造体である必要があります。 | |

## 配列ターゲット

配列割り当ての場合、各ターゲット変数は、スカラーまたは構造体の配列でなければなりません。

ソースは、そのターゲットと同じ次元数であり、またすべての次元についても 同じ境界を持つ、スカラーまたは式でなければなりません。

## 共用体のターゲット

共用体の割り当てはできません。

## 構造体ターゲット

BY NAME 構造体割り当てでは、各ターゲット変数は構造体でなければならず、右側は構造体参照でなければなりません。 BY NAME または BY DIMACROSS を使用しない構造体割り当てでは、各ターゲット変数は構造体でなければならず、右側はターゲット構造体と同じ構造を持つスカラー式または構造式でなければなりません。

- 構造体は、小構造化がすべて同じで、中に含まれるエレメントと配列の数が同じでなければなりません。

- 構造体内 (および小構造があれば、その内部で) のエレメントや配列の位置指定は、 同じでなければなりません。
- 対応する位置にある配列は、同じ境界を持たなければなりません。

BY DIMACROSS を使用する構造体割り当てでは、ターゲット変数は構造体である必要があります。 DIMACROSS 索引式は、DIMACROSS 属性を指定して宣言されたすべての構造体に適用され、ターゲット、またはソースの一部のいずれかとして割り当てで使用されます。 これらの割り当てには、以下の制約事項が適用されます。

- 1 つのターゲット参照のみが有効です。
- ソース内のすべての構造体の構造化および境界は、ターゲット内のそれらと一致しなければなりません。

BY NAME および BY DIMACROSS オプションを使用しない構造体割り当てでは、ターゲット構造体に非計算データが含まれている場合でも、ソースをヌル・ビット・ストリング (''b) にすることができます。 この場合、割り当ては以下のすべての条件が真であると想定して実行されます。

1. ターゲットはすべて '00'x で埋められている。
2. 数値ターゲット・フィールドはすべて 0 に設定されている。
3. NONVARYING CHARACTER、UCHAR、WIDECHAR、および GRAPHIC のフィールドはすべてブランクで埋められている。

# 割り当ての実行方法

このセクションでは、要素割り当ておよび集合割り当ての実行方法について説明します。

## エレメントの割り当て

エレメントの割り当ては、次のように行われます。

1. 最初に以下が評価されます。
   - 添え字
   - POSITION 属性式
   - ターゲット変数のロケーター修飾
   - SUBSTR 疑似変数参照の 2 番目と 3 番目の引数
2. 次に、右辺の式が評価されます。
3. それぞれのターゲット変数ごとに (左から右の順で)、式の値がデータ変換規則に従ってターゲット変数の特性に変換されます。 そのあと、変換後の値がターゲット変数に割り当てられます。

## 集合の割り当て

集合の割り当て (配列割り当ておよび構造体割り当て) は、 次のように一連のエレメントの割り当てに展開されます。

1. 元のステートメントのラベル接頭部が、生成された別のステートメントの先頭にある ヌル・ステートメントに付けられます。
2. 配列割り当てや構造体割り当ては、複数あるときは、繰り返し行われます。
3. 先行の配列割り当てまたは構造体割り当てによって、なんらかの代入ステートメントが 生成されることがあります。 集合割り当ての最初のターゲット変数は、制御変数と呼ばれます。 (これは疑似変数の最初の引数であることもあります)。 制御変数が配列の場合は、配列拡張が実行されます。それ以外の場合は、構造体拡張が実行されます。
4. 集合割り当てが以下のいずれかの条件を満たしている場合、集合割り当ては、一連の要素割り当てに展開せず、全体として行うことができます。
   - 配列が介在配列ではない。
   - 構造体が連続していて、同じフォーマットを持っている。

## 配列の割り当て

配列の割り当てでは、すべての配列オペランドの次元数が同じで、境界が一致していなければなりません。配列の割り当ては、次のようにループに展開されます。

```
do J1 = lbound(Control-variable,1) to
        hbound(Control-variable,1);
do J2 = lbound(Control-variable,2) to
        hbound(Control-variable,2);
.
.
.
do Jn = lbound(Control-variable,N) to
        hbound(Control-variable,N);

generated assignment statement

end;
```

この拡張では、n は、割り当てに関与する制御変数の次元の数です。 生成された代入ステートメントでは、すべての配列オペランドに、 ダミー変数 j1 から jn を 使用して (左から右へ) 完全に添え字が付けられます。 添え字を付けずに配列オペランドを書いた場合、その配列オペランドは、j1 から jn の添え字だけを持つことになります。 クロスセクション表記を使用する場合、アスタリスクは、j1 から jn で置き換えられます。 元の代入ステートメントにこの条件接頭語が付いている場合は、 生成された代入ステートメントにこの条件接頭語が付けられます。

生成された代入ステートメントが構造体割り当てであれば、そのステートメントは 202 ページの『BY NAME オプションを使用しない構造体割り当て』の記載どおりに展開されます。

## *BY NAME* オプションを使用しない構造体割り当て

BY NAME オプションが指定されていない構造体割り当てでは、以下の条件が適用されます。

- どのオペランドも、配列であってはなりませんが、配列を含む構造体に することはできます。
- 構造体に直接含まれる項目数は、すべての構造体オペランドについて同数 *k* でなければ なりません。

これらの構造体割り当ては、以下のように実行されます。

- 代入ステートメントは、生成された *k* 個の代入ステートメントで 置き換えられます。
- i 番目の生成された代入ステートメントが、 それぞれの構造体オペランドを i 番目に含まれる項目で置き換えることによって、 もとの代入ステートメントから派生します。 そのように生成された代入ステートメントは、さらに展開を要求することができます。
- 元のステートメントの条件接頭語が、生成されたすべての代入ステートメントに付けられます。

## *BY NAME* オプションを使用した構造体割り当て

BY NAME オプションの指定がある構造体割り当ての場合、構造体の割り当ては下記のステップに従って展開されます。これらのステップでは、さらに配列の割り当てや構造体の割り当てを生成することができます。 どのオペランドも配列であってはなりません。

1. 制御変数に直接含まれる最初の項目が考慮されます。
2. それぞれの構造体オペランドとターゲット変数に同じ名前の項目が直接 含まれている場合は、以下に述べるようにして 代入ステートメントが生成されます。
   a. それぞれの構造体オペランドとターゲット変数を、 それぞれに直接含まれている同名の項目で置き換えるという方法で、 代入ステートメントが派生します。 いずれかの構造体に同名の項目が含まれていない場合は、ステートメントは 生成されません。
   b. 生成された代入ステートメントが構造体の割り当てまたは構造体の配列の割り当てで ある場合は、BY NAME が付けられます。
   c. 元の代入ステートメントの条件接頭語が、生成されたすべての代入ステートメントに 付けられます。
   d. ターゲット構造体には、共用体は含まれません。

3. 制御変数に直接含まれる項目ごとに、ステップ 2 が繰り返されます。 割り当ては、制御変数に含まれる項目の順序で生成されます。

### *BY DIMACROSS* オプションを使用した構造体割り当て

BY DIMACROSS オプションの指定がある構造体割り当ての場合、構造体の割り当ては、DIMACROSS 式を、関連する配列エレメントへの索引として使用して展開されます。 どのオペランドも配列であってはなりません。

## 複数割り当て

割り当ては、1 つの代入ステートメントで複数の変数に対して 行うことができます。

次の例を考えてください。

```
A,X = B + C;
```

B + C の値が A と X の両方に 割り当てられます。 普通は、このステートメントは、 下記のステートメントが実行された場合と同じ結果になります。

```
Temporary = B + C;
A = Temporary;
X = Temporary;
```

代入ステートメント内のソースは、スカラーまたはスカラー配列でなくてはなりません。さらに、ソースが配列の場合は、すべてのターゲットも配列でなくてはなりません。 ソースが定数である場合、各ターゲットに対して左から右に割り当てられます。 ソースが定数ではない場合、一時変数に割り当てられ、 このあと各ターゲットに対して左から右に割り当てられます。

ターゲットは、単一の割り当てで可能な参照であればどの参照でもかまいません。

BY DIMACROSS は、複数の割り当てでは使用できません。

非推奨ですが、複数の代入で、複合代入演算子を使用できます。 ただし、結果は必ずしも単純に期待されるものにならないことがあります。例えば、以下のステートメントは一般的に同じ結果を生成しません。

```
c, c += c;
c, c = c + c;
```

## 例

このセクションでは、内部データ移動、式値の代入、および構造体の割り当てに使用する代入ステートメントの例を示します。

### 内部データ移動の例

この代入ステートメント例は、内部データ移動に使用できます。 代入記号の右側にある式の値は、左側の変数に代入されます。

```
NTOT=TOT;
```

### 式値の代入例

この例は、代入記号の左側にある変数に代入する値を持つ式です。

```
Av=(Av*Num+Tav*Tnum)/(Num+Tnum);
```

### BY NAME を使用した構造体の割り当て例

この例は、BY NAME オプションを使用した構造体割り当てを示しています。

```
declare      declare       declare
1 One,       1 Two,        1 Three,
```

```
2 Part1,     2 Part1,      2 Part1,
 3 Red,       3 Blue,       3 Red,
 3 Orange,    3 Green,      3 Blue,
2 Part2,      3 Red,        3 Brown,
 3 Yellow,   2 Part2,      2 Part2,
 3 Blue,      3 Brown,      3 Yellow,
 3 Green;     3 Yellow;     3 Green;
```

```
1  One = Two, by name;
2  One.Part1 = Three.Part1, by name;
```

**1**

最初の代入ステートメントは、次のステートメントと同じです。

```
One.Part1.Red    = Two.Part1.Red;
One.Part2.Yellow = Two.Part2.Yellow;
```

**2**

2 番目の代入ステートメントは、次のステートメントと同じです。

```
One.Part1.Red = Three.Part1.Red;
```

## BY DIMACROSS を使用した構造体の割り当て例

以下の例は、BY DIMACROSS オプションを使用した構造体割り当てを示しています。

### 例 1

このコードはすべての行エレメントを合計します。

```
dcl
  1 x,
     2 a fixed bin(31),
     2 b fixed bin(31),
     2 c fixed bin(31),
     2 d fixed bin(31);

dcl 1 xa(17) dimacross like x;

dcl jx fixed bin;

x = 0;

do jx = lboundacross( xa ) to hboundacross( xa );

   x = x + xa, by dimacross( jx );

end;
```

ループ内の代入は、以下のステートメントと同等です。

```
   x.a = x.a + xa.a(jx);
   x.b = x.b + xa.b(jx);
   x.c = x.c + xa.c(jx);
   x.d = x.d + xa.d(jx);
```

### 例 2

このコードは、xa の最初と 17 番目の列のエントリーを交換します。

```
dcl
  1 x,
     2 a fixed bin(31),
     2 b fixed bin(31),
     2 c fixed bin(31),
     2 d fixed bin(31);

dcl 1 xa(17) dimacross like x;
```

```
dcl y like x;

x = xa, by dimacross( 1  );
y = xa, by dimacross( 17 );
xa = y, by dimacross( 1  );
xa = x, by dimacross( 17 );
```

## ATTACH ステートメント

ATTACH ステートメントは、スレッドをサイズまたは作成します。

ATTACH ステートメントについて詳しくは、セクション 367 ページの『第 17 章 マルチスレッド化機能』の 368 ページの『ATTACH ステートメント』を参照してください。

## BEGIN ステートメント

BEGIN ステートメントとそれに対応する END ステートメントは、 開始ブロックを区切ります。

BEGIN ステートメントの詳細については、87 ページの『第 5 章 プログラムの編成』の 109 ページの『BEGIN ステートメント』を参照してください。

## CALL ステートメント

CALL ステートメントはサブルーチンを呼び出します。

CALL ステートメントの詳細については、121 ページの『CALL ステートメント』を参照してください。

## CANCEL THREAD ステートメント

スレッドを取り消すには、CANCEL THREAD ステートメントを使用します。

CANCEL THREAD ステートメントの詳細については、369 ページの『スレッドの取り消し』を参照してください。

## CLOSE ステートメント

CLOSE ステートメントは、そのデータ・セットからオープンされたファイルを分離します。

CLOSE ステートメントの詳細については、273 ページの『第 10 章 入出力』の 284 ページの『CLOSE ステートメント』を参照してください。

## DECLARE ステートメント

DECLARE ステートメントは、 名前の一部の属性またはすべての属性を指定します。

DECLARE ステートメントの詳細については、148 ページの『DECLARE ステートメント』を参照してください。

## DEFAULT ステートメント

DEFAULT ステートメントは、(属性セットが完全ではない場合は) データ属性のデフォルトを指定します。

DEFAULT ステートメントの詳細については、166 ページの『DEFAULT ステートメント』を参照してください。

## DEFINE ALIAS ステートメント

DEFINE ALIAS ステートメントは、 ユーザーが別名に割り当てたデータ・タイプ属性のセットのために、同義語として使用することができる名前を指定します。

DEFINE ALIAS ステートメントの詳細については、135 ページの『第 6 章 タイプ定義』の 135 ページの『DEFINE ALIAS ステートメント』を参照してください。

## DEFINE ORDINAL ステートメント

DEFINE ORDINAL ステートメントは、順序付けられた (名前付きの) 値の セットを表すタイプを指定します。

DEFINE ORDINAL ステートメントの詳細については、135 ページの『第 6 章 タイプ定義』の 136 ページの『DEFINE ORDINAL ステートメント』を参照してください。

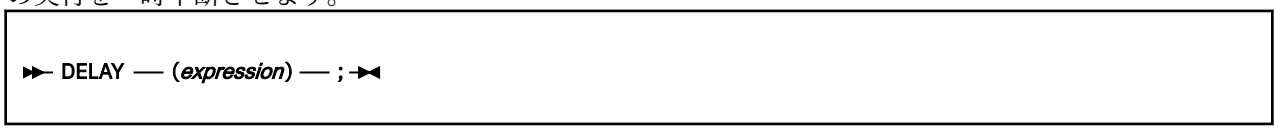
## DEFINE STRUCTURE ステートメント

DEFINE STRUCTURE ステートメントは名前付き構造体または 共用体タイプを指定します。

DEFINE STRUCTURE ステートメントの詳細については、135 ページの『第 6 章 タイプ定義』の 137 ページの『タイプ付き構造体と共用体の定義』を参照してください。

## DELAY ステートメント

DELAY ステートメントは、指定された時間の間、 アプリケーション・プログラム内の次のステートメントの実行を一時中断させます。




**expression**
式を指定します。この式は、計算されて FIXED BINARY(31,0) に変換されます。 指定されたミリ秒の間、実行が一時中断します。

最大待機時間は、23 時間 59 分です。

以下の例を参照してください：

- `delay (20);` 実行を 20 ミリ秒間中断します。
- `delay (10**3);` 実行を 1 秒間中断します。
- `delay (10*10**3);` 実行を 10 秒間中断します。

プログラムが CICS 下で実行されている場合、DELAY ステートメントは EXEC CICS DELAY コマンドを使用して実装されます。 EXEC CICS DELAY コマンドの時間間隔は現在、少なくとも 1 秒あります。 PL/I DELAY ステートメントで指定されたミリ秒数は、値が 1 秒未満の場合 (1 に設定されます) を除き、最も近い秒に切り捨てられます。

以下の例を参照してください：

- `delay(30);` CICS 下で 1 秒間実行を中断します。
- `delay(2100);` CICS 下で 2 秒間実行を中断します。

## DELETE ステートメント

DELETE ステートメントは、UPDATE ファイルからレコードを削除します。

DELETE ステートメントの詳細については、287 ページの『第 11 章 レコード単位データ伝送』の 289 ページの『DELETE ステートメント』を参照してください。

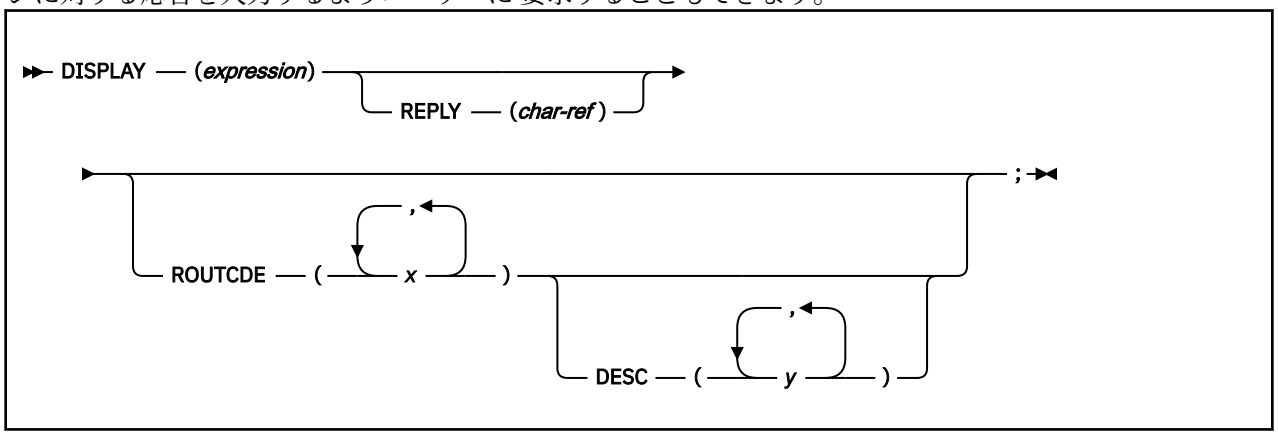
## DETACH ステートメント

DETACH ステートメントは、THREAD オプションで接続されたスレッドに関連付けられているシステム・リソースを解放します。

DETACH ステートメントの詳細については、367 ページの『第 17 章 マルチスレッド化機能』の 370 ページの『スレッドの切り離し』を参照してください。

# DISPLAY ステートメント

DISPLAY ステートメントは、ユーザーの表示画面にメッセージを表示し、 任意指定により、そのメッセージに対する応答を入力するようユーザーに 要求することもできます。




**expression**
必要に応じて、文字ストリングに変換されます。 この文字ストリングが表示されます。 このストリングに混合文字データが含まれていてもかまいません。 式が GRAPHIC 属性を持つ場合、この式は変換されません。

**REPLY (char-ref)**
ユーザーが入力した応答を受け取る文字参照を指定します。 応答には、CHARACTER、GRAPHIC、または混合データのいずれを 含んでいてもかまいません。

REPLY オプションは、ユーザーが応答を入力するまでプログラムの実行を 一時中断させることになります。

REPLY で GRAPHIC データが入力された場合、そのデータは、混合データを含む文字データとして受け取られます。 このような文字データは、 GRAPHIC BUILTIN を使用すれば GRAPHIC データに変換することができます。

**ルーチェ**
WTO で ROUTCDE として使用される 1 つ以上の値を指定します。 値は、1 から 16 までの符号なし整定数でなければなりません。 DISPLAY コンパイラー・オプションの WTO サブオプションは、デフォルト値を指定します。

**DESC**
WTO で DESC として使用される 1 つ以上の値を指定します。 値は、1 から 16 までの符号なし整定数でなければなりません。 DISPLAY コンパイラー・オプションの WTO サブオプションは、デフォルト値を指定します。

DISPLAY(WTO) オプションが有効になっている場合、ROUTCDE および DESC は無視されます。

### 例

ステートメント `display ('Communication link established.');` は、次のメッセージを表示します。

通信リンクが確立されました。

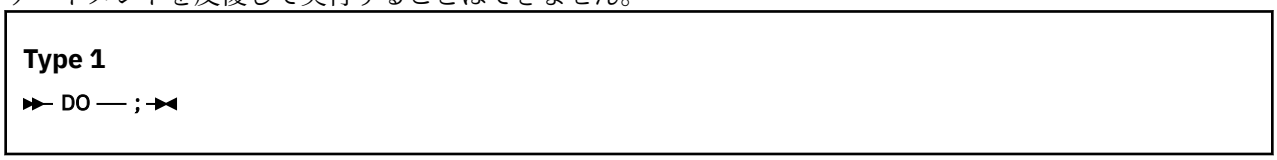
# DO ステートメント

DO ステートメントとそれに対応する END ステートメントは、DO グループと呼ばれるステートメント・グループを形成します。

**注 :** 条件接頭語は、DO ステートメントでは使用することはできません。

## タイプ 1

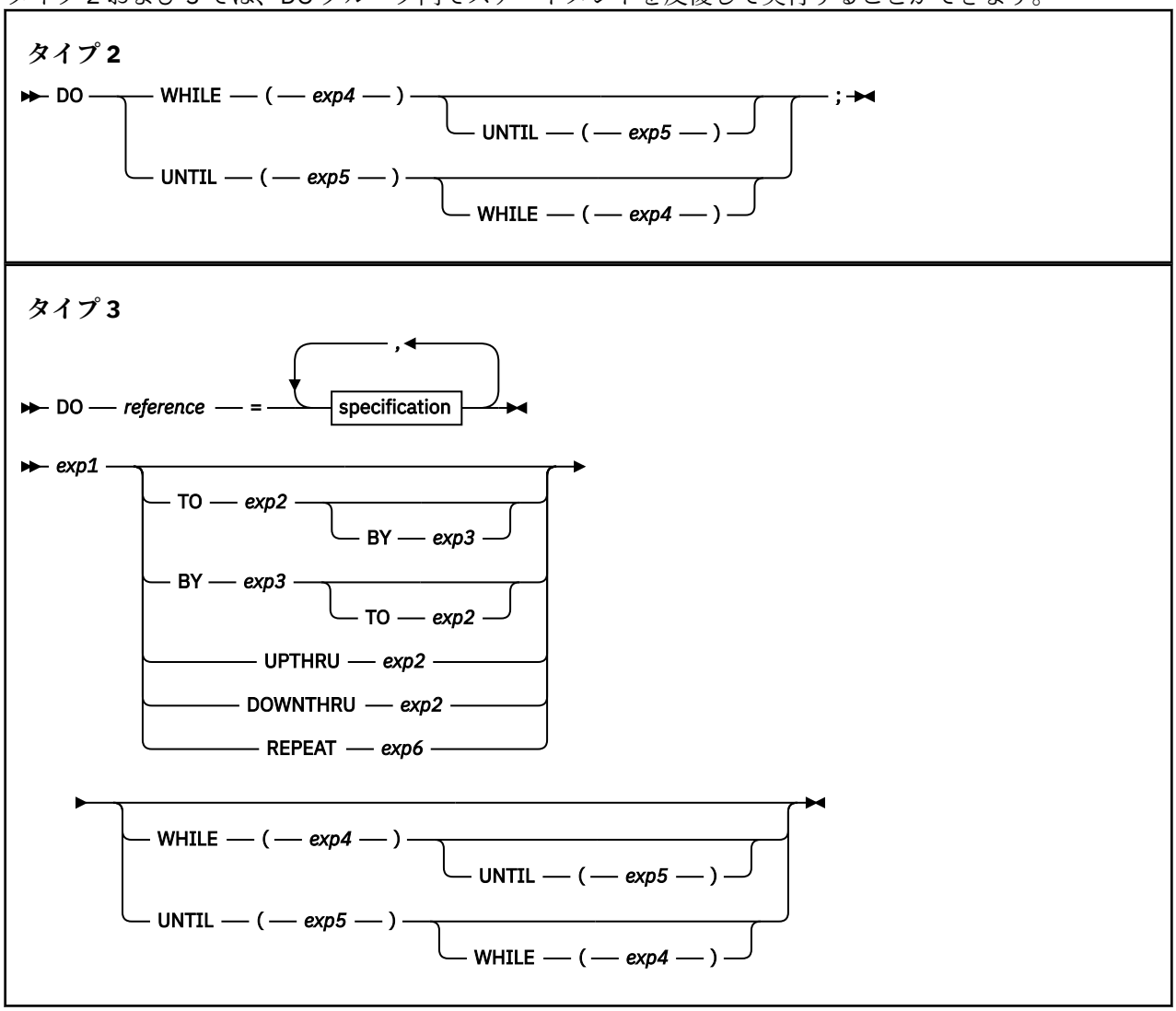
タイプ 1 の DO グループは、グループ内のステートメントが実行されることを示します。 グループ内のステートメントを反復して実行することはできません。




**expn**
*expression n* の省略形です。

## タイプ 2 およびタイプ 3

タイプ 2 および 3 では、DO グループ内でステートメントを反復して実行することができます。




**expn**
*expression n* の省略形です。

**WHILE (exp4)**
DO グループの各反復の前に、*exp4* を評価し、必要に応じてビット・ストリングに変換することを指定します。 結果のビット・ストリングのいずれかのビットが 1 であれば、その DO グループが実行されます。 すべてのビットが 0 の場合、またはストリングがヌルの場合、タイプ 2 DO グループの実行は終了します。 タイプ 3 の場合はこの WHILE オプションを含む指定に対応する実行だけが終了し、次の指定がある場合には、それが開始されます。

**UNTIL (exp5)**
DO グループの実行を反復すると、反復後に *exp5* が計算され、 必要に応じてビット・ストリングに変換されます。 結果ストリング内のすべてのビットが 0 の場合、またはストリングがヌルの場合は、DO グループの次の反復が実行されます。 いずれかのビットが 1 であれば、タイプ 2 の場合は DO グループの実行が終了し、タイプ 3 の場合はこの UNTIL オプションを含む指定に対応する実行だけが終了し、次の指定がある場合には、それが開始されます。

**参照**
参照として使用できる疑似変数は、SUBSTR、REAL、IMAG および UNSPEC のみです。 どのデータ・タイプでもかまいません。

参照の世代 g は、DO グループの始まりで、 初期値を表す式 (*exp1*) が計算される直前に、1 回だけ確立されます。 その DO グループ内で参照の世代が h に変更されても、DO グループの実行は、世代 g から派生した参照を使用して続行します。 ただし、DO グループ内でその参照が参照されると、世代 h が参照されます。 DO グループ内で世代 g を解放しようとすると、エラーになります。

最後の繰り返しの完了後に参照を参照した場合、その参照の値は指定でセットした 境界の範囲外の値になっています。 次の条件がアプリケーションに設定された境界に適合する場合は、 前述の内容が真になります。

- BY 値が正数で、参照の値が、指定でセットした境界の TO 値よりも大きい。
- BY 値が負数で、参照の値が、指定でセットした境界の TO 値よりも小さい。

参照がプログラム制御データ変数であり、 ロケーターでなければ、BY オプションと TO オプションは、指定では使用できません。

参照がプログラム制御変数であり、ロケーターまたは序数でなければ、UPTHRU オプションと DOWNTHRU オプションは、指定では使用できません。

**exp1**
参照の初期値を指定します。

TO、BY、および REPEAT の指定が省略された場合には、 DO グループは 1 回だけ実行され、参照は値として exp1 を持ちます。 WHILE(exp4) を指定した場合は、 exp4 が真でなければ 1 回も実行されません。

**TO exp2**
exp2 は、その指定に制御権が移動したときに計算され、保管されます。 この保管された値は、参照の終了値を示します。 DO グループの始めにテストされた参照の値が範囲内になければ、ただちに、指定に関して DO グループ内のステートメントの実行が終了します。 その後、次の指定があれば、その実行が開始されます。

TO exp2 の指定が省略され、BY exp3 が 指定されていると、WHILE オプションまたは UNTIL オプションによって 実行が終了するまで、あるいはほかのステートメントが DO グループの外へ 制御権を移動するまで、実行が繰り返されます。

**BY exp3**
exp3 は、その指定に制御権が移動したときに計算され、保管されます。 この保管された値は、DO グループの実行が反復されるたびに参照に 加算される増分を示しています。

BY exp3 の指定が省略され、 TO exp2 が指定されていると、exp3 は デフォルトの 1 になります。

BY 0 が指定されていると、WHILE オプションまたは UNTIL オプションによって停止されない限り、あるいは制御が DO グループの外部に移されるまで、DO グループの実行は無限に繰り返されます。

**UPTHRU exp2**
exp2 は、その指定に制御権が移動したときに計算され、保管されます。 この保管された値は、参照の終了値を示します。 DO グループの終わりでテストされた参照の値が所定の範囲内になければ、ただちに、その指定に関して DO グループ内のステートメントの実行が終了します。 その後、次の指定があれば、その実行が開始されます。

UPTHRU が指定されていると、ループ内のステートメントが 実行されたあと、想定し得る次の値で更新される前に、 参照は exp2 と比較されます。 ループは、少なくとも一度実行されます。

UPTHRU は主に、ループを使用した序数の処理中に使用されますが、算術変数またはロケーター制御変数である参照にも使用できます。 参照が序数でない場合、DO グループが実行されるたびに参照に加えられる増分は +1 と想定されます。

**DOWNTHRU exp2**
exp2 は、その指定に制御権が移動したときに計算され、保管されます。 この保管された値は、参照の終了値を示します。 DO グループの終わりでテストされた参照の値が所定の範囲内になければ、ただちに、その指定に関して DO グループ内のステートメントの実行が終了します。 その後、次の指定があれば、その実行が開始されます。

DOWNTHRU が指定されていると、ループ内のステートメントが 実行されたあと、参照が想定し得る次の値で更新される前に、 参照は exp2 と比較されます。 ループは、少なくとも一度実行されます。

DOWNTHRU は主に、ループを使用した序数の処理中に使用されますが、算術変数またはロケーター制御変数である参照にも使用できます。 参照が序数でない場合、DO グループが実行されるたびに参照に加えられる増分は -1 と想定されます。

**REPEAT exp6**
DO グループの実行が反復されるたびに、exp6 が計算され、 参照に割り当てられます。 反復実行は、WHILE オプションまたは UNTIL オプションによって停止されるまで、あるいは別のステートメントが DO グループの外部へ制御を移すまで続きます。

タイプ 3 の DO グループでは、exp1、exp2、および exp3 が計算される順序に依存してはいけません。 したがって、これらの式によって、他の式で使用される値を設定する関数で呼び出さないようにするのが適切と言えます。

DO グループの外から DO グループの中へ制御権を移動することができるのは、その DO グループがタイプ 1 の DO ステートメントで区切られている場合に限られます。 したがって、タイプ 2 とタイプ 3 の DO グループに ENTRY ステートメントを含めることはできません。 DO グループ内から呼び出したプロシージャーまたは ON ユニットから、 その DO グループへ制御を戻すことはできます。

以降の節では、タイプ 2 およびタイプ 3 の DO グループの 使用について説明します。 DO グループの例は、215 ページの『基本的な繰り返しの例』 ページから記載されています。

## タイプ 2 WHILE と UNTIL の使用方法

タイプ 2 の DO の指定で、WHILE オプションと UNTIL オプションの 両方を使用すると、その DO ステートメントによって、 次に示すように実行が反復されます。

```
Label:  do while (Exp4)
           until (Exp5)
           statement-1
           .
           .
           .
           statement-n
        end;
Next:   statement /* Statement following the do-group */
```

上記のものは、次のように展開されたものと同等です。

```
Label:  if (Exp4) then;
          else
           go to Next;
          statement-1
          .
          .
          .
          statement-n
Label2: if (Exp5) then;
          else
           go to Label;
Next:   statement /* Statement following the do-group */
```

WHILE オプションが省略されると、 ラベル Label の IF ステートメントは、 ヌル・ステートメントで置き換えられます。 WHILE オプションが省略されると、 ステートメント 1 からステートメント n までが 少なくとも一度は実行されます。

UNTIL オプションを省略したときは、 展開フォーマット内のラベル Label2 の IF ステートメントが GO TO Label ステートメントで 置き換えられます。

## タイプ 3 を 1 つの指定で使用する場合

1 つの指定で DO グループを実行した場合の結果を、以下に要約して示します。

1. 参照を指定し、BY、TO、UPTHRU、または DOWNTHRU の いずれかのオプションも指定した場合は、exp1 の参照への割り当てに先立って、exp1、 exp2、および exp3 が計算されます。 次に、初期値が参照に割り当てられます。例えば、次のようになります。

   ```
   do Reference = Exp1 to Exp2 by Exp3;
   ```

   疑似変数ではない変数の場合、前述の例の DO グループ定義によって取られる 処置は、下記の展開と同等です。

   ```
   E1=Exp1;
   E2=Exp2;
   E3=Exp3;
   V=E1;
   ```

   変数 V は、参照と同じ属性を持つ、コンパイラー作成の基底付き変数です。E1、E2、および E3 は、コンパイラー作成の変数です。
2. TO オプションがあると、制御変数の値が、前に評価された TO オプション内の式 (E2) がテストされます。
3. WHILE オプションを指定した場合は、WHILE オプション内の式が計算されます。 その結果が、偽 であれば、DO グループから制御が離れます。
4. DO グループ内のステートメントが実行されます。
5. UNTIL オプションを指定した場合は、 UNTIL オプション内の式が計算されます。 その結果が、真 であれば、DO グループから制御が離れます。
6. UPTHRU オプションが指定されていると、制御変数の値が、前に計算された UPTHRU オプション内の式と比較されます。
7. DOWNTHRU オプションが指定されていると、制御変数の値が、前に計算された DOWNTHRU オプション 内の式と比較されます。
8. 参照がある場合は、以下のようになります。
   a. TO オプションまたは BY オプションが指定されていると、前に計算された *exp3* (E3) が参照に加算されます。
   b. REPEAT オプションが指定されていると、 *exp6* が計算されて、参照に割り当てられます。
   c. TO、BY、REPEAT のどのオプションも指定していないときは、 DO グループから制御が離れます。
   d. UPTHRU オプションが指定されており、参照が序数を表していれば、現行値の後続値が 参照に割り当てられます。 参照が序数を表していないと、1 が参照に加算されます。
   e. DOWNTHRU オプションが指定され、参照が序数を表していれば、現行値の先行値が 参照に割り当てられます。 参照が序数を表していなければ、そこから 1 が減算されます。
   f. TO、BY、UPTHRU、DOWNTHRU、および REPEAT のオプションのうち 何も指定されていなければ、DO グループから制御が離れます。
9. 211 ページの『2』 に移動します。

## タイプ 3 を 2 つまたはそれ以上の指定で使用する場合

DO ステートメント内に複数の指定がある場合、2 番目の指定も最初の指定とあらゆる点で同じように展開されます。 ただし、DO グループ内のステートメントがプログラム内で実際に重複することはありません。後続の指定が実行されるのは、先行の指定が終了したあとです。

最後の指定の実行が終了すると、 制御はその DO グループのあとのステートメントに移ります。

## タイプ 3 を TO、BY、REPEAT を指定して使用する場合

TO オプションや BY オプションを使用すれば、参照を一定の増分 (正または負) で変化させることができます。 一方、TO オプションや BY オプションの代わりに REPEAT オプションを使用すれば、制御変数を非線形に変化させることができます。 REPEAT オプションは、算術タイプ以外の制御変数 (ポインターなど) にも使用することができます。

タイプ 3 の DO 指定で TO オプションと BY オプションの両方を使用すると、 その DO グループの処置は次のように定義されます。

```
Label:  do Variable=
            Exp1
            to Exp2
            by Exp3
            while (Exp4)
            until(Exp5);
            statement-1
            .
            .
            .
            statement-m
Label1:  end;
Next:    statement
```

前述の DO グループの定義によって取られる処置は、 次のように展開されたものとまったく同等です。 この展開において、V はコンパイラー生成変数で、Variable と 同じ属性を持ちます。また、E1、E2、および E3 も、 コンパイラー生成変数です。

```
Label:  E1=Exp1;
         E2=Exp2;
         E3=Exp3;
         V=E1;
Label2: if (E3>=0)&(V>E2)|(E3<0)&(V<E2) then
          go to Next;
        if (Exp4) then;
          else
           go to Next;
          statement-1
          .
          .
          .
          statement-m
Label1: if (Exp5) then
          go to Next;
Label3: V=V+E3;
        go to Label2;
Next:   statement
```

この指定で REPEAT オプションを使用したときは、DO グループの処置は 次のように定義されます。

```
Label:  do Variable=
            Exp1
            repeat Exp6
            while (Exp4)
            until(Exp5);
            statement-1
            .
            .
            .
            statement-m
Label1:  end;
Next:    statement
```

前述の DO グループの定義によって取られる処置は、 次のように展開されたものとまったく同等です。

```
Label:  E1=Exp1;
        V=E1;
Label2: ;
        if (Exp4) then;
        else
          go to Next;
        statement-1
        .
        .
```

```
           .
           statement-m
   Label1: if (Exp5) then
              go to Next;
   Label3: V=Exp6;
              go to Label2;
   Next:   statement
```

例に示した展開では、さらに次のような規則があります。

1. 前述の展開は、1 つの指定の結果だけを示しています。 DO ステートメント内に複数の指定がある場合は、展開フォーマット内のラベル NEXT が 付いているステートメントが、次の指定の最初のステートメントになります。 2 番目の展開も、最初の展開とあらゆる点で同じように展開されます。 しかし、プログラムでは、statement-1 から m は実際には重複しません。
2. WHILE 文節を省略したときは、それぞれの展開フォーマット内の statement-1 のすぐ前にある IF ステートメントも省略されます。
3. UNTIL 文節を省略したときは、それぞれの展開フォーマット内の statement-m のすぐあとにある IF ステートメントも省略されます。

## タイプ 3 を UPTHRU を指定して使用する場合

タイプ 3 の DO 指定に UPTHRU オプションが含まれる場合、DO グループの処置は、次のようにして定義されます。

```
   Label:  do Variable = Exp1 upthru Exp2 while (Exp4) until (Exp5);
              statement1
              .
              .
              .
              statementn
   Label1: end;
   Next: statement
```

上述の DO グループの処置は、次のように展開されたものと同等です。 この展開において、*V* はコンパイラー生成変数で、 *Variable* と同じ属性を持ちます。 また、*E1* と *E2* も、コンパイラー生成変数です。

```
   Label:  E1 = Exp1;
           E2 = Exp2;
           V = E1;
   Label2: if (Exp4) then;
           else
             go to next;
           statement1
           .
           .
           .
           statementn


   Label1: if (Exp5) then
              go to Next;
           if V ≥ E2 then
              go to Next;
           V = V + 1;
              go to Label2;
   Next: statement
```

参照が序数を表していれば、ステートメント `V = V + 1` は、`V = ordinalsucc(V)` で置き換えられます。

## タイプ 3 を DOWNTHRU を指定して使用する場合

タイプ 3 の DO 指定に DOWNTHRU オプションが含まれる場合、DO グループの処置は、次のようにして定義されます。

```
   Label:  do Variable = Exp1 downthru Exp2 while (Exp4) until (Exp5);
              statement1
              .
              .
              .
```

```
            statement_n
Label1: end;
Next: statement
```

上述の DO グループの処置は、次のように展開されたものと同等です。 この展開において、*V* はコンパイラー生成変数で、 *Variable* と同じ属性を持ちます。 また、*E1* と *E2* も、コンパイラー生成変数です。

```
Label:  E1 = Exp1;
        E2 = Exp2;
        V = E1;
Label2: if (Exp4) then;
        else
          go to Next;
        statement_1
        .
        .
        .
        statement_n


Label1: if (Exp5) then
          go to Next;
        if V ≤ E2 then
          go to Next;
        V = V - 1;
          go to Label2;
Next: statement
```

参照が序数を表していれば、ステートメント `V = V - 1` は、`V = ordinalpred(V)` で置き換えられます。

## タイプ 4

タイプ 4 はループを指定します。 ループ とは、繰り返し実行される一連の命令を指します。

**LOOP**
無限の繰り返しを指定します。 FOREVER は、LOOP の同義語です。

**例**

```
  dcl Payroll file;
  dcl 1 Payrec,
       2 Type char,
       2 Subtype char,
       2 * char(100);

  Readfile:
    do loop;

      read file(Payroll) into(Payrec);

      If Payrec.type = 'E'
        then leave; /* like goto After_ReadFile */

      If Payrec.type = '1' then
        do;
             /* process first part of record */

          If Payrec.subtype = 'S'
            then iterate Readfile; /* like goto End_ReadFile */

             /* process remainder of record */
        end;

  End_ReadFile:
    end;
  After_ReadFile:;
```

このループを抜けるには、LEAVE ステートメントまたは GO TO ステートメントを使用するか、プロシージャーまたはプログラムを終了する以外に方法はありません。

## 基本的な繰り返しの例

以下の例は、DO グループを使用して反復実行を行う方法を示しています。

次の例では、DO グループは 10 回実行され、 参照 I は 1 から 10 までの範囲を値にとります。

```
do I = 1 to 10;
.
.
.
end;
```

DO および END ステートメントを使用すると、 次のように書く場合と同等の結果になります。

```
   I = 1;
A: if I > 10 then go to B;
   .
   .
   .
   I = I +1;
   go to A;
B: next statement
```

以下の DO ステートメントは DO グループを、3 回実行します。Name に 'Tom'、'Dick'、および 'Harry' をそれぞれ割り当てる度に 1 回です。

```
do Name = 'Tom', 'Dick', 'Harry';
```

以下のステートメントは、DO グループを 13 回実行することを指定します。 I の値を 1 から 10 に等しくして 10 回、また I の 値を 13 から 15 に等しくして 3 回です。

```
do I = 1 to 10, 13 to 15;
```

## 参照を添え字として使用した反復

DO ステートメントの参照は、DO グループ内のステートメントで添え字として 使用することができます。そのため、各実行では、表または配列の連続する エレメントを処理します。

次の例では、 A の最初の 10 個のエレメントは、順次 1 から 10 にセットされます。

```
do I = 1 to 10;
   A(I) = I;
end;
```

## TO および BY を使用した反復

以下の例は、DO グループでの TO オプションおよび BY オプションの使用を示しています。

### 例 1

次の例では、DO グループを 5 回実行することを指定しています。 I の値は、2、4、6、8、10 となります。

```
do I = 2 to 10 by 2;
```

### 例 2

参照の値を減らしていく場合は、BY オプションを使用しなければなりません。 例えば、次のステートメントは、I が 10、8、6、4、2、0、-2 の値で実行されます。

```
do I = 10 to -2 by -2;
```

### 例 3

次の例では、I が 1、3、5 値で DO グループが実行されます。

```
I=2;
do I=1 to I+3 by I;
.
.
.
end;
```

これは次のステートメントと同等です。

```
do I=1 to 5 by 2;
.
.
.
end;
```

## WHILE、UNTIL を指定した DO ステートメントの例

WHILE オプションや UNTIL オプションを使用すると、指定の条件に従って、 DO グループを連続的に実行することができます。

次の例を考えてください。

```
do while (A=B);
.
.
.
end;
```

この例は以下のステートメントと同等です。

```
S:  if A=B then;
    else goto R;
         .
         .
         .
        goto S;
R:  next statement
```

次の例を考えてください。

```
do until (A=B);
.
.
.
end;
```

この例は以下のステートメントと同等です。

```
S:
  .
  .
  .
  if (A=B) then goto R;
     goto S;
R:  next statement
```

ほかのオプションが付いていない限り、DO UNTIL ステートメントで始まる DO グループは最低 1 回は実行されますが、DO WHILE ステートメントで始まる DO グループは 1 回も実行されないことがあります。 つまり、ステートメントの DO WHILE (A=B) と DO UNTIL (A¬=B) は、同じではないということです。

次の例では、DO ステートメントに最初に制御が 到達したときに A¬=B であれば、 DO グループは 1 回も実行されません。

```
do while(A=B) until(X=10);
```

ただし、A=B であれば、DO グループは実行されます。 DO グループの実行後、X=10 であれば、それ以上実行されません。 X=10 でない場合は、A がまだ B に等しい限り、さらに実行されます。

次の例では、DO グループは最低 1 回は実行され、 そのとき I は 1 です。

```
do I=1 to 10 until(Y=1);
```

DO グループの実行後、Y=1 であれば、それ以上実行されません。 Y=1 でない場合は、デフォルトの増分値 (BY 1) が I に 加算され、I の新しい値が 10 と比較されます。 I が 10 より大きい場合、これ以上の実行は行われません。 それ以外の場合は、新しい実行が開始されます。

次のステートメントは、C(I) がゼロより小さければ DO グループを 10 回実行し、次に A が B に等しい場合にのみ、 もう 1 回実行することを指定しています。

```
do I = 1 to 10 while (C(I)<0),
       11 while (A = B);
```

## UPTHRU と DOWNTHRU を指定した DO の例

UPTHRU オプションおよび DOWNTHRU オプションを使用すると、終了値に従って DO グループが連続的に実行されます。

次の例では DO グループを 5 回実行すると、ループの終わりに達して *i* の値が 5 になります。

```
do i = 1 upthru 5;
.
.
.
end;
```

UPTHRU オプションを使用すると、参照は、更新前に終了値と比較されます。 これは、終了値以降に何も値がない場合に非常に役立ちます。 例えば、FIXEDOVERFLOW 条件は、次に示すようなループでは発生しません。

```
do i = 2147483641 upthru 2147483647;
.
.
.
end;
```

同様に、次のループは、符号なしの値をゼロまで減分していく際に 生じる可能性がある問題を予防します。

```
dcl U unsigned fixed bin;
do U = 17 downthru 0;
.
.
.
end;
```

UPTHRU と DOWNTHRU は、序数とともに使用すると特に有効です。 次の例を考えてください。

```
define ordinal Color ( Red value (1),
                       Orange,
                       Yellow,
                       Green,
                       Blue,
                       Indigo,
                       Violet);
dcl C ordinal Color;


do C = Red upthru Violet;
.
.
.
end;


do C = Violet downthru Red;
.
```

```
.
.
end;
```

最初のループでは、*c* は、red から violet まで 昇順で連続する各色を想定しています。 2 番目のループでは、*c* は、violet から red まで降順で 連続する各色を想定しています。

## REPEAT の例

この例は、DO グループでの REPEAT オプションの使用を示しています。

次の例では、DO グループは I は 1、2、4、8、16 などの値で実行されます。

```
do I = 1 repeat 2*I;
.
.
.
end;
```

上の例は以下のステートメントと同等です。

```
  I=1;
A:
  .
  .
  .
  I=2*I;
  goto A;
```

次の例では、DO グループの最初の実行は I=1 で行われます。

```
do I=1 repeat 2*I until(I=256);
```

この実行のあと、およびそれ以後 DO グループが実行されるたびに、UNTIL 式がテストされます。 I=256 であれば、それ以上実行されません。 その他の場合、REPEAT 式は評価されて I に代入され、新たな実行が開始されます。

次の例では、チェーニングしたリストで特定の項目を探索するために 使用される DO ステートメントを示します。

```
do P=Phead repeat P -> Fwd
     while(P¬=sysnull())
     until(P->Id=Id_to_be_found);
end;
```

DO グループの最初の実行で、 値 Phead が P に割り当てられます。 それ以降 DO グループが実行されるたびに、その実行前に値 P -> Fwd が P に割り当てられます。 DO グループの最初の実行前および後続のそれぞれの実行前に、 P の値がテストされ、 その値が NULL であれば、それ以上実行されません。

以下のステートメントは、 DO グループを 9 回実行することを指定します。 I の値を 1 から 9 に等しくして 9 回です。 その後、引き続き I の値を 10、20、 40、などに等しくします。 10000 より大きい I の値で DO グループが実行されたあと、 実行の繰り返しが終わります。

```
do I = 1 to 9, 10 repeat 2*I
               until (I>10000);
.
.
.
end;
```

# END ステートメント

END ステートメントは、1 つまたは複数のブロックやグループを終了させます。 それぞれのブロックごと、またはグループごとに、 END ステートメントがなければなりません。

**statement-label**
添え字を付けることはできません。 END のあとにステートメント・ラベル (statement-label) を指定すると、 そのステートメント・ラベルを持ち、DO、 SELECT、PACKAGE、BEGIN、 または PROCEDURE ステートメントで始まっていて、まだ閉じられていない グループまたはブロックのうち、その END ステートメントより前で、 最も近くにあるものを閉じます。 DO、SELECT、 PACKAGE、BEGIN、または PROCEDURE ステートメントを持つ すべてのブロックには、対応する END ステートメントがなければなりません。

END のあとにステートメント・ラベルがない場合は、その END ステートメントより前で、 最も近くにある DO、SELECT、PACKAGE、BEGIN、または PROCEDURE ステートメントで 始まっているグループまたはブロックのうち、対応する END ステートメントが ほかにないものが閉じられます。

ブロックの END ステートメントに制御が到達すると、そのブロックの実行は 終了します。 ただし、各ブロックには END ステートメントがなければなりませんが、ブロックの実行を終了させる方法はほかにもあります。 (詳細については、92 ページの『プロシージャー』 および 109 ページの『開始ブロック』 を 参照してください。)

プロシージャーの END ステートメントに制御が到達したときは、 その END ステートメントは RETURN ステートメントと見なされます。

主プロシージャーの END ステートメントに制御が到達したときは、プログラムが正常終了します。

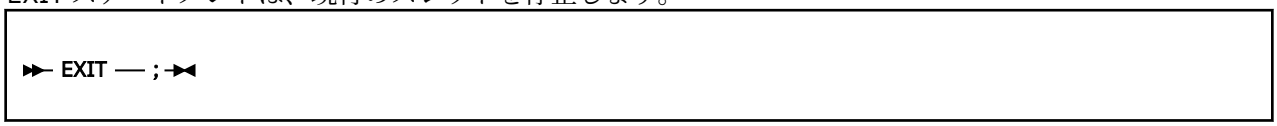
## ENTRY ステートメント

ENTRY ステートメントは、プロシージャーの 2 次エントリー・ポイントを指定します。

ENTRY ステートメントの詳細については、94 ページの『ENTRY ステートメント』を参照してください。

## EXIT ステートメント

EXIT ステートメントは、現行のスレッドを停止します。




## FETCH ステートメント

FETCH ステートメントは、指定されたプロシージャーが主記憶域に存在しているかを調べます。

FETCH ステートメントの詳細については、101 ページの『FETCH ステートメント』を参照してください。

## FLUSH ステートメント

FLUSH ステートメントはオープンしている 1 つの出力ファイルまたはオープンしているすべての出力ファイルに関連した バッファーをフラッシュします。

FLUSH ステートメントの詳細については、273 ページの『第 10 章 入出力』 の 285 ページの『FLUSH ステートメント』を参照してください。

## FORMAT ステートメント

FORMAT ステートメントは、伝送されるデータのフォーマットを制御するために、 編集ディレクティブ・データ伝送ステートメントで使用することのできるフォーマット・リストを指定する ものです。

FORMAT ステートメントの詳細については、295 ページの『第 12 章 ストリーム指向データ伝送』の 309 ページの『FORMAT ステートメント』を参照してください。

## FREE ステートメント

FREE ステートメントは、基底付き変数や被制御変数に割り振られている ストレージを解放します。

FREE ステートメントの詳細については、235 ページの『第 9 章 ストレージ制御』の 240 ページの『被制御変数のための FREE ステートメント』および 249 ページの『基底付き変数のための FREE ステートメント』を参照してください。

## GET ステートメント

GET ステートメントは、データ値をデータ・セットまたはストリングから 1 つ以上の変数に割り当てることができる STREAM 入力データ伝送ステートメントです。

GET ステートメントの詳細については、295 ページの『第 12 章 ストリーム指向データ伝送』の 295 ページの『GET ステートメント』を参照してください。

## GO TO ステートメント

GO TO ステートメントは、指定されたラベル参照によって識別されるステートメントに制御を移します。GO TO ステートメントは無条件分岐です。

**省略形**: GOTO

**label**
ラベル定数、ラベル変数、またはラベル値を返す関数参照を指定します。 ラベル変数の場合、GO TO ステートメントの実行のたびに異なる値に なっていることがあるので、常に同じステートメントに制御が移されるとは 限りません。

GO TO ステートメントによって、あるブロック内からそのブロックに 含まれていない場所に制御が移されると、そのブロックは終了します。 制御の移動先が、終了するブロックを直接に活動化したのではないブロック内にあると、活動化に際して介在したすべてのブロックも終了します (98 ページの『プロシージャーの終了』を参照してください)。

GO TO ステートメントが、複数回活動化を行うブロックに含まれるラベル定数を指定する場合には、その GO TO ステートメントの実行時に活動中のものに制御が移されます (99 ページの『再帰的プロシージャー』を参照してください)。

GO TO ステートメントは、以下のように制御権を移動することはできません。

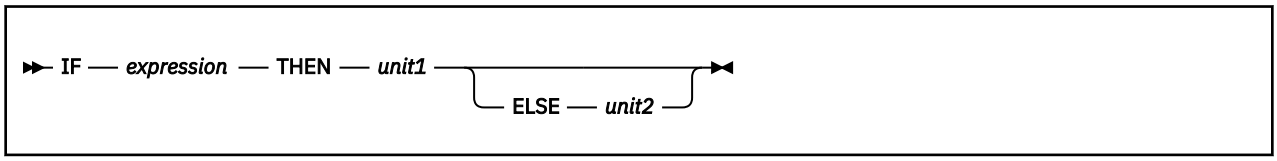
- 非アクティブ・ブロックへ。 このようなエラーが検出されるという保証はありません。
- DO グループの外から、タイプ 2 またはタイプ 3 の DO グループ内の ステートメントへ。ただし、DO グループ内から呼び出されたプロシージャー または ON ユニットを GO TO で終了させる場合は別です。
- FORMAT ステートメントへ。

GO TO の行先をラベル変数で指定し、ラベル定数をそのラベル変数に 割り当てれば、スイッチとして使用することができます。 また、そのラベル変数に添え字を付けておけば、添え字を変化させることによってスイッチを制御することができます。 ラベル変数または関数参照を使用すれば、きわめて複雑なスイッチ機能を実行することができます。 ただし、普通は、制御ステートメントは単純である方が効率はよくなります。 あるブロックから別のブロックへの GOTO 操作によって、ターゲット・ブロックでの 多くの最適化が妨げられます。ただし、ターゲット・ラベルがそのブロック内の最後の ステートメントである場合は例外です。

# IF ステートメント

IF ステートメントは、式を計算し、計算結果に応じて実行の流れを制御します。 したがって、IF ステートメントは、条件分岐です。

**注 :** 条件接頭語は、ELSE ステートメントでは使用することはできません。




**expression**
RULES(LAXIF) が使用されない場合は、式で属性 BIT(1) NONVARYING を使用する必要があります。

式で & 演算子または | 演算子を使用する場合は、68 ページの『各種演算の組み合わせ』で説明されているように評価されます。

**unit**
有効な単一のステートメント、グループ、または開始ブロックのいずれかです。 DECLARE、DEFAULT、END、FORMAT、PROCEDURE、 または %statement を除いて、 すべての単一ステートメントは有効と見なされ、実行可能です。 非実行ステートメントを指定すると、結果は予想できません。 それぞれのユニットには、制御の移動を指定する ステートメント (例えば、GO TO) が含まれていてもかまいません。 そのようなステートメントを使用すれば、IF ステートメントの 正規の順序を無効にすることができます。

それぞれのユニットには、ラベルを付けることも、条件接頭語を付けることもできます。

IF は複合ステートメントです。 最後のユニットを終了させるセミコロンは、IF ステートメントも終了させます。

ストリング式のいずれかのビットが '1'B であれば、*unit1* が実行され、*unit2* は存在していても無視されます。 すべてのビットが '0'B であるか、ストリングがヌルであれば、*unit1* は無視されて *unit2* が (存在していれば) 実行されます。

IF ステートメントはネストさせることができます。 つまり、一方または両方のユニットが IF ステートメントであってもかまいません。 それぞれの ELSE は、同じブロックまたは DO グループに含まれている IF ステートメントのうち、最も内側にあって、しかも対応する ELSE を持っていない IF に関連付けられます。 したがって、所要の制御順序を指定するために、ヌル・ステートメントを持つ ELSE が必要になることがあります。 次の例を考えてください。

```
if A = B then
     .
     .
     .
     else
       if A = C then
       .
       .
       .
       else
       .
       .
       .
```

B および C が定数である場合、この例は以下のステートメントと同等であり、以下のほうがより良いコードであると考えられます。

```
select( A );
  when ( B )
    .
    .
    .
  when ( C )
    .
    .
```

```
  .
  .
  .
end;
```

## 例

次の例では、比較の結果が真 (A が B に等しい) であれば、 D の値が C に割り当てられ、ELSE ユニットは実行されません。

```
if A = B then
  C = D;
else
  C = E;
```

比較の結果が偽 (A が B に等しくない) であれば、 THEN ユニットは実行されず、E の値が C に割り当てられます。

THEN ユニットと ELSE ユニットのどちらにでも、制御権を移動する (条件付きで、または無条件で) ステートメントを書くことができます。 THEN ユニットが GO TO ステートメントで終わる場合、ELSE ユニットを指定する必要はありません。 次の例を考えてください。

```
if all(Array1 = Array2) then
  go to LABEL_1;
next-statement
```

この式が真であれば、THEN ユニットの GO TO ステートメントによって、 LABEL_1 に制御が移されます。この式が偽であれば、THEN ユニットは実行されず、次のステートメントに制御が移されます。

## 短絡計算

次の場合に、IF 式のテストが短絡化されます。

- IF 式が 2 つの式の論理和で構成されているときに、最初の式が真の場合、2 番目の式は計算されず、コードは THEN 節を実行します。
- IF 式が 2 つの式の論理積で構成されているときに、最初の式が偽の場合、2 番目の式は計算されず、コードは ELSE 節を実行します。

ただし、コードの短絡化は以下の式についてのみ行われます。

- 比較式
- BIT(1) リテラル
- NONVARYING BIT(1) 変数
- NONVARYING BIT(1) を返す ENTRY 参照
- 3 つの引数を持ち、最後の引数が 1 と等しい REAL FIXED リテラルである、SUBSTR 組み込み関数参照
- 2 つの配列に適用される比較演算子か、単に NONVARYING BIT(1) の配列である変数のいずれかの引数を持つ、ALL または ANY 組み込み関数参照
- 次の組み込み関数のいずれかへの参照:
  - CHECKSTG
  - ENDFILE
  - FILEOPEN
  - ISFINITE
  - ISINF
  - ISMAIN
  - ISNAN
  - ISNORMAL
  - ISZERO
  - OMITTED

- PRESENT
- UNALLOCATED
- VALID
- VALIDDATE

必然的に、上記および NOT 接頭演算子および AND または OR の 2 項演算子で構成される式 (再帰的な 可能性のある) も短絡化されます。

**例**

この例は以下の宣言に基づいています。

```
dcl A   bit(1);
dcl B   bit(1);
dcl C   bit(2);
dcl D   bit(2);
dcl P   pointer;
dcl BX  based fixed bin(31);
```

これらの宣言の場合、以下の IF ステートメントはすべて短絡化されます。

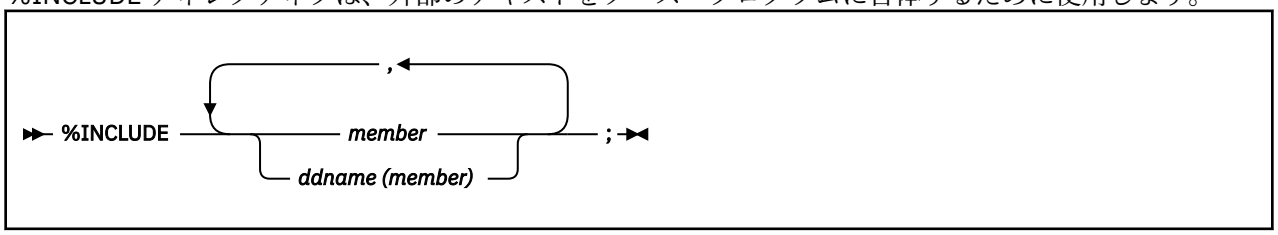
```
if A | B then
if P = sysnull() | P->BX = 0 then
if C = ''b & D = ''b then
if A | (substr(C,1,1) & substr(D,2,1)) then
```

しかし、次の IF 文は短絡化されません。

```
if C | D then
if C & D then
```

# %INCLUDE ディレクティブ

%INCLUDE ディレクティブは、外部のテキストをソース・プログラムに合体するために使用します。




組み込みメンバーには、絶対ファイル名を指定してもかまいません。 絶対ファイル名を単一引用符または二重引用符で囲みます。 例えば、以下のステートメントは有効です。

**INTEL**

```
%include "\ibmpli\include\sqlcodes.inc"
```

**AIX および z/OS UNIX**

```
%include "/ibmpli/include/sqlcodes.inc"
```

# ITERATE ステートメント

ITERATE ステートメントは、それに含まれる反復 DO グループを区切る END ステートメントに制御を移します。 現在行われている反復は完了され、次の反復が (必要であれば) 開始されます。

ITERATE ステートメントは、反復 DO グループが 非反復 DO グループを含んでいる場合には、 その非反復 DO グループ内で指定することができます。

***label-constant***
このステートメントを含んでいる DO グループのラベルでなければなりません。 *label-constant* を省略すると、ITERATE ステートメントを含む最新の反復 DO グループの END ステートメントに制御が移ります。

例えば、214 ページの『タイプ 4』を参照してください。

# LEAVE ステートメント

単純な DO グループに含まれているか、 または単純な DO グループを指定してある場合に、 LEAVE ステートメントは、そのグループから制御を離します。 反復 DO グループに含まれているか、 またはそのようなグループを指定してある場合には、 LEAVE ステートメントは、現在の繰り返しも含め、 そのグループのすべての繰り返しを終了します。

制御の流れは、DO グループが END ステートメントに達して 終了した場合に通常移動するはずの位置に移動します。 この位置は、必ずしも DO グループの END ステートメントに続くステートメントではありません (224 ページの『例』を参照)。

***label-constant***
このステートメントを含んでいる DO グループのラベルでなければなりません。 ここで指定するラベルを持つ DO グループが終了します。 *label-constant* を省略した場合、終了する DO グループは LEAVE ステートメントを含むグループです。

LEAVE ステートメントおよび参照される (または暗黙に指定される) DO ステートメントが、別々のブロックにあってはなりません。

### 例

次の例では、 ステートメント `leave A;` は、 `C` に制御を移します。

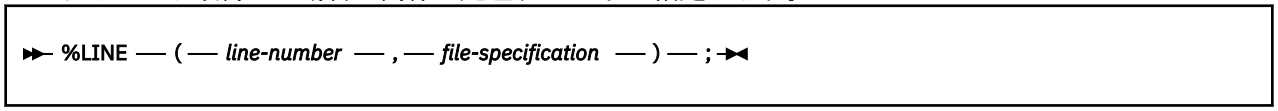
```
If Time_to_process_X then

A:  do I = lbound(X,1) to hbound(X,1);
       do J = lbound(X,2) to hbound(X,2);
          If X(I,J)=0 then
             leave A;          /* control goes to C, not B */
          else
             do;
             .
             .
             .
             end;
       end;
    end;


  Else

B:  do;
    .
    .
    .
    end;
```

```
C:  statement after group A;
```

他の例については、214 ページの『タイプ 4』を参照してください。

## %LINE ディレクティブ

%LINE ディレクティブは、メッセージおよびデバッグ用に生成される情報 の中で、その次の行を指定の行とファイルから取得した場合と同様に処理する ように指定します。




ディレクティブが認識されるためには、文字 %LINE が 入力行の列 1 から 5 までにあることが必要です (また逆に、 これらの 5 文字で始まる行はすべて %LINE ディレクティブとして扱われます)。 *line-number* は 7 桁以下の整数値であることが必要で、*file-specification* は引用符で囲んではなりません。 セミコロンの後に指定された文字はすべて無視されます。

%LINE ディレクティブは、LINEDIR コンパイラー・オプションが有効でない限り無効です。

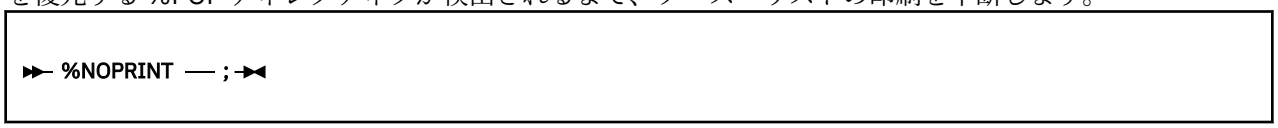
## LOCATE ステートメント

LOCATE ステートメントは、基底付き変数用に出力バッファー内のストレージを割り振り、次のレコードの位置を指すポインターを設定します。 LOCATE ステートメントは、OUTPUT SEQUENTIAL BUFFERED ファイルを位置指定モードで処理する場合にのみ使用することができます。

LOCATE ステートメントの詳細については、287 ページの『第 11 章 レコード単位データ伝送』の 289 ページの『LOCATE ステートメント』を参照してください。

## %NOPRINT ディレクティブ

%NOPRINT ディレクティブは、%PRINT ディレクティブが検出されるか、前の %PRINT ディレクティブを復元する %POP ディレクティブが検出されるまで、ソース・リストの印刷を中断します。




%NOPRINT ディレクティブの例については、228 ページの『%PUSH ディレクティブ』を参照してください。

## %NOTE ディレクティブ

%NOTE ディレクティブは、指定のテキストおよび重大度の診断メッセージを生成します。

***message***
診断メッセージを必要とする値を持つ文字式。

**コード**
メッセージの重大度を表す値が入っている固定式。重大度は次のとおりです。

| コード | 重大度 |
|---|---|
| 0 | I |
| 4 | W |
| 8 | E |

| コード | 重大度 |
|---|---|
| 12 | S |
| 16 | U |

*code* を省略すると、デフォルトとして 0 が設定されます。

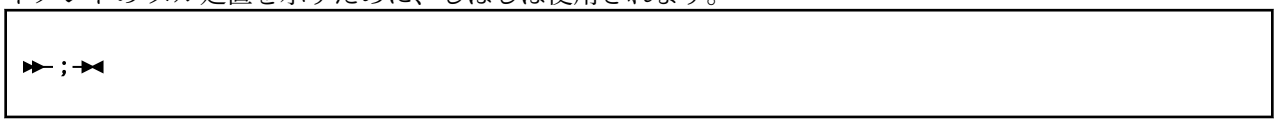
*code* が前述のリストに示した値以外の値を持つ場合は、診断メッセージが生成されます。 この場合、その結果としてのシステム処置は未定義です。

生成されたメッセージは、ほかのメッセージとともにファイルにされます。 特定のメッセージを続いて印刷するかどうかは、その重大度レベルと コンパイラー FLAG オプション (「プログラミング・ガイド」に説明されている) の設定に依存します。

重大度 U のメッセージが生成されると、プリプロセスとコンパイル処理が直ちに終了します。 重大度 S、E、または W のメッセージが生成されたときにコンパイル処理が終了するかどうかは、コンパイラー・オプションの設定値によってそれぞれ異なります。

## ヌル・ステートメント (null statement)

ヌル・ステートメントは、何も行わず、 ステートメントの実行順序を変更することもしません。 ヌル・ステートメントは、THEN 文節および ELSE 文節 ならびに WHEN ステートメントおよび OTHERWISE ステートメントの ヌル処置を示すために、しばしば使用されます。



## ON ステートメント

ON ステートメントは、割り込み可能条件がこれ以降に、設定された条件の有効範囲内で発生した場合に実行するアクションを設定します。

ON ステートメントの詳細については、339 ページの『第 15 章 条件処理』の 341 ページの『ON ステートメント』を参照してください。

## OPEN ステートメント

OPEN ステートメントは、ファイルをデータ・セットに関連づけます。

OPEN ステートメントの詳細については、273 ページの『第 10 章 入出力』の 281 ページの『OPEN ステートメント』を参照してください。

## OTHERWISE ステートメント

OTHERWISE ステートメントは、SELECT グループで、前の WHEN ステートメントのすべてのテストが失敗した場合に実行するユニットを指定します。

OTHERWISE ステートメントの詳細については、231 ページの『SELECT ステートメント』を参照してください。

## PACKAGE ステートメント

PACKAGE ステートメントはパッケージを定義します。 名前が再び宣言されない限りは、パッケージは、パッケージ内に含まれる すべての宣言およびプロシージャーによって共用される名前の有効範囲を構成します。

PACKAGE ステートメントの詳細については、87 ページの『第 5 章 プログラムの編成』の 90 ページの『パッケージ』を参照してください。

## %PAGE ディレクティブ

%PAGE ディレクティブを使用すれば、コンパイラーのソース・リストを 新しいページから開始できます。

## %POP ディレクティブ

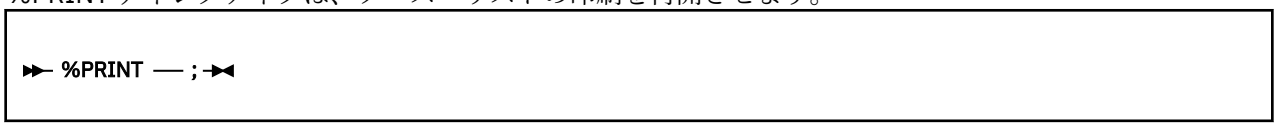
%POP ディレクティブを使用して、前回の %PUSH ディレクティブで保存された %PRINT および %NOPRINT ディレクティブの状況を復元することができます。

%PUSH ディレクティブと %POP ディレクティブの最も一般的な用途は、 インクルード・ファイルやマクロにあります。

例えば、228 ページの『%PUSH ディレクティブ』を参照してください。

## %PRINT ディレクティブ

%PRINT ディレクティブは、ソース・リストの印刷を再開させます。




関係のあるコンパイラー・オプションが指定されていれば、%PRINT が有効になります。 %PRINT ディレクティブの例については、228 ページの『%PUSH ディレクティブ』を参照してください。

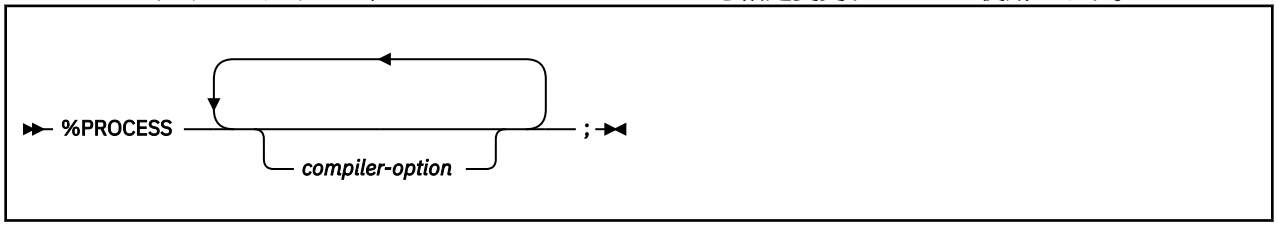
## PROCEDURE ステートメント

PROCEDURE ステートメントは、 プロシージャーを主プロシージャー、サブルーチン、 または関数として識別します。 プロシージャーおよびその他の特性によって予期されるパラメーターも、 プロシージャー・ステートメント上で指定することができます。

PROCEDURE ステートメントの詳細については、87 ページの『第 5 章 プログラムの編成』の 93 ページの『PROCEDURE ステートメント』を参照してください。

## %PROCESS ディレクティブ

%PROCESS ディレクティブは、コンパイラー・オプションを指定変更するために使用します。




% または * が、 ソース・レコードの最初のデータ・バイトでなければなりません。 任意の数の %PROCESS、*PROCESS、%PROCINC、および *PROCINC ディレクティブを指定できますが、それらはすべて最初の言語要素が現れる前になければなりません。

詳しくは、「プログラミング・ガイド」を参照してください。

## *PROCESS ディレクティブ

*PROCESS ディレクティブは、%PROCESS ディレクティブの同義語です。

**関連情報**

227 ページの『%PROCESS ディレクティブ』

%PROCESS ディレクティブは、コンパイラー・オプションを指定変更するために使用します。

## %PROCINC ディレクティブ

%PROCINC ディレクティブは、コンパイラー・オプションをオーバーライドするために使用されます。

% または * が、 ソース・レコードの最初のデータ・バイトでなければなりません。 任意の数の %PROCESS、*PROCESS、%PROCINC、および *PROCINC ディレクティブを指定できますが、それらはすべて最初の言語要素が現れる前になければなりません。

%PROCESS ディレクティブとは異なり、%PROCINC ディレクティブはコンパイラー・オプションを指定しません。 代わりに、%PROCESS、*PROCESS、%PROCINC、および *PROCINC ディレクティブを含むファイルを指定します。

詳しくは、「プログラミング・ガイド」を参照してください。

## *PROCINC ディレクティブ

*PROCINC ディレクティブは、%PROCINC ディレクティブのシノニムです。

**関連情報**

228 ページの『%PROCINC ディレクティブ』

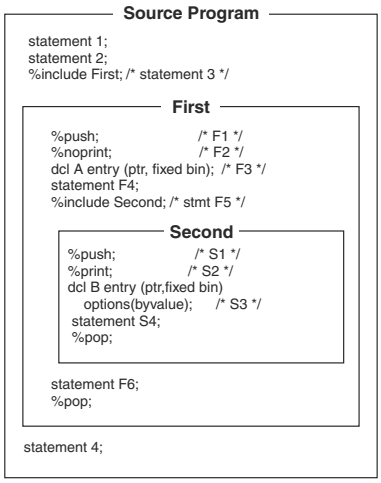
%PROCINC ディレクティブは、コンパイラー・オプションをオーバーライドするために使用されます。

## %PUSH ディレクティブ

%PUSH ディレクティブを使用して、%PRINT および %NOPRINT ディレクティブの現行状況を後入れ先出し方式で「プッシュダウン」スタックに保存できます。 %POP ディレクティブを使用すれば、後ほどこの保管した状況を同じく 後入れ先出し方式で復元できます。

%PUSH ディレクティブと %POP ディレクティブは、一般的にインクルード・ファイルやマクロで使用されます。

次の例では、ステートメント 1、2、3、S3、S4、および 4 がリストに印刷されます。 その他はすべて印刷されません。




Second 内の %POP ディレクティブの次に元の設定値が復元されます。

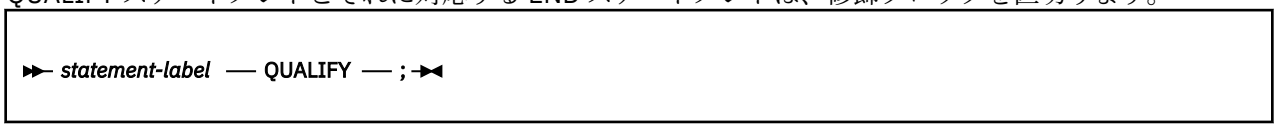
## PUT ステートメント

PUT ステートメントは、値をストリーム出力ファイルに伝送するか、値を文字変数に割り当てることができる、STREAM 出力データ伝送ステートメントです。

PUT ステートメントの詳細については、295 ページの『第 12 章 ストリーム指向データ伝送』の 296 ページの『PUT ステートメント』を参照してください。

## QUALIFY ステートメント

QUALIFY ステートメントとそれに対応する END ステートメントは、修飾ブロックを区切ります。




**statement-label**
タイプと値の宣言を修飾するために使用できる名前を指定します。 QUALIFY ステートメントは 1 つ (のみ) のラベルを持つ必要があります。

修飾ブロックに含めることができるのは DECLARE、DEFINE、および QUALIFY ステートメントのみで、その中の有効な DECLARE ステートメントのみが、VALUE 属性によりスカラーを指定している必要があります。

例えば、次のようなステートメントがあるとします。

```
paint: qualify;
define ordinal color ( red, orange, yellow, green, blue, indigo, violet );
end paint;
```

これらは、PAINT を ORDINAL タイプ COLOR に対する修飾子として定義し、また、値 RED、ORANGE などに対する修飾子として定義しています。

修飾ブロック内のそれぞれの名前は、包含ブロックに対してではなく、そのブロックに対して固有でなければなりません。 つまり、変数をタイプ PAINT.COLOR を持つ変数として宣言して、定数 PAINT.RED、PAINT.ORANGE などを参照できるということです。 修飾ブロックの名前は、そのブロックに対して固有でなければなりません。

また、修飾ブロックをネストすることもできます。 例えば、上のブロック内の修飾ブロックを次のようにネストできます。

```
paint: qualify;
define ordinal color ( red, orange, yellow );
depth: qualify;
define ordinal intensity ( high, medium, low );
```

```
    end depth;
    end paint;
```

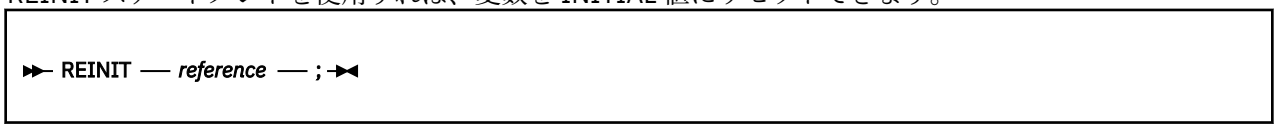
## READ ステートメント

READ ステートメントによって、レコードがデータ・セットからプログラム変数へ伝送されるか、 またはストレージ内のレコードを指すポインターがセットされます。

READ ステートメントの詳細については、287 ページの『第 11 章 レコード単位データ伝送』の 288 ページの『READ ステートメント』を参照してください。

## REINIT ステートメント

REINIT ステートメントを使用すれば、変数を INITIAL 値にリセットできます。




**リファレンス**

*reference* は、添え字なしでなければなりません (BASED の場合はロケーター修飾が可能)。 また、*reference* は AUTOMATIC、CONTROLLED、BASED、または STATIC のいずれかでなければなりません。*reference* は、スカラー、配列、または構造体または DEFINE STRUCTURE タイプ (あるいはこれらのいずれかの配列) にすることができます。

## RELEASE ステートメント

RELEASE ステートメントは、指定した entry constant (入り口定数) が識別する プロシージャーによって占められた主記憶域を解放します。

RELEASE ステートメントの詳細については、100 ページの『外部プロシージャーの動的ロード』の 101 ページの『RELEASE ステートメント』を参照してください。

## RESIGNAL ステートメント

RESIGNAL ステートメントによって、現行の ON ユニットは終了して、別の ON ユニットが同一の条件で制御を獲得できるようになります。

RESIGNAL ステートメントの詳細については、339 ページの『第 15 章 条件処理』の 345 ページの『RESIGNAL ステートメント』を参照してください。

## RETURN ステートメント

RETURN ステートメントは、RETURN ステートメントが入っている サブルーチンまたは関数プロシージャーの実行を終了し、 呼び出しプロシージャーに制御を戻します。

詳しくは、『122 ページの『RETURN ステートメント』』を参照してください。

## REVERT ステートメント

REVERT ステートメントは、指定の条件に対して当該ブロック内で実行された ON ユニットを取り消します。

REVERT ステートメントの詳細については、339 ページの『第 15 章 条件処理』の 344 ページの『REVERT ステートメント』を参照してください。

## REWRITE ステートメント

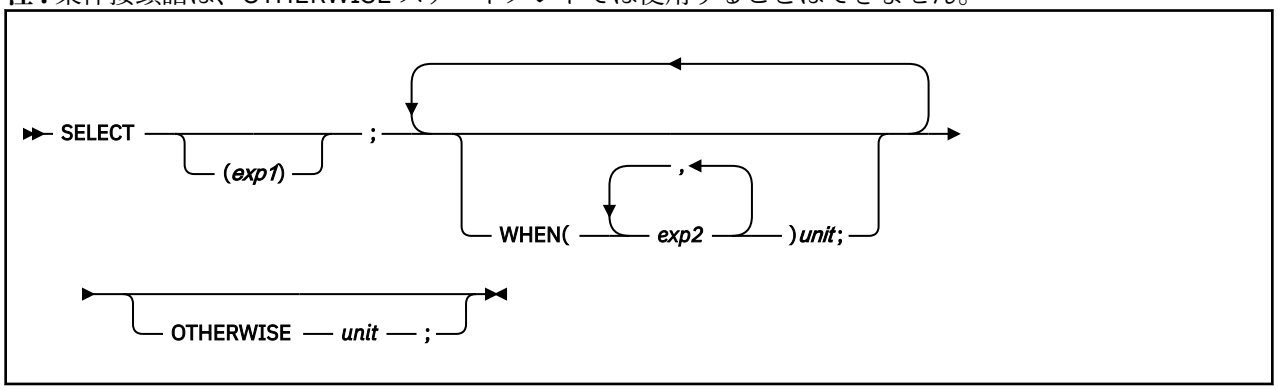
REWRITE ステートメントは、UPDATE ファイルのレコードを置き換えます。

REWRITE ステートメントの詳細については、287 ページの『第 11 章 レコード単位データ伝送』の 289 ページの『REWRITE ステートメント』を参照してください。

## SELECT ステートメント

SELECT グループを使用すれば、複数経路の条件付き分岐を行うことができます。 SELECT グループは、SELECT ステートメント、オプションの WHEN ステートメント (1 つまたは複数)、オプションの OTHERWISE ステートメント、および END ステートメントで構成されます。

**注 :** 条件接頭語は、OTHERWISE ステートメントでは使用することはできません。




省略形: OTHERWISE の場合は OTHER

**SELECT (exp1)**
SELECT ステートメントとこれに対応した END ステートメントが、SELECT グループと集合的に呼ばれるステートメントのグループを区切ります。 SELECT ステートメント内の式が計算され、その値が保管されます。

**WHEN (exp2, exp2, ...) UNIT**
1 つまたは複数の式を指定します。これらの式は計算されて、 SELECT ステートメントにより保管された値と比較されます。

保管されたものと同じ値を持つ式が見付かると、その時点で、 WHEN ステートメント内の式の計算は終了し、その WHEN ステートメントのユニットが実行されます。 保管されたものと同じ値を持つ式が見付からなかった場合は、 OTHERWISE ステートメントのユニットが実行されます。

WHEN ステートメントにラベルを付けることはできません。

**OTHERWISE unit**
先行の WHEN ステートメントのすべてのテストが失敗した場合に 実行されるユニットを指定します。

OTHERWISE ステートメントを省略し、グループ選択の実行によってユニットが選択されなかった場合は、ERROR 条件が発生します。

OTHERWISE ステートメントにラベルまたは条件接頭語を付けてはなりません。

**UNIT**
各ユニットは、有効な単一ステートメント、グループ、 または開始ブロックのいずれかです。 DECLARE、DEFAULT、END、ENTRY FORMAT、 PROCEDURE、および %statement ステートメントは無効です。 それぞれのユニットには、制御の移動を指定する ステートメント (例えば、GO TO) が含まれていてもかまいません。 そのようなステートメントを指定すれば、SELECT ステートメントの正規の順序を指定変更することができます。

*exp1* を省略した場合、それぞれの *exp2* が計算され、必要に応じてビット・ストリングに変換されます。 そのストリングのいずれかのビットが '1'B であれば、その WHEN ステートメントのユニットが実行されます。 すべてのビットが 0 であるか、そのストリングがヌルであれば、 OTHERWISE ステートメントのユニットが実行されます。

WHEN または OTHERWISE ステートメントのユニットが実行されたあとは、 そのユニット内で制御の通常の流れが変更されたのでない限り、 SELECT グループの次のステートメントに制御が渡されます。

*exp1* を指定した場合は、それぞれの *exp2* は、 次の比較式がスカラー・ビット値を持たねばなりません。

```
(exp1) = (exp2)
```

*exp1* と *exp2* は、両方ともスカラー式でなければなりません。 そのため、配列、構造体、および共用体は *exp1* または *exp2* で使用できますが、評価された式はスカラー値でなければなりません。

### 例

以下の例では、E, E1 などが式です。 SELECT ステートメントに制御が到達すると、 式 E が計算され、その値が保管されます。 次に、WHEN ステートメント内のそれぞれの式が 順に (書かれている順に) 計算され、その値が E の値と比較されます。

E の値と等しい値が見付かると、対応する THEN ステートメントの後に指定されている処置が行われます。先の WHEN ステートメント式は計算されません。

WHEN ステートメントのどの式も、SELECT ステートメントの式と等しくない場合は、 OTHERWISE ステートメントの後に指定されている処置が実行されます。

```
select (E);
  when (E1,E2,E3) action-1;
  when (E4,E5) action-2;
  otherwise action-n;
end;
Nl: next statement;
```

次の例では、*exp1* は省略されています。

```
select;
  when (A>B) call Bigger;
  when (A=B) call Same;
  otherwise call Smaller;
end;
```

SELECT グループ内に WHEN ステートメントが 1 つもない場合は、 OTHERWISE ステートメント内の処置が無条件に実行されます。 OTHERWISE ステートメントを省略し、グループ選択の実行によって WHEN ステートメントが選択されなかった場合は、ERROR 条件が発生します。

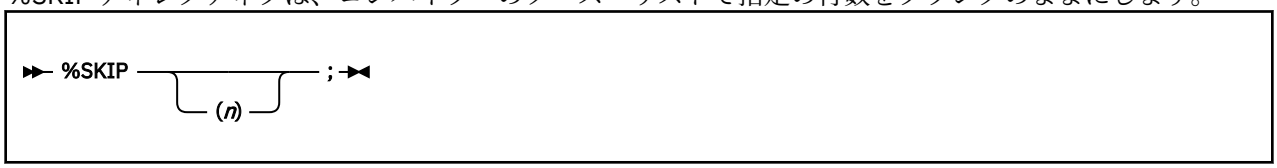
## SIGNAL ステートメント

SIGNAL ステートメントは、ON ユニットのアクションを検査し、正しいアクションが条件に関連付けられているかどうかを判別するために、プログラム・テストで使用します。

SIGNAL ステートメントの詳細については、339 ページの『第 15 章 条件処理』の 345 ページの『SIGNAL ステートメント』を参照してください。

## %SKIP ディレクティブ

%SKIP ディレクティブは、コンパイラーのソース・リストで指定の行数をブランクのままにします。




**n**

スキップしたい行数を指定します。 1 から 999 の範囲内の整数でなければなりません。 *n* を省略した場合、デフォルトは 1 です。 *n* がページに残っている行数より大きい場合は、%SKIP ディレクティブの代わりに %PAGE ディレクティブに相当するものが実行されます。

## STOP ステートメント

STOP ステートメントは、現在のアプリケーションを停止します。

## WAIT ステートメント

WAIT ステートメントは、指定したスレッドが終了するまでプロセスを中断することを指定します。

WAIT ステートメントの詳細については、367 ページの『第 17 章 マルチスレッド化機能』の 369 ページの『スレッドの完了の待機』を参照してください。

## WHEN ステートメント

WHEN ステートメントは、1 つまたは複数の式を指定します。これらの式は計算されて、SELECT ステートメントにより保管された値と比較されます。

WHEN ステートメントの詳細については、231 ページの『SELECT ステートメント』を参照してください。

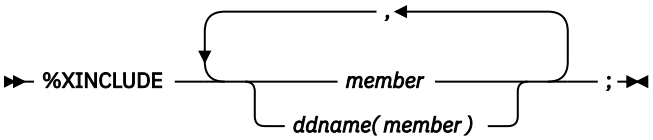
## WRITE ステートメント

WRITE ステートメントは、プログラムからレコードを伝送し、それを データ・セットに追加します。

WRITE ステートメントの詳細については、287 ページの『第 11 章 レコード単位データ伝送』の 288 ページの『WRITE ステートメント』を参照してください。

## %XINCLUDE ステートメント

%XINCLUDE ディレクティブは、 外部のテキストを、(以前に組み込まれていなければ) ソース・プログラム に組み入れるために使用します。




## XDECLARE ステートメント

XDECLARE ステートメントは、以下の点を除き、DECLARE ステートメントと同じです。

1. XDECLARE では、level-1 名の指定に * を使用してはなりません
2. XDECLARE ステートメントで指定された level-1 名が、すでに前の (X)DECLARE ステートメントで指定されされていた場合、2 つの指定での属性が一致しなければしなければなりません。

XDECLARE は XDCL と省略することができます。

DECLARE ステートメントの詳細については、148 ページの『DECLARE ステートメント』を参照してください。

## XDEFINE ALIAS ステートメント

XDEFINE ALIAS ステートメントは、 ユーザーが別名に割り当てたデータ・タイプ属性のセットのために、同義語として使用することができる名前を指定します。

XDEFINE ALIAS ステートメントは DEFINE ALIAS ステートメントと同じです。ただし、指定された名前が前の (X)DEFINE ALIAS ステートメントに既に出現している場合、この XDEFINE ステートメントは無視されます。

DEFINE ALIAS ステートメントの詳細については、135 ページの『DEFINE ALIAS ステートメント』および 135 ページの『第 6 章 タイプ定義』を参照してください。

## XDEFINE ORDINAL ステートメント

XDEFINE ORDINAL ステートメントは、順序付けられた (名前付きの) 値の セットを表すタイプを指定します。

XDEFINE ORDINAL ステートメントは DEFINE ORDINAL ステートメントと同じです。ただし、指定された名前が前の (X)DEFINE ORDINAL ステートメントに既に出現している場合、この XDEFINE ステートメントは無視されます。

DEFINE ORDINAL ステートメントの詳細については、136 ページの『DEFINE ORDINAL ステートメント』および 135 ページの『第 6 章 タイプ定義』を参照してください。

## XDEFINE STRUCTURE ステートメント

XDEFINE STRUCTURE ステートメントは名前付き構造体または 共用体タイプを指定します。

XDEFINE STRUCTURE ステートメントは DEFINE STRUCTURE ステートメントと同じです。ただし、指定された名前が前の (X)DEFINE STRUCTURE ステートメントに既に出現している場合、この XDEFINE ステートメントは無視されます。

DEFINE STRUCTURE ステートメントの詳細については、137 ページの『タイプ付き構造体と共用体の定義』および 135 ページの『第 6 章 タイプ定義』を参照してください。

## XPROCEDURE ステートメント

マクロ XPROCEDURE ステートメントは、マクロ PROCEDURE ステートメントとほぼ同じですが、XPROCEDURE ステートメントの左端の名前が、定義済みのプリプロセッサー・プロシージャーの名前になっている場合に、プリプロセッサーは、XPROCEDURE ステートメントの後続の出現箇所にフラグを立てずに、その箇所を無視します。

省略形: XPROC

PROCEDURE ステートメントについては、87 ページの『第 5 章 プログラムの編成』にある 93 ページの『PROCEDURE ステートメント』を参照してください。

# 第 9 章 ストレージ制御

変数は、すべてストレージを必要とします。 変数に指定する属性は、必要なストレージの大きさ、およびそのストレージをどのように解釈するかを表します。

次の例では、X への参照は、 すなわち、固定小数点 2 進数として解釈される値を含む ストレージの一部への参照です。

```
dcl X fixed binary(31,0) automatic;
```

X が自動であるため、 このストレージはその宣言ブロックが活動化されたときに割り振られ、 割り振られたストレージは、ブロックが非活動化されるまで有効です。

## ストレージのクラス、割り振り、および割り振り解除

ストレージの割り振りとは、変数で表されるデータ項目を内部で記録しておくために、ストレージの 1 つの区域をその変数に関連付ける作業のことです。 ある変数にストレージが関連付けられると、その変数は割り振られた ことになります。 ストレージ・クラスによって、ストレージの制御の程度、およびストレージの割り振り方法と解放方法が決まります。

変数の割り振りは、静的に (プログラムの実行前に) 行われることもあ れば、動的に (プログラムの実行中に) 行われることもあります。 静的に割り振られた変数は、アプリケーション・プログラムの実行中ずっと割り振られた ままになっています。 動的に割り振られた変数は、その変数を含んでいるブロックが終了するとき、 またはアプリケーションから明示的に要求されたときに、占有していたストレージを解放します。

変数に割り当てられたストレージ・クラスによって、その変数のストレージに適用されるストレージ制御の程度、およびその変数のストレージの割り振り方法と解放方法が決まります。 ストレージ・クラスは、明示的に、または暗黙に、もしくはコンテキスト宣言でその対応する属 性を使用することによって割り当てることができます。 以下の 4 つのストレージ・クラスがあります。

**AUTOMATIC**
ストレージ宣言を含むブロックに入るたびに、ストレージが割り振られることを指定します。 そのブロックから出る時点で、割り振られていたストレージが解放されます。

ブロックが、再帰的に呼び出されるプロシージャーの場合は、 直前に割り振られたストレージは、このプロシージャーの入り口でスタックに入れられます (プッシュダウン)。 ストレージの最新の割り振りは、各世代が終了した時に、再帰プロシージャー内でスタックから除去されます (ポップアップ)。 プッシュダウン・スタッキングおよびポップアップ・スタッキングについては、99 ページの『再帰的プロシージャー』を参照してください。

**STATIC**
プログラムのロード時にストレージを割り振ることを指定します。 プログラムの実行が完了するまで、そのストレージは解放されません。 取り出されたプロシージャーの場合は、そのプロシージャーが解放され るまで、そのストレージは解放されません。

**CONTROLLED**
ALLOCATE ステートメントおよび FREE ステートメントを使用して、ストレージの割り振りと解放を制御することを指定します。 同一プログラムで同一の被制御変数を複数回割り振り、しかもその間に解放処 理をしない場合は、その変数のその間の複数世代がスタックされます。 最新の世代を解放することによってのみ、初期の世代にアクセスすることができます。

**BASED**
CONTROLLED と同じように、ユーザーがストレージの割り振りと解放を制御することを指定します。 CONTROLLED と異なるのは、複数回割り振った場合に、割り振られたストレージはス タックされるのではなく、どれも常に使用可能である点です。 それぞれの割り振りは、ポインター変数の値で識別することができます。 また基底付き変数ストレージの区域と関連づけたり、オフセット変数の値によって 識別することができるという点でも異なります。

区域外にある基底付き変数は、ALLOCATE 組み込み関数によって割り振ることができ、PLIFREE 組み込みサブルーチンによって解放することができます。 基底付き変数は、AUTOMATIC 組み込み関数によっ

© Copyright IBM Corp. 1999, 2022

て割り振ることもできます。 このようにして割り振られた変数は、それらの変数が割り振られているブロックの終了時に、自動的に解放されます。

要素変数、配列変数、大構造変数、および共用体変数の場合は、ストレージ・クラス属 性を明示的に宣言することができます。 配列変数、大構造変数、および共用体変数に対して宣言されたストレージ・クラスは、 その配列、大構造体、または共用体のすべてのエレメントに適用されます。

ストレージ・クラス属性は、以下に対して指定することはできません。

- CONDITION 条件
- 定義済みデータ項目
- 入り口定数
- ファイル定数
- フォーマット定数
- DEFINE ステートメント内に定義された ID
- ラベル定数
- 構造体または共用体のメンバー
- 名前付き定数

変数へのストレージの割り振りは、PL/I によって管理されます。 ユーザーは、ストレージのどこに割り振りを行うかを指定することはできません。 ただし、変数が既存の AREA に割り振られるように指定することはできます。

**関連情報**

251 ページの『区域データとその属性』
区域変数は、基底付き変数を割り振るために予約される主記憶域の区域を 定義するものです。 ALLOCATE ステートメントや FREE ステートメントを使用すれば、この 予約されたストレージを基底付き変数に割り振ったり、基底付き変数から 解放したりすることができます。

# 静的ストレージとその属性

STATIC 属性で宣言された変数は、プログラムが実行を開始する前に割り振られます。 それらは、プログラムが終了するまで割り振られたままになっています。 静的変数の割り振りについての制御は、実行中のプログラムには与えられません。

STATIC は外部変数のデフォルトですが、内部変数が STATIC であってもかまいません。 またこれは、パッケージ内で宣言された変数のデフォルトでもあり、いずれのプロシージャーの範囲内でもありません。 静的変数を参照できるかどうかについては、有効範囲に関する通常の規則に従います。 以下の例では、変数 X がプログラムの存続期間中に割り振られますが、これはプロシージャー B、または B に含まれている任意のブロック内でのみ参照できます。 変数 Y は STATIC 属性を取得し、プログラムの存続期間中にも割り振られます。

```
Package: Package exports (*);
  dcl Y char(10);

  A: proc options(main);
    B: proc;
      declare X static internal;
    end B;
  end A;

  C: proc;
    Y = 'hello';
  end C;

end Package;
```

静的変数が INITIAL 属性によって初期化される場合、初期値は制限付き式でなければなりません。 またエクステントも、制限付き式として指定されなければなりません。

1 つの変数に 100 個を超える INITIAL 項目が指定され、その変数が STATIC 属性を使用して宣言された場合は、通常、コンパイラーおよびユーザー・コードのパフォーマンスが向上します。

## 自動ストレージとその属性

自動変数は、それが宣言されているブロックに制御が移されたときに 割り振られます。 自動変数の割り振りは、プログラムの実行中に何回も行うことができます。 ユーザーは、プログラムのブロック構造体をどのように設計するかによって、自動変数の割り振りを制御することができます。

省略形: AUTO

AUTOMATIC はデフォルトです。 自動変数は常に内部変数です。

以下の例では、プロシージャー B が呼び出されるたびに、変数 X と Y は ストレージを割り振られます。 B が終了すると、ストレージは解放され、X および Y に含まれる値は失われます。

```
A: proc;
 .
 .
 .
 call B;
 B: proc;
    declare X,Y auto;
    .
    .
    .
    end B;
  .
  .
  .
 call B;
```

解放されたストレージは、別の変数への割り振りのために使用可能になります。 このように、ブロック (プロシージャー・ブロックまたは開始ブロック) がアクティブである間は、そのブロック内で AUTOMATIC と宣言されたすべての変数にストレージが割り振られています。 ブロックが非アクティブであれば、そのブロック内の自動変数にストレージは 割り振られていません。 再帰的に呼び出されるか、または複数のプログラムによって呼び出される プロシージャーを除いて、個々の自動変数の割り振りは 1 つだけ存在することができます。

自動変数のエクステントは、式で指定することができます。 したがって、必要なときに、特定量のストレージを割り振ることができます。 次の例では、文字ストリング STR の長さは、プロシージャー B が 呼び出されたときに存在していた変数 N の値によ って定義された長さになります。

```
A: proc;
 declare N fixed bin;
 .
 .
 .
 B: proc;
    declare STR char(N);
```

宣言ステートメントが同じブロック内にある場合、PL/I では、変数 N を制限式または初期化された静的変数のいずれかに初期化する必要があります。 次の例では、割り振られた長さは Str1 に対して正しく割り振られます が、Str2 には正しくありません。 PL/I では、このタイプの宣言の依存性は 解決できません。

```
dcl N fixed bin (15) init(10),
    M fixed bin (15) init(N),
    Str1 char(N),
    Str2 char(M);
```

# 被制御ストレージとその属性

CONTROLLED として宣言された変数は、ALLOCATE ステートメントで指定されたと きにだけ割り振られます。 被制御変数は、その変数を指定している FREE ステートメントに制御権が到達する か、またはプログラムが終了するまで、割り振られたままになっています。

被制御変数は、プログラムのブロック構造体から、部分的に独立しています (ただ し、完全に無関係というわけではありません)。 被制御変数の有効範囲は、INTERNAL または EXTERNAL のいずれでもかまいません。 INTERNAL と宣言されている場合の有効範囲は、その変数を宣言しているブロッ クと、そのブロックに含まれているブロックです。 まだ割り振られていない被制御変数を参照すると、エラーになります。

CONTROLLED として宣言されていない変数は、CONTROLLED として変数を宣言しているプロシージャーに渡すことはできません。 しかし、CONTROLLED として宣言されている変数を、CONTROLLED として変数を宣言していないプロシージャーに渡すことはできます。

省略形: CTL

以下の例では、変数 X はプロシージャー B 内と、プ ロシージャー A の CALL ステートメントの実行に続く部分内で有 効に参照することができます。

```
A: proc;
   dcl X controlled;
   call B;
   .
   .
   .
   B: proc;
      allocate X;
      .
      .
   end B;
end A;
```

一般に、被制御変数が有用なのは、調整可能なエクステントを持つ大きなデータ集合がプログラムに必要なときです。 次の例のステートメントは、入力データに応じて必要なだけのストレージを割り振 り、不要になったときにそのストレージを解放します。

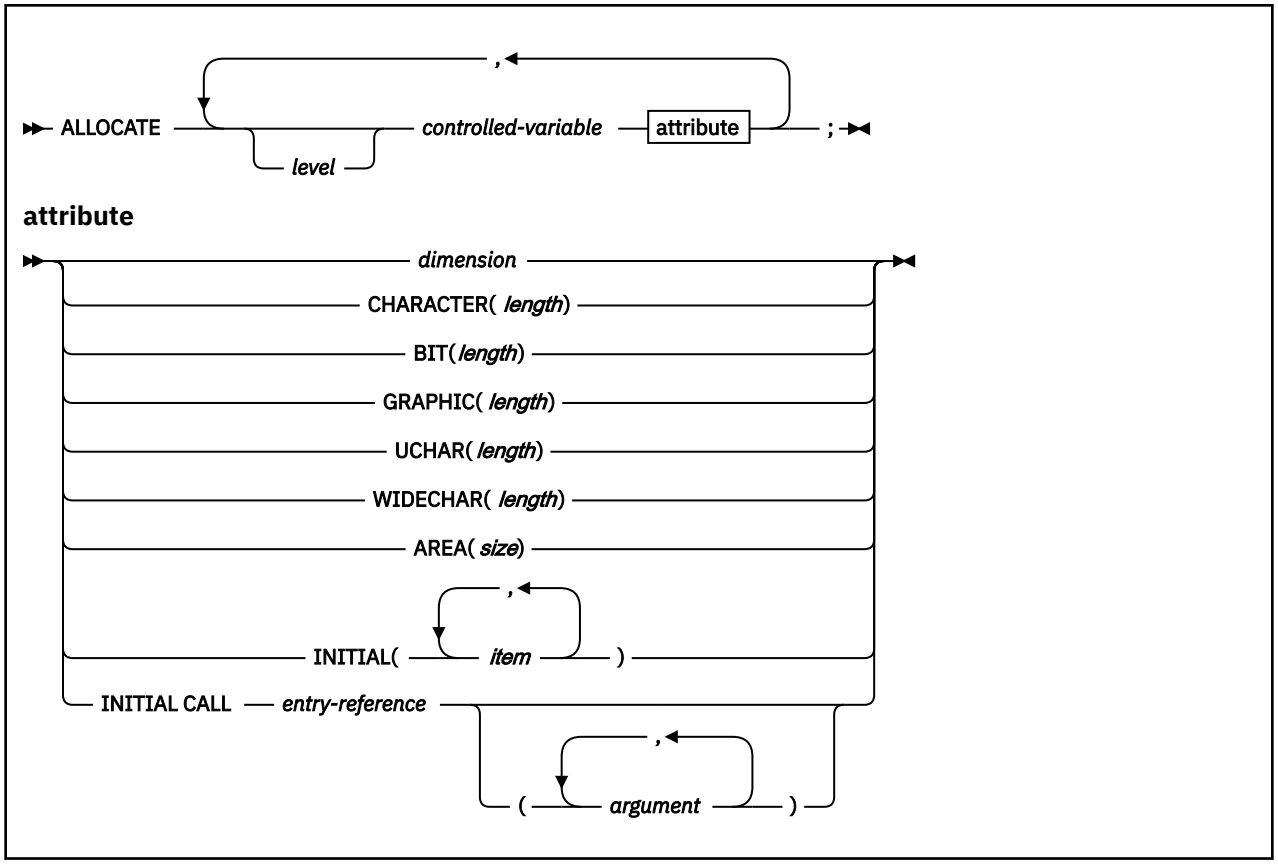
```
dcl A(M,N) ctl;
get list(M,N);
allocate A;
get list(A);
.
.
.
free A;
```

この方法を採用すると、ブロックを活動化したり終了したりする必要がないので、開始ブロックを設ける方法よりも効率がよくなります。

## 被制御変数のための ALLOCATE ステートメント

ALLOCATE ステートメントは、プロシージャー・ブロックの境界とは 関係なく、被制御変数にストレージを割り振ります。 同時に、被制御パラメーターも割り振ることができます。

制御されている配列の境界、制御ストリングの長さ、 制御域のサイズは、その初期値と同様に ALLOCATE ステートメントで 指定することができます。




省略形: ALLOC

**level**
レベル番号を示します。 レベル番号を指定しない場合、指定される被制御変数はレベル 1 変数で なければなりません。

**controlled-variable**
被制御変数または制御された小構造体のエレメントを指定します。 大構造体以外の構造体エレメントは、 エレメントを含む大構造体全体の相対構造化が、 その構造のための DECLARE ステートメント内にあるかのように示される 場合にのみ示すことができます。 この場合は、次元属性は、その次元属性で宣言されているすべての名前に指定されていなければなりません。

被制御変数と基底付き変数の両方を、同一の ALLOCATE ステートメントで 割り振ることができます。 基底付き変数の ALLOCATE ステートメントの構文については、248 ページの『基底付き変数のための ALLOCATE ステートメント』 を参照してください。

配列の境界、ストリングの長さ、 および区域のサイズ (エクステント) は、 ALLOCATE ステートメントの実行時に計算されます。

- ALLOCATE ステートメントまたは DECLARE または DEFAULT ステートメントは、 変数について、すべての必要な次元、サイズ、 または長さ属性 (エクステント) を指定しなければなりません。 DECLARE ステートメントから取られた式の計算は、ALLOCATE ステートメントの実 行時に割り込み可能になっている条件を使用して、割り振り時に行われます。 ただし、式の中の名前は、 その DECLARE または DEFAULT ステートメントを 有効範囲の中に持つ変数を参照しています。
- 1 つの ALLOCATE ステートメントに、境界値、長さ、またはサイズを 明示的に指定すると、 その変数について DECLARE ステートメントで 指定した値を指定変更します。
- 1 つの ALLOCATE ステートメントに、境界値、長さ、またはサイズを アスタリスクを指定して指定すると、 そのエクステントは現行世代から取得されます。 変数の世代が存在しない場合は、エクステントは未定義となり、 プログラムはエラーになります。
- ALLOCATE または DECLARE ステートメントのいずれかで、割り振ろうとしている変数の参照を含む式を使って、境界値、長さ、またはサイズを指定している場合は、変数の最新の世代の値を使用して式が計算されます。 次の例を考えてください。

```
declare X(N) fixed bin ctl;
N = 20;
allocate X;
allocate X(X(1));
```

X の最初の割り振りでは、declare ステートメントと N = 20; で、上限が指定されます。 2 番目の割り振りでは、 X の最初の世代の最初のエレメントの値によって、 上限が指定されます。

次元属性は、宣言したものと同じ次元数を指定しなければなりません。 次元属性は他の任意の属性とともに指定することができ、指定する属性の最初のものでなければなりません。 次の例を参照してください。

```
declare X(M) char(N) controlled;
M = 20;
N = 5;
allocate X(25) char(6);
```

BIT、CHARACTER、GRAPHIC、UCHAR、WIDECHAR、および AREA 属性は、 同じ属性を持つ変数についてのみ、それぞれ指定することができます。

変数が DECLARE または ALLOCATE ステートメントのいずれかに INITIAL 属性を持つ場合は、 初期値は割り振り時に変数に割り当てられます。 INITIAL 属性の式または CALL オプションは、 ALLOCATE ステートメントで使用可能な条件を使用して、 割り振り時に計算されます。 ただし、名前は宣言の環境で解釈されます。 INITIAL 属性が DECLARE と ALLOCATE ステートメントの両方に指定される場合は、ALLOCATE ステートメントの INITIAL 属性が使用されます。 初期設定時に、割り振られる変数が参照されている場合、その変数の 新しい世代を参照することになります。 初期設定について詳しくは、265 ページの『INITIAL 属性』を参照してください。

ALLOCATE ステートメントの実行時に行われる計算 (例えば、INITIAL 属性内の式の計算) は、相互に依存し合ってはなりません。

被制御変数のためのストレージが使用可能ではない場合、STORAGE 条件が発生します。

## 被制御変数のための FREE ステートメント

FREE ステートメントは、被制御変数に割り振られているストレージを解放します。 解放されたストレージは、別の割り振りに使用できます。 このとき、1 つ前に割り振られた被制御変数が使用可能になり、このあとでその 変数を参照すると、その割り振りを参照することになります。

**controlled-variable**
レベル 1、添え字なしの変数

基底付き変数と被制御変数の両方を、同一の FREE ステートメントで 解放することができます。 基底付き変数の FREE ステートメントの構文については、249 ページの『基底付き変数のための FREE ステートメント』を参照してください。

### 暗黙の開放

被制御変数は、必ずしも FREE ステートメントで明示的に解放する必要はありません。 しかし、被制御変数を (FREE ステートメントで) 明示的に解放することをお勧めします。

被制御ストレージはすべて、プログラムの終了時に解放されます。

## 被制御変数の複数世代

ストレージがすでに割り振られている (まだ解放されていない) 変数を 指定する ALLOCATE ステートメントが実行されると、その変数のストレージはスタックに入れられ (プッシュダウン) ま す。つまり、ストレ

ージがスタックされます。 スタッキングが行われると、その変数の新しい世代データが作られます。 この新しい世代が現行世代となり、

現行世代が解放されない限り、前の世代を直接にアクセスすることはできません。 FREE ステートメントによって、またはこの変数にストレージを割り振ったプログラムの終了時に、変数のストレージが解放されると、ストレージはスタックからポップアップ (除去) されます。

## アスタリスクの表記法

ALLOCATE ステートメントでは、次元、長さ、またはサイズがアスタリスクで示されたときは、1 つ前の世代から継承されます。 配列の場合、アスタリスクは、配列の 1 つではなくすべての次元に使用しなければなりません。

次の例を考えてください。

```
dcl X(M,N) char(A) ctl;
  M=10;
  N=20;
  A=5;

allocate X;
allocate X(10,10);
allocate X(*,*);
```

X の最初の世代は (10,20) の境界を持ちます。2 番目と 3 番目の 世代は (10,10) の境界を持ちます。 X の各世代のエレメントは、すべて長さ 5 の文字ストリングです。

DECLARE ステートメントでもアスタリスク表記を使用することができますが、その場合は、異なる意味を持ちます。 次の例を考えてください。

```
dcl Y char(*) ctl,
N fixed bin;

N=20;
allocate Y char(N);
allocate Y;
```

文字ストリング Y の長さが ALLOCATE ステートメントで 指定されていない場合は、直前の世代からそれを取ります。 その場合は、Y には指定された長さが与えられます。 そのため、ユーザーはストリングの長さの指定を、ストレージの実際の割り振りまで延期することができます。

## 調節可能エクステント

被制御スカラー、配列、および構造体や共用体のメンバーは、調整可能な配列エ クステント、ストリング長、および区域サイズを持つことができます。

例えば、次の例において、構造体が割り振られると、 A.B のエクステントは 1 から 10 になり、 A.C は最大の長さが 5 の VARYING 文字ストリングになります。

```
dcl 1 A ctl,
      2 B(N:M),
      2 C char(*) varying;
N = -10;
M = 10;
alloc 1 A,
         2 B(1:10),
         2 C char(5);
free A;
```

## 被制御変数のための組み込み関数

ALLOCATION 組み込み関数を使用すれば、特定の被制御変数に割り振られた世代の数を判別することができます。

変数が割り振られていない場合は、ゼロの値が戻されます。

# 基底付きストレージとその属性

基底付き変数の宣言は世代 (すなわち必要なストレージの量と その属性) の宣言です。

基底付き変数は、主記憶域での世代の位置を示しません。 世代の位置は、 ロケーター変数で示します。 割り振られていない基底付き変数の値を参照すると、エラーになります。

**locator-reference**
データの位置を指定します。

基底付き変数を参照するとき、データ属性と位置合わせ属性についてはその基底付き変数の属性が使用されますが、データの位置は、その変数を修飾しているロケーター変数が示します。

基底付き変数に EXTERNAL 属性を付けることはできませんが、基底付き変数の ロケーター参照には、基底付きストレージ・クラスを含む任意のストレージ・クラスを付けるこ とができます。

## BASED 宣言でのエクステント指定

ある BASED 変数のエクステントが 2 番目の変数の属性に依存することは可能ですが、これは、2 番目の変数が先に宣言されている場合に限られます。

例えば、A より前に B が宣言されていれば、A を BASED CHAR(LENGTH(B)) として宣言することができます。

REFER オプションを使用すれば、非定数エクステントを含む基底付き構造体または共用体を宣言することができます。 249 ページの『REFER オプション (自己定義データ)』 を参照してください。

REFER オプションを指定しない場合、BASED 宣言でのエクステント指定は、以下の例外を持つ制限付きの式でなければなりません。

- 配列が以下のすべての条件に該当する場合を除き、BASED 変数での非定数配列エクステントは無効です。
  - 配列が 1 次元である。
  - 配列の下限が定数である。
  - 配列が構造体の一部である場合に、その構造体における他のすべてのフィールドのエクステントが定数であり、 配列の後に、また配列の親構造体がある場合はその後にも、フィールドが存在しない。
- ストリングがスカラーの場合、または以下のすべての条件に該当する場合を除き、BASED 変数での非定数 CHAR エクステントは無効です。
  - このエクステントが構造体における最後のエレメントである。
  - 配列となっている親がない。
  - 属性が UNALIGNED、NONVARYING、または VARZ のいずれかである。
- BASED 変数での以下の非定数エクステントはいずれも、変数がスカラーの場合にのみ有効です。
  - 非定数 AREA エクステント
  - 非定数 BIT エクステント
  - 非定数 GRAPHIC エクステント
  - 非定数 UCHAR エクステント
  - 非定数 WIDECHAR エクステント

**例**

以下の宣言はすべて有効です。

**例 1**

```
dcl
  1 a1(n)  based,
    2 b,
      3 b1   fixed bin(31),
      3 b2   fixed bin(31);
```

**例 2**

```
dcl
  1 a2   based,
    2 b(n),
      3 b1   fixed bin(31),
      3 b2   fixed bin(31);
```

**例 3**

```
dcl
  1 a3   based,
    2 b,
      3 b1    fixed bin(31),
      3 b2(n) fixed bin(31);
```

**例 4**

```
dcl
  1 a4  based,
    2 b,
      3 b1    fixed bin(31),
      3 b2    char(n);
```

**例 5**

```
dcl
  1 a5  based,
    2 b,
      3 b1    fixed bin(31),
      3 b2(n) char(m);
```

## BASED VARYING ストリング

基底付きの VARYING、VARYING4、VARYINGZ のストリングの最大長は、それらのストリングがオーバーレイするストリングの最大長に等しくなければなりません。

次の例を参照してください。

```
declare A char(50) varying based(Q),
        B char(50) varying;
        Q=addr(B);
```

基底付き VARYING ストリングは VARYING ストリング上にのみ、基底付き VARYING4 ストリングは VARYING4 ストリング上にのみ、基底付き VARYINGZ ストリングは VARYINGZ ストリング上にのみ、それぞれオーバーレイすることができます。

## BASED 変数のストレージ割り振り

基底付き変数用のストレージは、ALLOCATE ステートメント、ALLOCATE 組み込み関数、 AUTOMATIC 組み込み関数、または LOCATE ステートメントを使用して割り振る ことができます。

基底付き変数は、(SET オプションを指定した) READ ステートメント、FETCH ステ ートメント、または ADDR 組み込み関数を使用することによって、既存のデータを アクセスするときにも使用できます。

基底付き AREA 変数は、ALLOCATE ステートメントを使用して割り振ることが できます。PL/I は割り振りを行うときに自動的に区域を EMPTY に初期設定します。 ただし、組み込み関数 ALLOCATE または AUTOMATIC によって区域変数用のストレージを取得する場合は、ストレージの取得後に EMPTY を変数に割り当てる必要があります。

## ロケーター変数

割り振られた世代の位置はロケーター変数で示されるので、適切なロケーター 値を使用すれば、プログラム内の任意の場所で基底付き変数のどの世代でも参 照することができます。

次の例では、参照が明示的に修飾されているときを除いて、X を参 照するときにはロケーター変数 P によって X の記憶場所 が示されることを宣言しています。

```
dcl X fixed bin based(P);
```

このようにしてロケーター参照に関連付けても、その関連付けは永続的なものではありません。 ロケーター参照は、他の基底付き変数の位置を識別するために使用できます。他のロケーター参照は、変数 X の他の世代を識別するために使用できます。 基底付き変数がロケーター参照なしで宣言されている場合、基底付き変数への参照は、常に明示的にロケーター修飾されていなければなりません。

次の例では、配列 A と配列 C は同じストレージを指します。 エレメント B とエレメント C(2,1) も同じストレージを指します。

```
dcl A(3,2) character(5) based(P),
    B char(5) based(Q),
    C(3,2) character(5);
P = addr(C);
Q = addr(A(2,1));
```

**注 :** 基底付き変数をこの方法でオーバーレイするときは、新しいストレージは 割り振られません。 基底付き変数は、オーバーレイされる変数 (この例では C(3,2)) と 同じストレージを使用します。

## DEFINED 属性および UNION 属性

DEFINED 属性および UNION 属性のいずれを使用しても変数をストレージにオーバーレイすることができますが、DEFINED および UNION によるオーバーレイは永続的な ものとなります。 ロケーター参照によって基底付き変数をオーバーレイしたときは、ロケーター 変数に新しい値を割り当てれば、プログラムの実行中いつでも、その結びつき を変更できます。

**関連情報**

261 ページの『DEFINED 属性および POSITION 属性』
DEFINED 属性は、宣言される変数が、指定された基本変数のストレージの 一部または全体に関連付けられることを指定します。

175 ページの『共用体』
共用体 とは、相互にオーバーレイし合い、同じストレージを有するメンバー・エレメントの集合です。 メンバーは、構造体、共用体、要素変数、および配列であることができます。 それらは、同一の属性を持っていなくても構わない。

## INITIAL 属性

基底付き変数に INITIAL 属性を指定してもかまいません。 初期値が割り当てられるのは、ALLOCATE または LOCATE ステートメントに よって基底付き変数が明示的に割り振られたときだけです。

## ロケーター・データ

ロケーター・データには 2 つのタイプ、ポインターとオフセットがあります。

ポインター変数 の値は、ストレージの位置のアドレスです。 いくつかの異なる記憶場所でストレージを割り振られた変数を参照するときは、この値を使って参照を修飾することができます。

オフセット変数 の値は、区域変数内の位置を指定し、その区域がストレージの別の場所に割り当てられている場合には引き続き有効です。

ロケーター値は、ロケーター変数にだけ割り当てることができます。 オフセット値がオフセット変数に割り当てられるときは、OFFSET 属性で 指定された区域変数は無視されます。

## ロケーターの変換

少数の例を除き、ロケーター・データは他のデータ・タイプに変換できません。

ロケーター・データは、以下のように他のデータ・タイプに変換できます。

- BINARYVALUE、POINTERVALUE、および OFFSETVALUE 組み込み関数を使用して、REAL FIXED BINARY (p,0) との間で行われる変換。
- POINTER および OFFSET 組み込み関数を明示的または暗黙的に使用して、ポインターとオフセット間で行われる変換。

オフセット変数が参照されると、OFFSET 属性およびオフセット変数で指定され た区域変数のアドレスによって、オフセット変数はポインター値に暗黙に変換 されます。 オフセットのポインター値への明示的変換は、POINTER 組み込み関数によって行うことができます。 次の例では、ステートメントは、区域 B 内のオフセット 0 に よって識別される基底付き変数の記憶位置を示すポインター値 を、P に割り当てます。

```
dcl P pointer, O offset(A),B area;
P = pointer(O,B);
```

この区域変数は、オフセット変数に関連付けられた区域変数とは異なっているので、そのオフセット値が別の区域で有効であるかを確認しなければなりません。 例えば、区域 A が区域 B に割り当てられたあとでこの 関数が呼び出された場合は、そのオフセット値は有効です。

OFFSET 組み込み関数は、POINTER 組み込み関数とは対照的に、与えられたポインター と区域から得られたオフセット値を返します。 この場合、そのポインター値は、その区域内での基底付き変数の位置を 示していなければなりません。

ポインター値は、そのポインター値およびその区域のアドレスを使用して、オフセットに変換されます。したがって、この変換ができるのは、OFFSET 属性内で指定された区域内のアドレスに結び付けられたポインター値に限られます。

NULL 組み込み関数または SYSNULL 組み込み関数の値に割り当てる場合を除いて、区域 に関連付けられていないオフセット変数への変換や、このオフセット変数から の変換を試みると、エラーになります。

複数割り当ての場合、ロケーターは暗黙に変換されることはありません。

## ロケーター参照

ロケーター参照とは、ロケーター変数 (修飾付きまたは添え字付きでもよい) また は、ロケーター値を返す関数参照のいずれかのことです。

ロケーター参照は、次のように使用することができます。

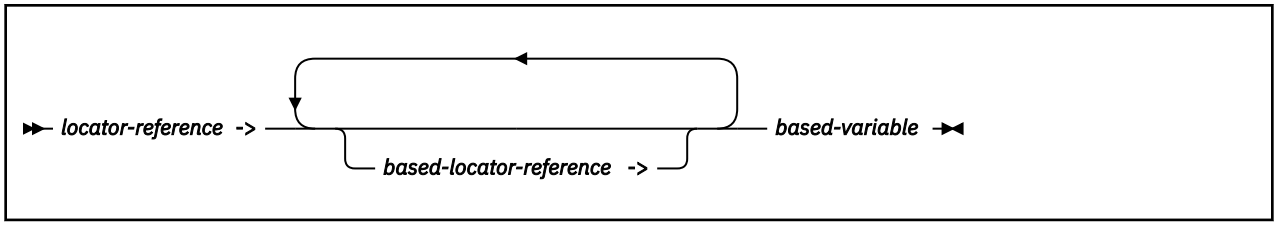
- 基底付き変数の宣言に関連付けて、ロケーター修飾子として使用できます。
- 比較演算で使用できます。例えば、IF ステートメント内で使用できます。
- プロシージャー参照内で引数として使用することができます。

PL/I はオフセットをポインター値に暗黙に変換するため、 オフセット参照とポインター参照のどちらも使用することができます。

## ロケーター修飾

ロケーター修飾とは、ロケーター基底付き変数の特定の世代を識別するために、1 つ以上のロケーター参照を基底付き参照に関連付けることです。 これを、ロケーターで修飾された参照といいます。

複合記号 -> は、"修飾元"または"ポイント先"を意味します。 明示的に修飾された参照の構文図を次に示します。




**locator-reference**
**based-locator-reference**
データの位置を指定します。

次の例では、X は基底付き変数であり、P はロケーター 変数であり、Q は基底付きロケーター変数です。

```
P -> Q -> X
```

この参照は、基底付きロケーター Q によって識別され、また、ロケーター P の値によっても識別される、X の世代であることを意味します。 X および Q は明示的にロケーター修飾されています。

複数のロケーター修飾子を使用している場合、それらの値の計算は順に左から右へ行われます。

基底付き変数の参照を暗黙に修飾する こともできます。 暗黙に修飾された基底付き変数の世代を判別するために使われるロケーター参照は、基底付き変数と一緒に宣言されたものです。 次の例では、ALLOCATE ステートメントは、ポインター変数 P の値を セットします。したがって、X を参照すると、割り振られたストレージ が参照されます。

```
dcl X fixed bin based(P) init(0);
allocate X;
X = X + 1;
```

代入ステートメントで X を参照していますが、これは P によって暗黙にロケーターで修飾されます。X を参照するとき、次の例のように明示的にロケーターで修飾することもできます。

```
P->X = P->X + 1;
```

次の代入ステートメントが実行されると、 上記の場合と同じ結果になります。

```
Q = P;
Q->X = Q->X + 1;
```

基底付き変数と一緒に宣言されるロケーターも基底付きであってもかまいま せん。したがって、ロケーター修飾子のチェーンになることがあります。 例えば、 次のようなポインターおよび基底付き変数を使用することができます。

```
declare (P(10),Q) pointer,
         R pointer based (Q),
         V based (P(3)),
         W based (R),
         Y based;
allocate R,V,W;
```

先のように宣言し、割り振りを行った場合、 下記の参照はすべて正しい参照です。

```
P(3) -> V
V
Q -> R -> W
R -> W
W
```

最初の 2 つの参照は同じことを意味し、その次の 3 つも同じことを意味します。 Y を参照するときは、それを修飾するロケーター変数を指定しなければなりません。

### ロケーター修飾のレベル

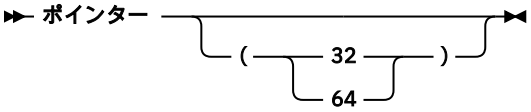
基底付き変数を修飾するポインターは、1 レベルのロケーター修飾を表します。 オフセットは、区域内で暗黙に修飾されているので、2 レベルのロケーター修飾を表します。 レベル数は、添え字付きのロケーターによって、あるいは構造体または共用体の要素によって影響を受けません。

以下の例では、参照 X、P -> X、および Q -> P -> X は、3 つのレベルのロケーター修飾を表しています。

```
declare X based (P),
        P pointer based (Q),
        Q offset (A);
```

## POINTER 変数とその属性

ポインター変数は、基底付き変数の宣言内に書いたとき、ロケーター修飾子と して書いたとき、BASED 属性内に書いたとき、あるいは ALLOCATE、LOCATE、 READ、または FETCH の各ステートメントの SET オプション内に書いたときに コンテキスト上から宣言されます。 ポインター変数を明示的に宣言することもできます。




省略形: PTR

**32**
POINTER(32) は、サイズが 4 バイトであり、デフォルトでフルワード位置合わせされます。

**64**
POINTER(64) はサイズが 8 バイトであり、デフォルトでダブルワード位置合わせされます。

LP(32) コンパイラー・オプションが有効な場合、デフォルトは POINTER(32) です。 LP(64) コンパイラー・オプションが有効な場合、デフォルトは POINTER(64) です。 POINTER(64) は LP(64) でのみ有効です。

POINTER(32) の POINTER(64) への割り当ては常に有効です。 その逆は、POINTER(64) の先頭 4 バイトがゼロの場合にのみ有効です。

基底付き変数の世代をもはや識別しないポインター変数、例えば、基底付き変数がすでに解放された場合の値は、未定義です。 ポインターで修飾される変数を参照するときは、そのポインターに値を 入れておかなければなりません。

## 基底付き変数のための組み込み関数

基底付き変数のための組み込み関数には、ALLOCATE、PLIFREE、AUTOMATIC、ADDR、ENTRYADDR などがあります。

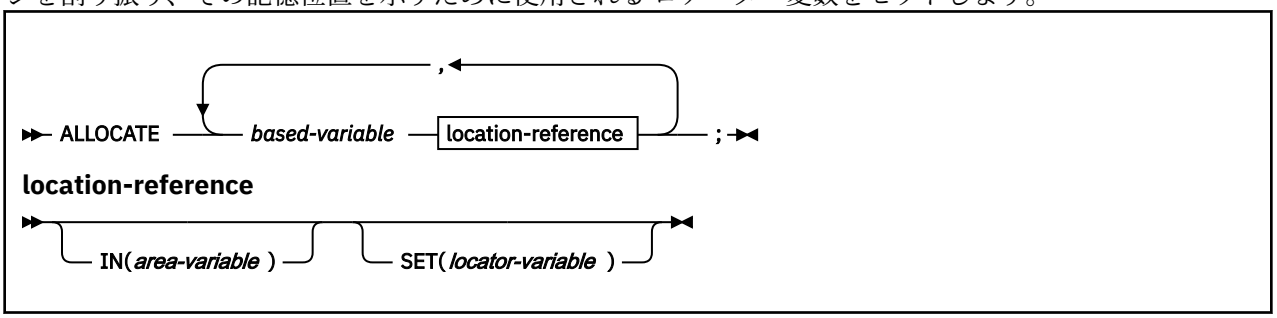
ALLOCATE 組み込み関数を使用すれば、基底付き変数のストレージを獲得できます。また PLIFREE 組み込みサブルーチンを使用すれば、そのストレージを解放できます。 AUTOMATIC 組み込み関数を使用しても、基底付き変数のストレージを獲得できますが、 この関数を使用して獲得したストレージは、明示的に解放してはなりません。 AUTOMATIC 組み込み関数で割り振られたストレージは、それが割り振られているブロックが終了したときに自動的に解放されます。

ADDR 組み込み関数は、変数の先頭バイトを指すポインター値を返します。 ENTRYADDR 組み込み関数は、入り口が呼び出されたときに、最初に実行される命令のアドレスを示すポインター値を返します。 NULL 組み込み関数は、PL/I のヌル・ポインターを返し、SYSNULL 組み込み関数は、システ ムのヌル・ポインターを返します。

**注 :** NULL 組み込み関数と SYSNULL 組み込み関数は、同じように比較を行えますが、必ずし も等しいというわけではありません。 アプリケーション・プログラムの作成において、関数の等価性に依存するような記述をす べきでは**ありません**。

# 基底付き変数のための ALLOCATE ステートメント

ALLOCATE ステートメントは、プロシージャー・ブロックの境界とは関係なく、 基底付き変数にストレージを割り振り、その記憶位置を示すために使用される ロケーター変数をセットします。




省略形: ALLOC

**基底付き変数 (based variable)**
レベル 1 の添え字なし変数

**IN**
ストレージを割り振る区域変数を指定します。 区域について詳しくは、251 ページの『区域データとその属性』を参照してください。

**SET**
割り振られたストレージの位置に設定されるロケーター変数を指定します。 SET オプションを省略する場合は、基底付き変数の宣言で指定されたロケータ ーを使用しなければなりません。 基底付き変数を宣言する場合の構文の詳細については、242 ページの『基底付きストレージとその属性』および 244 ページの『ロケーター・データ』を参照してください。

基底付き変数と被制御変数の両方を、同一の ALLOCATE ステートメントで 割り振ることができます。 被制御変数の ALLOCATE ステートメントの構文については、238 ページの『被制御変数のための ALLOCATE ステートメント』を参照してください。

IN オプションを指定した場合、または SET オプションでオフセット変数を指定した場合、ストレージは区域内でが割り振られます。 これらのオプションの指定順序は任意です。

区域内での割り振りの場合、区域で基底付き変数用のストレージが不足していると、AREA 条件が発生します。 IN オプションなしでオフセット変数を使用する場合は、オフセット変数の宣言で区域参照を指定する必要があります。

区域を使用しない場合は、ロケーター変数はポインター変数でなければなりません。 基底付き変数のためのストレージが使用可能でない場合、STORAGE 条件が起こります。

基底付き変数が REFER を使用している場合、そのサイズは実行時に計算されます。 この計算で生成される値が大きすぎて *size_t*[2] 変数に収まらない場合、プログラムはエラーとなり、修正する必要があります。 この状態では、STORAGE 条件は発生しません。 代わりに、以下のいずれかの条件に該当する場合は、ONCODE=3809 の ERROR 条件が発生します。

- SIZE 条件が割り込み可能である。
- BASED 構造体がライブラリー呼び出しによってマップされている。

これらの条件のどちらも該当しない場合、予測不能な結果が生じます。

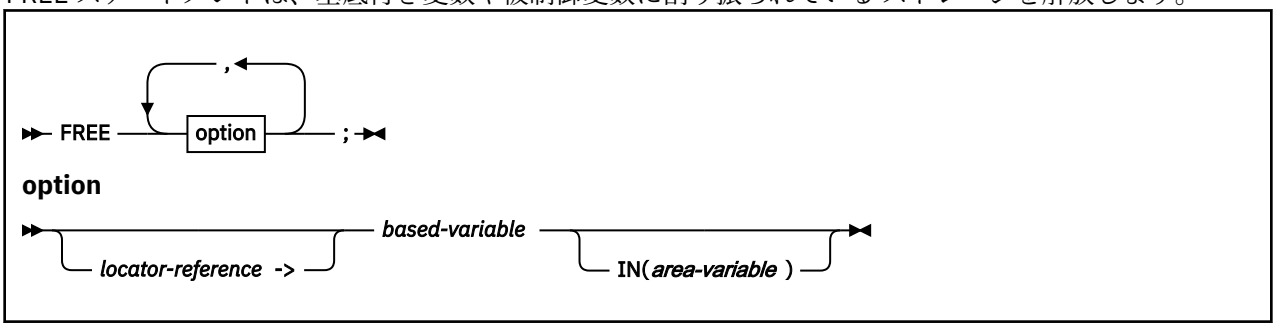
基底付き変数に割り振られるストレージの大きさは、その変数の属性に左右されるほか、その変数の次元、長さ、またはサイズが割り振り時に指定されるときは、それらにも左右されます。 これらの属性は、基底付き変数の宣言から判別されます。

基底付きの構造体や共用体には、調整可能な配列境界、ストリングの長さ、 または区域のサイズを含むことができます (249 ページの『REFER オプション (自己定義データ)』を参照)。 基底付き変数の場合、エクステントとしてアスタリスクを表記することはできません。

[2] LP(32) コンパイラー・オプションが有効な場合、*size_t* は FIXED BIN(31) です。 LP(64) コンパイラー・オプションが有効な場合、*size_t* は FIXED BIN(63) です。

# 基底付き変数のための FREE ステートメント

FREE ステートメントは、基底付き変数や被制御変数に割り振られている ストレージを解放します。




**locator-reference ->**
基底付き変数の特定の世代を解放します。 複合記号 -> は "修飾元" または "ポイント先" を意味します。 基底付き変数が明示的にロケーターで修飾されていない場合は、BASED 属性で宣 言されたロケーター変数が、解放されるデータの世代を識別するために使われます。 ロケーターが 1 つも宣言されていない場合、このステートメントはエラーに なります。

**基底付き変数 (based variable)**
添え字なしの、レベル 1 の基底付き変数でなければなりません。

**IN**
解放されるストレージが区域内に割り振られている場合は、このオプションを指定するか、または基底付き変数を、対応する区域と一緒に宣言されたオフセットで修飾しなければなりません。 基底付き変数が区域内に割り振られていない場合は、IN オプションを 指定することはできません。 割り振り時に区域を指定すると、基底付きストレージがターゲット区域の中に割り振られます。 このような割り振りは、IN オプションでターゲット区域を指定すれば解放することができます。

基底付き変数と被制御変数の両方を、同一の FREE ステートメントで 解放することができます。 被制御変数の FREE ステートメントの構文については、240 ページの『被制御変数のための FREE ステートメント』を参照してください。

基底付き変数を使用してストレージを解放することができるのは、それと同じデータ属性を持つ基底付き変数に割り振られているストレージを解放する場合だけです。

解放されるストレージの大きさは、基底付き変数の属性 (ストレージ解放の時点での境界または長さを含む) に左右されます。 ユーザーは、この大きさが割り振られたストレージの大きさと一致しているかを判別する必要があります。 変数が割り振られていない場合の結果は予測不能です。

## 暗黙の開放

基底付き変数は、必ずしも FREE ステートメントで明示的に解放する必要はありません。しかし、(FREE ステートメントで) 基底付き変数を明示的に解放することをお勧めします。

基底付きストレージはすべて、プログラムの終了時に解放されます。

# REFER オプション (自己定義データ)

自己定義構造体または共用体とは、それ自身のフィールドについての情報 (ストリング の長さなど) を含んでいる構造体または共用体です。 このデータを処理できるように、基底付き構造または共用体を宣言することが できます。 ストリングの長さ、配列の境界、区域のサイズはすべて、構造体内または共用体内で 宣言した変数 (参照オブジェクト) によって定義することができます。 基底付き構造体または共用体の宣言で、REFER オプションは、その構造体または共用体の割り振り時に、式の値が参照オブジェクトに割り当てられ、その値が構造体または共用体内の別の変数の長さ、境界、またはサイズを表すことを指定します。

その構造体または共用体が割り振られる (ALLOCATE ステートメントまたは LOCATE ス テートメントによって) とき、式の値が参照オブジェクト変数に割り当てられます。 それ以外にこの構造体または共用体を参照したときは、参照オブジェクトの値が使われます。

REFER オプションで長さ、境界、またはサイズを指定する場合の構文は次のとおりです。

**expression**
この式の値は、構造体または共用体の (ALLOCATE または LOCATE による) 割り振り時に、メンバーの長さ、境界、またはサイズを定義します。 式が評価され、*size_t*[3] に変換されます。式でオペランドとして使用される変数は、REFER オプションを含む構造体または共用体に属していてはなりません。

構造体または共用体への後続の参照では、REFER オプションのメンバーの長さ、 境界、もしくはサイズを、*member-variable* (参照オブジェクト) の 現行値から獲得します。

**member-variable**
参照オブジェクトは、次の規則に従わなければなりません。

- 同じレベル 1 の構造体または共用体のメンバーでなければならず、 REFER オプションの中でそのオブジェクトを指定するメンバーの前に なければなりません。
- 計算できるものでなければなりません。 スケール係数がゼロの FIXED BIN でなければなりません。 タイプが FIXED BIN(p,0) 以外である場合、コンパイラーは W レベル・メッセージを出します。
- ロケーター修飾 (245 ページの『ロケーター修飾』を参照) をしたり、添え字を 付けたりすることはできません。
- 配列の一部であってはなりません。

次の例の宣言では、基底付き構造体 STR が配列 Y と エレメント X で構成されていることを指定しています。

```
declare 1 STR based(P),
          2 X fixed binary(31,0),
          2 Y (L refer (X)),
        L fixed binary(31,0) init(1000);
```

STR が割り振られるとき、上限は、X に割り当てられて いる L の現行値にセットされます。 ほかの場所 (P をセットするための READ ステートメントなど で) Y を参照すると、X に入っている制限値が使われます。

REFER オプションを持つメンバーに INITIAL 属性を指定すると、参照オブジェクトに値が割り当てられてから、メンバーの初期設定が行われます。

構造体または共用体の宣言で使用できる REFER オプションの数に制限はありません。

参照オブジェクトの値は、プログラムの実行中に変更してはなりません。 参照オブジェクトの値が変更された場合、集合などを解放するエラーが発生します。

また、REFER エクステントを定義する式の中で使用する変数は、その REFER を使用する DECLARE を含むブロック (またはその親ブロック の 1 つ) 内で宣言されている必要があります。 変数のいずれかが宣言されていない場合、 その変数は暗黙宣言に関する通常の規則に従って暗黙的に宣言されます。 つまり、DECLARE を含む最外部のブロックに、 その変数の DECLARE が追加されます。

したがって、次のコードの中で、サブルーチン `inner_proc` 内での変数 `m` の 宣言と割り当ては、ALLOCATE ステートメントに影響しません。 ALLOCATE ステートメントは、メイン・ブロックで暗黙的に宣言された未初期化の `m` を使用します。

```
refertst: proc options(main);

   dcl
     1 a based,
       2 n fixed bin(31),
       2 c char(m refer(n));

   call inner_proc;

   inner_proc: proc;
```

[3] LP(32) コンパイラー・オプションが有効な場合、*size_t* は FIXED BIN(31) です。 LP(64) コンパイラー・オプションが有効な場合、*size_t* は FIXED BIN(63) です。

```
      dcl m fixed bin(31);
      dcl p pointer;

      m = 15;
      allocate a set(p);
    end;
  end;
```

## 区域データとその属性

区域変数は、基底付き変数を割り振るために予約される主記憶域の区域を 定義するものです。 ALLOCATE ステートメントや FREE ステートメントを使用すれば、この 予約されたストレージを基底付き変数に割り振ったり、基底付き変数から 解放したりすることができます。

区域変数には任意のストレージ・クラスを指定でき、その位置合わせを行うこと、または行わないことが可能です。 AREA 変数を UNALIGNED として宣言すると、ダブルワードではなくバイトで位置合わせされます。

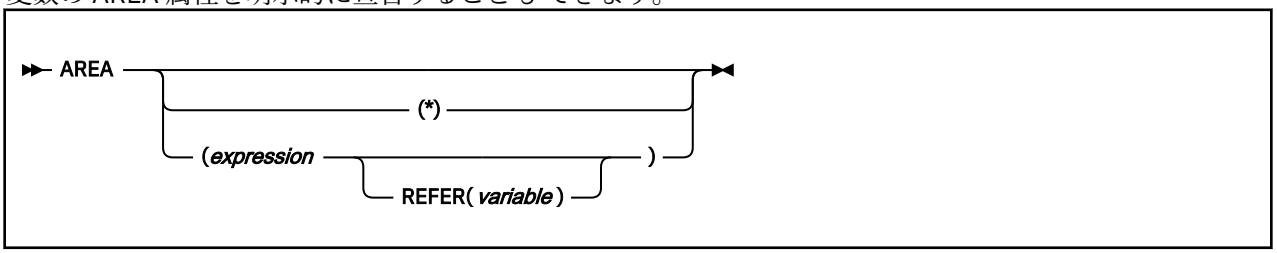
基底付き変数の割り振り時に、区域を指定しない場合、ストレージは使用可能な任意の場所から取得されます。 したがって、割り振られた複数の基底付き変数が主記憶域全体にわたって分散していることもあります。 内部操作の場合、項目はポインターによって容易にアクセスできるため、分散していても支障はありません。 しかし、割り振られた項目をデータ・セットに伝送する場合は、それらを 1 つにまとめる 必要があります。 区域変数に割り振られた項目はすでに 1 つにまとまっているので、1 つの 単位として伝送や割り当てを行うことができ、しかもそれらを個別に処理することもできます。

ある区域変数内での基底付き変数の位置は、その区域変数の開始位置からの変位として識別できると便利です。 この目的のためにオフセット変数が用意されています。

区域はその含まれる割り振りで、完全に割り当てまたは伝送することができます。そのため、基底付き割り振りのセットは、割り当てと入出力のための単位として扱うことができ、各割り振りは独自の識別を保持します。

区域のサイズは、ストリングの長さや配列の境界と同じ方法で調節できるので、式 やアスタリスク (被制御区域パラメーターの場合)、または REFER オプション (基底付き区 域の場合) を使用して指定することができます。

OFFSET 属性または IN オプション内に書いた変数には、コンテキスト上から、AREA 属性が与えられます。変数の AREA 属性を明示的に宣言することも できます。




**expression**
区域のサイズを指定します。 *expression* またはアスタリスクを指定しない場合、デフォルトは 1000 です。

*****
宣言される区域変数がパラメーターの場合は、サイズをアスタリスクで指定することができます。

**REFER**
REFER オプションの説明については、249 ページの『REFER オプション (自己定義データ)』を 参照してください。

ストレージ・クラスが AUTOMATIC または CONTROLLED である区域のサイズは、予約され るストレージのバイト数を表す値になる式で指定します。

区域が BASED 属性を持っているときは、区域のサイズは定数で指定しなければな りません。ただし、その区域が基底付き構造体または共用体のメンバーで、しか も REFER オプションを使用する場合は例外です。

静的ストレージ・クラスの区域サイズは、制限付き式で指定しなければなりません。

以下に AREA 宣言の例を示します。

```
declare area1 area(2000),
        area2 area;
```

宣言されたサイズの区域が予約され、その予約された区域の前にさらに 16 バイトが付けられます。 この 16 バイトには、使用中のストレージの大きさなどの制御情報の詳細が入ります。

実際に使用される予約済みのストレージの大きさは、区域のエクステント といいます。 区域変数が割り振られた時点では、そこにはなにも入っていないので、区域のエクステントはゼロです。 最大エクステントはその区域のサイズです。 実行中いつでも、区域内で基底付き変数を割り振ることも解放することもできます。したがって、区域のエクステントは変化します。

基底付き変数が解放されると、その変数が占有していたストレージは別の割り振りに使用可能になります。ある区域内の使用可能なストレージのチェーンは保持されます。チェーンの 先頭は制御情報内に保持されます。 基底付き変数に必要なストレージの容量はそれぞれ異なっているので、それらを割り振ったり解放したりすれば、割り振りの容量と使用可能なスペースの容量が一致しないときには必然的にすき間が生じます。 これらのすき間は区域のエクステントに含まれます。

どのような演算子 (比較演算子を含む) も、区域変数に使用することはできません。

# オフセット・データとその属性

オフセット・データは、必ず区域変数と一緒に使われます。 オフセット変数の値は、区域変数内での基底付き変数の位置を、その 区域変数の開始位置からの変位として示します。

基底付き変数は相対的な位置で示されるので、区域変数が主記憶域の 別の部分に割り当てられても、オフセット値は無効になりません。

オフセット変数を使用しても、区域内でポインター変数を使用できなくなる わけではありません。

区域変数とオフセット変数の関連付けは、永続的なものではありません。 POINTER 組み込み関数を使用すれば、オフセット変数を任意の区域変数に 関連付けることができます (245 ページの『ロケーターの変換』を参照)。 このような関連付けを宣言しておくことの利点は、このオフセット変数を参照すると、関連付けられている区域変数を参照することになるという点です。 区域変数の指定を省略すると、このオフセット変数は、POINTER 組み込み関数の使用を介してしかロケーター修飾子として使用することができません。

## オフセット変数のセット

オフセット変数の値は、下記のいずれかの方法でセットすることができます。

- ALLOCATE ステートメントによってセットします。
- 別のロケーター変数の値、またはユーザー定義の関数から戻されたロケーター値の 割り当てによってセットします。
- 組み込み関数 NULL、SYSNULL、ADDR、ENTRYADDR、OFFSETADD、 OFFSETSUBTRACT、OFFSETVALUE、または OFFSET によってセットします。

区域変数の指定を省略したときは、そのオフセット変数は、POINTER 組み込み関数の使用を介してロケーター修飾子として使用することしかできません。

## オフセット変数の例

次の例を考えてください。

```
dcl X based(O),
    Y based(P),
    A area,
```

```
      O offset(A);

      allocate X;
      allocate Y in(A);
```

区域 A とオフセット O のストレージ・クラスは、デフォルトによって AUTOMATIC になります。 最初の ALLOCATE ステートメントは、次のステートメントと同等です。

```
   allocate x in(A) set(O);
```

2 番目の ALLOCATE ステートメントは、次のステートメントと同等です。

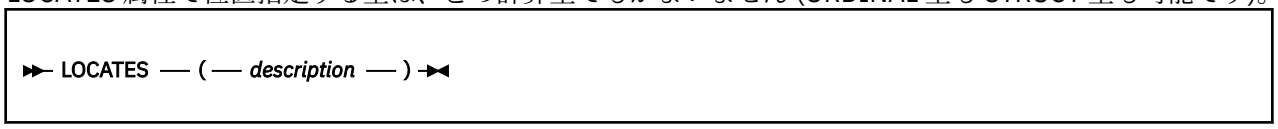
```
   allocate Y in(A) set(P);
```

次の例は、オフセット変数を使用して区域変数内にリストを作成するときの方法を示しています。

```
   dcl A area,
       (T,H) offset(A),
       1 STR based(H),
         2 P offset(A),
         2 data;

       allocate STR in(A);
       T=H;

     do loop;
       allocate STR set(T->P);
       T=T->P;
       .
       .
       .
   end;
```

## LOCATES 属性

LOCATES 属性を指定すると、コンパイラーによってストリングの疎配列が作成されて渡されるときに、ストレージが削減されます。 LOCATES 属性は、OFFSET 属性を使用する場合にのみ有効です。 さらに、LOCATES 属性で位置指定する型は、どの計算型でもかまいません (ORDINAL 型も STRUCT 型も可能です)。




***description***
一連の属性を、ブランクまたは括弧で区切って指定する必要があります。 属性は、DECLARE ステートメントで宣言する場合と同じ方法で指定することができます。 コンパイラーは、通常の方法でデフォルトを適用します。

*description* 内の属性には、 スカラー BIT、CHARACTER、UCHAR、および WIDECHAR ストリングの任意のデータ属性および位置合わせ属性を指定できます。 ストリングの長さは、定数で指定する必要があります。コンパイラーは、関連ストレージを割り振るときに指定された位置合わせ属性を無視します。 次の例 1 は、LOCATES 属性の指定方法を示しています。

**例 1**

```
          declare
              1 data based(data_ptr) unaligned,
                 2 actual_count fixed bin(31),
                 2 orderinfo(order_count refer(actual_count)),
                    3 name    offset(pool) locates(char(30) varying),
                    3 address offset(pool) locates(char(62) varying),
                 2 pool area(10_000);
```

LOCATES 属性は、関連オフセットをタイプ付きオフセットにします。 その後、LOCVAL 組み込み関数および疑似変数を使用して、タイプ付きオフセットを参照解除することができます。 参照解除プロセスは、C 言語で * 演算子がタイプ付きポインターを参照解除するプロセスと同じです。

次の組み込みサブルーチンおよび組み込み関数を使用すると、LOCATES 属性を持つオフセットを割り振り、値を関連区域に割り当てて、これらのオフセットに関連付けられた値を取得することができます。

- LOCNEWSPACE

  467 ページの『LOCNEWSPACE』 組み込みサブルーチンを使用すると、LOCATES 属性で必要な最大長に十分なスペースを指定して OFFSET 変数を割り振ることができます。

- LOCNEWVALUE

  468 ページの『LOCNEWVALUE』 組み込みサブルーチンを使用すると、LOCATES 属性で指定された値を保持するために十分なスペースを指定して OFFSET 変数を割り振ることができます。 また、指定された値が関連アドレスに割り当てられます。

- LOCVAL

  469 ページの『LOCVAL』 組み込み関数および疑似変数を使用すると、LOCATES 属性を持つ OFFSET 変数を参照解除することができます。

- LOCSTG

  469 ページの『LOCSTG』 組み込み関数を使用すると、LOCATES 属性を持つ変数または LOCATES 属性を持つエレメントが含まれている変数で、間接的に保持できるすべての値を保持するために、AREA 変数で必要な最大サイズを判別することができます。

例えば、例 1 の宣言を前提とすると、以下のステートメントはどちらも、ストリングを保持するための pool 区域にスペースを割り振り、そのオフセットを name(1) に割り当ててから、値 Sherlock Holmes を可変文字ストリングとして区域内のその位置に割り当てます。 call locnewvalue ステートメントは、指定された値を保持するために 17 バイトを割り振ります。 call locnewvalue ステートメントは、LOCATES 属性内のストリング属性 (CHAR(30) VARYING) に必要なフルサイズの 32 バイトを割り振ります。

以下のステートメントについて考察します。

**ステートメント 1**

```
call locnewspace(name(1));
locval(name(1)) = 'Sherlock Holmes';
```

**ステートメント 2**

```
call locnewvalue ('Sherlock Holmes', name(1));
```

LOCATES 属性を持ち、AREA が指定された OFFSET 変数への割り当てでは、ソースが計算である場合、割り当ては LOCNEWVALUE 組み込みサブルーチンへの呼び出しに変換されます。 そのため、ステートメント 1 および 2 は、次の割り当てによって置き換えることができます。

```
name(1) = 'Sherlock Holmes';
```

LOCNEWVALUE が明示的または暗黙的に使用される場合、後続のコードで、より長いストリングを区域内のそのオフセットに割り当てようとしてはなりません。 例えば、以下のステートメントでは、address(1) のオフセットにあるストレージ用に 2 バイトのみが割り振られます。LOCVAL 関数を使用して、より長いストリングを区域内のその位置に割り当てることは無効です。

```
name(1) = 'Shrelock Holmes';
address(1) = '';
```

ただし、次のステートメントを使用して、新規オフセット区域を割り振ることができます。

```
address(1) = '221B Baker Street';
```

未設定の LOCATES 属性を持つオフセット変数に sysnull() を割り当てることをお勧めします。

- sysnull() をオフセット変数に割り当てると、ヌル・ストリングを割り当てる場合よりも、プロセッサーの消費量とストレージの使用量の点でより効率的です。
- オフセットが sysnull() と等しくないことを最初にテストすることによって、LOCVAL 参照が有効かどうかを判別することができます。

LOCNEWVALUE(*v*, *x*) では、*x* の LOCATES 属性の型が ORDINAL または STRUCTURE である場合、*v* は同じ型でなければなりません。

LOCATES 属性の中に STRUCT 型があれば、そのためのスペースを割り振った後に、その LOCATES 属性を、直接その型を持っている LOCATES 属性の場合と同じように参照できます。 次の例を考えてください。

この例は以下の宣言に基づいています。

```
      define struct
        1 ted,
          2 t1 fixed bin(31),
          2 t2 char(12);

       dcl
        1 xmit based(p) unal,
         2 asize   fixed bin(31),
         2 bcount  fixed bin(31),
         2 b(bx refer(bcount))
              offset(a) locates( type ted ),
         2 a     area(ax refer(asize));
```

上記の宣言の場合、以下のステートメントは有効です。

```
      call locnewspace( b(1));

      b(1).t1 = 17;
      b(1).t2 = '314159265358';
```

ただし、b(1) の値は、LOCVAL 疑似変数を使用して割り当てることも可能です。

## 区域変数のための組み込み関数

EMPTY 組み込み関数は、区域変数を初期設定してこれを空にし、それに対して行わ れている可能性があるすべての割り振りを解放します。 これが区域変数の初期状態で、そこにはまだなにも割り振りがされていません。 AVAILABLEAREA 組み込み関数は、その区域内で行うことができる割り振りの最大サイズを返します。

## 区域の割り当て

代入ステートメントを使用して、区域参照の値を 1 つ以上の区域変数に割り当てることができます。 区域から区域への割り当てを行うと、ターゲット区域内のすべての割り振りが解放され、次に、ソース区域のエクステントがターゲット区域に割り当てられ、ソース区域のすべてのオフセットがターゲット区域で有効になります。

次の例を考えてください。

```
   declare X based (O(1)),
           O(2) offset (A),
           (A,B) area;

           alloc X in (A);
           X = 1;
           alloc X in (A) set (O(2));
           O(2) -> X = 2;
           B = A;
```

POINTER 組み込み関数を使用すると、 POINTER (O(2),B)->X と O(2)->X という参照は、それぞれ区域 B と区域 A で割り振られた同じ値を表します。

割り振りを含んでいない区域がターゲット区域に割り当てられると、ターゲット区域内のすべての割り振りが解放されるだけです。

区域割り当てを使用すれば、元の区域の境界を超えて基底付き変数のリストを拡張することができます。 ある区域内に基底付き変数を割り振ろうとしても、その変数を収容するのに必 要なフリー・ストレージがその区域内にないか、またはある区域を別の区域に割り当て ようとして、十分な大きさがないと、AREA 条件が起こります。 この条件の ON ユニットを使用すれば、 不適当な区域の参照を修飾しているポインタ

ーの値を変更して、 別の区域を指すようにすることができます。ON ユニットからの戻りにおいて、 割り振りが新規の区域内で再度試されます。 あるいは、AVAILABLEAREA 組み込み関数を使用して、作成しようとしている割り振りが AREA 条件を発生させずにエリア内で実行できるかどうかを判別することができます。 また、ON ユニットはエリアを書き出して EMPTY にリセットすることができます。

## 区域の入出力

区域の機能を使用すれば、基底付き変数のリスト全体を 1 つの単位として、 RECORD ファイルとの間で入出力を行うことができます。

出力の場合、区域エクステントが、16 バイトの制御情報とともに伝送されます。 ただし、区域が構造体内または共用体内にあり、その中の最後の項目でない場合を除きます。 その場合は、宣言されたサイズが伝送されます。 したがって、区域内の未使用部分がデータ・セットのスペースを取ることは ありません。

区域のエクステントは変化するので、可変長レコードを使用してください。 必要になる最大レコードの長さは、区域の長さ (区域サイズ +16) です。

# リスト処理

リスト処理とは、データの集まりを処理するための多くの手法に付けられた名前です。 配列、構造体、および共用体もデータの集まりを処理するために使われますが、リ スト処理手法は、プログラムの実行中にデータの集まりを無限に再配列したり、 広げたりすることができるという点で、融通性があります。

ここでは、これらの手法については説明しませんが、基底付き変数とロケーター変数がこの種の処理の基礎としてどのような働きをするかを説明します。

リスト処理では、多数の基底付き変数 (それぞれが 1 つ以上の世代を持っている) を 1 つのリスト内に入れることができます。 このリスト内のメンバーは、ほかのメンバーまたはほかのリストの位置を指し 示す 1 つ以上のポインターをメンバー内に持つことによって、相互に関連してい ます。 基底付き変数を割り振るときには、主記憶域内のどこにその変数を割り振るか を指定することはできません (ただし、どの区域内に割り振りたいかを指定する ことはできます)。 実際には、チェーニングしている項目は主記憶域全体にわたって分散していることもあります。しかし、 それぞれのポインターをアクセスすれば、次のメンバーが見つかります。リスト内のメンバーは、通常は、ポインター変数を含む構造体または共用体です。 次の例では、構造体のリストが作成されます。

```
dcl 1 STR based(H),
      2 P pointer,
      2 data,
    T  pointer;

      allocate STR;
      T=H;

      do loop;
        allocate STR set(T->P);
          T=T->P;
          T->P=null;
          .
          .
          .
     end;
```

これらの構造体は、STR のいくつかの世代であり、各世代の中のポイ ンター変数 P によって相互にリンクされます。 リストを作成している間、ポインター変数 T は前の世代を識別します。 最初の ALLOCATE ステートメントは、世代を示すようポインター H をセットします。 つまり、ポインター H はリストの開始点 (リストの先頭) を示します。 2 番目の ALLOCATE ステートメントは、 前の世代の中にあるポインター P が この新しい世代の位置を示すよう、 ポインター P をセットします。 代入ステートメント T=T->P; は ポインター T を更新し、 新規の世代の位置を識別できるようにします。 代入ステートメント T->P=NULL; は、リストの最後の肯定のディレクティブを指定して、 NULL の最後の世代内にポインターを設定します。

257 ページの図 13 は、一方向のチェーンを図解したものです。

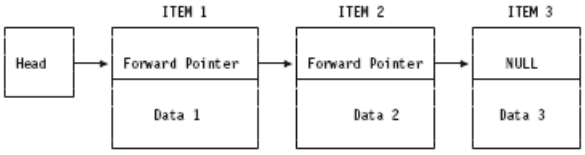



図 *13.* 一方向のチェーンの例

各世代の中にある P の値が、各世代ごとに別々のポインター変数に 割り当てられているのでない限り、STR の各世代は、リストが作成 された際の順序でしかアクセスできません。 上の例の場合、下記のステートメントを使用すれば、各世代を順に アクセスすることができます。

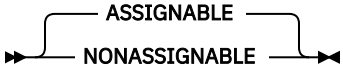
```
   do T=H
     repeat(T->P)
     while (T¬=null);
        .
        .
        .
       T->data;
        .
        .
        .
end;
```

先に挙げたいくつかの例は、一方向のリストを作成する簡単なリスト処理 手法です。別のポインター変数をこの構造体内または共用体内に追加すれば、さらに複雑なリストを作成することができます。 別のポインター変数を追加したとすれば、それで前の世代を指すことができます。それにより、リストは双方向となります。リストの任意の項目から直前の項目と次の項目に、該当するポインター値を使用してアクセスすることができます。 最新のポインター値を NULL の値にセットする代わりに、リスト内の最初の項目を指すようにセットすれば、リング状リスト (循環リスト) を作成することができます。

リストは、1 つの基底付き変数の世代だけで構成する必要はありません。 複数のポインター値を適切にセットすれば、複数の異なる基底付き構造体または 共用体の世代を 1 つのリスト内に入れることができます。それらのポインターの値を操作することによって、リストに項目を追加したり、リストから項目を削除したりすることができます。 ポインターを操作することによってリストを再構成することができ、その結果リスト内のデータの処理が簡素化されます。

## ASSIGNABLE 属性および NONASSIGNABLE 属性

ASSIGNABLE 属性と NONASSIGNABLE 属性は、関連づけられた変数が、割り当てのターゲットとなり得るかどうかを指定します。




**省略形:** ASGN、NONASGN

**デフォルト:** ASSIGNABLE

変数が NONASSIGNABLE 属性を持つ場合には、この変数は割り当てることはできません。

項目記述子に NONASSIGNABLE 属性がある場合、その引数は、対応する ENTRY が 呼び出されても変更されないものと見なされます。 引数が定数であると、仮引数は作成されません。

ASSIGNABLE 属性および NONASSIGNABLE 属性は、構造体または共用体のメンバーに影響を与えます。

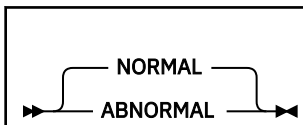
対応するパラメーターに ASSIGNABLE 属性があるときに、NONASSIGNABLE 属性を持つ変数を引数としてエントリーに渡すと、その変数を変更することができ、これによってコードが無効になります。 例えば、以下のコードは保護例外を引き起こす可能性があります。

```
call test( 17 );

test: proc( x );
  dcl x fixed bin(31) NONASSIGNABLE;
  dcl e ext entry( ASSIGNABLE fixed bin(31) );
  call e(x);
end;
```

## NORMAL 属性および ABNORMAL 属性

NORMAL 属性と ABNORMAL 属性は、関連づけられた変数が、いつでも変更の対象となるかどうかを指定します。

ABNORMAL 属性は、変数の値が、複数のステートメント間または 1 つのステート メント内で変更可能であることを指定します。 ABNORMAL 属性を持つ変数は、必要とされるたびにストレージから取り出され、変更 されるたびにストレージに記憶されます。 ABNORMAL 属性を持つ変数の場合、どのような最適化も行うことはできません。



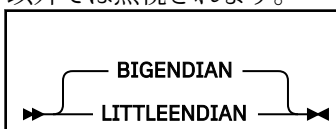

**デフォルト:** NORMAL

NORMAL 属性および ABNORMAL 属性は、構造体または共用体のメンバーに影響を与えます。

ABNORMAL 属性が、INITIAL 値を持つ INTERNAL STATIC 変数に適用される 場合、その変数 (INITIAL 値とともに) は、他の用途に使用されていない 場合でも、生成オブジェクト・コードに表示されます。

## BIGENDIAN 属性および LITTLEENDIAN 属性

BIGENDIAN 属性と LITTLEENDIAN 属性は、関連した変数を格納する際に、最上位数字または最下位数字のどちらを最初に格納するかを指定します。 BIGENDIAN 属性と LITTLEENDIAN 属性は、FIXED BINARY 変数、ORDINAL 変数、OFFSET 変数、AREA 変数、VARYING ストリング変数、VARYING4 ストリング変数以外では無視されます。




**デフォルト:** BIGENDIAN。ただし、Intel の場合は LITTLEENDIAN がデフォルトです。

**BIGENDIAN**
変数 (可変ストリングの場合、変数の長さを表す接頭部) が格納されるときに、最上位バイトが最初に格納されることを表します。 このフォーマットは、z/OS と RS/6000 に固有のスタイルです。

**LITTLEENDIAN**
それとは逆に変数の最下位バイトが最初に格納されることを表します。 このフォーマットは、Windows に固有のスタイルです。

AREA 変数に LITTLEENDIAN または BIGENDIAN 属性が設定されると、コンパイラーおよび ライブラリーによって管理される制御値を保持するフォーマットだけに影響を与えます。 AREA 変数に格納されているユーザー変数、または AREA 変数のユーザー変数を 指すのに使われるオフセット変数には、影響しません。

次の例は、BIGENDIAN 変数と LITTLEENDIAN 変数が保管される方法を示したものです。

X および Y が以下のように宣言されているとします。

```
dcl X fixed bin(15) bigendian;
dcl Y fixed bin(15) littleendian;
```

```
X = 258;
Y = 258;
```

HEXIMAGE 組み込み関数は、X と Y が、 実際に保管される方法を明らかにします。

```
display( heximage( addr(X), stg(X) ) );     /* displays 0102 */
display( heximage( addr(Y), stg(Y) ) );     /* displays 0201 */
```

一方、HEX 組み込み関数は、X および Y のストレージ表現を示します。

```
display (hex(X));                            /* displays 0102   */
display (hex(Y));                            /* displays 0102   */
```

BIGENDIAN と LITTLEENDIAN は、命令のセマンティクスにはなにも影響を及ぼさず、また変数のストレージ所要量にも影響しません。

BIGENDIAN 属性と LITTLEENDIAN 属性は、構造体または共用体のメンバーに影響を与えます。

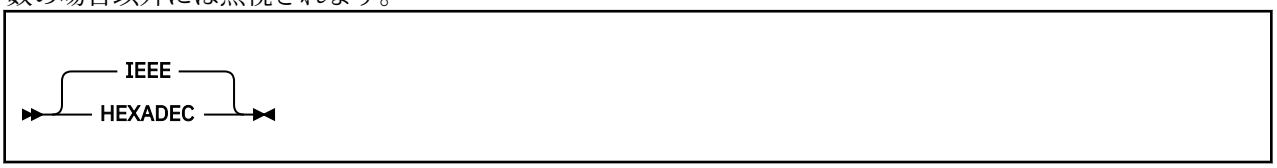
BIGENDIAN と LITTLEENDIAN の使用に関する詳細は、「プログラミング・ガイド」を参照してください。

NATIVE 属性と NONNATIVE 属性は、BIGENDIAN と LITTLEENDIAN の同義語ですが、 これらの属性が BIGENDIAN と LITTLEENDIAN のどちらを意味するかは、システムによって異なります。

- z/OS と RS/600 では、NATIVE は BIGENDIAN を意味します。
- Windows では、NATIVE は LITTLEENDIAN を意味します。

## HEXADEC 属性と IEEE 属性

HEXADEC 属性および IEEE 属性は、関連した変数を保管する際に、IBM 16 進浮動小数点フォーマットを使用するか、IEEE フォーマットを使用するかを指定します。 HEXADEC 属性と IEEE 属性は、浮動小数点変数の場合以外には無視されます。




**デフォルト:** IEEE。ただし、z/OS の場合は HEXADEC がデフォルトです。

**HEXADEC**
変数が 16 進数 (z/OS) フォーマットで保管されることを表します。

**IEEE**
変数が IEEE フォーマットを使用して保管されることを表します。

DEFAULT コンパイラー・オプションの HEXADEC と IEEE サブオプションを使用して、この属性のデフォルトを変更することができます。

Windows および AIX プラットフォームでは、すべての計算は、IEEE の浮動小数点を使用して行われるので、HEXADEC で宣言された変数は必要に応じて IEEE に変換されます。

z/OS プラットフォームでは、浮動小数点の計算は、次の 3 セットの浮動小数点命令を使用して行うことができます。

- IBM 16 進浮動小数点
- IEEE 2 進浮動小数点
- IEEE 10 進浮動小数点

z/OS プラットフォームで浮動小数点の計算にどのセットの命令を使用するかは、次の 2 つのコンパイラー・オプションによって決まります。

- FLOAT(DFP) を指定した場合
  - 結果が FLOAT DEC となる計算はすべて、IEEE 10 進浮動小数点命令を使用して行われます。

  - 結果が FLOAT BIN となる計算はすべて、DEFAULT コンパイラー・オプションの HEXADEC および IEEE サブオプションで指定されたフォーマット用の浮動小数点命令を使用して行われます。
- FLOAT(NODFP) を指定した場合
  - 結果が FLOAT となる計算はすべて、DEFAULT コンパイラー・オプションの HEXADEC および IEEE サブオプションで指定されたフォーマット用の浮動小数点命令を使用して行われます。

したがって、FLOAT(NODFP) および DEFAULT(HEXADEC) オプションのもとでは、すべての計算は 16 進浮動小数点命令を使用して行われ、IEEE で宣言された変数は HEXADEC に変換されます。 一方、FLOAT(NODFP) および DEFAULT(IEEE) オプションのもとでは、すべての計算は、IEEE 2 進浮動小数点命令を使用して行われ、HEXADEC で宣言された変数は必要に応じて IEEE に変換されます。

FLOAT(DFP) コンパイラー・オプションのもとでは、IEEE および HEXADEC 属性は FLOAT BIN でのみ有効で、DEFAULT(IEEE/HEXADEC) オプションは FLOAT BIN にのみ適用されます。

## CONNECTED 属性と NONCONNECTED 属性

CONNECTED 属性は、パラメーターが連結ストレージのみに対する参照であることを指定します。 NONCONNECTED 属性は、パラメーターが不連続ストレージおよび連続ストレージを占有することを許可します。

エレメント、配列、および大構造体または複数の共用体は、常に連結ストレージに割り振られます。 非連結ストレージへの参照は、より大きな集合からの不連続項目から構成される集合へ参照を行う場合にのみ行われます (174 ページの『配列のクロスセクション』 を参照してください。) 例えば、次の構造体では、介在配列 `A.B` および `A.C` は、 両方とも非連結ストレージにあります。

```
1 A(10),
  2 B,
  2 C;
```

**省略形:** CONN、NONCONN

**デフォルト:** NONCONNECTED

CONNECTED 属性は、被制御集合パラメーターにしか適用されず、レベル 1 の 名前でしか指定することができません。 この属性によって、パラメーターの参照が連結ストレージに限られます。 そのため、パラメーターをレコード単位の入出力のターゲットまたはソースとして使 用するか、もしくはストリング・オーバーレイ定義の基数として使用することができ ます。 パラメーターが連結であり、CONNECTED 属性が使用される場合、この連結パラメ ーターへの参照用に、より効率の良いオブジェクト・コードが生成されます。

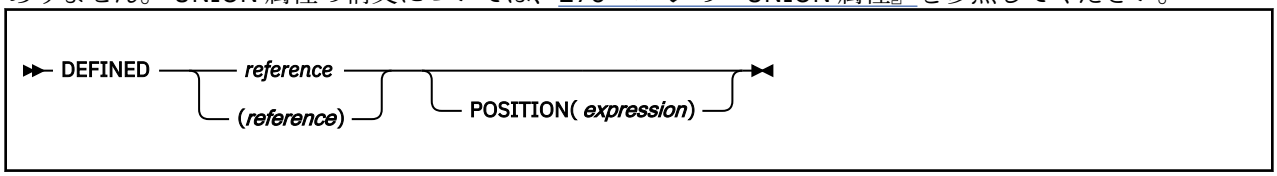
パラメーターが不連続ストレージを占有している場合、NONCONNECTED を指定しなければなりません。 次の例では、NONCONNECTED 属性は、`sum_Slice` ルーチンが、連続していない可能性のあるエレメントを持つ 1 次元の配列を処理することを示しています。 最初の呼び出しでは、`sum_Slice` は最初の行に渡されます。これは連結ストレージにあります。 しかし、2 番目の呼び出しでは、`sum_Slice` は最初の列に渡されます。これは非連結ストレージにあります。

```
dcl A(10,10) fixed bin(31);

display( sum_Slice( A(1,*) ) );     /* first row    */
display( sum_Slice( A(*,1) ) );     /* first column */

sum_Slice:proc(X) returns(fixed bin(31));

  dcl X (*) fixed bin(31) nonconnected; /* default */
return(sum(X) );
end;
```

# DEFINED 属性および POSITION 属性

DEFINED 属性は、宣言される変数が、指定された基本変数のストレージの 一部または全体に関連付けられることを指定します。

UNION 属性を指定すると、より明快な方法で同じ目的を達することができ、 また属性や精度の異なる変数にオーバーレイすることもできます。 また、DEFINED 属性では、定義済み変数または基本変数によるアクセスがすべて の定義済み変数で反映されますが、 共用体では、共用体のメンバーは常に 1 つしか有効ではありません。 UNION 属性の構文については、176 ページの『UNION 属性』を参照してください。




**省略形:** DEF (DEFINED の場合)、POS (POSITION の場合)

**参照**
宣言される変数に関連付けられるストレージを持つ 変数 (基本変数) に対する参照。 宣言される変数は、定義済み変数 です。 基本変数は EXTERNAL でも INTERNAL でもよく、 パラメーターでも構いません (ストリング・オーバーレイ定義の 中で、パラメーターは連結ストレージを参照している必要があります)。 ただし、BASED または DEFINED であってはなりません。 基本変数の値を変更すると、定義済み変数の値がそれに応じて変更されます。 その逆も同様です。

基本変数がデータ集合である場合、定義済み変数はデータ全体で構成するこ とも、データの指定部分だけで構成することもできます。

定義済み変数は、基本変数の属性を継承しません。 定義済み変数は INTERNAL にする必要があり、レベル 1 の ID で なければなりません。 この変数は次元属性を持つことができます。 属性は INITIAL、AUTOMATIC、BASED、CONTROLLED、 STATIC、または パラメーターであってはなりません。

単純、iSUB、およびストリング・オーバーレイの 3 つの定義タイプがあります。

有効になる定義タイプは、次のように決定されます。

1. POSITION 属性が指定されている場合は、ストリング・オーバーレイ 定義が有効になります。
2. 基本変数に指定されている添え字に iSUB 変数への参照が 含まれている場合、iSUB 定義が有効になります。
3. iSUB 変数も POSITION 属性も存在せず、基本変数と定義済み変数が後述の基準に従って一致している場合は、単純定義が有効になります。
4. どれも当てはまらない場合は、ストリング・オーバーレイ定義が有効になります。

POSITION 属性が指定されている場合は、基本変数に iSUB 参照が 含まれていてはなりません。

基本変数と定義済み変数が一致 するのは、 引数として渡されたときの基本変数が、定義済み変数の属性 (DEFINED 属性 を除く) を持つパラメーターと一致している場合です。 このためには、パラメーターの配列の境界、ストリングの長さ、 および区域サイズがすべてアスタリスクによって指定されていることが 前提になります。

単純定義と iSUB 定義の場合、PICTURE 属性は同一の PICTURE 属性 (反復因数を除く) にのみ一致します。ストリング・オーバーレイ定義内で参照が有効な基本変数を指定するためには、 参照が連結ストレージ内にあることが必要です。 マッチング規則は完全にオーバーライドできますが、 オーバーライドによってプログラムに意図しない副次作用が出る可能性があります。

配列の境界、ストリングの長さ、または区域サイズに対して指定される値、または 派生する値は、基本変数の値と必ずしも一致していなくても構いません。 ただし、定義済み変数は、対応する基本配列、ストリング、または 区域に収まるようにする必要があります。

定義済みデータを参照する際には、STRINGRANGE、SUBSCRIPTRANGE、およ び STRINGSIZE の条件が、基本変数ではなく、定義済み変数の配列境界とストリング の長さに生じます。

値の決定と名前の解釈は、次の順序で行われます。

1. 変数を宣言するブロックに入ったときに、 定義済み変数の配列境界、ストリングの長さ、および区域サイズが計算されます。
2. 定義済み変数を参照するということは、基本変数の現行世代を参照することです。 仮引数が作成されずに、定義済み変数が引数として渡されると、それに対応する パラメーターは、引数が渡された時点での基本変数の現行世代を参照します。 呼び込まれたプロシージャー内で基本変数が再割り振りされた場合にも、 このことは引き続き当てはまります。
3. 定義済み変数への参照が行われたときに、基本変数と定義済み変数の添え字を評価する順序は定義されていません。

定義済み変数が、位置合わせされていない固定長 BIT であるエレメントが入っている構造体または共用体である場合は、 定義済み変数の中のすべての配列境界およびストリング長を、定数として指定する必要があります。

定義済み変数に BIT 属性がある場合、 次の条件下では予期しない結果が生じる可能性があります。

- 基本変数がバイト境界上にない。
- 定義済み変数が基本変数の最初の位置に定義されておらず、定義済み変数が以下として使用されている場合。
  - サブルーチン呼び出しのパラメーター (つまり、 内部保管データとして参照されている)
  - PUT ステートメントの引数
  - 組み込み関数の引数 (ライブラリー呼び出し)
  - 基本変数が被制御であり、定義済み変数が 次元付きで可変の配列境界を指定して宣言されている場合
- 定義済み変数全体が、位置合わせされていない固定長ビット・ストリングで構成されている場合は、定義済み変数の配列境界、ストリング長、および区域サイズがコンパイル時に分かっている必要がある。

## 非連結ストレージ

DEFINED 属性は配列をオーバーレイできます。 このため、配列式は非連結ストレージ内の配列エレメント (ストレージ内で 隣接していない配列エレメント) を参照できます。

次の場合に、連続したエレメントを含む配列式が 非連結ストレージを参照できます。

- ストリング配列を、より長いエレメントを持つストリング配列に対して 定義する場合。 定義済み配列の連続エレメントは、基本配列と定義済み配列の エレメントの長さに差があるために分離され、非連結ストレージに 保管されます。

別の配列に対してオーバーレイ定義された配列は、 常に非連結ストレージに入れられることが前提になります。

## 単純定義

単純定義によって、要素変数、配列変数、または構造変数を 別の名前で参照できます。

単純定義は、スカラー、基本変数内の定数エクステントと一致する定数エクステントの構造体、およびこのようなスカラーと構造体の配列 (ただし、これらが被制御変数に基づいていない場合) についてのみサポートされます。

定義済み変数と基本変数は、任意のデータ・タイプで構成できますが、 一致している必要があります。 定義済み変数のエレメントと、対応する基本変数のエレメントごとに、ALIGNED 属性と UNALIGNED 属性が一致している必要があります。

定義済み変数は次元属性を持つことができます。

配列の単純定義では、以下の条件が適用されます。

- 基本変数には、配列のクロスセクションを指定できます。
- 定義済み変数に対して指定する次元数は、 基本変数に対して指定する次元数と等しくなければなりません。
- 定義済み配列の境界の対で指定される範囲は、対応する基本配列の境界の対で指定される範囲と等しいか、その範囲内に含まれる必要があります。

ストリングの単純定義では、定義済みストリングの長さは 基本ストリングの長さ以下であることが必要です。

区域の単純定義では、定義済み区域のサイズは基本区域のサイズと 等しくなければなりません。

定義済み変数の対応部分が同じ最大長の可変ストリングであれば、基本変数は可変ストリングであっても、また可変ストリングを含んで いても構いません。

**例**

```
DCL A(10,10,10),
    X1(2,2,2) DEF A,
    X2(10,10) DEF A(*,*,5),
    X3 DEF A(L,M,N);
```

X1 は、A のそれぞれの行、列、および平面のうち、最初の 2 つのエレメントからなる 3 次元配列です。 X2 は、A の 5 つ目の平面からなる 2 次元配列です。 X3 は、添え字式 L、M、および N によって識別されるエレメントからなるエレメントです。

```
DCL B CHAR(10),
    Y CHAR(5) DEF B;
```

Y は、B の最初の 5 文字からなる文字ストリングです。

```
DCL C AREA(500),
    Z AREA(500) DEF C;
```

Z は、C 上で定義される区域です。

```
DCL 1 D UNALIGNED,
         2 E,
         2 F,
              3 G CHAR(10) VAR,
              3 H,
    1 S UNALIGNED DEF D,
         2 T,
         2 U,
              3 V CHAR(10) VAR,
              3 W;
```

S は D で定義される構造体です。 単純定義の場合、2 つの構造体の編成は同一でなければなりません。 T への参照は E への参照になり、V は G に、その他も同様です。

## iSUB 定義

iSUB 定義を使用すると、基本配列の指定したエレメント からなる定義済み配列を作成できます。

定義済み配列と基本配列はスカラー配列でなければならず、任意のデータ・タイプで構成でき、同一の属性を持つ必要があります (次元属性を除く)。

定義済み変数には、次元属性が必要です。 定義済み配列の宣言の中で、 基本配列には添え字を付ける必要があり、添え字の位置をアスタリスクとして 指定することはできません。

iSUB 変数は、基本配列の添え字リスト内での、定義済み配列の次元に対する参照です。 基本配列の添え字リストにある少なくとも 1 つの添え字は、iSUB 式でなければなりません。この式は、評価時に基本配列に必要な添え字を指定します。 i の値は、*n* を定義済み配列の次元数 として、1 から *n* までの範囲です。 基本配列の添え字の数は、基本配列の次元数と等しくなければなりません。

定義済み配列への参照が添え字式を指定していない場合、 添え字の評価は、参照を含む式または割り当ての評価時に行われます。

i の値は整数として指定されます。 iSUB 式の中で、iSUB 変数は REAL FIXED BINARY(31,0) 変数として 扱われます。

定義済み変数への参照にある添え字は、基本変数の添え字リストに 対応する iSUB がない場合でも評価されます。

iSUB 定義済み変数は、GET DATA ステートメントまたは PUT DATA ステートメントのデータ・リストに入れてはなりません。

**例**

```
DCL A(10,10) FIXED BIN
    X(10) FIXED BIN DEF( A(1SUB,1SUB) );
```

X は、A の対角線で構成される 1 次元配列です。 X(i) は、A(i,i) と同じストレージを参照します。

```
DCL B(5,10) FIXED BIN
    Y(10,5) FIXED BIN DEF( A(2SUB,1SUB) );
```

Y は、境界を転置した B のエレメントで構成される 2 次元配列です。 Y(i,j) と X(j,i) は同じストレージを参照します。

## ストリング・オーバーレイ定義

ストリング・オーバーレイ定義を使用すると、 定義済み変数を基本変数のストレージに関連付けることができます。 定義済み変数と基本変数は、両方ともストリングまたは ピクチャー・データでなければなりません。

定義済み変数でも基本変数でも、ALIGNED 属性、VARYING 属性、VARYING4 属性は設定できません。

定義済み変数および基本変数はいずれも、以下のいずれかのクラスに属していなければなりません。

- 以下の変数からなるビット・クラス。

  固定長のビット変数
  固定長のビット変数の集合体
- 以下の変数からなる文字クラス。

  固定長の文字変数
  文字ピクチャー変数または数字ピクチャー変数
  上記 2 つの集合体
- 以下の変数からなるグラフィック・クラス。

  固定長のグラフィック変数
  固定長のグラフィック変数の集合体
- 以下の変数からなる uchar クラス。

  固定長の uchar 変数
  固定長の uchar 変数の集合体
- 以下の変数からなるワイド文字クラス。

  固定長のワイド文字変数
  固定長のワイド文字変数の集合体

**例**

```
DCL A CHAR(100),
    V(10,10) CHAR(1) DEF A;
```

V は、文字ストリング A のすべてのエレメントからなる 2 次元の配列です。

```
DCL B(10) CHAR(1),
    W CHAR(10) DEF B;
```

W は、配列 B のすべてのエレメントからなる文字ストリングです。

## POSITION 属性

POSITION 属性はストリング・オーバーレイ定義だけで使用でき、定義済み変数の先頭になる基本変数内の文字、ビット、グラフィック、uchar、またはワイド文字を指定します。

POSITION 属性の式は、基本変数の先頭からの相対的な位置を指定します。 式で指定できる値の範囲は 1 から *n* までで、*n* は次のように定義されます。

```
n = N(b) - N(d) + 1
```

ここで、N(b) は基本変数の文字、ビット、グラフィック、uchar、またはワイド文字の数、N(d) は定義済み変数の文字、ビット、グラフィック、uchar、またはワイド文字の数です。

定義済み項目を参照するたびに、この式が計算され、整数値に変換されます。

POSITION 属性を省略すると、POSITION(1) と見なされます。

定義済み変数がビット・クラス集合の場合、POSITION 属性には整数のみを入れることができ、基本変数は添え字付きであってはなりません。

基本変数は、連結ストレージ内のデータを参照している必要があります。

**例**

```
DCL C(10,10) BIT(1),
    X BIT(40) DEF C POS(20);
```

X は、C の 20 番目のエレメントから 40 個の エレメントで構成されるビット・ストリングです。

```
DCL E PIC'99V.999',
    Z1(6) CHAR(1) DEF (E),
    Z2 CHAR(3) DEF (E) POS(4),
    Z3(4) CHAR(1) DEF (E) POS(2);
```

Z1 は、10 進数ピクチャー E のすべてのエレメントからなる文字ストリング配列です。 Z2 は、ピクチャー E のエレメント '999' からなる文字ストリングです。 Z3 は、ピクチャー E のエレメント '9.99' からなる文字ストリング配列です。

```
DCL A(20) CHAR(10),
    B(10) CHAR(5) DEF (A) POSITION(1);
```

B の最初の 50 文字は、A の最初の 50 文字で構成されます。 POSITION(1) を明示的に指定する必要があります。 そうしない場合は、単純定義が使用され、結果は異なったものになります。

# INITIAL 属性

INITIAL 属性は、変数にストレージが割り振られるときにその変数に 割り当てられる初期値を指定するものです。

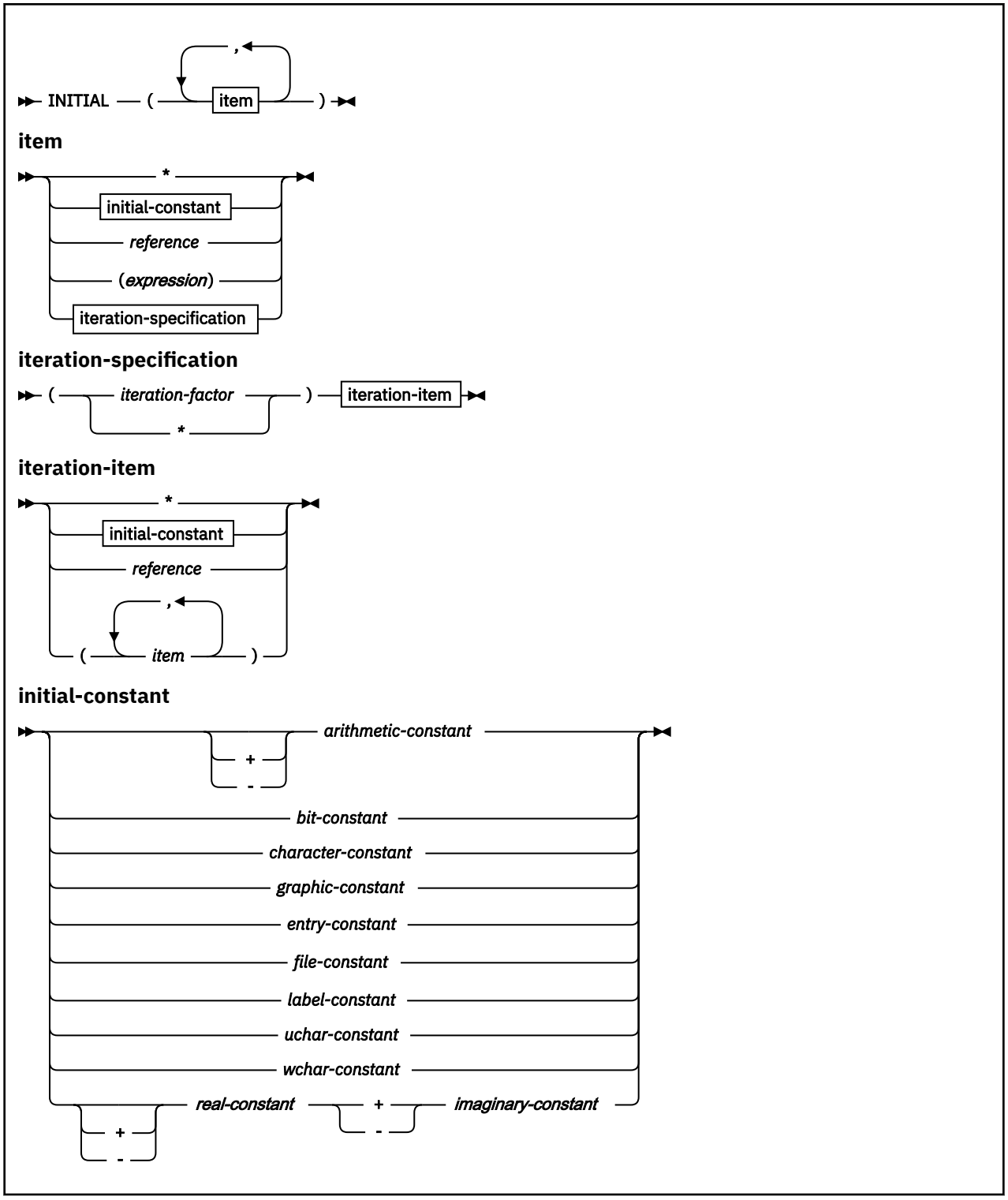
要素変数には、初期値を 1 つしか指定することができません。 配列変数には複数の初期値を指定することができます。 共用体変数は、要素変数か配列変数かにかかわらず、その基本名を個別に初期化するという方法でのみ初期化することができます。 構造体型が定義された変数は、VALUE 型関数を INITIAL 属性と共に使用することで初期化できます。 定数、定義済みデータ、制御なしパラメーター、および非 LIMITED 静的入り口変数 に、INITIAL 属性を与えることはできません。

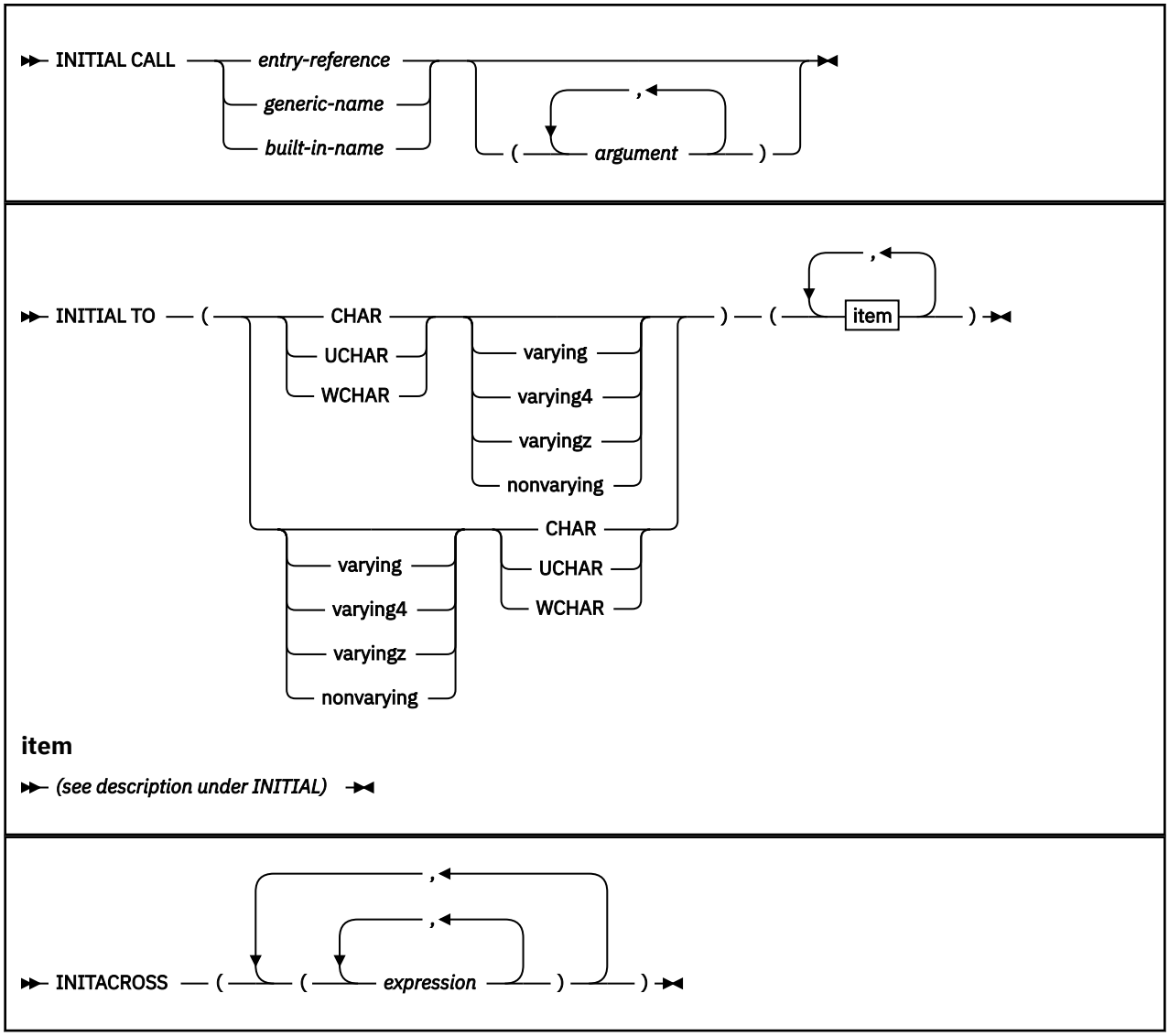
INITIAL 属性には以下の形式があります。

1. 最初の形式 INITIAL では、初期定数、式、または関数参照を指定します。 その値が、ストレージ割り振り時に変数に割り当てられます。
2. 2 番目の形式 INITIAL CALL では、初期化を実行するためにプロシージャーを呼び出すことを (CALL オプションで) 指定します。 変数は、呼び出されたルーチンの実行中に割り当てによって初期化されます。 (ルーチンは、呼び出し点に値を戻す関数としては呼び出されません。)

3. 3 番目の形式 INITIAL TO では、ポインター (またはポインターの配列) が INITIAL LIST に指定されたストリングのアドレスで初期化されることを指定します。 また、ストリングにも TO キーワードで示された属性があります。

INITIAL 形式は DEFINE STRUCTURE ステートメントの基本名で使用できますが、INITIAL CALL 形式および INITIAL TO 形式は使用できません。 型付き変数の初期化について詳しくは、578 ページの『VALUE』を参照してください。






**省略形:** INIT、INIT CALL、INIT TO

*****
エレメントを、反復因数として使用する場合を除き、初期化しないままにしておくことを指定します。

**反復因数 (iteration factor)**
配列のエレメントを初期設定するときに、項目を何回反復するかを指定します。

反復因数は、式であってもアスタリスクであってもかまいません。

- 式は、FIXED BINARY(31) に変換されます。 静的変数の場合は、定数でなければなりません。
- アスタリスクは、残りのエレメントを指定された値に初期設定しなければならないことを示します。

反復因数と初期値の両方にアスタリスクを使用することはできません。

反復因数に負またはゼロの値を指定すると、初期設定は行われません。

**定数 (constant)**
**参照**
**expression**
これらは、初期設定される変数に割り当てられる初期値を指定します。

**INITIAL CALL**
INITIAL CALL では、渡される入り口参照および引数リストは、89 ページの『ブロックの活動化』に記載されているような、ブロックを活動化するために 設定された条件を、満たしていなければなりません。

INITIAL CALL は、静的データの初期設定に使用することはできません。

次の例では、各エレメントに INITIAL 属性を必要とせずに、A のエレメントがすべて '00'X に初期設定されています。

```
dcl  1  A automatic,
       2  …,
       2  …,
       2  * char(0) initial call plifill( addr(A), '00'X, stg(A) );
```

INITIAL CALL 属性を持つ AUTOMATIC 変数は、 他の用途に使用されない場合でも保存されます (プログラムのロジック によってコールを実行する必要が生じた場合に備えて)。

INITIAL CALL ステートメントによって呼び出されたプロシージャーが、 FETCH ステートメントまたは RELEASE ステートメントで指定されており、 主記憶域に存在しない場合、INITIAL CALL ステートメントは、そのプロシージャーの 動的ロードを開始します。 (動的ロードについて詳しくは、100 ページの『外部プロシージャーの動的ロード』を参照してください。)

**INITIAL TO**
静的固有のポインターにのみ使用してください。 ポインター (または、ポインターの配列) が INITIAL LIST で指定されたストリングのアドレスで初期化されることを指定します。 また、ストリングに TO キーワードで示された属性があることも指定します。

次の例では、pdays は曜日を含む varyingz ストリングのア ドレスの文字で初期化されます。

```
 dcl pdays(7) static ptr init to(varyingz)
                ('Sunday',
                 'Monday',
                 'Tuesday',
                 'Wednesday',
                 'Thursday',
                 'Friday',
                 'Saturday'  );
```

INITIAL TO で初期化されたポインターによって識別された値を変更しないでください。 この値を読み取り専用のストレージに入れることができますが、値を変更しようとすると 記憶保護例外になることがあります。 上の例にある配列 pdays の場合、以下の割り当ては正しくありません。

```
dcl x char(30) varz based;

pdays(1)->x = 'Sonntag';
```

**INITACROSS**

INITACROSS 属性は、すべての構造体メンバーがスカラーである構造体の 1 次元配列へのエレメントの追加または削除を容易にすることで、それらの配列の初期化に役立ちます。 この属性は、一連の式のカンマリストを指定します。各カンマリストは、配列内の構造体エレメントの初期値のセットを順に指定します。

例えば、以下の宣言について考えてみます。

```
             dcl
               1 a(3)
                 ,2 b     char(2)
                                 init( 'DE', 'FR', 'SP' )
                 ,2 c     char(40) var
                                 init( 'Germany', 'France', 'Spain' )
               ;
```

INITACROSS を使用すると、次のように記述することによって、この宣言を単純化することができます。

```
             dcl
               1 a(3)
                         initacross(  ( 'DE', 'Germany' )
                                     ,( 'FE', 'France' )
                                     ,( 'SP', 'Spain' )
                                   )
```

```
  ,2 b     char(2)
  ,2 c     char(40) var
;
```

DIMACROSS と組み合わせると、この宣言にエレメントを追加することがさらに容易になります。

```
dcl
  1 a(*)   dimacross
           initacross(  ( 'DE', 'Germany' )
                      ,( 'FE', 'France' )
                      ,( 'SP', 'Spain' )
                    )
  ,2 b     char(2)
  ,2 c     char(40) var
;
```

制約事項:

- INITACROSS 属性と INITIAL 属性は、同じ名前に対して指定してはなりません。
- INITACROSS 属性は、継承されたディメンションを持たない構造体の 1 次元配列にのみ適用する必要があります。
- INITACROSS 構造体メンバーはすべて、INITIAL 属性のないスカラーでなければなりません (INITACROSS 属性によって INITIAL 属性が与えられるため)。
- 式の各カンマリスト内の式の数は一致する必要があります。 次のものは無効です

```
    initacross(  ( 'DE', 'Germany', 'Berlin' )
               ,( 'FR', 'France', 'Paris' )
               ,( 'SP', 'Spain' )
             )
```

- INITACROSSS 構造体の明示的なメンバー数は、INITACROSS の各カンマリストの要素数で定義される暗黙的なメンバー数としなければならない。 例えば、この (無効な) 宣言では、構造体の明示的なメンバー・カウントは 2 ですが (A にはメンバー B と C があるため)、暗黙的なメンバー・カウントは 3 です (各式の k カンマカンマリストは、「DE」、「Germany」、および「ベルリン」などの 3 つの要素で構成されているため)。

```
dcl
  1 a(3)
           initacross(  ( 'DE', 'Germany', 'Berlin' )
                      ,( 'FR', 'France', 'Paris' )
                      ,( 'SP', 'Spain', 'Madrid' )
                    )
  ,2 b     char(2)
  ,2 c     char(40) var
;
```

## 配列変数の初期設定

配列に指定した初期値は、行を主体とした順序で (最後の添え字が最も頻繁に変化する)、配列の連続している各エレメントに割り当てられます。 指定されている初期値が多過ぎる場合は、超過している値は無視されます。指定が十分でない場合は、残りの配列は初期設定されません。

ストリングの配列を初期設定するときは、ストリング反復因数と反復因数の両方を指定することができます。 どちらか一方だけを指定したときは、ストリング定数が括弧で囲まれていない限り、ストリング反復因数と見なされます。

反復因数は * で指定できます。この場合は、残りのエレメントがすべて指定された値で初期設定されます。

以下の例は、ストリング反復因数と反復因数の使用法 (および両者の違い) を 示しています。

((2)'A') は ('AA') と同等です
((2)('A')) は ('A','A') と同等です
((2)(1)'A') は ('A','A') と同等です
((*)(1)'A') は ('A','A'…'A') と同等です

区域変数は、割り振られると EMPTY 組み込み関数の値で初期設定されます。 区域変数の INITIAL 文節は無視されます。

INITIAL 属性内の項目の属性とデータ項目自体の属性とが異なっている場合は、 それらの属性の間に互換性がある限りにおいて、変換が行われます。

INITIAL 属性は、REFER 文節の対象として使用することはできません。

## 共用体の初期設定

共用体メンバーは、初期値を持つことができます。 ただし、共用体が静的である場合、INITIAL 属性を持つことができる共用体のメンバーは 1 つだけです。 共用体が静的でない場合、INITIAL 属性は指定された順に適用されます。 後続の初期値は、その前の値を重ね書きします。

次の例では、NT1 の宣言は、これが静的ストレージ属性を持つ場合は無効になります。

```
dcl
  1 NT1 union automatic,
    2 Numeric_translate_table1 char(256)
                  init( (256)'00'X),
    2 *,
      3 * char(240),
      3 * char(10) init('0123456789'),
    2 * char(0);

dcl
  1 NT2 union static,
    2 Numeric_translate_table2 char(256),
    2 *,
      3 * char(        rank('0')    )
          init((1)(low(rank('0'))) ),
      3 * char(10) init('0123456789'),
      3 * char(         (256-(rank('0'))-10)    )
          init((1)(low( (256-(rank('0'))-10) )) ),
```

NT2 の宣言は、これが静的ストレージ・クラスを持っていても有効です。 さらに、NT2 宣言は、EBCDIC の実行モードと ASCII の実行モードの間で移植性があります。

## 静的変数の初期設定

プログラムのロード時に割り振られる変数 (つまり、静的変数) の場合、その変数はプログラムの実行中割り振られたままになっているので、INITIAL 属性で指定した値は 1 回だけ割り当てられます。 (フェッチされるプロシージャーの静的ストレージは、そのプロシージャーがロードさ れるたびに割り振られ、初期設定されます。)

静的変数が INITIAL 属性を使用して初期設定される場合、その初期値は制限付き 式として指定されなければなりません。 エクステントの指定は、制限付き式でなければなりません。

静的変数を初期設定する際の制約事項は、次のとおりです。

- STATIC ENTRY 変数は、LIMITED 属性を持たなければなりません。
- INITIAL は、静的フォーマット変数には使用することができません。
- INITIAL は、構造体または共用体の一部ではないラベル変数に使用することができます。ただし、以下の制限が適用されます。 INITIAL を使用する場合、ラベル変数は CONSTANT 属性になります。
  - すべての初期値は、添え字なしのユーザー・ステートメント・ラベルでなければなりません。
  - すべての初期値がラベル宣言と同じブロック内になければなりません。
  - ラベルが配列の場合は、完全に初期化されている必要があります。
- INITIAL は、AREA 変数には無効です。
- 静的共用体のただ 1 つのメンバーだけが、INITIAL を指定できます。
- RESERVED 属性を持たない STATIC EXTERNAL 項目が、複数の 宣言で INITIAL 属性を指定された場合、指定された値はどのような場合でも同じでなけ ればなりません。

## 自動変数の初期設定

自動変数の場合、その変数は、それを宣言しているブロックが活動化されるたびに割り振られるので、指定した初期値は割り振りのたびに割り当てられます。

## 基底付き変数と被制御変数の初期設定

基底付き変数と被制御変数の場合、その変数は ALLOCATE ステートメント (基 底付き変数のときは LOCATE ステートメントも可) の実行時に割り振られるので、 指定した初期値は割り振りのたびに割り当てられます。

ALLOCATE 組み込み関数または AUTOMATIC 組み込み関数を使用して基底付き変数のストレージを割り振る場合、初期値は割り当てられません。 区域変数の場合、区域は暗黙には EMPTY に初期設定されません。

## 例

以下の例は、割り振り時に変数を初期設定する方法を示しています。

次の例では、Name にストレージが割り振られたときに、 文字定数 'John Doe' (右側にブランクが埋め込まれて 10 文字になる) が Name に割り当てられます。

```
dcl Name char(10) init('John Doe');
```

次の例では、Pi が割り振られたときに、それに 3.1416 という値が初期設定されます。

```
dcl Pi fixed dec(5,4) init(3.1416);
```

次の例では、式 B*C の値で A を初期設定することを指定しています。

```
declare A init((B*C));
```

以下の例では、A の最初の 920 個のエレメントがそれぞれ 0 に設定されます。 次の 80 個のエレメントは、シーケンス 5,5,5,9 の 20 回の繰り返しで構成されます。

```
declare A (100,10) initial
        ((920)0, (20) ((3)5,9));
```

次の例では、A の第 1、第 3、 および第 4 のエレメントが初期設定されます。 配列のその他の部分は初期設定されません。 配列 B は全体が初期設定され、最初の 25 個の エレメントは 0 に、次の 25 個は 1 に、残りのエレメントは 0 にそれぞれセットされます。 構造体 C では、次元 (8) が D と E に 継承されており、D の最初のエレメントだけが初期設定されます。 E は全部のエレメントが初期設定されます。

```
declare A(15) character(13) initial
          ('John Doe',
           *,
           'Richard Row',
           'Mary Smith'),

        B (10,10) decimal fixed(5)
          init((25)0,(25)1,(*)0),

        1 C(8),
          2 D initial (0),
          2 E initial((*)0);
```

構造体または共用体の配列を宣言するときに、別の構造体または共用体と同じ構 造体フォーマットにするための LIKE 属性を指定した場合、その別の構造体または 共用体のエレメントがすでに初期設定されていると、この構造体または共用体の配列における最初 の構造体または共用体だけが初期設定されます。

次の例では、構造体の配列において J(1).H と J(1).I だけが初期設定されます。

```
declare 1 G,
          2 H initial(0),
          2 I initial(0),
        1 J(8) like G;
```

# 第 10 章 入出力

PL/I の入出力ステートメント (READ、WRITE、GET、 PUT など) を使用すれば、 計算機の主記憶装置と補助記憶装置との間でデータを伝送することができます。

プログラムの外部にあるデータの集合をデータ・セット といいます。 データ・セットからプログラムへデータを伝送することを入力 といい、プログラムからデータ・セットへデータを伝送することを出力 といいます。(端末を使用している場合、"データ・セット"は端末を意味することもあります。)

PL/I の入出力ステートメントは、 データ・セットの論理的な編成に関係するものであり、 データ・セットの物理特性とは関係がありません。 したがって、プログラムの実行時に使用される入出力装置についての特別な知 識がなくても、プログラムを設計することができます。 データ・セット内の物理編成ではなく、データの論理的な側面を主眼にソース・プログラムを処理することができるように、PL/I ではファイル と呼ばれるデータ・セットのモデルを採用しています。 同じファイルを、プログラムの実行中の別々の時点で、別々のデータ・セットに関連づけることができます。

PL/I では、2 つのタイプのデータ伝送であるストリームとレコードが使用されます。

**ストリーム指向データ伝送**

データ・セット内のデータの編成はプログ ラムでは無視され、データは、文字フォーマットの個々のデータ値が連続するストリー ムと見なされます。 入力の場合は、データは文字フォーマットから内部フォーマットに変換され、出力の場合は、 内部フォーマットから文字フォーマットに変換されます。

ストリーム指向データ伝送は、文字フォーマットで用意された入力データを処理したり、可読フォーマットの出力を作成したりする (編集する必要がある) 場合に使用することができます。 プログラムが対話式システムのもとで実行されているときは、ストリーム指向データ伝送を使用すれば、実行時に端末からプログラムと同期通信すること ができます。

データの書式作成が可能という点で、ストリーム指向データ伝送はレコード単位データ伝送よりも多様性があります。しかし、その結果、実行時間は増加します。

**レコード単位データ伝送**

データ・セットは、離散的レコードの集合です。 外部メディア上のレコードは、普通は内部ストレージに入っているレコードをそのままコピーしたものです。 レコード単位データ伝送時にはデータ変換は行われません。 すなわち、入力の場合は、データはデータ・セットに記録されている とおりの形で伝送され、出力の場合は、データは内部で記録されている とおりの形で伝送されます。

レコード単位データ伝送は、2 進数、10 進数、または文字などのフォーマットで表記されているデータが入っているファイルを処理する場合に使用することができます。

レコード単位データ伝送は、ストリーム指向データ伝送に比べて、データの処理方法の点と、処理できるデータ・セットのタイプの点で多様性があります。 データは、それが主記憶域にあるとおりの形でデータ・セットに記録されるため、どのようなデータ・タイプでも可能です。 なにも変換は行われませんが、ユーザーはデータ構造について確実に把握している必要があります。

同じデータ・セットを、ある時点ではストリーム指向データ伝送で処理し、 別の時点ではレコード単位データ伝送で処理することもできます。 ただし、その場合は、データ・セット内の項目はすべて文字フォーマットでなければ なりません。

以下のトピックでは、データ・セットの種類、ファイルを記述するための属性、およびデータを伝送するためにファイルをオープンおよびクローズする方法について説明します。 PL/I で認識されるデータ・セット編成のタイプの詳細については、 「プログラミング・ガイド」を参照してください。

**関連情報**

295 ページの『ストリーム指向データ伝送』
この章では、ストリーム指向データ伝送に使用される入出力ステートメントを説明します。

287 ページの『レコード単位データ伝送』
この章では、レコード単位データ伝送に使用される入出力ステートメントの機能を説明します。

© Copyright IBM Corp. 1999, 2022

# データ・セット

データ・セットは、端末から入力したり端末に出力したりする場合に使用され るほかにも、磁気テープ装置、直接アクセス記憶装置 (DASD) などの各種の補助 記憶装置メディアに記憶されます。 これらの記憶メディアは、その種類に関係なく、データの収集、記憶、伝送を行うための共通の方法を使用できるという特性を持っています。 データがデータ・セットにどのように記録され、そのあと、プログラムに伝送されるためにどのように取り出されるかは、データ・セットの編成の仕方によって異なります。

データ・セットへのレコードの記憶と、データ・セットからのレコードの取り出しは、連続する物理位置または論理位置に基づいて順次に行われるか、またはデータ伝送ステートメントで指定されたキーを使用して直接行われます。

PL/I は、次のタイプのデータ・セット編成をサポートします。

連続
索引
Relative
領域内

このようなデータ・セット編成は、データの記憶方法とデータのアクセスに使用する手法において異なります。

## 連続

連続データ・セット編成では、レコードは、それらの連続する物理的な位置のみをもとに編成されています。

データ・セットが作成されると、レコードは、それらが示される順に続けて書き込まれます。 レコードは、それらが書き込まれた順でしか取り出されません。

## 索引

索引付きデータ・セット編成では、レコードは、各レコードのキーにもとづいて、論理的な順序で書き込まれます。

索引付きデータ・セットは、直接アクセス装置に置かれる必要があります。 文字ストリング・キーによって、レコードが識別され、レコードの直接検索、置換、追加、および削除を行うことができます。 また、順次処理も行うことができます。

## Relative

相対データ・セット編成では、番号付きレコードが互いに相対的な位置に書き込まれます。

レコードは、1 から始まり、続き番号で番号付けされます。 相対データ・セットは、直接アクセス装置に置かれる必要があります。 レコード番号を指定するキーによって、レコードが識別され、レコードの直接検索、置換、追加、および削除が行われます。 また、順次処理も行うことができます。

## 領域内

領域内データ・セット編成は、番号付けされた複数の領域に分割され、その各 領域は、レコードを 1 つ持つことができます。

領域は、ゼロから始まり、順番に番号付けされます。 領域は、データ伝送ステートメント内でその領域番号 (また、おそらくキー) を 指定することによってアクセスすることができます。 キーは領域番号を指定し領域を識別することによって、レコードのすべて直接 検索、置換、追加、および削除を最適化することができます。

# ファイル

データ・セット内の物理編成ではなく、 データの論理的な側面を主眼にソース・プログラムを処理することができるように、 PL/I ではファイル と呼ばれるデータ・セットのモデルを採用しています。

入出力ステートメントがどのように関連データ・セットにアクセスし、 処理するかは、これらのモデルによって異なります。 データ・セットとは異なり、ファイルは原始プログラム内でしか意味を持たず、 プログラムの外部で物理的に存在するものではありません。

ファイルを表す名前は、FILE 属性を持っています。

# FILE 属性

FILE 属性は、対応する名前がファイル定数またはファイル変数であることを指定します。

ファイル定数に、いずれかのファイル記述属性が付いていると、そのファイル定数には FILE 属性が暗黙に付けられます。 ある名前を、入出力ステートメントの FILE オプションに書くか、または任意の入出力条件に対する ON ステートメント内に書けば､その名前をファイル定数としてコンテキスト上から宣言することができます。

## ファイル定数

PL/I プログラムで処理するデータ・セットは、 それぞれファイル定数に関連づけられていなければなりません。

それぞれのファイル定数の個々の特性は、ファイル記述属性によって記述されます。 これらの属性は、代替属性と追加属性の 2 つのカテゴリーに分類されます。

代替 属性とは、属性のグループ内から選択される属性です。 あるグループ内の択一属性の 1 つが明示的にも暗黙的にも宣言されておらず、しかもそのグループ内の属性が必須である場合には、デフォルトの属性が使用されます。

275 ページの表 43 では、PL/I の代替ファイル属性をリストしています。

表 *43.* 代替ファイル属性

| グループ・タイプ | 代替属性 | デフォルトの属性 |
|---|---|---|
| 使用法 | STREAM または RECORD | STREAM |
| 関数 | INPUT または OUTPUT または UPDATE | INPUT |
| アクセス | SEQUENTIAL または DIRECT | SEQUENTIAL |
| バッファ リング | BUFFERED または UNBUFFERED | BUFFERED (SEQUENTIAL ファイルの場合) |
| | | UNBUFFERED (DIRECT ファイルの場合) |
| 有効範囲 | EXTERNAL または INTERNAL | EXTERNAL |

追加 属性とは、明示的に宣言しなければならない属性、または明 示的に宣言された別の属性によって暗黙に定義される属性です。 追加属性には、ENVIRONMENT、 KEYED、および PRINT があります。 追加属性 KEYED は、DIRECT 属性によって暗黙に定義されます。 追加属性 PRINT は、出力ファイル名 SYSPRINT によって暗黙に定義することができます。

275 ページの表 44 は、それぞれのデータ伝送タイプに適用される 属性を示したものです。

表 *44.* データ伝送のタイプによる属性

| 伝送のタイプ | 属性 |
|---|---|
| ストリーム指向 | ENVIRONMENT |
| | INPUT および OUTPUT |
| | PRINT |
| | STREAM |

表 *44.* データ伝送のタイプによる属性 *(*続き*)*

| 伝送のタイプ | 属性 |
|---|---|
| レコード単位 | BUFFERED および UNBUFFERED |
| | DIRECT および SEQUENTIAL |
| | ENVIRONMENT |
| | INPUT、OUTPUT、および UPDATE |
| | KEYED |
| | 記録 |

276 ページの表 45 には、ファイル属性の正しい組み合わせが示されています。

表 *45. PL/I* ファイル宣言の属性

| ファイル タイプ | STREAM | 記録 | | | | | 凡例: I 指定または暗黙指定が必要 D Default O オプション S 指定が必要 - 無効 |
|---|---|---|---|---|---|---|---|
| | | SEQUENTIAL | | | DIRECT | | |
| データ を設定します Organization | Consecutive | Consecutive | Relative | Indexed | Relative | Indexed | |
| ファイル属性 | | | | | | | 暗黙指定属性 |
| FILE | I | I | I | I | I | I | |
| INPUT[1] | D | D | D | D | D | D | FILE |
| OUTPUT | O | O | O | O | O | O | FILE |
| ENVIRONMENT | O | O | O | O | O | O | FILE |
| STREAM | D | - | - | - | - | - | FILE |
| PRINT[1] | O | - | - | - | - | - | FILE STREAM OUTPUT |
| RECORD | - | I | I | I | I | I | FILE |
| UPDATE[2] | - | O | O | O | O | O | FILE RECORD |

表 45. *PL/I* ファイル宣言の属性 (続き)

| ファイル タイプ / データ を設定します Organization | STREAM | 記録 | | | | | 凡例: I 指定または暗黙指定が必要 D Default O オプション S 指定が必要 - 無効 |
|---|---|---|---|---|---|---|---|
| | | SEQUENTIAL | | | DIRECT | | |
| | Consecutive | Consecutive | Relative | Indexed | Relative | Indexed | |
| SEQUENTIAL | - | D | D | D | - | - | FILE RECORD |
| KEYED[3] | - | - | O | O | I | I | FILE RECORD |
| DIRECT | - | - | - | - | S | S | FILE RECORD KEYED |

**注記 :**

[1] INPUT 属性を持つファイルは、PRINT 属性を 持つことはできません。
[2] UPDATE は、テープ・ファイルには無効です。
[3] KEYED は、*INDEXED* 出力と *RELATIVE* 出力の場合は必須です。

有効範囲については、150 ページの『宣言の有効範囲』で説明されています。

ファイル定数に、この章で説明しているいずれかのファイル記述属性が付いていると、そのファイル 定数には FILE 属性が暗黙に付けられます。 ある名前を、入出力ステートメントの FILE オプションに書くか、または任意の入出力条件に対する ON ステートメント内に書けば、その名前をファイル定数としてコンテキスト上から宣言することができます。

次の例では、名前 Sample がファイル定数として宣言されています。

```
declare Sample file;
```

## ファイル変数

ファイル変数は、FILE 属性と VARIABLE 属性を持ちますが、 どのファイル定数記述属性も持つことはできません。 ファイル変数には、ファイル定数を割り当てることができます。 割り当て後、ファイル変数を参照することは、割り当てられたファイル定数を参照することと同じになります。

レコード単位の伝送ステートメントを使用して、データ・セットのファイル変数の値を伝送することができます。 伝送後は、そのデータ・セットのファイル変数の値は有効でないことがあります。

VARIABLE 属性は、45 ページの『VARIABLE 属性』で説明されている環境において 暗黙に定義されます。

次の例に示す宣言 Account はファイル変数として宣言され、 Acct1 と Acct2 は、ファイル定数として宣言されています。 このあと、これらのファイル定数をこのファイル変数に割り当てることができます。

```
declare Account file variable,
  Acct1 file,
  Acc2 file;
```

構文については、45 ページの『VARIABLE 属性』を参照してください。

### ファイル参照の指定

ファイル参照は、ファイル定数、ファイル変数、または FILE 属性を持つ値を戻す関数参照のいずれかです。

ファイル参照は、次のように使用することができます。

- FILE または COPY オプション内で、使用されます。
- 関数またはサブルーチンに渡される引数として使用されます。
- ON、SIGNAL、および REVERT ステートメントの入出力条件を修飾するために使用されます。
- RETURN ステートメント内の式として、使用されます。

ファイル定数の値を表すファイル変数を利用して、そのファイル定数用の ON ユニットを設定できます (343 ページの『ファイル変数のための ON ユニット』を参照してください)。 次の例では、L1 と L2 のラベルが付いている 2 つのステートメントは、同じファイルに対してヌルの ON ユニットを指定しています。

```
dcl F file,
    G file variable;
    G=F;
L1:  on endfile(G);
L2:  on endfile(F);
```

## RECORD 属性と STREAM 属性

RECORD 属性と STREAM 属性は、ファイルに使用されるデータ伝送の種類を表します。

**デフォルト設定:** STREAM

**記録**
そのファイルが物理的に別個のレコードの集まりから構成されており、各レコードは任意のフォーマットの 1 つまたは複数のデータ項目から構成されていることを示すものです。 各レコードは、1 つのエンティティーとして、変数との間で伝送されます。

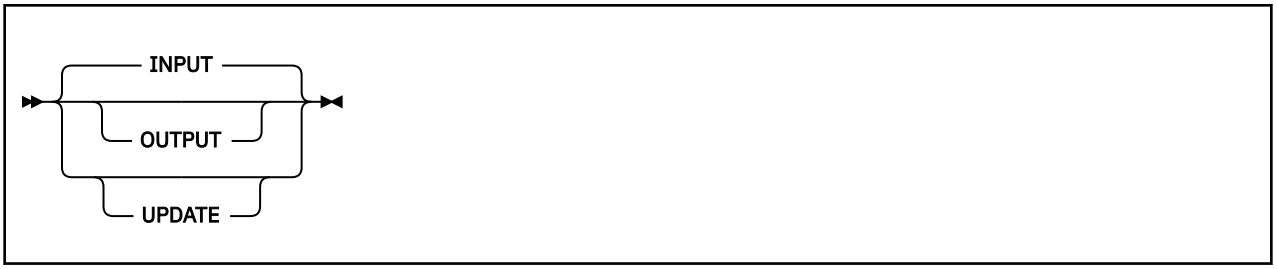
RECORD 属性を持つファイルは、OPEN、CLOSE、READ、WRITE、REWRITE、LOCATE、および DELETE 入出力ステートメントの FILE オプションでのみ指定することができます。

**STREAM**
そのファイルのデータが文字フォーマットのデータ項目の連続するストリームであり、そのストリームから各変数へ、またはそれぞれの式からそのストリームへ割り当てられることを示すものです。

STREAM 属性を持つファイルは、OPEN、CLOSE、GET、 および PUT 入出力ステートメントの FILE オプションでのみ指定することができます。

## INPUT、OUTPUT、および UPDATE 属性

INPUT、OUTPUT および UPDATE の各機能属性は、ファイルに対して行うことができるデータ伝送の方向を表します。




**デフォルト設定**: INPUT

**INPUT**
データ・セットからプログラムへデータを伝送することを表します。

**OUTPUT**
新しいデータ・セットを作成するために、または既存データ・セットを拡張するために、プログラムからデータ・セットへデータを伝送すること を表します。

**UPDATE**
データをどちらの方向にも伝送できることを指定します。 UPDATE 属性は RECORD ファイルにのみ適用されます。 SEQUENTIAL ファイルの場合、UPDATE 属性の宣言は「その場所で更新」モードを表します。

## SEQUENTIAL 属性と DIRECT 属性

SEQUENTIAL アクセス属性と DIRECT アクセス属性は、RECORD ファイルにのみ適用 され、ファイル内のレコードをどのようにアクセスするかを表します。

**省略形**: SEQUENTIAL の場合は SEQL

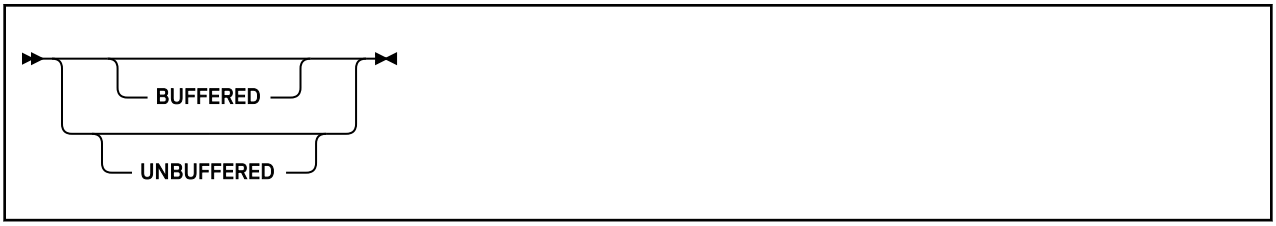
**デフォルト設定**: SEQUENTIAL

**DIRECT**
データ・セットのレコードを直接アクセスできることを表します。 データ・セット内のレコードの位置は、文字ストリングであるキーによって突き止められます。 したがって、DIRECT 属性を指定すると、KEYED 属性が暗黙に定義されます。 関連データ・セットは、直接アクセス記憶装置上になければなりません。

**SEQUENTIAL**
連続データ・セットまたは相対データ・セットのレコード を物理的な順序でアクセスすること、および索引付きデータ・セットのレコー ドをキー・シーケンス順でアクセスすることを表します。 ある一定のデータ・セット編成では、SEQUENTIAL 属性を持つファイルを直接ア クセスのために使用したり、ランダム・アクセスと順次アクセスの混合のため に使用したりすることができます。 この場合、そのファイルは追加属性 KEYED を持っていなければなりません。 SEQUENTIAL UPDATE ファイルの場合は、データ・セットの既存のレコードを変更することも、無視することも、(データ・セットが INDEXED のときに) 削除することもできます。

## BUFFERED 属性および UNBUFFERED 属性

バッファリング属性は、RECORD ファイルにのみ適用され、伝送中に、各レコードが中間ストレージ・バッファーを通過しなければならないか、各レコードを直接送信できるかを指定します。




**省略形**: BUFFERED の場合は BUF、 また UNBUFFERED の場合は UNBUF

**デフォルト設定**: BUFFERED は SEQUENTIAL ファイルの デフォルトです。 UNBUFFERED は、DIRECT ファイルのデフォルトです。

**BUFFERED**
RECORD ファイルの各レコードが、データ・セットへ伝送され るとき、またはデータ・セットから伝送されるときに、中間記憶バッファーを 通らなければならないことを表します。 これによって、移動モード処理と位置指定モード処理の両方を行うことができます。

**UNBUFFERED**
データ・セットのレコードがバッファーを通る必要がなく、 変数に関連づけられている主記憶域へ、またはその主記憶域から、直接に伝送できることを表します。 このため、移動モード処理しか行うことができません。

## ENVIRONMENT 属性

ENVIRONMENT 属性の特性リストは、PL/I 言語の一部でない各種のデータ・セット特性を指定します。

特性と使用法の全リストおよび説明については、「プログラミング・ガイド」を参照してください。

**注 :** 特性は PL/I 言語の一部ではないため、それらの特性をファイル宣言で使用すると、 アプリケーション・プログラムの移植性が制限されることがあります。

## KEYED 属性

KEYED 属性は、RECORD ファイルにのみ適用され、索引付きデータ・セットおよび 相対データ・セットに関連づけられなければなりません。 この属性は、ファイル内のレコードを、レコード入出力ステートメントのキー ・オプションのいずれか 1 つ (KEY、KEYTO、または KEYFROM) によってアクセスで きることを表します。

どのキー・オプションも使用しない場合は、KEYED 属性を指定する必要はありません。

## PRINT 属性

PRINT 属性は、STREAM 属性と OUTPUT 属性を持つファイルの場合に指定できます。 この属性はそのファイルが印刷される予定であることを示します。 すなわち、ファイルと関連したデータを、 最初はほかのメディアに書き出されていても、 印刷されたページに現れることを示します。

詳しくは、『312 ページの『PRINT 属性』』を参照してください。

# ファイルのオープンとクローズ

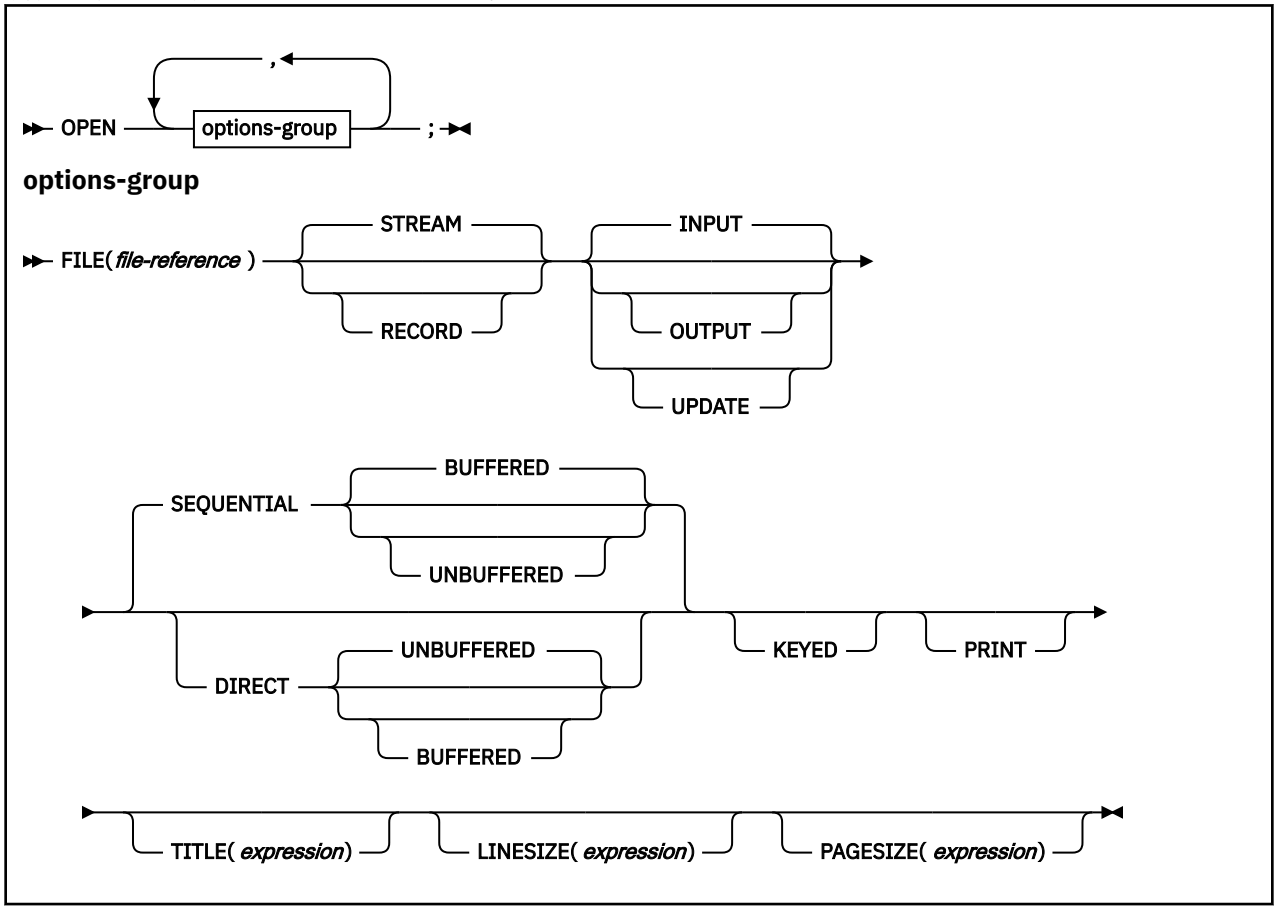
ファイルがデータ・セットに関連づけられていないうちは、そのファイルを入 出力ステートメントによるデータ伝送に使用することはできません。 ファイルをオープンするということは、ファイルをデータ・セットに関連づけ ることであり、外部記憶メディアが使用可能かどうかを調べ、そのメディアの位置を決め、必要なオペレーティング・システム・サポートを割り振るということです。 処理の完了時には、ファイルはクローズされなければなりません。 ファイルをクローズするということは、ファイルとデータ・セットの関係を 解くことです。

PL/I には、これらの機能を行うために、 OPEN と CLOSE という 2 つのステートメントがあります。 ただし、これらのステートメントを使用するかどうかはオプションです。 OPEN ステートメントが実行されなかったファイルは、そのファイルを参照する 最初のデータ伝送ステートメントが実行されている間に暗黙にオープンされます。 この場合、DECLARE ステートメントで指定されたファイル情報 (および伝送ステートメントから派生するデフォルトの値) を使用してファイルがオープンされます。 同様に、ファイルが PL/I の終了前にクローズされない場合、 PL/I は終了処理中にそのファイルをクローズします。

ストリーム入力、順次入力、または順次更新のファイルがオープンされたときは、関連データ・セットは最初のレコードに位置付けられます。

## OPEN ステートメント

OPEN ステートメントは、ファイルをデータ・セットに関連づけます。 これによって、OPEN ステートメントで指定された属性と DECLARE ステートメント で指定された属性が組み合わされます。 また、オープンされるファイルに必要な属性の一部がまだ宣言されていない場 合は、このステートメントでファイルの属性の指定を完成させることができます。




OPEN ステートメントのオプションの指定順序は任意です。

**FILE**
データ・セットに関連付けられるファイルの名前を指定します。

**STREAM、RECORD、**
**INPUT、OUTPUT、UPDATE、**
**DIRECT、SEQUENTIAL、**
**BUFFERED、UNBUFFERED、**
**KEYED、および PRINT**
これらのオプションは、ファイル宣言で指定した属性を補足する属性を表します。 同一ファイルを参照する OPEN ステートメントと DECLARE ステートメントの両方で、同じ属性を指定する必要はありません。 レコード入出力とストリーム入出力の属性のリストは、275 ページの表 44 を 参照してください。

STREAM ファイルがオープンされると、最初の GET または PUT ステートメントは、ステートメント・オプションまたはフォーマット項目により、アクセスする最初のレコ ードを指定できます。 ステートメント・オプションまたはフォーマット項目は、*n* 行をスキップした あとにレコードにアクセスすることを指定します。 ついで、ファイルは *n* 番目のレコードの先頭に位置づけられます。 ステートメント・オプションまたはフォーマット項目がない場合、初期ファイルは、最 初の行もしくは最初のレコードの先頭に位置づけられます。 そのファイルが PRINT 属性を持っているときは、最初の行または最初のレコードの 1 桁目に物理的に位置づけられます。

すでにオープンされているファイルをオープンしても、そのファイルに影響はありません。

**TITLE**
*expression* (式) の内容によって、なにが指定されているかがわかります。 TITLE 属性の詳細については、「プログラミング・ガイド」を参照してください。

**LINESIZE**
このファイルに対する後続の処理での行の長さ (バイト数) を表す値で、整数値に変換されます。 印刷フォーマット項目や制御フォーマット項目を使用することによって、あるいは GET または PUT ステートメントの オプションによって、新しい行を開始することができます。 新しい行を開始するための明示的な処置が取られる前に、行の終わりを超えた位置にファイルを位置付けようとすると、新しい行が開始され、この新しい行の始まりにファイルが位置付けられます。 PRINT ファイルのデフォルトの行サイズは 120 です。

LINESIZE オプションは、STREAM OUTPUT ファイルの場合にのみ指定する ことができます。 式の値は、2G より小さくなければなりません。

**PAGESIZE**
計算後、整数値に変換される、1 ページあたりの行数を表す値です。 この限界値を超えるような操作が試みられた最初の時点で ENDPAGE 条件が起こります。 そのあと、PRINT ファイルへの伝送時に、PAGE フォーマット項目を使用することによって、あるいは PUT ステートメントの PAGE オプションによって、新しいページを開始することができます。 デフォルトのページ・サイズは 60 です。

PAGESIZE オプションは、PRINT 属性を持っているファイルにのみ指定することができます。 式の値は、2G より小さくなければなりません。

# 暗黙オープン

OPEN ステートメントがまだ実行されていないファイルを参照する GET、PUT、READ、WRITE、LOCATE、REWRITE、 または DELETE ステートメントが実行されると、そのファイルは暗黙にオープンされます。

COPY オプション付きの GET ステートメントが実行されると、COPY オプションで指定したファイル、または出力ファイル SYSPRINT (ファイルを指定しない場合) が暗黙にオープンされることがあります。 COPY オプションで指定したファイルが暗黙にオープンされると、STREAM 属性と OUTPUT 属性が暗黙に定義されます。

282 ページの表 46 に、左欄のステートメントによってファイルが暗黙にオープンされたとき、暗黙に定義される属性を示します。

表 *46.* 暗黙オープンによって暗黙に定義される属性

| ステートメント | 暗黙に定義される属性 |
|---|---|
| 取得 | STREAM、INPUT |
| PUT | STREAM、OUTPUT |
| READ | RECORD、INPUT [注] |
| WRITE | RECORD、OUTPUT [注] |
| LOCATE | RECORD、OUTPUT、SEQUENTIAL |
| REWRITE | RECORD、UPDATE |
| 削除 | RECORD、UPDATE |

表 46. 暗黙オープンによって暗黙に定義される属性 (続き)

| ステートメント | 暗黙に定義される属性 |
|---|---|
| **注 :** UPDATE が明示的に宣言されていない場合に限り、INPUT および OUTPUT が READ および WRITE ステートメントのデフォルト属性になります。 | |

282 ページの表 46 のいずれかのステートメントによって ファイルを暗黙にオープンするということは、上記と同じ属性が指定されている ファイルに明示的な OPEN ステートメントを使用することと同じ効果があります。

ファイル宣言内に指定した属性と、ファイルをオープンした結果として暗黙に定義される属性との間に、矛盾があってはなりません。 例えば、INPUT 属性と UPDATE 属性は矛盾しており、UPDATE 属性と STREAM 属性も矛盾しています。

275 ページの表 43 にリストされたデフォルト属性が適用される前に、暗黙に定義される属性が適用されます。 暗黙に定義される属性が矛盾を引き起こすこともあります。 デフォルトの属性が適用されたあとで属性が矛盾していると、UNDEFINEDFILE 条件が起こります。

表 47. マージ後の属性と暗黙に定義される属性

| マージ後の属性 | 暗黙に定義される属性 |
|---|---|
| UPDATE | RECORD |
| SEQUENTIAL | RECORD |
| DIRECT | RECORD、KEYED |
| PRINT | OUTPUT、STREAM |
| KEYED | RECORD |

## ファイル定数の例

この例では、ファイル定数で指定されたファイルを明示的にオープンするために組み合わせる属性を示します。

```
declare Listing file stream;
open file(Listing) print;
```

OPEN ステートメントが実行されると、属性が組み合わされて、STREAM と PRINT という属性が付けられます。 暗黙の属性が適用されて、属性は STREAM、PRINT、および OUTPUT となります。 デフォルトが適用されたあとの属性は、STREAM、PRINT、OUTPUT、および EXTERNAL となります。

## ファイル変数の例

この例では、ファイル変数を使用してファイルを明示的にオープンするために組み合わせる属性を示します。

```
declare Account file variable,
        (Acct1,Acct2)  file
        output;

Account = Acct1;
open file(Account) print;

Account = Acct2;
open file(Account) record unbuf;
```

ファイル Acct1 がオープンされて、 その属性 (明示的に宣言された属性と暗黙に定義された属性) は STREAM、 EXTERNAL、PRINT、および OUTPUT になります。 ファイル Acct2 が、RECORD、EXTERNAL、および OUTPUT の各属性を付けてオープンされます。

### 暗黙オープンの例

この例では、ファイルを明示的にオープンする (READ ステートメントの実行によって行われます) ために組み合わせる属性を示します。

```
declare Sample file keyed internal;

read file (Sample)
     into (Sample_Record)
     keyto(Sample_Key);
```

マージ (READ ステートメントの実行による暗黙オープンのため) 後の属性は、 KEYED、INTERNAL、RECORD、INPUT です。 (そのほかに暗黙に定義される属性はありません。) デフォルトが適用されたあとの属性は、KEYED、INTERNAL、RECORD、INPUT、SEQUENTIAL です。

### ファイル定数の宣言の例

```
declare File3 input direct environment( regional(1) )
```

この宣言は 3 つのファイル属性、INPUT、DIRECT、および ENVIRONMENT を指定します。 その他の暗黙に定義される属性は、FILE (3 つのうちどの属性によっても暗黙に 定義される)、および RECORD と KEYED (DIRECT によって暗黙に定義される) です。 有効範囲は、デフォルトによって EXTERNAL となります。 ENVIRONMENT 属性は、データ・セットが REGIONAL(1) 編成であることを表します。

先の宣言では、必要なすべての属性が DECLARE ステートメントで指定されているか、または暗黙に定義されています。 これらの属性を、OPEN ステートメントで変更 (または無効に) することはできません。

次の例のように宣言しておけば、`invntry` を別々の目的でオープンすることができます。

```
declare invntry file;
```

次の例では、ファイル属性は、先の例の DECLARE ステートメントで 指定された (あるいは暗黙に定義される) 属性と同じになります。

```
open file (Invntry)
     update sequential;
```

このようにしてオープンされたファイルがクローズされたあとであれば、別の一組の属性を指定して同じファイルをオープンすることができます。 例えば、次の OPEN ステートメントを使用すれば、KEYTO または KEY オプションを使用してレコードを読み取ることができます。

```
open file (Invntry)
     input sequential keyed;
```

ファイルは SEQUENTIAL であるので、このデータ・セットを順次にアクセスすることも、 さらに、READ ステートメントで KEY オプションを指定すれば直接にアクセスすることもできます。 この種のファイルの場合、READ ステートメントで KEY オプションを指定すれば、指定したレコードが読み取られます。 後続の READ ステートメントで KEY オプションを省略すると、KEY 順の次のレコードから順次に読み取られます。

## CLOSE ステートメント

CLOSE ステートメントは、そのデータ・セットからオープンされたファイルを分離します。

**FILE**
データ・セットから分離されるファイルの名前を指定します。 CLOSE FILE(*) と指定すると、オープンしているファイルがすべて クローズされます。

CLOSE ステートメントは、明示的にまたは暗黙にオープンされたときにセットされたすべてのファイル属性を、ファイルから切り離すこともします。 必要ならば、後続の OPEN ステートメントで、この同じファイルに対して新しい属性を指定することができます。 ただし、DECLARE ステートメントでこのファイル定数に明示的に与えた属性は、そのまま効力を持ち続けます。

すでにクローズされたファイルをクローズしても、なんの影響もありません。 クローズされたファイルは再びオープンすることができます。 CLOSE ステートメントを使用してファイルをクローズしなくても、ファイルは、プログラムの終了時にクローズされます。

## FLUSH ステートメント

FLUSH ステートメントは、1 つまたはすべてのファイルを フラッシュするのに使用します。

**FILE**
出力ファイルの名前を指定します。

FLUSH ステートメントは、1 つのオープン・ストリーム出力ファイル (または * が指定されている場合はすべてのオープン・ストリーム出力ファイル) に関連したバッファー内容の即時書き込みを強制します。 このような処理は通常、ファイルのクローズ時またはプログラムの終了時に行われますが、FLUSH ステートメントを指定すると、他の処理が発生する前に確実にバッファーを書き込むことができます。

FLUSH ステートメントは、レコード出力ファイルには影響しません。

# SYSPRINT および SYSIN

すべての PL/I プログラムで、入力ファイル SYSIN および出力ファイル SYSPRINT を使用できます。 これらのファイルは、明示的に宣言したりオープンしたりする必要はありません。

SYSIN のデフォルト属性は STREAM INPUT で、SYSPRINT のデフォルト属性 は STREAM OUTPUT PRINT です。 SYSPRINT は 7 文字を超えていますが、SYSIN と SYSPRINT のどちらのファイル名もデフォルト属性 EXTERNAL を 持っています。

コンパイラーは、上に説明した特殊用途のために SYSIN および SYSPRINT と いう名前を予約していません。 これらの名前は、SYSIN ファイルと SYSPRINT ファイルを識別すること以外の用途にも使用できます。 SYSIN と SYSPRINT には他の属性を適用できますが、SYSPRINT が STREAM OUTPUT ファイルとして宣言またはオープン される場合は、INTERNAL 属性が宣言されていなければデフォルト で PRINT 属性が適用されます。

# 第 11 章 レコード単位データ伝送

この章では、レコード単位データ伝送に使用される入出力ステートメントの機能を説明します。

レコード単位データ伝送とストリーム指向データ伝送の両方に同じように 適用される PL/I の機能 (ファイルの宣言、ファイル属性、およびファイルの オープンとクローズなど) については、273 ページの『第 10 章 入出力』で説明しています。 ENVIRONMENT 属性の構文については、280 ページの『ENVIRONMENT 属性』を参照してください。 それぞれのデータ・セット編成ごとの環境特性とレコード入出力データ伝送ステートメントの詳細については、「プログラミング・ガイド」を参照してください。

レコード単位データ伝送では、データ・セット内のデータは、オペレーティング・システムで 受け入れ可能な任意のフォーマットで記録されているレコードの集まりです。 レコード単位データ伝送時には、データ変換は行われません。 つまり、入力の場合は、READ ステートメントが実行されると、1 つのレコードがデータ・セット内に記録されているとおりの形でプログラム変数に伝送される か、あるいはレコードを指すポインターがセットされます。 出力の場合は、ステートメント WRITE、REWRITE、または LOCATE が実行されると、1 つのレコードが 内部的に記録されているとおりの形でプログラム変数から伝送されます。 ファイルに伝送された情報の長さが N (設定されたレコード長 M より小さい) であ る場合、最新の M から N を引いたバイト数の結果の値は未定義です。

## データ伝送

パラメーターや DEFINED 変数も含むほとんどの変数を、レコード単位データ伝送ステートメントで 伝送することができます。 一般に、この章で説明することがらは、すべての変数にあてはまります。

**注 :** データ集合は、連結ストレージ内になければなりません。入力または出力に漢字ストリングを指定する場合は、そのファイルに SCALARVARYING オプションを指定しなければなりません。データについてのその他の考慮事項は、以降の節で説明しています。

### 位置合わせされていないビット・ストリング

位置合わせされていないビット・ストリングは伝送できない場合があります。

次のものを伝送することはできません。

- 位置合わせされていない固定長ビット・ストリングである BASED 変数、DEFINED 変 数、パラメーター変数、添え字付き変数、または構造体基本エレメント変数。
- 位置合わせされていない固定長ビット・ストリングを最初または最後の基本エレメントとして持つ小構造体 (ただし、その基本エレメントが、その小構造体を含んでいる大構造体 の最初または最後のエレメントである場合を除きます)。
- DEFINED 属性を持つ大構造体か、またはパラメーターである大構造体のうち、位置合わせされていない固定長ビット・ストリングを最初または最後のエレメントとして持つもの。

### 可変長ストリング

読み取りと書き込みに可変長ストリングを使用すると、未定義または未知の長さを持つことができるレコードにアクセス可能になります。

可変長ストリングを指定した位置指定モードの出力ステートメント (289 ページの『LOCATE ステートメント』を参照してください) は、 そのストリングの最大長に等しい長さを持つフィールドを伝送します。 VARYINGZ ストリングでは、ヌル終了文字も伝送されます。 VARYING ストリングでは、ストリングの現行の長さを示す 2 バイトの接頭部も 伝送されます。このため、ENVIRONMENT 属性の SCALARVARYING オプションを ファイルに指定する必要があります。

可変長ストリング変数を指定してある移動モードの出力ステートメント (288 ページの『WRITE ステートメント』および 289 ページの『REWRITE ステートメント』を参照してください) は、 ストリングの現行の長さだけ伝送します。 VARYINGZ ストリングでは、ヌル終了文字も伝送されます。 VARYING ストリングでは、ENVIRONMENT 属性の SCALARVARYING オプションがファイ ルに指定されている場合のみ、2 バイトの接頭部が一緒に伝送されます。

© Copyright IBM Corp. 1999, 2022

## 区域変数

位置指定モードの出力ステートメントで区域変数を指定すると、宣言された区 域サイズに 16 バイト (制御情報が入っている接頭部の長さ) を加えた長さのフィ ールドが伝送されます。

移動モードの出力ステートメントでエレメント区域変数を指定するか、または最後のエレメントが 区域変数である構造体を指定すると、区域の現在のエクステントと 16 バイトの接頭部だけが伝送されます。

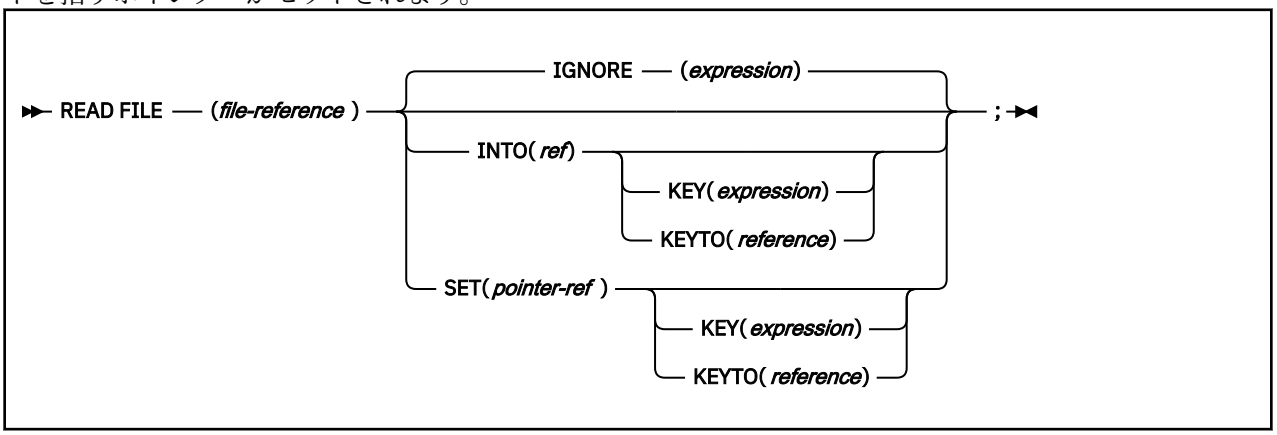
# データ伝送ステートメント

データ・セットとの間でレコードを伝送するためのデータ伝送ステートメント は、READ、WRITE、LOCATE、および REWRITE です。

DELETE ステートメントは、UPDATE ファイルからレコードを削除します。 どのデータ伝送ステートメントを使用できるかは、ファイルの属性によって決まります。 ステートメント・オプションについては、290 ページの『データ伝送ステートメントのオプション』を参照してください。 データ伝送ステートメントの変数については、「プログラミング・ガイド」を参照してください。

## READ ステートメント

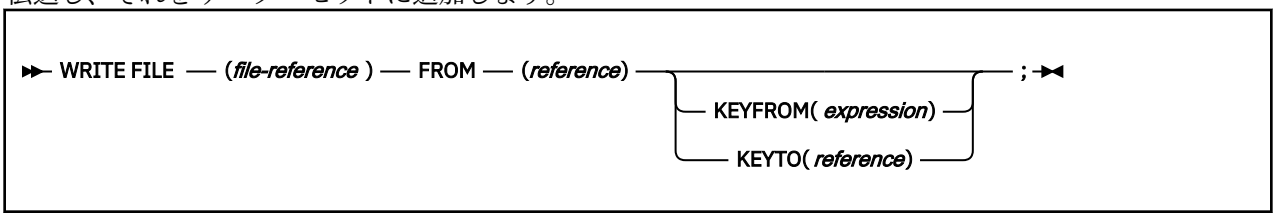
READ ステートメントは、任意の INPUT または UPDATE ファイルに対して 使用することができます。 これによって、レコードがデータ・セットからプログラム変数へ伝送されるか、 またはストレージ内のレコードを指すポインターがセットされます。




キーワードの指定順序は任意です。 INTO、SET、IGNORE のどのオプションも付いていなければ、READ ステートメントは、READ IGNORE(1) と同等です。

## WRITE ステートメント

WRITE ステートメントは、SEQUENTIAL UPDATE ファイル (VSAM の場合)、DIRECT UPDATE ファイル、および任意の OUTPUT ファイルに対して使用できます。 このステートメントは、プログラムからレコードを伝送し、それを データ・セットに追加します。




キーワードの指定順序は任意です。

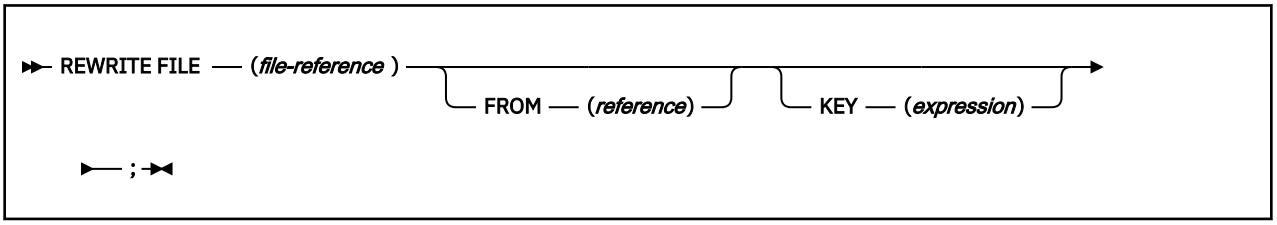
WRITE ステートメントは、SEQUENTIAL UPDATE ファイルとしてアクセスされる連続データ・セットを更新するためには使用できません。 SEQUENTIAL UPDATE ファイルによって連続データ・セットを更新するには、REWRITE ステートメントによる更新の前に、READ ステートメントでレコードを取り出す必要があります。

また、既存の順次ファイルの終わりにレコードを追加したい場合は、ファイルを OUTPUT として開く必要があり、その DD ステートメントで DISP=MOD を指定する (プログラムが z/OS バッチ下で実行されている場合) か、環境変数で APPEND(Y) を指定する (プログラムが Windows、AIX、または z/OS UNIX 下で実行されている場合) 必要があります。

## REWRITE ステートメント

REWRITE ステートメントは、UPDATE ファイルのレコードを置き換えます。

SEQUENTIAL UPDATE ファイルの場合は、REWRITE ステートメント は、ファイルから読み込まれた最後のレコードが再書き込みされることを 指定します。 必然的に、再書き込みまでにレコードが読み込まれている 必要があります。 DIRECT UPDATE ファイルの場合は、どのレコードも、まず読み取られたかどうかに関係なく、再書き込みができます。




キーワードの指定順序は任意です。 DIRECT 属性を持つ UPDATE ファイルや、SEQUENTIAL 属性と UNBUFFERED 属性の両方 を持つ UPDATE ファイルの場合、FROM オプションを指定しなければなりません。

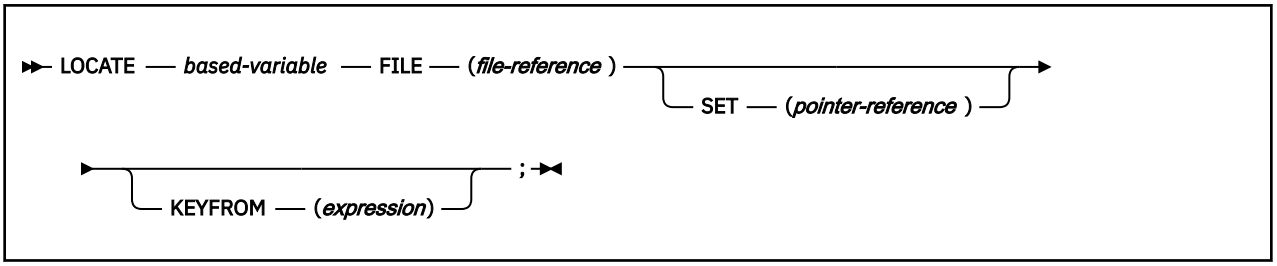
FROM オプションの指定のない REWRITE ステートメントを実行すると、次のような結果になります。

- 最後に読み取られたレコードが、INTO オプションが指定された READ ステートメントで 読み取られたものである場合は、FROM オプションが指定されていない REWRITE ステートメントを 実行しても、データ・セット内のそのレコードは変わりません。
- 最新のレコードが、SET オプションが指定された READ ステートメントで読み取ら れた場合、そのレコードは、SET オプション内のポインター変数によって識別さ れる変数でどのような割り当てが行われても更新されます。

## LOCATE ステートメント

LOCATE ステートメントは、OUTPUT SEQUENTIAL BUFFERED ファイルを位置指定モードで処理する場合にのみ使用することができます。 このステートメントは、出力バッファー内のストレージを基底付き変数に割り振り、 次のレコードの位置を指すポインターをセットします。

位置指定モードの処理の詳細については、293 ページの『位置指定モード』を参照してください。




キーワードの指定順序は任意です。

**based-variable**
添え字なしのレベル 1 基底付き変数でなければなりません。

## DELETE ステートメント

DELETE ステートメントは、UPDATE ファイルからレコードを削除します。

キーワードの指定順序は任意です。 KEY オプションを省略すると、最後に読み取られたレコードが削除されます。 別の READ ステートメントが処理されるまで、KEY が指定されていない 後続の DELETE または REWRITE ステートメントを実行することはできません。 KEY オプションが含まれる場合、キーによってアドレス指定されたレコードが 検出されると、これを削除します。

# データ伝送ステートメントのオプション

レコード単位データ伝送ステートメントでどのオプションを使用できるかは、ファイルの属性や、関連データ・セットの特性によって異なります。

## FILE オプション

どのレコード単位データ伝送ステートメントでも、FILE オプションを指定しなければなりません。 このオプションは、行われる操作の対象となるファイルを表します。

FILE オプションの例は、本節の各ステートメントごとに記載しています。 指定したファイルが現行処理でまだオープンされていないときは、暗黙にオープンされます。

## FROM オプション

FROM オプションは、レコードをどの要素変数または集合変数から書き出すかを指定します。 このオプションは、任意の OUTPUT または DIRECT UPDATE ファイルを 参照する WRITE ステートメント内で使用しなければなりません。 また、任意の UPDATE ファイルを参照する REWRITE ステートメント内で 使用することもできます。

変数が集合の場合、その変数は連結ストレージ内になければなりません。 位置合わせされていない固定長ビット・ストリングは、場合によっては許可されない ことがあります (詳細は、287 ページの『データ伝送』を参照してください)。

FROM で参照する変数は、可変長のエレメント・ストリング変数であってもかまいません。 WRITE ステートメントを FROM オプション付きで指定すると、可変長ストリングの現行長のみがデータ・セットに伝送されます。 VARYINGZ ストリングでは、ヌル終了文字が付加され、 これも伝送されます。 VARYING ストリングでは、 ENVIRONMENT 属性の SCALARVARYING オプションが ファイルに指定されている場合のみ、長さを指定する 2 バイトの接頭部が付加されます。

レコードは、整数バイトの長さとして伝送されます。 したがって、バイト境界に合わせられていないビット・ストリング (または、最 初または最後のエレメントがビット・ストリングである構造体) が伝送される場合は、レ コードの最初または最後に、ストリングの一部ではないビットが含まれることになります。

SEQUENTIAL UPDATE ファイルを参照する REWRITE ステートメントでは、FROM オプションを 省略することができます。 最後に読み取られたレコードが、INTO オプションが指定された READ ステートメントで読み取られたものである場合は、FROM オプションが指定されていない REWRITE ステートメントを 実行しても、データ・セット内のそのレコードは変わりません。 しかし、最後に読み取られたレコードが、SET オプションが指定された READ ステ ートメントで読み取られたものである場合は、そのレコード (どのような割り当て によっても更新される) がデータ・セットに書き戻されます。

以下の例では、ステートメントは、変数 Sample_Rec の値が出力ファイル Sample に書き込まれることを指定しています。

```
write file (Sample) from (Sample_Rec);
```

REWRITE ステートメントは、Sample_Rec が UPDATE ファイルから読み取られた最後のレコードを置き換えることを指定します。

```
rewrite file (Sample) from (Sample_Rec);
```

## IGNORE オプション

IGNORE オプションは、任意の SEQUENTIAL INPUT または SEQUENTIAL UPDATE ファイルを参照する READ ステートメント内で使用することができます。

IGNORE オプションの式が評価され、整数値 *n* に変換されます。 *n* がゼロより大きい場合、*n* レコードは無視されます。 つまり、同じファイルを参照する後続の READ ステートメントは、 (*n*+1) 番目のレコードをアクセスすることになります。 *n* が 1 未満であると、READ ステートメントにはなにも影響しません。

次の例では、ファイル内の次からの 3 つのレコードを無視することを指定しています。

```
read file (In) ignore (3);
```

## INTO オプション

INTO オプションは、論理レコードをどの要素変数または集合変数の中に読み込むかを指定します。

このオプションは、任意の INPUT または UPDATE ファイルを参照する READ ステートメント内で使用することができます。

変数が集合の場合、その変数は連結ストレージ内になければなりません。 位置合わせされていない固定長ビット・ストリングは、場合によっては許可されない ことがあります (詳細は、287 ページの『データ伝送』を参照してください)。

変数が GRAPHIC、UCHAR、または WIDECHAR の場合は、NONVARYING でなければなりません。

INTO で参照する変数は、可変長のエレメント・ストリング変数であってもかまいません。 VARYINGZ ストリングでは、各レコードにヌル終了文字が含まれます。 VARYING ストリングでは、ファイルに ENVIRONMENT 属性の SCALARVARYING オプションが 指定されていると、各レコードにはストリング・データの長さを指定する 2 バイトの接 頭部が含まれます。

入力時に SCALARVARYING が宣言されていなければ、ストリングの長さはレコードの長 さから計算され、2 バイトの接頭部として付加されます (VARYING ストリングの場合)。 VARYING ビット・ストリングの長さは、この計算で 8 の倍数に切り上げられるの で、計算後の長さは宣言された最大の長さより大きくなることがあります。

次の例は、次の順序のレコードを 変数 RECORD_1 の中に読み込むことを指定しています。

```
read file (Detail) into (Record_1);
```

## KEY オプション

KEY オプションは、レコードを識別するための文字キー、グラフィック・キーまたはワイド文字キーを指定します。 このオプションは、INPUT または UPDATE ファイルを参照する READ ステートメント内や、DIRECT UPDATE ファイルを参照する REWRITE ステートメント内で使用することができます。

KEY オプションは、KEYED ファイルにのみ適用されます。 KEY オプションは、ファイルに DIRECT 属性がある場合は必須、ファイルに SEQUENTIAL 属性および KEYED 属性がある場合はオプションです。

KEY オプション内の式が評価され、それが文字、グラフィック、ワイド文字のいずれでもない場合、式はキーを表す文字値に変換されます。 式の値は、2G より小さくなければなりません。 どのレコードが読み取られるかは、この文字値、グラフィック値、またはワイド文字値によって決まります。

次の例では、変数 Stkey に入っている文字値で識別されるレコードを、変数 Item の中に読み込むように指定しています。

```
read file (Stpck) into (Item) key (Stkey);
```

## KEYFROM オプション

KEYFROM オプションは、レコードの伝送先となるデータ・セット上のレコードを 識別するための文字キー、グラフィック・キー、またはワイド文字キーを指定します。 このオプションは、ファイル KEYED

OUTPUT または DIRECT UPDATE を 参照する WRITE ステートメント内、もしくは LOCATE ステートメント内で使用することができます。

KEYFROM オプションは、KEYED ファイルにのみ適用します。 KEY オプション内の式が評価され、それが文字、グラフィック、ワイド文字のいずれでもない場合、式は文字ストリングに変換され、書き込み時のレコードのキーとして使用されます。 式の値は、2G より小さくなければなりません。

相対データ・セットは、KEYFROM オプションを使用して作成することができます。 この場合、レコード番号をキーとして指定します。

KEYFROM オプションを使用して、REGIONAL(1) データ・セットを作成する ことができます。 この場合、領域番号をキーとして指定します。

索引付きデータ・セットの場合、KEYFROM は、記録されるキーを指定しますが、 その長さはデータ・セットに指定されたキーの長さと等しくなければなりません。

次の例では、Loanrec の値をファイル Loans のレコー ドとして書き出すことと、Loanno の文字ストリング値をキーとして使用す る (このキーを使用すれば、このレコードを取り出すことができる) ことを指 定しています。

```
write file (Loans) from (Loanrec) keyfrom (Loanno);
```

## KEYTO オプション

KEYTO オプションは、レコードのキーが割り当てられる文字変数、グラフィック変数、またはワイド文字変数を指定します。

KEYTO オプションでは、STRING 以外の任意のストリング疑似変数を指定することができます。 数字ピクチャー指定を付けて宣言した変数を指定することはできません。 KEYTO オプションは、SEQUENTIAL INPUT ファイル、または SEQUENTIAL UPDATE ファイルを 参照する READ ステートメントで指定することができます。

KEYTO オプションは、KEYED ファイルにのみ適用されます。 式の値は、2G より小さくなければなりません。

KEYTO 変数への割り当ては、常に、INTO 変数への割り当て後に行われます。 キー指定が正しくないことが検出されると、KEY 条件が起こります。 値は次のように割り当てられます。

- 索引付きデータ・セットの場合、記録されたキーは、文字変数の宣言された長 さに合わせて、右側が埋め込まれるかまたは切り捨てられます。
- 相対 データ・セットの場合、レコード番号は文字ストリングに変換されます。 このとき、先行ゼロは、文字変数の宣言された長さに合わせて、消去される か、切り捨てられるか、または左側に埋め込まれます。
- REGIONAL(1) データ・セットの場合、8 文字の領域番号は、文字変数の宣言された長さに合わせて、左側に埋め込まれるかまたは切り捨てられます。 文字変数が可変長のときは、領域番号の先行ゼロは切り捨てられ、ストリングの長さは有効数字の桁数にセットされます。 領域番号がすべてゼロであるときは、ゼロが 1 つだけになるように切り捨てられます。

このような埋め込みまたは切り捨てに関しては、KEY 条件は起こりません。

次の例では、 ファイル Detail 内の次のレコードを 変数 Invntry の中に読み込むことと、 そのレコードのキーを変数 Keyfld に 割り当てることを指定しています。

```
read file (Detail) into (Invntry) keyto (Keyfld);
```

## SET オプション

SET オプションは、READ ステートメントまたは LOCATE ステートメントで 使用することができます。 READ ステートメントの場合、読み取られるレコードを指すためにセットされるポインター変数を指定します。 LOCATE ステートメントの場合、次に出力するレコードを指すためにセットされるポインター変数を指定します。

LOCATE ステートメントで SET オプションを省略すると、レコード変数と一緒に宣言されているポインターがセットされます。 VARYING ストリングを伝送するときは、そのファイルに対して SCALARVARYING オプションを指定しておかなければなりません。

次の例では、次の順次レコードのバッファー内の場所を指す ように、ポインター変数 P の値をセットすることを指定しています。

```
read file (X) set (P);
```

# 処理モード

レコード単位データ伝送には、2 通りのデータ処理方法があります。

**移動モード**
データを変数に移動させたり、変数から移動させたりして、データを処理します。

**位置指定モード**
データをバッファーに入れたまま処理することができます。 データ伝送ステートメントが実行されると、バッファー内のそのレコードに割 り振られているストレージの位置がポインター変数に割り当てられます。 位置指定モードは BUFFERED ファイルにのみ適用されます。

どちらの処理モードが使用されるかは、指定するデータ伝送ステートメントとオプションによって決まります。

## 移動モード

移動モードでは、READ ステートメントは、レコードをデータ・セットから INTO オプションで 指定された変数に伝送します。

WRITE ステートメントまたは REWRITE ステートメントは、レコードを FROM オプションで指定された変数から データ・セットに伝送します。 INTO オプションや FROM オプションで指定する変数は、どのストレージ・クラスであってもかまいません。

次に、移動モードの入力の例を示します。

```
Eof_In = '0'b;
on endfile(In) Eof_In = '1'B;
read file(In) into(Data);
do while (¬Eof_In);
   .
   .
   .
      /* process record  */
   read file(In) into(Data);
end;
```

## 位置指定モード

位置指定モードでは、バッファーの位置がポインター変数に割り当てられます。

基底付き変数は、レコードについて記述しています。 別々の基底付き変数を使用すれば、同じデータを別々に解釈させることが できます。 また、位置指定モードを使用すれば、自己定義レコードを読み取ることもできます。 自己定義レコードでは、そのレコードの一部に入っている情報が、そのレコードの残りの部分の 構造体を表します。 例えば、そのような情報としては、配列の境界や、そのデータの属性に合わせてどの基底付き 構造体を使用する必要があるかを表すコードなどがあります。

SET オプションが指定された READ ステートメントは、レコードが入っているバッ ファーを示すために、SET オプションで指定されたポインター変数をセットします。 したがって、このポインター変数で修飾された基底付き変数を使用すれば、 レコードのデータを参照することができます。

このポインター値は、同じファイルを参照する次の READ ステートメントまたは CLOSE ステートメントが実行されるまで有効です。

SET オプションで指定したポインター変数が使用されるか、または SET を 省略した場合は基底付き変数の宣言内で指定したポインター変数が使用されます。 このポインター値は、同じファイルを参照する次のス

テートメント LOCATE、WRITE、 または CLOSE が実行されるまで有効です。 REFER オプションで指定された基底付き変数のコンポーネントを初期設定します。

LOCATE ステートメントは、次のレコードを構築することができるだけの大きさを持つ 区域に、ポインター変数をセットします。

LOCATE ステートメントを実行したあとは、LOCATE ステートメントによってセッ トされたポインター変数で修飾された基底付き変数に、値を直接割り当てることができます。

### 例 1

次に、位置指定モードの入力の例を示します。

```
dcl 1 Data based(P),
     2
     .
     .
     .
     ;

on endfile(In);
read file(In) set(P);
do while (¬endfile(In));
   .
   .
   .
         /* process record */
   read file(In) set(P);
end;
```

### 例 2

次に、位置指定モードの出力の例を示します。

```
dcl 1 Data based(P);
      2
      .
      .
      .
      ;

do while (More_records_to_write);
   locate Data file(Out);
   .
   .
   .            /* build record */
end;
```

# 第 12 章 ストリーム指向データ伝送

この章では、ストリーム指向データ伝送に使用される入出力ステートメントを説明します。

ストリーム指向データ伝送とレコード単位データ伝送の両方に適用される機能 (ファイル、ファイル属性、ファイルのオープンとクローズなど) については、273 ページの『第 10 章 入出力』で説明しています。

ストリーム指向データ伝送では、データ・セットは、文字フォーマット、グラフィック・フォーマット、または混合フォーマットのデータ値の連続するストリームとして処理されます。 プログラムでは、一般的にレコード境界が無視されます。 ただし、データ・セットは一連のデータ行から構成されていると見なされるので、ストリーム指向データ伝送によって作成またはアクセスされるデータ・セットにはそれぞれ、行のサイズが対応しています。 通常、1 行はデータ・セットの 1 レコードと同じですが、行サイズは必ずしもレ コード・サイズと同じではありません。

ストリーム指向データ伝送ステートメントで FILE オプションを指定する代わりに STRING オプションを指定すれば、内部でデータを移動させることができます。 STRING オプションを指定したときは入出力操作ではありませんが、この章では STRING オプションの使用方法も説明します。

ストリーム指向データ伝送は、リスト・ディレクティブ伝送、データ・ディレクティブ伝送、編集ディレクティブ伝送の 3 種類に分けられます。

**リスト・ディレクティブ・データ伝送**
データ・リスト項目の値を伝送します。 ストリーム内の値のフォーマットを指定する必要はありません。 それらの値は、外部では、ブランクまたはコンマで区切られた定数のリストとして記録されます。

**データ・ディレクティブ・データ伝送**
データ・リスト項目の値だけでなく、それらの名前も伝送されます。ユー ザーがストリーム内の値のフォーマットを指定する必要はありません。 たとえ DBCS データが存在しなくても、変数名に DBCS 文字が含まれる場合は、 ENVIRONMENT 属性の GRAPHIC オプションを指定する必要があります。

**編集ディレクティブ・データ伝送**
データ・リスト項目の値が伝送されますが、ユーザーがストリーム内の値のフォーマットを指定する必要があります。 それらの値は、外部では、フォーマット・リストに従って 1 文字ずつ (またはグラフィック 1 文字ずつ) 処理される文字または漢字ストリングとして記録されます。

以降のセクションでは、データ伝送ステートメントとそのオプションに関する詳細を示し、リスト・ディレクティブ、データ・ディレクティブ、および編集ディレクティブの各データの指定方法について説明します。 2 バイト文字への対応方法については、313 ページの『ストリーム入出力での DBCS データ』を参照してください。

## データ伝送ステートメント

ストリーム指向データ伝送では、GET ステートメントと PUT ステートメントを使用します。 GET ステートメントおよび PUT ステートメントを使用して処理できるのは連続ファイルだけです。

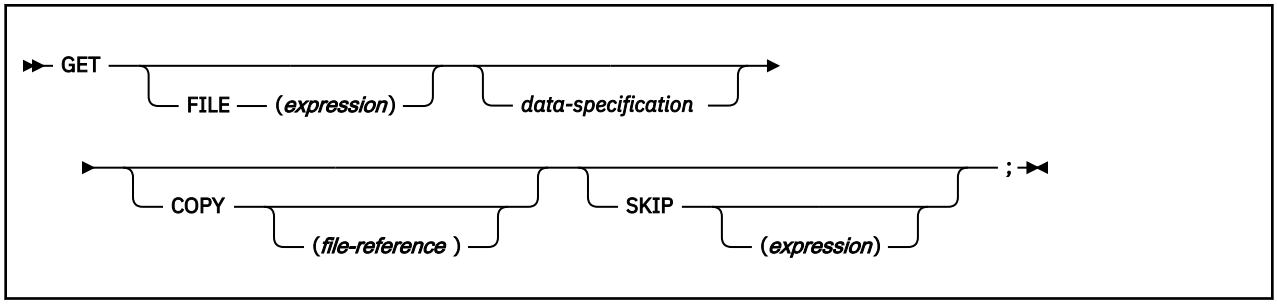
データ値が割り当てられる変数または疑似変数や、データ値の伝送元の式は、 通常、それぞれの GET または PUT ステートメントのデータ指定で指定します。 また、これらのステートメントでは、データ値の送り出し元や受け入れ先を表すオプションを指定したり、ストリーム内で先行のデータ値から相対的にどの位置にデータ値があるかを示すオプションを指定したりすることもできます。 ストリーム方式のデータ伝送ステートメントのオプションに ついては、296 ページの『データ伝送ステートメントのオプション』を参照してください。

### GET ステートメント

GET ステートメントは STREAM 入力データ伝送ステートメントであり、データ・セットまたはストリングから 1 つ以上の変数にデータ値を割り当てることができます。

ストリーム入力ファイルの場合、次の GET ステートメントの構文を 使用してください。

© Copyright IBM Corp. 1999, 2022




キーワードの指定順序は任意です。 SKIP オプションを指定しない場合、データ指定を書かなければなりません。

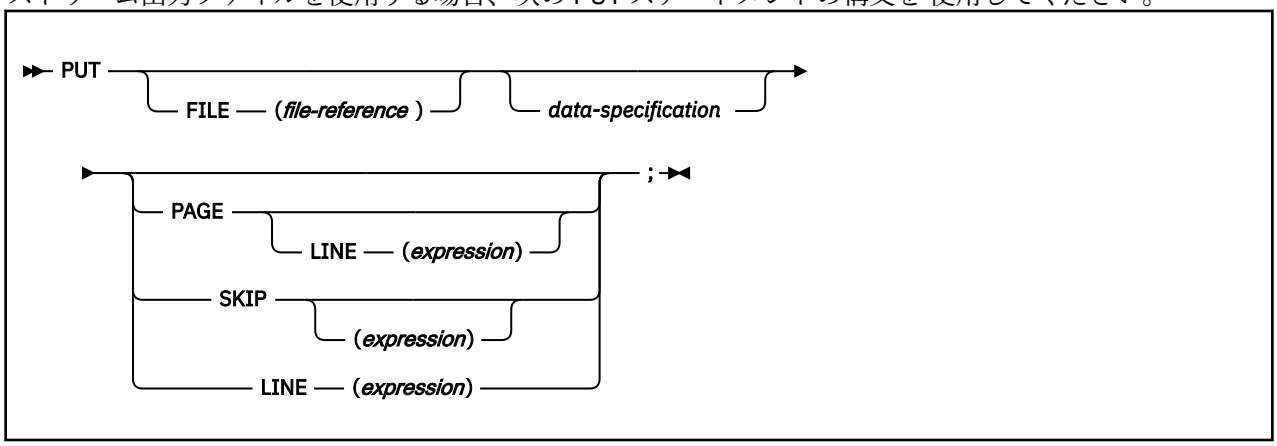
ストリングから伝送する場合、次の GET ステートメントの構文を 使用してください。

FILE オプションまたは STRING オプションを指定しない場合は、FILE(SYSIN) が想定され、SYSIN が FILE STREAM INPUT EXTERNAL として暗黙に宣言されます。

## PUT ステートメント

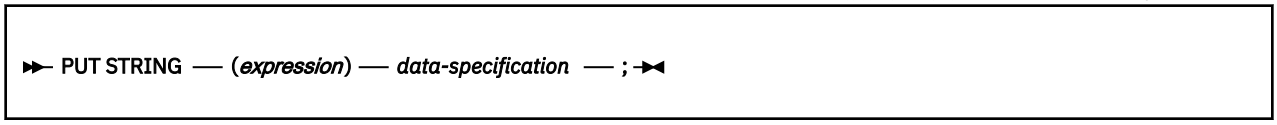
PUT ステートメントは、値をストリーム出力ファイルに伝送するか、値を文字変数に割り当てることができる、STREAM 出力データ伝送ステートメントです。

ストリーム出力ファイルを使用する場合、次の PUT ステートメントの構文を 使用してください。




キーワードの指定順序は任意です。 いずれかの制御オプション (PAGE、SKIP、または LINE) を指定するときにのみ、データ指定を省略することができます。

しかし、文字ストリングへ伝送する場合は次の PUT ステートメントの構文を 使用してください。




# データ伝送ステートメントのオプション

このセクションでは、データ伝送ステートメントで使用できるオプションについて説明します。

## COPY オプション

COPY オプションは、読み取られたソース・データ・ストリームを変更せずに、STREAM OUTPUT ファイルへ書き出すことを指定します。

ファイル参照の指定を省略すると、出力ファイル SYSPRINT が想定されます。 入力ストリーム中に新しいレコードが現れると、COPY ファイルで新しいレコードが始められます。 次の例を考えてください。

```
get file(sysin) data(A,B,C) copy(DPL);
```

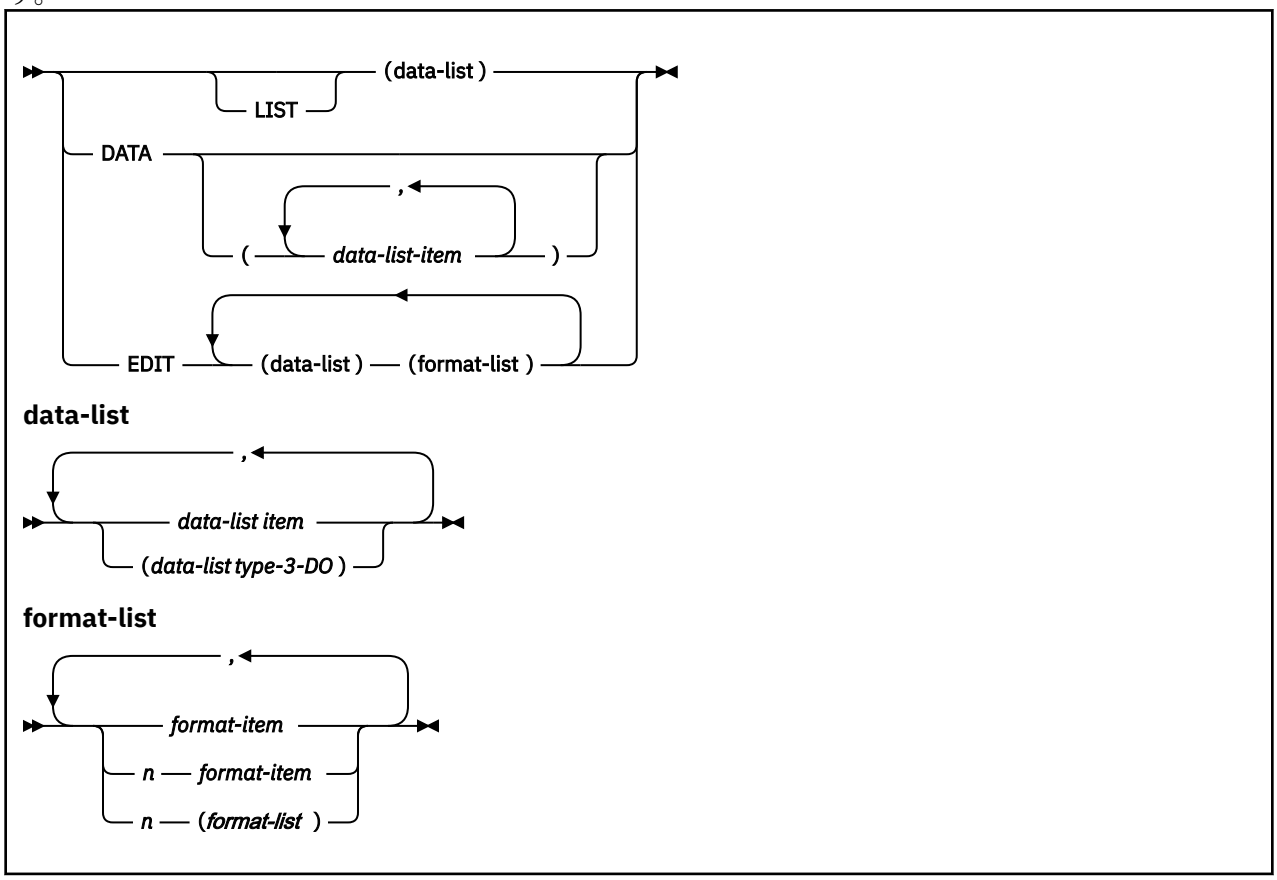
このステートメントは、入力ストリーム内の A、B、および C に割り当てられている値を、同じ名前を持つ変数に伝送するだけでなく、入力ストリーム内にある状態のままでファイル DPL に書き込みます。 入力時にスキップされ、内部変数に伝送されないデータ値も、そのまま 出力ストリーム中にコピーされます。

COPY オプションが指定された GET ステートメントの実行中になんらかの条件が起こり、ON ユニットに制御が移され、その ON ユニット内で同じファイルを参照する別の GET ステートメントが実行されたとします。次に、その ON ユニットから最初の GET ステートメントに制御が戻されると、その GET ステートメントは、COPY オプションに指定されなかった場合と同様に実行されます。 ON ユニット内で、COPY オプションに関連付けられているファイルを 参照する PUT ステートメントが実行された場合には、伝送されるデータの位置は、最後に 伝送された COPY データ項目のすぐあとになるとは限りません。

COPY オプションで指定したファイルが現行プログラム内でまだオープンされて いないときは、そのファイルは、ストリーム出力伝送用に現行プログラム内で 暗黙にオープンされます。

## データ指定オプション

GET ステートメントや PUT ステートメント内のデータ指定は、伝送されるデータを指定するためのものです。




GET または PUT ステートメントでデータ・リストを記入し、 キーワード LIST、DATA、EDIT のいずれも付けなかったときには、LIST が想定されます。

**重要 :** データ・リストの前に LIST、DATA、 または EDIT が付いていないステートメントでは、データ・リストは、GET キーワ ードまたは PUT キーワードの**直後に**なければなりません。 必須オプションは、データ・リストのあとに指定してください。

**data**
302 ページの『データ・ディレクティブのデータ指定』を参照してください。

**EDIT**
306 ページの『編集ディレクティブ・データ指定』を参照してください。

**LIST**
310 ページの『リスト・ディレクティブ・データ指定』を参照してください。

**data-list item**
入力の場合、編集ディレクティブ伝送とリスト・ディレクティブ伝送のデータ・リスト項目は、要素、配列、または構造体変数のいずれでもかまいません。 データ・ディレクティブ・データ指定のデータ・リスト項目は、要素変数、配列変数、 または構造変数のいずれでもかまいません。 データ・ディレクティブのデータ・リスト内の名前は、添え字付きまたはロケーター修飾付きであってはなりません。 ただし、修飾付き (つまり、構造体のメンバー) またはストリング・オーバーレイ定義済みの名前を 使用することはできます。

出力の場合、編集ディレクティブ・データおよびリスト・ディレクティブ・データ指定のデータ・リスト項 目は、要素式、配列式、または構造式のいずれでもかまいません。 データ・ディレクティブ・データ指定のデータ・リスト項目は、要素変数、配列変数、 または構造変数のいずれでもかまいません。 ロケーター修飾付きであってはなりませんが、 修飾付き (つまり、構造体のメンバー) またはストリング・オーバーレイ定義済みの名前を使用することはできます。

データ・リスト項目のデータ・タイプとしては、任意の計算データを使用できます (ただし、GET ステートメントは例外で、VALUE 属性を持つことができません)。また PUT ステートメントでは、データ・タイプとして POINTER も使用できます。 データ・タイプが非計算タイプである場合、項目の内容は、HEX 組み込み関数を項目に適用して項目が指定された場合のように伝送されます (PUT DATA の場合、その 16 進値は引用符で囲まれ、その後に接尾部 BX が付けられます)。

データ・リスト内の配列変数または構造変数は、データ・リスト内に *n* 個の項目 (*n* は、その配列または構造体のエレメント項目の数) を書くのと同じことです。 編集ディレクティブ伝送のときは、それぞれのエレメント項目の 1 つがデータ・フォーマット項目の別々の使用方法に関連付けられます。

**data-list type-3-DO**
Type 3 DO 指定の構文は、207 ページの『DO ステートメント』で説明しています。 type 3 DO が指定されたデータ・リスト項目は、GET ステートメントの データ・ディレクティブ・データのリストでは使用できません。

最後の反復指定が処理されると、次のデータ・リスト項目の処理が続けられます。

構文図で示されているように、それぞれの反復指定は括弧で囲まなければなりません。 データ指定として 1 つの反復指定だけを書くときは、データ・リストを囲むため の括弧と反復指定を囲むための括弧が別々に必要になるので、外側に 2 組の括弧が必要になります。

反復指定がネストされているときは、最右端の DO が、ネスティングの外側のレベルになります。 次の例を考えてください。

```
get list (((A(I,J)
     do I = 1 to 2)
     do J = 3 to 4));
```

上の例では、添え字を区別するための括弧の他に、三組の括弧があります。 最外部の一組の括弧は、データ指定に必要な括弧です。 次の一組は、外側の反復指定に必要な括弧です。 3 番目の一組は、内側の反復指定に必要な括弧です。

このステートメントは、次のネストされた DO グループと機能的に同等です。

```
do J = 3 to 4;
   do I = 1 to 2;
     get list (A (I,J));
   end;
end;
```

上の例では、次の順序で、配列 A のエレメントに値が割り当てられます。

```
A(1,3), A(2,3), A(1,4), A(2,4)
```

**フォーマット・リスト (format list)**
format-list (フォーマット・リスト) の説明については、306 ページの『編集ディレクティブ・データ指定』を 参照してください。

## FILE オプション

FILE オプションは、行われる操作の対象となるファイルを指定します。 そのファイルは、STREAM ファイルでなければなりません。

ファイル・タイプ・データ項目の宣言方法については、274 ページの『ファイル』を 参照してください。

GET ステートメントに FILE オプションあるいは STRING オプションのいずれも指定されていない場合は、入力ファイル SYSIN がデフォルトです。PUT ステートメントにいずれも指定されていない場合は、出力ファイル SYSPRINT がデフォルトです。

## LINE オプション

LINE オプションは、PRINT ファイルの場合にのみ指定することができます。 LINE オプションは、データ・セットの新しい現在行を定義します。

式が計算され、整数値 *n* に変換されます。 式の値は、2G より小さくなければなりません。 新しい現在行は、現行ページの *n* 行目になります。 現行ページで少なくとも *n* 行がすでに書き出されている場合や、OPEN ステート メントの PAGESIZE オプションで指定した限界値より *n* の方が大きい 場合は、ENDPAGE 条件が起こります。 *n* がゼロ以下の場合には、値として 1 が使用されます。 *n* が現在行を指定していると、ENDPAGE 条件が起こります。ただし、フ ァイルが 1 桁目に位置づけられているときは例外で、その場合は、SKIP(0) オプ ションを指定したときと同じ結果になります。

データ指定 (ある場合) によって定義された値が伝送される前に、LINE オプションの効力が生じます。 1 つの PUT ステートメントで PAGE オプションと LINE オプションの両方を 指定したときは、まず PAGE オプションが適用されます。 次の例を考えてください。

```
put file(List) data(P,Q,R) line(34) page;
```

このステートメントは、変数 P、Q、および R の値を、新規ページの 34 行目から、データ・ディレクティブ・フォーマットで印刷します。

ファイルのオープン後、最初の GET ステートメントで LINE オプションを 指定したときの効果については、281 ページの『OPEN ステートメント』を参照してください。

対話モードで端末に出力する場合、LINE オプションを指定すると 3 行スキップされます。

## PAGE オプション

PAGE オプションは、PRINT ファイルの場合にのみ指定することができます。 このオプションは、データ・セット内の新しい現行ページを定義します。

1 つの PUT ステートメントで PAGE と LINE の両方を指定したときは、まず PAGE オプションが 適用されます。 データ指定 (ある場合) によって定義された値が伝送される前に、PAGE オプションの効力が生じます。

PAGE オプションが指定された PUT ステートメントが実行されるか、PAGE フォーマット項目 が検出されるか、または ENDPAGE 条件が起こるまで、現行ページのままです。上 記のいずれかが起こると、新しいページが定義されます。 新しいページでは 1 行目が暗黙に定義されます。

対話モードで端末に出力する場合、PAGE オプションを指定すると 3 行スキップされます。

## SKIP オプション

SKIP オプションは、データ・セット内の新しい現在行 (またはレコード) を指定します。

式が計算され、整数値 *n* に変換されます。 データ・セットは、現在行 (レコード) から数えて *n* 番目の行 (レコード) の先頭に位置付けられます。 *expression* (式) を省略すると、デフォルトとして SKIP(1) が使用されます。

データ指定 (ある場合) によって定義された値が伝送される前に、SKIP オプションの効力が生じます。 次の例を考えてください。

```
put list(X,Y,Z) skip(3);
```

このステートメントは、変数 X、Y、および Z の値を、出力ファイル SYSPRINT の現在行の後の 3 行目から印刷します。

PRINT 以外のファイルや入力ファイルの場合に、SKIP オプションの式の値がゼロ以下のときには、1 の値が使用されます。 PRINT ファイルの場合に、*n* がゼロ以下であると、現在行の先頭に位置づけられます。

ファイルのオープン後、最初の GET ステートメントで SKIP オプションを 指定したときの効果については、281 ページの『OPEN ステートメント』を参照してください。

SKIP(n) が発行されたときに現行ページに *n* 行未満しか残っていないと、ENDPAGE 条件が起こります。

会話モードを使って端末で印刷する場合、SKIP(n) の *n* が 3 より大きいときは、SKIP(3) と 見なされます。 3 行より多くスキップすることはできません。

## STRING オプション

GET ステートメントおよび PUT ステートメントで STRING オプションを指定すると、主記憶域とデ ータ・セットの間をデータが伝送されるのではなく、主記憶域のある記憶場所 から別の記憶場所へデータが伝送されます。 DBCS データ項目は、STRING オプションを指定して使用することはできません。

STRING オプションが指定された GET ステートメントは、文字ストリングに変換したあとで、 式から得られるデータ値をデータ・リスト項目に割り当てることを指定します。 このオプションを使用する各 GET 操作は、常に、そのストリングの左端の文字位置から開始します。 そのストリングの文字数が、データ指定で指定した文字の総数より少ないときは、ERROR 条件が起こります。

STRING オプションが指定された PUT ステートメントは、データ・リスト項目の値 を、指定された文字変数または疑似変数に割り当てることを指定します。 PUT 操作は、必要に応じて変換を行ったあと、ストリングの左端の文字位置から 値の割り当てを開始します。 ブランクや区切り文字は、通常の入出力操作と同様に挿入されます。 ストリングにデータが入りきらないときは、ERROR 条件が起こります。

GET DATA ステートメントに STRING オプションが付いているときは、NAME 条件は起こりません。 その代わりに、GET DATA ステートメントに FILE オプションがついており、NAME 条件が起こる というような状況では、ERROR 条件が起こります。

STRING オプションには、次のような制約事項があります。

- COLUMN 制御フォーマット・オプションと STRING オプションを併用することはできません。
- PUT ステートメントの STRING オプションに疑似変数を指定することはできません。

STRING オプションは編集ディレクティブ伝送で使用する場合に最も役立ちます。 このオプションを使用すれば、データ収集またはデータ分散処理を 1 つのステー トメントで行うことができ、また、レコード単位のステートメントによって伝送される文字ストリングをストリーム指向で処理することもできます。

次の例を考えてください。

```
read file (Inputr) into (Temp);
get string(Temp) edit (Code) (F(1));
If Code = 1 then
   get string (Temp) Edit (X,Y,Z)
      (X(1), 3 F(10,4));
```

この READ ステートメントは、入力ファイル Inputr からレコードを読み取ります。 最初の GET ステートメントは STRING オプションを使用して、そのレコードの最初 のバイトからコードを取り出し、それを Code に割り当てます。 コードが 1 の場合、2 番目の GET ステートメントは STRING オプションを使用して、レコード内の値を X、Y、および Z に割り当てます。 2 番目の GET ステートメントは、ストリング Temp の最初の文字が無視されることを指定します (フォーマット・リストの X(1) フォーマット項目)。 無視された文字は、最初の GET ステートメントで Code に割り当てられた文字と同じものです。

PUT ステートメント内の STRING オプションの例を以下に示します。

```
put string (Record) edit
   (Name)    (X(1), A(12))
   (Pay#)    (X(10), A(7))
   (Hours*Rate) (X(10), P'$999V.99');

write file (Outprt) from (Record);
```

PUT ステートメントはスペース処理フォーマット項目 X(1) を使用して、文字 変数に割り当てられる最初の文字が 1 つのブランクであることを指定しています。 このブランクは、印刷前にシングル・スペースの行送りをすることを表す ANS 縦方向キャリッジ位置指定文字 コードです。 このあと、変数 Name の値、変数 Pay# の値、および 式 Hours*Rate の値が割り当てられます。 WRITE ステートメントは、レコード単位の伝送を使用してレコードを ファイル Outprt に書き出すように指定します。

STRING オプションで参照した変数を、データ・リスト内の名前や別名として 使用してはなりません。 次の例を考えてください。

```
declare S char(8) init('YYMMDD');
put string (S) edit
    (substr (S, 3, 2), '/',
    substr (S, 5, 2), '/',
    substr (S, 1, 2))
    (A);
```

この PUT ステートメントが実行されると、S の値は 'MM/DD/YY' ではなく、'MM/bb/MM' になります。最初のデータ項目の伝送後に S がブランクにされるためです。 STRING オプションで指定した変数にもとづく基底付き変数または定義済み変数 をデータ・リスト内に書いた場合も、上記と同じような結果になります。

# データ・リスト項目の伝送

データ・リスト項目の伝送は、データ・リスト項目タイプに応じて異なる方法で処理されます。

データ・リスト項目が複素数モードのときは、実数部分が伝送されたあと虚数部分が伝送されます。

データ・リスト項目が配列式の場合は、 配列のエレメントは行の大きい順に伝送されます。すなわち、 配列の右端の添え字が最も頻繁に変化します。

データ・リスト項目が構造式のときは、構造体のエレメントは、構造体の宣言で指定された順序で伝送されます。

### 例 1

この例は以下のステートメントに基づいています。

```
declare 1 A (10),
        2 B,
        2 C;
put file(X) list(A);
```

これらのステートメントの結果、出力は次のような順序になります。

```
A.B(1) A.C(1) A.B(2) A.C(2) A.B(3)
    A.C(3)…
```

ただし、宣言は以下のように指定されているとします。

```
declare 1 A,
          2 B(10),
          2 C(10);
```

同じ PUT ステートメントを使用しても、次の順序で出力されます。

```
A.B(1) A.B(2) A.B(3) … A.B(10)
A.C(1) A.C(2) A.C(3) … A.C(10)
```

**例 2**

リスト・ディレクティブ伝送または編集ディレクティブ伝送の入力ステートメントで 使用するデータ・リスト内で、 ある変数に値が割り当てられ、 同じデータ・リスト内の後方でこの変数が参照されると、 その新しい値が使用されます。 次の例を考えてください。

```
get list (N,(X(I) do I=1 to N),J,K,);
     substr (Name, J,K));
```

このステートメントが実行されると、下記の順序で値が伝送され、割り当てられます。

1. 新しい値が N に割り当てられます。
2. 割り当てられた項目の数を指定する新しい値 N を使用して、X(1),X(2),…X(N), の順序で反復指定で指定されたとおりにエレメントが配列 X に割り当てられます。
3. 新しい値が J に割り当てられます。
4. 新しい値が K に割り当てられます。

# データ・ディレクティブのデータ指定

データ・リスト項目内の構造体エレメントの名前は、あいまいさが生じない程度に修飾すればよいので、完全に修飾する必要はありません。 データ・リストを省略すると、データ・ディレクティブ・ステートメントで指定可能なすべて の計算関係の変数からなるデータ・リストが想定されます。

出力の場合は、データ・リスト内のすべての項目が伝送されます。

DATA データ指定の構文については、297 ページの『データ指定オプション』を参照してください。

## データ・ディレクティブ・データの制限

データ・ディレクティブ・データ伝送をプログラムで使用する際には、以下の制限が適用されます。

添え字付きの変数はデータ・ディレクティブ入力では、使用できません。

データ・ディレクティブ入出力用のデータ・リストにおける基底付き変数の参照を、明示的にロケーター修飾することはできません。 次の例を考えてください。

```
dcl Y based(Q), Z based;
put data(Y);
```

変数 *Z* は伝送できません。伝送するには、明示的にロケーターで修飾される必要があります。

データ・リストの基底付き変数には、以下の制限があります。

- 変数は、OFFSET 変数を基底とすることはできません。
- 変数を基底付きにするポインターは、DEFINED ストレージであることはできません。
- 変数を基底付きにするポインターが BASED 自身である場合は、基底付き ポインターのチェーンは、BASED でも DEFINED でもないポインターで 終了しなければなりません。

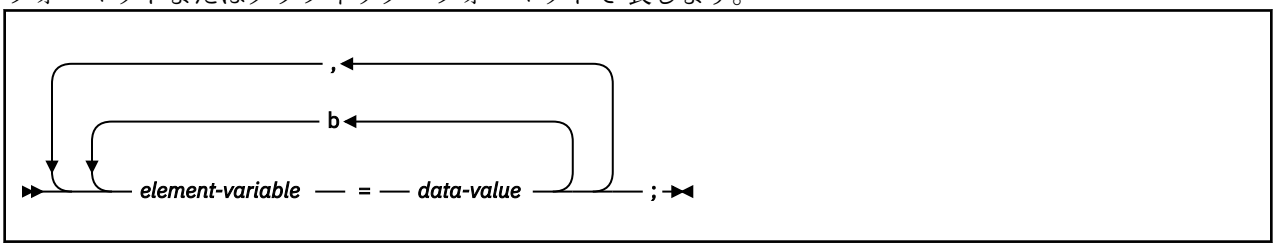
データ・リストにある定義済み変数は、以下の要件を満たしていなければなりません。

- ピクチャー変数または文字変数でなければならない。
- 被制御変数上で定義されていてはならない。
- エレメントまたは配列のクロスセクション上で定義されていてはならない。
- 非定数 POSITION 属性で定義されていてはならない。

タイプ付き構造体は、GET DATA ステートメントでは使用できませんが、PUT DATA ステートメントでは使用できます。

## データ・ディレクティブ・データの構文

データ・ディレクティブ・データの伝送を行うときのストリームは、エレメント割り当てのリストという形をとります。 エレメント割り当ては定数の符号の有無にかかわらず、変数名や等号と同じように、文字フォーマットまたはグラフィック・フォーマットで 表します。




入力の場合は、エレメント割り当てをブランクまたはコンマのどちらかで区切ることができます。 修飾された名前の中のピリオド、添え字、添え字の括弧、および代入記号の前後には、 ブランクがいくつあってもかまいません。 出力の場合は、エレメント割り当てはブランクで区切られます。 PRINT ファイルの場合、項目はプログラムのタブ設定値に従って区切られます。

ストリーム中のそれぞれのデータ値は、リスト・ディレクティブ伝送のところで説明したいずれかの構文フォーマットをとります。 リスト・ディレクティブ伝送の構文についての説明は、310 ページの『リスト・ディレクティブ・データの構文』を 参照してください。

ストリーム中のデータ値の長さは、対応する変数の宣言されている属性 および (名前も含まれているので) 完全に修飾された添え字付きの名前の長さによって決まります。 コード化算術データ、数字データ、ビット・ストリング・データから変換された 出力項目の長さは、リスト・ディレクティブ伝送での出力データの長さと 同じであり、73 ページの『第 4 章 データ変換』で説明した文字タイプへの データ変換の規則に従います。

入力ストリーム中の修飾された名前は、完全に修飾されていなければなりません。

ストリーム中の修飾された名前に介在添え字が付いていてはなりません。 例えば、Y が次のように宣言されているとします。

```
declare 1 Y(5,5),
          2 A(10),
            3 B,
            3 C,
            3 D;
```

この場合は、エレメント名をストリーム中に入れるときは、次のようなフォーマットでなければなりません。

```
Y.A.B(2,3,8)= 8.72
```

## GET データ・ディレクティブ

このトピックには、データ・ディレクティブ・データ伝送での GET ステートメントの使用に関する情報があります。

データ・リストを使用するときは、各データ・リスト項目は要素変数、配列変数、または構造変数でなければなりません。 データ・リスト内の名前に添え字が付いていてはなりませんが、修飾された名前を使用することはできます。 ストリーム内のすべての名前は、データ・リストに現れている必要があります。 ただし、名前の順序は同じである必要はなく、 またデータ・リストはストリーム内にない名前を含むことができます。

データ・リスト内の名前がストリーム中にない場合は、その名前を持つ変数の値は変りません。

データ・リスト内にない名前や識別不能な要素変数がストリーム中にあると、NAME 条件が起こります。

引用符で囲まれていないセミコロンが検出されるか、ファイルの終わりに達すると、伝送は終了します。セミコロンまたはファイルの終わりの認識により、データ・リストが指定されているかどうかにかかわらず、特定のステートメントによって実際に伝送された要素割り当て数が判別されます。

例えば、次のデータ・リストを考えてみましょう。 A、B、C、 D はそれぞれ要素変数の名前です。

```
Data (B, A, C, D)
```

このデータ・リストを、次のような入力データ・ストリームに関連付けたとします。

```
A= 2.5, B= .0047, D= 125, Z= 'ABC';
```

C はデータ・リスト内に書かれていますが、ストリーム中にはないため、値は変わりません。Z はデータ・リストにないので、NAME 条件が発生します。

データ・リスト内に配列名を書く場合、添え字付きの名前をデータ・リスト内に 書くことはできませんが、ストリーム中にはその配列の添え字付き参照があって もかまいません。 配列全体がストリーム内に現れる必要はありません。実際にストリーム内に現れるこれらのエレメントのみが割り当てられます。 添え字が範囲外であるか、欠落しているときは、NAME 条件が起こります。

次の例を考えてください。

X が以下のように宣言されているとします。

```
declare X (2,3);
```

このとき、下記のデータ・リストと入力データ・ストリームについて考えてみましょう。

| データ指定 | 入力データ・ストリーム |
|---|---|
| data (X) | X(1,1)= 7.95,<br>X(1,2)= 8085,<br>X(1,3)= 73; |

データ・リストには配列名しか書いていませんが、入力ストリームには配列の個々のエレメントの値を入れることができます。 この場合は、3 つのエレメントのみが割り当てられます。配列のその他の部分は変更されません。

データ・リスト内に構造体、小構造体、または構造体エレメントの名前を書く場合、 データ・リスト内では完全に修飾する必要はありませんが、ストリーム中には完全修飾名を入れなければなりません。 次の例を考えてください。

```
dcl 1 In,
      2 Partno,
      2 Descrp,
      2 Price,
        3 Retail,
        3 Whsl;
```

In.Price.Retail の中に値を読み込むときは、入力データ・ストリ ームは次のフォーマットになっていなければなりません。

```
In.Price.Retail=1.23;
```

データ指定は、下のリストのいずれでもかまいません。

```
data(In)
data(Price)
data(In.Price)
data(Retail)
data(Price.Retail)
data(In.Retail)
data(In.Price.Retail)
```

**関連情報**

295 ページの『GET ステートメント』

GET ステートメントは STREAM 入力データ伝送ステートメントであり、データ・セットまたはストリングから 1 つ以上の変数にデータ値を割り当てることができます。

# PUT データ・ディレクティブ

このトピックには、データ・ディレクティブ・データ伝送での PUT ステートメントの使用に関する情報があります。

データ・リスト項目は、要素変数、配列変数、構造変数、反復指定のどれでもかまいません。 データ・リスト内に書かれている名前が、その値と一緒に、ブランクで区 切られたセミコロンで終了するエレメント割り当てのリストの形で伝送されます。 PRINT ファイルの場合は、項目はプログラムのタブ設定値によって 区切られます (312 ページの『PRINT 属性』を参照してください)。

それぞれの PUT ステートメントで伝送された最後のデータ項目のあとで、セミコロンがストリーム中に書き出されます。

名前は混合ストリングとして伝送されます。 混合ストリングには、SBCS 文字、DBCS 文字、またはその両方を使用できます。 DBCS フォーマットで表現された SBCS 文字がある場合は、まずそれが SBCS に変換されます。 例えば、`put data (<.A>B<.Ckk>);` は次のように伝送されます。

```
ABC<kk>=value-of-variable
```

**注 :** この例では、<.A>B<.Ckk> はスカラー変数です。

数字変数の文字ストリング値が、有効な算術定数 (符号付きまたは符号なしの) もしくは 複素数式を表していないとき、データ・ディレクティブ出力の出力結果を後続のデータ・ディレクティブ 入力で使用することはできません。

文字データの場合、文字ストリングの内容は引用符で囲まれて書き出されます。 文字ストリング内にあるそれぞれの引用符は、連続する 2 つの引用符で表されます。

### 例 1

以下の例は、データ・ディレクティブ伝送 (入力と出力の両方) を示しています。

```
declare (A(6), B(7)) fixed;
get file (X) data (B);
do I  = 1 to 6;
     A (I) = B (I+1) + B (I);
end;
put file (Y) data (A);
```

**入力ストリーム:**

```
B(1)=1, B(2)=2, B(3)=3,
B(4)=1, B(5)=2, B(6)=3, B(7)=4;
```

**出力ストリーム:**

```
A(1)= 3 A(2)= 5 A(3)= 4 A(4)= 3
A(5)= 5 A(6)= 7;
```

### 例 2

```
dcl 1 A,
      2 B FIXED,
      2 C,
        3 D FIXED;
A.B = 2;
A.D = 17;
put data (A);
```

出力ストリーム中のデータ・フィールドは次のようになります。

```
A.B= 2 A.C.D= 17;
```

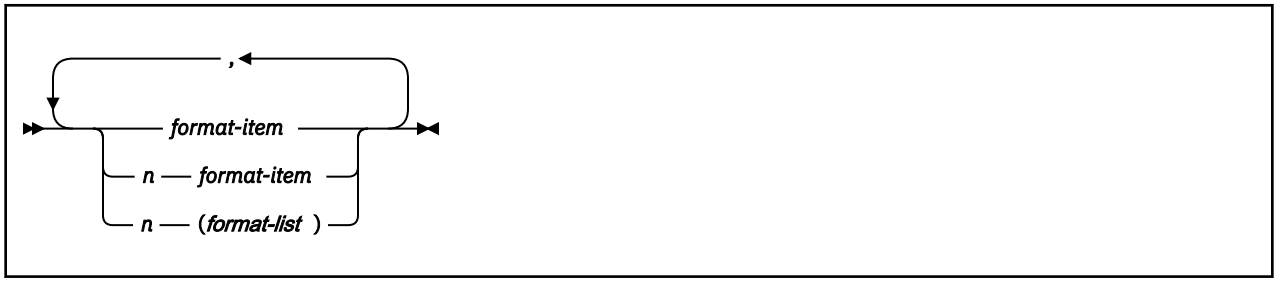
**関連情報**

296 ページの『PUT ステートメント』
PUT ステートメントは、値をストリーム出力ファイルに伝送するか、値を文字変数に割り当てることができる、STREAM 出力データ伝送ステートメントです。

## 編集ディレクティブ・データ指定

編集ディレクティブ・データ指定は、ストリーム出力のフォーマットを容易にします。

EDIT データ指定の構文については、297 ページの『データ指定オプション』を参照してください。




**n**

反復因数を指定します。括弧で囲んだ式か、整数のどちらかです。 整数を指定するときは、整数とそのあとのフォーマット項目 (format-item) をブランクで区切らなければなりません。

反復因数は、関連づけられたフォーマット項目またはフォーマット・リストが、連続して *n* 回使用されることを指定します。 ゼロまたは負数の反復因数は、対応するフォーマット項目またはフォーマット・リストをスキップ し、使用しないことを表します (そのデータ・リスト項目は次のデータ・フォーマット項目に関連付けられます)。

反復因数を式で表したときは、その式が計算され、整数に変換されます。この計算と変換は、一組の繰り返しにつき 1 回だけ行われます。

反復因数に対応するフォーマット項目またはフォーマット・リストは、反復因数のすぐ右に書かれている項目または項目リストです。

**フォーマット項目**

データ・フォーマット項目、制御フォーマット項目、またはリモート・フォーマット項目を指定します。 構文およびフォーマット項目について詳しくは、315 ページの『第 13 章 編集ディレクティブのフォーマット項目』を参照してください。

**データ・フォーマット項目**

単一データ項目の文字表記またはグラフィック表記を記述するものです。

**A**
文字

**B**
bit

**C**
複合

**E**
浮動小数点数

**F**
固定小数点

**G**
graphic

**L**
明細

**P**
ピクチャー

**V**
行の表示

**制御フォーマット項目**
ファイルに関連付けられているデータ・セットのレイアウトを指定するものです。

COLUMN
明細
PAGE
SKIP
X

**リモート・フォーマット項目**
別の場所にある FORMAT ステートメントのラベル定数を値として持つラベル参照を指定するものです。 その FORMAT ステートメント内にその場所に適応するフォーマット項目を書きます。 ラベル参照項目は以下のとおりです。

**R(ラベル参照)**
このラベルとは、FORMAT ステートメントのラベル定数名のことです。 R フォーマット項目の指定については、322 ページの『R フォーマット項目』を参照してください。

最初のデータ・フォーマット項目は最初のデータ・リスト項目に、2 番目のデータ・フォーマット項目 は 2 番目のデータ・リスト項目に、というように対応づけられます。 フォーマット・リスト内のデータ・フォーマット項目の数が、関連データ・リスト内の項目の数 より少ないときは、そのフォーマット・リストが再使用されます。 余分なフォーマット項目は無視されます。

例えば、フォーマット・リスト内に 5 つのデータ・フォーマット項目があり、関連データ・リストでは、10 個の項目を伝送することを指定したとします。 このとき、データ・リスト内の 6 番目の項目は最初のデータ・フォーマット項目に対応づけられます (以下同様)。 フォーマット・リスト内に 10 個のデータ・フォーマット項目があり、関連データ・リストには 5 つの項目しかないとすれば、6 番目から 10 番目までのフォーマット項目は無視されます。

制御フォーマット項目が検出されると、制御処置が実行されます。

PAGE 制御フォーマット項目と LINE 制御フォーマット項目は、PRINT ファイルの場合にのみ使用でき、したがって PUT ステートメント内でだけ使用できます。 SKIP、COLUMN、および X フォーマット項目は、入力のときも出力のときも使用できます。

フォーマット項目 PAGE、SKIP、および LINE はそれぞれ、PUT ステート メント (SKIP の場合は GET ステートメントも) の対応するオプションと同じ働きをします。 ただし、フォーマット項目は、フォーマット・リストで検出されたときに効力を生じますが、オプションは、 データが伝送される前に効力を生じる点が異なっています。

COLUMN フォーマット項目は、GET STRING ステートメントまたは PUT STRING ステートメントで 使用することはできません。

ファイルのオープン後､最初の GET または PUT ステートメントで 制御フォーマット項目を指定したときの効果については、281 ページの『OPEN ステートメント』を 参照してください。

変数の中に読み込まれた値を、同じデータ・リスト内の後方にある別の変数に対応するフォーマット項目内で 使用することができます。

```
get edit (M,String_A,I,String_B)(F(2),A(M),X(M),F(2),A(I));
```

この例では、最初の 2 文字が M に割り当てられます。 この M の値は、String_A に割り当てられる文字の個数、 および I に 2 文字が割り当てられる前に無視される文字の個数を表します。 I に割り当てられた値は、String_B に割り当てられる文 字の個数を示すために使用されます。

入力処理中に変数に割り当てられた値を、後方にあるデータ項目に対応するフォーマット項目内の式で使用することができます。 フォーマット項目内の式は、そのフォーマット項目が使用されるたびに計算され、整数に変換されます。

最後のデータ・リスト項目の処理が終わると、伝送が完了します。 後続のフォーマット項目 (制御フォーマット項目も含めて) は無視されます。

# GET 編集ディレクティブ

このトピックには、編集ディレクティブ・データ伝送での GET ステートメントの使用に関する情報があります。

ストリーム中のデータは、文字やグラフィックの連続したストリングであり、ある値と次の値の間に区切り文字はありません。 それぞれのデータ値の文字数は、フォーマット・リスト内のフォーマット項目によって指定します。 それらの文字は、対応するフォーマット項目に従って解釈されます。 データ・リストの処理が終わると、GET ステートメントの実行が終わり、フォーマット項目が残っていてもそれらは無視されます。

それぞれのデータ・フォーマット項目は、データ・リスト項目に関連付けられる文字または グラフィックの個数を表し、また、データ値をどのように解釈するかも表します。 データ値は、必要に応じて変換され、関連データ・リスト項目に割り当てられます。

入力ストリーム中の固定小数点 2 進データ値と浮動小数点 2 進データ値は、必ず 10 進数で表記しなければなりません。 F、P、E のフォーマット項目を使用してそれらの値をアクセスすれば、それらの値は割り当ての時点で 2 進表記に変換されます。

ストリーム中のブランクや引用符はすべて、文字と見なされます。 ストリングを引用符で囲まないこと、引用符を二重にしないこと、および ビット・ストリングを識別するために文字 B や漢字ストリングを識別するために文字 G を使用しないことに注意してください。 ストリーム中の文字を、指定された方法で解釈できない場合には、CONVERSION 条件が起こります。

### 例

```
get edit (Name, Data, Salary)(A(N), X(2), A(6), F(6,2));
```

- ストリーム中の最初の N 個の文字は、文字ストリングと見なされ、Name に割り当てられます。
- 次の 2 文字はスキップされます。
- その次の 6 文字は、文字フォーマットで Data に割り当てられます。
- さらに次の 6 文字は 10 進固定小数点定数 (符号の有無は任意) と見なされ、Salary に割り当てられます。

### 関連情報

295 ページの『GET ステートメント』
GET ステートメントは STREAM 入力データ伝送ステートメントであり、データ・セットまたはストリングから 1 つ以上の変数にデータ値を割り当てることができます。

# PUT 編集ディレクティブ

編集ディレクティブ・データ伝送では、各データ・リスト項目の値は、関連するデータ・フォーマット項目で指定された文字表記またはグラフィック表記に変換され、フォーマット項目で指定された幅を持つフィールド内のストリームに入れられます。 データ・リストの処理が終わると、PUT ステートメントの実行が終わり、フォーマット項目が 残っていてもそれらは無視されます。

出力の場合、2 進数項目は 10 進数値に変換されるので、対応する F フォーマット項目 や E フォーマット項目では、変換後の 10 進数に換算してフィールドの幅と小数点の位置を指 定しなければなりません。 P フォーマット項目の場合は、これらをピクチャー指定で指定します。

出力の場合、出力ストリーム中のデータ値を区切るためにブランクが 挿入されるわけではありません。 ストリング・データは、フィールド内で左寄せされ、指定の幅を占めます。算術データは右寄せされます。 算術データを文字タイプに変換するときの規則に従うので、(先行ゼロがブランクで置き換えられるほかに) 最高 3 つの先行ブランクが挿入されることがありますが、フィールド内の変換後の算術項目には、通常、

最低 1 つはブランクが先頭に付いています。 ただし、指定したフィールド幅が短くて先行ブランクを挿入できない場合は、 先行ブランクはストリーム中に挿入されません。 指定したフィールド幅が短すぎるときは、算術項目では左側が切り捨てられ、ストリング項目では右側が切り捨てられ、 SIZE 条件または STRINGSIZE 条件が起こります。

**例 1**

```
put edit('Inventory='‖Inum,Invcode)(A,F(5));
```

この例では、'Inventory=' が Inum の値と連結されて、 その結果の文字ストリングの長さと同じ幅のフィールドに入れてストリーム中に書き出されます。 そのあと、Invcode の値が、F フォーマット項目で指定されているとおりに文 字に変換され、5 文字の幅のフィールド内で右寄せされて (先行文字はブランク)、ストリーム中に書き出されます。

**例 2**

次の例では、COLUMN、LINE、PAGE、および SKIP フォーマット項目を、相互に組み合わ せて使用する方法を示しています。

```
put edit ('Quarterly Statement')
    (page, line(2), A(19))(Acct#, Bought, Sold, Payment, Balance)
    (skip(3), A(6), column(14), F(7,2), column(30), F(7,2),
    column(45), F(7,2), column(60), F(7,2));
```

1. 見出しとして Quarterly Statement が、出力ファイル SYSPRINT の新しいページの 2 行目に書き出されます。
2. 2 行スキップします。 出力内の次の行は、見出しから 3 行目か、もしくは報告書の 5 行目になります。
3. 次の値が書き出されます。

   Acct#、 1 桁目から開始します。
   Bought、 14 桁目から開始します。
   Sold、 30 桁目から開始します。
   Payment、 45 桁目から開始します。
   Balance 60 桁目です。

**例 3**

この例では、 Name の値は N 文字のフィールド内で 左寄せされた文字ストリングとしてストリーム中に挿入されます。

```
put edit (Name,Number,City) (A(N),A(N-4),A(10));
```

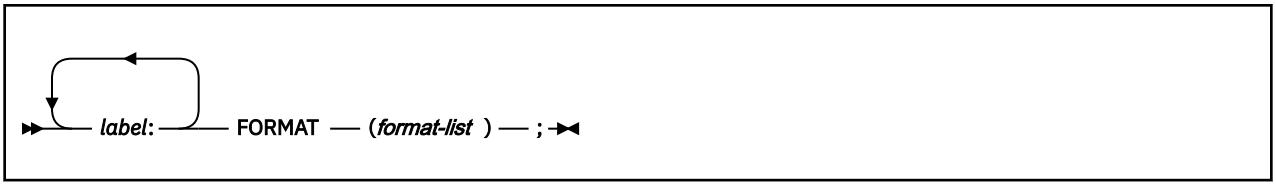
Number は N-4 文字のフィールド内で左寄せされます。 また City は 10 文字のフィールド内で左寄せされます。

**関連情報**

296 ページの『PUT ステートメント』
PUT ステートメントは、値をストリーム出力ファイルに伝送するか、値を文字変数に割り当てることができる、STREAM 出力データ伝送ステートメントです。

# FORMAT ステートメント

FORMAT ステートメントは、伝送されるデータのフォーマットを制御するために、 編集ディレクティブ・データ伝送ステートメントで使用することのできるフォーマット・リストを指定する ものです。




**label**
リモート・フォーマット・アイテム R のラベル参照と同じ。 322 ページの『R フォーマット項目』を参照してください。

**フォーマット・リスト (format list)**
306 ページの『編集ディレクティブ・データ指定』の説明に従って指定します。

GET または PUT EDIT ステートメントでは、このフォーマット・リスト・オプション内に、R フォーマット項目を記入することができます。 R フォーマット項目で表されるフォーマット・リストのその部分は、そこで識別された FORMAT ステートメントによって与えられます。

FORMAT ステートメントに関連付けられた条件接頭語は無効です。

# リスト・ディレクティブ・データ指定

リスト・ディレクティブ・データ伝送は、データ・リスト項目の値を伝送します。 このとき、ストリーム内の値のフォーマットを指定する必要はありません。

LIST データ指定の構文については、297 ページの『データ指定オプション』を参照してください。

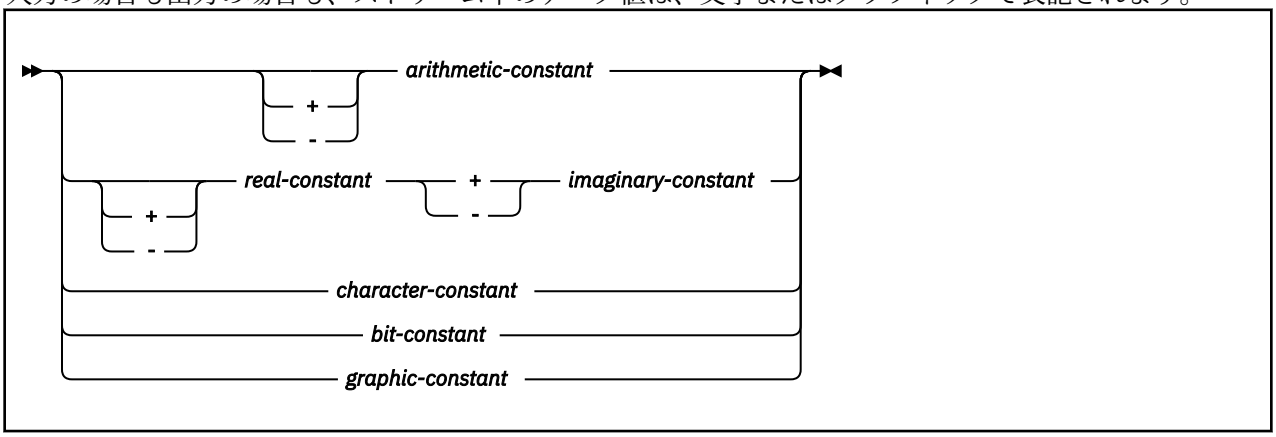
以下に、リスト・ディレクティブ・データ指定の例をいくつか示します。

```
list (Card_Rate, Dynamic_Flow)

list ((Thickness(Distance)
     do Distance = 1 to 1000))

list (P, Z, M, R)

list (A*B/C, (X+Y)**2)
```

最後の例のデータ指定には式が書かれているので、これは出力の場合にしか 使用できません。 これらの式は、ステートメントの実行時に計算され、結果の値がストリームの 中に入れられます。

## リスト・ディレクティブ・データの構文

入力の場合も出力の場合も、ストリーム中のデータ値は、文字またはグラフィックで表記されます。




ストリングの反復因数を使用することはできません。 実定数の前に付ける符号のあとや、複素数式の中央にある (+) または (-) の 前後に、ブランクがあってはなりません。

ストリーム中のデータ値の長さは、そのデータ値の属性 (精度や長さも含めて) によって決まります。 変換規則およびそれが精度に及ぼす影響の詳細については、73 ページの『第 4 章 データ変換』の 中の文字タイプへの変換の箇所で説明しています。

# GET リスト・ディレクティブ

このトピックには、リスト・ディレクティブ・データ伝送での GET ステートメントの使用に関する情報があります。

入力の場合、ストリーム中のデータ値はブランクまたはコンマで区切られなければなりません。 この分離文字の前後には、ブランクがいくつあってもかまいません。 データ・ストリーム中のブランク以外の最初の文字がコンマである場合や、2 つのコンマがブランク (いくつでも構いません) で区切られている場合、それはストリーム中のヌル・フィールドを表します。 ヌル・フィールドは、関連データ・リスト項目の値を変更せずそのままにしておくことを表します。

入力の場合、定数または複素数式のリストの伝送が終了するのは、リストが使 い尽くされたとき、またはファイルの終わりに達したときです。 定数の伝送の場合は、次の GET ステートメントに備えて、ファイルがストリームの中で位置付けられます。

項目をコンマで区切ったとき、次の GET ステートメントの実行時に走査される 最初の文字は、コンマのすぐあとの文字です。

```
Xbb,bbbXX
    –
```

項目をブランクだけで区切ったとき、走査される最初の文字は、次の非ブランク文字です。

```
XbbbbXXX
     –
```

レコードの終わりが検出されると、ファイルはレコードの終わりに位置付けられます。

```
Xbb–bbXXX
   –
```

ただし、レコードの終わりが (コンマ以外の) 非ブランク文字のすぐあとにあり、 次のレコードがブランクで始まる場合は、ファイルは次のレコード内の最初 の非ブランク文字に位置づけられます。

```
X–bbbXXX
     –
```

レコードがコンマで終っているときは、後続レコードは、次の GET ステートメン トで要求されるまで読み取られません。

データが文字定数のときは、前後の引用符が取り除かれて、囲まれていた文字が文字ストリングとして解釈されます。 二重引用符は一重引用符と見なされます。

データがビット定数のときは、前後の引用符と末尾の文字 B が取り除かれて、囲まれていた文字がビット・ストリングとして解釈されます。

データが 16 進定数 (X、BX、B4、GX) のときは、前後の引用符と接尾部が取り除かれて、囲まれていた文字が、文字、ビット、または漢字ストリングの 16 進表記として解釈されます。

データが混合定数のときは、前後の引用符と接尾部 M が取り除かれたあと、囲ま れていた定数が文字ストリングとして解釈されます。

データがグラフィック定数のときは、前後の引用符と末尾の文字 G が取り除かれ て、囲まれていたグラフィックが漢字ストリングとして解釈されます。

データが算術定数または複素数式のときは、定数によってまたは式を計算するときの規則によって暗黙に定義される基数、スケール、モード、および精度を持つコード化算術データとして解釈されます。

**関連情報**

295 ページの『GET ステートメント』
GET ステートメントは STREAM 入力データ伝送ステートメントであり、データ・セットまたはストリングから 1 つ以上の変数にデータ値を割り当てることができます。

## PUT リスト・ディレクティブ

リスト・ディレクティブ・データ伝送では、データ値の変換方法および書き出し方法は、値のタイプとファイル属性によって決まります。

データ・リスト項目の (グラフィック以外の) 値が文字表記に変換されて、データ・ストリームに伝送されます。 伝送された連続したデータ値は、ブランクで区切られます。 PRINT ファイルの場合は、項目はプログラムのタブ設定値によって区切られます (312 ページの『PRINT 属性』を参照してください)。

算術値は文字に変換されます。

2 進データ値は、10 進表記に変換されてから、ストリーム中に書き出されます。

数字値の場合は、文字値が伝送されます。

ビット・ストリングは、文字ストリングに変換されます。 その文字ストリングは引用符で囲まれて、あとに文字 B が付きます。

ストリングは、次のようにして書き出されます。

- ファイルに PRINT 属性が付いていない場合は、前後に引用符が付けられ、ストリング内の一重引用符やアポストロフィは 2 つの引用符で置き換えられます。 フィールドの幅は、ストリングの現在の長さに、付加された引用符の数を加えた長さになります。
- ファイルに PRINT 属性が付いている場合は、前後の引用符が付けられることはなく、ストリング内の一重引用符やアポストロフィが変更されることもありま せん。 フィールドの幅は、ストリングの現在の長さです。

混合ストリングは、次のようにして書き出されます。

- ファイルに PRINT 属性が付けられていない場合は、SBCS 引用符と文字 M が付けられます。 含まれている SBCS 引用符は、2 つの引用符で置き換えられます。
- ファイルが PRINT 属性を持つと、前後の引用符と文字 M が付けられることはなく、含まれている一重引用符は変更されません。

漢字ストリングは次のようにして書き出されます。

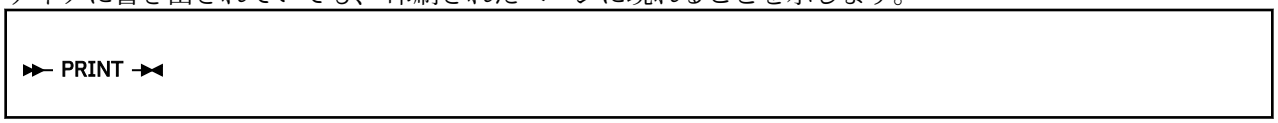
- ファイルに PRINT 属性が付けられていない場合は、前後の SBCS 引用符と 文字 G が付けられます。 前後の引用符は SBCS 引用符なので、ストリング内のグラフィック引用符は一重グラ フィック引用符で表されます (つまり、変更されません)。
- ファイルが PRINT 属性を持つと、前後の引用符と文字 G は付けられず、グラフィ ック引用符は、一重グラフィック引用符になります (つまり変更されません)。

**関連情報**

296 ページの『PUT ステートメント』
PUT ステートメントは、値をストリーム出力ファイルに伝送するか、値を文字変数に割り当てることができる、STREAM 出力データ伝送ステートメントです。

## PRINT 属性

PRINT 属性は、STREAM 属性と OUTPUT 属性を持つファイルの場合に指定できます。 この属性はそのファイルが印刷される予定であることを示します。 すなわち、ファイルと関連したデータを、 最初はほかのメディアに書き出されていても、 印刷されたページに現れることを示します。




PRINT を指定したときは、PRINT ファイルの各レコードの最初のデータ・バイトは米国標準規格 (ANS) 印刷制御文字を入れるために予約されます。 この制御文字は、PL/I によって挿入されます。

リスト・ディレクティブ・データ伝送またはデータ・ディレクティブ・データ伝送で伝送されるデータ値は、自動的に左マージンに合わせられ、また、システムで定義され、事前設定されているタブ位置に合わせられます。

PRINT ファイルのレイアウトは、313 ページの表 48 に リストされているオプションおよびフォーマット項目を使用して制御することができます。

表 *48. PRINT* ファイルのオプションとフォーマット項目

| 記述 | ステートメント・オプション | 編集ディレクティブの フォーマット項目 | 結果 |
|---|---|---|---|
| OPEN | LINESIZE(n) | – | 行幅を設定します。 |
| OPEN | PAGESIZE(n) | – | ページの行数を設定します。 |
| PUT | PAGE | PAGE | 新しいページにスキップします。 |
| PUT | LINE(n) | LINE(n) | 指定された行にスキップします。 |
| PUT | SKIP[(n)] | SKIP[(n)] | 指定された行数だけスキップします。 |
| PUT | – | COLUMN(n) | 行の中の指定された桁にスキップします。 |
| PUT | – | X(n) | 行の中にブランク桁を作って位置を設定します。 |

LINESIZE と PAGESIZE は、ページの印刷区域 (脚注を除く) の大きさを設定するためのものです。 LINESIZE オプションは、各印刷行に印刷される文字の最大文字数を表します。 PRINT ファイルの場合、このオプションを省略すると、120 文字と見なされます。 PRINT 以外のファイルではデフォルトの値はありません。 PAGESIZE オプションは各印刷ページの最大の印刷行数を指定します。指定しない場合は、デフォルトの値の 60 が使用されます。 次の例を考えてください。

```
open file(Report) output stream print PAGESIZE(55) LINESIZE(110);
on endpage(Report) begin;
     put file(Report) skip list (Footing);
     Pageno = Pageno + 1;
     put file(Report) page list ('Page '||Pageno);
     put file(Report) skip (3);
end;
```

この OPEN ステートメントは、ファイル Report を PRINT ファイルとしてオープンします。 PAGESIZE(55) という指定は、各ページに最大 55 行書き出せることを 示しています。 すでに 55 行が書き出された (またはスキップされた) あと、同じページへの書き出しが試みられると、ENDPAGE 条件が起こります。 ENDPAGE 条件が起こったとき、暗黙に取られる処置は新しいページへのスキップですが、この例に示したように、ON ステートメントを使用してユーザー独自の処置を設定することもできます。

LINESIZE(110) は、ページの各行に最大 110 文字書き出せることを示しています。 110 文字を超える行を書き出そうとすると、余分の文字は次の行に書き出されます。

56 行目に書き出そうとすると (または 55 行目を超えてスキップしようとすると)、ENDPAGE 条件が起こり、ここに示した開始ブロックが実行されます。 ENDPAGE 条件は、1 ページあたり 1 回しか起こりません。 したがって、ENDPAGE 条件が起こったあと、指定の PAGESIZE を超えて印刷を続けることができます。 例えば、各ページの最下部にフッティングを書き出すような場合には、この方法が役立ちます。

最初の PUT ステートメントでは、1 行スキップすることと、文字ストリングと推定される フッティングの値を 57 行目に印刷する (ENDPAGE 条件が起こったとき、現在行は 常に PAGESIZE+1 です) ことを指定しています。 ページ番号 Pageno が 1 つだけ増やされ、ファイル Report が 次のページにセットされ、文字定数 'Page' と 新しいページ番号が連結されて印刷されます。 最後の PUT ステートメントは 3 行をスキップして、次の印刷が行 4 になるようにします。 制御は、開始ブロックから、ENDPAGE 条件を発生させた PUT ステートメントに戻ります。 ただし、その PUT ステートメントに SKIP または LINE オプションが指定されていても、それらは影響を及ぼしません。

## ストリーム入出力での DBCS データ

リスト・ディレクティブ伝送またはデータ・ディレクティブ伝送で DBCS データを使用する場合は、 そのファイルの ENVIRONMENT 属性で GRAPHIC オプションを指定して おかなければなりません。 また、デー

タ・ディレクティブ伝送では、DBCS データがないときでも、DBCS 名を指定する場合は、GRAPHIC オプションを指定しておかなければなりません。

このとき DBCS 継続規則が適用されますが、この規則は 13 ページの『DBCS 継続規則』で説明した規則と同じものです。 編集ディレクティブ伝送でグラフィックがどのように処理されるかに ついては、306 ページの『編集ディレクティブ・データ指定』を参照してください。

# 第 13 章 編集ディレクティブのフォーマット項目

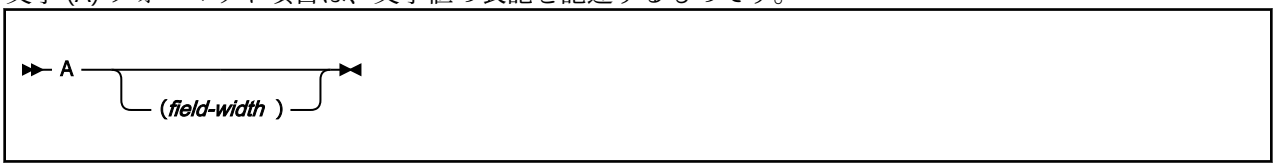
この章では、GET、PUT、または FORMAT ステートメントのフォーマット・リスト内に書くこ とができる編集ディレクティブの各フォーマット項目を説明します。 フォーマット項目は、アルファベット順に説明してあります。

**関連情報**

306 ページの『編集ディレクティブ・データ指定』
編集ディレクティブ・データ指定は、ストリーム出力のフォーマットを容易にします。

## A フォーマット項目

文字 (A) フォーマット項目は、文字値の表記を記述するものです。




***field-width***

データ・ストリーム中で、そのストリングが入っている (または、これから入る) 文字桁の数を指定します。 この式は、このフォーマット項目が使用されるたびに評価され、整数値 (負であってはならない) に変換されます。

GET EDIT ステートメントで長さを指定せずに A フォーマット項目を指定すると、コンパイラーから警告メッセージが発行され、その項目は L フォーマット項目として 扱われます (エラー・メッセージが発行され、長さ 1 が割り当てられるのではありません)。

入力の場合は、指定した文字数がデータ・ストリームから取り出され、 データ・リスト項目に割り当てられます (必要に応じて、変換、切り捨て、または埋め込みが行われます)。 入力の場合、フィールド幅を必ず指定しなければなりません。フィールド幅がゼロのときは、ストリングはヌル・ストリングになります。ストリーム中の引用符は、ストリング内の文字と見なされます。

次の例を考えてください。

```
get file (Infile) edit (Item) (A(20));
```

この GET ステートメントは、`Infile` 内にある次の 20 文字を `Item` に割り当てます。 この値は、フォーマット項目 `A(20)` で指定された文字表記から、 `Item` 用に宣言された属性によって指定される表記に変換されます。

出力の場合は、データ・リスト項目が必要に応じて文字ストリングに変換され、指定の フィールド幅に合わせて、切り捨てまたはブランクの追加が右側で行われた あと、データ・ストリーム中に書き出されます。フィールド幅がゼロのときは、データ・ストリーム中になにも書き出されません。 ストリングを囲むための引用符が挿入されることはなく、ストリング内の引用符も二重にされません。 フィールド幅を省略すると、データ・リスト項目の文字ストリングの長さに 等しい幅が (必要に応じて、73 ページの『第 4 章 データ変換』に示した規則に 従って変換後に) 想定されます。

## B フォーマット項目

ビット (B) フォーマット項目は、ビット値の文字表記を記述するものです。 各ビットは、0 または 1 という文字で表されます。

© Copyright IBM Corp. 1999, 2022

***field-width***
データ・ストリーム中で、ビット・ストリングが入っている (または、これから入る) 文字桁の数を指定します。 この式は、このフォーマット項目が使用されるたびに評価され、整数値 (負であってはならない) に変換されます。

入力の場合は、文字表記のビット・ストリングは指定のフィールド内の どこにあってもかまいません。 フィールド内のビット・ストリングの前後にブランクがあってもよく、それらの ブランクは無視されます。 ビット・ストリングがデータ・リスト項目に割り当てられるとき、必要に応じて変換が行われます。 入力の場合、フィールド幅を必ず指定しなければなりません。フィールド幅がゼロのときは、ストリングはヌル・ストリングになります。 ストリング中に 0 と 1 以外の文字 (埋め込まれたブランク、引用符、文字 B など) があると、CONVERSION 条件が起こります。

出力の場合は、文字表記のビット・ストリングが指定のフィールド内で左寄せ され、必要に応じて右側で切り捨てまたはブランクの追加が行われます。 また、必要に応じて、ビット・ストリングへの変換も行われます。 引用符は挿入されません。また、識別文字 B も挿入されません。 フィールド幅がゼロの場合、データ・ストリームに文字は入れられません。 フィールド幅を省略すると、データ・リスト項目のビット・ストリングの長さに 等しい幅が (必要に応じて、73 ページの『第 4 章 データ変換』に示した規則に 従って変換後に) 想定されます。

次の例を考えてください。

```
declare Mask bit(25);
put file(Maskfle) edit (Mask) (B);
```

この PUT ステートメントは、Mask の値を 0 と 1 から なる 25 文字のストリングとして Maskfle に書き出します。

## C フォーマット項目

複素数 (C) フォーマット項目は、複素数データ値の文字表記を記述するものです。 実数フォーマット項目 (real-format-item) で、データ・ストリーム中の 複素数データ値の実数部分と虚数部分を記述します。

***real-format-item***
F フォーマット項目、E フォーマット項目、または P フォーマット項目のいずれか 1 つを 使用して指定します。 P フォーマット項目では、数字データを記述しなければなりません。

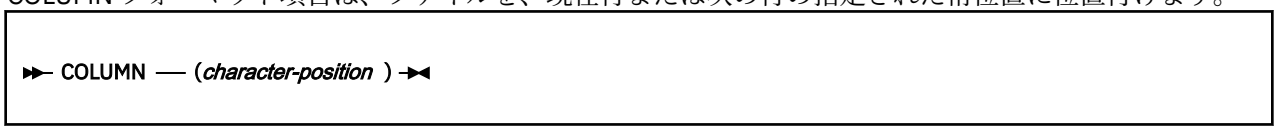
入力の場合は、入力ストリーム中に文字 I があると、CONVERSION 条件が起こります。

出力の場合は、文字 I が虚数部分に付けられることはありません。 2 番目の実数フォーマット項目 (1 つしかないときは最初の実数フォーマット項目) が F 項目か E 項目 の場合、虚数部分の値がゼロより小さいときにのみ符号が伝送されます。 実数フォーマット項目が P 項目の場合は、S、-、+ のいずれかのピクチャー文字が指定されているときにのみ符号が伝送されます。

I を付けたいときは、データ・リスト内の別個のデータ項目として、複素数項目を指定する変数のすぐあとに I を指定します。 そして、I にフォーマット項目 (A または P) を対応させます。 2 番目の実数フォーマット項目が指定された場合、これは無視されます。

## COLUMN フォーマット項目

COLUMN フォーマット項目は、ファイルを、現在行または次の行の指定された桁位置に位置付けます。




***character-position***
式を指定します。この式は、このフォーマット項目が使用されるたびに評価され、整数値 (負であってはならない) に変換されます。

ファイルが現在行の指定された桁位置をまだ通り過ぎていない場合、ファイルはその位置に位置付けられます。 ファイルがすでに指定された文字位置の後ろに位置付けされている場合は、現在行は完了し新しい行が開始されます。フォーマット項目は、その後、次の行に適用されます。

指定した桁位置が現在行の最右端の桁位置より右にある場合や、桁位置を表す 式の値が 1 より小さい場合は、桁位置 1 が想定されます。

最右端の桁位置は、次のようにして決められます。

- 出力ファイルの場合、行サイズによって決まります。
- 入力ファイルの場合、現行論理レコードの長さを使用して行サイズが決定され、これにより右端の文字位置が決まります。

GET STRING ステートメントまたは PUT STRING ステートメントで COLUMN を使用してはなりません。

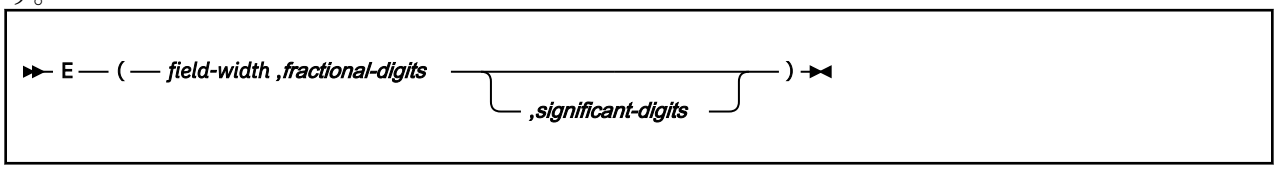
グラフィックまたはワイド文字を含む入力行や出力行に対しては、 COLUMN を使用することはできません。

入力の場合、介在文字桁は無視されます。

出力の場合、介在文字桁にはブランクが入ります。

## E フォーマット項目

浮動小数点 (E) フォーマット項目は、実数の浮動小数点 10 進算術データ値の 文字表記を記述するものです。




***field-width***
フィールド内の桁の総数を表します。 この値は、フォーマット項目が使用されるたびに評価され、整数値 *w* に変換されます。 *w* は負であってはなりません。

***fractional-digits***
小数点の右側にある小数部の桁数を表します。 この値は、フォーマット項目が使用されるたびに評価され、整数値 *d* に変換されます。 *d* は負であってはなりません。

***significant-digits***
小数部になければならない数字の桁数を表しています。 この値は、フォーマット項目が使用されるたびに評価され、整数値 *s* に変換されます。 *s* は負であってはなりません。

PUT ステートメントでは、*w* が正、*p* が最大浮動小数点精度、*e* が指数を表すために使用する桁数である場合、以下の項目が真でなければなりません。

- s > 0
- d <= p
- s <= (p + 1)
- s >= d
- d = 0 の場合、w >= s+e+2
- d > 0 かつ s > d の場合、w >= s+e+3
- d > 0 かつ s = d の場合、w >= s+e+4

*w*、*d*、および *s* の値は、それぞれ field-width、fractional-digits、および significant-digits を表します。 *e* の値は、DEFAULT コンパイラー・オプションの E サブオプションで決まります。

入力の場合は、データ・ストリーム中のデータ値は、実数の 10 進浮動小数点定 数または実数の 10 進固定小数点定数 (どちらの場合も符号はあってもなくてもよ い) でなければならず、それ以外のときは CONVERSION 条件が起こります。データ 値は、指定したフィールド内のどこにあってもかまいません。(便宜上、符号付き指数の前に付ける E は省略することができます。)

フィールド幅を表す値には、 先行ブランクや後書きブランク、 指数部の桁、正符号 (+) または負符号 (-) (オプション) の桁、 文字 E (オプション) の桁、および小数部の 小数点 (オプション) の桁を計算に入れます。

データ値は、指定されたフィールドの任意の場所に現れることができます。ブランクをフィールド内のデータ値の前後に置くことができ、それらは無視されます。 フィールド全体がブランクのときには、CONVERSION 条件が起こります。 小数点がない場合、*fractional-digits* は小数部の桁のうち仮想小数点の右側にある桁の桁数を示します。 小数点が数の中にある場合は、*fractional-digits* の指定は無効にされます。

*field-width* が 0 のときは、データ・リスト項目にはなにも割り当てられません。

次のステートメントは、次の 10 文字を A から取得し、それらを浮動小数点 10 進数として解釈します。 小数点は小数部の右端 6 桁の前にあると見なされます。 この数の値は変数 COST の属性に変換され、この変数に割り当てられます。

```
get file(A) edit (Cost) (E(10,6));
```

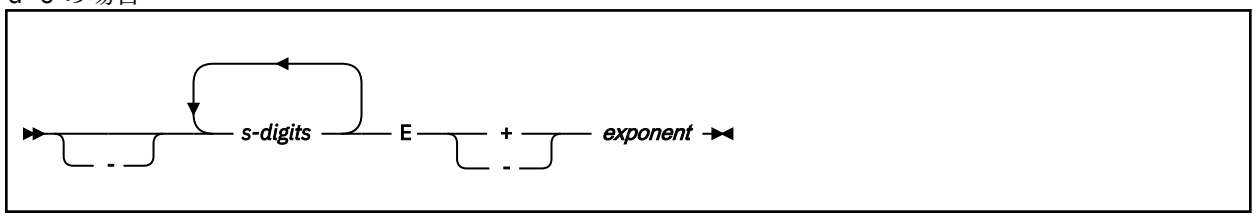
出力の場合は、データ・リスト項目が浮動小数点に変換され、必要に応じて四捨五入されます。 データの丸めは次のように行われます。 切り捨てによって右から数字が失われ、この数字が 5 以上である場合は、切り捨てられる数字の左の数字に 1 が加算されます。 この加算が行われると、指数が調整されることがあります。

出力ストリーム中に書き出された文字ストリングは、下記のいずれかの構文になります。

**注 :**

1. 文字ストリングのエレメントの間にブランクを入れることはできません。
2. 指数の長さは、浮動小数点データ・タイプおよび DEFAULT コンパイラー・オプションの E サブオプションの設定に応じて、2 桁または 4 桁になります。 以下の説明では、この長さを *e* で示しています。

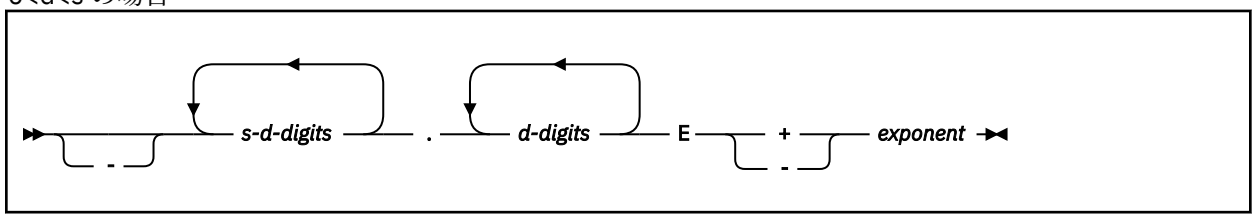
- d=0 の場合




w は、正の値の場合は >=s+e+2、負の値の場合は >=s+e+3 でなければなりません。

値がゼロ以外のときは、小数部の先頭行がゼロ以外の値になるように指数が調整されます。 値がゼロのときは、小数部のすべての桁 (最右端の桁は除く) でゼロ消去が行われます。

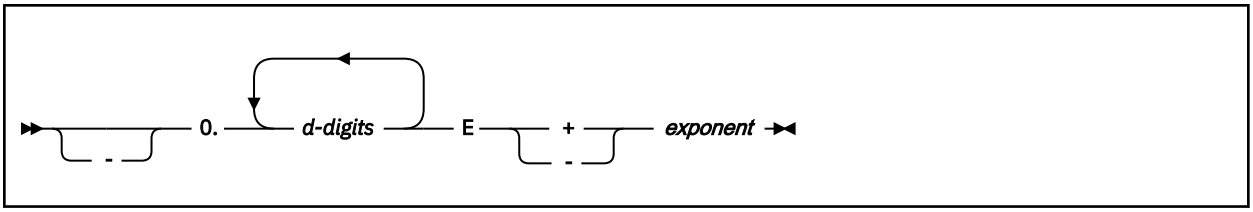
- 0<d<s の場合




w は、正の値の場合は >=s+e+3、負の値の場合は >=s+e+5 でなければなりません。

値がゼロ以外のときは、小数部の先頭行がゼロ以外の値になるように指数が調整されます。 値がゼロのときは、小数点の左側にあるすべての桁 (最初の桁は除く) でゼロ消去が行われます。 ほかのすべての桁にはゼロが入ります。

- d=s の場合



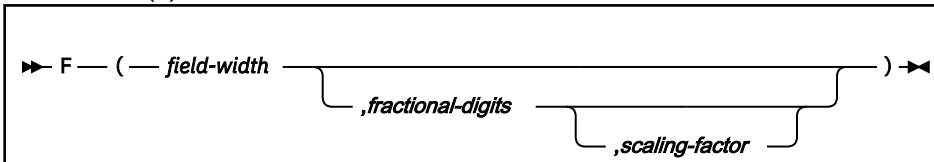

w は、正の値の場合は >=d+e+5、負の値の場合は >=d+e+6 でなければなりません。

値がゼロ以外の値のときは、小数部分の最初の桁がゼロ以外の値になるように指数が調整されます。 値がゼロの値のときは、どの桁にもゼロが入ります。

フィールド幅が短すぎて有効数字や符号が失われる場合は、SIZE 条件が起こります。 出力の場合、指定したフィールドが文字ストリングでいっぱいにならないときは、文字ストリングは右寄せされ、左側にブランクが埋め込まれます。

## F フォーマット項目

固定小数点 (F) フォーマット項目は、実数の固定小数点 10 進算術値の文字表記を 記述するものです。




***field-width***
フィールド内の桁の総数を表します。 この値は、フォーマット項目が使用されるたびに評価され、整数値 *w* に変換されます。 変換された値は、負数であってはなりません。

***fractional-digits***
小数点の右側にある小数部の桁数を表します。 この値は、フォーマット項目が使用されるたびに評価され、整数値 *d* に変換されます。 変換された値は、負数であってはなりません。 *fractional-digits* が省略された場合、この値は 0 と見なされます。

***scaling-factor***
小数部になければならない数字の桁数を表しています。 この値は、フォーマット項目が使用されるたびに評価され、整数値 *p* に変換されます。

入力の場合は、データ・ストリーム中のデータ値は、実数の 10 進固定小数点定 数 (符号はあってもなくてもよい) でなければならず、それ以外のときは、 CONVERSION 条件が起こります。データ値は、指定したフィールド内のどこにあ ってもかまいません。 フィールド内のデータ値の前後にブランクがあってもよく、それらのブランクは 無視されます。 フィールド全体がブランクのときは、ゼロと解釈されます。

*scaling-factor* を省略し、しかもフィールド内に小数点がない場合、 *fractional-digits* を表す式は、データ値の桁のうち仮想小数点の右 側にある桁の桁数を示します。 データ値の中に小数点がある場合は、*fractional-digits* を表す式は 無効にされます。

*scaling-factor* を指定すると、データ・ストリーム中のデータ値に、10 の *p* 乗 (p は位取り係数を表す整数値) が乗算されます。 したがって、*p* が正の数ならば、そのデータ値の小数点は、指定された位置から右へ *p* 桁分移動したところにあるものと見なされます。 また、*p* が負の数ならば、そのデータ値の小数点は、与えられた位置 から左へ *p* 桁分移動したところにあるものと見なされます。 小数点がつく位置は、実際に小数点があればその位置であり、小数点が示されていなければ *fractional-digits* を表す式によって指定された位置です。

*field-width* が 0 のときは、データ・リスト項目にはなにも割り当てられません。

出力の場合は、データ・リスト項目が必要に応じて固定小数点に変換されます。 浮動小数点データは、FIXED DECIMAL (N,q) に変換されます。 ここで *q* は、*fractional-digits* に指定した値です。 ストリーム中のデータ値は、必要に応じて四捨五入された文字表記の実数の 10 進固定小数点であり、指定したフィールドに右寄せで入れられます。

10 進固定小数点タイプから文字タイプへの変換は、通常の変換規則に従って 行われます。 余分な桁は、変換後のストリングの数値の前にブランクとして示されます。 また先行ゼロはブランクに変換されるので (ただし小数点のすぐ左のゼロは除く)、数値の前に余分のブランクが現れることがあります。 小数点または負符号 (-) が書き出されるときは、どちらも先行ブランクの 1 つと置き換わります。

*field-width* だけを指定したときは、数の整数部分だけが書き出されます。小数点は示されません。

*field-width* と *fractional-digits* を指定したときは、 数の整数部分と小数部分の両方が書き出されます。 *fractional-digits* の値 (*d*) がゼロより大きければ、右端 *d* 桁の前に小数点が挿入されます。 *fractional-digits* が *d* より小さければ、後続のゼロが付けられます (値 *d* は *field-width* より小さくなければなりません)。 項目の絶対値が 1 より小さいときは、小数点の前に 0 が 1 つ付けられます。 小数点の左側にあるすべての桁 (すぐ左の 1 桁は除く) では、先行ゼロのゼロ消去が行われます。

データの丸めは次のようにして行われます。 切り捨てによって、右の切り捨てられる数字が 5 以上の場合は、 切り捨てられる数字の左の数字に 1 が追加されます。

出力において、データ・リスト項目が 0 より小さい場合は、 負符号 (-) が文字表示の前に付けられます。0 より大きいか等しい場合は、符号は付けられません。 したがって、負の値の場合には、符号、小数点、および小数点のすぐ左の 0 を桁数 として *field-width* に含めることを考慮する必要があります。

*field-width* が短すぎて文字が失われる場合は、SIZE 条件が起こります。

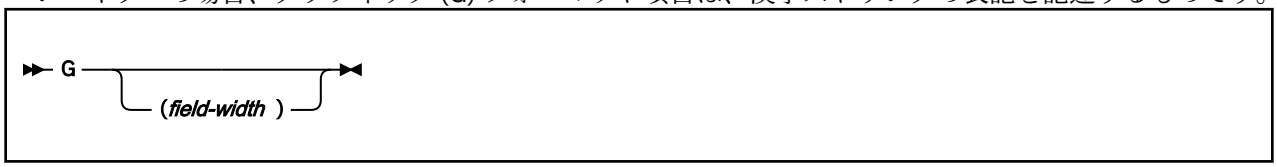
次の例を考えてください。

```
declare Total fixed(4,2);
put edit (Total) (F(6,2));
```

この PUT ステートメントでは、Total の値を文字表記の固定小数点数に変換し、 出力ファイル SYSPRINT に書き出すことを指定しています。 最後の 2 つの数字の前に小数点が挿入され、この数は 6 文字フィールドに 右寄せで入れられます。 先行ゼロ (小数点のすぐ左にあるゼロは除く) はブランクに変えられ、必要に応じて 負符号 (-) が最初の数字の左に付きます。

## G フォーマット項目

コンパイラーの場合、グラフィック (G) フォーマット項目は、漢字ストリングの表記を記述するものです。




***field-width***
データ・ストリーム中で、漢字ストリングが入っている (または入れら れる) グラフィック桁 (2 バイト) の個数を指定します。 この式は、このフォーマット項目が使用されるたびに評価され、整数値 (負であってはならない) に変換されます。 1 つのグラフィックを形成する 2 バイトの中間で行の終わりになってはなりません。

入力の場合は、指定した桁数のグラフィックスがデータ・ストリームから取り出され、必要な切り捨てまたは埋め込みを行ってデータ・リスト項目に割り当てられます。 入力の場合、*field-width* を必ず指定しなければなりません。 field-width がゼロのときは、ストリングはヌル・ストリングになります。

出力の場合は、 データ・リスト項目が *field-width* に合わせて 右側で切り捨てられたか、(埋め込み用グラフィックで) 拡張されたあと、 データ・ストリーム中に書き出されます。 ストリングを囲むための引用符が挿入されることはなく、識別用接尾部 *G* が挿入されることもありません。 *field-width* がゼロのときは、データ・ストリーム中にはなにも書き出されません。 *field-width* を指定しない場合は、デフォルト値が使用されます。 これは、データ・リスト項目のグラフィック・ストリング長に等しい値です。

次の例では、ファイル OUT に GRAPHIC オプションが指定してあると、 6 バイトが伝送されます。

```
declare A graphic(3);
put file(Out) edit (A) (G(3));
```

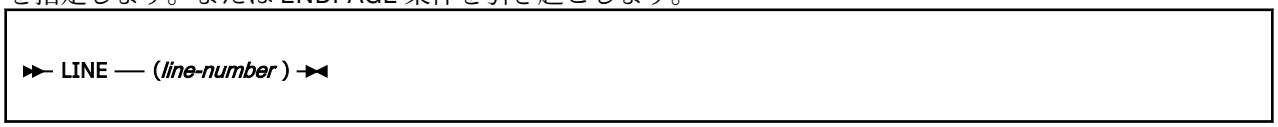
# L フォーマット項目

入力の場合は、L は、行の終わりまでのすべてのデータがデータ項目に割り当てられるものであることを表します。

出力の場合は、L は、データ項目 (必要に応じて、右側をブランクで埋め込まれている) が出力行の残りを埋めるものであることを表します。

# LINE フォーマット項目

LINE フォーマット項目は、PRINT ファイルの現行ページのどの行に次のデータ・リスト 項目を印刷するかを指定します。または ENDPAGE 条件を引き起こします。




***line-number***
式で表すことができます。この式は、このフォーマット項目が使用されるたびに評価され、整数値 (負であってはならない) に変換されます。

必要に応じて、ブランク行が挿入されます。

指定した *line-number* (行番号) が現在行の行番号以下である場合や、指定した行が OPEN ステートメントの PAGESIZE オプションでセットされた限界 (またはデフォルトの限界) を超えている場合は、ENDPAGE 条件が起こります。 ただし、指定した *line-number* が現在行の行番号に等しく、 しかも 1 桁目の文字がまだ伝送されていない場合は例外であり、そのときは、 SKIP(0) 項目を指定した場合と同じ結果になります (つまり、行送りせずに復帰し ます)。

*line-number* がゼロのときは、1 と見なされます。

# P フォーマット項目

ピクチャー (P) フォーマット項目は、実数の数字の値や文字の値の文字表記を記述するものです。

入力の場合は、P フォーマット項目のピクチャー指定によって、データ・ストリーム中に あるデータ項目のフォーマットを記述したり、数字指定の場合は、項目の算術値がどの ように解釈されるかを記述したりします。 ここに指定した文字がストリーム中にないときは、CONVERSION 条件が起こります。

出力の場合は、データ・リスト内の対応するエレメントの値が、ピクチャー指定で 指定したフォーマットに変換されてから、データ・ストリーム中に書き出されます。

***picture-specification***
325 ページの『第 14 章 ピクチャー指定文字』を参照してください。

次の例を考えてください。

```
get edit (Name, Total) (P'AAAAA',P'9999');
```

このステートメントが実行されるとき、ファイル SYSIN が入力ファイルで あると見なされます。 SYSIN から入力された最初の 5 文字は、英字またはブランクでなければならず、 それらは Name に割り当てられます。 次の 4 文字は数字でなければならず、それらは Total に割り当てられます。

# PAGE フォーマット項目

PAGE フォーマット項目は、新しいページを設定することを指定します。 これは、PRINT ファイルの場合にのみ使用できます。

新しいページを開始すると、ファイルは次のページの 1 行目に位置づけられます。

# R フォーマット項目

リモート (R) フォーマット項目は、FORMAT ステートメント内の フォーマット・リストを使用することを指定します。

***label-reference***
FORMAT ステートメントのラベル定数

R フォーマット項目とそこで指定した FORMAT ステートメントは、同一ブロックの内部にな ければならず、両方ともそのブロックの同一の呼び出し内にある必要があります。

リモート FORMAT ステートメントには、それ自体をラベル参照として参照する R フォーマット項目が含まれていてはなりません。 また、結果的に元の FORMAT ステートメントを参照することになる別のリモート FORMAT ステートメントを参照することもできません。

GET ステートメントまたは PUT ステートメントに対して割り込み可能な条件は、参照される リモート FORMAT ステートメントに対しても割り込み可能になっていなければなりません。

ON ユニットが 1 つの GET ステートメントまたは PUT ステートメントだけで構成されているときは、そのステートメントが 1 つのブロックになるので、そのステートメント内にリモート・フォーマット項目を 書くことはできません。

**例**

```
declare Switch label;
get file(In) list(Code);
if Code = 1 then
  Switch = L1;
else
  Switch = L2;
get file(In) edit (W,X,Y,Z)
    (R(Switch));
L1:  format (4 F(8,3));
L2:  format (4 E(12,6));
```

Switch は、ラベル変数として宣言されています。 2 番目の GET ステートメントは、2 つの FORMAT ステートメントのどちらでも使うことができます。

**関連情報**

309 ページの『FORMAT ステートメント』
FORMAT ステートメントは、伝送されるデータのフォーマットを制御するために、 編集ディレクティブ・データ伝送ステートメントで使用することのできるフォーマット・リストを指定する ものです。

# SKIP フォーマット項目

SKIP フォーマット項目は、新しい行を現在行として定義することを指定します。

***relative-line***
式を指定します。この式は、このフォーマット項目が使用されるたびに評価され、整数値 *n* に変換されます。 変換された値は、負の数であってはならず、また、32768 未満でなければなりません。 PRINT ファイル以外のファイルのときは、この値はゼロより大きくなければ なりません。 この値がゼロであるか、または省略されていると、1 と見なされます。

新しい行は、現在行から *n* 行だけあとにある行になります。

*n* が 1 より大きい場合は、1 行または複数の行が入力時に無視されます。出力時には、1 行または複数のブランク行が挿入されます。

値 *n* は、PRINT ファイルに限り、ゼロであることがあります。この場合には、位置付けは、現在行の先頭になります。 前に印刷された文字に上重ね印刷されることがあります。

PRINT ファイルの場合、指定した *relative-line* が OPEN ステートメントの PAGESIZE オプションでセットされた限界 (またはデフォルトの限界) を超えていると、ENDPAGE 条件が起こります。

SKIP フォーマット項目が、ファイルのオープン後に最初に実行される項目である 場合は、最初のページの *n* 行目から出力が開始されます。 *n* がゼロまたは 1 である場合、最初のページの最初の行から出力が開始されます。

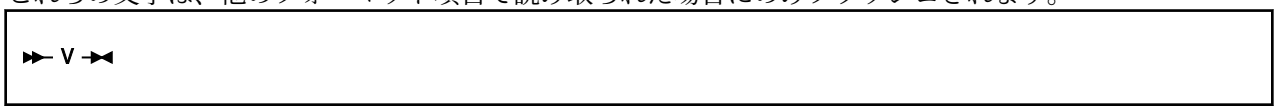
**例**

```
get file(In) edit(Man,Overtime)
     (skip(1), A(6), COL(60), F(4,2));
```

このステートメントは、ファイルに関連付けられたデータ・セットを新しい行に位置付けます。 その行の最初の 6 文字が Man に割り当てられ、60 桁目から 始まる 4 文字が Overtime に割り当てられます。

## V フォーマット項目

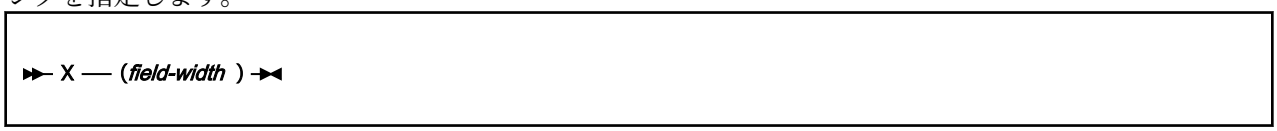
入力の場合は、V は、行の終わりまでのすべてのデータがデータ項目に割り当てられるものであることを表します。 ただし、V フォーマット項目で読み取られた文字は、フラッシュされず、表示されるだけです。これらの文字は、他のフォーマット項目で読み取られた場合にのみフラッシュされます。




V フォーマット項目は、出力では無効です。

## X フォーマット項目

スペーシング (または X) フォーマット項目は、データ・ストリーム内の 2G 未満のデータ値の相対スペーシングを指定します。




***field-width***
式を指定します。この式は、このフォーマット項目が使用されるたびに評価され、整数値 (負であってはならない) に変換されます。 この整数値は、データ・ストリーム中の現在位置から次のフィールドまでの 間にある桁の桁数を表します。

入力の場合は、指定した桁数だけデータ・ストリーム中で水平送りされ、 それらの桁はプログラムに伝送されません。

次の例を考えてください。

```
get edit (Number, Rebate)
       (A(5), X(5), A(5));
```

入力ファイル SYSIN からの次の 15 文字は次のように 処理されます。 最初の 5 文字は Number に割り当てられ、次の 5 文字は無視され、 残りの 5 文字は Rebate に割り当てられます。

出力の場合は、指定した個数のブランク文字がストリーム中に挿入されます。

次の例を考えてください。

```
put file(Out) edit (Part, Count) (A(4), X(2), F(5));
```

Part の値を表す 4 文字、2 つのブランク文字、Count の固 定小数点値を表す最後の 5 文字は、Out という名前のファイルに書き 出されます。

# 第 14 章 ピクチャー指定文字

ピクチャー指定とは、一連のピクチャー文字を一重または二重引用符で囲んだものです。 この文字は、文字データ項目または数字データ項目の各位置の内容、および出力の内容を記述します。

この指定は、次の 2 つの方法で行われます。

- 宣言内の PICTURE 属性の一部として
- 編集ディレクティブ入力および編集ディレクティブ出力の 321 ページの『P フォーマット項目』の一部として

ピクチャー指定は、文字データ項目または数字データ項目を記述します。 A または X のピクチャー文字は、文字ピクチャー指定としてピクチャー指定を定義します。それ以外の場合は、数字ピクチャー指定です。

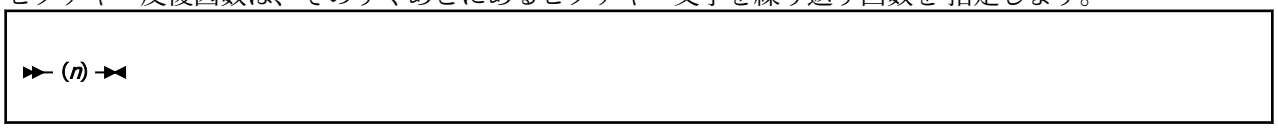
文字ピクチャー項目 は、英字、10 進数、ブランク、通貨記号、および句読点文字で構成できます。

数字ピクチャー項目 は、10 進数、オプションの小数点、オプションの文字 E、およびオプションの 1 つまたは 2 つの正符号 (+) と負符号 (-) のみで構成できます。算術データには普通に対応づけられる その他の文字 (通貨記号など) も指定できますが、そのような文字は数字変数の算術値の一部ではありません。 ただし、そのような文字は、数字と一緒にストレージに保管され、 その変数の文字値の一部となります。

値がピクチャー変数に割り当てられるときや、P フォーマット項目によって印刷されると きに、ピクチャー指定が異なれば値の表記がどのように異なるかを、この章に記 載されている図で示します。 それらの図には、データの元の値、そのデータが割り当てられる (または書き出される) 際に入っていた変数の属性、ピクチャー指定、および数字変数またはピクチャー文字変数の文字値が示されています。

## ピクチャー反復因数

ピクチャー反復因数は、そのすぐあとにあるピクチャー文字を繰り返す回数を 指定します。




**n**
整数です。 括弧内にブランクがあってはなりません。 *n* が 0 のときは、ピクチャー文字は無視されます。

例えば、次の 2 つのピクチャー指定は、同じことを記述しています。

```
'999V99'
'(3)9V(2)9'
```

## 文字データ用のピクチャー文字

文字ピクチャー指定は、無変更の文字データ項目を記述します。 これによって、使用可能な文字セット全体の特定のサブセットに属する文字し かデータ項目のその文字位置に入れられないことを指定できます。 データは、英字、10 進数、およびブランクから形成できます。

文字ピクチャー指定で有効な文字は、X、A、および 9 のみです。 これらはそれぞれ、文字値の中に 1 つの文字位置が存在することを指定します。

**X**
1 バイトをなす 8 ビットで表されており、ビットを組み合わせてできる 256 文字のうちの任意の文字。

**A**
任意の英字、または特別言語 (#、@、$) 文字、 またはブランク。

**9**
任意の数字またはブランク ( 9 ピクチャー指定文字は、文字データの場合にのみブランクを使用することができます)。

© Copyright IBM Corp. 1999, 2022

文字値がピクチャー文字データ項目に割り当てられる (または伝送される) とき には、1 文字ずつ、対応するピクチャー指定文字で指定されたとおりであるかが 検査されます。 文字データがその位置の指定と一致しない場合、先行ゼロの除外が、 CONVERSION 条件が起こります。 (ただし、レコード単位の伝送を行うか、別名を使用するかして値を変更した場合は、 検査は行われません。) 次の例を考えてください。

```
declare Part# picture 'AAA99X';
put edit (Part#) (P'AAA99X');
```

次の値は、Part# の値として有効です。

```
'ABC12M'
'bbb09/'
'XYZb13'
```

次の値は、Part# の値としては有効ではありません (下線で示した文字が無効文字です)。

```
'AB123M'
'ABC1/2'
'Mb#A5;'
```

326 ページの表 49 に、文字ピクチャー指定の例を示します。

表 *49.* 文字ピクチャー指定の例

| ソース属性 | ソース・データ (定数フォーマット) | ピクチャー指定 | 文字値 |
|---|---|---|---|
| CHARACTER(5)<br>CHARACTER(5)<br>CHARACTER(5) | '9B/2L'<br>'9B/2L'<br>'9B/2L' | XXXXX<br>XXX<br>XXXXXXX | 9B/2L<br>9B/<br>9B/2Lbb |
| CHARACTER(5)<br>CHARACTER(5)<br>CHARACTER(5) | 'ABCDE'<br>'ABCDE'<br>'ABCDE' | AAAAA<br>AAAAAA<br>AAA | ABCDE<br>ABCDEb<br>ABC |
| CHARACTER(5)<br>CHARACTER(5) | '12/34'<br>'L26.7' | 99X99<br>A99X9 | 12/34<br>L26.7 |

## 数字データ用のピクチャー文字

数字データは数値を表します。 ピクチャー指定には、文字データ・ピクチャー文字 X または A を含めることはできません。 数字データのピクチャー文字は、データの編集を指定することもできます。

数字変数は、その使用方法に応じて、2 種類の値を持つと考えられます。 変数のタイプは次のとおりです。

**算術**
算術値は、データ項目の 10 進数、仮想小数点の位置、符号 (省略可能)、お よび指数 (符号は省略可能) か位取り係数で表される値です。 数字変数の算術値が使用されるのは、次の場合です。

- 数字変数が、コード化算術値またはビット値を求める式の中で使われている とき (これには、¬、&、|、および比較演算子を用いる式も含まれます。 文字ストリングを持つ式でも数字変数の算術値を使用します。)
- 数字変数がコード化算術変数、数字変数、またはビット変数に 割り当てられるとき
- 編集ディレクティブの入出力において、C、E、F、B、および P (数字) フォーマット項目とともに使用されるとき

数字変数の算術値が内部コード化算術表記に変換されます。

**文字値**
文字値は、データ項目の 10 進数と、ピクチャー指定内に書かれている すべての編集文字と挿入文字とで表される値です。 ただし、文字値には、ピクチャー文字 V、K、または F で指定された小数点の想定位置は含まれません。 数字変数の文字値が使用されます。

- 数字変数が文字式の中で使用されているとき
- 文字変数に割り当てられるとき
- リスト・ディレクティブ出力またはデータ・ディレクティブ出力によって、そのデータが印刷されるとき
- 数字変数で定義された、または数字変数にもとづいた文字変数が参照されたとき
- 編集ディレクティブ出力で A または P (文字) フォーマット項目によって、数字変数が印刷されるとき

データ変換は不要です。

数字データは、10 進数、小数点 (省略可能)、文字 E (省略可能)、および 1 つまた は 2 つの正符号 (+) または負符号 (-) だけから形成することができます。算術データには普通に関連づけられるその他の文字 (通貨記号など) も指定できますが、そのような文字は数字変数の算術値の一部ではありません。ただし、そのような文字は、数字と一緒にストレージに保管され、その変数の文字値の一部となります。

数字指定は、1 つまたは複数のフィールドから構成され、各フィールドが固定小数点を記述します。浮動小数点指定は、2 つのフィールド (小数部を記述する フィールドと指数を記述するフィールド) からなります。第 1 フィールドは、V ピクチャー指定文字を挿入することによって、サブフィールドに分けることができます。V の前のデータ (ある場合) と V のあとのデータ (ある場合) が 数字指定のサブフィールドです。

数字データのピクチャー指定では、各フィールドに、数字桁を指定する ピクチャー文字が最低 1 つはなければなりません。 ただし、このピクチャー文字は数字 9 である必要はありません。 ゼロ抑制文字 (Z または *) などの他のピクチャー文字も、数字の位置を指定します。

**注 :** K、V、F 以外のすべての文字は、文字表記の中に文字が存在することを表します。

**関連情報**
330 ページの『挿入文字と小数点文字』
小数点、コンマ、スラッシュ、またはアポストロフィを V と併用すると、固定小数点 (または浮動小数点) の整数部分の終わりと小数部分の始まりを区切る位置に、小数点、コンマ、スラッシュ、または挿入することができます。 これは印刷時に必要になることがあります。 V 自体ではピリオドその他の区切り文字を印刷することはできません。

325 ページの『ピクチャー反復因数』
ピクチャー反復因数は、そのすぐあとにあるピクチャー文字を繰り返す回数を 指定します。

## 数字と小数点

固定小数点 10 進数値を表す数字指定では、ピクチャー文字 9 と V を使用します。

**9**
関連データ項目内のこの位置に 10 進数が入ることを指定します。 (数字データのピクチャー指定文字 9 は、文字データの場合は対応する文字がブ ランクであってはならないという点で文字データの指定とは異なります。)

n 個のピクチャー文字 9 からなるストリングは、その項目が長さ n の文字ストリング (変更 不能) で、各文字は数字 (0 から 9) であることを表します。 次の例を参照してください。

```
dcl digit picture'9',
    Count picture'999',
    XYZ picture '(10)9';
```

使用例を以下に示します。

```
dcl 1 Record,
      2 Data char(72),
      2 Identification char(3),
      2 Sequence pic'99999';
dcl Count fixed dec(5);
⋮
```

```
Count=Count+1;
Sequence=Count;
write file(Output) from(Record);
```

**V**

関連データ項目内のこの位置に小数点があると見なすことを指定します。 ただし、実際の小数点または 10 進コンマをこの位置に挿入することを指定するわけではありません。 割り当てられる値の整数部分と小数部分は、位取り係数 F(±x) が指定されている場合はそれによって修正されてから、文字 V に位置合わせされます。 したがって、割り当てられる値は左端と右端のどちらかで切り捨てられたり、数字ゼロで拡張されたりすることがあります。 (左端で有効数字が切り捨てられたときは、結果は未定義となり、可能であれば SIZE 条件が起こります。)

固定小数点 10 進数値のピクチャー指定 (または浮動小数点 10 進数値のピクチャー指定の第 1 フィールド) 内に文字 V がないときは、フィールド指定の最右端に V があると見なされます。 したがって、割り当てられる値は必要に応じて切り捨てられ、整数に なることがあります。

文字 V は、1 つのピクチャー指定に 1 回しか記入できません。

次の例を考えてください。

```
dcl Value picture 'Z9V999';
Value = 12.345;
dcl Cvalue char(5);
Cvalue = Value;
```

Cvalue には、Value の値が 割り当てられた後、'12345' が入っています。

328 ページの表 50 に、数字と小数点文字の例を示します。

表 *50.* 数字と小数点文字の例

| ソース属性 | ソース・データ (定数フォーマット) | ピクチャー指定 | 文字値 |
|---|---|---|---|
| FIXED(5) | 12345 | 99999 | 12345 |
| FIXED(5) | 12345 | 99999V | 12345 |
| FIXED(5) | 12345 | 999V99 | 未定義 (undefined) |
| FIXED(5) | 12345 | V99999 | 未定義 (undefined) |
| FIXED(7) | 1234567 | 99999 | 未定義 (undefined) |
| FIXED(3) | 123 | 99999 | 00123 |
| FIXED(5,2) | 123.45 | 999V99 | 12345 |
| FIXED(7,2) | 12345.67 | 9V9 | 未定義 (undefined) |
| FIXED(5,2) | 123.45 | 99999 | 00123 |

**注 :** 文字値が未定義のときは、SIZE 条件が起こります。

## ゼロ消去

ピクチャー文字 Z とアスタリスク (*) は、文字値の条件付き数字桁を指定します。これを 使用すれば、先行ゼロをアスタリスクまたはブランクで置き換えることができます。

先行ゼロとは、固定小数点数の左端の桁位置、または浮動小数点数の 2 つの部分の左端の桁位置にあるもので、小数点の想定された位置の左側にあり、前に 1 から 9 までの数字がないものです。 数値の左端のゼロ以外の数字と、その右側のすべての数字 (ゼロまたはゼロ以外) は、有効数字を表します。

**Z**

条件付き数字桁を指定するもので、データのこの位置に先行ゼロがあれば、それはブランクで置き換えられます。 先行ゼロ以外の数字はそのまま変りません。 ピクチャー文字 Z は、ピクチャー文字 * または浮動文字が記入されているのと同 じフィールド内に記入することはできません。また、フィールド内で任意のピクチャー文 字の右側に記入することもできません。

*****
条件付き数字桁を指定します。 先行ゼロがアスタリスクで置き換えられることを除き、使用方法はピクチャー文字 Z と同じです。 ピクチャー文字 * は、ピクチャー文字 Z や浮動文字が記入されているのと同じフィールド内に記入することはできません。また、フィールド内で任意のピクチャー文字の 右側に記入することもできません。

329 ページの表 51 に、ゼロ抑制文字の例を示します。

表 *51.* ゼロ抑制文字の例

| ソース属性 | ソース・データ (定数フォーマット) | ピクチャー指定 | 文字値 |
|---|---|---|---|
| FIXED(5) | 12345 | ZZZ99 | 12345 |
| FIXED(5) | 00100 | ZZZ99 | bb100 |
| FIXED(5) | 00100 | zzzzz | bb100 |
| FIXED(5) | 00000 | ZZZZZ | bbbbb |
| FIXED(5,2) | 123.45 | ZZZ99 | bb123 |
| FIXED(5,2) | 001.23 | ZZZV99 | bb123 |
| FIXED(5) | 12345 | ZZZV99 | 未定義 (undefined) |
| FIXED(5,2) | 000.08 | ZZZVZZ (ZZVZZ) | bbb08 |
| FIXED(5,2) | 000.00 | ZZZVZZ (ZZVZZ) | bbbbb |
| FIXED(5) | 00100 | ***** | **100 |
| FIXED(5) | 00000 | ***** | ***** |
| FIXED(5,2) | 000.01 | ***V * * | ***01 |
| FIXED(5,2) | 95 | $**9.99 | $**0.95 |
| FIXED(5,2) | 12350 | $**9.99 | $123.50 |

**注 :** 文字値が未定義のときは、SIZE 条件が起こります。

ピクチャー文字 Z またはアスタリスクをピクチャー文字 V の右側に記入するときは、その指定 の小数部分の全数字桁と整数部分の全数字桁に、それぞれ Z またはアスタリスク・ピクチャー 文字を記入しなければなりません。 ピクチャー文字 V の右側にある全数字桁がゼロ消去ピクチャー文字であるとき、値の小数部分のゼロが抑制されるのは、小数部分のすべての桁にゼロが入っており、しかも整数部分のすべての桁が抑制されたときだけです。 したがって、そのデータ項目の文字値はブランクまたはアスタリスクだけになります。 小数部分に 1 つでも有効数字が含まれているときは、小数部分の数字はブランクやアスタリスクでは置き換えられません。

## 挿入文字

ピクチャー文字のコンマ (,)、小数点 (.)、スラッシュ (/)、アポストロフィ (')、およびブランク (B) を指定すると、数字データの関連する位置にその文字が挿入されます。 これらの文字は、数字桁や文字桁を示すものではなく、数字や文字の間に 挿入されます。 ただし、文字値の中では、いずれの文字も、その文字が消去されるか否かに関係なく 1 つの文字桁を表しています。

コンマ、小数点、スラッシュ、アポストロフィは、条件付き挿入文字であり、一連のゼロ抑制文字の間に入っているときには消去されることもあります。 ブランクは無条件挿入文字であり、 対応する位置にブランクを常に挿入することを指定します。

挿入文字は、文字値にだけ適用され、 そのデータ項目の算術値に関してはなにも指定しません。 挿入文字を使用しても、固定小数点 10 進数のピクチャー指定内で小数点 (または 10 進コンマ) 位置合わせを指定したことにはなりませんし、挿入文字はデータ項目の算術値の一部ではありません。 小数点位置合わせは、ピクチャー文字 V および F によって制御されます。

**コンマ (,)、小数点 (.)、スラッシュ (/)、アポストロフィ (')**
ゼロ消去が行われない限り、数字データ内の対応する位置にこれらの 文字が挿入されます。 ゼロ消去が行われたときは、下記の場合にのみこれらの文字が挿入されます。

- 挿入位置より左側に、抑制されていない数字がある場合。
- 挿入位置のすぐ左に V があり、データ項目の小数部分に有効数字がある場合。
- ピクチャー指定の先頭が挿入位置である場合。
- 挿入位置より前にあるのが、数字桁を指定しない文字だけである場合。

上記以外の場合にゼロ消去が行われるときは、コンマ、小数点、スラッシュ、アポストロフィの挿入文字は、その前にある文字と等しくゼロ抑制文字と見なされます。

**B**
数字データの文字値の対応する位置に、ブランク文字を挿入することを指定します。

## 挿入文字と小数点文字

小数点、コンマ、スラッシュ、またはアポストロフィを V と併用すると、固定小数点 (または浮動小数点) の整数部分の終わりと小数部分の始まりを区切る位置に、小数点、コンマ、スラッシュ、または挿入することができます。 これは印刷時に必要になることがあります。 V 自体ではピリオドその他の区切り文字を印刷することはできません。

小数点は V の直前または直後になければなりません。 小数点が V の前にある場合は、すべての小数桁が有意である場合でも、消去されていない数字が V の左側にある場合にのみ挿入されます。 V のすぐあとにある小数点は、V の右側にあるすべての数字が消去されたときに消去されますが、小数桁に消去されていない数字がある (間にゼロがあってもよい) ときは挿入されます。

次に各国で使用されている 10 進数の規則の例を示します。

```
declare A picture 'Z,ZZZ,ZZZV.99',
        B picture 'Z.ZZZ.ZZZV,99',
        C picture 'ZBZZZBZZZV,99',
        D picture 'Z'ZZZ'ZZZV.99';
A,B,C,D = 1234;
A,B,C,D = 1234.00;
```

A、B、C、D はそれぞれ 9 桁の数を表し、7 番目と 8 番目の数字の間に小数点または小数コンマが想定されています。 小数点挿入文字によって指定された実際の小数点は、算術値の一部ではありませんが、 文字値ではその一部です。 2 つの代入ステートメントは、次のように A、B、C、D に同じ文字値を割り当てます。

```
1,234.00     /* value of A */
1.234,00     /* value of B */
1 234,00     /* value of C */
1'234.00     /* value of D */
```

次の例では、割り当て時に文字 V で小数点の位置合わせが行われます。 Rate が印刷される場合、'762.00' と表示されますが、その算術値は 7.6200 です。

```
declare Rate picture '9V99.99';
Rate = 7.62;
```

330 ページの表 52 に、挿入文字の例を示します。

表 *52.* 挿入文字の例

| ソース属性 | ソース・データ (定数フォーマット) | ピクチャー指定 | 文字値 |
|---|---|---|---|
| FIXED(4) | 1 2 3 4 | 9,999 | 1,234 |
| FIXED(6,2) | 1234.56 | 9'999V.99 | 1'234.56 |
| FIXED(4,2) | 12.34 | ZZ.VZZ | 12.34 |

表 52. 挿入文字の例 (続き)

| ソース属性 | ソース・データ (定数フォーマット) | ピクチャー指定 | 文字値 |
|---|---|---|---|
| FIXED(4,2) | 00.03 | ZZ.VZZ | bbb03 |
| FIXED(4,2) | 00.03 | ZZV.ZZ | bb.03 |
| FIXED(4,2) | 12.34 | ZZV.ZZ | 12.34 |
| FIXED(4,2) | 00.00 | ZZV.ZZ | bbbbb |
| FIXED(9,2) | 1234567.89 | 9'999'999.V89 | 1'234'567.89 |
| FIXED(7,2) | 12345.67 | **,999V.99 | 12,345.67 |
| FIXED(7,2) | 00123.45 | **,999V.99 | ***123.45 |
| FIXED(9,2) | 1234567.89 | 9.999.999V,99 | 1.234.567,89 |
| FIXED(6) | 123456 | 99/99/99 | 12/34/56 |
| FIXED(6) | 123456 | 99.9/99.9 | 12.3/45.6 |
| FIXED(6) | 001234 | ZZ/ZZ/ZZ | bbb12/34 |
| FIXED(6) | 000012 | ZZ/ZZ/ZZ | bbbbbb12 |
| FIXED(6) | 000000 | ZZ/ZZ/ZZ | bbbbbbbb |
| FIXED(6) | 000000 | **/**/** | ******** |
| FIXED(6) | 000000 | **B**B** | **b**b** |
| FIXED(6) | 123456 | 99B99B99 | 12b34b56 |
| FIXED(3) | 123 | 9BB9BB9 | 1bb2bb3 |
| FIXED(2) | 12 | 9BB/9BB | 1bb/2bb |

## 通貨記号の定義

通貨記号は、数字データの文字値を表示するピクチャー文字として使用することができます。 この記号には、ドル符号 ($) またはユーザーが選択した任意の記号を使用できます。 記号は、< および > 文字で囲まれた任意の文字シーケンスにすることができます。 このトピックでは、ユーザー独自の文字を通貨記号として定義する方法について説明します。

**<**
通貨記号の開始を表します。 この文字は、エスケープ文字としての役割を果たします。 < を文字として使用したい場合は、<< と指定しなければなりません。

**char**
これは、通貨記号 (複数の場合もある) の一部となる任意の文字です。

**>**
通貨記号の終わりを示します。 文字 > を使用する場合は、<> を指定する必要があります。

複数の > は浮動ストリングを示します (333 ページの『浮動的使用』 を参照)。

以下に、一般的な挿入ストリングの例を示します。

**<DM>**
ドイツ・マルクを表します。

**<Fr>**
フランス・フランを表します。

**<K$>**
カリスタン・ドルを表します。

**<Sur.f>**
スリナム・ギルダーを表します。

**<$>**
ドル符号を表します。

文字 < または > をシーケンスに含める必要がある場合は、その前に別の < を付ける必要があります。したがって、< はエスケープ文字としても機能します。

<> で囲まれたシーケンス全体は、1 つの「シンボル」を表し、したがって、1 つの数字の文字値を表します。 シンボルを浮動ピクチャー文字として表す必要がある場合は、「<>」の後に > を指定して、各オカレンスを表します。

以下の例を参照してください：

**Pic '<DM>>>.>>9,V99'**
10 文字の数字ピクチャーを表し、割り当てのあとは 11 文字になります。

**Pic '<Sur.f>999,V99'**
7 文字の数字ピクチャーを表し、割り当てのあとは 11 文字になります。

**Pic '<K$>>>,>>9.V99'**
10 文字の数字ピクチャーを表し、割り当てのあとは 11 文字になります。

**Pic '<$>>>,>>9.V99'**
10 文字の数字ピクチャーを表し、割り当てのあとは 10 文字になります。

**Pic '$$$,$$9.V99'**
上のピクチャー指定と同じ値になります。

これ以外の通貨記号の定義例を以下に示します。

```
dcl P pic'<DM>9.999,V99';
P = 1234.40;                          /* Yields 'DM1.234,40'   */


dcl P pic'<DM>9.999,V99';
P =   34.40;                          /* Yields 'DM   34,40'   */


dcl P pic'<DM>>.>>9,V99';
P = 1234.40;                          /* Yields 'DM1.234,40'   */


dcl P pic'<DM>>.>>9,V99';
P =   34.40;                          /* Yields '   DM34,40'   */


dcl P pic'9.999,V99<K$>';
P = 1234.40;                          /* Yields '1.234,40K$' */
```

このセクションでは、用語通貨記号 と $ 記号は、ドル記号または任意のユーザー定義通貨記号を指します。

## 符号および通貨記号の使用

ピクチャー文字 S、+、および - は、数字データ内の符号を指定します。 ピクチャー文字 $ (通貨記号) は、数字データの文字値の中の通貨記号を指定します。 各フィールドでは、1 種類の符号文字しか使用できません。

**通貨記号**
通貨記号を指定します。

次の例を考えてください。

```
dcl Price picture '$99V.99';
Price = 12.45;
```

Price の文字値は、'$12.45' です。 Price の算術値は 12.45 です。

文字を通貨記号として指定する場合の詳細については、331 ページの『通貨記号の定義』を 参照してください。

**S**
データ値が >= 0 の場合は、正符号文字 (+) を指定します。それ以外の場合は、負符号文字 (-) を指定します。これらの規則は、通貨記号の規則と同じです。

次の例を考えてください。

```
dcl Root picture 'S999';
```

値 50 は '+050' として、 値 0 は '+000' として、 値 -243 は '-243' としてそれぞれ保持されます。

**+**
データ値が >= 0 の場合は、正符号文字 (+) を指定します。それ以外の場合は、ブランクを指定します。使用規則は、通貨記号の場合と同じです。

**-**
データ値が <0 の場合は負符号 (-) を指定し、 それ以外の場合はブランクを指定します。 使用規則は、通貨記号の場合と同じです。

符号および通貨記号は、固定的に使用することも浮動的に使用することもできます。

## 固定的使用

固定的に使用すると、符号、通貨記号、またはブランクが対応する位置に入ります。

固定文字として使用される S、+、または - は、浮動小数点指定の小数部フィールド内や指数フィールド内の全数字桁の右側または左側に記入することができ、固定小数点の全数字桁の右側または左側に記入することができます。

## 浮動的使用

浮動的に使用すると、先行ゼロが消去されます。 この場合、ピクチャー文字と対応した右端の抑制位置に符号、ブランク、 または通貨記号が含まれます (すべての数字桁が浮動文字で占められ、 データ項目の値がゼロであり、浮動文字が挿入されない場合は除きます)。

浮動文字であることを示すには、ピクチャー・フィールド内にその文字を複数記入します。 浮動文字が浮動する範囲内のすべての数字桁に、その浮動文字を 記入しなければなりません。 浮動文字は、同じ浮動文字のシーケンス内になければなりません。オプションで、V と、挿入文字のコンマ、ポイント、スラッシュ、または B のいずれかが含まれます。 ストリング内またはストリングの直後にある挿入文字のスラッシュ、コンマ、またはポイントはいずれも、浮動ストリングの一部です。 文字 B が記入されている位置には常に、ブランクが挿入されます。 データ項目の算術値がゼロの場合以外は、V は浮動ストリングを終了します。その場合、V は無視されます。 ピクチャー指定の 1 つのフィールドにつき、浮動ストリングを 1 つだけ記入することができます。 浮動ストリングの前に数字桁があってはならず、ピクチャー文字 * や Z が記入されて いるフィールド内に浮動ストリングがあってはなりません。

浮動文字のストリング内にスラッシュ、コンマ、または小数点がある場合、データ内の対 応する桁には下記のいずれかの文字が入ります。

- その位置の左側に有効数字があるときは、スラッシュ、コンマ、または小数点が入ります。
- その位置のすぐ右側に、フィールドの最左端の有効数字があるときは、 浮動記号が入ります。
- その位置より複数桁右側に、フィールドの最左端の有効数字があるときは、ブランクが入ります。

浮動ストリングに *n* 個の浮動文字が含まれていると、そのストリングは、*n*-1 個の条件付き数字桁に対応させられます。 最左端の浮動文字に関連付けられる桁には、その浮動文字かブランクしか入れてはならず、数字を入れることはできません。 1 つのフィールド内で 2 つの異なるピクチャー文字を浮動的に使用することは できません。

浮動ストリング内に V がある場合、V の前にある部分が 1 つのサブフィールドになり、V のあとにあるサブフィールドの全数字桁もまた、第 2 サブフィールドとなる浮動ストリングの一部でなければなりません。

V のあとのすべての数字桁に浮動文字を記入した場合、そのサブフィールドでゼロ消去が行われるのは、整数桁の数字も小数桁の数字もすべてゼロのときだけです。 このとき、編集されたデータ項目はすべてブランクになります (ただし、フィールドの先頭に挿入文字がある場合、その挿入文字は例外です)。 小数桁にゼロ以外の数字があると、小数桁全体がゼロ消去されずに残っています。

ピクチャーへの割り当ての途中または割り当て前に、10 進数の小数桁の数字が切り捨てられ、その結果、ゼロの値になった場合は、切り捨てが行われる前の 10 進数の値に対応する符号がピクチャー内に挿入されます。 したがって、ピクチャー内の符号は、10 進数値がどのように計算されたかに 左右されます。

334 ページの表 53 に、符号と通貨記号の例を示します。

表 *53.* 符号と通貨記号の例

| ソース属性 | ソース・データ (定数フォーマット) | ピクチャー指定 | 文字値 |
|---|---|---|---|
| FIXED(5,2) | 123.45 | $999V.99 | $123.45 |
| FIXED(5,2) | 012.00 | 99$ | 12$ |
| FIXED(5,2) | 001.23 | $ZZZV.99 | $bb1.23 |
| FIXED(5,2) | 000.00 | $ZZZV.ZZ | bbbbbbb |
| FIXED(1) | 0 | $$$.$$ | bbbbbb |
| FIXED(5,2) | 123.45 | $$$9V.99 | $123.45 |
| FIXED(5,2) | 001.23 | $$$9V.99 | bb$1.23 |
| FIXED(2) | 12 | $$$,999 | bbb$012 |
| FIXED(4) | 1 2 3 4 | $$$,999 | b$1,234 |
| FIXED(5,2) | 2.45 | SZZZV.99 | +bb2.45 |
| FIXED(5) | 214 | SS,SS9 | bb+214 |
| FIXED(5) | -4 | SS,SS9 | bbbb-4 |
| FIXED(5,2) | -123.45 | +999V.99 | b123.45 |
| FIXED(5,2) | -123.45 | -999V.99 | -123.45 |
| FIXED(5,2) | 123.45 | 999V.99S | 123.45+ |
| FIXED(5,2) | 001.23 | ++B+9V.99 | bbb+1.23 |
| FIXED(5,2) | 001.23 | - - -9V.99 | bbb1.23 |
| FIXED(5,2) | -001.23 | SSS9V.99 | bb-1.23 |

# 貸方記号、借方記号、オーバーパンチ、およびゼロ置き換え文字

このセクションでは、ピクチャー文字 CR、DB、T、I、R、および Y について説明します。 これらは貸方記号、借方記号、オーバーパンチ、およびゼロ置き換え文字の各機能に使用されます。 同じフィールド内で、ピクチャー文字 CR、DB、T、I、および R をほかの符号文字と一緒に使用することはできません。

## 貸方と借方

CR (貸方) と DB (借方) は、実数の数字データ項目の符号を指定します。

**CR**
データ値が <0 の場合は、対応する位置に CR という文字を入れることを指定します。 それ以外の場合は、その位置に 2 つのブランクが入ります。 CR を記入できる位置は、フィールドの全数字桁の右側だけです。

**DB**
データ値が <0 の場合は、対応する位置に DB という文字を入れることを指定します。 それ以外の場合は、その位置に 2 つのブランクが入ります。 DB を記入できる位置は、フィールドの全数字桁の右側だけです。

## オーバーパンチ

任意のピクチャー文字 T、I、または R (オーバーパンチ文字として 知られる) は、 1 つの文字が対応する数字およびデータ項目の符号を表すことを 指定します。 浮動小数点指定は、2 つのフィールド (小数部を記述するフィールドと 指数を記述するフィールド) からなります。 オーバーパンチ文字は、フィールド内の任意の数字桁について 指定することができます。

T、I、および R ピクチャー文字は、入力文字が解釈される方法を 指定します。335 ページの表 54 に示します。

表 *54. T*、*I*、および *R* ピクチャー文字の解釈

| T または I | T または R | 数字 |
|---|---|---|
| + 付きの数字 | - 付きの数字 | |
| 文字 | 文字 | |
| { | } | 0 |
| A | J | 1 |
| B | K | 2 |
| C | L | 3 |
| D | M | 4 |
| E | N | 5 |
| F | O | 6 |
| G | P | 7 |
| H | Q | 8 |
| I | R | 9 |

T、I、および R は以下の値を指定します。

**T**
入力時には、T は、文字 { から I と数字 0 から 9 が正の値を表し、文字 } から R が負の値を表すことを指定します。

出力時には、T は、入力データが正の値を表す場合には、対応する位置に文字 { から I の 1 つを含み、入力データが負の値を表す場合には文字 } から R の 1 つを含むことを指定します。 T は '9' ピクチャー指定文字が生じるすべての場所に記入することができます。 次の例を考えてください。

```
dcl Credit picture 'ZZV9T';
```

文字表示は 4 文字です。+21.05 は '210E' として保持されます。 -0.07 は 'bb0P' として保持されます。

**I**
入力時には、I は、文字 { から I と数字 0 から 9 が正の値を表すことを指定します。

出力時には、I は、入力データが正の値を表す場合には、対応する位置に文字 { から I の 1 つを含み、入力データが負の値を表す場合には数字 0 から 9 の 1 つを含むことを指定します。

**R**
入力時には、R は、文字 } から R が負の値を表し、数字 0 から 9 が正の値を表すことを指定します。

出力では、R は、入力データが負の値を表す場合、関連する位置に } から R までの文字の 1 つが含まれることを指定します。それ以外の場合は、0 から 9 までの数字の 1 つが含まれます。 次の例を考えてください。

```
dcl X fixed decimal(3);
get edit (x) (P'R99');
```

この例では、入力ストリームの次の 3 つの位置で '132' が見つかると X を 132 に設定し、'J32' が見つかると X を -132 に設定します。

## ゼロ置き換え文字

ピクチャー文字 Y は、指定された桁位置にあるゼロを、無条件にブランク文字で置き換えることを指定します。

336 ページの表 55 に、貸方記号、借方記号、オーバーパンチ、および ゼロ置き換え文字の例を示します。

表 *55.* 貸方記号、借方記号、オーバーパンチ、およびゼロ置き換え文字の例

| ソース属性 | ソース・データ (定数フォーマット) | ピクチャー指定 | 文字値 |
|---|---|---|---|
| FIXED(3) | -123 | $Z.99CR | $1.23CR |
| FIXED(4,2) | 12.34 | $ZZV.99CR | $12.34bb |
| FIXED(4,2) | -12.34 | $ZZV.99DB | $12.34DB |
| FIXED(4,2) | 12.34 | $ZZV.99DB | $12.34bb |
| FIXED(4) | 1021 | 999I | 102A |
| FIXED(4) | -1021 | Z99R | 102J |
| FIXED(4) | 1021 | 99T9 | 10B1 |
| FIXED(5) | 00100 | yyyyy | bb1bb |
| FIXED(5) | 10203 | 9Y9Y9 | 1b2b3 |
| FIXED(5,2) | 000.04 | YYYVY9 | bbbb4 |

# 指数文字

ピクチャー文字 K と E は、浮動小数点 10 進数を記述する数字指定の 指数フィールドを区切ります。 指数フィールドは、浮動小数点の数字ピクチャー指定の最後のフィールドです。 ピクチャー文字 K と E を、同じピクチャー指定内に記入することはできません。

**K**
対応する桁の右側が指数フィールドであることを指定します。 この文字は、数字データ項目内の文字を指定するものではありません。

**E**
対応する桁に、指数フィールドの始まりを示す文字 E が入ることを指定します。

文字値の中では指数の値が調整されて、第 1 フィールド (小数部) の最初の有効数字が、ピクチャー指定内の 最初の数字指定子 (それがゼロ抑制文字であっても) に対応する桁に入るようにされます。

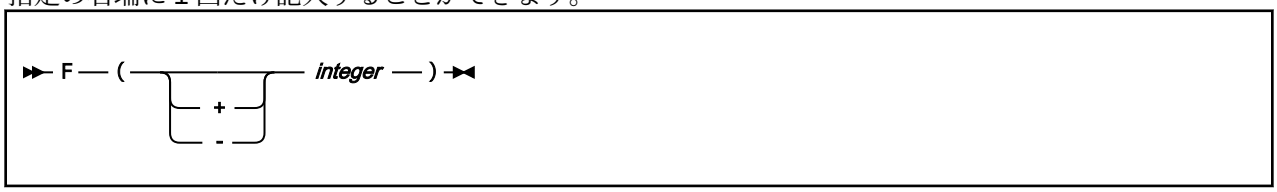
337 ページの表 56 に、指数文字の例を示します。

表 56. 指数文字の例

| ソース属性 | ソース・データ (定数フォーマット) | ピクチャー指定 | 文字値 |
|---|---|---|---|
| FLOAT(5) | .12345E06 | V.99999E99 | .12345E06 |
| FLOAT(5) | .12345E-06 | V.99999ES99 | .12345E-06 |
| FLOAT(5) | .12345E+06 | V.99999KS99 | .12345+06 |
| FLOAT(5) | -123.45E+12 | S999V.99ES99 | -123.45E+12 |
| FLOAT(5) | 001.23E-01 | SSS9.V99ESS9 | +123.00Eb-3 |
| FLOAT(5) | 001.23E+04 | ZZZV.99KS99 | 123.00+02 |
| FLOAT(5) | 001.23E+04 | SZ99V.99ES99 | +123.00E+02 |
| FLOAT(5) | 001.23E+04 | SSSSV.99E-99 | +123.00Eb02 |

## 位取り係数

ピクチャー文字 F は、固定小数点 10 進数のピクチャー位取り係数を指定する。 この文字は、ピクチャー指定の右端に 1 回だけ記入することができます。




**F**

ピクチャー位取り係数を指定します。 ピクチャー位取り係数は、変数の算術値における小数点の位置が、文字値における仮想小数点の位置からどれだけ右に (ピクチャー位取り係数が正の場合) または左に (負の場合) あるかを表します。

V ピクチャー文字に続く桁数から、F で指定した整数を減算した値は、-128 から 127 の範囲内になければなりません。

337 ページの表 57 に、ピクチャー位取り係数文字の例を示します。

表 57. 位取り係数文字の例

| ソース属性 | ソース・データ (定数フォーマット) | ピクチャー指定 | 文字値 |
|---|---|---|---|
| FIXED(4,0) | 1200 | 99F(2) | 12 |
| FIXED(7,0) | -1234500 | S999V99F(4) | -12345 |
| FIXED(5,5) | .00012 | 99F(-5) | 12 |
| FIXED(6,6) | .012345 | 999V99F(-4) | 12345 |

# 第 15 章 条件処理

PL/I プログラムの実行中には、さまざまなイベントが発生します。ユーザーは、 それらのイベントに対してテスト、応答、または回復アクションを行うことができます。 これらのイベントは、条件 と呼ばれ、検出されたときに発生 します。

条件には、起こることが予期できないエラー (オーバーフローや 入出力伝送エラーなど) もあれば、起こると分かっている エラー (入力ファイルの終わりなど) もあります。 SIGNAL ステートメントを使用して条件をプログラム内で直接発生させることができます (この方法は特にテスト中に役立ちます)。

条件をアプリケーションで制御できるようにするには、その条件を 割り込み可能 にします。また、割り込み可能な条件が起こると アクションが確立 されます。条件が割り込み禁止の場合、コンパイラーは条件が発生することはないものと想定し、それに従ってコードを生成します。 条件が発生した場合、プログラムはエラーになります。 確立されるアクションは、ON ユニットであることもあれば、 その条件に対して定義された暗黙アクションである場合もあります。

呼び出された ON ユニットは、パラメーターのないプロシージャーと見なされます。 ON ユニットの使用に役立つように、条件の原因に関して調べるための組み込み関数および疑似変数が提供されています。 疑似変数は、エラーを訂正して回復する際によく使用されます。

多くの条件では、暗黙アクションとして ERROR 条件が発生します。したがって、ERROR 条件は、それぞれの条件を個別に検査するのではなく、多数の異なる条件を検査するのに使用される共通の条件です。 この点で ONCODE 組み込み関数は特に便利なもので、これを使用すれば それらの条件を起こしている状況を区別することができます。検出される条件やエラーに対応するコードについては、「メッセージおよびコード」にリストが記載されています。

**関連情報**

373 ページの『組み込み関数、疑似変数、およびサブルーチン』
多くの共通タスクが組み込み関数、サブルーチン、および疑似変数の形で使用可能です。 これらの共通タスクを使用することにより、作成するコードの数が減り、信頼性の高い迅速な処理が できます。 この章では、PL/I プログラムで使用できる組み込み関数、サブルーチン、および疑似変数について説明します。

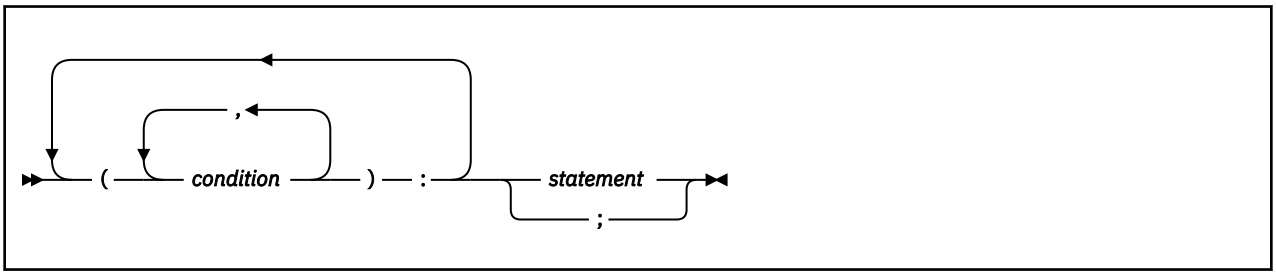
## 条件接頭語

条件接頭語を使用して、一部の条件を割り込み可能にするか割り込み禁止にするかを指定することができます。 条件が割り込み可能な場合、コンパイラーは、条件の検出に必要な追加のコードを生成します。 条件が割り込み禁止の場合は、追加のコードは生成されません。

条件を割り込み禁止にするということは、つまり条件が起こらないようにするということです。条件が起こった場合は、プログラムにエラーがあるということを意味します。 例えば、SUBSCRIPTRANGE 条件が割り込み可能な場合、コンパイラーは追加のコードを生成して、あらゆる配列指標が必ずその配列の境界内に収まるようにします。 SUBSCRIPTRANGE 条件が割り込み禁止だと、追加コードは生成されないため、無効な配列指標が使用されて予期不能な結果となります。

条件がハードウェアによって検出されると、条件の割り込み禁止は無効になります。

条件接頭語は適格条件に対してのみ指定できます。




**condition**

条件の中には、常に割り込み可能になっており、 割り込み禁止にすることができないものもあります。さらに、割り込み禁止にするのでない限り、割り込み可能になっているものもあり ます。逆に、割り

© Copyright IBM Corp. 1999, 2022

込み可能にするのでない限り、割り込み禁止になっているものもあります。 条件は、347 ページの『第 16 章 条件』にリストされています。

**ステートメント**
条件接頭語は、ステートメント DECLARE、DEFAULT、FORMAT、 OTHERWISE、END、ELSE、ENTRY、および %statements では使用することができません。 条件接頭語の有効範囲については、341 ページの『条件接頭語の有効範囲』を参照してください。

次の例では、(size): が条件接頭語です。 条件付き接頭部は、対応する条件が接頭部の有効範囲内で割り込み可能であることを表します。

```
(size):  L1:  X=(I**N) / (M+L);
```

条件を割り込み可能にするには、条件名を使用して条件接頭語を指定します。 条件を割り込み禁止にするには、NO が前に付く、途中にブランクのない条件名を使用して条件接頭語を指定します。

条件のタイプと状況を 340 ページの表 58 に示します。

表 *58.* 条件のクラスと状況

| **クラスと条件** | **状況** |
|---|---|
| **計算** (データの処理、式の計算、および計算) | |
| ASSERTION | 常に割り込み可能 |
| CONVERSION | デフォルトにより割り込み可能 |
| FIXEDOVERFLOW | デフォルトにより割り込み可能 |
| INVALIDOP | デフォルトにより割り込み可能 |
| OVERFLOW | デフォルトにより割り込み可能 |
| UNDERFLOW | 常に割り込み可能 |
| ZERODIVIDE | デフォルトにより割り込み可能 |
| **入出力** | |
| ENDFILE | 常に割り込み可能 |
| ENDPAGE | 常に割り込み可能 |
| KEY | 常に割り込み可能 |
| NAME | 常に割り込み可能 |
| 記録 | 常に割り込み可能 |
| TRANSMIT | 常に割り込み可能 |
| UNDEFINEDFILE | 常に割り込み可能 |
| **プログラム・チェックアウト** (プログラムの開発、デバッグに有効) | |
| CONFORMANCE | デフォルトにより割り込み禁止 |
| SIZE | デフォルトにより割り込み禁止 |
| STRINGRANGE | デフォルトにより割り込み禁止 |
| STRINGSIZE | デフォルトにより割り込み禁止 |
| SUBSCRIPTRANGE | デフォルトにより割り込み禁止 |
| **その他** | |
| ANYCONDITION | 常に割り込み可能 |
| AREA | 常に割り込み可能 |

表 58. 条件のクラスと状況 (続き)

| クラスと条件 | 状況 |
|---|---|
| ATTENTION | 常に割り込み可能 |
| CONDITION | 常に割り込み可能 |
| ERROR | 常に割り込み可能 |
| FINISH | 常に割り込み可能 |
| STORAGE | 常に割り込み可能 |

条件を割り込み可能にしたり、割り込み禁止にしたりすることがパフォーマンスに与える影響については、「プログラミング・ガイド」を参照してください。

## 条件接頭語の有効範囲

条件接頭語の有効範囲 (つまり、その接頭部が適用されるプログラムの部分) は、 その接頭部が付けられているステートメントまたはブロックです。 このステートメントの実行時に呼び出される可能性のあるプロシージャーや ON ユニットには、その接頭部は必ずしも適用されません。

ステートメント PACKAGE、PROCEDURE または BEGIN に付けられた条件接頭語は、対応する END ステートメントまでのすべてのステートメント (END も含めて) に 適用されます。 これには、そのブロック内にネストされている他の PROCEDURE ステートメントや BEGIN ステートメントも含まれます。

あるブロック内で条件の状況を再定義するには、そのブロック内のステートメ ント (PROCEDURE ステートメントや BEGIN ステートメントを含む) に接頭部を付け ます (このようにすれば、ネストされたブロック内で条件の割り込み可能または割り 込み禁止を再定義することができます)。 そのような再定義は、その接頭部が付けられているステートメントを実行する 場合にのみ適用されます。 ネストされた PROCEDURE ステートメントまたは BEGIN ステートメントの場合は、そのステートメントが定義されているブロックとそのブロックに含まれている他のブロック (ある場合) にのみ適用されます。

## OPTIMIZATION によって起こる条件

OPTIMIZATION が有効である場合、ある同じ式について複数回出てくる条件は、 1 回しか起こらない可能性があります。

次の例では、IX の条件 SUBSCRIPTRANGE は、1 回しか起こりません。

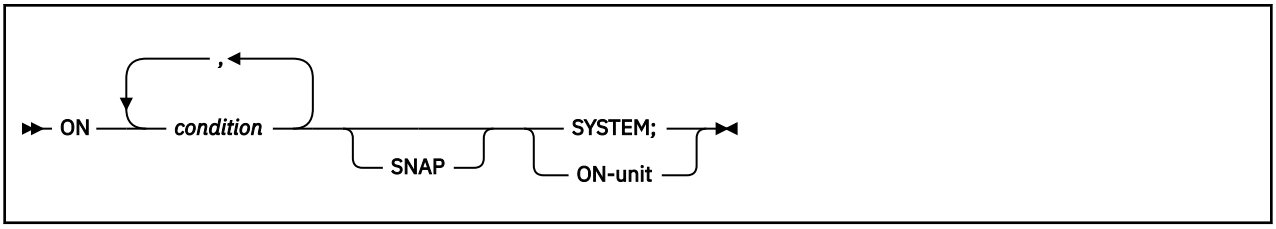
```
call P (55);
(subscriptrange): P: proc (IX);
  dcl (Ar, Br, Cr) (10);
  Ar(IX) = Ar(IX) + Br(IX);
  T = Cr(IX);
End P;
```

# ON ユニット

どの条件についても暗黙アクションが定められています。 割り込み可能な条件が起こると、この暗黙アクションが実行されます。 ただし、その条件についての ON ユニットが有効になっているときは別です。 暗黙アクションが ERROR を起こすもので、この条件に対して ON ユニットが確立されていない場合は、エラー条件が起こる前にメッセージが書き込まれます。

## ON ステートメント

ON ステートメントは、割り込み可能条件がこれ以降に、設定された条件の有効範囲内で発生した場合に実行するアクションを設定します。




**condition**
347 ページの『第 16 章 条件』に説明されている条件、または CONDITION 属性に よって定義されている条件のいずれかです。

**SNAP**
割り込み可能条件が起こったとき、その条件に関しての診断情報が印刷されることを指定します。 SNAP オプションのアクションが行われてから、ON ユニットのアクションが行われます。

SNAP と SYSTEM を指定すると、暗黙アクションが取られた直後に SNAP 情報が印刷されます。

**SYSTEM**
暗黙アクションを実行することを指定します。 すべての条件に関して同じ暗黙アクションが取られるとは限りませんが、ほとんどの条件の場合、メッセージが印刷され、ERROR 条件が引き起こされます。 条件ごとの暗黙アクションについては、347 ページの『第 16 章 条件』で説明します。

**ON ユニット (ON-unit)**
指定された条件が起こり、しかも割り込み可能であるときに実行されるアクションを指定します。 ON-unit そのものの中にステートメント (複数も可) を書いて、実行されるアクションを定義します。 ON ステートメントが実行されることを、指定の条件に 関する ON ユニットが確立される といいます。 ON ユニットは、ON ステートメントが実行される時点では実行されません。指定された条件が生じた場合のみ実行されます。

ON ユニットは、ラベルなしの単一の単純ステートメント、またはラベルなしの開始ブロックのどちらでもかまいません。 単純ステートメントの場合は、BEGIN、 DECLARE、 DEFAULT、 DO、 END、 ENTRY、 FORMAT、 ITERATE、 LEAVE、 OTHERWISE、 PROCEDURE、 RETURN、 SELECT、 WHEN、 または %statements 以外の任意のステートメントとすることができます。 ON ユニットが開始ブロックの場合は、 RETURN ステートメントは開始ブロック内のネストされた プロシージャー内にのみ現れます。 LEAVE ステートメントは開始ブロック内の ネストされた DO グループ内にのみ現れます。

ただし、セミコロン (;) または RESIGNAL ステートメントのいずれかでのみ 形成されている ON ユニットを除き、ON ユニットは、それがあるブロックに対し て内部的なプロシージャー (パラメーターなしの) と見なされます。 ON ユニット内で参照された名前は、その条件が起こったときの環境で 知られている名前ではなく、その ON ユニットを指定した ON ステートメントが 実行されたときの環境で知られている名前です。

ON ユニットの実行が完了すると、通常は、その ON ユニットに制御を 渡したブロックに制御が戻されます。 プロシージャーの場合と同様に、GO TO ステートメントを使用して ON ユニットの外に制御を移すことができ、 その場合は、GO TO ステートメントで指定した場所に制御権が移され、正常な戻りは行われません。

ON ユニットからどこの場所に制御が戻されるかは、それぞれの条件によって異なります。 条件ごとの正常な戻りについては、347 ページの『第 16 章 条件』で説明します。

# ヌル ON ユニット

ヌル・ステートメント ON ユニットが実行されると、その条件からの 正常な戻りが実行されます。

ヌル ON ユニットを使用することは、次の 2 つの理由から、条件を割り込み禁止 にすることと同じではありません。

- ヌル ON ユニットはどの条件に対しても指定できますが、条件のなかには割り込み禁止にすることができないものがあります。
- 条件を割り込み禁止にした場合、その条件が発生したかどうかは検査されないため、実行時間を節約することができます。 (ヌル ON ユニットを指定した場合は、PL/I はその条件が発生したかどうかを 検査します。)

## ON ユニットの有効範囲

ON ステートメントが実行されると、条件の発生時に取られる処置が指定されます。 設定されたアクションは、別の ON ステートメントまたは REVERT ステートメントの実行によって無効にされない限り、また ON ステートメントが実行されるブロックが終了しない限り、そのブロック全体および動的に派生したすべてのブロック全体で設定されたままになっています。

別の ON ステートメントで同じ条件を指定した場合は、以下のようになります。

- 前に実行された ON ステートメントで指定したものと同じ条件を、あとで実行 されるステートメントで指定し、前の ON ステートメントを含んでいるブロッ クから動的に派生したブロック内であとの ON ステートメントが実行されたと きは、前の ON ステートメントで指定した処置が一時的に中断され、ス タックされます。 REVERT ステートメントが実行されるか、またはあとの ON ステートメントを含んでいるブロックが終了すると、保留されていたアクションが復元されます。

  ブロックから制御が戻されると、そのブロックを活動化したときにすでに設定されていたすべてのアクションが再設定されます。 したがって、あるブロックですでに設定されているアクションを、そのブロックから呼び出されたサブルーチンで変更することはできません。
- 後ろの ON ステートメントと前の ON ステートメントの双方が、同一の呼び出しで呼び出された同一のブロックの内部にある場合は、前の ON ステートメントによって設定されたアクションは論理的に無効になります。 別の ON ステートメントを実行する (または、無効にされた ON ステートメント を再び実行する) のでない限り、無効にされたアクションが再び設定される ことはありません。

**関連情報**

343 ページの『動的に派生した ON ユニット』
ON ユニットの実行中に条件が起こり、その条件によりさらに別の ON ユニットを指定することがあります。 ある ON ユニット内で条件が起こったか、または SIGNAL ステートメントによって 条件が引き起こされたために別の ON ユニットに入った場合、その別の ON ユニット を、動的に派生した ON ユニットといいます。 動的に派生した ON ユニットから正常に戻ると、その条件を 引き起こした ON ユニットの環境が再び設定されます。

## 動的に派生した ON ユニット

ON ユニットの実行中に条件が起こり、その条件によりさらに別の ON ユニットを指定することがあります。 ある ON ユニット内で条件が起こったか、または SIGNAL ステートメントによって 条件が引き起こされたために別の ON ユニットに入った場合、その別の ON ユニット を、動的に派生した ON ユニットといいます。 動的に派生した ON ユニットから正常に戻ると、その条件を 引き起こした ON ユニットの環境が再び設定されます。

ERROR ON ユニット内で ERROR 条件が起こったとき、同一の ERROR ON ユニットが 実行されると、再び ERROR 条件が起こってループ状態になります。 ループ状態になって最大ネスト・レベルを超えると、 メッセージが印刷され、アプリケーションは終了します。 次の方法を使用すれば、 このような状況で発生するループを防止することができます。

```
on error begin;
   on error system;
   .
   .
   .
   end;
```

## ファイル変数のための ON ユニット

ON ステートメントでファイル変数を指定した場合は、その ON ユニットの設定時にその変数の現行値であるファイル定数が参照されます。

**例 1**

```
dcl  F file,
     G file variable;
     G = F;
```

```
L1:  on endfile(G);
L2:  on endfile(F);
```

ラベル L1 と L2 が付いている 2 つのステートメントは同等です。

**例 2**

```
declare FV file variable,
        FC1 file,
        FC2 file;
FV = FC1;
on endfile(FV) go to Fin;
.
.
.
FV = FC2;
read file(FC1) into (X1);
read file(FV) into (X2);
```

最初の READ ステートメントの実行中に ENDFILE 条件が起こると、 ON ユニットに入ります。これは、 この ON ユニットがファイル FC1 を参照しているためです。 しかし、2 番目の READ ステートメントの実行中に ENDFILE 条件が起こっても、 この READ ではファイル FC2 を参照しているので、 この ON ユニットには入りません。

**例 3**

```
E:  procedure;
  declare F1 file;
  on endfile (F1) goto L1;
  call E1 (F1);
  .
  .
  .
E1:  procedure (F2);
   declare F2 file;
   on endfile (F2) go to L2;
   read file (F1);
   read file (F2);
   end E1;
```

E1 の中で F1 がファイルの終わりになると、E1 の中の F2 用の ON ユニットに入ります。 E1 の中で ON ユニットを指定しなかったとき、 F1 または F2 の ENDFILE 条件が起こった場合には、 E の中の F1 用の ON ユニットに入ることになります。

**例 4**

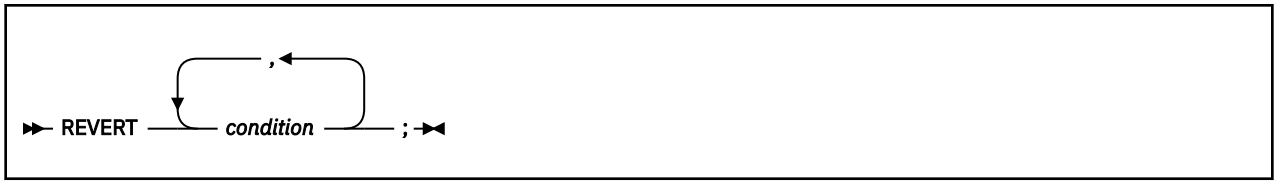
```
declare FV file variable,
        FC1 file,
        FC2 file;


do FV=FC1,FC2;
   on endfile(FV) go to Fin;
end;
```

ファイル変数を指定している ON ステートメントが複数回実行され、なおかつ、その変数がそのつど 異なる値を取る場合は、それぞれの実行のたびに別々の ON ユニットが設定されます。

# REVERT ステートメント

あるブロックで REVERT ステートメントが実行されると、指定の条件に関してそ のブロック内で実行された ON ユニットが取り消されます。 そして、そのブロックが活動化されたときにすでに設定されていた ON ユニット が再び設定されます。

REVERT ステートメントの影響を受ける ON ステートメントは、その REVERT ステー トメントを含んでいるブロックの内部にある ON ステートメントのうち、同一の 呼び出しで呼び出されたそのブロック内ですでに実行されたものだけです。




**condition**
347 ページの『第 16 章 条件』に説明されている条件、または CONDITION 属性に よって定義されている条件のいずれかです。

REVERT ステートメントが ON ユニットを取り消すのは、 次の 2 つの条件が両方とも成立する場合に限ります。

1. 派生元に適用可能であり、かつ REVERT ステートメントでリストされた条件 を指定する ON ステートメントが、ブロックの活動化後に実行された。
2. 指定された条件付きの REVERT ステートメントが、同じブロック内で 実行されたことがない。

これらの 2 つの条件のどちらかが成立しない場合には、REVERT ステートメントはヌル・ステートメントと見なされます。

## SIGNAL ステートメント

SIGNAL ステートメントを使用すれば、条件を引き起こすことができます。 プログラムのテスト時にこのステートメントを使用して、ON ユニットのアクションが妥当であるかどうかを検査したり、指定の条件に正しいアクションが対応しているかを判別したりすることができます。 条件が割り込み禁止になっていない限り、設定されているアクションが取られます。

指定した条件が割り込み禁止になっているときは、SIGNAL ステートメントはヌル・ステートメントと同等になります。

**condition**
347 ページの『第 16 章 条件』に説明されている条件、または CONDITION 属性によって 定義されている条件のいずれかです。

## RESIGNAL ステートメント

RESIGNAL ステートメントによって、現行の ON ユニットは終了して、別の ON ユニットが同一の条件で制御を獲得できるようになります。

処理は、RESIGNAL を実行している ON ユニットが存在せず、 制御を獲得しなかった場合と同様に続けられます。 これによって、複数の ON ユニットが同一条件で制御を獲得することができます。

ただし、RESIGNAL は 1 個の ON ユニット内、またはそこから動的に派生したものにおいてのみ有効です。

## 複数条件

複数条件とは、複数の条件が同時に起こることをいいます。

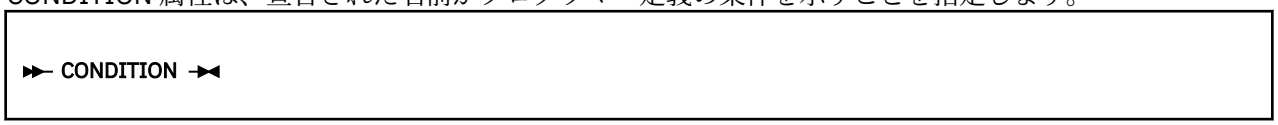
複数条件が発生し得る条件は、358 ページの『RECORD 条件』および 362 ページの『TRANSMIT 条件』です。 TRANSMIT 条件が常に最初に処理されます。 TRANSMIT の ON ユニットから正常に戻らなかったときは、RECORD 条件は 無視されます。

複数条件は、順番に処理されます。 次のイベントのいずれかが起こると、後続の条件は処理されません。

- ある条件の処理中に、その条件の暗黙アクション、ON ユニットからの正常な戻 り、または ON ユニット内の異常終了によって、プログラムが終了した場合。
- GO TO ステートメントによって ON ユニットの外に制御が移され、その結果、正常に戻ることができなかった場合。

## CONDITION 属性

CONDITION 属性は、宣言された名前がプログラマー定義の条件を示すことを指定します。




ON、SIGNAL、 または REVERT ステートメント内に CONDITION 条件と一緒に書いた名前は、条件名であることがコンテキスト上から宣言されます。

デフォルトの有効範囲は EXTERNAL です。 CONDITION 条件の例については、350 ページの『CONDITION 条件』を参照してください。

# 第 16 章 条件

この章では、各条件をアルファベット順に説明します。

一般的には、それぞれの条件ごとに下記のことがらを説明します。

**状況**
プログラムの開始時における条件の割り込み可能状況または割り込み禁止状況、および条件を割り込み禁止 (可能な場合) または割り込み可能にする方法の表示。

340 ページの表 58 は、条件をタイプ別に分類し、状況を示したものです。また、 これは、割り込み可能なものを割り込み禁止にする条件をリストします。

**Result**
条件を引き起こした操作の結果。 条件が割り込み可能になっている場合と割り込み禁止になっている場合に分けて説明します。 場合によっては、結果が未定義のこともあります。

**原因と構文**
どのような場合にその条件が起こるかを含め、条件について述べます。 SIGNAL ステートメントによって発生する条件については、345 ページの『SIGNAL ステートメント』を参照してください。

**暗黙アクション**
使用可能な条件が生じたときに、その条件用に現在確立されている ON ユニットがない場合にとられる処置。

**正常な戻り**
ON ユニットが正常終了した場合に制御が戻される場所。 ON ユニットの外へ制御権を移動する GO TO ステートメントは、ON ユニットの異常終了です。 SIGNAL ステートメントによって条件 (ERROR 条件は除く) が引き起こされた場合、正常な戻りでは、SIGNAL のすぐあとのステートメントに戻ります。

**条件コード**
プログラムの検査で検出される条件やエラーに対応するコード。 各コードの説明は、「メッセージおよびコード」の『"条件コード"』の章に記載されています。

## ANYCONDITION 条件

**状況**
ANYCONDITION は常に使用可能です。

**Result**
結果は、基礎的な条件の場合と同じです。

**原因と構文**
SIGNAL ANYCONDITION は使用できません。 ANYCONDITION は、条件をトラップする ON ユニットを確立 (または取り消し) する ON (および REVERT) ステートメントでのみ使用できます。 それらの条件には、CONDITION 条件も含みます。 この条件はブロック内で起こるもので、そのブロック内にあるほかの適格な ON ユニットによってトラップされません。

次に示す例では、すべての ERROR 条件は、開始ブロック内で処理され、 FINISH 条件は、システムによって処理され、その他のすべての条件は、 `handle_All_Others` という名前のルーチンの呼び出しによって処理さ れることになります。

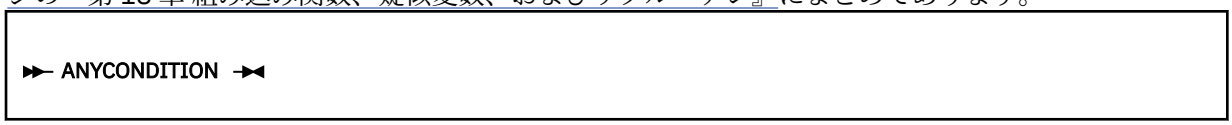
```
on error
  begin;
  .
  .
  .
  end;

on finish system;
on anycondition call Handle_all_others;
```

© Copyright IBM Corp. 1999, 2022

**注 :** 無限ループを避けるために、ON ANYCONDITION の使用時には、ON FINISH の使用 (前の例に記載) が必要になる場合があります。

条件が発生すると、呼び出しスタックが調べられ (逆方向に)、その条件の ON ユニットを含むブロックが検索されます。 そのような ON ユニット、または ON ANYCONDITION ON ユニットを 含む最初のブロックが検出されると、検索は停止します。 そのような ON が検出されず、その条件が暗黙アクションによって ERROR に プロモートされる場合は (またその場合に限り)、スタックが再度調べられ、ON ERROR ON ユニットが検索されます。

ANYCONDITION ON ユニット内で ONCONDID 組み込み関数を使用して、どの条件が処理 されているかを判別することができます。また、ONCONDCOND 組み込み関数を使用 すれば、CONDITION 条件の名前を判別することができます。 ONFILE のようなほかの ON 組み込み関数を使用すれば、正確な原因を判別し、その他の関連する情報を知ることができます。 これらの組み込み関数については、373 ページの『第 18 章 組み込み関数、疑似変数、およびサブルーチン』にまとめてあります。




**省略形**
ANYCOND

**暗黙アクション**
暗黙アクションは、基礎的な条件の場合と同じです。

**正常な戻り**
正常な戻りは、基礎的な条件の場合と同じです。

**条件コード**
ANYCONDITION に固有の条件コードはありません。

## AREA 条件

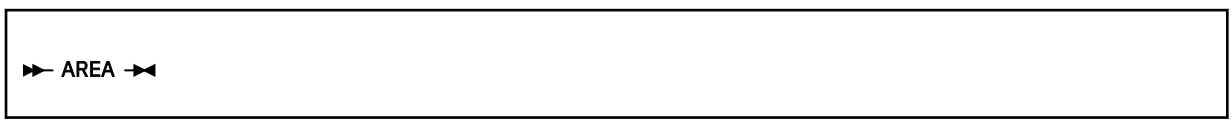
**状況**
AREA 条件は常に割り込み可能です。

**Result**
AREA 条件を引き起こす割り振りまたは割り当ての試みは、なんの影響も及ぼしません。

**原因と構文**
AREA 条件は、次のいずれかの場合に起こります。

- ある区域内に基底付き変数を割り振ろうとしたが、その割り振りに必要なフリー・ストレージがその区域内にない場合。
- 区域の割り当てを実行しようとしたが、ソース区域内の割り振りを収容するのに必要なストレージがターゲット区域内にない場合。

ON ユニットで ONAREA 組み込み関数を使用して、AREA 条件を引き起こした AREA 参照の名前を取得できます。 ONAREA 組み込み関数について詳しくは、373 ページの『第 18 章 組み込み関数、疑似変数、およびサブルーチン』の中の 491 ページの『ONAREA』を参照してください。




**暗黙アクション**
メッセージが印刷され、ERROR 条件が起こります。

**正常な戻り**
ON ユニットからの正常な戻りに際して、次の処置が取られます。

- この条件が割り振りによって起こり、しかも ON ユニットがヌル ON ユニットである 場合は、割り振りが再び試みられることはありません。
- この条件が割り振りによって起こった場合は、割り振りが再び試みられますが、その前に、区域参照が計算し直されます。 したがって、ON ユニットで、不適当な区域を参照するよう修飾しているポイ

ン ターの値を変更して、そのポインターが別の区域を指し示すようにしておけば、 新しい区域内で割り振りが再び試みられます。

- この条件が区域割り当てまたは SIGNAL ステートメントによって起こった場合は、この条件が起こった場所から実行が続行されます。

**条件コード**
360、361、362

## ASSERTION 条件

**状況**
ASSERTION 条件は常に割り込み可能です。

**Result**
この条件が起こると、ASSERTION の ON ユニットに入ります。 ASSERTION ON ユニットがない場合は、アプリケーションが終了し、メッセージが出され、ERROR 条件が発生します。

**原因と構文**
ASSERT ステートメントが失敗し、ASSERT(CONDITION) コンパイラー・オプションが有効な場合、ASSERTION 条件が発生します。

またこの条件は、SIGNAL ASSERTION ステートメントによっても起こります。

ASSERTION ステートメントによって ASSERT 条件が引き起こされると、ONTEXT 組み込み関数はその text 節の値を返します。

ASSERT COMPARE ステートメントによって ASSERTION 条件が引き起こされると、ONACTUAL、ONEXPECTED、および ONOPERATOR 組み込み関数は、 その COMPARE 節の 3 つのエレメントの値を返します。

**暗黙アクション**
メッセージが印刷され、ERROR 条件が起こります。

**正常な戻り**
ASSERTION ON ユニットから正常に戻ると、条件を発生させたステートメントの直後の次のステートメントから処理が再開されます。

**条件コード**
430,431,432,433,434,435,436

## ATTENTION 条件

**状況**
ATTENTION 条件は常に割り込み可能です。

**Result**
この条件が起こると、ATTENTION の ON ユニットに入ります。 ATTENTION ON ユニットがない場合は、この条件は無視されるため、アプリケーションは終了されます。

**原因と構文**
ATTENTION 条件は、ユーザーが、特定のキーの組み合わせを押してアプリケーションに割り込んだときに起こります。 この特定のキーは、オペレーティング・システムによって以下のように異なります。

- Windows の場合、CTRL-BRK および CTRL-C。ユーザーが CTRL-BRK または CTRL-C キーの組み合わせを入力しても、Windows では ATTENTION ON ユニットは駆動されません。 暗黙アクションが実行されます。
- ホストでは、(使用可能であれば) ATTN キー。

またこの条件は、SIGNAL ATTENTION ステートメントによっても起こります。

**省略形**
ATTN

**暗黙アクション**
アプリケーションは終了しました。

**正常な戻り**
ATTENTION の ON ユニットから戻ると、プログラム内の、この条件が起こった場所のすぐあとから処理が再開されます。

**条件コード**
400

## CONDITION 条件

**状況**
CONDITION 条件は常に割り込み可能です。

**結果**
CONDITION 条件を使用すると、該当する CONDITION 条件の SIGNAL ステートメ ントが実行されるたびに実行されるような ON ユニットを確立することができま す。

デバッグの援助として、この条件を使用して、プログラムの現在の状況を示す 情報を印刷する ON ユニットを確立することができます。

**原因と構文**
CONDITION 条件は、SIGNAL ステートメントによって引き起こされます。 SIGNAL ステートメントにどの名前が指定されているかによって、どの CONDITION 条件が引き起こされるかが決まります。 SIGNAL ステートメントを使用すれば、プログラム内のどこからでもその ON ユニットを実行させることができます。 名前の有効範囲に関しては、通常の規則が適用されます。 条件名はデフォルトでは EXTERNAL ですが、INTERNAL と宣言することもできます。

次に、CONDITION 条件の使用例を示します。

```
dcl Test condition;


on condition (Test)
begin;
⋮
end;
```

開始ブロックは、 次に示すステートメントが実行されるときは必ず実行されます。

```
  signal condition (Test);
```

**省略形**
COND

**暗黙アクション**
メッセージが印刷され、SIGNAL ステートメントの次のステートメントから 実行が続行されます。

**正常な戻り**
SIGNAL ステートメントの次のステートメントから実行が続けられます。

**条件コード**
500

## CONFORMANCE 条件

**状況**
CONFORMANCE は、CONFORMANCE 条件接頭語の有効範囲内を除き、プログラム全体で使用不可になります。

**結果**
CONFORMANCE が発生した場合は、プログラムにエラーがあります。

**原因と構文**
CONFORMANCE 条件は、以下の場合に常に発生します。

1. ORDINALNAME、ORDINALPRED、および ORDINALSUCC 内の引数が無効である。
2. PROCEDURE に渡された引数の属性が、以下のいずれかの状況で予期される属性と異なります。
   - パラメーターが、定数の長さで宣言されたストリング (またはストリングの配列) であり、渡された引数の長さが一致しない場合。
   - パラメーターがストリング (またはストリングの配列) であり、引数が同じ長さのタイプ (VARYING、NONVARYING、または VARYINGZ) ではない場合。
   - パラメーターが (スカラーまたは構造体の) 配列であり、すべてのエクステントが定数であるが、引数内の配列エレメントのサイズとスペーシングがパラメーター内のサイズとスペーシングと一致しない場合。
   - パラメーターが定数エクステントを持つ構造体または共用体であり、最後のエレメントのオフセットが渡された引数のオフセットと一致しない場合。
   - プロシージャーにストリング・タイプの RETURNS BYADDR 属性があり、RETURNS 値に渡されたストリングに、一致するストリング・タイプまたは一致する長さがない場合。

     以下のいずれかの条件が該当する場合、引数とパラメーターの不一致によって CONFORMANCE 条件が発生することはありません。

     – PROCEDURE に NODESCRIPTOR オプションが適用されている。
     – PROCEDURE に ENTRY ステートメントが含まれている。
     – CMPAT(LE) オプションが有効である。

**暗黙アクション**
メッセージが印刷され、ERROR 条件が起こります。

**正常な戻り**
CONFORMANCE ON ユニットから正常に戻ると、ERROR 条件が発生します。

**条件コード**
550-559

## CONVERSION 条件

**状況**
CONVERSION は、NOCONVERSION 条件接頭語の有効範囲内を除き、プログラム全体にわたって割り込み可能です。 ONSOURCE、ONCHAR、ONUCHAR、ONUSOURCE、ONGSOURCE、および ONWSOURCE 疑似変数を CONVERSION の ON ユニットで使用して、変換エラーを訂正できます。

**結果**
CONVERSION 条件が起こったとき、結果フィールド全体の内容は未定義です。

**原因と構文**
CONVERSION 計算条件は、文字、uchar、ワイド文字、またはグラフィックのデータに対して無効な変換が試みられると発生します。 この試みは、内部的に行われることも、入出力処理中に行われることもあります。 例えば、以下のいずれかの変換が行われると、この条件が発生します。

- ビット・データに変換される文字データ内に、0 でも 1 でもない文字がある場合。
- 数字フィールドまたはコード化算術値に変換される文字値の中に、算術定数 (符 号は任意) の表記でない文字や、複素定数を表す式が含まれている場合。
- 文字に変換されるグラフィック (DBCS) ストリングが、SBCS に変換できないグラフィックを含む場合。
- 文字ピクチャー項目に変換される値の中に、そのピクチャー指定で許可されていない文字が含まれている場合。

文字データの変換は、必ず左から右へ文字単位で行われ、 無効文字が検出されるたびに、この条件が起こります。 また、すべての文字がブランクのときもこの条件が起こります。ただし、下記の場合は例外です。

- F フォーマット項目の入力の場合。ゼロの値が想定されます。
- E フォーマット項目の入力の場合。場合によっては、ON ユニットに繰り返し入ることがあるので注意してください。

ヌル・ストリングまたは 1 つ以上のブランクのストリングが数値変数に割り当てられると、CONVERSION 条件が not 発生することに注意してください。

CONVERSION 条件が発生すると、ONSUBCODE 組み込み関数は、問題の文字またはグラフィックの索引を返します。

先行ゼロの除外が、この条件に対して指定されている現在の処置が実行されます (CONVERSION が割り込み禁止でない場合)。 指定された処置が ON ユニットである場合、無効文字はその ON ユニット内で置き換えることができます。

- ソース・データが文字のときは、ONSOURCE または ONCHAR 疑似変数を使用します。
- ソース・データが uchar のときは、ONUSOURCE または ONUCHAR 疑似変数を使用します。
- ソース・データがワイド文字のときは、ONWSOURCE または ONWCHAR 疑似変数 を使用します。
- ソース・データがグラフィックのときは、ONGSOURCE 類似変数を使用します。

CONVERSION 条件が起こったとき、この条件が割り込み禁止になっていると、プログラムはエラーになります。

グラフィック・データから非グラフィック・データへの変換中に CONVERSION 条件が 起きた場合、ONCHAR および ONSOURCE 組み込み関数には有効なソース・データは 含まれません。 ONGSOURCE 組み込み関数には、もとのグラフィック・ソース・データが入っています。 グラフィック変換は、ONGSOURCE 疑似変数が CONVERSION の ON ユニットで使 用され、CONVERSION 条件を起こしたグラフィック・データの修正を試みる場合 に再試行されます。 ONGSOURCE 疑似変数を CONVERSION の ON ユニットで使用しない場合、ERROR 条件が 起こります。

**省略形**
CONV

**暗黙アクション**
メッセージが印刷され、ERROR 条件が起こります。

**正常な戻り**
CONVERSION 条件が文字ストリング・ソース (グラフィック・ソースではない) で引き起 こされ、かつ ONSOURCE または ONCHAR 疑似変数のいずれかが ON ユニットで使用さ れる場合、プログラムは ON ユニットからの戻り時に変換を再試行します。

CONVERSION がグラフィック・ソースで引き起こされ、かつ ONGSOURCE 疑似変数が ON ユニット で使用される場合、プログラムは ON ユニットからの戻り時に変換を 再試行します。

CONVERSION が uchar ソースで引き起こされ、かつ ONUSOURCE 疑似変数が ON ユニットで使用される場合、プログラムは ON ユニットからの戻り時に変換を再試行します。

CONVERSION がワイド文字ソースで引き起こされ、かつ ONWSOURCE 疑似変数が ON ユニット で使用される場合、プログラムは ON ユニットからの戻り時に変換を 再試行します。

変換エラーがこれらの疑似変数によって修正されないと、プログラムはループします。

**条件コード**
600-672

## ENDFILE 条件

**状況**
ENDFILE 条件は常に割り込み可能です。

**結果**
指定されたファイルがこの条件が起こってからクローズされない場合、こ のファイルに対する後続の GET または READ ステートメントは失敗し、これによっ て新たな ENDFILE 条件が引き起こされます。

ただし、ファイルは、ENDFILE ON ユニットでクローズしてはいけません。つまり、ファイルは ON ユニットの終了後にのみクローズします。

**原因と構文**
ENDFILE 入出力条件は処理中に起こる可能性があり、GET ステートメントま たは READ ステートメントで指定したファイルの終わりを超えて読み取ろうとする と起こります。 この条件は、SEQUENTIAL INPUT ファイル、SEQUENTIAL UPDATE ファイル、および STREAM INPUT ファイルにのみ適用されます。

レコード単位データ伝送では、READ ステートメントの実行中にファイルの終わりが検出されると、ENDFILE 条件が起こります。

ストリーム指向データ伝送では、GET ステートメントの実行中に、GET ステートメントのデータ・リスト内の項目の伝送が完了する前、またはデータ項目が伝送されたあと次のデータ項目が伝送される前のいずれかでファイルの終わりが検出されると、ENDFILE 条件が起こります。 データ項目の処理中にファイルの終わりが検出された場合や、X フォーマット項目の処理中 にファイルの終わりが検出された場合は、ERROR 条件が起こります。

***file-reference***
ファイル参照はスカラー参照でなければなりません。 ファイル参照を省略する場合、SYSIN が想定されます。

**暗黙アクション**
メッセージが印刷され、ERROR 条件が起こります。

**正常な戻り**
ENDFILE を引き起こした GET または READ ステートメントのすぐあとのステートメントから実行が続けられます。

この条件の ON ユニット内でファイルをクローズした場合は、正常な戻りの結果がどうなるかはわかりません。 ファイルをクローズした ON ユニットから出るときは、GO TO ステートメントを使用しなければなりません。

**条件コード**
70

## ENDPAGE 条件

**状況**
ENDPAGE 条件は常に割り込み可能です。

**Result**
ENDPAGE 条件が起こると、現在行の行番号は、PAGESIZE オプション (デフォルトは 60) で指定した行番号より 1 だけ大きい値になります。 したがって、同じページに書き続けることができます。 ON ユニットで PAGE オプションまたは PAGE フォーマット項目を実行して現在行を 1 にセット すれば、新しいページを開始することができます。

ON ユニットで新しいページを開始しない限り、現在行の行番号は無限に増えてゆく可能性があります。 後続の LINE オプションまたは LINE フォーマット項目で現在行の行番号以下の行番号を指定すれば、ENDPAGE 条件は起こらず、現在行が 1 にセットされて新しいページが開始されます。 ただし例外として、現在行の行番号が指定された行番号に等しく、しかもファイルが現在行の 1 桁目に位置づけられている場合は、ENDPAGE 条件は起こりません。

データ伝送中に ENDPAGE 条件が起こったときは、ON ユニットから戻ると、データが現在行 (ON ユニットで変更された可能性がある) に書き出されます。 ENDPAGE 条件が LINE または SKIP オプションによって引き起こされた ときは、ON ユニットから戻ると、LINE または SKIP で指定された処置は 無視されます。

**原因と構文**
PUT ステートメントが実行された結果として、現行ページに関して指定されている限界を超えて新しい行を開始しようとすると、ENDPAGE 入出力条件が起こります。 この限界は OPEN ステートメントの PAGESIZE オプションで指定することができます。PAGESIZE を指定しないと、デフォルトの限界である 60 が適用されます。 データ伝送時に (編集ディレクティブの PUT ステートメントの場合は対応するフォーマット項目も含む)、LINE オプションまたは SKIP オプションを使用すれば、この限界を超えることができます。 また、LINE オプションまたは LINE フォーマット項目で、現在行の行番号より小さい行番号を指定したときも、ENDPAGE 条件が起こることがあります。 ENDPAGE 条件は、SIGNAL ステートメントによって引き起こされる場合を除いて、1 ページに 1 回しか起こりません。

***file-reference***
ファイル参照はスカラー参照でなければなりません。 ファイル参照を省略する場合、SYSPRINT が想定されます。

**暗黙アクション**
新しいページが開始されます。 この条件が SIGNAL ステートメントによって引き起こされた場合、実行は影響を 受けず、SIGNAL ステートメントの次のステートメントから実行は継続されます。

**正常な戻り**
上で述べたようにして PUT ステートメントの実行が続けられます。

**条件コード**
90

# ERROR 条件

**状況**
ERROR 条件は、常に割り込み可能です。

**結果**
エラー・メッセージは、ON ユニットがアクティブでない場合に ERROR 条件が起 こったときか、または ON ユニットが GOTO (ブロックから出る) を使用してこの条 件からの回復を行わないときに出されます。

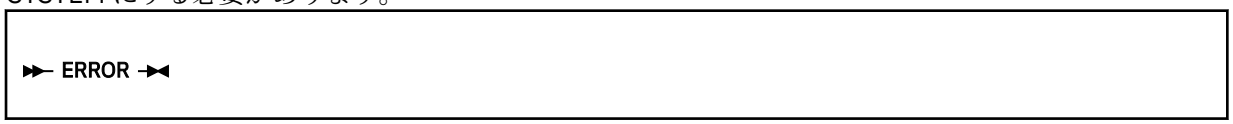
**原因と構文**
ERROR 条件は、多くの条件の暗黙アクションです。 したがって、ERROR 条件は、それぞれの条件を個別に検査するのではなく、多数の異なる条件を検査するのに使用される共通の条件です。

ERROR 条件は、次の場合に起こります。

- ある条件で暗黙アクションが取られ、それによって ERROR 条件が引き起こされた場合。

- SUBSCRIPTRANGE CONVERSION などの条件の正常な戻り処置が取られたとき、 または再試行が行われない場合。
- プログラムの実行中にエラー (ほかの PL/I 定義の条件を引き起こさないもの) が起こった場合。
- SIGNAL ERROR ステートメントを実行した場合。

ERROR 条件のループを防ぐには、ON ERROR ブロック内の最初のステートメントを ON ERROR SYSTEM にする必要があります。




**暗黙アクション**
メッセージが印刷され、FINISH 条件が起こります。

**正常な戻り**
暗黙アクションが取られます。

**条件コード**
1000 以上のすべてのコードは ERROR 条件です。

## FINISH 条件

**状況**
FINISH 条件は常に割り込み可能です。

**Result**
制御が FINISH の ON ユニットに移り、処理を続行します。

**原因と構文**
FINISH 条件は、プロシージャーを終了するステートメントの実行中に起こります。 次の処置が行われます。

- 正常終了の場合 - 主プロシージャーが PL/I の場合にのみ、FINISH ON ユニット (確立されている場合) に制御が移ります。
- 異常終了の場合 - FINISH ON ユニットがアクティブ・ブロックに ある場合は、FINISH ON ユニットに制御が移ります。

**暗黙アクション**

- この条件がメジャー・タスクで起こった場合、なんの処置も取られません。 つまり、この条件が起こった場所から処理が続けられます。
- この条件が別の条件の暗黙アクションの一部として起こった場合、 プログラムは終了されます。

**正常な戻り**
処理は、その条件が起こった場所から再開します。 条件が SIGNAL ステートメントによって引き起こされた場合、その SIGNAL ステー トメントのあとのステートメントで処理が再開されます。

**条件コード**
4

## FIXEDOVERFLOW 条件

**状況**
NOFIXEDOVERFLOW を指定している条件接頭語の有効範囲を除けば、FIXEDOVERFLOW 条件はプログラム全体にわたって割り込み可能です。

**結果**
無効な FIXED DECIMAL 操作の結果は未定義です。

**原因と構文**
計算関係の条件である FIXEDOVERFLOW 条件は、FIXED DECIMAL 算術演算の結果の長さが、有効な最大の長さを超えたときに起こります。

FIXED BINARY 操作の場合は、FIXEDOVERFLOW 条件は起こりません。

FIXEDOVERFLOW 条件と SIZE 条件は次の点で異なっています。SIZE 条件は、演算結果が変数の宣言されているサイズを超えたときに起こります。一方、FIXEDOVERFLOW 条件は、演算結果の長さが計算機で許可されている最大の長さを超えたときに起こります。

FIXEDOVERFLOW 条件が起こったとき、この条件が割り込み禁止になっていると、プログラムはエラーになります。

**省略形**
FOFL

**暗黙アクション**
メッセージが印刷され、ERROR 条件が起こります。

**正常な戻り**
この条件が起こったすぐあとの位置に制御が戻されます。

**条件コード**
310

**注 :** SIZE 条件が割り込み禁止になっているときに、固定小数点 10 進変数に大きすぎる数を割り当てようとすると、FIXEDOVERFLOW 条件が起こることがあります。

## INVALIDOP 条件

**状況**
INVALIDOP 条件は、NOINVALIDOP 条件接頭語の有効範囲を除いて、プログラムの全体にわたって割り込み可能です。

**結果**
無効な操作の結果は未定義です。

**原因と構文**
計算関係の条件である INVALIDOP 条件は、次のいずれかが IEEE 浮動小数点式の計算中に検出されたときに起こります。

- 無限大と無限大の減算
- 無限大に 0 を乗算
- 無限大と無限大の除算
- ゼロをゼロで除算
- 無効な浮動小数点データ

一部の固定小数点の高い精度での除算は、Decimal Floating-Point (DFP) 機能を使用して実行できます。これにより、一部の ZERODIVIDE 例外は INVALIDOP として報告される可能性があります。

**暗黙アクション**
ERROR 条件が起こります。

**正常な戻り**
メッセージが印刷され、ERROR 条件が起こります。

**条件コード**
290

**関連情報**
365 ページの『ZERODIVIDE 条件』

## KEY 条件

**状況**
KEY 条件は常に割り込み可能です。

**Result**
キー順レコードが未定義のため、キー順レコードが入っているステートメントは無視されます。

**原因と構文**
KEY 入出力条件は、指定されたキーを持つレコードが検出されないときに起こります。 この条件は、キー順レコードの処理中にのみ起こる可能性があります。 また、以下にリストした条件コードの場合にこの条件が起こります。

データ・セットを参照する LOCATE ステートメントを使用したとき、 この LOCATE ステートメントに対する KEY 条件は、 そのファイルを参照する次の WRITE または LOCATE ステートメントが 実行されるときまで、 もしくはファイルがクローズされるときまで起こりません。

file-reference (ファイル参照) はスカラー参照でなければなりません。

**暗黙アクション**
メッセージが印刷され、ERROR 条件が起こります。

**正常な戻り**
KEY 条件を引き起こしたステートメントのすぐあとのステートメントに制御が渡されます。

この条件の ON ユニット内でファイルをクローズした場合は、正常な戻りの結果がどうなるかはわかりません。 ファイルをクローズした ON ユニットから出るときは、GO TO ステートメントを使用しなければなりません。

**条件コード**
50-58

## NAME 条件

**状況**
NAME 条件は常に割り込み可能です。

**Result**
指定されたデータは未定義です。

**原因と構文**
NAME 入出力条件は、FILE オプションが指定されたデータ・ディレクティブ GET ステートメントの実行中に起こる可能性があります。 この条件は、次のいずれの場合にも起こります。

- 303 ページの『データ・ディレクティブ・データの構文』で説明した構文規則に従っていない場合。
- 名前が欠落しているか、正しくない場合。 以下に例をいくつか示します。
  - データ・リスト中に対応する名前がない。
  - データ・リストがない場合に、その名前がこのブロック内で知られていない。
  - 修飾された名前が完全には修飾されていない。
  - DBCS に、有効範囲 ('41'X から 'FE'X まで) にないバイトが含まれている。
- 添え字リストが欠落しているか、正しくない場合。 以下に例をいくつか示します。

- 添え字が欠落している。
- 添え字の個数が正しくない。
- 添え字が 10 桁を超える (先行ゼロは無視される)。
- 添え字が、変数の現在の割り振りで使用できる範囲内にない。

ON ユニット内で DATAFIELD 組み込み関数を使用すると、不正なデータ・フィールドを検索する場合があります。

file-reference (ファイル参照) はスカラー参照でなければなりません。

**暗黙アクション**
正しくないデータ・フィールドが無視され、メッセージが印刷されて、GET ステートメントの実行が続行されます。

**正常な戻り**
ストリーム中の次の名前を使用して GET ステートメントの実行が続けられます。

**条件コード**
10

## OVERFLOW 条件

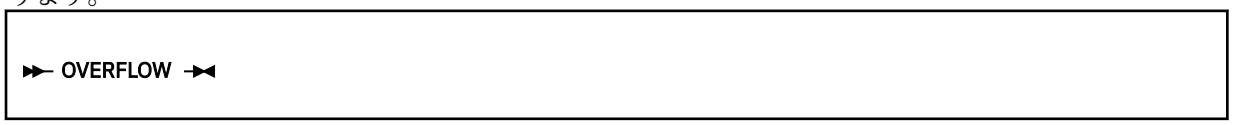
**状況**
NOOVERFLOW を指定している条件接頭語の有効範囲を除けば、OVERFLOW 条 件はプログラム全体にわたって割り込み可能です。

**Result**
無効な浮動小数点演算の結果は未定義です。

**原因と構文**
計算関係の条件である OVERFLOW 条件は、浮動小数点数の大きさがシステムで 定められている最大の大きさを超えたときに起こります。

OVERFLOW 条件と SIZE 条件は次の点で異っています。SIZE 条件は、演算結果が変数の宣言されているサイズを超えたときに起こります。一方 OVERFLOW 条件は、演算結果が計算機で許可されている最大の大きさを超えたときに起こります。

OVERFLOW 条件が起こったとき、この条件が割り込み禁止になっていると、プログラムはエラーになります。




**省略形**
OFL

**暗黙アクション**
メッセージが印刷され、ERROR 条件が起こります。

**正常な戻り**
ERROR 条件が起こります。

**条件コード**
300

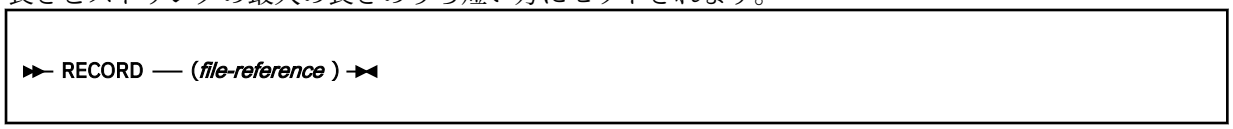
## RECORD 条件

**状況**
RECORD 条件は常に割り込み可能です。

**結果**
指定されたファイルの長さ接頭部が間違って伝送される可能性があります。

**原因と構文**
RECORD 入出力条件は、指定されたレコードが切り捨てられたときに起こります。 この条件は、READ、WRITE、LOCATE、または REWRITE 処理中にのみ起こる可能性があります。

SCALARVARYING オプションがファイルに適用されている場合は (位置指定モードを使用して可変長ストリングを伝送するときのファイルにはこのオプションを適用しなければならない)、2 バイトの長さの接頭部が可変長ストリング・エレメントと一緒に伝送されます。 RECORD 条件が起こったとき、長さ接頭部はリセットされません。 SCALARVARYING オプションがファイルに適用されていない場合は、長さ接頭部は伝送されません。 つまり、入力の場合、可変長ストリングの現在の長さは、レコードの長さとストリングの最大の長さのうち短い方にセットされます。




file-reference (ファイル参照) はスカラー参照でなければなりません。

**暗黙アクション**
メッセージが印刷され、ERROR 条件が起こります。

**正常な戻り**
RECORD 条件を引き起こしたステートメントのすぐあとのステートメントから実行が続けられます。

この条件の ON ユニット内でファイルをクローズした場合は、正常な戻りの結果がどうなるかはわかりません。 ファイルをクローズした ON ユニットから出るときは、GO TO ステートメントを使用しなければなりません。

**条件コード**
20-24

## SIZE 条件

**状況**
SIZE 条件は、SIZE 条件接頭語の有効範囲内を除き、プログラム全体にわたって割り込み禁止です。

**結果**
割り当ての結果は未定義です。

**原因と構文**
計算関係の条件である SIZE 条件は、変数や中間結果への割り当て時に、または入出力処理中に、高位の (つまり、左端の) 有効数字 (2 進数または 10 進数) が失われたときにのみ起こります。 データ・タイプ、基数、スケール、または精度が異なっているときの変換時にも有効数字が失われることがあります。 SIZE 条件が使用不可であっても、インラインで実行されない変換であれば、条件が起こる可能性があります。

SIZE 条件は、あるデータ項目に割り当てられているサイズの値が、(この値がそ の項目が占有しているストレージの実際のサイズではない場合も) そのデータ項目の 宣言されている (またはデフォルトの) サイズを超えるときに起こります。 例えば、精度 (20) の固定小数点 2 進数項目は、ストレージではフルワードを 占有しますが、FIXED BINARY (20) を超える大きさの値をこの項目に 割り当てようとすると SIZE 条件が起こります。

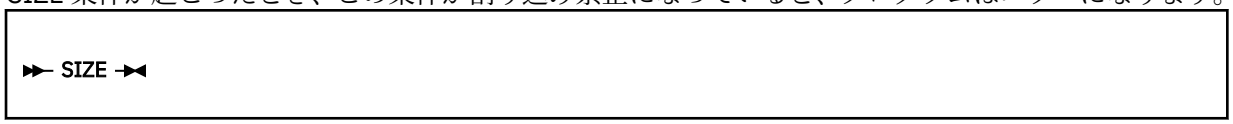
最適化されたコードでは、以下の動作を確認できます。

- SIZE の代わりに FOFL が発生することがあります。
- 固定小数点から固定 2 進値への 2 進数変換が関係していて、ソースが大きすぎる場合は、SIZE の代わりに ZDIV が発生する可能性があります。

SIZE 条件の検査が行われると、ストレージ・スペースと実行時間の両方の点でオーバ ーヘッドが必要です。したがって、この条件は主にプログラム・テストに使用 されます。 実稼働プログラムではこの条件を取り除くことができます。 テストと実稼働アプリケーション・プログラムについての詳細は、「プログラミング・ガイド」を参照してください。

SIZE 条件と FIXEDOVERFLOW 条件は次の点で異なっています。FIXEDOVERFLOW 条件は、 計算された固定小数点値の値が、システムの許可する最大値を超えるときに起こります。 値を割り当てるとき、その値の計算で FIXEDOVERFLOW 条件が起こったか 否かには関係なく、SIZE 条件が起こる可能性があります。

SIZE 条件が起こったとき、この条件が割り込み禁止になっていると、プログラムはエラーになります。




**暗黙アクション**
メッセージが印刷され、ERROR 条件が起こります。

**正常な戻り**
この条件が起こったすぐあとの位置に制御が戻されます。

**条件コード**
340, 341, 342, 343

## STORAGE 条件

**状況**
STORAGE は常に使用可能です。

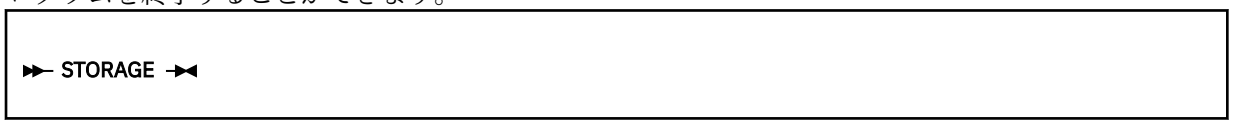
**結果**
結果は、この条件を起こしたストレージ割り振りの試みを示す変数のタイプによって異なります。

- 被制御変数の ALLOCATE ステートメントの後には、その変数の世代は 割り振られません。 その被制御変数を参照すると、ALLOCATE ステートメントの前にあるその変数の 世代 (ある場合) をアクセスすることになります。
- 基底付き変数の ALLOCATE ステートメントのあとには、その変数は割り振られ ないため、対応するポインターも未定義です。
- 基底付き変数の ALLOCATE 組み込み関数のあとには、その変数は割り振られない ため、対応するポインターの使用も未定義です。

**原因と構文**
STORAGE 条件が起こると、プログラムは、BASED または CONTROLLED のストレージを AREA の 外部に割り振ろうと試みる ALLOCATE 組み込み関数または ALLOCATE ステートメントが失敗した場合に制御を獲得します。 AREA 内の ALLOCATE ステートメントの失敗によって、AREA 条件が引き起こされます。

AUTOMATIC 組み込み関数が失敗しても、STORAGE 条件は起こりません。

STORAGE 条件の ON ユニットは、割り振り済みストレージの解放を試みることができます。 ON ユニットは、十分なストレージを解放できない場合、必要なステップを行い、診 断情報を失うことなくプログラムを終了することができます。




**暗黙アクション**
ERROR 条件が起こります。

**正常な戻り**
ERROR 条件が起こります。

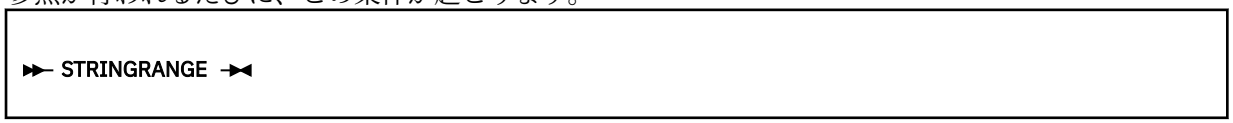
**条件コード**
450、451

# STRINGRANGE 条件

**状況**
STRINGRANGE 条件は、STRINGRANGE 条件接頭語の有効範囲内を除き、プログラム全体にわたって割り込み禁止です。

**Result**
指定された SUBSTR/SUBTO の値が変更されます。

**原因と構文**
STRINGRANGE プログラム・チェックアウト条件は、SUBSTR/SUBTO 参照への引数の値が、SUBSTR/SUBTO 組み込み関数について記述されている規則に従っていない場合に発生します。 無効な引数への参照が行われるたびに、この条件が起こります。




**省略形**
STRG

**暗黙アクション**
メッセージが印刷され、「正常な戻り」の説明どおりに処理が続けられます。

**正常な戻り**
値が以下のように定義された、改訂済み SUBSTR/SUBTO 参照を使用して実行が続行されます。

ソース・ストリングの長さ (ON ユニットが指定されている場合はその実行後) が *k*、開始点が *i*、サブストリングの長さが *j* であるとします。

- *i* が *k* より大きい場合、この値はヌル・ストリングです。
- *i* が *k* 以下の場合、この値は、ソース・ストリングの *m* 番目の文字、ビット、グラフィック、uchar、またはワイド文字で始まり、*n* 個の文字、ビット、グラフィック、uchar、またはワイド文字の長さとなるサブストリングです。*m* および *n* は以下のように定義されます。

```
M = max( I,1 )
```

```
N = max( 0,min( J + min(I,1) - 1,K - M + 1 ))
```

上記は、J を指定した場合です。

```
N = K - M + 1
```

上記は、J を指定しない場合です。

新しい引数は必ず範囲内にあります。

*i* と *j* の値は、ON ユニットに入る前に確立されます。 これらの値は、ON ユニットから戻ったときに計算し直されません。

SUBSTR/SUBTO の最初の引数が可変長ストリングの場合、*k* の値が ON ユニットで変更される可能性があります。 値 *n* は、ON ユニットから戻る際に *k* の新しい値を使用して再計算されます。

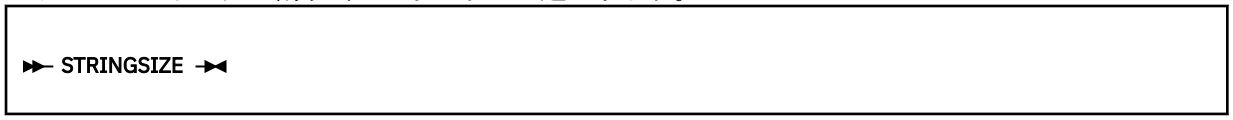
**条件コード**
350

# STRINGSIZE 条件

**状況**
STRINGSIZE 条件は、STRINGSIZE 条件接頭語の有効範囲内を除き、プログ ラム全体にわたって割り込み禁止です。

**Result**
この条件の処置が取られたあと、切り捨てられたストリングがそのターゲット・ストリングに割り当てられます。 ソース・ストリングがターゲット・ストリング内に収まるように、ソース・ストリングの右端の文字、ビット、uchar、ワイド文字、またはグラフィックが切り捨てられています。

**原因と構文**
STRINGSIZE プログラム・チェックアウト条件は、あるストリングを、それより短い最大長を持つターゲット・ストリングに割り当てようとすると起こります。




**省略形**
STRZ

**暗黙アクション**
メッセージが印刷され、処理が続けられます。

**正常な戻り**
この条件が起こった位置から実行が続けられます。

**条件コード**
150、151

## SUBSCRIPTRANGE 条件

**状況**
SUBSCRIPTRANGE 条件は、SUBSCRIPTRANGE 条件接頭語の有効範囲内を除き、 プログラム全体にわたって割り込み禁止です。

**結果**
SUBSCRIPTRANGE 条件が起こった場合、無効な添え字の値は未定義のため、どのような参照になるかわかりません。

**原因と構文**
SUBSCRIPTRANGE プログラム・チェックアウト条件は、計算後の添え字が指定された限界内に存在しないことが検出されたときに起こります。 SUBSCRIPTRANGE 条件の発生とその他の添え字の計算がどのような順序で行われるかはわかりません。

SUBSCRIPTRANGE 条件が発生すると、ONSUBSCRIPT、ONLBOUND、および ONHBOUND 組み込み関数は、無効な配列添字の値、索引付き配列次元の下限、および索引付き配列次元の上限を戻します。

**省略形**
SUBRG

**暗黙アクション**
メッセージが印刷され、ERROR 条件が起こります。

**正常な戻り**
SUBSCRIPTRANGE の ON ユニットから正常に戻ると、ERROR 条件が起こります。

**条件コード**
520, 521

## TRANSMIT 条件

**状況**
TRANSMIT 条件は常に割り込み可能です。

**Result**
TRANSMIT 条件が起こると、伝送されたデータのいずれかが正しくない可能性があります。

**原因と構文**

TRANSMIT 入出力条件は、どの入出力処理時にも起こる可能性があります。 この条件は、レコード (ブロック化されていればブロック) の訂正不能な伝送エ ラーによって引き起こされます。この訂正不能な伝送エラーとは、実行中に訂 正できなかった入出力エラーのことです。 記録メディアが損傷していたり、指定やセットアップが正しくないような場合、 このエラーが起こることがあります。

入力の場合は、潜在的に不正なレコードが伝送されたあとで、TRANSMIT 条件が起こります。 レコードがブロック化されているときは、そのブロック内の後続の 各レコードごとに TRANSMIT 条件が起こります。

出力の場合は、伝送後に TRANSMIT 条件が起こります。 レコードがブロック化されているときの伝送は、それぞれの出力ステートメントが実行されたときではなく、ブロックがいっぱいになったときに行われます。

スパン・レコードの更新処理時には、レコードの最後のセグメントでのみ TRANSMIT 条件が起こります。 同一ブロック内の後続のレコードではこの条件は起こりませんが、それらの レコードの保全性は保証できません。

file-reference (ファイル参照) はスカラー参照でなければなりません。

**暗黙アクション**

メッセージが印刷され、ERROR 条件が起こります。

**正常な戻り**

エラーが起こらなかった場合と同様に処理が続けられます。したがって、TRANSMIT 条件を引き起こしたステートメントまたはデータ項目によって、別の条件 (例えば、RECORD 条件) が引き起こされることがあります。

この条件の ON ユニット内でファイルをクローズした場合は、正常な戻りの結果がどうなるかはわかりません。 ファイルをクローズした ON ユニットから出るときは、GO TO ステートメントを使用しなければなりません。

**条件コード**

40-46

## UNDEFINEDFILE 条件

**状況**

UNDEFINEDFILE 条件は常に割り込み可能です。

**Result**

指定されたファイルは、アプリケーション・プログラムに対して未定義です。

**原因と構文**

UNDEFINEDFILE 入出力条件は、ファイルのオープンに失敗するたびに起こります。 複数のファイルを指定している OPEN ステートメントによってオープンしよう とした場合、指定されているすべてのファイルのオープンが試みられたあとで、 この条件が起こります。

同じ OPEN ステートメント内の複数のファイルについて UNDEFINEDFILE 条件が起こった場合は、その OPEN ステートメントにファイル名が記述されている順に (右から左へ)、ON ユニットが実行されます。

データ伝送ステートメントで、ファイルが暗黙にオープンされるときにこの条 件が起こった場合は、ON ユニットから正常に戻ると、そのデータ伝送ステート メントの残りの部分から処理が続けられます。 ON ユニット内でファイルがオープンされなかった場合は、そのステートメントは処理を続けられず、ERROR 条件が起こります。

UNDEFINEDFILE 条件は、属性の指定が矛盾している場合 (例えば、 DIRECT と PRINT をともに指定する) に起こるだけでなく、次の場合にも起こります。

- ブロック・サイズがレコード・サイズより小さいとき (ただし、スパン・レコードの場合は除く)。
- LINESIZE が許可されている最大値を超えているとき。

- INDEXED データ・セットの作成時に、KEYLENGTH がゼロであるか、または指定されていないとき。
- INDEXED データ・セットの場合に、KEYLENGTH + KEYLOC の値が レコード長を超えるような KEYLOC オプションを指定したとき。
- STREAM データ・セットの場合に、V フォーマットの論理レコードの長さとして 18 バイト未満の値を指定したとき。
- FB フォーマット・レコードのブロック・サイズとして、レコード・サイズの整数倍でない値を指定したとき。
- VB フォーマット・レコードの場合に、指定した論理レコードの長さが、指定したブロック・サイズより少なくとも 4 バイト小さい値になっていないとき。

file-reference (ファイル参照) はスカラー参照でなければなりません。

**省略形**
UNDF

**暗黙アクション**
メッセージが印刷され、ERROR 条件が起こります。

**正常な戻り**
最後の ON ユニットが正常に完了すると、この条件を引き起こしたステートメントのすぐあとのステートメントに制御が移されます。

**条件コード**
80-89、91-95

## UNDERFLOW 条件

**状況**
UNDERFLOW 条件は、NOUNDERFLOW 条件接頭語の有効範囲を除き、プログラ ム全体にわたって割り込み可能です。

**結果**
結果が未定義である場合は、z/OS 上での IEEE 浮動小数点を除いて、無効な浮動小数点数値が 0 にセットされます。

**原因と構文**
計算関係の条件である UNDERFLOW 条件は、浮動小数点数の大きさが、許可されている最小値より小さいときに起こります。 等しい数値が減算されたとき (有効桁エラーと呼ばれます)、UNDERFLOW は発生しません。

式 $\mathbf{X}^{(-Y)}$ (ここで、Y>0) は、$\mathbf{X}^{Y}$ の逆数を取ることによって評価できます。したがって、UNDERFLOW 条件の代わりに OVERFLOW 条件が発生することがあります。

**省略形**
UFL

**暗黙アクション**
z/OS 上で IEEE 浮動小数点式を (2 進数または 10 進数で) 評価中に例外が発生しない限り、メッセージが印刷され、条件が発生した位置から実行が続行されます。 z/OS 上で IEEE 浮動小数点例外を評価中に例外が発生した場合は、メッセージが印刷され、ERROR 条件が発生します。

**正常な戻り**
z/OS 上で IEEE 浮動小数点式を (2 進数または 10 進数で) 評価中に例外が発生しない限り、条件が発生した位置の直後の位置に制御が戻されます。 z/OS 上で IEEE 浮動小数点例外を評価中に例外が発生した場合は、ERROR 条件が発生します。

**条件コード**
330

## ZERODIVIDE 条件

**状況**
ZERODIVIDE 条件は、NOZERODIVIDE 条件接頭語の有効範囲内を除き、プログラム全体にわたって割り込み可能です。

**結果**
ゼロによる除算の結果は未定義です。

**原因と構文**
計算関係の条件である ZERODIVIDE 条件は、ゼロによる除算を試みたときに 起こります。 この条件は、固定小数点の除算でも浮動小数点の除算でも起こります。 浮動小数点の除算で割られる数もゼロである場合は、INVALIDOP 条件が起こります。

ZERODIVIDE 条件が起こったとき、この条件が割り込み禁止になっていると、プログラムはエラーになります。

一部の固定小数点の高い精度での除算は、Decimal Floating-Point (DFP) 機能を使用して実行できます。これにより、一部の ZERODIVIDE 例外は INVALIDOP として報告される可能性があります。

一部の固定 10 進数から固定 2 進数への変換では、ソースが大きすぎる場合に ZERODIVIDE が発生する可能性があります。

これは、従来、これらのタイプの変換は CVB または CVBG 命令で行われ、ソースが大きすぎると CVB または CVBG 命令によって ZDIV が発生するためです。

**省略形**
ZDIV

**暗黙アクション**
メッセージが印刷され、ERROR 条件が起こります。

**正常な戻り**
ERROR 条件が起こります。

**条件コード**
320

**関連情報**
356 ページの『INVALIDOP 条件』

# 第 17 章 マルチスレッド化機能

PL/I のマルチスレッド機能によって、PL/I プログラムの各部分の実行が複数の スレッドを使用して非同期に行えるようになります。

PL/I のプログラムは、1 つまたは複数のプロシージャーからできています。 プログラムは、普通は単一の実行単位として実行されるか、単一の実行単位の一部として実行されます。 プロシージャーが別のプロシージャーを呼び出す場合、制御は呼び出されたプ ロシージャーに渡され、呼び出した側のプロシージャーの実行は、呼び出され たプロシージャーが制御を戻すまで中断されます。 このような単一の制御フローをともなう実行は、同期 フローといいます。

PL/I のマルチスレッド機能を使用すると、呼び出す側のプロシージャーから、 呼び出されたプロシージャーに制御が解放されません。 その代わりに、さらに追加の制御フローが確立され、両方のプロシージャーが並行して実行されます。 このような並行プロシージャーの実行は、非同期 フローといいます。

PL/I マルチスレッド化機能により、PL/I プログラムの各部分を複数のスレッドで非同期に実行できます。スレッド は、コンピューター・システムのリソースを獲得するために競合する作業単位です。 スレッドは、OS PL/I のタスクと同等ではありません。 プログラム内のメイン・スレッドを除き、スレッドは常にプログラム内の他のスレッドから独立しており、無関係です。 あるプロシージャーが別のプロシージャーをスレッドとして呼び出す場合、 この処置は、スレッドの接続 または作成といいます。

1 つまたは複数のスレッドの実行が、1 つの**プロセス**内で起こること を PL/I プログラムと考えることができます。 PL/I は、複数のプロセスまたはタスクを作成したり管理したりする機能を持っ ていませんが、単一のプログラム (プロセス) において複数のスレッドを作成し たり管理したりすることはできます。

通常は、プロセスごとに 1 つのアプリケーション・スレッドがあります。 以下の目的で、複数のスレッドを接続 (作成) できます。

- より短い経過時間内に作業の一部を行う。 そのようなアプリケーションはユーザーと対話を行わないバッチ・アプリケーションにすることが できます。 例えば、あるプロシージャーは、 PL/I プログラムをコンパイルするスレッドを接続する場合があります。
- 1 つのプログラムの頻繁な応答部分を 1 つ目のスレッドが分担し、入出力を別の スレッドが分担し、通常の応答部分を 3 つ目のスレッドが分担して実行する。
- 場合によっては活動停止中の計算機システムのリソースを使用する。 このようなリソースには、入出力装置ばかりでなく CPU も含まれることがあります。
- 応答時間が問題になるリアルタイムのマルチユーザー・アプリケーションを導入する。
- 信頼性を確保するために作業の一部を単独に分離する。 つまり、独立しているほかの各部分が処理されている一方で、ジョブの一部に起こった障害を 分離できます。

**注 :** オペレーティング・システムのサービスは、PL/I が適切な機能を提供する場合には、直接に使用する必要はありません。

## スレッドの作成

スレッドは、コンピューター・システムのリソースを獲得するために 競合する作業単位です。 あるプロシージャーが別のプロシージャーをスレッドとして呼び出す場合、 この処置は、スレッドの接続または作成といいます。

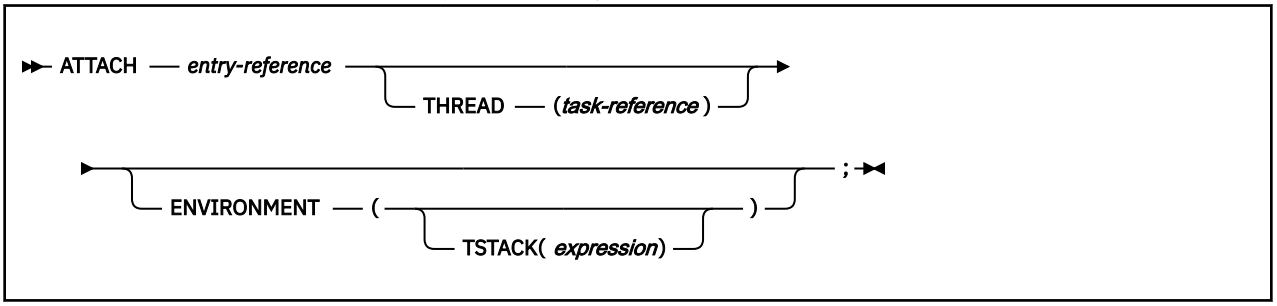
- スレッドは、独立した作業単位です。
- スレッドは、プロセスやシステム内において他のスレッドと並行的にかつそれらから独立して実行されます。
- スレッドは他のスレッドに接続できます。
- スレッドはスレッドの完了を待機できます。
- スレッドは、自分自身または他のスレッドを停止できます。

スレッドによって同期的に呼び出されたプロシージャーや関数は、そのスレッドの実行の一部となります。

© Copyright IBM Corp. 1999, 2022

# ATTACH ステートメント

スレッドの接続 (または作成) は、ATTACH ステートメントを実行して行います。 デフォルトを使用しない場合は、スレッドの特性を明示的に指定できます。




**entry**
LIMITED 入り口変数の名前、または外部の入り口名やレベル 1 のプロシージャー名を指定します。 内部プロシージャーや取り出し可能プロシージャーの名前は指定できません。 ATTACH された入り口は、パラメーターを持たない、または厳密に 1 つの BYVALUE POINTER パラメーターを 持つものとして宣言されなければなりません。 ただし、プロシージャーに LIMITED 入り口変数を割り当て、その入り口変数をスレッ ドとして接続すれば、プロシージャーを取り出すことはできます。

新しいスレッドに引数を渡すことができます。これはちょうど CALL ステートメントで同期した入り口に引数を渡すのと同様です。

**THREAD (task reference)**
スレッドと関連づけられるタスク変数の名前を指定します。 タスク変数は、名前を付ければスレッドを参照するために使用できます。

明示的に宣言しない限り、コンテキスト宣言で変数に名前が付与されます。

THREAD オプションを指定しない場合、接続されたスレッドを、別のスレッドに よって停止させたり待機させたりすることができません。

スレッドが THREAD オプションを使用して接続された場合、そのスレッドに関連付けられたシステム・リソースをすべて解放するには、DETACH ステートメントを使用してスレッドを切り離す必要があります。

スレッドの作成にオペレーティング・システムのサービスを直接に使用することはできません。

**ENVIRONMENT (省略形: ENV)**
環境的な特性を指定します。 通常はオペレーティング・システムに依存します。

**TSTACK (expression)**
Intel 上では、接続されるスレッドで使用されるスタックのサイズを指定します。 式は FIXED BINARY(31,0) にする必要があります。 スタック・サイズを指定しないと、実行時のデフォルトが使用されます。

z/OS 上では、TSTACK は無視され、スタックのサイズは LE に よって決定されます。

付加されたプロシージャーは、サポートされる任意のリンケージを持つことができます。

### 例

```
attach input (File1);

attach input (File2)
   thread (T2);
```

### 関連情報

370 ページの『スレッドの切り離し』
DETACH ステートメントは、THREAD オプションを使用して接続されたスレッドに 関連付けられているシステム・リソースを解放するために使用します。

## スレッドの終了

END、EXIT、および STOP の各ステートメントを実行すると、スレッドが終了する場合があります。 ERROR 条件が発生した場合にも、スレッドが終了することがあります。

スレッドは、以下のいずれかの状況が発生すると終了します。

- 先頭のプロシージャー (当該スレッド内の初期プロシージャー) に対応している END ステートメントに達する場合。
- ERROR 条件が発生し、ERROR ON ユニットが存在しないか、あるいは ERROR ON ユニット が正常に終了する場合 (その ON ユニットの END ステートメントに達するか、 RESIGNAL ステートメントを実行する場合)。
- 当該スレッド内のいずれかのプロシージャーで EXIT ステートメントが実行される場合。
- プログラム内の開始スレッドが終了する場合。
- そのプログラムのいずれかのスレッドで STOP ステートメントが実行される場合。 これはプログラム全体を停止し、主スレッドを含めてすべてのスレッドを終了させます。

FINISH 条件は、プログラム終了を引き起こしたスレッド内でのみ発生します。 スレッドが実際に終了する前に そのスレッド内に確立されているいずれかの ON ユニットに制御が渡されます。

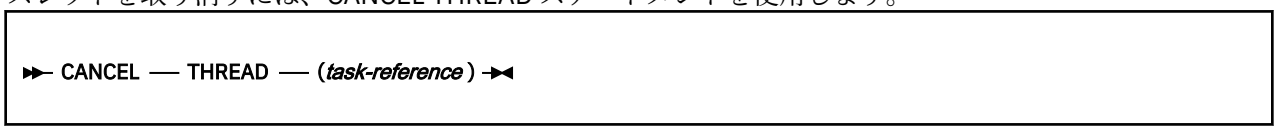
前述の場合を除き、あるスレッドが終了するとき、終了するそのスレッドが 主スレッドでない限り、他のスレッドは終了しません。 主スレッドが終了する場合は、それが終了する前に、他のすべてのスレッドが停止し終了します。

あるスレッドが終了すると、そのスレッドのスタック空間のみが解放されます。 割り振られたストレージや開いているファイルなど、他のすべてのリソースはそのまま残ります。 スレッドが必要としたなんらかのリソースが解放され、オープンしたファイルがク ローズされたことをユーザーは確認しなければなりません。ただし、それらが まだアクティブの他のスレッドで必要であるならば別です。

主スレッドが終了すると、すべてのリソースが解放され、ファイルがクローズされます。

## スレッドの取り消し

スレッドを取り消すには、CANCEL THREAD ステートメントを使用します。



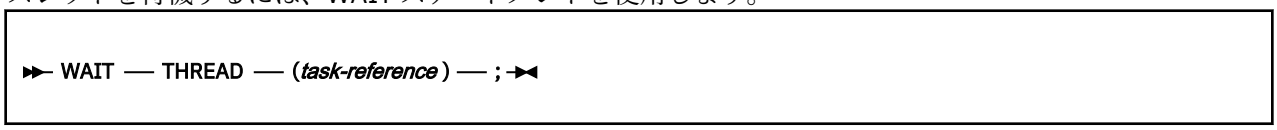

**THREAD *task-reference***
THREAD オプションには、プロセスを取り消すスレッドのタスク変数 *task-reference* を指定します。 タスク変数は、スレッドの ATTACH 実行中に生成されます。

次の例では、スレッド TI1 が取り消されます。

```
CANCEL THREAD (TI1);
```

## スレッドの完了の待機

スレッドを待機するには、WAIT ステートメントを使用します。

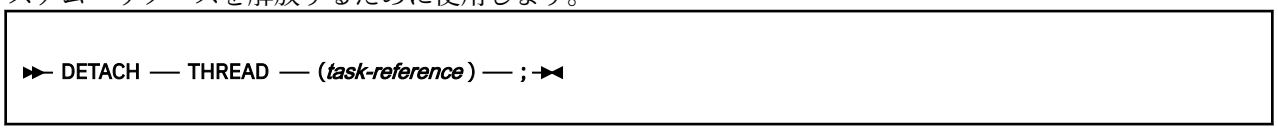


**THREAD (task-reference)**
THREAD オプションには、プロセスが待機しているスレッドを指定します。 現行のスレッドは、指定したスレッドが終了するまで中断されます。 指定したスレッドが終了すると、すぐに現行のスレッドが再開します。

```
WAIT THREAD (TI1);
```

## スレッドの切り離し

DETACH ステートメントは、THREAD オプションを使用して接続されたスレッドに 関連付けられているシステム・リソースを解放するために使用します。




**THREAD (task-reference)**
THREAD オプションには、切り離すスレッドを指定します。

通常このステートメントは、終了するスレッドの WAIT ステートメントの直後に実行します。

## 条件処理

新しいスレッドが作成されたとき、ON ユニットが確立されることを想定しているわけではありません。 スレッドが作成されたときに有効である ON ユニットは、新しく接続されたスレッドによって継承はされません。 あるスレッド内で起こった条件は、そのスレッド内で処理されるだけで、スレッド境界を超えて処理されることはありません。

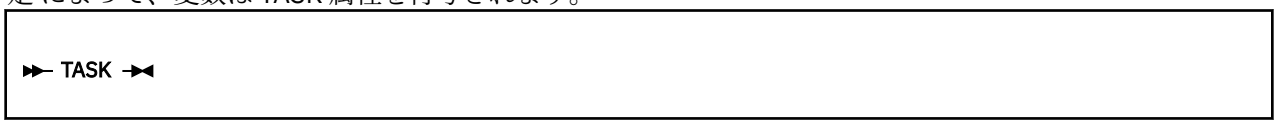
例えば、スレッド **A** がファイル **F** を開き、次に **A** がスレッド **T** を作成し、続いて **T** が ENDFILE 条件を発生させるとします。 スレッド **T** 自体で ON ENDFILE 条件が確立されていない場合は、**T** で ERROR 条件が発生し、通常の条件処理がすべてスレッド **T** 内で行われます。 **A** が ENDFILE または ERROR の ON ユニットを確立したかどうかは、スレッド **T** の実行に影響しません。

スレッドは、条件処理の必要がある場合には、該当する条件に対応した ON ユニットを確立しておかなければなりません。 スレッドを超えて条件を知らせるメカニズムはありません。

ATTENTION 条件を起こすために CTRL-BREAK を使用すると、 その ATTENTION 条件は、ATTACH ステートメントによって 作成されたどのスレッドでもなく、主スレッドの中でのみ起こります。

## タスク・データとタスク属性

タスク変数は、スレッド ID、サービス・カテゴリー、および ディスパッチング優先順位などのスレッド関連情報を保持しています。 明示的な宣言、または THREAD オプションの中に示されたことによる暗黙の指定 によって、変数は TASK 属性を付与されます。




タスク変数とスレッドとの関連付けは、そのスレッドを作成する ATTACH ステートメントの THREAD オプションで タスク参照を指定することによって行われます。 タスク変数は、それが関連付けられているスレッドがアクティブであれば、アクティブです。 タスク変数は、それがスレッドと関連づけられる前に割り振られていなければ ならず、関連づけられたスレッドがアクティブである間は解放できません。 アクティブなタスク変数は、別のスレッドと関連づけることはできません。

## THREADID 組み込み関数

THREADID (スレッド ID の短縮形) は、POINTER 値を戻します。この値は、接続されたスレッドのオペレーティング・システム・スレッド ID のアドレスです。

この組み込み関数によって使用される値は、DosSetPriority のようなシステム機 能に対するパラメーターとして使用することができますが、DosKillThread への パラメーターとして使用してはなりません。

**x**
タスク参照。 *x* の値は、ATTACH ステートメントの THREAD オプションですでに設定されているはずです。

## スレッド間のデータの共用

すべての静的データと被制御データは、スレッドの間で共用されます。 その他のデータもすべて、そのデータが割り振られており、すべてのスレッドでデータの使用が終了するまで解放されない限り、引数/パラメーターを使用して、また基底付き参照を使用して共用できます。

例えば、接続する側のスレッドの持つ自動変数は、 接続される側のスレッドと共用され、接続する側のブロックは、 接続される側のスレッドが共用変数の使用を終わりにするまで 終了することはできません。

データの逐次化に関しては、ユーザーが責任を持たねばなりません。 被制御データの新しい世代が割り振られたり、既存の世代が解放される場合、 一部のスレッドがより古い世代やもはや存在しない世代に依然としてアクセス していることがあります。 このような場合、予期せぬ結果を引き起こします。

すべての割り振られたストレージは、明示的に解放しない限り、プログラムが終了するまで解放されません。

PL/I は、AREA 変数内の ALLOCATE や FREE は逐次化しません。

## スレッド間のファイルの共用

すべてのファイルがスレッド間で共用されます。

スレッド (MAIN 以外) がファイルをオープンする場合、そのファイルは、そのスレッドの終了前にクローズする必要があります。

MAIN スレッドでオープンされたファイルは、明示的にクローズされるか、または、プログラムが終了するまでクローズされません。 z/OS での言語環境プログラム・メッセージ・ファイルを除き、ファイル使用をシリアライズしない場合の結果は、異常終了を含めた予測不能なものになるおそれがあります。

逐次化に関しては、ユーザーが責任を持たねばなりません。 371 ページの『スレッド間のデータの共用』を参照してください。

メッセージ・ファイルと表示ステートメントは、 PL/I によって自動的に逐次化されます。

# 第 18 章 組み込み関数、疑似変数、およびサブルーチン

多くの共通タスクが組み込み関数、サブルーチン、および疑似変数の形で使用可能です。 これらの共通タスクを使用することにより、作成するコードの数が減り、信頼性の高い迅速な処理が できます。 この章では、PL/I プログラムで使用できる組み込み関数、サブルーチン、および疑似変数について説明します。

PL/I では、31 ビットと 64 ビットの両方のアプリケーションを開発することができます。 64 ビット・アプリケーションを開発する場合は、一部の組み込み関数の引数および戻りの型は 31 ビットの場合とは異なることに注意してください。 これらの引数および戻り値の型は *size_t* です。 このような *size_t* 引数は、ストレージの一部のサイズを表しています。

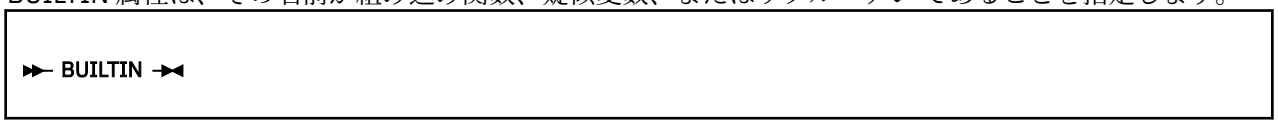
LP(32) コンパイラー・オプションが有効な場合、*size_t* は FIXED BIN(31) です。 LP(64) コンパイラー・オプションが有効な場合、*size_t* は FIXED BIN(63) です。

## 組み込み関数、疑似変数、および組み込みサブルーチンの宣言および呼び出し

組み込み関数、疑似変数、およびサブルーチンは、内容に従って宣言するか明示 的に宣言することができます。

### BUILTIN 属性

BUILTIN 属性は、その名前が組み込み関数、疑似変数、またはサブルーチン であることを指定します。




組み込まれた名前は、プログラマー定義の名前として使用できます。 プログラマー定義の宣言または同じ名前の使用を、含まれているブロックから継 承している、どのブロック内でも、組み込まれた名前に対して BUILTIN を宣言で きます。

#### 例

この例は、BUILTIN 属性を持つ組み込み名を示しています。

```
1    A: procedure;
        declare Sqrt float binary;
2       X = Sqrt;

3      B:  Begin;
          Declare Sqrt builtin;
          Z = Sqrt(P);
        end B;

       end A;
```

**1**
Sqrt は、プログラマー定義の名前です。

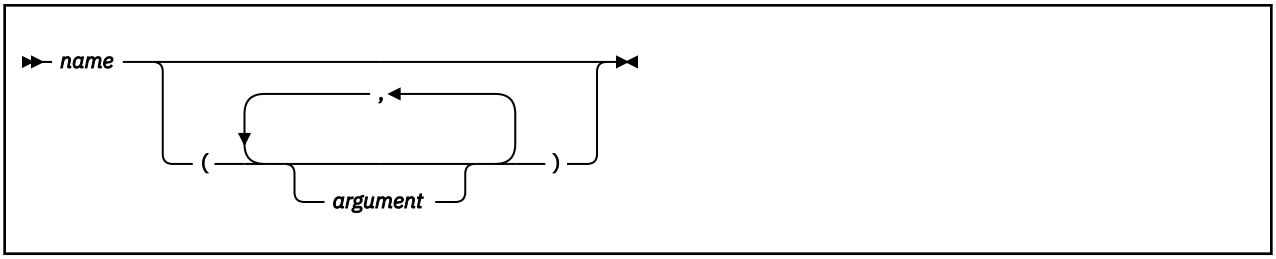
**2**
変数 X への割り当ては、プログラマー定義の名前 Sqrt への参照です。

**3**
Sqrt は、B 内の Sqrt への参照が組み 込み関数への参照で、**1** で宣言されたプログラマー定義の名前 Sqrt でないことが認識されるように、BUILTIN 属性によって宣言 されます。

### 組み込み関数および疑似変数の呼び出し

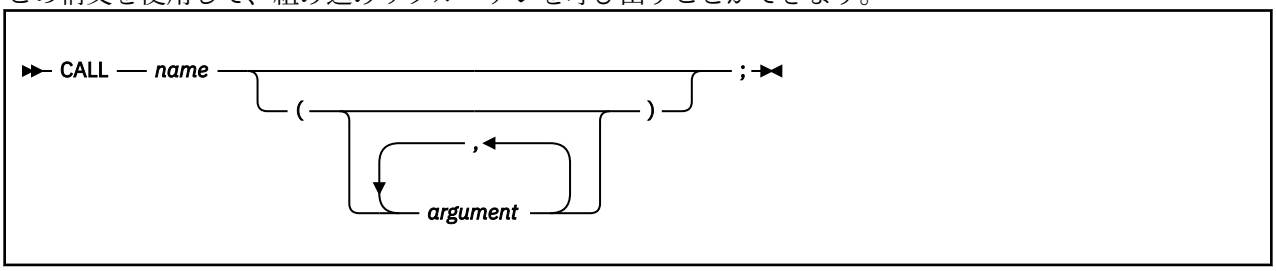
この構文を使用して、組み込み関数および疑似変数を呼び出すことができます。

© Copyright IBM Corp. 1999, 2022




## 組み込みサブルーチンの呼び出し

この構文を使用して、組み込みサブルーチンを呼び出すことができます。




# 組み込み関数、疑似変数、および組み込みサブルーチンの引数の指定

引数は式で表すことができ、評価、およびデータ変換規則に従って組み込み関数に 適したデータ・タイプに変換されます。

## 引数の集合

引数を持つすべての組み込み関数および疑似変数は、配列引数を持つことが できます (複数の引数が配列の場合は、境界は同一でなければなりません)。

- ADDR、ALLOCATION、CURRENTSIZE、SIZE、STRING、 および配列処理組み込み関数は、エレメント値を戻します。
- コンパイラー・オプション USAGE(UNSPEC(ANS)) の使用時には、UNSPEC は要素値を返します。USAGE(UNSPEC(IBM)) の使用時には、値の配列を返します。
- その他のすべての関数は、値の配列を戻します。

配列引数の指定は、1 つ以上の引数が制御変数によって修正される添え字付き配列参照である DO グループに、関数参照または疑似変数を指定するのと同等です。

次の例を考えてください。

```
dcl A(2) char(2) varying;
dcl B(2) char(2)
   init('AB','CD');
dcl C(2) fixed bin
   init(1,2);
A=substr(B,1,C);
```

この例の結果、A(1) には値 A が、A(2) には値 CD が入ります。

構造体または共用体の引数を受け入れることができる組み込み関数および疑似変数は、374 ページの表 59 にリストされています。

表 59. 構造体または共用体の引数を受け入れることができる組み込み関数および疑似変数

| 組み込み関数および疑似変数 | | | |
|---|---|---|---|
| ADDR | JSONGETCOMMA | JSONPUTOBJECTEND | PRESENT |
| ADDRDATA | JSONGETMEMBER | JSONPUTOBJECTSTART | SIZE |

表 59. 構造体または共用体の引数を受け入れることができる組み込み関数および疑似変数 (続き)

| 組み込み関数および疑似変数 | | | |
|---|---|---|---|
| ALLOCATION | JSONGETOBJECTEND | JSONPUTVALUE | STRING |
| ALLCOMPARE | JSONGETOBJECTSTART | JSONVALID | UNSPEC[1] |
| BITLOCATION | JSONGETVALUE | LOCATION | XMLCHAR |
| CURRENTSIZE | JSONPUTARRAYEND | OMITTED | XMLUCHAR |
| INDICATORS | JSONPUTARRAYSTART | PLISAXA | |
| JSONGETARRAYEND | JSONPUTCOLON | PLISAXB | |
| JSONGETARRAYSTART | JSONPUTCOMMA | PLISAXC | |
| JSONGETCOLON | JSONPUTMEMBER | PLISAXD | |

1. UNSPEC を構造体または共用体に適用できるのは、コンパイラー・オプション USAGE(UNSPEC(ANS)) が有効な場合のみです。

## ヌル引数およびオプション引数

組み込み関数には、引数が必要ないものもあります。 これらは、BUILTIN 属性を使用して明示的に宣言するか、または参照に ONCHAR() などのヌル引数リストを組み込んでコンテキスト上で宣言する必要があります。それ以外の場合、名前は組み込みとして認識されません。

# 数学関数の正確度

結果の正確度は、引数の正確度とアルゴリズムの正確度という、2 つの要因に影響を受けます。

ほとんどの引数には、エラーが含まれています。 与えられた引数のエラーは、関数を評価する前に、いくつかのステップで累積 されていることがあります。 入力変換による新しいデータにもエラーが含まれていることがあります。 結果の精度に対する引数のエラーの影響は、結果を計算するアルゴリズムではなく、すべて数学関数の性質によって決まります。 本書では、このタイプの引数エラーは、説明しません。

インライン機械命令を使って組み込まれた数学組み込み関数では、異なる精度の 結果が生成されます。

# 組み込み関数のカテゴリー

このセクションでは、組み込み関数、サブルーチン、および疑似変数をカテゴリー別に示しています。

これらの表には、完全な関数名だけがリストされています。 既存の省略形は、各組み込み関数、サブルーチン、および疑似変数の説明の節に 記述されています。

以下の変換の説明では、*M* および *N* は最大精度を指定しています。

- *M* は FIXED BINARY の最大精度です。 これは、コンパイラー・オプション LIMITS(FIXEDBIN(M1,M2)) からの値 M2 です。
- *N* は FIXED DECIMAL の最大精度です。 これは、コンパイラー・オプション LIMITS(FIXEDDEC(N1,N2)) からの値 N2 です。

## 算術組み込み関数

算術組み込み関数を使用すると、算術値のプロパティー (例えば、SIGN 関数は算術変数の符号を示します) を判別でき、ルーチン算術演算を実行できます。

376 ページの表 60 には、算術組み込み関数とそれぞれの簡単な説明をリ ストしています。

算術関数には、1 つまたは複数の引数から結果のデータ・タイプを派生するも のもあります。 引数のデータ・タイプが異なるときは、73 ページの『第 4 章 データ変換』で説明されている ように変換されます。

表 *60.* 算術組み込み関数

| 機能 | 説明 |
|---|---|
| ABS | 値の絶対値を計算する。 |
| CEIL | ある値より大きいか等しい最小の整数を計算する。 |
| COMPLEX | 与えられた実数および虚数部分によって複素数を戻す。 |
| CONJG | ある値の共役複素数を戻す。 |
| FLOOR | ある値より小さいか等しい最大の整数を計算する。 |
| IMAG | 複素数の虚数部分を戻す。 |
| MAX | 2 つまたはそれ以上の値の最大値を計算する。 |
| MAXVAL | 数値オペランドの最大値を返す。 |
| MIN | 2 つまたはそれ以上の値の最小値を計算する。 |
| MINVAL | 数値オペランドの最小値を返す。 |
| MOD | ある値で別の値を除算したときの剰余のモジュラー等価を戻す。 |
| RANDOM | 類似ランダム値を戻す。 |
| REAL | 複素数の実数部分を戻す。 |
| REM | ある値を別の値で除算したときの剰余を計算する。 |
| ROUND | 指定された桁で値を丸める。 |
| ROUNDAWAYFROMZERO | 指定された桁で 10 進数値を丸める。 |
| ROUNDTOEVEN | 最も近い偶数値に丸めた値を返す。 |
| SIGN | 値が負、ゼロ、または正の場合に、それぞれ -1, 0 または 1 を戻す。 |
| TRUNC | ある値を切り捨ててできる整数を計算する。 |

## 配列処理組み込み関数

配列処理組み込み関数は、配列引数を演算して、1 つのエレメント値を戻します。

次の関数に必要な引数の変換は、関数の説明に示されています。 376 ページの表 61 には、配列処理組み込み関数がリストされています。

表 *61.* 配列処理組み込み関数

| 機能 | 説明 |
|---|---|
| ALL | 配列のすべてのエレメントのビット単位の"論理積"を計算します |
| ANY | 配列のすべてのエレメントのビット単位の"論理和"を計算します |
| DIMENSION | 配列のある次元のエレメントの数を戻す。 |
| HBOUND | 配列の次元の上限を戻す。 |
| HBOUNDACROSS | DIMACROSS 配列の上限を戻す |
| INARRAY | 式が配列のいずれかのエレメントと等しいかどうかを示す BIT(1) 値を返す。 |
| LBOUND | 配列の次元の下限を戻す。 |
| LBOUNDACROSS | DIMACROSS 配列の下限を戻す |

表 *61.* 配列処理組み込み関数 *(続き)*

| 機能 | 説明 |
|---|---|
| POLY | 2 つの配列の浮動小数点近似値を戻す。 |
| PROD | 配列のすべてのエレメントの積を計算する。 |
| QUICKSORT | 単純比較を使用してクイック・ソートを実行する。 |
| QUICKSORTX | 指定された比較関数を使用した配列のクイック・ソートを実行する。 |
| SUM | 配列のすべてのエレメントの和を計算する。 |

## バッファー管理組み込み関数

バッファー管理組み込み関数は、「バッファー」(アドレスおよびバイト数で指定するストレージ域) で演算を行います。

また、PLIFILL、PLIMOVE、および PLIOVER 組み込みサブルーチンは、バッファーの管理に役立ちます。377 ページの表 62 には、バッファー管理組み込み関数がリストされています。

表 *62.* バッファー管理組み込み関数

| 機能 | 説明 |
|---|---|
| BASE64DECODE | CHARACTER としてエンコードされる Base64 からソース・バッファーをデコードする。ターゲット・バッファーに書き込まれたバイト数を示す *size_t* 値を返す。 |
| BASE64DECODE8 | UTF-8 としてエンコードされる Base64 からソース・バッファーをデコードする。 ターゲット・バッファーに書き込まれたバイト数を示す *size_t* 値を返す。 |
| BASE64DECODE16 | UTF-16 としてエンコードされる Base64 からソース・バッファーをデコードする。ターゲット・バッファーに書き込まれたバイト数を示す *size_t* 値を返す。 |
| BASE64ENCODE | CHARACTER としてエンコードされる Base64 にソース・バッファーをエンコードする。ターゲット・バッファーに書き込まれたバイト数を示す *size_t* 値を返す。 |
| BASE64ENCODE8 | UTF-8 としてエンコードされる Base64 にソース・バッファーをエンコードする。ターゲット・バッファーに書き込まれたバイト数を示す *size_t* 値を返す。 |
| BASE64ENCODE16 | UTF-16 としてエンコードされる Base64 にソース・バッファーをエンコードする。ターゲット・バッファーに書き込まれたバイト数を示す *size_t* 値を返す。 |
| CHECKSUM | 指定したバッファーのチェックサム値を返す。 |
| COMPARE | 2 つのバッファーを比較し、最初のバッファーが 2 番目のバッファーより小さいか、等しいか、または大きいかを示す値を返します。 |
| HEXDECODE | DEFAULT コンパイラー・オプションの ASCII/EBCDIC サブオプションで指定された文字セットでエンコードされた基数 16 からソース・バッファーをデコードする。ターゲット・バッファーに書き込まれたバイト数を示す *size_t* 値を返す。 |
| HEXDECODE8 | UTF-8 でエンコードされた基数 16 からソース・バッファーをデコードする。ターゲット・バッファーに書き込まれたバイト数を示す *size_t* 値を返す。 |
| HEXIMAGE | バッファーの 16 進表記である文字ストリングを返す。 |

表 62. バッファー管理組み込み関数 (続き)

| 機能 | 説明 |
|---|---|
| HEXIMAGE8 | バッファーの UTF-8 16 進表記である文字ストリングを返す。 |
| MEMCONVERT | ソース・バッファー内のデータを、指定したソース・コード・ページから指定したターゲット・コード・ページに変換する。ターゲット・バッファー内に結果を保管する。 ターゲット・バッファーに書き込まれたバイト数を示す *size_t* 値を返す。 |
| MEMCOLLAPSE | これは、指定された文字のすべての複数オカレンスを 1 つのオカレンスで置き換え、同時にその文字の先行インスタンスと後続インスタンスのトリミングも行い、ソース・バッファーの内容をターゲット・バッファーに充てんする。 ターゲット・バッファーに書き込まれたバイト数を示す *size_t* 値を返す。 |
| MEMCU12 | ソース・バッファー内のデータを UTF-8 から UTF-16 に変換する。ターゲット・バッファー内に結果を保管する。 ターゲット・バッファーに書き込まれたバイト数を示す *size_t* 値を返す。 |
| MEMCU14 | ソース・バッファー内のデータを UTF-8 から UTF-32 に変換する。ターゲット・バッファー内に結果を保管する。 ターゲット・バッファーに書き込まれたバイト数を示す *size_t* 値を返す。 |
| MEMCU21 | ソース・バッファー内のデータを UTF-16 から UTF-8 に変換する。ターゲット・バッファー内に結果を保管する。 ターゲット・バッファーに書き込まれたバイト数を示す *size_t* 値を返す。 |
| MEMCU24 | ソース・バッファー内のデータを UTF-16 から UTF-32 に変換する。ターゲット・バッファー内に結果を保管する。 ターゲット・バッファーに書き込まれたバイト数を示す *size_t* 値を返す。 |
| MEMCU41 | ソース・バッファー内のデータを UTF-32 から UTF-8 に変換する。ターゲット・バッファー内に結果を保管する。 ターゲット・バッファーに書き込まれたバイト数を示す *size_t* 値を返す。 |
| MEMCU42 | ソース・バッファー内のデータを UTF-32 から UTF-16 に変換する。ターゲット・バッファー内に結果を保管する。 ターゲット・バッファーに書き込まれたバイト数を示す *size_t* 値を返す。 |
| MEMINDEX | バッファー中にある 1 つのストリングまたはバッファーの位置を検出する。 |
| MEMREPLACE | これは、指定された 3 番目のバッファーの 1 つ以上のオカレンスを 4 番目のバッファーで置き換えて、ソース・バッファーの内容をターゲット・バッファーに充てんする。ターゲット・バッファーに書き込まれたバイト数を示す *size_t* 値を返す。 |
| MEMSEARCH | バッファー内で、あるストリングの 1 つのエレメントの最初のオカレンスを検索する。 |
| MEMSEARCHR | バッファー内でストリングの要素のいずれか 1 つが最初に現れる位置を検索する。 検索は右から開始する。 |
| MEMSQUEEZE | これは、指定された文字のすべての複数オカレンスを 1 つのオカレンスで置き換えて、ソース・バッファーの内容をターゲット・バッファーに充てんする。ターゲット・バッファーに書き込まれたバイト数を示す *size_t* 値を返す。 |
| MEMVERIFY | バッファー内で、あるストリングの 1 つのエレメントの最初の非オカレンスを検索する。 |

表 *62.* バッファー管理組み込み関数 *(*続き*)*

| 機能 | 説明 |
|---|---|
| MEMVERIFYR | バッファー内でストリングの要素のいずれか 1 つが最初に現れる位置を検索する。 検索は右から開始する。 |
| WHEREDIFF | 2 つのバッファーを比較し、異なる最初のバイトの索引を返します。 |
| WSCOLLAPSE | CHARACTER としてエンコードされたソース・バッファー内のすべての空白文字を縮小する。 ターゲット・バッファーに書き込まれたバイト数を示す *size_t* 値を返す。 |
| WSCOLLAPSE16 | これは、UTF-16 としてエンコードされたソース・バッファー内のすべての空白文字を縮小する。ターゲット・バッファーに書き込まれたバイト数を示す *size_t* 値を返す。 |
| WSREPLACE | CHARACTER としてエンコードされたソース・バッファー内の \t、\f、\v、\n、および \r のすべての文字をブランクで置き換えます。 ターゲット・バッファーに書き込まれたバイト数を示す *size_t* 値を返す。 |
| WSREPLACE16 | UTF-16 としてエンコードされたソース・バッファー内の \t、\f、\v、\n、および \r のすべての文字をブランクで置き換えます。 ターゲット・バッファーに書き込まれたバイト数を示す *size_t* 値を返す。 |
| XMLCHAR | 構造体に対応する XML をバッファーに書き込む。 |
| XMLSCRUB | CHARACTER ソース・バッファーを消し込む。 ターゲット・バッファーに書き込まれたバイト数を示す *size_t* 値を返す。 |
| XMLSCRUB16 | これは、UTF-16 XML ソース・バッファーを消し込む。ターゲット・バッファーに書き込まれたバイト数を示す *size_t* 値を返す。 |
| XMLUCHAR | 構造体に対応する XML を UTF-8 としてバッファーに書き込みます。 |

## 条件処理組み込み関数

条件処理組み込み関数を使うと、発生した条件の原因を判別できます。

これらの関数の使用は、次の ON ユニットまたは動的子孫の範囲内でのみ有 効です。

- 組み込み関数への条件特定
- 暗黙処置として発生したときの、ERROR 条件または FINISH 条件

その他の用途は、状況に応じて判別されます。

表 *63.* 条件処理組み込み関数

| 機能 | 説明 |
|---|---|
| DATAFIELD | NAME 条件を発生させたストリングの値を返します。 |
| ONACTUAL | ASSERT COMPARE ステートメントの実際の値を表すストリングを返します。 |
| ONAREA | AREA 条件の発生対象である AREA 参照の名前を返します。 |
| ONCHAR | CONVERSION 条件の原因となった文字の値を返します。 |
| ONEXPECTED | ASSERT COMPARE ステートメントの予期される値を表すストリングを返します。 |
| ONCODE | 条件コード値を返します。 |
| ONCONDCOND | 処理対象の CONDITION 条件の名前を返します。 |

表 63. 条件処理組み込み関数 (続き)

| 機能 | 説明 |
|---|---|
| ONCONDID | 特定の条件を識別する番号を返します。 |
| ONCOUNT | 未解決の条件の数を返します。 |
| ONFILE | 条件が発生したファイルの名前を返します。 |
| ONGSOURCE | CONVERSION 条件の原因となった漢字ストリングの値を返します。 |
| ONHBOUND | 配列索引によって SUBSCRIPTRANGE が発生した場合に、上限の値を戻します。 |
| ONKEY | 条件を発生させたレコードのキーを返します。 |
| ONLBOUND | 配列索引によって SUBSCRIPTRANGE が発生した場合に、下限の値を戻します。 |
| ONLINE | 条件が発生したソースの行番号を返します。 |
| ONLOC | ONPROC の同義語です。 |
| ONOFFSET | 条件が発生したブロック内のオフセットを返します。 |
| ONOPERATOR | ASSERT COMPARE ステートメントの演算子の値を返します。 |
| ONPACKAGE | ASSERTION 条件が発生した PACKAGE の名前を返します。 |
| ONPROCEDURE | 条件が発生したプロシージャーの名前を返します。 |
| ONSOURCE | CONVERSION 条件の原因となったストリングの値を返します。 |
| ONSUBSCRIPT | SUBSCRIPTRANGE の発生の原因となった無効な配列索引の値を戻します。 |
| ONTEXT | ASSERTION 条件を発生させた ASSERT ステートメントの TEXT 節の値を返します。 |
| ONUCHAR | CONVERSION 条件の原因となった UCHAR(1) ストリングを返します。 |
| ONUSOURCE | CONVERSION 条件の原因となった UCHAR ストリングを返します。 |
| ONWCHAR | CONVERSION 条件の原因となった WIDECHAR(1) ストリングを返します。 |
| ONWSOURCE | CONVERSION 条件の原因となった WIDECHAR ストリングの値を返します。 |

## 日付/時刻組み込み関数

これらの組み込み関数は、日付および時刻の情報を日、秒数、および文字の日 付/タイム・スタンプで戻しまたは処理します。

これらの組み込み関数のなかには､使われる日付/時刻のパターンを指定で きるものがあります。 381 ページの表 64 には、サポートされている日付/時刻組み込み関数をリストしてあります。 382 ページの表 65 には、サポートされる日付/時刻パターンと 383 ページの表 66 がリストされていますサポートされる時間のみのパターンをリストします。

これらの関数の時間帯と精度は、システムよって決まります。

**リリアン形式 :** リリアン形式という名前は、グレゴリオ暦の作成者である Luigi Lilio の名前から付けられています。このフォーマットでは、グレゴリー・カレンダーの始めからの日数または秒数で日付を表します。この形式は、経過時間を含む計算を実行するのに役立ちます。

リリアン形式では、経過した日を 1582 年 10 月 14 日から数えます。 したがって、1 日目は 1582 年 10 月 15 日金曜日です。 例えば、1988 年 5 月 16 日は、リリアン形式では 148138 です。 リリアン形式の日付の有効範囲は、1 から 3,074,324 (1582 年 10 月 15 日から 9999 年 12 月 31 日) までです。

経過秒数の場合は、リリアン形式では、1582 年 10 月 14 の 00:00:00 で開始されます。 例えば、1582 年 10 月 15 日の 00:00:01 は 86,401 (24*60*60+1) リリアン秒で、1988 年 5 月 16 日の 19:01:01 は 12,799,191,661 リリアン秒です。 リリアン形式の秒の有効範囲は、86,400 から 265,621,679,999.999 (9999 年 12 月 31 日 23:59:59:) 秒です。

表 *64.* 日付/時刻組み込み関数

| 機能 | 説明 |
|---|---|
| DATE | YYMMDD のパターンの現在の日付を戻す。 |
| DATETIME | ユーザー指定のパターンまたはデフォルトのパターン YYYYMMDDHHMISS999 で現 在の日付と時刻を戻す。 |
| DAYS | 日付/時刻パターンのストリングに対応する日を示す数、または今日の日付を示す数を戻す。 |
| DAYSTODATE | 日を示す数を日付/時刻パターンのストリングに変換する。 |
| DAYSTOMICROSECS | 日数をマイクロ秒数に変換する。 |
| DAYSTOSECS | 日数を秒数に変換する。 |
| JULIANTOSMF | 日付をユリウス形式から SMF 形式に変換する。 |
| MAXDATE | 指定された日付/時刻パターンの最遅日付/時刻値を返す。 |
| MICROSECS | これは、日時パターン・ストリングに対応する時間 (マイクロ秒)、または現在日時を表すマイクロ秒数を返す。 |
| MICROSECSTODATE | これは、マイクロ秒数を日時パターン・ストリングに変換する。 |
| MICROSECSTODAYS | これは、マイクロ秒数を日数に変換する。 |
| MINDATE | 指定された日付/時刻パターンの最早日付/時刻値を返す。 |
| REPATTERN | 1 つのパターンでは日付を保持している値をとり、2 番目のパターンでは日付に変換された値を戻します。 |
| SECS | 日付/時刻パターンのストリングに対応する秒数、または今日の日付を示す秒数を戻す。 |
| SECSTODATE | 秒数を日付/時刻パターンのストリングに変換する。 |
| SECSTODAYS | 秒数を日を示す数に変換する。 |
| SMFTOJULIAN | 日付を SMF 形式からユリウス形式に変換する。 |
| STCKETODATE | STCKE 値を日付/時刻パターンのストリングに変換する。 |
| STCKTODATE | STCK 値を日付/時刻パターンのストリングに変換する。 |
| TIME | パターン HHMISS999 で現在の時刻を戻す。 |
| TIMESTAMP | 現在時刻をパターン *YYYY-MM-DD-HH.MI.SS*.999999 で返す。 |
| UTCDATETIME | 現在の協定世界時 (UTC) をパターン *YYYYMMDDHHMISS*999 で返す。 |
| UTCMICROSECS | 現在の UTC 時刻に対応するマイクロ秒数を返す。 |
| UTCSECS | 現在の協定世界時 (UTC) を秒単位のリリアン形式で返す。 |
| VALIDDATE | ストリングに有効な日付があるかどうかを示す。 |
| WEEKDAY | 現在の日または指定された DAYS 値に対応する曜日を戻す。 |
| Y4DATE | パターン 'YYMMDD' を使って日付の値をとり、2 桁の年を 4 桁の年にして日付の値を戻す。 |
| Y4JULIAN | パターン 'YYDDD' を使って日付の値をとり、2 桁の年を 4 桁の年にして日付の値を戻す。 |
| Y4YEAR | パターン 'YY' を使って日付の値をとり、2 桁の年を 4 桁の年にして日付の値を戻す。 |

382 ページの表 65 と 383 ページの表 66 では、以下のフォーマットを使用しています。

**YYYY**
4 桁の年

**YY**
2 桁の年

**ZY**
2 桁の年 (先行ゼロを消去)

**MM**
2 桁の月

**ZM**
2 桁の月 (先行ゼロを消去)

**MMM**
3 文字の月 (例 :DEC)

**Mmm**
3 文字の月 (例 :Dec)

**DD**
指定された月での 2 桁表示の日付

**ZD**
指定された月での 2 桁表示の日付 (先行ゼロを消去)

**DDD**
指定された年での日数

**HH**
指定された日での時間数

**MI**
指定された時間内の分数

**SS**
指定された分内の秒数

**999**
指定された秒内のミリ秒数

**999999**
指定された秒内のマイクロ秒数

**注 :** 3 文字の月パターンでは、大文字と小文字を正確に一致させる必要があります。

表 *65.* 日付/時刻のパターン

| | **4 桁の年** | **2 桁の年** |
|---|---|---|
| 年から書き始める場合 | YYYYMMDD<br>YYYY/MM/DD<br>YYYYMMMDD<br>YYYYMmmDD<br>YYYYDDD<br>YYYYMM<br>YYYYMMM<br>YYYYMmm<br>YYYY<br>YYYYMMDDHHMISS999<br>YYYY-MM-DD-HH.MI.SS.999999<br>YYYY-MM-DD HH:MI:SS.999999<br>YYYY-MM-DDTHH:MI:SS.999999<br>YYYYMMDDHHMISS999999 | YYMMDD<br>YY/MM/DD<br>YYMMMDD<br>YYMmmDD<br>YYDDD<br>YYMM<br>YYMMM<br>YYMmm<br>YY |

表 65. 日付/時刻のパターン (続き)

| | 4 桁の年 | 2 桁の年 |
|---|---|---|
| 月から書き始める場合 | MMDDYYYY<br>MMMDDYYYY<br>MmmDDYYYY<br>MMYYYY<br>MMMYYYY<br>MmmYYYY | MMDDYY<br>MMMDDYY<br>MmmDDYY<br>MMYY<br>MMMYY<br>MmmYY |
| 日から書き始める場合 | DDMMYYYY<br>DD/MM/YYYY<br>DDMMMYYYY<br>DDMmmYYYY<br>DDDYYYY | DDMMYY<br>DD/MM/YY<br>DDMMMYY<br>DDMmmYY<br>DDDYY |
| Db2 フォーマット | YYYY-MM-DD<br>MM/DD/YYYY<br>DD.MM.YYYY | YY-MM-DD<br>MM/DD/YY<br>DD.MM.YY |
| ゼロなし | | ZY-ZM-ZD<br>YY-ZM-ZD<br>ZM/ZD/ZY<br>ZM/ZD/YY<br>ZD.ZM.ZY<br>ZD.ZM.YY |

表 66. 時刻のみのパターン

| 基本フォーマット | 拡張フォーマット |
|---|---|
| HHMISS | HH:MI:SS |
| HHMI | HH:MI |
| HH | |

日がパターンから省略されると、値 1 が想定されます。 月と日の両方が省略された場合も、値 1 が想定されます。

MMM を使用しているときは日付を 3 つの大文字で書き、Mmm を使用しているときは最初の文字を大文字で書き後の 2 文字を小文字で書かなければなりません。

入力では、「ゼロなし」のパターンの日付値は、8 文字未満になる場合があります。例えば、2008 年 1 月 20 日という日付は、パターン「ZY-ZM-ZD」に合わせると 8-1-20 と指定されます。 出力では、これらのいずれかのパターンとして生成されるストリングは、ゼロ消去の場合、末尾ブランクで補われ、常に 8 文字になります。

## 浮動小数点の照会組み込み関数

浮動小数点の照会組み込み関数は、指定した浮動小数点変数の引数についての情報を戻します。

表 67. 浮動小数点の照会組み込み関数

| 機能 | 説明 |
|---|---|
| EPSILON | 1 と次の正数間のスペースを戻す。 |
| HUGE | 浮動小数点変数が保持できる最大の正の有限値を戻す。 |

表 67. 浮動小数点の照会組み込み関数 (続き)

| 機能 | 説明 |
|---|---|
| ISFINITE | 浮動小数点値が NAN でなく、正または負の無限大でないかどうかを示す。 |
| ISINF | 浮動小数点値が無限大であるかどうかを示す。 |
| ISNAN | 浮動小数点値が NAN であるかどうかを示す。 |
| ISNORMAL | 浮動小数点値がゼロ、非正規化数、無限大、または NaN でないかどうかを示す。 |
| ISZERO | 浮動小数点値がゼロであることを示す。 |
| MAXEXP | 指数の最大値を戻す。 |
| MINEXP | 指数の最小値を戻す。 |
| PLACES | 浮動小数点の値のモデル精度を戻す。 |
| RADIX | 浮動小数点の値のモデル基数を戻す。 |
| TINY | 浮動小数点変数が保持できる正の最小値を戻す。 |

## 浮動小数点演算組み込み関数

浮動小数点演算組み込み関数は、指定した浮動小数点変数で数学演算を実行し、演算結果を戻します。

表 68. 浮動小数点演算組み込み関数

| 機能 | 説明 |
|---|---|
| EXPONENT | 浮動小数点の値の指数部分を戻す。 |
| PRED | 浮動小数点の値の前の次に表示可能な値を戻す。 |
| SCALE | 浮動小数点数を基数の整数乗で乗算する。 |
| SUCC | 浮動小数点の値の後の次に表示可能な値を戻す。 |

## 入出力組み込み関数

入力および出力組み込み関数により、ファイルの現在の状態を判別できます。

表 69. 入出力組み込み関数

| 関数 | 説明 |
|---|---|
| COUNT | 最後の GET または PUT 時に伝送されたデータ項目の数を戻す。 |
| ENDFILE | ファイルがオープンされていて、ファイルの終わりに到達しているかどうかを示す。 |
| FILEDDINT | 指定したファイル属性の値を戻す。 |
| FILEDDTEST | 指定したファイルに指定の属性が適用される場合、値 1 を戻す。 |
| FILEDDWORD | 指定したファイル属性の文字ストリングを戻す。 |
| FILEID | ファイルのシステム・トークン値を戻す。 |
| FILENEW | 自動ストレージ内の新規ファイル定数を指す FILE 変数を返す。 |
| FILEOPEN | ファイルがオープンしているかどうかを示す。 |
| FILEREAD | ファイルから読み取る。 |
| FILESEEK | 現在のファイル位置を新しい位置に変更する。 |
| FILETELL | 現在のファイル位置を示している値を戻す。 |

表 69. 入出力組み込み関数 (続き)

| 関数 | 説明 |
|---|---|
| FILEWRITE | ファイルに書き込む。 |
| LINENO | プリント・ファイルに関連した現在の行番号を戻す。 |
| ONSUBCODE | 特定の入出力エラー、JSON エラー、または変換エラーに関する追加情報を提供する整数値を返す。 |
| ONSUBCODE2 | 特定の入出力エラーに関する追加情報を提供する整数値を戻す。 |
| PAGENO | プリント・ファイルに関連した現在のページ番号を戻す。 |
| SAMEKEY | 同じキーで別のレコードが続くレコードかどうかを示す。 |

## 整数演算組み込み関数

整数演算組み込み関数は、整変数で演算を実行し、演算結果を戻します。

表 70. 整数演算組み込み関数

| 機能 | 説明 |
|---|---|
| IAND | 2 つ以上の固定 2 進値のビット単位 "and" を計算します。 |
| ICLZ | FIXED BIN 値内の先行ゼロ数をカウントする。 |
| IEOR | 2 つの固定 2 進値のビット単位"排他 OR" を計算します。 |
| INOT | 固定 2 進値のビット単位 "not" を計算します。 |
| IOR | 2 つ以上の固定 2 進値のビット単位 "or" を計算します。 |
| ISIGNED | ビット・パターンを変更せずに、整数を符号付き整数にキャストする。 |
| ISLL | 固定 2 進値を"論理的に"左にシフトします。 |
| ISRL | 固定 2 進値を"論理的に"右にシフトします。 |
| IUNSIGNED | ビット・パターンを変更せずに、整数を符号なし整数にキャストする。 |
| LOWER2 | 固定小数点 2 進数の値を 2 の整数乗によって除算する。 |
| RAISE2 | 固定小数点 2 進数の値を 2 の整数乗で乗算する。 |

## JSON 組み込み関数

このトピックには JSON 組み込み関数がリストされています。

表 71. *JSON* 組み込み関数

| 機能 | 説明 |
|---|---|
| JSONGETARRAYEND | JSON テキスト内の (空白文字を無視した場合の) 次の文字が右大括弧 (]) であるかどうかを検査する。 |
| JSONGETARRAYSTART | JSON テキスト内の (空白文字を無視した場合の) 次の文字が左大括弧 ([) であるかどうかを検査する。 |
| JSONGETCOLON | JSON テキスト内の (空白文字を無視した場合の) 次の文字がコロンであるかどうかを検査する。 |
| JSONGETCOMMA | JSON テキスト内の (空白文字を無視した場合の) 次の文字がコンマであるかどうかを検査する。 |
| JSONGETMEMBER | JSON テキストからメンバー (名前/値のペア) を読み取る。 |

表 71. JSON 組み込み関数 (続き)

| 機能 | 説明 |
|---|---|
| JSONGETOBJECTEND | JSON テキスト内の (空白文字を無視した場合の) 次の文字が右中括弧 (}) であるかどうかを検査する。 |
| JSONGETOBJECTSTART | JSON テキスト内の (空白文字を無視した場合の) 次の文字が左中括弧 ({) であるかどうかを検査する。 |
| JSONGETVALUE | JSON テキストから値を読み取る。 |
| JSONPUTARRAYEND | 右大括弧 (]) を JSON テキストに追加する。 |
| JSONPUTARRAYSTART | 左大括弧 ([) を JSON テキストに追加する。 |
| JSONPUTCOLON | コロンを JSON テキストに追加する。 |
| JSONPUTCOMMA | コンマを JSON テキストに追加する。 |
| JSONPUTMEMBER | メンバー (名前/値のペア) を UTF-8 として JSON テキストに追加する。 |
| JSONPUTOBJECTEND | 右中括弧 (}) を JSON テキストに追加する。 |
| JSONPUTOBJECTSTART | 左中括弧 ({) を JSON テキストに追加する。 |
| JSONPUTVALUE | 値を UTF-8 として JSON テキストに追加する。 |
| JSONVALID | バッファーに、有効な JSON テキストが入っているかどうかを検査する。 |

## 数学組み込み関数

これらの関数はすべて、浮動小数点の値で演算を行い、浮動小数点の結果を生成します。 浮動小数点でない引数は、変換されます。

これらの関数の精度については、375 ページの『数学関数の正確度』で説明されています。

386 ページの表 72 に、数学組み込み関数をリストします。

表 72. 数学組み込み関数

| 関数 | 説明 |
|---|---|
| ACOS | 逆余弦を計算する。 |
| ASIN | 逆正弦を計算する。 |
| ATAN | 逆正接を計算する。 |
| ATAND | 逆正接を度で計算する。 |
| ATANH | 双曲線逆正接を計算する。 |
| COS | 余弦を計算する。 |
| COSD | 余弦を度で計算する。 |
| COSH | 双曲線余弦を計算する。 |
| ERF | 誤差関数を計算する。 |
| ERFC | 誤差関数の補数を計算する。 |
| EXP | e の累乗を計算する。 |
| GAMMA | ガンマ関数を計算する。 |
| LOG | 自然対数を計算する。 |
| LOG10 | 底 10 の常用対数を計算する。 |

表 72. 数学組み込み関数 (続き)

| 関数 | 説明 |
|---|---|
| LOG2 | 底 2 の対数を計算する。 |
| LOGGAMMA | ガンマ関数のログを計算する。 |
| SIN | 正弦を計算する。 |
| SIND | 正弦を度で計算する。 |
| SINH | 双曲線正弦を計算する。 |
| SQRT | 平方根を計算する。 |
| SQRTF | SQRT をインラインで計算する (ハードウェア・アーキテクチャーによって処理可能な場合)。 |
| TAN | 正接を計算する。 |
| TAND | 正接を度で計算する。 |
| TANH | 双曲線正接を計算する。 |

## その他の組み込み関数

このトピックでは、他のどのカテゴリーにも該当しない組み込み関数をリストします。

次の表はその他の組み込み関数をリストしたものです。

表 73. その他の組み込み関数

| 機能 | 説明 |
|---|---|
| ALLCOMPARE | 2 つの構造体の比較結果を示す BIT(1) 値を返す。 |
| BETWEEN | 最初の引数が 2 番目と 3 番目の引数で定義されている閉区間にあるかどうかを示す BIT(1) 値を返す。 |
| BETWEENEXCLUSIVE | 最初の引数が 2 番目と 3 番目の引数で定義されている開区間にあるかどうかを示す BIT(1) 値を返す。 |
| BETWEENLEFTEXCLUSIVE | 最初の引数が 2 番目と 3 番目の引数で定義されている左開区間にあるかどうかを示す BIT(1) 値を返す。 |
| BETWEENRIGHTEXCLUSIVE | 最初の引数が 2 番目と 3 番目の引数で定義されている右開区間にあるかどうかを示す BIT(1) 値を返す。 |
| BINSEARCH | 単純比較を使用して二分探索を実行する。 |
| BINSEARCHX | 指定された比較関数を使用して二分探索を実行する。 |
| バイト | CHARVAL の同義語。 |
| BYTELENGTH | UCHAR ストリングで使用されるバイト数である FIXED BINARY(31) 値を返す。 |
| CDS | ダブル比較およびスワップ の以前の値と現行値が 等しいかどうかを示す FIXED BINARY(31) 値を戻す。 |
| CHARVAL | 整数に対応する文字値を戻す。 |
| CODEPAGE | CODEPAGE コンパイラー・オプションの値を保持する FIXED BINARY(31) 値を返す。 |
| COLLATE | 照合順序を指定する文字 (256) ストリングを戻す。 |

表 73. その他の組み込み関数 (続き)

| 機能 | 説明 |
|---|---|
| CS | 比較およびスワップ の以前の値と現行値が 等しいかどうかを示す FIXED BINARY(31) 値を戻す。 |
| FOLDEDFULLMATCH | Unicode の完全な 大/小文字変換規則に従って小文字に変換されたときに 2 つのストリングが同じかどうかを示す FIXED BINARY(31) 値を返す。 |
| FOLDEDSIMPLEMATCH | Unicode の単純な 大/小文字変換規則に従って小文字に変換されたときに 2 つのストリングが同じかどうかを示す FIXED BINARY(31) 値を返す。 |
| GETENV | 指定した環境変数を表す値を戻す。 |
| GETJCLSYMBOL | これは、エクスポートされた JCL シンボルの値である文字ストリングを返す (z/OS のみ)。 |
| GETSYSINT | 要求されたシステム情報の値である *size_t* 値を返す。 |
| GETSYSWORD | 要求されたシステム情報の値である文字ストリングを返す。 |
| GTCA | LE 制御ブロックへのポインターを戻します。 |
| HEX | 値の 16 進表示の文字ストリングを戻す。 |
| HEX8 | 値の UTF-8 16 進表記である文字ストリングを返す。 |
| IFTHENELSE | C 条件式 (`x?y:z`) と等価な値を返す。 |
| INDICATORS | 構造体内の次の論理レベルにあるエレメントの数を示す値を返す。 |
| INLIST | 最初の引数が残りの引数のいずれかと等しいかどうかを示す BIT(1) 値を返す。 |
| ISJCLSYMBOL | 入力引数名が有効なエクスポート済み JCL シンボルであるかどうかを示す BIT(1) 値を返す。 |
| ISMAIN | 現行プロシージャーがメインかどうかを示す。 |
| MAINNAME | 現行呼び出しスタックでの MAIN 関数の名前である CHARACTER ストリングを返す。 |
| OMITTED | 呼び出しでパラメーターが指定されていないかどうかを示す。 |
| PACKAGENAME | 包含パッケージの名前を戻す。 |
| PLIRETV | PL/I の戻りコード値を戻す。 |
| POPCNT | 対応するバイトの 1 に等しいビット数を各バイトに保持する FIXED BINARY 値を返します。 |
| PRESENT | 呼び出しでパラメーターが提供されていないかどうかを示す。 |
| PROCEDURENAME | ネストされた最も近いプロシージャーの名前を戻す。 |
| PUTENV | 新しい環境変数を追加するか、または既存の環境変数の値を変更する。 |
| RANK | CHARACTER または WIDECHAR に対応する整数値を返す。 |
| SOURCEFILE | ステートメントが含まれているファイルの名前を戻す。 |

表 73. その他の組み込み関数 (続き)

| 機能 | 説明 |
|---|---|
| SOURCELINE | ステートメントが含まれている行の番号を戻す。 |
| STACKADDR | 現在の動的保存域のアドレスを返す。 |
| STRING | ストリングの集合のすべてのエレメントの連結であるストリングを戻す。 |
| SYSTEM | コマンドによって戻される値を戻す。 |
| THREADID | タスクのスレッド ID を戻す。 |
| UNHEX | 16 進入力ストリングのデコード済みの値である文字ストリングを返す。 |
| UNSPEC | 値の内部表示のビット・ストリングを戻す。 |
| UUID | バージョン 5 フォーマットの汎用固有 ID である CHARACTER(36) ストリングを返す。 |
| UUID4 | バージョン 4 の汎用固有 ID である CHARACTER(36) ストリングを返す。 |
| VALID | 変数の内容が宣言に有効であるかどうかを示す。 |
| VALIDVALUE | 式の値が変数の値のセットにおけるエレメントの 1 つと一致するかどうかを示す。 |
| WCHARVAL | 整数に対応する WIDECHAR 値を返す。 |

## 序数処理組み込み関数

序数処理データ組み込み関数は、指定された序数に関する情報を戻します。

表 74. 序数処理組み込み関数

| 機能 | 説明 |
|---|---|
| ORDINALNAME | 序数の名前を指定する文字ストリングを戻します。 |
| ORDINALPRED | 序数が次に低い値を戻します。 |
| ORDINALSUCC | 次に高い序数値を戻します。 |

## 精度処理組み込み関数

精度処理組み込み関数により、指定された精度で変数を演算し、演算結果の値 を戻します。

表 75. 精度処理組み込み関数

| 機能 | 説明 |
|---|---|
| ADD | 指定された精度で、2 つの値を追加する。 |
| BINARY | 値を 2 進数に変換する。 |
| DECIMAL | 値を 10 進数に変換する。 |
| DIVIDE | 指定された精度で、2 つの値を除算する。 |
| 固定 | 値を固定小数点に変換する。 |
| FIXEDBIN | 値を固定小数点 2 進数に変換する。 |
| FIXEDDEC | 値を固定小数点 10 進数に変換する。 |

表 75. 精度処理組み込み関数 (続き)

| 機能 | 説明 |
|---|---|
| FLOAT | 値を浮動小数点に変換する。 |
| FLOATBIN | 値を浮動小数点 2 進数に変換する。 |
| FLOATDEC | 値を浮動小数点 10 進数に変換する。 |
| MULTIPLY | 指定された精度で、2 つの値を乗算する。 |
| PRECVAL | 数値オペランドの精度を返す。 |
| PRECISION | 値を、指定の精度に変換する。 |
| SCALEVAL | 数値オペランドの位取り係数を返す。 |
| SIGNED | 値を符号付き固定小数点 2 進数に変換する。 |
| SUBTRACT | 指定された精度で、2 つの値を減算する。 |
| UNSIGNED | 値を符号なし固定小数点 2 進数に変換する。 |

# 疑似変数

疑似変数は、受信フィールドを表します。 疑似変数をネストすることはできません。 このトピックには組み込み疑似変数がリストされています。

例えば、次のものは正しくありません。

```
unspec(substr(A,1,2)) = '00'B;
```

疑似変数が存在できるのは、次のような場合のみです。

- 代入ステートメントの左側。
- DO 指定内のターゲットであり、なおかつ、その疑似変数が SUBSTR、REAL、IMAG、UNSPEC のいずれかである場合。
- GET ステートメントのデータ・リスト内、または PUT ステートメントの STRING オプション内。
- KEYTO オプションまたは REPLY オプション内のストリング名として。

表 76. 組み込み疑似変数

| 機能 | 説明 |
|---|---|
| ENTRYADDR | 呼び出される入り口のアドレスで入り口変数を設定する。 |
| IMAG | 複素数の虚数部分を割り当てる。 |
| ONCHAR | 変換条件の原因となる文字の値を設定する。 |
| ONGSOURCE | 変換条件の原因となる漢字ストリングの値を設定する。 |
| ONSOURCE | 変換条件の原因となるストリングの値を設定する。 |
| REAL | 複素数の実数部分を割り当てる。 |
| STRING | ストリングの集合のすべてのエレメントの連結であるストリングを割り当てる。 |
| SUBSTR | 長さで指定された、ストリングのサブストリングを割り当てます。 |
| SUBTO | 終了位置で指定された、ストリングのサブストリングを割り当てます。 |
| ONUCHAR | ONUCHAR 組み込み関数の現行値を設定します。 |
| ONUSOURCE | ONUSOURCE 組み込み関数の現行値を設定します。 |

表 76. 組み込み疑似変数 (続き)

| 機能 | 説明 |
|---|---|
| ONWCHAR | 変換条件の原因となるワイド文字の値を設定する。 |
| ONWSOURCE | 変換条件の原因となるワイド文字ストリングの値を設定する。 |
| TYPE | タイプ付き構造体または共用体をハンドルによって位置付けられたストレージに割り当てる。 |
| UNSPEC | 値の内部表示であるビット・ストリングを割り当てる。 |

## ストレージ制御組み込み関数

ストレージ制御組み込み関数を使用すれば、ストレージ要件および変数の位置を判別し、特殊値を区域変数およびロケーター変数に割り当て、オフセット値とポインター値の間で変換を実行し、被制御変数の世代番号を取得し、オブジェクトおよびクラスのデータおよびメソッドを参照することができます。

391 ページの表 77 に、ストレージ制御組み込み関数をリストします。

表 77. ストレージ制御組み込み関数

| 機能 | 説明 |
|---|---|
| ADDR | 変数のアドレスを戻す。 |
| ADDRDATA | 可変ストリングに適用されるときに、ストリングの最初のデータ・バイトのアドレスを戻す。 |
| ALLOC31 | 指定したサイズのストレージを 2 GB 境界より下のヒープに割り振る。 |
| ALLOCATE | ヒープ内に、指定されたサイズのストレージを割り振る。 |
| ALLOCATION | 被制御変数の現在の世代番号を戻す。 |
| ALLOCNEXT | 使用可能な最初のチャンク内に十分なスペースがあれば、指定されたサイズのストレージを AREA 内に割り振る。 |
| ALLOCSIZE | 特定のポインターとともに割り振られたストレージ量を示す FIXED BIN(N,0) 値を戻す。 |
| AUTOMATIC | スタック内に、指定されたサイズのストレージを割り振る。 |
| AVAILABLEAREA | 区域内で、単一の作成可能な最大割り振りサイズを戻す。 |
| BINARYVALUE | ポインター、オフセット、または序数を整数に変換する。 |
| BITLOCATION | バイト内での変数のビット・オフセットを戻す。 |
| CHECKSTG | 割り振られていたストレージが破壊されていないかどうかを判別するビット (1) 値を戻す。 |
| CURRENTSIZE | 変数の現行サイズを戻す。 |
| CURRENTSTORAGE | CURRENTSIZE の同義語。 |

表 77. ストレージ制御組み込み関数 (続き)

| 機能 | 説明 |
|---|---|
| EMPTY | "空の"領域を返します。 |
| ENTRYADDR | 入り口に関連したルーチンのアドレスを戻す。 |
| HANDLE | タイプ付き構造体または共用体にハンドルを戻す。 |
| LOCATION | 構造体内の変数のバイト・オフセットを戻す。 |
| LOCSTG | LOCATES を使用して間接的に保持できるすべての値を保持するために必要となるすべての割り振り済みストレージを保持するために必要なバイト数を返す。 |
| LOCVAL | LOCATES の description で指定されたタイプの区域内で、指定されたオフセットにある値を返す。 |
| NULL | ヌル・ポインター値を戻す。 |
| NULLENTRY | ヌル値の LIMITED 入り口値を戻す。 |
| OFFSET | ポインターをオフセットに変換する。 |
| OFFSETADD | 整数をオフセットに追加する。 |
| OFFSETDIFF | 2 つのオフセットを減算する。 |
| OFFSETSUBTRACT | オフセットから整数を減算する。 |
| OFFSETVALUE | 整数をオフセットに変換する。 |
| POINTER | オフセットをポインターに変換する。 |
| POINTERADD | 整数をポインターに加える。 |
| POINTERDIFF | 2 つのポインターを減算する。 |
| POINTERSUBTRACT | ポインターから整数を減算する。 |
| POINTERVALUE | 整数またはハンドルをポインターに変換する。 |
| SIZE | 変数の最大サイズを戻す。 |
| STORAGE | SIZE の同義語。 |
| SYSNULL | システム・ヌル・ポインターの値を戻す。 |

表 77. ストレージ制御組み込み関数 (続き)

| 機能 | 説明 |
|---|---|
| TYPE | ハンドルによって位置付けされるタイプ付き構造体または共用体を戻す。 |
| UNALLOCATED | 指定されたポインター値が割り振り済みストレージの開始点であるかどうかを示すビット (1) 値を戻す。 |
| VARGLIST | プロシージャーに渡された最初のオプション・パラメーターのアドレスを戻す。 |
| VARGSIZE | byvalue パラメーターが占めるバイト数を戻す。 |

## ストリング処理組み込み関数

ストリング処理組み込み関数は、BIT、CHARACTER、GRAPHIC、UCHAR、および WIDECHAR のストリングの処理を簡素化するものです。

ストリングの引数は、データ変換規則、または関数の記述で与えられた規則 のどちらかに従ったストリングに変換される算術式で表されます。

**注 :** これらの一部の関数 (LOWERCASE や UPPERCASE など) は、CHARACTER データのみをサポートします。

表 78. ストリング処理組み込み関数

| 機能 | 説明 |
|---|---|
| ビット | 値をビットに変換する。 |
| BOOL | 2 ビット・ストリングでブール演算を実行する。 |
| CENTERLEFT | 値が中央から左方向に配置されたストリングを戻す。 |
| CENTERRIGHT | 値が中央から右方向に配置されたストリングを戻す。 |
| CENTRELEFT | CENTERLEFT の同義語。 |
| CENTRERIGHT | CENTERRIGHT の同義語。 |
| CHARACTER | 値を文字ストリングに変換する。 |
| CHARGRAPHIC | GRAPHIC ストリングを混合文字ストリングに変換する。 |
| COLLAPSE | 1 つの文字のすべての複数オカレンスを 1 つのオカレンスに削減し、その文字の先行インスタンスと末尾インスタンスをトリミングしたストリングを返す。 |
| COPY | ストリングのコピーが n 個含まれるストリングを戻す。 |
| EDIT | 与えられたピクチャー指定に変換された値で構成されるストリングを戻す。 |
| GRAPHIC | 値をグラフィックに変換する。 |
| HIGH | 照合順序の最も高い文字を n 回コピーして構成される文字 ストリングを戻す。 |
| INDEX | ストリング中にある 1 つのストリングの位置を検出する。 |
| INDEXR | 右から検索する場合に、ストリング中にある 1 つのストリングの位置を検出する。 |
| LEFT | 左詰めの値が入っているストリングを戻す。 |
| LENGTH | ストリングの現行の長さを戻す。 |
| LOW | 照合順序の最も低い文字を n 回コピーして構成される文字 ストリングを戻す。 |

表 78. ストリング処理組み込み関数 (続き)

| 機能 | 説明 |
|---|---|
| LOWERASCII | ASCII 文字すべてを対応する小文字に変換した UCHAR ストリングを返す。 |
| LOWERCASE | A から Z までの文字すべてを対応する小文字に変換した文字ストリングを戻す。 |
| LOWERLATIN1 | ASCII および Latin-1 補助文字すべてを対応する小文字に変換した UCHAR ストリングを返す。 |
| MAXLENGTH | ストリングの最大長を戻す。 |
| MPSTR | 論理境界でストリングを切り捨て、混合文字ストリングを戻す。 |
| PICSPEC | 与えられたピクチャー指定を持つと想定される値で構成されるストリングを戻す。 |
| 正規表現 | ストリングに正規表現との一致があるか検索する。 |
| REPEAT | n+1 回コピーしたストリングを戻す。 |
| 置換 | 別のサブストリングに置き換えられたサブストリングのオカレンスが 1 つ以上あるストリングを返す。 |
| REPLACEBY2 | 一部の文字が、 文字のペアに「変換された」ストリングを戻す。 |
| REVERSE | ストリングの反転表示を戻す。 |
| RIGHT | 右寄せの値が入っているストリングを戻す。 |
| SCRUBOUT | 2 番目のストリングからすべての文字を除去したストリングを返す。 |
| SEARCH | 別のストリング内で、あるストリングの 1 つのエレメントの最初のオカレンスを検索する。 |
| SEARCHR | 別のストリング内でストリングの要素のいずれか 1 つが最初に現れる位置を検索する。 検索は右から開始する。 |
| SQUEEZE | 1 つの文字のすべての複数オカレンスを 1 つのオカレンスに削減したストリングを返す。 |
| SUBSTR | 長さで指定された、ストリングのサブストリングを戻します。 |
| SUBTO | 終了位置によって指定された、ストリングのサブストリングを戻します。 |
| TALLY | ストリングで 1 つのストリングが発生する回数を戻す。 |
| TRANSLATE | 2 つの変換ストリングに基づいてストリングを変換する。 |
| TRIM | 指定された文字をストリングの左および右からトリムする。 |
| UHIGH | 各 UTF-8 データ項目が最も大きい UCHAR 値 ('F48FBFBF'UX) を持つ、長さ *x* の UCHAR ストリングを返す。 |
| ULENGTH | CHAR または WIDECHAR ストリングの UTF 文字数を返す。 |
| ULENGTH8 | CHAR または WIDECHAR ストリングの UTF ストリングが UTF-8 に変換されている場合に必要な CHAR ストリングの長さを返す。 |
| ULENGTH16 | CHAR または WIDECHAR ストリングの UTF ストリングが UTF-16 に変換されている場合に必要な WIDECHAR ストリングの長さを返す。 |
| ULOW | 各 UTF-8 データ項目が最も小さい UCHAR 値 ('00'UX) を持つ、長さ *x* の UCHAR ストリングを返す。 |
| UPOS | CHAR または WIDECHAR ストリングの *n* 番目の UTF 文字の位置を返す。 |

表 78. ストリング処理組み込み関数 (続き)

| 機能 | 説明 |
|---|---|
| UPPERASCII | ASCII 文字すべてを対応する大文字に変換した UCHAR ストリングを返す。 |
| UPPERCASE | a から z までの文字すべてを対応する大文字に変換した文字ストリングを戻す。 |
| UPPERLATIN1 | ASCII および Latin-1 補助文字すべてを対応する大文字に変換した UCHAR ストリングを返す。 |
| USUBSTR | UTF ストリングのサブストリングを返す。 |
| USUPPLEMENTARY | CHAR または WIDECHAR ストリングの最初の UTF サロゲート・ペアの索引を返す。 |
| UTF8 | *x* と同等の UTF-8 で CHAR 値を返す。 |
| UTF8STG | 入力文字が有効な UTF-8 文字の始まりである場合に、存在しなければならないバイト数を返す。 |
| UTF8TOCHAR | UTF-8 から変換された *x* を保持する CHAR 値を返す。 |
| UTF8TOWCHAR | UTF-8 から UTF-16 に変換された *x* を保持する WCHAR 値を返す。 |
| UVALID | CHAR、UCHAR、または WIDECHAR ストリングに有効な UTF データが入っているかどうかを示す。 |
| UWIDTH | CHAR または WIDECHAR ストリングの *n* 番目の UTF 文字の幅を返す。 |
| VERIFY | 別のストリング内でストリングの 1 つのエレメントの最初の非オカレンスを検索する。 |
| VERIFYR | 別のストリング内でストリングの 1 つのエレメントの最初の非オカレンスを右から検索する。 |
| WHIGH | 最上位の WIDECHAR ('FFFF'WX) を *n* 回コピーして構成される WIDECHAR ストリングを返す。 |
| WIDECHAR | 値を WIDECHAR ストリングに変換する。 |
| WLOW | 最下位の WIDECHAR ('0000'WX) を *n* 回コピーして構成される WIDECHAR ストリングを返す。 |

## サブルーチン

組み込みサブルーチンは、組み込み関数が結果を戻すのに対して、結果を戻す必要がない その他の演算を実行します。

表 79. 組み込みサブルーチン

| 機能 | 説明 |
|---|---|
| LOCNEWSPACE | LOCATES 属性で記述された変数タイプのスペースを割り振る。 |
| LOCNEWVALUE | オフセットに関連付けられている LOCATES 属性で記述された変数タイプのスペースを割り振り、その区域に値を割り当てる。 |
| PLIASCII | EBCDIC から ASCII に変換する。 |
| PLIATTN | コンパイラーがアテンション・ブレークポイントを挿入する場所に対する明示制御を提供する。 このサブルーチンを呼び出すたびに、コード内のそのポイントで ATTENTION 条件が発生します。 |
| PLICANC | 自動再始動機能を取り消し (z/OS のみ)。 |

表 79. 組み込みサブルーチン (続き)

| 機能 | 説明 |
|---|---|
| PLICKPT | 実行時リスタートのためのチェックポイント (z/OS のみ)。 |
| PLIDELETE | ハンドルに関連したストレージを解放する。 |
| PLIDUMP | 現在のオープン・ファイル、現在場所への呼び出しパスなどについての情 報をダンプする。 |
| PLIEBCDIC | ASCII から EBCDIC に変換する。 |
| PLIFILL | 指定されたバイト値によって、アドレスに n バイトを埋める。 |
| PLIFREE | ヒープ・ストレージを指すポインターに関連したストレージを解放する。 |
| PLIMOVE | 1 つのアドレスから別のアドレスへ n バイト移動する。 |
| PLIOVER | ソースとターゲットの、可能性のあるオーバーラップを補いながら、1 つのアドレスから別のアドレスへ n バイト移動する。 |
| PLIPARSE | 文字ストリングをサブストリングに構文解析します。 |
| PLIREST | プログラム実行を再始動 (z/OS のみ)。 |
| PLIRETC | PL/I 戻りコード値を設定する。 |
| PLISAXA | プログラムのバッファー内にある XML 文書に対して、SAX フォーマットの 構文解析を実行する。 |
| PLISAXB | ファイル内にある XML 文書に対して、SAX フォーマットの 構文解析を実行する。 |
| PLISAXC | プログラムのバッファー内にある XML 文書に対して、SAX フォーマットの 構文解析を実行する。 |
| PLISAXD | プログラムのバッファー内にある XML 文書に対して、XML 妥当性検査による SAX スタイル構文解析の実行を可能にする。 |
| PLISRTA | DFSORT を使用することにより、入力ファイルをソートして、ソートされた出力ファイルの作成を可能にする。 |
| PLISRTB | これは、DFSORT を使用することにより、E15 PL/I 終了プロシージャーで提供される 入力レコードをソートして、ソート済み出力ファイルを作成できるようにする。 |
| PLISRTC | これは、DFSORT を使用することにより、入力ファイルをソートして、E35 PL/I 終了プロシージャーで処理されるソート済みレコードを作成できるようにする。 |
| PLISRTD | これは、DFSORT を使用することにより、E15 PL/I 終了プロシージャーで提供される入力レコードをソートして、E35 PL/I 終了プロシージャーで処理されるソート済みレコードを作成できるようにする。 |
| PLISTCK | これは、対応するストア・クロック・ハードウェア命令を生成し、その命令によって設定される条件コードを返す (z/OS のみ)。 |
| PLISTCKE | これは、対応するストア・クロック・ハードウェア命令を生成し、その命令によって設定される条件コードを返す (z/OS のみ)。 |
| PLISTCKF | これは、対応するストア・クロック・ハードウェア命令を生成し、その命令によって設定される条件コードを返す (z/OS のみ)。 |
| PLISTCKLOCAL | これは、対応するストア・クロック・ハードウェア命令を生成し、STCK 値を調整して現地時間を提供する (z/OS のみ)。 |
| PLISTCKELOCAL | これは、対応するストア・クロック・ハードウェア命令を生成し、STCKE 値を調整して現地時間を提供する (z/OS のみ)。 |

表 79. 組み込みサブルーチン (続き)

| 機能 | 説明 |
|---|---|
| PLISTCKP | 対応する Perform Timing Facility Function ハードウェア命令を生成します (z/OS のみ)。 |
| PLISTCKPLOCAL | 対応する Perform Timing Facility Function ハードウェア命令を生成し、現地時間を指定するように PTFF 値を調整します (z/OS のみ)。 |
| PLISTCKPUTC | 対応する Perform Timing Facility Function ハードウェア命令を生成し、UTC 時刻を指定するように PTFF 値を調整します (z/OS のみ)。 |
| PLISTCKUTC | これは、対応するストア・クロック・ハードウェア命令を生成し、STCK 値を調整して UTC 時間を提供する (z/OS のみ)。 |
| PLISTCKEUTC | これは、対応するストア・クロック・ハードウェア命令を生成し、STCKE 値を調整して UTC 時間を提供する (z/OS のみ)。 |
| PLITRANxy | x バイト・バッファーを y バイト・バッファーに変換する。*x* および *y* は、1 と 2 の任意の組み合わせを表す。 |

## 個々の組み込み関数、疑似変数、およびサブルーチンの説明

このセクションでは、組み込み関数、サブルーチン、および疑似変数をアルファベット順に示し、それぞれの関数、サブルーチン、および疑似変数について詳しく説明します。

一般的に、次のようなフォーマットで記述します。

- 参照構文を示す見出し
- 戻される値の説明、または、疑似変数の値のセット
- 引数の説明
- 関数または疑似変数の使用時のその他の修飾

組み込み関数の省略形には、宣言 (明示的またはコンテキスト) と名前の有効範囲が分かれています。

次の例は、複数の宣言ではありません。

```
dcl (Dim, Dimension) builtin;
```

次の例は、*Bin* が *Binary* 組み込み関数の省略形であっても有効です。

```
dcl Binary file;
X = Bin (var, 6,3);
```

**注 :** 一部の引数および戻り値のタイプは *size_t* です。LP(32) コンパイラー・オプションが有効な場合、*size_t* は FIXED BIN(31) です。 LP(64) コンパイラー・オプションが有効な場合、*size_t* は FIXED BIN(63) です。

### ABS

ABS は、*x* の絶対値を返します。 これは、*x* の正の値です。

**x**
式。

結果は、REAL モードです。 結果には、*x* の基数、スケール、および精度が含まれています。ただし、x が COMPLEX FIXED(p,q) の場合は除きます。 x が COMPLEX FIXED(p,q) の場合、結果は REAL FIXED(min(n,p+1),q) になります。n には、DECIMAL では N が、BINARY では M が入ります。

# ACOS

ACOS は、*x* ラジアンの逆余弦の近似を実数の浮動小数点の値で戻します。

**x**
実数式。ABS(x) <= 1

結果の範囲は以下のとおりです。

```
0 ≤ ACOS(x) ≤ π
```

また、結果は、*x* の基数と精度になります。

# 追加

ADD は、*p* と *q* で指定された精度の *x* と *y* の合計を返します。 結果の基数、スケール、およびモードは、PRECTYPE コンパイラー・オプションによって規則が変更されない限り、式の評価規則によって決まります。

**x および y**
式。

**p**
制限付き式。 演算を通して維持される桁数を指定します。

**q**
結果の位取り係数を指定する制限付き式。 結果が固定小数点の場合に、*q* が省略されるときは、 位取り係数ゼロがデフォルトです。 結果が浮動小数点の場合は、*q* は省略します。

ADD は、減算されるオペランドの接頭部に負符号 (-) を付けて、減算に使用で きます。

# ADDR

ADDR は、*x* の世代を識別するポインター値を戻します。

**x**
参照。 ADDR は、次のもの以外のデータ・タイプ変数、データ編成変数、位置合わせ 変数、およびストレージ・クラス変数を参照します。

- 位置合わせされていない固定長のビット・ストリングである変数への添え字付き参照。
- DEFINED 変数または BASED 変数、または位置合わせされていない固定長のビット・ストリングの単純パラメーターへの参照。
- 最初の基本エレメントが位置合わせされていない固定長のビット・ストリングで ある小構造体または共用体 (ただし、その小構造体を含んでいる大構造体または共用 体の最初のエレメントになっている場合を除く)。
- DEFINED 属性を持つ大構造体か共用体、またはパラメーターである大構造体か 共用体で、位置合わせされていない固定長のビット・ストリングを最初 のエレメントとするもの。
- 連結ストレージを参照しない参照。

*x* の参照先と結果を次に示します。

- 集合パラメーターの場合は、CONNECTED 属性を保持している必要があります。
- 集合の場合は、戻り値は最初のエレメントを識別します。
- コンポーネントまたはセクション間の集合の場合は、戻り値では添え字および構造体 または共用体の修飾が考慮されます。
- VARYING ストリングの場合は、戻り値は 2 バイトの接頭部を識別します。
- VARYING4 ストリングの場合は、戻り値は 4 バイトの接頭部を識別します。
- 区域。戻り値は、制御情報を示しています。
- 現行プログラムに割り振られていない被制御変数の場合は、ヌル・ポインター値が戻されます。
- 基底付き変数の場合は、結果は明示的に x を修飾するポインター (表示される場 合) か、*x* の宣言 (存在する場合) で *x* に関連付けられた ポインター、またはヌル・ポインターの値です。
- パラメーターであり、仮引数が作成されている場合は、戻り値は仮引数を識別します。

## ADDRDATA

ADDRDATA は *x* の世代を識別するポインター値を戻します。

**x**
参照。

ADDRDATA は、以下のインスタンスを除き、ADDR 組み込み関数と同じです。

- 可変ストリングに適用されるときに、ADDRDATA は、(長さフィールドの最初のデータ・バイトのアドレスではなく) ストリングの最初のデータ・バイトのアドレスを戻します。
- LOCATES 属性および暗黙的な AREA 修飾が指定された OFFSET 参照に適用されるときには、以下のようになります。
  - OFFSET 参照が非ヌルの場合、ADDRDATA は、位置指定されたデータのアドレスを返します。
  - OFFSET 参照がヌルの場合、ADDRDATA は SYSNULL を返します。

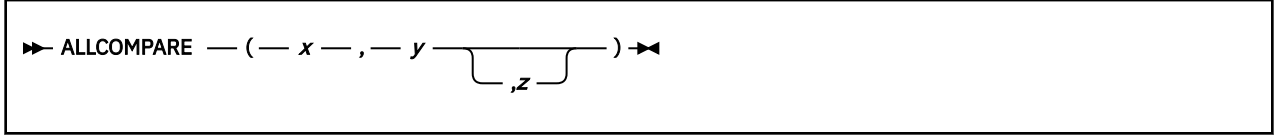
## すべて

ALL は、*x* の各エレメントに対応するビットが存在し、それが 1 である場合、各ビットが 1 であるビット・ストリングを戻します。 結果の長さは、最も長いエレメントの長さと等しくなります。

**x**
計算配列式。 *x* がビット・ストリング配列ではない場合、*x* はビット・ストリング配列に変換されます。

## ALLCOMPARE

ALLCOMPARE(*x, y, z*) は、2 つの構造体のすべてのエレメントの比較結果を示す BIT(1) 値を返します。




***x***
構造体参照。

***Y***
構造体参照。

***z***

CHAR(2) 定数。 大文字の場合、この定数の値は、EQ、LE、LT、GT、GE、または NE のいずれかでなければなりません。 *z* を指定しない場合、EQ がデフォルト値です。

**EQ**
次と等しい

**LE**
より小か等しい

**LT**
より小

**GT**
より大

**GE**
より大か等しい

**NE**
等しくない

*x* と *y* は類似する構造体参照でなければなりません。

*x* と *y* の対応する要素が比較可能でなければなりません。

例えば、ALLCOMPARE(*x*, *y*, 'lt') は、*x* のすべてのリーフ要素が、対応する *y* のリーフ要素よりも小さい場合、'1'B を返します。

# ALLOC31

ALLOC31 は、2 GB 境界より下のヒープ・ストレージにサイズ *n* のストレージを割り振り、割り振ったストレージへのポインターを返します。

**n**

式。 割り振るストレージ・サイズを表す、負でない値。 必要に応じて、n はタイプ *size_t* [1] に変換されます。

ストレージで要求量が使用可能でない場合は、STORAGE 条件が発生します。

# ALLOCATE

ALLOCATE は、ヒープ・ストレージにサイズ *n* のストレージを割り振り、割り振り済みストレージにポインターを戻します。 また、ALLOCATE を使用すると、指定された区域に、指定されたサイズを割り振ることができます。

**省略形**: ALLOC

**n**

式。 割り振るストレージ・サイズを表す、負でない値。 必要に応じて、*n* はタイプ *size_t* [1] に変換されます。

ストレージで要求量が使用可能でない場合は、STORAGE 条件が発生します。

**x**

AREA 参照。 ALLOCATE(*n*, *x*) を指定すると、指定した数のバイト *n* がその区域内で割り振られます。 この数は 8 の倍数に切り上げられます。

指定した区域内に十分なスペースがない場合は、AREA 条件が起こります。

# ALLOCATION

ALLOCATION は、*x* の現行プログラムでアクセス可能な世代番号を指定する FIXED BINARY(31 ,0) を戻します。

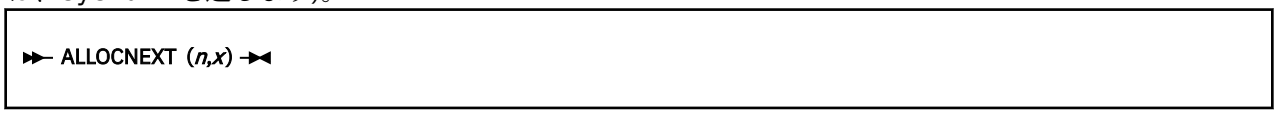
**省略形:** ALLOCN

**x**
レベル 1 の添え字なし被制御変数です。

現行プログラムで *x* が割り振られていない場合は、結果はゼロになります。

# ALLOCNEXT

ALLOCNEXT は、使用可能な最初のチャンク内に十分なスペースがあれば、指定されたサイズのストレージを AREA 内に割り振り、割り振られたストレージへのポインターを返します (十分なスペースがない場合は、sysnull を返します)。




***n***
式。 割り振るストレージ・サイズを表す、負でない値。 これは、最も近い 8 の倍数に切り上げられます。 必要に応じて、*n* は *size_t* 値に変換されます。

***x***
AREA 参照。

*x* 内の使用可能な最初のチャンクが十分に大きい場合、*x* からストレージが割り振られ、割り振られたストレージへのポインターが返されます。

使用可能な最初のチャンクが十分に大きくない場合、sysnull が返されます。 ALLOCATE 組み込み関数とは異なり、STORAGE 条件や AREA 条件が引き起こされることはありません。

ALLOCNEXT(*n,x*) が生成するコードは ALLOCATE(*n,x*) よりはるかに短く高速ですが、ユーザーは非ヌルのポインターが返されることを確認する必要があります。 FREE なしで AREA からの割り振りが繰り返されている場合や、AREA からのすべての FREE がすべての ALLOCATE と逆の順序になっている場合は、これが最適です。

# ALLOCSIZE

ALLOCSIZE は、指定されたポインターとともに割り振られたストレージ量を与えている FIXED BIN(31 ,0) 値を戻します。 この組み込み関数を使用するには、CHECK(STORAGE) コンパイル時オプションも指定しなければ なりません。

**p**
ポインターの式。

ポインターが割り振り済みストレージの始まりを示していない場合、ALLOCSIZE は 0 を戻します。

ALLOCSIZE に渡されたポインターは、最も近いダブルワードに「切り捨て」られ、切り捨てられた値は、同様に切り捨てられた場合のすべての割り振り済みアドレスに対して比較されることに注意してください。

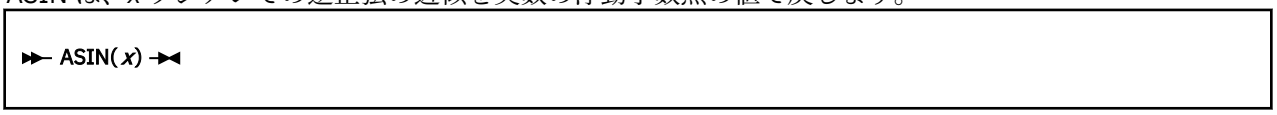
## ANY

ANY は、*x* の任意のエレメントの対応するビットがあり、それが 1 の場合は、各ビットが 1 のビット・ストリングを戻します。 結果の長さは、最も長いエレメントの長さと等しくなります。

**x**
計算配列式。 *x* がビット・ストリング配列ではない場合、*x* はビット・ストリング配列に変換されます。

## ASIN

ASIN は、*x* ラジアンでの逆正弦の近似を実数の浮動小数点の値で戻します。




**x**
実数式。ABS(x) <= 1

結果の範囲は以下のとおりです。

```
-π/2 ≤ ASIN(x) ≤ π/2
```

結果は、*x* の基数および精度になります。

## ATAN

ATAN は、*x* ラジアンまたは比率 *x/y* の逆正接の近似を 浮動小数点の値で戻します。

**x および y**
式。

*x* が単独で指定されている場合は、結果は *x* の基数と精 度になり、次の範囲内の値です。

```
-π/2 < ATAN(x) < π/2
```

*x* と *y* が指定される場合は、それぞれ実数でなければなりません。 *x* と *y* が両方ともゼロの場合は、エラーになります。 *x* と *y* のその他のすべての値の結果には、長い方の引数の精 度、式の規則で決まる基数、および次の式で指定される値が含まれます。

| | |
|---|---|
| ATAN(x/y) | y>0 の場合 |
| π/2 | y=0 および x>0 の場合 |
| -π/2 | y=0 および x<0 の場合 |
| π + ATAN(x/y) | y<0 および x>=0 の場合 |
| -π + ATAN(x/y) | y<0 および x<0 の場合 |

## ATAND

ATAND は、*x* ラジアンまたは比率 *x/y* の逆正接の近似を 実数の浮動小数点の値で戻します。

**x および y**
式。

*x* が単独で指定される場合は、x は実数でなければなりません。 結果は、*x* の基数と精度になり、次の範囲内の値です。

```
-90 < ATAND(x) < 90
```

*x* と *y* が指定される場合は、それぞれ実数でなければなりません。 結果の値は、以下のようになります。

```
(180/π) * ATAN(x,y)
```

引数の要件および結果の属性については、402 ページの『ATAN』を参照してください。

## ATANH

ATANH は、*x* の基数、モード、および精度を保持する浮動小数点数で、 *x* の双曲線逆正接の近似値を戻します。

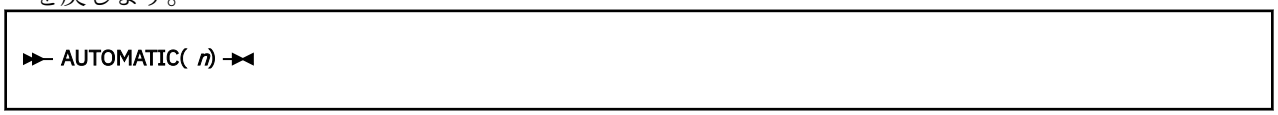
**x**
式。 ABS(x)<1

結果は、次の式で指定される値になります。

```
LOG((1 + x)/(1 - x))/2
```

## AUTOMATIC

AUTOMATIC は、自動ストレージにサイズ *n* のストレージを割り振り、割り振り済みストレージにポインターを戻します。




**省略形**: AUTO

***n***
式。 *n* は負であってはなりません。 必要に応じて、*n* はタイプ *size_t* [1] に変換されます。

獲得されたストレージを明示的に解放することはできません。ストレージは、そのブロックが終了したときに自動的に解放されます。

## AVAILABLEAREA

AVAILABLEAREA は、区域 *x* から取得可能な最大の単一割り振りのサイズを示す *size_t* [1] 値を返します。

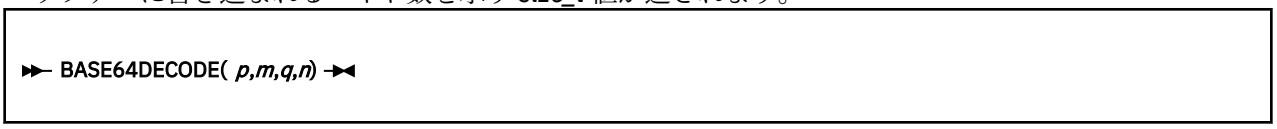
***x***
AREA 属性付き参照

**例**

```
dcl Uarea area(1000);
dcl Pz ptr;
dcl C99z char(99) varyingz based(Pz);
dcl (SizeBefore, SizeAfter) fixed bin(31);
SizeBefore = availablearea(Uarea);         /* returns 1000    */
Alloc C99z in(Uarea);
SizeAfter = availablearea(Uarea);          /* returns 896     */
dcl C9 char(896) based(Pz);
Alloc C9 in(Uarea);
```

# BASE64DECODE

BASE64DECODE は、DEFAULT コンパイラー・オプションの ASCII または EBCDIC のサブオプションで指定された文字セットでエンコードされた Base64 からソース・バッファーをデコードします。 ターゲット・バッファーに書き込まれるバイト数を示す *size_t* 値が返されます。




**p**
ターゲット・バッファーのアドレスを指定します。

**m**
ターゲット・バッファーの長さを指定します。計算タイプでなければならず、タイプ *size_t* に変換されます。

**q**
ソース・バッファーのアドレスを指定します。

**n**
ソース・バッファーの長さを指定します。計算タイプでなければならず、タイプ *size_t* に変換されます。

**注 :** 一部の引数および戻り値のタイプは *size_t* です。LP(32) コンパイラー・オプションが有効な場合、*size_t* は FIXED BIN(31) です。 LP(64) コンパイラー・オプションが有効な場合、*size_t* は FIXED BIN(63) です。

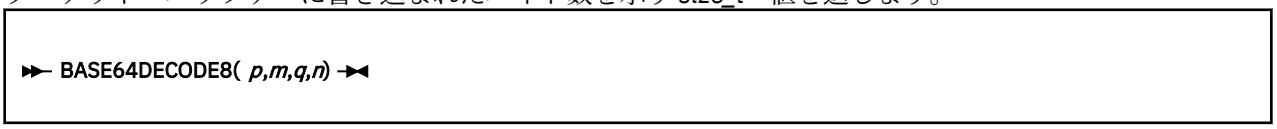
戻り値は、以下のように、ターゲット・バッファーのアドレスまたはターゲット・バッファーのサイズによって異なります。

- ターゲット・バッファー *p* のアドレスがゼロの場合は、想定される書き込みバイト数が返されます。
- ターゲット・バッファーの大きさが十分でなかった場合には、値 -1 が返されます。
- ターゲット・バッファーの大きさが十分であった場合には、バッファーに書き込まれたバイト数が返されます。

この組み込み関数は、組み込み関数 BASE64ENCODE の逆であり、BASE64ENCODE 組み込み関数で使用される規則と同じ規則を使用して Base64 ソースがエンコードされたものと予想します。 詳しくは、406 ページの『ソース・バッファーを EBCDIC として Base64 にエンコードする際の規則』を参照してください。 その他の規則が使用された場合は、結果が予測不能になります。

# BASE64DECODE8

BASE64DECODE8 は、UTF-8 としてエンコードされる Base64 からソース・バッファーをデコードします。ターゲット・バッファーに書き込まれたバイト数を示す *size_t*[1] 値を返します。




***p***
ターゲット・バッファーのアドレスを指定します。

***M***
ターゲット・バッファーの長さを指定します。計算タイプでなければならず、タイプ *size_t* に変換されます。

***Q***
ソース・バッファーのアドレスを指定します。

***n***
ソース・バッファーの長さを指定します。計算タイプでなければならず、タイプ *size_t* に変換されます。

ターゲット・バッファーのアドレスがゼロである場合は、書き込まれることが想定されたバイト数が返されます。 ターゲット・バッファーの大きさが十分でなかった場合には、値 -1 が返されます。 ターゲット・バッファーの大きさが十分であった場合には、バッファーに書き込まれたバイト数が返されます。

この関数は、BASE64ENCODE8 関数の逆であり、BASE64ENCODE8 関数で使用された規則と同じ規則を使用して Base64 ソースがエンコードされたことを予期しています。 詳しくは、407 ページの『ソース・バッファーを UTF-8 として Base64 にエンコードする際の規則』を参照してください。 その他の規則が使用された場合は、結果が予測不能になります。

## BASE64DECODE16

BASE64DECODE16 は、UTF-16 としてエンコードされる Base64 からソース・バッファーをデコードします。 ターゲット・バッファーに書き込まれたバイト数を示す *size_t*[1] 値を返します。

**p**
ターゲット・バッファーのアドレスを指定します。

**m**
ターゲット・バッファーの長さを指定します。計算タイプでなければならず、タイプ *size_t* に変換されます。

**q**
ソース・バッファーのアドレスを指定します。

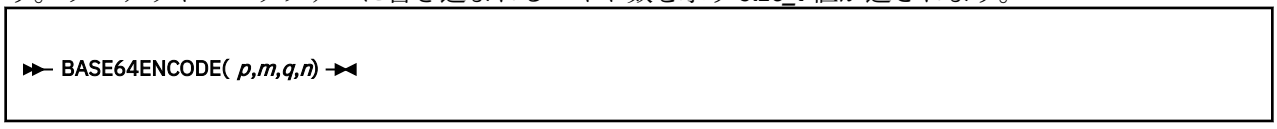
**n**
ソース・バッファーの長さを指定します。計算タイプでなければならず、タイプ *size_t* に変換されます。

ターゲット・バッファーのアドレスがゼロである場合は、書き込まれることが想定されたバイト数が返されます。 ターゲット・バッファーの大きさが十分でなかった場合には、値 -1 が返されます。 ターゲット・バッファーの大きさが十分であった場合には、バッファーに書き込まれたバイト数が返されます。

この関数は、BASE64ENCODE16 関数の逆であり、BASE64ENCODE16 関数で使用された規則と同じ規則を使用して Base64 ソースがエンコードされたことを予期しています。 詳しくは、408 ページの『ソース・バッファーを UTF-16 として Base64 にエンコードする際の規則』を参照してください。 その他の規則が使用された場合は、結果が予測不能になります。

## BASE64ENCODE

BASE64ENCODE は、DEFAULT コンパイラー・オプションの ASCII または EBCDIC のサブオプションで指定された文字セット内に Base64 値を保持しているバッファーに、ソース・バッファーをエンコードします。 ターゲット・バッファーに書き込まれるバイト数を示す *size_t* 値が返されます。




**p**
ターゲット・バッファーのアドレスを指定します。

**m**
ターゲット・バッファーの長さを指定します。計算タイプでなければならず、タイプ *size_t* に変換されます。

**q**
ソース・バッファーのアドレスを指定します。

**n**
ソース・バッファーの長さを指定します。計算タイプでなければならず、タイプ *size_t* に変換されます。

戻り値は、以下のように、ターゲット・バッファーのアドレスまたはターゲット・バッファーのサイズによって異なります。

- ターゲット・バッファー *p* のアドレスがゼロの場合は、想定される書き込みバイト数が返されます。
- ターゲット・バッファーの大きさが十分でなかった場合には、値 -1 が返されます。
- ターゲット・バッファーの大きさが十分であった場合には、バッファーに書き込まれたバイト数が返されます。

**注 :** 一部の引数および戻り値のタイプは *size_t* です。LP(32) コンパイラー・オプションが有効な場合、*size_t* は FIXED BIN(31) です。 LP(64) コンパイラー・オプションが有効な場合、*size_t* は FIXED BIN(63) です。

### ソース・バッファーを EBCDIC として Base64 にエンコードする際の規則

このエンコード方式では、以下の Base64「ディジット」セットが使用されます。

ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/

ソースは 6 ビットごとに、対応する EBCDIC「ディジット」に変換され、Base64 ストリングを形成します。 ソースのビット長が 6 の倍数でない場合は、結果の末尾に 1 つまたは 2 つの '='e 記号が必要に応じて付加されます。

ソース・バッファーはビット・ストリングとして扱われるため、ターゲット・バッファー内の結果は、ソースのコード・ページによって異なります。

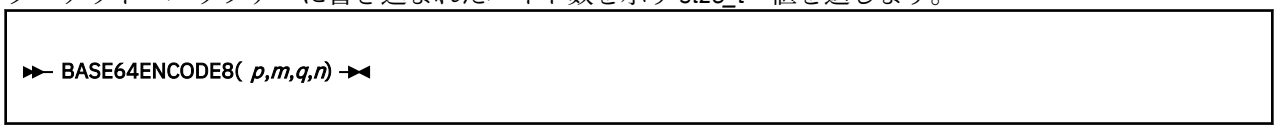
次の表に、BASE64ENCODE を使用して、EBCDIC としてエンコードされる Base64 にソース・バッファーを変換したときの、ソースとそれに対応する結果の例を示します。

表 *80.* ソース・バッファーを *EBCDIC* として *Base64* にエンコードする例

| ソースの長さ | ソースの値 | 結果の長さ | 結果の値 |
|---|---|---|---|
| 6 | 'please'A | 8 | cGxlYXNl |
| 5 | 'pleas'A | 8 | cGxlYXM= |
| 4 | 'plea'A | 8 | cGxlYQ== |

## BASE64ENCODE8

BASE64ENCODE8 は、UTF-8 としてエンコードされる Base64 にソース・バッファーをエンコードします。ターゲット・バッファーに書き込まれたバイト数を示す *size_t* [1] 値を返します。




**p**
ターゲット・バッファーのアドレスを指定します。

**m**
ターゲット・バッファーの長さを指定します。計算タイプでなければならず、タイプ *size_t* に変換されます。

**q**
ソース・バッファーのアドレスを指定します。

**n**
ソース・バッファーの長さを指定します。計算タイプでなければならず、タイプ *size_t* に変換されます。

ターゲット・バッファーのアドレスがゼロである場合は、書き込まれることが想定されたバイト数が返されます。 ターゲット・バッファーの大きさが十分でなかった場合には、値 -1 が返されます。 ターゲット・バッファーの大きさが十分であった場合には、バッファーに書き込まれたバイト数が返されます。

### ソース・バッファーを UTF-8 として Base64 にエンコードする際の規則

このエンコード方式では、以下の Base64「ディジット」セットが使用されます。

ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/

ソースは 6 ビットごとに、対応する UTF-8「ディジット」に変換され、Base64 ストリングを形成します。ソースのビット長が 6 の倍数でない場合は、結果の末尾に 1 つまたは 2 つの = 記号が必要に応じて付加されます。= 記号は UTF-8 です。

ソース・バッファーはビット・ストリングとして扱われるため、ターゲット・バッファー内の結果は、ソースのコード・ページに応じて異なります。 特に、ソースが EBCDIC である場合は、ソースが ASCII である場合の結果と異なります。

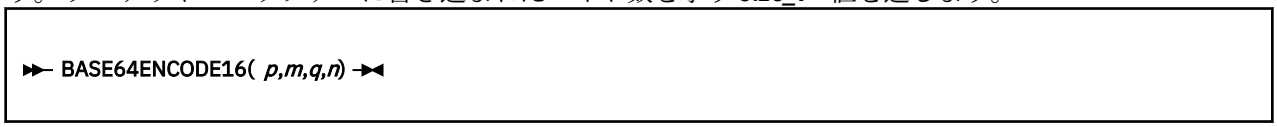
次の表に、BASE64ENCODE8 を使用して、UTF-8 としてエンコードされる Base64 にソース・バッファーを変換したときの、ソースとそれに対応する結果の例を示します。

表 *81.* ソース・バッファーを *UTF-8* として *Base64* にエンコードする例

| ソースの長さ | ソースの値 | 結果の長さ | 結果の値 |
|---|---|---|---|
| 6 | 'please'A | 8 | UTF8('cGxlYXNl') |
| 5 | 'pleas'A | 8 | UTF8('cGxlYXM=') |
| 4 | 'plea'A | 8 | UTF8('cGxlYQ==') |
| 6 | 'please'E | 8 | UTF8('l5OFgaKF') |
| 5 | 'pleas'E | 8 | UTF8('l5OFgaI=') |
| 4 | 'plea'E | 8 | UTF8('l5OFgQ==) |

## BASE64ENCODE16

BASE64ENCODE16 は、UTF-16 としてエンコードされる Base64 にソース・バッファーをエンコードします。 ターゲット・バッファーに書き込まれたバイト数を示す *size_t*[1] 値を返します。




**p**
ターゲット・バッファーのアドレスを指定します。

**m**
ターゲット・バッファーの長さを指定します。計算タイプでなければならず、タイプ *size_t* に変換されます。

**q**
ソース・バッファーのアドレスを指定します。

**n**
ソース・バッファーの長さを指定します。計算タイプでなければならず、タイプ *size_t* に変換されます。

ターゲット・バッファーのアドレスがゼロである場合は、書き込まれることが想定されたバイト数が返されます。 ターゲット・バッファーの大きさが十分でなかった場合には、値 -1 が返されます。 ターゲット・バッファーの大きさが十分であった場合には、バッファーに書き込まれたバイト数が返されます。

### ソース・バッファーを UTF-16 として Base64 にエンコードする際の規則

このエンコード方式では、以下の Base64「ディジット」セットが使用されます。

ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/

ソースは 6 ビットごとに、対応する UTF-16「ディジット」に変換され、Base64 ストリングを形成します。 ソースのビット長が 6 の倍数でない場合は、結果の末尾に 1 つまたは 2 つの = 記号が必要に応じて付加されます。= 記号は UTF-16 です。

ソース・バッファーはビット・ストリングとして扱われるため、ターゲット・バッファー内の結果は、ソースのコード・ページに応じて異なります。 特に、ソースが EBCDIC である場合は、ソースが ASCII である場合の結果と異なります。

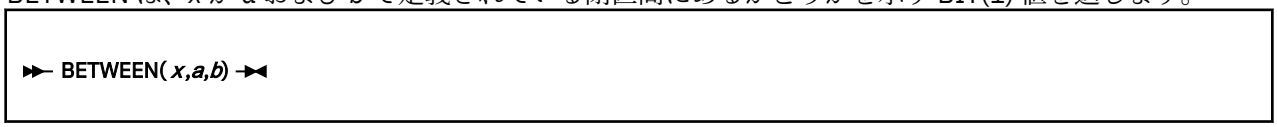
次の表に、BASE64ENCODE16 を使用して、ソース・バッファーを UTF-16 としてエンコードされる Base64 に変換したときの、ソースとそれに対応する結果の例を示します。

表 *82.* ソース・バッファーを *UTF-16* として *Base64* にエンコードする例

| ソースの長さ | ソースの値 | 結果の長さ | 結果の値 |
|---|---|---|---|
| 6 | 'please'A | 16 | WCHAR('cGxlYXNl') |
| 5 | 'pleas'A | 16 | WCHAR('cGxlYXM=') |
| 4 | 'plea'A | 16 | WCHAR('cGxlYQ==') |
| 6 | 'please'E | 16 | WCHAR('l5OFgaKF') |
| 5 | 'pleas'E | 16 | WCHAR('l5OFgaI=') |
| 4 | 'plea'E | 16 | WCHAR('l5OFgQ==') |

## BETWEEN

BETWEEN は、*x* が *a* および *b* で定義されている閉区間にあるかどうかを示す BIT(1) 値を返します。



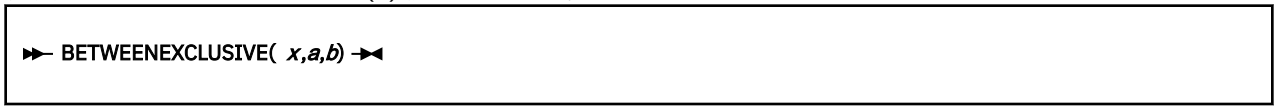

***x*、*a*、*b***
式。 すべてを同じ型の ORDINAL にするか、すべてを計算型にする必要があります。

BETWEEN(x,a,b) は、(a <= x) & (x <= b) のテストと等価です。 したがって、いずれかの引数が数値であれば、すべてが REAL でなければなりません。

BETWEEN(x, a, b) では、<= b は true でなければなりません。true でない場合、プログラムはエラーになり、その動作は未定義です。

## BETWEENEXCLUSIVE

BETWEENEXCLUSIVE は、最初の引数 *x* が、2 番目の引数 *a* と 3 番目の引数 *b* によって定義されている開区間にあるかどうかを示す bit(1) 値を返します。



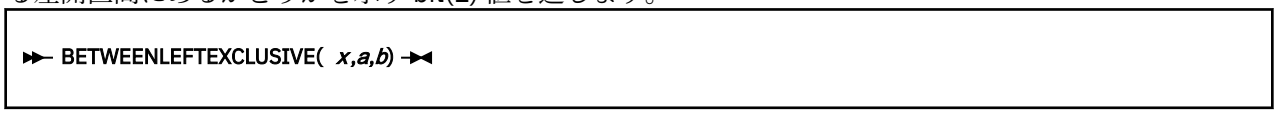

***x*、*a*、*b***
式。 すべてを同じ型の ORDINAL にするか、すべてを計算型にする必要があります。

BETWEENEXCLUSIVE(*x*,*a*,*b*) は、テスト (*a* < *x* ) & (*x* < *b*) と同等です。 したがって、いずれかの引数が数値であれば、REAL でなければなりません。

BETWEENEXCLUSIVE(*x*,*a*,*b*) では、*a* < *b* は true でなければなりません。true でない場合、プログラムはエラーになり、その動作は未定義です。

## BETWEENLEFTEXCLUSIVE

BETWEENLEFTEXCLUSIVE は、最初の引数 *x* が、2 番目の引数 *a* と 3 番目の引数 *b* によって定義されている左開区間にあるかどうかを示す bit(1) 値を返します。



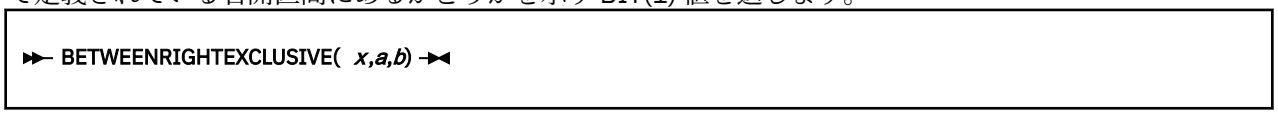

***x*、*a*、*b***
式。 すべてを同じ型の ORDINAL にするか、すべてを計算型にする必要があります。

BETWEENLEFTEXCLUSIVE(*x*,*a*,*b*) は、テスト (*a* < *x* ) & (*x* <= *b*) と同等です。 したがって、いずれかの引数が数値であれば、REAL でなければなりません。

BETWEENLEFTEXCLUSIVE(*x*,*a*,*b*) では、*a* < *b* は true でなければなりません。true でない場合、プログラムはエラーになり、その動作は未定義になります。

## BETWEENRIGHTEXCLUSIVE

BETWEENRIGHTEXCLUSIVE 組み込み関数は、最初の引数 *x* が、2 番目の引数 *a* と 3 番目の引数 *b* によって定義されている右開区間にあるかどうかを示す BIT(1) 値を返します。



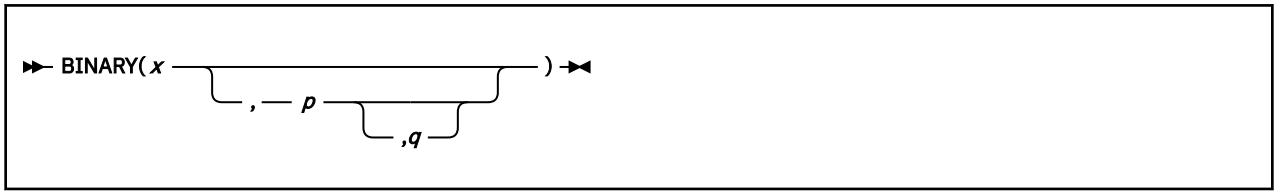

***x*、*a*、*b***
式。 すべてを同じ型の ORDINAL にするか、すべてを計算型にする必要があります。

BETWEENRIGHTEXCLUSIVE(*x*,*a*,*b*) はテスト (*a* <= *x* ) & (*x* < *b*) と同等です。 したがって、いずれかの引数が数値であれば、REAL でなければなりません。

BETWEENRIGHTEXCLUSIVE(*x*,*a*,*b*) では、*a* < *b* は true でなければなりません。true でない場合、プログラムはエラーになり、その動作は未定義です。

## BINARY

BINARY は、*p* および *q* で指定される精度で、*x* の 2 進数の値を戻します。 結果には、*x* のモードとスケールが含まれています。




**省略形**: BIN

**x**
式。

**p**
制限付き式。 演算を通して維持する桁数を指定します。実行の限界を超えることはできません。

**q**
制限付き式。 結果の位取り係数を指定します。 固定小数点の結果の場合は、*p* が指定され、*q* が省略されると、位取り係数ゼロがデフォルトになります。 結果が浮動小数点の場合は、*q* は省略します。

*p* と *q* が両方とも省略される場合は、結果の精度は基数変換の規則により決まります。

# BINARYVALUE

BINARYVALUE は、ポインター、オフセット、または序数のいずれかである *x* を整数に変換します。 この関数は、変換された値である FIXED BIN 値を返します。

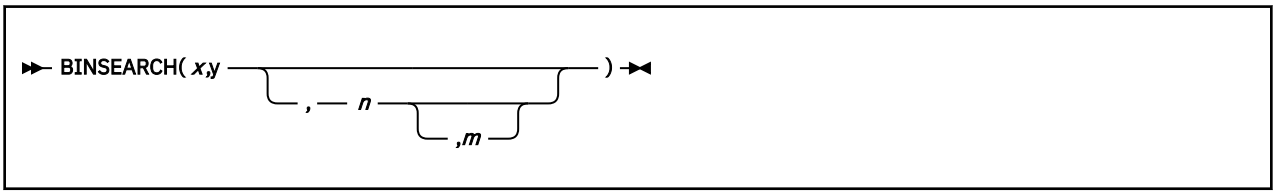
*x* がポインターである場合、戻り値のタイプは *size_t* [1] です。 *x* が序数の場合、戻り値のタイプは FIXED BIN(31) です。 *x* がオフセットの場合、戻り値のタイプは OFFSETSIZE(4) での FIXED BIN(31) および OFFSETSIZE(8) での FIXED BIN(63) です。

**省略形:** BINVALUE

**x**
式

# BINSEARCH

BINSEARCH は、単純比較を使用して、指定されたキー値の配列の二分探索を実行し、*size_t* 値を返します。




**x**
検索が行われるターゲット配列を指定する式。 *x* はスカラーの 1 次元配列であり、*x* のエレメントは昇順でなければなりません。 *x* が NONVARYING BIT の配列である場合は、位置合わせされている必要があります。

**y**
検索対象のキー値を指定する式。

**n**
調査される最初の配列エレメントの指標を指定する式。 デフォルトは LBOUND(x) になります。

**m**
調査される配列エレメントの数を指定する式。 カウントは *n* 番目から開始し、デフォルトは HBOUND(x) – n + 1 になります。

配列 *x* のエレメントとキー値は、以下のいずれかの条件を満たしている必要があります。

- 両方とも計算タイプであり、どちらも COMPLEX ではない
- 両方とも POINTER である
- 両方とも同じ構造化タイプへの HANDLE である
- 両方とも同じタイプの ORDINAL である

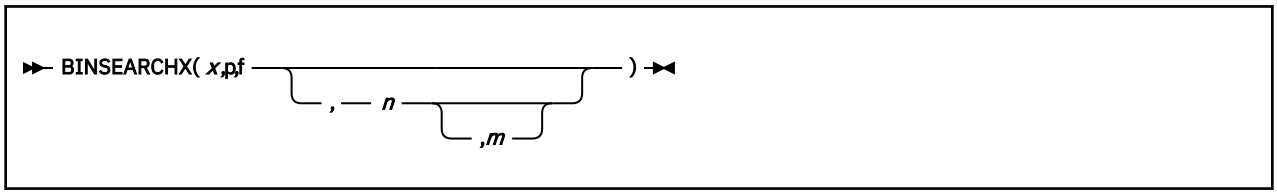
返される値は、この配列内のキー値の相対指標です。 配列内でキー値 *y* が見つからない場合、返される *size_t* 値はゼロです。

相対索引は、配列の下限が 1 の場合の索引です。 したがって、真の索引は、戻り値 + LBOUND(x) – 1 として計算されます。 以下に例を示します。

- 配列 *x* の下限が 0 で、上限が 11 の場合、戻り値は 0 から 12 までの範囲になります。 戻り値が非ゼロの場合、検出された値の真の指標は、戻り値から 1 を差し引いた値です。
- 配列 *x* の下限が -12 で、上限が 12 の場合、戻り値は 0 から 25 までの範囲になります。 戻り値が非ゼロの場合、検出された値の真の指標は、戻り値から 13 を差し引いた値です。

# BINSEARCHX

BINSEARCHX は、指定された比較関数を使用して、指定されたキー値の配列の二分探索を実行し、*size_t* 値を返します。




**x**
検索が行われるターゲット配列を指定する式。 *x* は 1 次元配列であり、*x* のエレメントは昇順でなければなりません。 *x* が NONVARYING BIT の配列である場合は、位置合わせされている必要があります。

**p**
検索対象のキー値のアドレスを指定する式。

**f**
必要なすべての比較を実行するために呼び出される関数を指定する式。

**n**
調査される最初の配列エレメントの指標を指定する式。 デフォルトは LBOUND(x) になります。

**m**
調査される配列エレメントの数を指定する式。 カウントは *n* 番目から開始し、デフォルトは HBOUND(x) – n + 1 になります。

関数 *f* は、OPTLINK リンケージを持っている必要があり、次の 2 つの POINTER BYVALUE 引数が渡されます。

- 配列エレメントのアドレス。
- 検索対象のキー値 (*p* の値) のアドレス。

関数 *f* は、属性 RETURNS( BYVALUE FIXED BINARY(31) ) を持ち、以下のように、値 -1 、0、または +1 のいずれかを返す必要があります。

- 配列エレメントの値がキー・エレメントの値より小さい場合、戻り値は -1 でなければなりません。
- 配列エレメントの値がキー・エレメントの値に等しい場合、戻り値は 0 でなければなりません。
- 配列エレメントの値がキー・エレメントの値より大きい場合、戻り値は +1 でなければなりません。

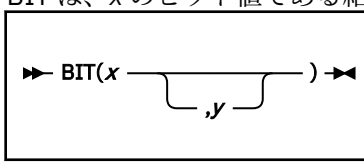
BINSEARCHX 組み込み関数自体によって返される値は、この配列内のキー値の相対指標です。 配列内でキー値 *y* が見つからない場合、返される *size_t* 値はゼロです。

相対索引は、配列の下限が 1 の場合の索引です。 したがって、真の索引は、戻り値 + LBOUND(x) – 1 として計算されます。 以下に例を示します。

- 配列 *x* の下限が 0 で、上限が 11 の場合、戻り値は 0 から 12 までの範囲になります。 戻り値が非ゼロの場合、検出された値の真の指標は、戻り値から 1 を差し引いた値です。
- 配列 *x* の下限が -12 で、上限が 12 の場合、戻り値は 0 から 25 までの範囲になります。 戻り値が非ゼロの場合、検出された値の真の指標は、戻り値から 13 を差し引いた値です。

# ビット

BIT は、*x* のビット値である結果を戻し、*y* で指定される長さをとります。

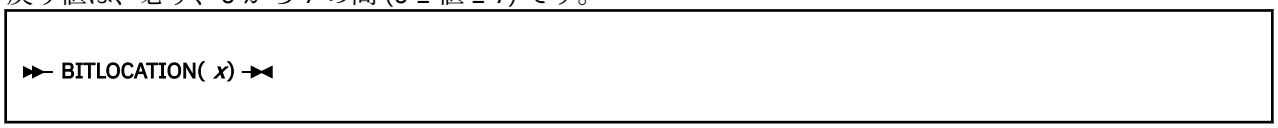



**x**
式。

**y**
式。 必要に応じて、*y* は実数の固定小数点 2 進数値に変換されます。 *y* が省略される場合は、長さはタイプ変換の規則によって決められます。 *y* = 0 の場合は、結果はヌルのビット・ストリングです。 *y* は負であってはなりません。

## BITLOCATION

BITLOCATION は、*x* を含むバイト内で、ビット *x* の位置で ある FIXED BINARY(31 ,0) の結果を戻します。 戻り値は、必ず、0 から 7 の間 (0 ≤ 値 ≤ 7) です。




**省略形**: BITLOC

**x**
位置合わせされていないビット・タイプの参照。 *x* に位置合わせがされていないビット・タイプがある場合は、値 0 が戻されます。

*x* は、添え字であってはいけません。

BITLOCATION は、制限付きの式で使用できます。制限は次のとおりです。 *BITLOC(x) が

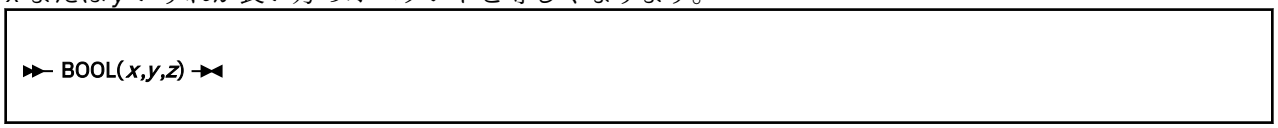
- 定数エクステントを保持する必要がある変数 *y* のエクステント
- または、定数値を保持する必要がある変数 *y* の値

BITLOC(x) を使用して上記の制限を設定する場合は、*y* の前に *x* を宣言する 必要があります。

例については、467 ページの『LOCATION』 を参照してください。

## BOOL

BOOL は、*x* および *y* に対するブール演算 *z* の結果であるビット・ストリングを返します。 結果の長さは、*x* または *y* いずれか長い方のオペランドと等しくなります。




**x および y**
式。 必要な場合には、*x* および *y* は、ビット・ストリングに変 換されます。 *x* および *y* の長さが異なる場合は、長さが同じになるよう に短い方の右側がゼロで埋められます。

**z**
式。 必要な場合には、*z* は、長さ 4 のビット・ストリングに変換されます。 *x* からのビットが *y* からのビットと一致するときは、結果 の対応するビットは *z* のビットによって次のように選択されます。

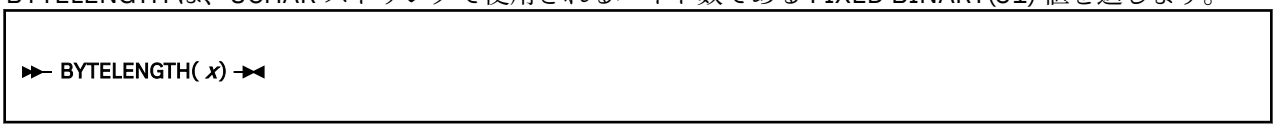
| x | y | 結果 |
|---|---|---|
| 0 | 0 | z のビット 1 |
| 0 | 1 | z のビット 2 |
| 1 | 0 | z のビット 3 |
| 1 | 1 | z のビット 4 |

## バイト

BYTE は、CHARVAL の同義語です。
**関連情報**
417 ページの『CHARVAL』
CHARVAL は、整数に対応する CHARACTER(1) 値を戻します。

# BYTELENGTH

BYTELENGTH は、UCHAR ストリングで使用されるバイト数である FIXED BINARY(31) 値を返します。




**x**
式。 *x* は、UCHAR タイプを保持する必要があります。

*x* が UCHAR VARYING または UCHAR VARYING4 タイプを保持している場合、BYTELENGTH(*x*) により返される値は先頭のバイトの数をカウントしません。 *x* が UCHAR VARYINGZ タイプを保持している場合、BYTELENGTH(*x*) により返される値は終了のヌル・バイトをカウントしません。

BYTELENGTH(*x*) によって返される値は、LENGTH(*x*) によって返される値より常に大きくなりますが、LENGTH(*x*) によって返される値の 4 倍以下になります。

**例 1:**

DCL X UCHAR(1) とし、LENGTH(X) = 1 および STG(X) = 4 とした場合でも、次のようになります。

```
X = 'A';
BYTELENGTH(X) = 1 (since X holds '41'ux )
X = 'Ä';
BYTELENGTH(X) = 2 (since X holds  'C3_84'ux )
X = '€';
BYTELENGTH(x) = 3 (since X holds 'E2_82_AC'ux )
```

**例 2:**

DCL X UCHAR(6) とし、LENGTH(X) = 6 および STG(X) = 24 とした場合でも、次のようになります。

```
X = 'Straße' ;
BYTELENGTH(X) = 7 (since X holds '53_74_72_C39F_61_65'ux )
```

**例 3:**

DCL X UCHAR(8) VARYING とし、STG(X) = 34 とした場合でも、次のようになります。

```
X = 'Straße' ;
LENGTH(X) = 6
BYTELENGTH(X) = 7 (since X holds '53_74_72_C39F_61_65'ux )
CSTG(X) = 9
```

# CDS

CDS は、ダブル比較およびスワップ の以前の値と現行値が 等しいかどうかを示す FIXED BINARY(31) 値を戻します。

**p**
以前の FIXED BINARY(63) 値のアドレス。

**q**
現行の FIXED BINARY(63) 値のアドレス。

**x**
新しい FIXED BINARY(63) 値。

CDS は、「現行」値と「以前の」値を比較します。 これらの値が等しい場合は、「現行」値を上書きして「新しい」値が コピーされ、値 0 が戻されます。 これらの値が等しくない場合は、「以前の」値を上書きして「現行」値が コピーされ、値 1 が戻されます。

z/OS 上では、CDS 組み込み関数は CDS 命令を インプリメントしています。 この関数について詳しくは、「*Principles of Operations*」の 付録を参照してください。

Intel 上では、CDS 組み込み関数は Intel cmpxchg8 命令を使用します。 同じように、CS 組み込み関数は cmpxchg4 命令を使用します。

# CEIL

CEIL は、*x* より大きいか等しい最も小さい整数値を判別し、結果にこの値を割り当てます。

**x**
実数式。

結果は、*x* のモード、スケール、および精度になります。 ただし *x* が精度 (*p,q*) の固定小数点のとき、 結果の精度は、次の式で表されます。

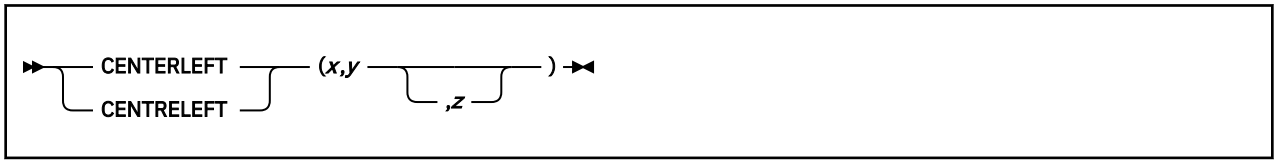
```
(min(N,max(p-q+1,1)),0)
```

この場合 N は、使用できる最大の桁数です。

# CENTERLEFT

CENTERLEFT は、長さ *y* で必要な場合に左側と右側が文字 *z* で埋められたストリング の中央 (または中央から 1 つ左よりの位置) に、ストリング *x* を挿入した結果の ストリングを戻します。

*z* の値の指定は、オプションです。




**省略形:** CENTER

**x**
文字に変換される式。

**y**
FIXED BINARY(31 ,0) に変換される式。

**z**
オプションの式。 *z* が指定された場合は、CHARACTER(1) NONVARYING タイプでなければなりません。

**例**

```
dcl Source char value('Feel the Power');
dcl Target20 char(20);
dcl Target21 char(21);

Target20 = center (Source, length(Target20), '*');
           /* '***Feel the Power***' - exactly centered */


Target21 = center (Source, length(Target21), '*');
           /* '***Feel the Power****' - leaning left! */
```

*z* が省略される場合は、埋め込み文字にブランクが使われます。

# CENTRELEFT

CENTRELEFT は、CENTERLEFT の同義語です。

**省略形**: CENTRE

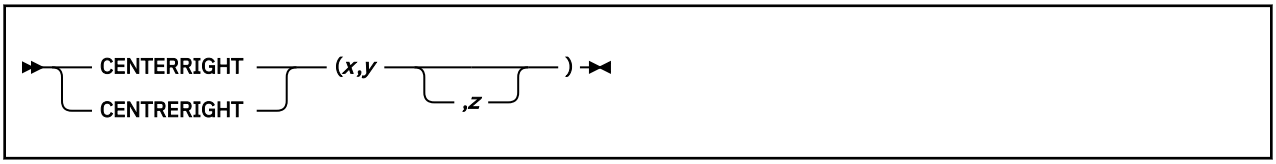
**関連情報**

414 ページの『CENTERLEFT』
CENTERLEFT は、長さ *y* で必要な場合に左側と右側が文字 *z* で埋められたストリング の中央 (または中央から 1 つ左よりの位置) に、ストリング *x* を挿入した結果の ストリングを戻します。

## CENTERRIGHT

CENTERRIGHT は、長さ *y* で必要な場合に左側と右側が文字 *z* で埋められた ストリングの中央 (または中央より 1 つ右よりの位置) に、ストリング *x* を挿入した結果のストリングを戻します。

*z* の値の指定は、オプションです。




**x**
文字に変換される式。

**y**
FIXED BINARY(31 ,0) に変換される式。

**z**
オプションの式。 *z* が指定された場合は、CHARACTER(1) NONVARYING タイプでなければなりません。

**例**

```
dcl Source char value('Feel the Power');
dcl Target20 char(20);
dcl Target21 char(21);

Target20 = centerright (Source, length(Target20), '*');
           /* '***Feel the Power***' - exactly centered */

Target21 = centerright (Source, length(Target21), '*');
           /* '****Feel the Power***' - leaning right! */
```

*z* が省略される場合は、埋め込み文字にブランクが使われます。

## CENTRERIGHT

CENTRERIGHT は、CENTERRIGHT の同義語です。

**関連情報**

415 ページの『CENTERRIGHT』
CENTERRIGHT は、長さ *y* で必要な場合に左側と右側が文字 *z* で埋められた ストリングの中央 (または中央より 1 つ右よりの位置) に、ストリング *x* を挿入した結果のストリングを戻します。

## CHARACTER

CHARACTER は、*y* で指定された長さの *x* の文字値を返します。 CHARACTER は、グラフィック・タイプから文字タイプへの変換もサポートします。

**省略形**: CHAR

**x**
式。

*n* は、計算タイプでなくてはなりません。

*x* がグラフィックでない場合は、CHARACTER は文字に変換された *x* を 戻します。

*x* が GRAPHIC のときは、CHARACTER は SBCS 文字に変換された *x* を戻します。 DBCS 文字が SBCS 等価なものに変換できない場合は、CONVERSION 条件が発生 します。

*x* の値は、チェックされません。

**y**

式。 必要に応じて、y は実数の固定小数点 2 進数値に変換されます。

*y* が省略される場合は、長さはタイプ変換の規則によって決められます。

*y* は負であってはなりません。

*y* = 0 の場合は、結果はヌル文字ストリングです。

**例: GRAPHIC から文字への変換**

```
dcl X graphic(6);
dcl A char (6);
A = char(X);
```

| X の値 | 中間結果 | A に割り当てられる値 |
|---|---|---|
| .A.B.C.D.E.F | ABCDEF | ABCDEF |

# CHARGRAPHIC

CHARGRAPHIC は、GRAPHIC (DBCS) ストリング *x* を *y* で指定された長さで混合文字ストリングに変換します。

**省略形**: CHARG

**x**

式。

*x* は、GRAPHIC ストリングでなければなりません。

**y**

式。 必要に応じて、*y* は実数の固定小数点 2 進数値に変換されます。

*y* が省略される場合は、長さはタイプ変換の規則によって決められます。

*y* は負であってはなりません。

CHARGRAPHIC は、*x* から変換された混合文字ストリングを返します。

次の規則が適用されます。

- *y* = 0 の場合は、結果はヌル文字ストリングです。
- *y* = 1 の場合は、結果は 1 ブランクの文字ストリングです。
- *y* が文字ストリングを入れるのに必要な長さより大きい場合は、 結果は SBCS ブランクで埋まります。
- *y* が文字ストリングを入れるのに必要な長さより小さい場合は、結果が切り捨てられます。 GRAPHIC の後ろで切り捨て、必要ならば SBCS のブランクを追加してストリングの長さを 完全にすることによって、保全性が保持されます。

**例 1**

この例は、グラフィックから文字への変換を示しています。 *y* は結果を収容するために十分な長さです。

```
dcl X graphic(6);
dcl A char (12);
A = char(X,12);
```

| X の値 | 中間結果 | A に割り当てられる値 |
|---|---|---|
| .A.B.C.D.E.F | .A.B.C.D.E.F | .A.B.C.D.E.F |

**例 2**

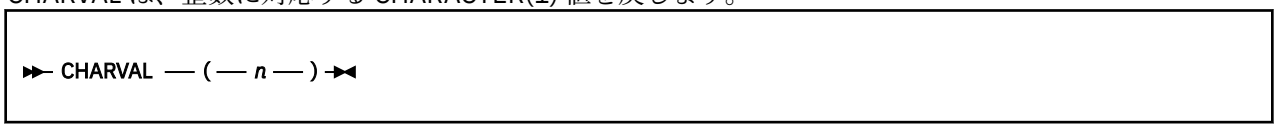
この例は、グラフィックから文字への変換を示しています。 ただし、*y* は結果を収容するためには短すぎます。

```
dcl X graphic(6);
dcl A char (12);
A = char(X,11);
```

| X の値 | 中間結果 | A に割り当てられる値 |
|---|---|---|
| .A.B.C.D.E.F | .A.B.C.D.E.F | .A.B.C.D.Eb |

## CHARVAL

CHARVAL は、整数に対応する CHARACTER(1) 値を戻します。



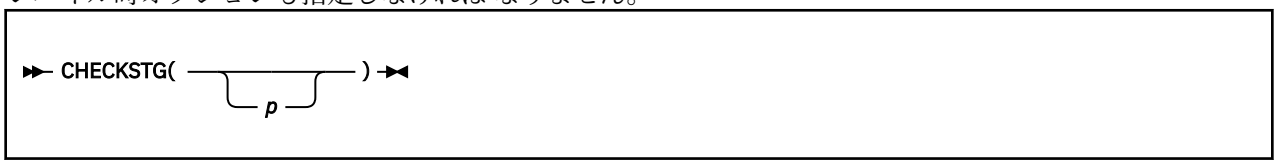

**n**
式。必要な場合には、UNSIGNED FIXED BIN(8) に変換されます。

CHARVAL(n) のビット値は n と同じですが (つまり、 UNSPEC(CHARVAL(n)) は UNSPEC(n) と等しい)、属性は CHARACTER(1) です。

CHARVAL は RANK の逆です (文字に適用した場合)。

## CHECKSTG

CHECKSTG は、指定されたポインター値が、割り振られた正常なストレージの開始点であるかどうかを示すビット (1) 値を戻します。 ポインター値が与えられていない場合、CHECKSTG は、すべての割り振り済みストレージが 正しいかどうかを判別します。 この組み込み関数を使用するには、CHECK(STORAGE) コンパイル時オプションも指定しなければ なりません。



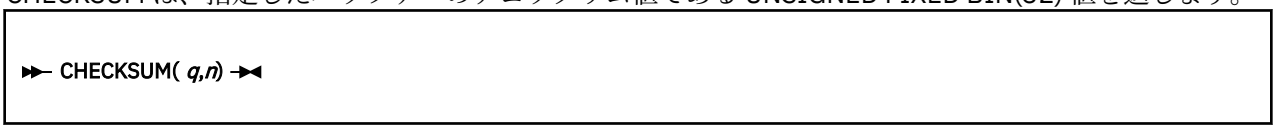

**p**
ポインターの式。

割り振りが行われる場合、その後ろに、'ff'x に設定される余分の 8 バイトが続きます。 それらのバイトが変更されていない場合、割り振りは、正しい と見なされます。

ポインターの式は、BASED 変数に割り振られたストレージを示さなければなりません。

## CHECKSUM

CHECKSUM は、指定したバッファーのチェックサム値である UNSIGNED FIXED BIN(32) 値を返します。

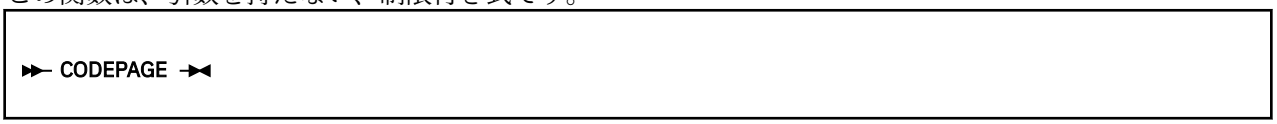



***Q***
ソース・バッファーのアドレスを指定します。

***n***
ソース・バッファーの長さを指定します。 計算タイプでなければならず、タイプ *size_t* に変換されます。[1]

## CODEPAGE

CODEPAGE は、CODEPAGE コンパイラー・オプションの値を保持する FIXED BINARY(31) 値を返します。この関数は、引数を持たない、制限付き式です。




## COLLATE

COLLATE は、256 の可能な CHARACTER(1) 値を各照合命令で 1 回比較する CHARACTER(256) ストリングを戻します。

## COLLAPSE

COLLAPSE は、オプションで指定された位置から始めて、1 つの文字のすべての複数オカレンスを 1 つのオカレンスに削減したストリングを返します。 その文字の先行インスタンスと末尾インスタンスもトリムされます。

**x**
ストリング式。 *x* は、*y* によって定義された文字のすべての複数オカレンスが 1 つのオカレンスに削減されるストリングを指定します。 *x* は、CHARACTER 属性を持っている必要があります。

**y**
式。 *y* は、タイプ CHARACTER(1) NONVARYING でなければなりません。 *y* の先行インスタンスと末尾インスタンスもトリムされます。

**n**
式。 *n* は、*y* の最初の出現の検索を開始する、*x* 内の位置を指定します。

*n* は計算タイプでなければならず、タイプ *size_t* に変換されます。 *n* のデフォルトは 1 です。

- *n* < 1 の場合、デフォルト値 1 が使用されます。
- *n* > length(*x*) の場合、完全ストリング *x* が返されます。

**例**

```
dcl s1 char value( ' abc  :  def   gh  ' );
dcl s  char(20);

s = collapse(s1, ' ', 1);
      /* 'abc : def gh ' */
s = collapse(s1, ' ', 2);
      /* ' abc : def gh ' */
s = collapse(s1, ' ', index(s1,':'));
      /* ' abc  : def gh ' */
```

# COMPARE

COMPARE は、アドレス *x* および *y* にある 2 つのバッファーの *z* バイトを比較します。

COMPARE は、FIXED BINARY(31 ,0) 値を戻します。 値は以下のいずれかです。

**ゼロ**
アドレス *x* および *y* にある *z* バイトは同一です。

**マイナス**
*x* にある *z* バイトは *y* にあるそれを下回っています。

**プラス**
*x* にある *z* バイトは *y* にあるそれを上回っています。

**x および y**
式。 両方に POINTER タイプまたは OFFSET タイプを保持していなければなりません。 OFFSET の場合は、式を AREA 修飾を使用して宣言する必要があります。

**z**
式。 *size_t* [1] に変換されます。

2 つのバッファーが異なる場合、COMPARE 組み込み関数はその差異がどこにあるかを示しません。 それらの相違点を知りたい場合は、代わりに WHEREDIFF 組み込み関数を使用してください。

**例**

```
dcl Result fixed bin;
dcl 1 Str1,
      2 B fixed bin(31),
      2 C pointer,
      2 * union,
        3 D char(4),
        3 E fixed bin(31),
        3 *,
          4 * char(3),
          4 F fixed bin(8) unsigned,
      2 * char(0);
dcl 1 Template nonasgn static,
      2 * fixed bin(31) init(16),     /* ''X */
      2 * pointer init(sysnull()),
      2 * char(4) init(''),
      2 * char(0);

call plimove(addr(Str1), addr(Template), stg(Str1));
Result = compare(addr(Str1), addr(Template), stg(Str1));   /*  0 */
D = 'DSA ';
Result = compare(addr(Str1), addr(Template), stg(Str1));   /*  1 */
B = 15;      /* '00000F00'X */
D = 'DSA ';
Result = compare(addr(Str1), addr(Template), stg(Str1));   /* -1 */
```

# COMPLEX

COMPLEX は、複合値 *x* + *y* を戻します。

**省略形**: CPLX

**x および y**
実数式。

*x* および *y* の基数が異なる場合は、10 進数の引数は 2 進 数に変換されます。 スケールが異なる場合は、固定小数点の引数は浮動小数点に変換されます。 結果は、共通の基数およびスケールになります。

固定小数点の場合は、結果の精度は次の式で指定されます。

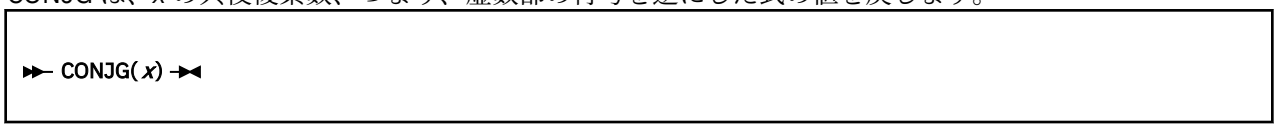
```
(min(N,max(p1-q1,p2-q2)+max(q1,q2)),max(q1,q2))
```

上記の場合、(p1,q1) と (p2,q2) は、それぞれ *x* と *y* の精度であり、 N は許可される最大の桁数です。

必要な変換が行われたあとで、引数が浮動小数点の場合には、結果には長い方の引数の 精度になります。

# CONJG

CONJG は、*x* の共役複素数、つまり、虚数部の符号を逆にした式の値を戻します。

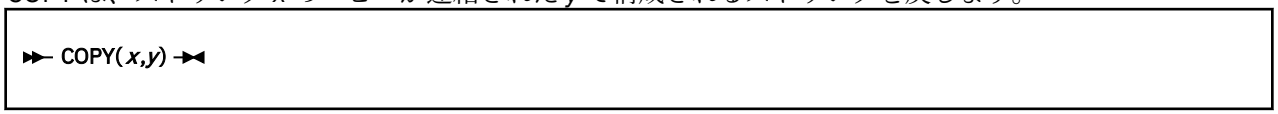



**x**
式。

*x* が実数の場合は、x は複素数に変換されます。 結果は、*x* の基数、スケール、モードおよび精度になります。

# COPY

COPY は、ストリング *x* のコピーが連結された *y* で構成されるストリングを戻します。




**x**
式。

*x* に、計算タイプ (必須) とストリング・タイプを保持している必要があります。 そうでないと、文字に変換されます。

**y**
負以外の値を使用した整数式。 繰り返し回数を指定します。 計算タイプでなければならず、FIXED BINARY(31 ,0) に変換されます。

*y* がゼロの場合は、結果はヌル・ストリングです。

**例**

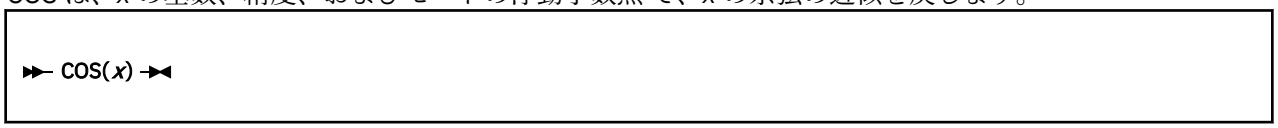
以下のコードを例として示します。

```
copy('Walla  ',1)         /*  returns 'Walla  '  */
repeat('Walla  ',1)       /*  returns 'Walla  Walla  '  */
```

この例では、`repeat(x,n)` は `copy(x,n+1)` と同等です。

## COS

COS は、*x* の基数、精度、およびモードの浮動小数点 で、*x* の余弦の近似を戻します。




**x**
ラジアン値を使用した式。

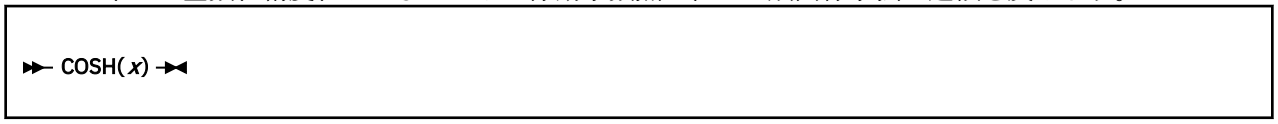
## COSD

COSD は、*x* の基数および精度の浮動小数点で、 *x* の余弦の近似を戻します。

**x**
度の値の実数式。

## COSH

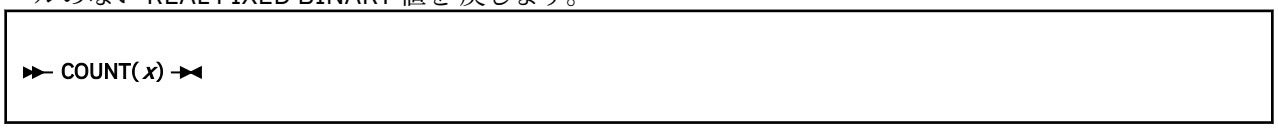
COSH は、*x* の基数、精度、およびモードの浮動小数点で、 *x* の双曲線余弦の近似を戻します。




**x**
式。

## COUNT

COUNT は、*x* に対する最新の GET 命令または PUT 命令での、 伝送されるデータ項目の数を指定するスケールのない REAL FIXED BINARY 値を 戻します。




**x**
ファイル参照。 ファイルは、オープン状態で、STREAM 属性を持っている必要があります。

*x* に対する GET 命令または PUT 命令の伝送項目のカウントは、最初の データ項目が伝送される前にゼロに初期化され、リストの各データ項目が伝送 された後で 1 ずつ増加されます。 *x* が、現行プログラムでオープンされていない場合は、ゼロが戻され ます。

GET 命令または PUT 命令時に ON ユニットまたはプロシージャーに入り、 ON ユニットまたはプロシージャー内で、*x* に対して GET 命令または PUT 命令が 実行される場合には、COUNT の値は新しい命令を実行するためにリセットされ、 オリジナルの GET 命令または PUT 命令がカウントされるときに復元されます。

BIFPREC コンパイラー・オプションによって、戻される結果の精度が決まります。

## CS

CS は、比較およびスワップ の以前の値と現行値が 等しいかどうかを示す FIXED BINARY(31) 値を戻します。

**p**
以前の FIXED BINARY(31) 値のアドレス。

**q**
現行の FIXED BINARY(31) 値のアドレス。

**x**
新しい FIXED BINARY(31) 値。

CS は、「現行」値と「以前の」値を比較します。 これらの値が等しい場合は、「現行」値を上書きして「新しい」値が コピーされ、値 0 が戻されます。 これらの値が等しくない場合は、「以前の」値を上書きして「現行」値が コピーされ、値 1 が戻されます。

したがって、CS は次の PL/I 関数としてインプリメントすることができますが、その場合はアトミックではなくなります:

```
  cs: proc( old_Addr, current_Addr, new )
   returns( fixed bin(31) byvalue )
   options( byvalue );

   dcl old_Addr     pointer;
   dcl current_Addr pointer;
   dcl new          fixed bin(31);

   dcl old          fixed bin(31) based(old_addr);
   dcl current      fixed bin(31) based(current_addr);

   if current = old then
     do;
       current = new;
       return( 0 );
     end;
   else
     do;
       old = current;
       return( 1 );
     end;
 end;
```

z/OS 上では、CS 組み込み関数は CS 命令を インプリメントしています。 この関数について詳しくは、「*Principles of Operations*」の 付録を参照してください。

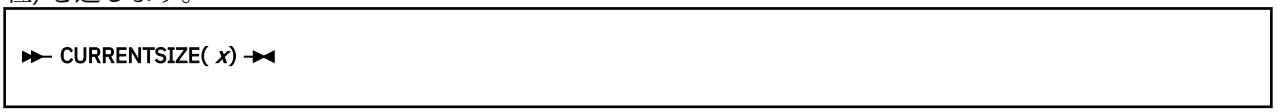
Intel 上では、CDS 組み込み関数は cmpxchg4 命令を使用します。 cmpxchg4 関数は、「現行」値、「新しい」値、および「以前の」値 のアドレスを受け取ります。 関数は元の「現行」値を戻し、「現行」値が「以前の」値と等しい場合だけ、 「現行」値を「新しい」値に更新します。

このため Intel 上では、CS 組み込み関数は 次のインライン関数によってインプリメントされます。

```
  cs: proc( old_Addr, current_Addr, new )
   returns( fixed bin(31) byvalue )
   options( byvalue );

   dcl old_Addr     pointer;
   dcl current_Addr pointer;
   dcl new          fixed bin(31);

   dcl old          fixed bin(31) based(old_addr);
   dcl current      fixed bin(31) based(current_addr);

   if cmpxchg4( current_Addr, new, old ) = old then
     do;
       return( 0 );
     end;
   else
     do;
       old = current;
       return( 1 );
     end;
 end;
```

# CURRENTSIZE

CURRENTSIZE は、*x* によって要求される、実装環境定義ストレージを提供する FIXED BIN 値 (バイト単位) を返します。




**x**
任意のデータ・タイプ、データ編成、およびストレージ・クラスの変数。 ただし、下のリストにあるものは除きます。

- 位置合わせされていない固定長ビット・ストリングである BASED 変数、DEFINED 変数、パラメーター変数、添え字付き変数、構造変数、または共用体基底付き変数。
- 最初または最後の基本エレメントが位置合わせされていない固定長の ビット・ストリングである小構造体または共用体 (ただし、その小構造体を含んでいる 大構造体または共用体の最初または最後のエレメントになっている場合を除く)。
- BASED 属性、DEFINED 属性、またはパラメーター属性を持つ大構造体か共用体、またはパラメーターである大構造体か 共用体で、位置合わせされていない固定長のビット・ストリングを最初または最後のエレメントとするもの。
- 連結ストレージにない変数。

CURRENTSIZE(x) によって戻される値は、次の状況で伝送されるバイト数として定義されます。

```
declare F file record output
        environment(scalarvarying);
write file(F) from(S);
```

*x* がスカラー可変長ストリングの場合は、戻り値にはストリングの長 さの接頭辞と現在使われているバイト数が含まれます。 <synph><var>x</var></synph> には、ストリングの未使用のバイト数は含まれません。

*x* がスカラー区域の場合は、戻り値には、区域制御バイトと区域 の現在のエクステントが含まれます。 <synph><var>x</var></synph> には、区域の最後の未使用のバイト数は含まれません。

*x* が区域または可変長ストリングを含む集合の場合は、戻り値に は、区域制御バイト、区域の最大サイズ、ストリングの長さの接頭辞、およびストリング の最大長でのバイト数が含まれます。 この規則には以下の例外があります。

*x* が、非次元区域を最後のエレメントとして持つ構造体または共用体の場合は、戻り値には、区域の制御バイトおよび区域の現在のエクステントが含まれます。 <synph><var>x</var></synph> には、区域の最後での未使用のバイト数は含まれません。

変数が割り振られていない場合、CURRENTSIZE 組み込み関数を調節可能エクステントとともに、BASED 変数上で使用してはなりません。

CMPAT(V3) コンパイラー・オプションでは、CURRENTSIZE は FIXED BIN(63) 値を返します。 その他のすべての CMPAT オプションでは、FIXED BIN(31) 値を返します。

CURRENTSIZE 組み込み関数の例については、537 ページの『SIZE』を参照してください。

# CURRENTSTORAGE

CURRENTSTORAGE は、CURRENTSIZE の同義語です。

**省略形**: CSTG

**関連情報**

423 ページの『CURRENTSIZE』
CURRENTSIZE は、*x* によって要求される、実装環境定義ストレージを提供する FIXED BIN 値 (バイト単位) を返します。

## DATAFIELD

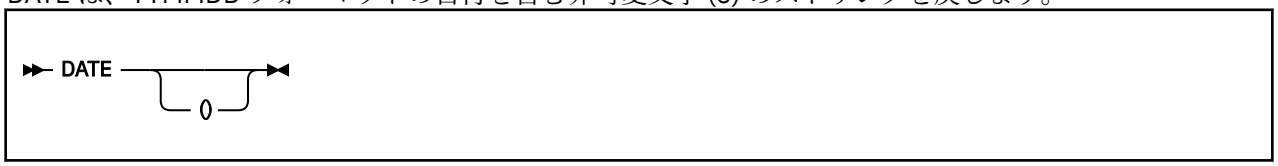
DATAFIELD は、NAME 条件の ON ユニット (またはその動的子孫) のコンテキス トにあります。 文字ストリングを返しますが、その値は条件を発生したフィールドのコンテキストです。 また、DATAFIELD も、NAME 条件の暗黙処置の一部として発生した ERROR 条件または FINISH 条件の ON ユニット (またはその動的子孫) のコンテキストにもあります。

条件が発生したストリングに DBCS ID、GRAPHIC データ、または混合文字デ ータが含まれる場合は、DATAFIELD は混合文字ストリングを戻します。

コンテキスト以外で DATAFIELD が使われている場合は、ヌル・ストリングが戻されます。

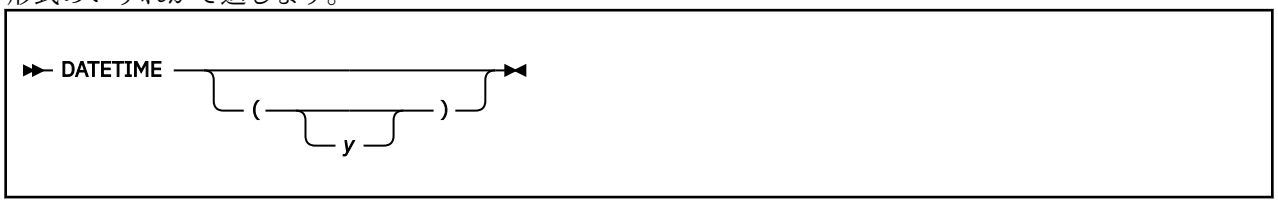
## DATE

DATE は、YYMMDD フォーマットの日付を含む非可変文字 (6) のストリングを戻します。




## DATETIME

DATETIME は、本日の日付の文字ストリング・タイム・スタンプを、デフォルト形式またはユーザー指定形式のいずれかで返します。




**y**

式

存在する場合は、戻される日付/時刻のパターンを指定します。 *y* が指定されない場合は、デフォルトの日付/時刻のパターン 「YYYYMMDDHHMISS999」が想定されます。

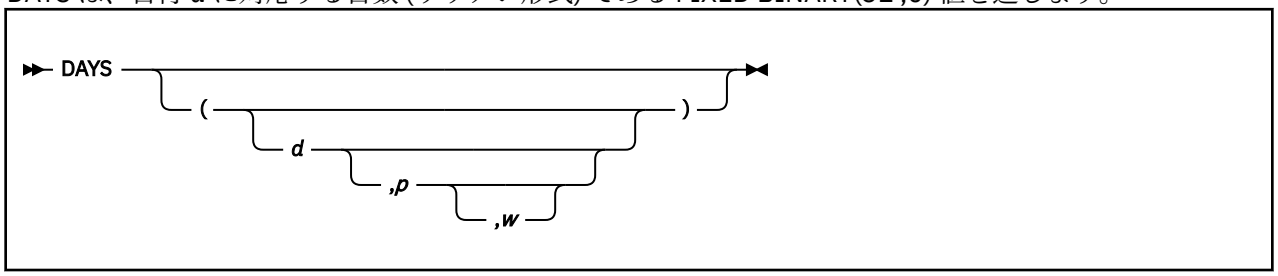
許可されるパターンについては、382 ページの表 65 を参照してください。

*y* に計算タイプ (必須) および文字タイプを保持する必要があります。 そうでないと、文字に変換されます。

DATETIME の使用例については、424 ページの『DAYS』 を参照してください。

## DAYS

DAYS は、日付 *d* に対応する日数 (リリアン形式) である FIXED BINARY(31 ,0) 値を返します。



**d**

日付を表すストリング式。 省略される場合は、DATETIME() によって戻される値が想定されます。

*d* の値に、計算タイプ (必須) および文字タイプを保持している必要があります。 そうでない場合は、*d* は文字に変換されます。

**p**

サポートされている日付/時刻パターンの 1 つ。 省略される場合は、「YYYYMMDDHHMISS999」が想定されます。

*p* に計算タイプ (必須) および文字タイプを保持する必要があります。 そうでないと、文字に変換されます。

**w**

2 桁の年フォーマットを処理するために使用されるように世紀ウィンドウを定義する整数式。

- 値が正 (1950 など) の場合、その値は年として扱われます。
- 値が負またはゼロの場合、その値は、現行のシステム提供の年から 減算されるようにオフセットを指定します。
- 省略される場合は、*w* は、WINDOW コンパイル時オプションに指定された値に デフォルトで設定されます。

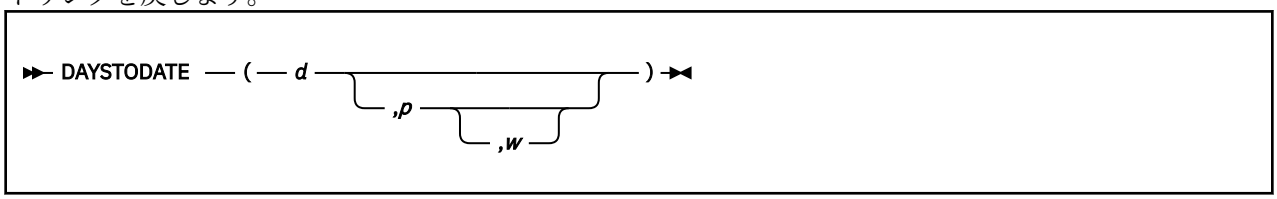
**例**

```
dcl Date_format value ('MMDDYYYY') char;
dcl Todays_date char(length(Date_format));
dcl Sep2_1993 char(length(Date_format));
dcl Days_of_July4_1993 fixed bin(31);
dcl Msg char(100) varying;
dcl Date_due char(length(Date_format));

Todays_date = datetime(date_format);          /* e.g. 06161993  */

Days_of_July4_1993 = days('07041993','MMDDYYYY');
Sep2_1993 = daystodate(days_of_July4_1993 + 60, Date_format);
             /*  09021993    */

Date_due = daystodate(days() + 60, Date_format);
        /* assuming today is July 4, 1993, this would be Sept. 2, 1993  */

Msg = 'Please pay amount due on or before ' ||
         substr(Date_due, 1, 2) || '/' ||
         substr(Date_due, 3,2)  || '/' ||
         substr(Date_due, 5);
```

許可されるパターンを 382 ページの表 65 にリストしてあります。 リリアン形式の説明については、380 ページの『日付/時刻組み込み関数』を参照してください。

## DAYSTODATE

DAYSTODATE は、*d* 日 (リリアン形式) に対応する *p* で指定されたフォーマットの日付を含む非可変文字ストリングを戻します。




**d**

日数 (リリアン形式)。 *d* は、計算タイプでなければなりません。また、必要な場合には、FIXED BINARY(31 ,0) に変換されます。

**p**

サポートされている日付/時刻パターンの 1 つ。

省略される場合、*p* は、デフォルトの日付/時刻パターン「YYYYMMDDHHMISS999」(DATETIME で返されるデフォルトのフォーマット) が想定されます。

**w**
2 桁の年フォーマットを処理するために使用されるように世紀ウィンドウを定義する整数式。

- 値が正 (1950 など) の場合、その値は年として扱われます。
- 値が負またはゼロの場合、その値は、現行のシステム提供の年から 減算されるようにオフセットを指定します。
- 省略される場合は、*w* は、WINDOW コンパイル時オプションに指定された値に デフォルトで設定されます。

許可されるパターンを 382 ページの表 65 にリストしてあります。 リリアン形式の説明については、380 ページの『日付/時刻組み込み関数』を参照してください。

DAYSTODATE の使用例については、424 ページの『DAYS』を参照してください。

## DAYSTOMICROSECS

DAYSTOMICROSECS は、日数に対応するマイクロ秒数である FIXED BINARY(63) 値を返します。

**x**
日数を指定する式。

*x* は、計算タイプでなければならず、必要に応じて FIXED BINARY(31) に変換されます。

DAYSTOMICROSECS(*x*) は、*x**(24*60*60*1_000_000) と同じです。

## DAYSTOSECS

DAYSTOSECS は、日数に対応する秒数である FLOAT BINARY(53) の値を戻します。

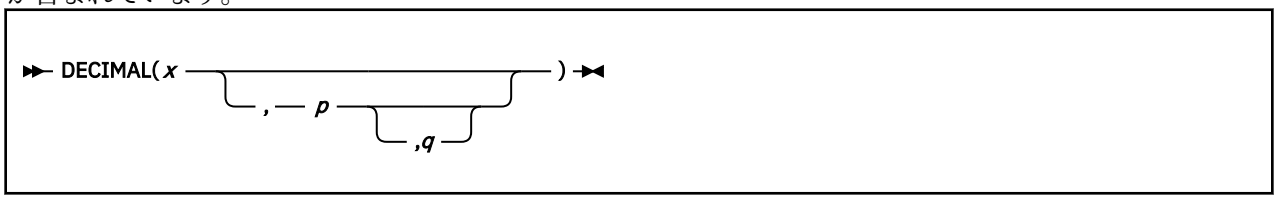
**x**
日数を指定する式。

*x* は、計算タイプでなければなりません。また、必要な場合には、 FIXED BINARY(31,0) に変換されます。

DAYSTOSECS(x) は、*x**(24*60*60) と同じです。

## DECIMAL

DECIMAL は、*p* および *q* で指定された精度で *x* の 10 進値を戻します。 結果には、*x* のモードとスケールが含まれています。




**省略形**: DEC

**x**
参照。

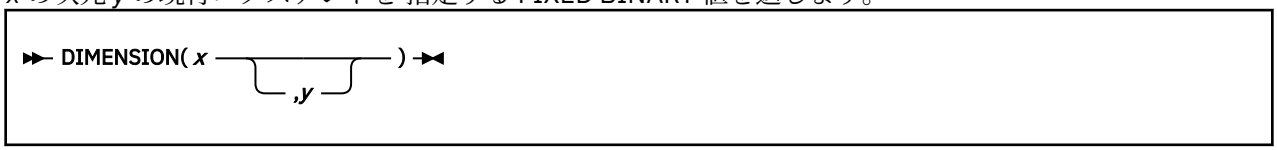
**p**
演算で維持される桁数を指定する制限付き式。

**q**
結果の位取り係数を指定する制限付き式。 結果が固定小数点の場合に、*p* が与えられ、*q* が省略されるときは、 位取り係数はゼロと見なされます。 結果が浮動小数点の場合は、*q* は省略します。

*p* と *q* が両方とも省略される場合は、結果の精度は基数変換の規則により決まります。

# DIMENSION

*x* の次元 *y* の現行エクステントを 指定する FIXED BINARY 値を返します。




**省略形**: DIM

**x**
配列参照。 *x* の次元数は、*y* よりも小さくてはなりません。

**y**
*x* の特定の次元を指定する式。 必要な場合には、*y* は FIXED BINARY(31 ,0) に変換されます。 *y* は 1 以上でなければなりません。 *y* が指定されない場合、デフォルトは 1 に設定されます。

配列が 1 次の場合のみ、*y* を省略できます。

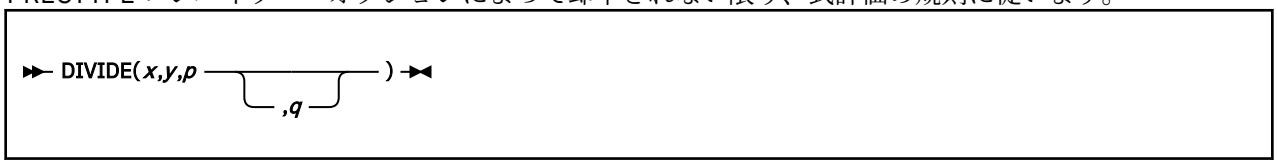
*y* が *x* の次元よりも大きい場合は、DIMENSION 関数は未定 義の値を戻します。

CMPAT(V3) コンパイラー・オプションでは、DIMENSION は FIXED BIN(63) 値を返します。 CMPAT(V2) および CMPAT(LE) の各コンパイラー・オプションでは、DIMENSION は FIXED BIN(31) 値を返します。

DIMENSION の代わりに LBOUND および HBOUND を使用することをお勧めします。

# DIVIDE

DIVIDE は、*p* および *q* で指定された精度の *x/y* の商を返します。 結果の基数、位取り、およびモードは、PRECTYPE コンパイラー・オプションによって却下されない限り、式評価の規則に従います。




**x**
式。

**y**
式。 *y* = 0 の場合は、ZERODIVIDE 条件が発生します。

**p**
演算で維持される桁数を指定する制限付き式。

**q**
結果の位取り係数を指定する制限付き式。 結果が固定小数点の場合に、*q* が省略されるときは、 位取り係数ゼロがデフォルトです。 結果が浮動小数点の場合は、*q* は省略します。

# EDIT

EDIT は、長さ LENGTH(y) の文字ストリングを戻します。 EDIT の値は、*y* で与えられたピクチャー指定を使用して宣言された変 数に *x* が指定された場合の結果と同じです。

有効なピクチャー文字については、325 ページの『第 14 章 ピクチャー指定文字』を参照してください。

**x**
式

*x* に、計算タイプを保持する必要があります。

**y**
ストリング式。

*y* に文字タイプを保持し、PICTURE データ項目に有効なピクチャー文 字を指定する必要があります。 *y* に有効なピクチャー指定がない場合は、ERROR 条件が発生します。

**例**

```
dcl pic1 char(9) init ('ZZZZZZZZ9');
dcl pic2 char(7) init ('ZZ9V.99');
dcl num fixed dec (9) init (123456789);
z = edit (num, pic1);                    /* '123456789'       */
z = edit (num, pic2);                    /*    '789.00'       */
z = edit (num, substr(pic1,8));          /*     '89'          */
z = edit (num, substr(pic2,1,5));        /*    '789.'         */
z = edit (num, substr(pic1,7,3));        /*    '789'          */
z = edit (num, substr(pic2,3,5));        /*    '9.00'         */
z = edit ('1', substr(pic1,7,3));        /*    '  1'          */
z = edit ('PL/I', 'AAXA');               /*    'PL/I'         */
z = edit ('PL/I', 'AAAA');               /* raises conversion */
```

*x* を *y* で指定したピクチャー指定に編集できない場合は、発生した条件は *x* が *y* に設定されているピクチャー指定と同じピクチャー指定が ある PICTURE データ項目に割り当てられた場合に発生する条件です。

## EMPTY

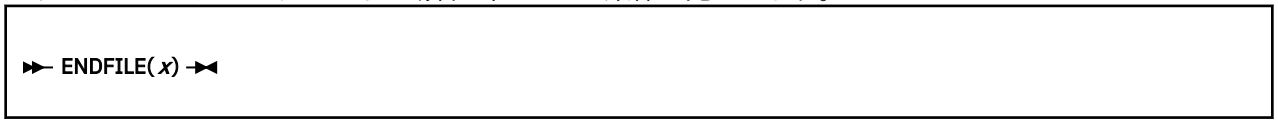
EMPTY は、ゼロ・エクステントの区域を戻します。 EMPTY は、区域のすべての割り振りを解放するのに使われます。

この関数の値は、変数が割り振られたときに、区域変数に割り当てられます。 以下に例を示します。

```
declare A area,
        I based (P),
        J based (Q);

allocate I in(A), J in (A);
A = empty();

/* Equivalent to:  free I in (A), J in (A); */
```

## ENDFILE

ENDFILE は、ファイルの終わりに到達したときに、'1'B を返します。到達しない場合は、'0'B を返します。ファイルがオープンされていない場合は、ERROR 条件が発生します。

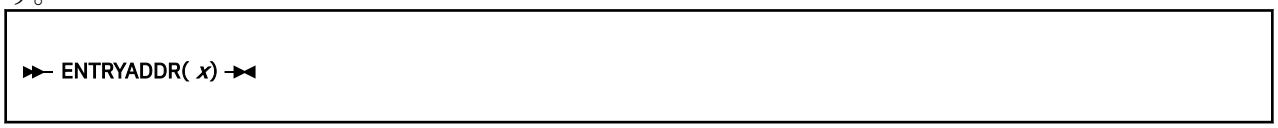



**x**
ファイル参照。

ENDFILE は、バイトストリーム・ファイル (例えば、FILEREAD 組み込み関数の使用を必要とするファイル) のファイルの終わり条件を検出するために使用できます。

## ENTRYADDR

ENTRYADDR は、エントリー参照 *x* のエントリー・ポイントのアドレスであるポインター値を戻します。これは、*x* の呼び出しに使用される AMODE 変更グルー・コードを指すポインターである可能性があります。

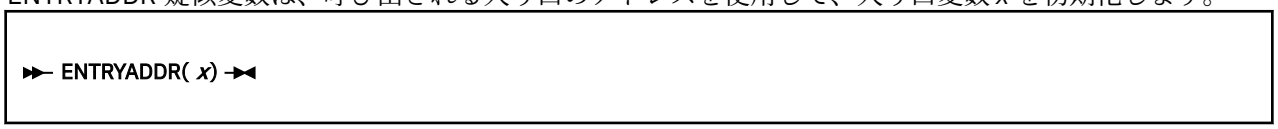



**x**
入り口参照。

*x* がフェッチ可能な入り口定数の場合は、ENTRYADDR が 実行される前に x をフェッチする必要があります。 ただし、*x* が解放された場合は、ENTRYADDR は SYSNULL を戻します。

## ENTRYADDR 疑似変数

ENTRYADDR 疑似変数は、呼び出される入り口のアドレスを使用して、入り口変数 *x* を初期化します。

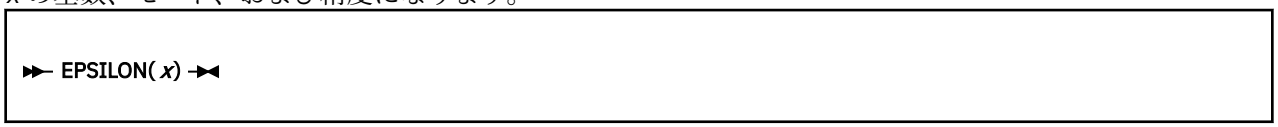



**x**
入り口参照。

**注 :** ENTRYADDR 変数に提供されたアドレスが内部プロシージャーのアドレスの場合、結果は保証されません。

## EPSILON

EPSILON は、*x* が 1 のときに、*x* と次の正の数の間のスペースである浮動小数点値を返します。 戻り値は、*x* の基数、モード、および精度になります。

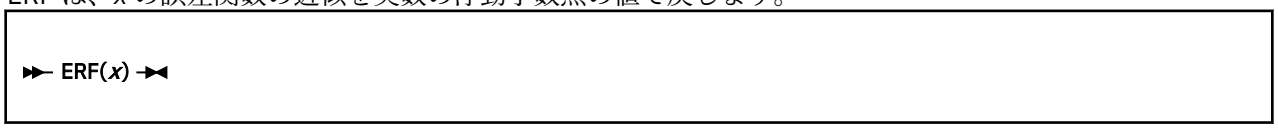



**x**
REAL FLOAT 式。

EPSILON(x) は定数であり、制限付き式に使用できます。

## ERF

ERF は、*x* の誤差関数の近似を実数の浮動小数点の値で戻します。




**x**
実数式。

結果は *x* の基数と精度であり、次の式で指定されます。

$(2/\sqrt{\pi}) \int_0^x EXP(-(t^2))dt$

## ERFC

ERFC は、*x* の誤差関数の補数の近似を実数の浮動小数点の値で戻します。

**x**
実数式。

結果は *x* の基数と精度であり、次の式で指定されます。

```
1 - ERF(x)
```

# EXP

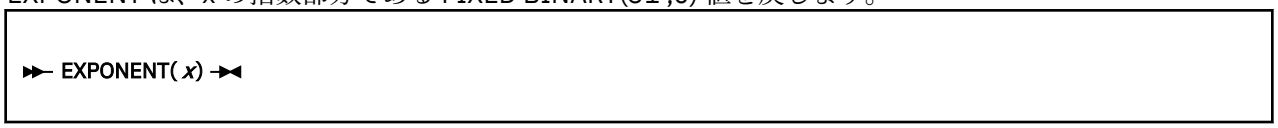
EXP は、自然対数の底 e の *x* 乗の近似を 浮動小数点の値で戻します。

**x**
式。

結果は、*x* の基数、モード、および精度になります。

# EXPONENT

EXPONENT は、x の指数部分である FIXED BINARY(31 ,0) 値を戻します。




**x**
式。 *x* は、REAL FLOAT として宣言しなければなりません。

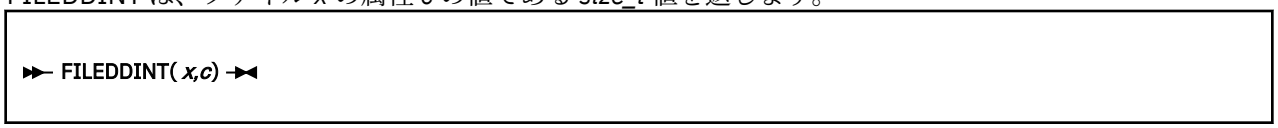
EXPONENT(x) は *x* の "数学"指数ではありません。 *x* = 0 の場合は、EXPONENT(x) = 0 です。 *x* が 0 以外の場合は、EXPONENT(x) は次のような固有の数 *e* です。

$$\text{radix}(x)^{(e-1)} \le \text{abs}(x) < \text{radix}(x)^{e}$$

したがって、EXPONENT(1e0) は 1 であり、0 ではありません。

# FILEDDINT

FILEDDINT は、ファイル *x* の属性 *c* の値である *size_t* 値を返します。




**x**
ファイル参照。

**c**
照会される属性を保持する文字ストリング。

FILEDDINT を使用している場合、以下は、*c* の有効な値です。

| | |
|---|---|
| blksize | keylen |
| bufsize | keyloc |
| delay | recsize |
| filesize | retry |

ONCODE 1010 の ERROR 条件は、ファイルがオープンしていないか、照会されている属性がファイルに無効な場合に発生します。

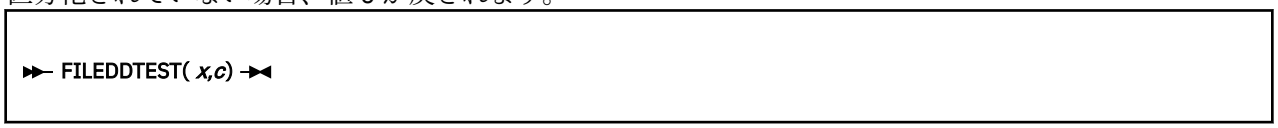
FILEDDINT(x,'BLKSIZE') は、z/OS でのみ有効です。 FILEDDINT(x,'BLKSIZE') は CONSECUTIVE ファイルのブロック・サイズを戻します。 zFS ファイルの場合は 0 を戻し、VSAM ファイルの場合にも 0 を戻します。

FILEDDINT(x,'FILESIZE') は、z/OS で、zFS ファイルの場合を除き 0 の値を戻します。

FILEDDINT(x,'KEYLOC') および FILEDDINT(x,'KEYLEN') は VSAM KSDS ファイルにのみ有効です。

## FILEDDTEST

FILEDDTEST は、属性 *c* がファイル *x* に適用される場合、値 1 を保持する FIXED BIN(31) 値を返します。区分化されていない場合、値 0 が戻されます。




**x**
ファイル参照。

**c**
照会される属性を保持する文字ストリング。

FILEDDTEST を使用している場合、以下は、*c* の有効な値です。

| | |
|---|---|
| append | graphic |
| bkwd | lrmskip |
| ctlasa | プリント |
| 区切る (delimit) | プロンプト |
| descendkey | scalarvarying |
| genkey | skip0 |

ONCODE 1010 の ERROR 条件は、ファイルがオープンしていないか、照会されている属性がファイルに無効な場合に発生します。

## FILEDDWORD

FILEDDWORD は、ファイル *x* の属性 *c* の値である文字ストリングを戻します。

**x**
ファイル参照。

**c**
照会される属性を保持する文字ストリング。

FILEDDWORD を使用する場合、*c* では以下のオプションが有効です。

| | |
|---|---|
| ACCESS | ORGANIZATION |
| AMTHD | PUTPAGE |
| ACTION | RECFM |
| CHARSET | SHARE |
| DSORG | TYPE |
| FILENAME | TYPEF |

これらのオプションは以下の値を戻します。

- ACCESS は、SEQUENTIAL または DIRECT を戻します。
- ACTION は、INPUT、OUTPUT、または UPDATE を戻します。
- AMTHD は、z/OS プラットフォームでは VSAM KSDS、VSAM ESDS、または VSAM RRDS を戻し、Windows または AIX プラットフォームでは FILESYS、DDM、BTRIEVE、または ISAM を戻します。
- CHARSET は ASCII または EBCDIC を戻します。
- DSORG は、ファイル参照のデータ・セット編成を返します。 このオプションは、z/OS プラットフォームでのみ有効です。 現在、以下のデータ・セット編成が使用可能です。

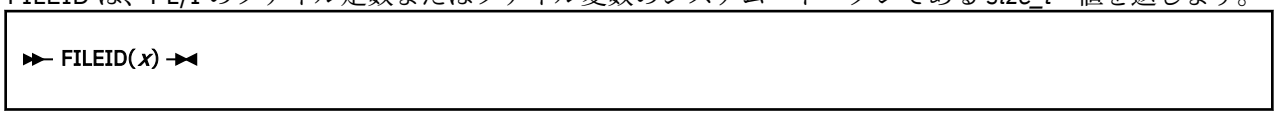
  PS (物理的順次データ・セット)
  PSU (位置依存情報を含む物理的順次データ・セット)
  DA (直接アクセス・データ・セット)
  DAU (位置依存情報を含む直接アクセス・データ・セット)
  PO (区分データ・セット (PDS または PDSE))
  POU (位置依存情報を含む区分データ・セット (PDS))
  GS (グラフィック・データ制御ブロック)
  zFS (UNIX システム・ファイル)
  VSAM (仮想記憶アクセス方式データ・セット)

  ファイル編成がサポートされていない場合、DSORG の FILEDDWORD はブランクの値を返します。
- z/OS プラットフォームの場合、 FILENAME は、zFS ファイルの完全修飾パス名、およびその他のすべてのファイルの MVS データ・セット名を戻します。 ただし、DSN=NULLFILE と DD DUMMY のいずれかで指定されたファイルについては、値「NULLFILE」を戻します。 PDS または PDSE のメンバーである MVS データ・セットについては、戻された名前にメンバー名が含まれています。 Windows および AIX プラットフォームでは、ファイルの完全修飾パス名を戻します。
- ORGANIZATION は、CONSECUTIVE、RELATIVE、REGIONAL(1)、または INDEXED を戻します。
- RECFM は、ファイルの適切なレコード・フォーマット設定、および VSAM ファイルについては U を戻します。 このオプションは z/OS でのみ有効です。
- SHARE は NONE、READ、または ALL を戻します。
- TYPE は、RECORD または STREAM を戻します。
- TYPEF は、固有ファイルのタイプを戻します。

ONCODE 1010 の ERROR 条件は、ファイルがオープンしていないか、照会されている属性がファイルに無効な場合に発生します。

## FILEID

FILEID は、PL/I のファイル定数またはファイル変数のシステム・トークンである *size_t* [1] 値を返します。




**x**
ファイル参照

このトークンを PL/I ステートメントによって達成するどのような目的のためにも使用しないでください。

z/OS では、トークンは、RECORD または STREAM ファイルに関連した DCB のアドレス、または VSAM RECORD ファイルに関連した ACB のアドレスを保持します。 トークンは他のファイルでは無効です。

**注 :** アプリケーションが DCB または ACB を読み取れるように、DCB または ACB アドレスが提供されています。 DCB および ACB を変更しないでください。

ONCODE 1010 の ERROR 条件は、ファイルがオープンしていない場合に発生します。

## FILENEW

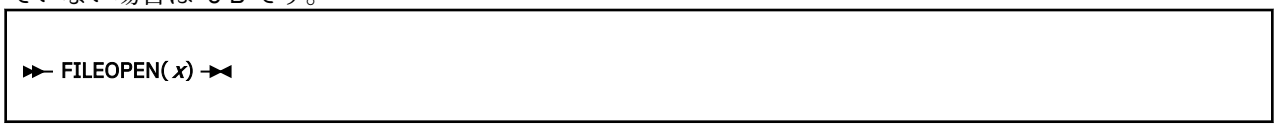
FILENEW は、自動ストレージ内の新規ファイル定数を指す FILE 変数を返します。

**x**
制限付き式。 *x* は、ファイル定数または初期化されたファイル変数でなければなりません。

新規ファイル変数は、引数が指定されていない場合、デフォルト・ファイル属性を取ります。 *x* が指定されている場合、そのファイルの宣言内の属性が使用されます。 新規ファイルが有効で使用可能な状態であるのは、FILENEW 関数が呼び出されたブロックの終了までに限ります。

## FILEOPEN

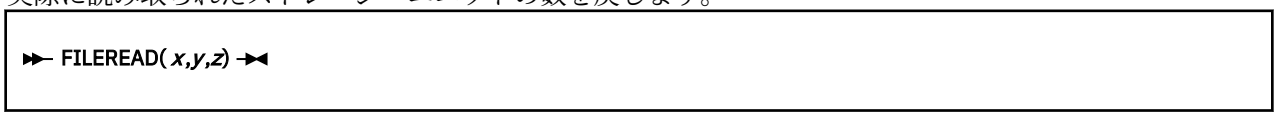
FILEOPEN によって返される値は、ファイル *x* がオープンされている場合は '1'B、ファイルがオープンされていない場合は '0'B です。




**x**
ファイル参照。

## FILEREAD

FILEREAD は、*z* ストレージ・ユニット (バイト) をファイル *x* からロケーション *y* に読み取ろうとします。 実際に読み取られたストレージ・ユニットの数を戻します。




**x**
ファイル参照

**y**
タイプ POINTER または OFFSET による式。 タイプが OFFSET の場合は、式は、AREA 属性によって宣言される OFFSET 変数でなければなりません。

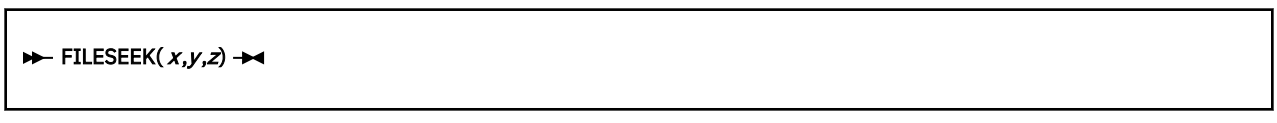
**z**
式。 計算タイプでなければならず、タイプ *size_t* に変換されます。[1]

FILEREAD は zFS TYPE(U) ファイルのみを読み取ることができます。

## FILESEEK

FILESEEK は、ファイル *x* と関連した現在のファイル位置をファイル内の新しい位置に変更します。 ファイル上での次の操作は、新しい位置で行われます。 FILESEEK は、C での **fseek** 関数と同等です。

FILESEEK は、FIXED BIN(31) 値を返します。 この値は、ファイル位置の変更が正常に行われた場合は 0、その他の場合はゼロ以外です。




**x**
ファイル参照。

**y**
ファイル・ポインターを *z* に対して相対的に移動する位置の数を示す *size_t* 値。

**z**
ファイル・ポインターの移動の起点を示す FIXED BINARY(31) 値。 次の値が有効です。

**-1**
ファイルの始め

**0**
ファイル・ポインターの現在位置

**1**
ファイルの終わり

FILESEEK は、zFS TYPE(U) ファイルでのみ使用できます。

## FILETELL

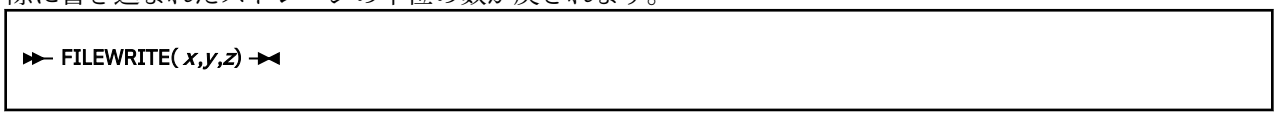
FILETELL は、ファイル *x* の現在位置を示す *size_t* [1] 値を返します。 戻り値は、ファイルの先頭からの相対オフセットです。 FILETELL は、C での **ftell** 関数と同等です。

**x**
ファイル参照

FILETELL は、zFS TYPE(U) ファイルでのみ使用できます。

## FILEWRITE

FILEWRITE により、位置 *y* からファイル *x* に *z* ストレージの 単位 (バイト) が書き込まれます。そして、実際に書き込まれたストレージの単位の数が戻されます。




**x**
ファイル参照。

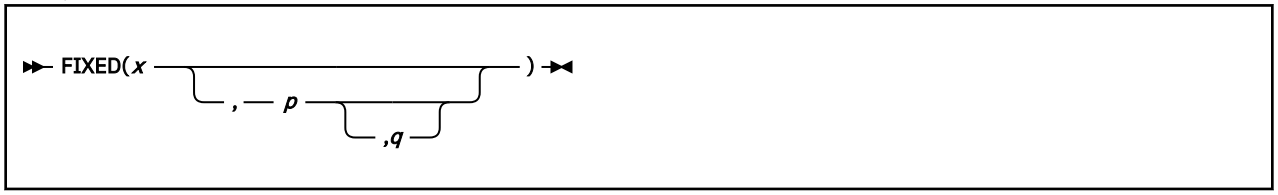
**y**
タイプ POINTER または OFFSET による式。 タイプが OFFSET の場合は、式は、AREA 属性によって宣言される OFFSET 変数でなければなりません。

**z**
式。 計算タイプでなければならず、タイプ *size_t* に変換されます。[1]

FILEWRITE は、zFS TYPE(U) ファイルにのみ書き込みを行うことができます。

## 固定

FIXED は、*p* および *q* で指定された精度で *x* の固定小数点値を戻します。 結果は、*x* の基数とモードになります。




**x**
式。

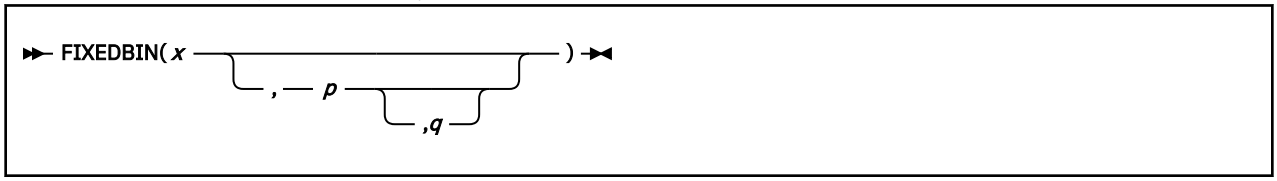
**p**
結果の合計桁数を指定する制限付き式。 これは、実行の限界を超えてはいけません。

**q**
結果の位取り係数を指定する制限付き式。 *q* が省略される場合は、位取り係数はゼロと想定されます。

p と q の両方が省略される場合、デフォルト値として、2 進数の結果の場合 (15,0)、10 進数の結果の場合 (5,0) が使用されます。

# FIXEDBIN

FIXEDBIN は、関数のパラメーターとして明示的に指定されない限り、ソースから導き出した精度およびスケールで FIXED BIN 値を返します。




**x**
式。

**p**
結果の合計桁数を指定する制限付き式。 これは、実行の限界を超えてはいけません。

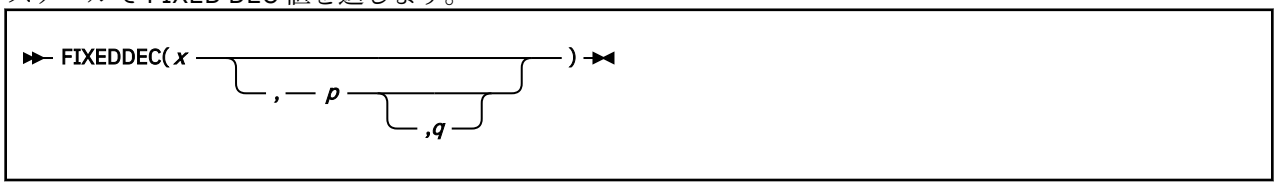
**q**
結果の位取り係数を指定する制限付き式。 *q* が省略される場合は、位取り係数はゼロと想定されます。

*p* と *q* が両方とも省略される場合は、結果の精度は次の表に従ってソースにより決まります。

| source | 結果 |
|---|---|
| FIXED BIN(p,q) | FIXED BIN(p,q) |
| FIXED DEC(p,q) | FIXED BIN(r,s)<br>ここで、r = min(M,1+CEIL(p*3.32))<br>s = CEIL(ABS(q*3.32))*SIGN(q) |
| FLOAT BIN(p) | FIXED BIN(p,0) |
| FLOAT DEC(p) | FIXED BIN(r,0)<br>ここで、r = min(M,CEIL(p*3.32)) |
| ビット | FIXED BIN(M,0) |
| CHAR、GRAPHIC、UCHAR、または WIDECHAR | FIXED BIN(r,0)<br>ここで、r = min(M,1+CEIL(N*3.32)) |

# FIXEDDEC

FIXEDDEC は、関数のパラメーターとして明示的に指定されない限り、ソースから導き出した精度およびスケールで FIXED DEC 値を返します。




**x**
式。

**p**
結果の合計桁数を指定する制限付き式。 これは、実行の限界を超えてはいけません。

**q**
結果の位取り係数を指定する制限付き式。 *q* が省略される場合は、位取り係数はゼロと想定されます。

*p* と *q* が両方とも省略される場合は、結果の精度は次の表に従ってソースにより決まります。

| source | 結果 |
|---|---|
| FIXED BIN(p,q) | FIXED DEC(r,s)<br>ここで、r = min(N,1+CEIL(p/3.32))<br>s=CEIL(ABS(q/3.32))*SIGN(q) |
| FIXED DEC(p,q) | FIXED DEC(p,q) |
| FLOAT BIN(p) | FIXED DEC(r,0)<br>ここで、r = min(N,CEIL(p/3.32) |
| FLOAT DEC(p) | FIXED DEC(p,0) |
| ビット | FIXED DEC(r,0)<br>ここで、r = min(N,1+CEIL(M/3.32)) |
| CHAR、GRAPHIC、UCHAR、または WIDECHAR | FIXED DEC(N,0) |

## FLOAT

FLOAT は、*p* で指定された精度で、*x* の近似浮動小数点値を返します。 結果は、*x* の基数とモードになります。

**x**
式。

**p**
結果の最小桁数を指定する制限付き式。

*p* が省略される場合は、結果の精度は基数変換の規則により決まります。

*p* が省略される場合は、デフォルト値として、2 進数の結果の場合 15、10 進数の結果の場合 5 が使用されます。

## FLOATBIN

FLOATBIN は、関数のパラメーターとして明示的に指定されない限り、ソースから導き出した精度で FLOAT BIN 値を返します。

**x**
式。

**p**
結果の合計桁数を指定する制限付き式。 これは、実行の限界を超えてはいけません。

*p* が省略された場合は、結果の精度は、次の表に従ってソースにより決まります。

| source | 結果 |
|---|---|
| FIXED BIN(p,q) | FLOAT BIN(p) |

| source | 結果 |
|---|---|
| FIXED DEC(p,q) | FLOAT BIN(r)<br>ここで、r = CEIL(p*3.32) |
| FLOAT BIN(p) | FLOAT BIN(p) |
| FLOAT DEC(p) | FLOAT BIN(r)<br>ここで、r = CEIL(p*3.32) |
| ビット | FLOAT BIN(M) |
| CHAR、GRAPHIC、UCHAR、または WIDECHAR | FLOAT BIN(r)<br>ここで、r = CEIL(N*3.32) |

## FLOATDEC

FLOATDEC は、関数のパラメーターとして明示的に指定されない限り、ソースから導き出した精度で FLOAT DEC 値を返します。

**x**
式。

**p**
結果の合計桁数を指定する制限付き式。 これは、実行の限界を超えてはいけません。

*p* が省略された場合は、結果の精度は、次の表に従ってソースにより決まります。

| source | 結果 |
|---|---|
| FIXED BIN(p,q) | FLOAT DEC(r)<br>ここで、r = CEIL(p/3.32) |
| FIXED DEC(p,q) | FLOAT DEC(p) |
| FLOAT BIN(p) | FLOAT DEC(r)<br>ここで、r = CEIL(p/3.32) |
| FLOAT DEC(p) | FLOAT DEC(p) |
| ビット | FLOAT DEC(r)<br>ここで、r = CEIL(M/3.32) |
| CHAR、GRAPHIC、UCHAR、または WIDECHAR | FLOAT DEC(N) |

## FLOOR

FLOOR は、*x* 以下の最大の整数値を返します。

**x**
実数式。

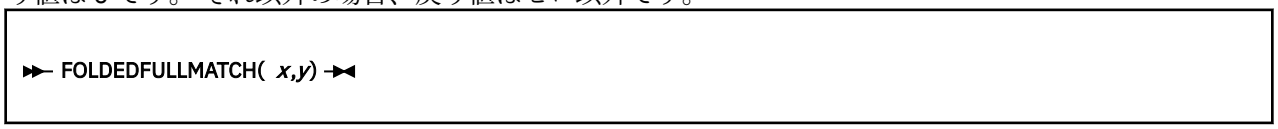
結果のモード、基数、スケール、および精度は、引数と一致します。 *x* が精度 (*p*,*q*) を使用した固定小数点のときは、結果 の精度は次の式で与えられます。

```
(min(n,max(p-q+1,1)),0)
```

この場合、n は、使用できる最大桁数です。FIXED DECIMAL の場合は N が入り、FIXED BINARY の場合は M が入ります。

## FOLDEDFULLMATCH

FOLDEDFULLMATCH は、Unicode の完全な 大/小文字変換規則に従って小文字に変換されたときに 2 つのストリングが同じかどうかを示す FIXED BINARY(31) 値を返します。 2 つのストリングが同一の場合、戻り値は 0 です。 それ以外の場合、戻り値はゼロ以外です。




**x**
式。 *x* は計算タイプを保持している必要があり、必要に応じて UCHAR タイプに変換されます。

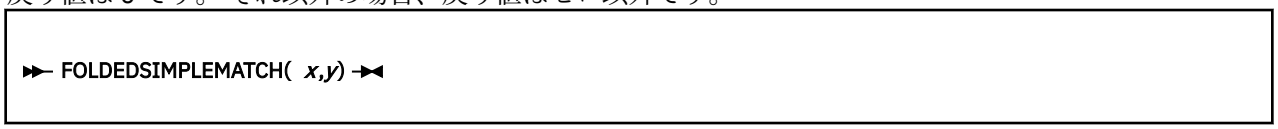
**y**
式。 *y* は計算タイプを保持している必要があり、必要に応じて UCHAR タイプに変換されます。

FOLDEDFULLMATCH 組み込み関数を使用すると、すべてのコード・ブロックからのすべての UTF-8 データ項目が必要に応じて大文字変換されます。

完全 大文字変換では、*x* と *y* の長さが同じでなくてもかまいません。 例えば、Haus と HAUS が一致するだけでなく、Straße と STRASSE も一致します。

## FOLDEDSIMPLEMATCH

FOLDEDSIMPLEMATCH は、Unicode の単純な 大/小文字変換規則に従って小文字に変換されたときに 2 つのストリングが同じかどうかを示す FIXED BINARY(31) 値を返します。 2 つのストリングが同一の場合、戻り値は 0 です。 それ以外の場合、戻り値はゼロ以外です。




**x**
式。 *x* は計算タイプを保持している必要があり、必要に応じて UCHAR タイプに変換されます。

**y**
式。 *y* は計算タイプを保持している必要があり、必要に応じて UCHAR タイプに変換されます。

FOLDEDSIMPLEMATCH 組み込み関数を使用すると、すべてのコード・ブロックからのすべての UTF-8 データ項目が必要に応じて大文字変換されます。

単純 大文字変換では、*x* と *y* の長さが同じである必要があります。 例えば、Haus と HAUS は一致しますが、Straße と STRASSE は一致しません。

## GAMMA

GAMMA は、*x* の基数、モード、および精度で浮動小数点の値を戻します。

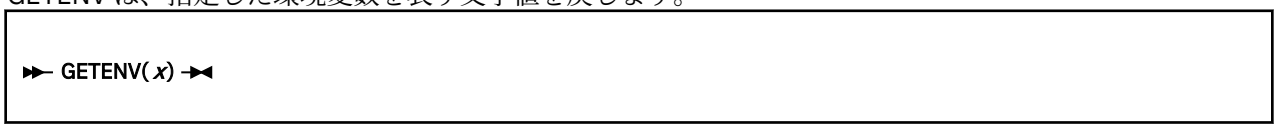
GAMMA は、次の式で与えられる *x* のガンマの近似値です。

$\mathrm{gamma}(x) = \int_0^\infty (u^{x-1})(e^{-x})du$

**x**
実数式。 *x* の値は、ゼロより大きくなければなりません。

## GETENV

GETENV は、指定した環境変数を表す文字値を戻します。




**x**
環境変数を命名する式。

## GETJCLSYMBOL

GETJCLSYMBOL は、要求されたエクスポート済み JCL シンボルを表す CHARACTER ストリング値を返します。

**x**
エクスポートされた JCL シンボルの名前を指定します。

*x* と同じ値を持つ JCL シンボルがない場合は、ヌル・ストリングが返されます。

以下に示す JCL ステートメントでは、GETJCLSYMBOL('S1') の指定により、STEWART が返されます。

```
//EX1 EXPORT SYMLIST=(S1,L1)
//SET1 SET S1=STEWART,L1=LAGUARDIA
//EX2 EXEC PGM=GETSYM   /* PLI program call GETJCLSYMBOL */
```

.

## GETSYSINT

GETSYSINT は、要求されたシステム情報の値である *size_t* 値を返します。

**x**
要求されたシステム情報。 *x* の有効なキーワードは、以下のとおりです。

MAXACTINFO

MAXACTINFO キーワードは、ジョブカードに指定されている JOB アカウンティング情報内のジョブ・アカウンティング・フィールドの数を返します。 GETSYSWORD 組み込み関数の ACTINFO と組み合わせて、個々のジョブ・アカウンティング・フィールド情報を取得できます。

## GETSYSWORD

GETSYSWORD は、要求されたシステム情報の値である CHARACTER ストリングを返します。

**x**
要求されたシステム情報を指定する文字式。 *x* の有効なキーワードは、以下のとおりです。

ASID
ACTINFO
JESNODE
JOBCLASS
JOBNAME
JOBNUMBER
MSGCLASS
PROCSTEP
SMFID
STEPNAME
SYSNAME
SYSNODE
SYSPLEX

ASID キーワードは、組み込み関数を呼び出すプログラムのアドレス・スペース ID 値を 16 進数で返します。

ACTINFO キーワードは、ジョブカードに指定されている JOB アカウンティング情報であるコンマ区切りのストリングを返します。 GETSYSINT 組み込み関数の MAXACTINFO キーワードと組み合わせて、個々のジョブ・アカウンティング・フィールド情報を取得できます。 返されるアカウント情報の最大長は 144 バイトです。 サブパラメーターがアポストロフィで囲まれている場合、 アポストロフィはアカウンティング情報ストリングに含められません。 例えば、ジョブカードに次のように指定されているとします。

```
//JOB44  JOB  (D548-8686,'12/8/85',PGMBIN)
```

GETSYSWORD(ACTINFO) は、次のストリングを返します。

```
D548-8686,12/8/85,PGMBIN
```

JOB アカウンティング情報のパラメーターの詳細については、「z/OS MVS JCL 解説書」を参照してください。

JESNODE キーワードは、組み込み関数を呼び出すプログラムのシステムの JES ノード名を返します。

JOBCLASS キーワードは、組み込み関数を呼び出すプログラムのバッチ・ジョブに割り当てられたジョブ・クラスを返します。

JOBNAME キーワードは、組み込み関数を呼び出す JOB または TASK の名前を返します。

JOBNUMBER キーワードは、 組み込み関数を呼び出すプログラムのバッチ・ジョブに割り当てられた JES JOBID を返します。

MSGCLASS キーワードは、 組み込み関数を呼び出すプログラムのジョブのメッセージ・クラスを返します。

PROCSTEP キーワードは、JCL プロシージャーを呼び出すジョブ・ステップ名を返します。この JCL プロシージャーには、PL/I プログラムを実行するステップが入っています。 JCL プロシージャーから呼び出されない場合は、ヌル・ストリングが返されます。

SMFID キーワードは、組み込み関数を呼び出すプログラムのシステムの SMFID (システム ID) を返します。

STEPNAME キーワードは、PL/I プログラムを呼び出すステップ名を返します。

SYSNAME キーワードは、組み込み関数を呼び出す TSO/USS/バッチ・ジョブのシステム・インストールの LPAR 名を戻します。

SYSNODE キーワードは、組み込み関数を呼び出すプログラムのシステムの JES ノード名を戻します。 結果は、キーワード JESNODE を使用した場合と同じです。

SYSPLEX キーワードは、組み込み関数を呼び出す TSO/USS/バッチ・ジョブのシステム・インストールの SYSPLEX 名を戻します。

CICS トランザクションから GETSYSWORD 組み込み関数が呼び出された場合、JOBNAME に対しては CICS 制御領域を開始する JOB または TASK の名前が返され、STEPNAME に対しては CICS 制御領域を初期化するステップが返されます。

## GTCA

GTCA は、LE 制御ブロックへのポインターを返します。

ストレージを変更するために GTCA 組み込み関数を使用すると、予期しない結果が生じる可能性があります。

GTCA 組み込み関数は、z/OS でのみサポートされます。

## GRAPHIC

GRAPHIC は、明示的に文字 (または混合文字) データを GRAPHIC データに変換します。 その他のすべてのデータは、最初に文字に変換され、次に GRAPHIC データ・タイプ に変換されます。

GRAPHIC は、*y* で指定されるグラフィック記号での長さを とる *x* のグラフィック値を戻します。

*x* の内容は、変換時に妥当性がチェックされます。このとき、グラフィック定 数と混合文字定数をチェックするときに使用するのと同じ規則が使われます。

**x**

式。 *x* が GRAPHIC のときは、x は長さ変更の対象で、適用可能 な埋め込みまたは切り捨てが行われます。 *x* がグラフィック以外のときは、必要な場合には文字に変換されます。 SBCS 文字は、同等の DBCS 文字に変換されます。

**y**

式。 必要に応じて、*y* は実数の固定小数点 2 進数値に変換されます。 *y* が省略される場合は、長さはタイプ変換の規則によって決められます。

*y* は負であってはなりません。

*y* = 0 の場合は、結果はヌルの漢字ストリングです。

次の規則が適用されます。

- *y* が漢字ストリングを入れるのに必要な長さより大きい場合は、結果は グラフィックのブランクで埋まります。
- *y* が漢字ストリングを入れるのに必要な長さより小さい場合は、結 果が切り捨てられます。

**例 1**

この例は、CHARACTER から GRAPHIC への変換を示しています。 ターゲットは結果を収容するために十分な長さです。

```
dcl X char (11) varying;
dcl A graphic (11);
A = graphic(X,8);
```

| X の値 | 中間結果 | A に割り当てられる値 |
|---|---|---|
| ABCDEFGHIJ | .A.B.C.D.E.F.G.H.I.J | .A.B.C.D.E.F.G.H.b.b.b |
| 123 | .1.2.3 | .1.2.3.b.b.b.b.b.b.b.b |
| 123A.B.C | .1.2.3.A.B.C | .1.2.3.A.B.C.b.b.b.b.b |

この場合、.b は、DBCS ブランクです。

**例 2**

この例は、CHARACTER から GRAPHIC への変換を示しています。 ただし、ターゲットは結果を収容するためには短すぎます。

```
dcl X char (10) varying;
dcl A graphic (8);
A = graphic(X);
```

| X の値 | 中間結果 | A に割り当てられる値 |
|---|---|---|
| ABCDEFGHIJ | .A.B.C.D.E.F.G.H.I.J | .A.B.C.D.E.F.G.H |

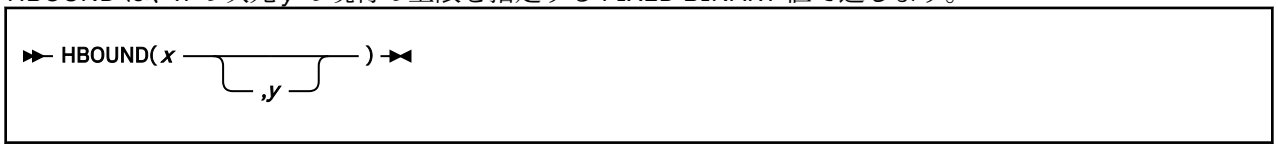
# HANDLE

HANDLE は、タイプ付き構造体 *x* にハンドルを戻します。

**x**
タイプ付き構造体。

# HBOUND

HBOUND は、*x* の次元 *y* の現行の上限を指定する FIXED BINARY 値で返します。




**x**
配列参照。 *x* の次元数は、*y* よりも小さくてはなりません。

**y**
*x* の特定の次元を指定する式。 必要な場合には、*y* は FIXED BINARY(31 ,0) に変換されます。 *y* は 1 以上でなければなりません。 *y* が指定されない場合、デフォルトは 1 に設定されます。

配列が 1 次の場合のみ、*y* を省略できます。

CMPAT(V3) コンパイラー・オプションでは、HBOUND は FIXED BIN(63) 値を返します。 CMPAT(V2) および CMPAT(LE) の各コンパイラー・オプションでは、HBOUND は FIXED BIN(31) 値を返します。

# HBOUNDACROSS

HBOUNDACROSS は、現行の上限の DIMACROSS 参照 *x* を指定する FIXED BINARY 値を戻します。

**x**
DIMACROSS 参照

CMPAT(V3) コンパイラー・オプションでは、HBOUNDACROSS は FIXED BIN(63) 値を返します。
CMPAT(V2) および CMPAT(LE) の各コンパイラー・オプションでは、HBOUNDACROSS は FIXED BIN(31) 値を返します。

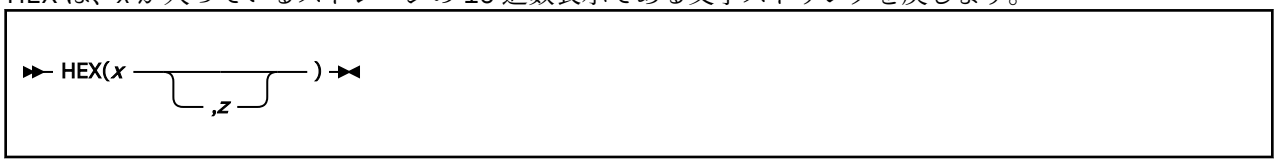
### 例

次の例は、HBOUNDACROSS の使い方を示したものです。

```
dcl jx fixed bin(31);

dcl
  1 a,
    2 b fixed bin,
    2 c fixed bin;
dcl 1 xa( 100 ) like a dimacross;

...

do jx = 1 to hboundacross(xa);
  a = xa, by dimacross(jx);
  ...
end;
```

## HEX

HEX は、*x* が入っているストレージの 16 進数表示である文字ストリングを戻します。




HEX(x) は、長さ 2 * size(x) の文字ストリングを返します。

HEX(x,z) は、出力ストリングの 8 文字のすべてのセット間に挿入された文字 *z* 付きの *x* を含む文字ストリングを戻します。 その長さは 2 * size(x) + ((size(x) - 1)/4) です。

コンパイラー・オプション USAGE(HEX(CSTG)) の場合、VARYING、VARYING4、VARYINGZ のストリングでは、上記計算で使用される長さは、stg(x) ではなく、cstg(x) に基づきます。

**x**
変数を表す式。 *x* を含む全バイト数は、16 進数に変換されます。

**z**
式。 *z* が指定された場合は、CHARACTER(1) NONVARYING タイプでなければなりません。

整数、オフセット、およびポインター値は、ビッグ・エンディアン形式で与えられます。

コンパイル時に 16 進数に変換するバイト数が不明な場合、変換バイト数は 32767 バイト以下になります。

**注 :** この関数は、ストレージの *x* の正確なイメージを戻しません。 正確なイメージが必要な場合は、HEXIMAGE 組み込み関数を使用してください。

### 例 1

```
dcl Sweet char(5) init('Sweet');
dcl Sixteen fixed bin(31) init(16) littleendian;
dcl XSweet char(size(Sweet)*2+(size(Sweet)-1)/4);
dcl XSixteen char(size(Sixteen)*2+(size(Sixteen)-1)/4);

XSweet = hex(Sweet,'-');
           /* '53776565-74' */
```

```
XSweet = heximage(addr(Sweet),length(Sweet),'-');
          /* '53776565-74' */

XSixteen = hex(Sixteen,'-');
          /* '00000010' - bytes reversed */

XSixteen = heximage(addr(Sixteen),stg(Sixteen),'-');
          /* '10000000' - bytes NOT reversed */
```

**例 2**

```
dcl X fixed bin(15) littleendian;
dcl Y fixed bin(15) bigendian;

X = 258;                                    /* stored as '0201'B4 */
Y = 258;                                    /* stored as '0102'B4 */

display (hex(X));                           /* displays 0102 */
display (hex(Y));                           /* displays 0102 */

display (heximage( addr(X), stg(X) ));      /* displays 0201 */
display (heximage( addr(Y), stg(Y) ));      /* displays 0102 */
```

**関連情報**

446 ページの『HEXIMAGE』
HEXIMAGE は、指定された位置のストレージの 16 進数表示である文字ストリングを戻します。

# HEX8

HEX8 は、*x* が入っているストレージの UTF-8 16 進表記である文字ストリングを返します。

HEX8(*x*) は、長さ `2 * size(x)` の文字ストリングを戻します。

HEX8(*x,z*) は、出力ストリングの 8 文字のすべてのセット間に挿入された文字 *z* 付きの *x* を含む文字ストリングを戻します。 その長さは `2 * size(x) + ((size(x) - 1)/4)` です。

コンパイラー・オプション USAGE(HEX(CSTG)) の場合、VARYING、VARYING4、VARYINGZ のストリングでは、上記計算で使用される長さは、`stg(x)` ではなく、`cstg(x)` に基づきます。

**x**
変数を表す式。 *x* を含む全バイト数は、16 進数に変換されます。

**z**
式。 z が指定された場合、*z* は CHARACTER(1) NONVARYING タイプでなければならず、有効な 1 バイト UTF-8 文字でなければなりません。

整数、オフセット、およびポインター値は、ビッグ・エンディアン形式で与えられます。

コンパイル時に 16 進数に変換するバイト数が不明な場合、変換バイト数は 32767 バイト以下になります。

**注 :** この関数は、ストレージの *x* の正確なイメージを戻しません。 正確なイメージが必要な場合は、HEXIMAGE8 組み込み関数を使用してください。

**例 1**

```
dcl Sweet char(5) init('Sweet');
dcl Sixteen fixed bin(31) init(16) littleendian;
dcl XSweet char(size(Sweet)*2+(size(Sweet)-1)/4);
dcl XSixteen char(size(Sixteen)*2+(size(Sixteen)-1)/4);

XSweet = hex8(Sweet,'-');
          /* '53776565-74'a */

XSweet = heximage8(addr(Sweet),length(Sweet),'-');
```

```
            /* '53776565-74'a */

XSixteen = hex8(Sixteen,'-');
            /* '00000010' - bytes reversed */

XSixteen = heximage8(addr(Sixteen),stg(Sixteen),'-');
            /* '10000000' - bytes NOT reversed */
```

**例 2**

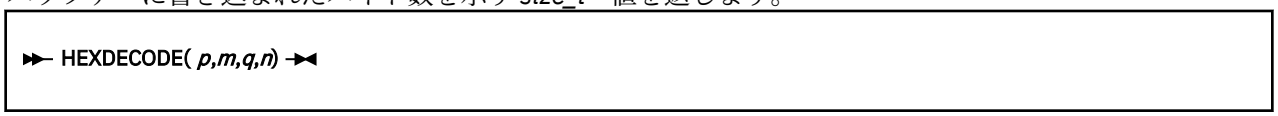
```
dcl X fixed bin(15) littleendian;
dcl Y fixed bin(15) bigendian;

X = 258;                                        /* stored as '0201'B4 */
Y = 258;                                        /* stored as '0102'B4 */

display (hex8(X));                              /* displays 0102 */
display (hex8(Y));                              /* displays 0102 */

display (heximage8( addr(X), stg(X) ));         /* displays 0201 */
display (heximage8( addr(Y), stg(Y) ));         /* displays 0102 */
```

**関連情報**

446 ページの『HEXIMAGE8』
HEXIMAGE8 は、指定された位置のストレージの UTF-8 16 進表記である文字ストリングを返します。

# HEXDECODE

HEXDECODE は、DEFAULT コンパイラー・オプションの ASCII/EBCDIC サブオプションで指定された文字セットでエンコードされた基数 16 からソース・バッファーをデコードします。 この関数は、ターゲット・バッファーに書き込まれたバイト数を示す *size_t* [1] 値を返します。




***p***
ターゲット・バッファーのアドレスを指定します。

***M***
ターゲット・バッファーの長さを指定します。計算タイプでなければならず、タイプ *size_t* に変換されます。

***Q***
ソース・バッファーのアドレスを指定します。

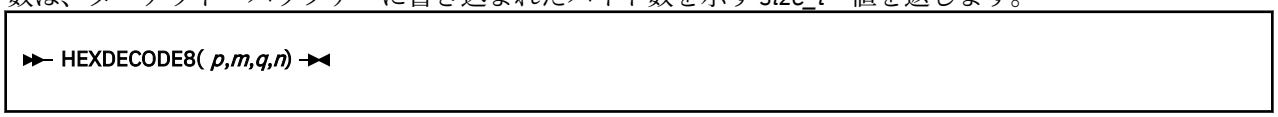
***n***
ソース・バッファーの長さを指定します。計算タイプでなければならず、タイプ *size_t* に変換されます。

ターゲット・バッファーのアドレスがゼロである場合は、書き込まれることが想定されたバイト数が返されます。 ターゲット・バッファーの大きさが十分でなかった場合には、値 -1 が返されます。 ターゲット・バッファーの大きさが十分であった場合には、バッファーに書き込まれたバイト数が返されます。

ソースに 16 進数字以外の文字は含まれている場合は、CONVERSION 条件が発生します。

# HEXDECODE8

HEXDECODE8 は、UTF-8 でエンコードされた基数 16 からソース・バッファーをデコードします。 この関数は、ターゲット・バッファーに書き込まれたバイト数を示す *size_t* [1] 値を返します。




***p***
ターゲット・バッファーのアドレスを指定します。

***M***
ターゲット・バッファーの長さを指定します。計算タイプでなければならず、タイプ *size_t* に変換されます。

***Q***
ソース・バッファーのアドレスを指定します。

***n***
ソース・バッファーの長さを指定します。計算タイプでなければならず、タイプ *size_t* に変換されます。

ターゲット・バッファーのアドレスがゼロである場合は、書き込まれることが想定されたバイト数が返されます。 ターゲット・バッファーの大きさが十分でなかった場合には、値 -1 が返されます。 ターゲット・バッファーの大きさが十分であった場合には、バッファーに書き込まれたバイト数が返されます。

ソースに 16 進数字以外の文字は含まれている場合は、CONVERSION 条件が発生します。

## HEXIMAGE

HEXIMAGE は、指定された位置のストレージの 16 進数表示である文字ストリングを戻します。

HEXIMAGE(p,n) は、ロケーション *p* のストレージの *n* バイトの 16 進表記である文字ストリングを返します。 その長さは 2 * n です。

HEXIMAGE(p,n,z) は、出力ストリングの 8 文字の各セット間に挿入された文 字 *z* 付きの位置 *p* でのストレージの *n* の 16 進表 示である文字を戻します。 その長さは (2 * n) + ((n - 1)/4) です。

**p**
ロケーター・タイプ (POINTER または OFFSET) を保持していなければなら ない制限付き式。 *p* が OFFSET の場合、AREA 属性を保持している必要があります。

**n**
式。 *n* は計算タイプでなければならず、FIXED BINARY(31,0) に変換されます。

**z**
*z* が指定された場合は、CHARACTER(1) NONVARYING タイプでなければなりません。

コンパイル時に 16 進数に変換するバイト数が不明な場合、変換バイト数は 32767 バイト以下になります。

HEXIMAGE 組み込み関数の例については、443 ページの『HEX』を参照してください。

## HEXIMAGE8

HEXIMAGE8 は、指定された位置のストレージの UTF-8 16 進表記である文字ストリングを返します。

HEXIMAGE8(*p,n*) は、*p* の位置にあるストレージの *n* バイトの 16 進表記である文字ストリングを戻します。 その長さは 2 * n です。

HEXIMAGE8(*p,n,z*) は、出力ストリングの 8 文字の各セット間に挿入された文 字 *z* 付きの位置 *p* でのストレージの *n* バイトの 16 進表示である文字を戻します。 その長さは (2 * n) + ((n - 1)/4) です。

**p**
ロケーター・タイプ (POINTER または OFFSET) を保持していなければならない制限付き式。 *p* が OFFSET の場合、AREA 属性を保持している必要があります。

**n**
式。 *n* は計算タイプでなければならず、FIXED BINARY(31,0) に変換されます。

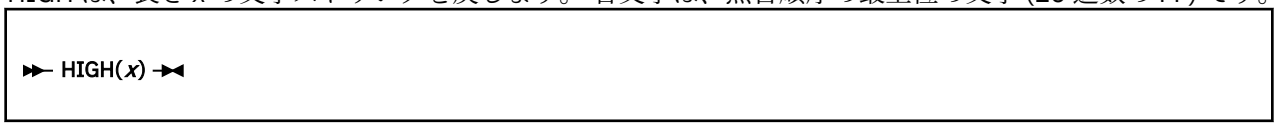
**z**
式。 z が指定された場合、*z* は CHARACTER(1) NONVARYING タイプでなければならず、有効な 1 バイト UTF-8 文字でなければなりません。

コンパイル時に 16 進数に変換するバイト数が不明な場合、変換バイト数は 32767 バイト以下になります。

HEXIMAGE8 組み込み関数の例については、444 ページの『HEX8』を参照してください。

## HIGH

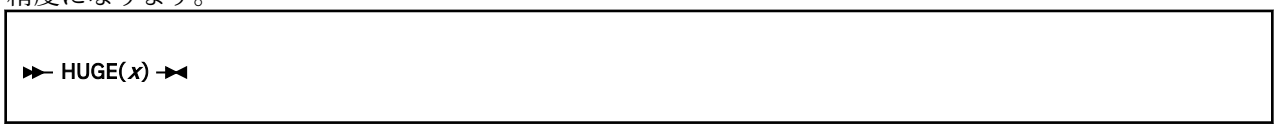
HIGH は、長さ *x* の文字ストリングを戻します。 各文字は、照合順序の最上位の文字 (16 進数の FF) です。




**x**
式。 必要な場合には、*x* は正の実数固定小数点の 2 進数の値に変換されます。 *x* = 0 の場合は、結果はヌル文字ストリングです。

## HUGE

HUGE は、*x* の想定可能な正の最大値を浮動小数点の値で戻します。 戻り値は、*x* の基数、モード、および精度になります。




**x**
式。 *x* は、属性 REAL FLOAT を保持する必要があります。

HUGE(x) は定数であり、制限付き式で使用できます。

## IAND

IAND は、引数の論理 AND を戻します。

***x* および *y***
計算タイプを保持する必要がある式。

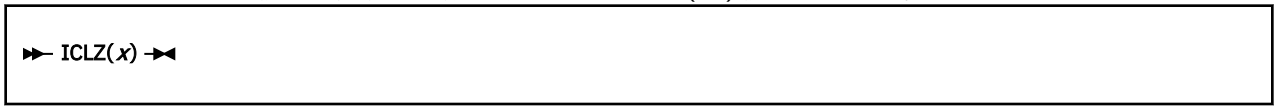
REAL FIXED BIN(p,0) ではない引数があると、その引数は SIGNED REAL FIXED BIN(p,0) に変換されます。

引数のいずれかが SIGNED の場合、UNSIGNED の引数はすべて SIGNED に変換されます。

結果は REAL FIXED BIN( max(p1,p2,...), 0) です。 すべての引数が UNSIGNED の場合は、結果も UNSIGNED です。

## ICLZ

ICLZ は、FIXED BIN 値内の先行ゼロ数を示す FIXED BIN(31) 値を返します。



***x***
スケール係数がゼロである REAL FIXED BIN 値を指定します。

返される値は、*x* が占有するビット数に対する相対値です。

したがって、*x* が精度 *p* の SIGNED である場合、次のようになります

- (*p* < 8) の場合、返される値は 0 から 8 の間です
- (*p* < 16) の場合、返される値は 0 から 16 の間です
- (*p* < 32) の場合、返される値は 0 から 32 の間です
- それ以外の場合、返される値は 0 から 64 の間です

*x* が精度 *p* の UNSIGNED である場合、次のようになります。

- (*p* <= 8) の場合、返される値は 0 から 8 の間です
- (*p* <= 16) の場合、返される値は 0 から 16 の間です
- (*p* <= 32) の場合、返される値は 0 から 32 の間です
- それ以外の場合、返される値は 0 から 64 の間です

## IEOR

IEOR は、*x* と *x* の論理排他 OR を戻します。 すべての引数が UNSIGNED の場合は、結果も UNSIGNED です。

***x* および *y***
計算タイプを保持する必要がある式。

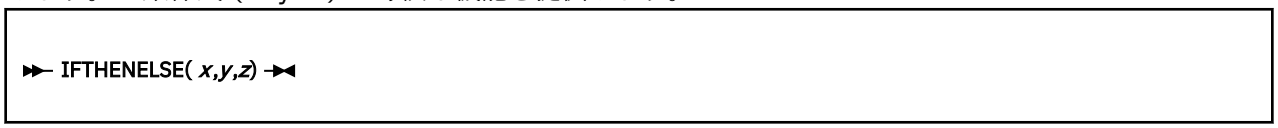
REAL FIXED BIN(p,0) ではない引数があると、その引数は SIGNED REAL FIXED BIN(p,0) に変換されます。

引数のいずれかが SIGNED の場合、UNSIGNED の引数はすべて SIGNED に変換されます。

結果は REAL FIXED BIN( max(p1,p2,...), 0 ) です。 すべての引数が UNSIGNED の場合は、結果も UNSIGNED です。

## IFTHENELSE

IFTHENELSE は、その最初の引数の *true* または *false* 値それぞれに応じて 2 番目または 3 番目の引数を返します。 C 条件式 (x?y:z) と等価な機能を提供します。




**x**
ビットに変換できるオペランド。 変換されたビット・ストリング内のいずれかのビットの値が '1'b の場合、*x* は *true* です。

**y および z**
どちらも次の条件を満たす必要があるオペランド。

- 計算値である (かつ、いずれかがストリングの場合、一定の長さの NONVARYING でなければなりません)
- ポインターである
- 同じタイプの序数である
- 同じ構造体タイプに対するハンドルである

IFTHENELSE(*x, y, z* ) の場合、以下の規則が適用されます。

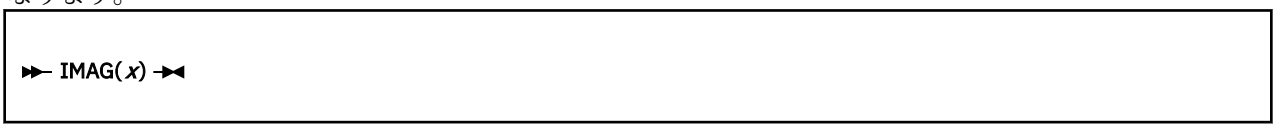
- 第 1 オペランドが評価され、 その値によって第 2 オペランドか第 3 オペランドのどちらを評価するかが決定されます。

  値が *true* の場合、第 2 オペランドが評価されます。
  値が *false* の場合、第 3 オペランドが評価されます。

  結果は、第 2 オペランドまたは第 3 オペランドの値になります。
- *y* と *z* が計算値の場合、結果タイプは *y* と *z* の共通タイプになります。
- *y* または *z* のいずれかが算術演算の場合、結果は MAX(*y, z*) と同じ精度で、*y* および *z* の共通基数、モード、およびスケールを持つ算術演算になります。 それ以外の場合、結果は、*y* と *z* の連結の場合と同じタイプで、長さが *y* と *z* の最大長に等しいストリングになります。
- *y* と *z* が非計算値の場合、結果タイプは同じタイプになります。

## IMAG

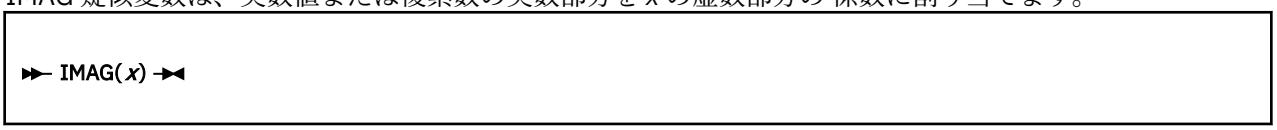
IMAG は、*x* の虚数部を返します。 結果のモードは実数です。結果は、*x* の基数、スケール、および 精度になります。




**x**
式。 *x* は、実数の場合は複素数に変換され、該当するゼロ値が返されます。

## IMAG 疑似変数

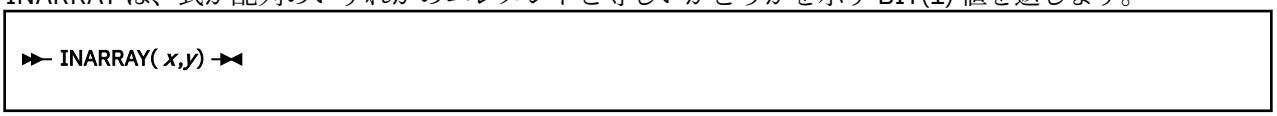
IMAG 疑似変数は、実数値または複素数の実数部分を *x* の虚数部分の 係数に割り当てます。




**x**
複素数参照。

## INARRAY

INARRAY は、式が配列のいずれかのエレメントと等しいかどうかを示す BIT(1) 値を返します。




**x**
スカラー式。 *x* は、*y* のエレメントのタイプと比較可能なタイプを保持している必要があります。

**y**
配列式。

*y* が単純な INITIAL リストを持つ 1 次元の STATIC NONASSIGNABLE 配列への参照である場合、コンパイラーは、*y* のエレメントが定数であると想定し、INARRAY( *x, y* ) を INLIST( *x, ...* ) の場合と同様に処理します。 ここで *... y* のエレメントを示します。

例えば、以下のように指定します。

```
dcl countryCode char(2);
dcl ccs(3) char(2) static nonasgn init( 'AT', 'DE', 'CH' );
```

then

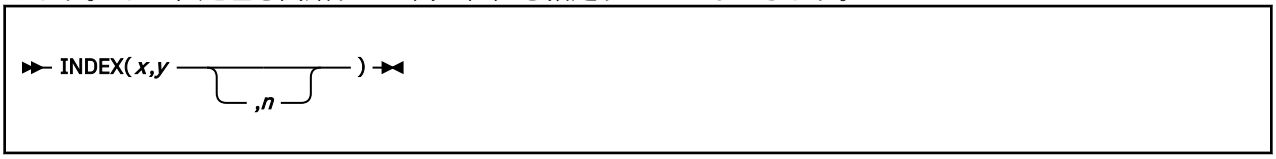
```
INARRAY( countryCode, ccs )
```

は次と等しい

```
INLIST( countryCode, 'AT', 'DE', 'CH' )
```

# INDEX

INDEX は、*y* と同じサブストリングの *x* 内の開始位置を示す、スケールなしの REAL FIXED BINARY 値を返します。 また、処理を開始する *x* 内の位置を指定することもできます。




**x**
検索されるストリング式。

**y**
検索元のストリング式。

**n**
*n* は、処理を開始する *x* 内の位置を指定します。 計算タイプでなければならず、FIXED BINARY(31 ,0) に変換されます。

*x* 内で *y* が発生しない場合、または *x* か *y* のどちらかが長さゼロがある場合は、値ゼロが戻されます。

*n* が 1 より小さい場合、または *n* が 1 + length(x) より も大きい場合は、STRINGRANGE 条件が発生し、結果は 0 になります。

BIFPREC コンパイラー・オプションによって、戻される結果の精度が決まります。

INDEX は、2 番目および 3 番目の引数がリテラル、VALUE 属性を使用して宣言された名前付き定数、または制限付き式のいずれかである場合に、最も機能を発揮します。

**例**

```
dcl tractatus char
      value( 'Wovon man nicht sprechen kann, ' ||
             'darueber muss man schweigen.' );

dcl pos fixed bin init(1);

pos = index( tractatus, 'man', pos+1 ); /* pos = 07 */

pos = index( tractatus, 'man', pos+1 ); /* pos = 46 */

pos = index( tractatus, 'man', pos+1 ); /* pos = 00 */
```

# INDEXR

INDEXR は、*y* の検索が *x* の右端から開始される場合、*y* と同一のサブストリングの *x* 内の開始位置を示すスケールなしの REAL FIXED BINARY 値を返します。 また、処理を開始する *x* 内の位置を指定することもできます。

INDEXR 関数は、次のこと以外は INDEX 組み込み関数と同じ演算を行います。

- 検索は、右から左に行われる。
- *n* のデフォルト値は LENGTH(*x*) です。
- 0 ≤ *n* ≤ LENGTH(*x*) でないと、STRINGRANGE 条 件 (割り込み可能な場合) が発生します。 暗黙処置と通常の戻りは、結果ゼロになります。

BIFPREC コンパイラー・オプションによって、戻される結果の精度が決まります。

INDEXR は、2 番目および 3 番目の引数がリテラル、VALUE 属性を使用して宣言された名前付き定数、または制限付き式のいずれかである場合に、最も機能を発揮します。

**関連情報**

450 ページの『INDEX』

INDEX は、*y* と同じサブストリングの *x* 内の開始位置を示す、スケールなしの REAL FIXED BINARY 値を返します。 また、処理を開始する *x* 内の位置を指定することもできます。

# INDICATORS

INDICATORS は、構造体 *x* 内の次の論理レベルにあるエレメントの数を示す FIXED BIN 値を返します。

***x***
式。

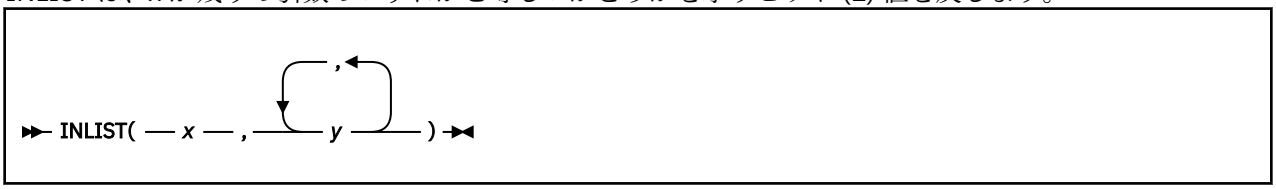
*x* は構造体参照でなければなりません。

INDICATORS(x) は必ず制限付き式を形成します。

INDICATORS 組み込み関数は、SQL ステートメントで使用するインディケーター配列を宣言する場合に役立ちます。

# INLIST

INLIST は、*x* が残りの引数のいずれかと等しいかどうかを示すビット (1) 値を戻します。



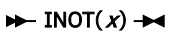

**x および y**
式。 すべてを同じ型の ORDINAL にするか、すべてを計算型にする必要があります。

INLIST(x,y1,y2,y3,...,yn) は、(x=y1) | (x=y2) | (x=y3) | ... | (x=yn) と等価です (n は 2 から 63 までの範囲の値)。

最初の引数 *x* の評価の後、残りの引数の評価によって最初の引数のアドレスまたは値を変更することはできません。 この条件は、最初の引数以外がすべて定数になっている場合にも当てはまります。 さらに、2 番目以降の引数が、最初の引数に関連したストレージを変更するユーザー関数の呼び出しに依存していない場合にも当てはまります。

# INOT

INOT は、*x* の論理 NOT を戻します。




***x***
式。 *n* は、計算タイプでなくてはなりません。

*x* が REAL FIXED BIN(p,0) の場合、その結果は REAL FIXED BIN(p,0) で あり、*x* が UNSIGNED の場合、結果も UNSIGNED です。 それ以外の場合、*x* は SIGNED REAL FIXED BIN(p,0) に変換され、その結果は同じ属性を持ちます。

INOT(x) には *x* と反対の符号が付きますが、INOT(x) は -*x* と同じではありません。

**例**

```
inot(0)       /*  produces -1  */
inot(-1)      /*  produces  0  */
inot(+1)      /*  produces -2  */
```

# IOR

IOR は、引数の論理 OR を戻します。

***x* および *y***
計算タイプを保持する必要がある式。

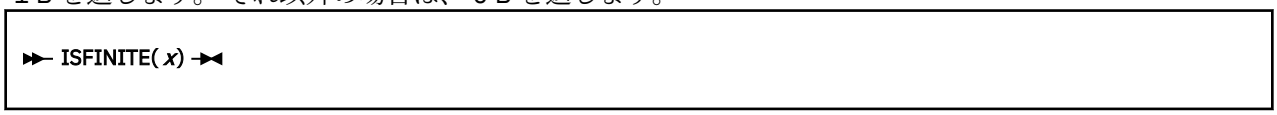
REAL FIXED BIN(p,0) ではない引数があると、その引数は SIGNED REAL FIXED BIN(p,0) に変換されます。

引数のいずれかが SIGNED の場合、UNSIGNED の引数はすべて SIGNED に変換されます。

結果は REAL FIXED BIN( max(p1,p2,...), 0 ) です。 すべての引数が UNSIGNED の場合は、結果も UNSIGNED です。

# ISFINITE

ISFINITE の呼び出し時に指定された引数が NAN でなく、正または負の無限大でもない場合、ISFINITE は '1'B を返します。 それ以外の場合は、'0'B を返します。



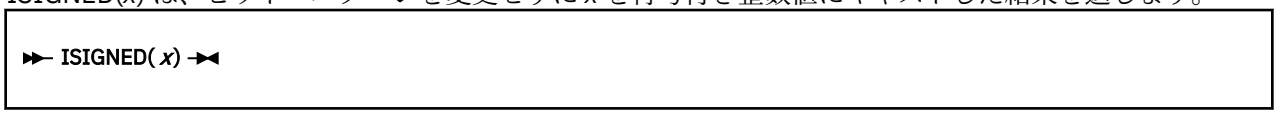

**x**
REAL FLOAT DECIMAL 式。

FLOAT(DFP) コンパイラー・オプションは有効でなければなりません。

引数のフォーマットが何であれ、浮動小数点例外は発生しません。

# ISIGNED

ISIGNED(*x*) は、ビット・パターンを変更せずに *x* を符号付き整数値にキャストした結果を返します。




***x***
式。 *n* は、計算タイプでなくてはなりません。

*x* が整数でない (つまり位取り係数ゼロの REAL FIXED BIN でない) 場合、*x* は REAL FIXED BIN(*p*,0) に変換されます。

ISIGNED( *x* ) は、整数 *x* に対して、*x* と同じビット・パターンを持つ値を返しますが、その値は属性 SIGNED FIXED BIN(*p*) を持ちます。

*x* が UNSIGNED である場合、*p* は以下のようになります。

- precision(*x*) = 8、16、32、または 64 の場合は、*p* = precision(*x*) - 1。 その他の場合は、*p* = precision(*x*)。
- *x* が SIGNED の場合は、*p* = precision(*x*)。

**例**

```
ISIGNED('ff_ff_ff_ff'xu) equals the SIGNED FIXED BIN(31) value -1.
```

# ISINF

ISINF の呼び出し時に指定された引数が無限大である場合、ISINF は '1'B を返します。 それ以外の場合は、'0'B を返します。

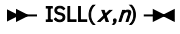
**x**
REAL FLOAT DECIMAL 式。

FLOAT(DFP) コンパイラー・オプションは有効でなければなりません。

引数のフォーマットが何であれ、浮動小数点例外は発生しません。

# ISLL

ISLL(*x*,*n*) は、論理的に *x* を左に *n* 桁シフトし、右側にゼロを埋め込んだ結果を返します。




***x***
式。 *n* は、計算タイプでなくてはなりません。

***n***
式。 *n* は、計算タイプでなくてはなりません。

*x* が REAL FIXED BIN(p,0) および SIGNED である場合、結果は SIGNED REAL FIXED BIN(*r*,0) です。 ここで、p <= M1 の場合は r = M1、p > M1 の場合は r = M2 です。

*x* が REAL FIXED BIN(p,0) および UNSIGNED である場合、結果は UNSIGNED REAL FIXED BIN(r+1,0) です。 ここで、p <= (M1+1) の場合は r = (M1+1)、p > (M1+1) の場合は r = (M2+1) です。

それ以外の場合、*x* は SIGNED REAL FIXED BIN(p,0) に変換され、その結果は上記と同じ属性を持ちます。

*n* が負の場合、または *n* が r より大きい場合、結果は未定義です。

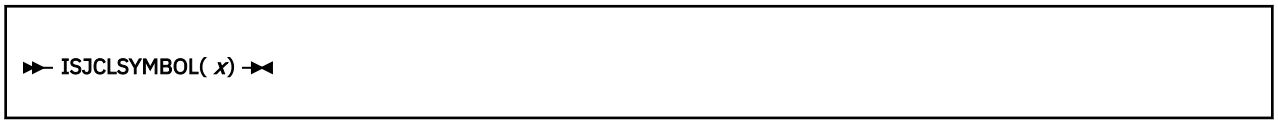
**注 :** RAISE2(*x*,*n*) と異なり、ISLL(*x*,*n*) には *x* とは別の符号が付く場合があります。

**例**

```
isll(+6,1)               /*  produces 12   */
isll(2147483645,1)       /*  produces  -6   */
```

# ISJCLSYMBOL

ISJCLSYMBOL は、引数が有効なエクスポート済み JCL シンボルの場合、'1'B を返します。 それ以外の場合は、'0'B を返します。




**x**
文字式。 テストするシンボル名を指定します。

GETJCLSYMBOL がヌル・ストリングを返した場合、ISJCLSYMBOL 組み込み関数を使用して、シンボルがエクスポート済み JCL シンボルでないことが原因か、あるいはシンボルがヌル・ストリング値に設定されていたことが原因かを判別できます。

# ISMAIN

ISMAIN() の呼び出しが行われるプロシージャーが OPTIONS(MAIN) 属性を持つ場合、ISMAIN() は '1'B を返します。 それ以外の場合は、'0'B を返します。

# ISNAN

ISNAN の呼び出し時に指定された引数が NAN である場合、ISNAN は '1'B を返します。 それ以外の場合は、'0'B を返します。

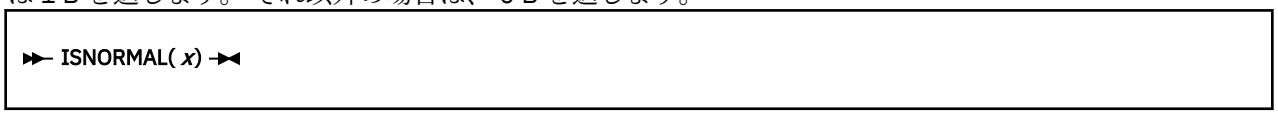
**x**
REAL FLOAT DECIMAL 式。

FLOAT(DFP) コンパイラー・オプションは有効でなければなりません。

引数のフォーマットが何であれ、浮動小数点例外は発生しません。

# ISNORMAL

ISNORMAL の呼び出し時に指定された引数がゼロ、非正規数、無限大、または NaN でない場合、ISNORMAL は'1'B を返します。 それ以外の場合は、'0'B を返します。



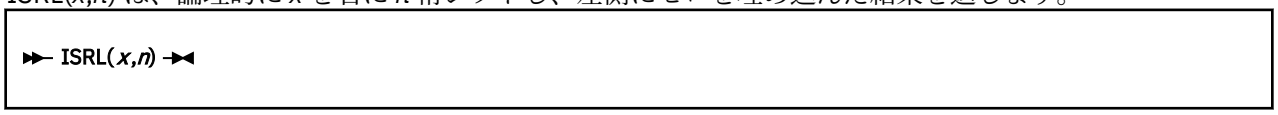

**x**
REAL FLOAT DECIMAL 式。

FLOAT(DFP) コンパイラー・オプションは有効でなければなりません。

引数のフォーマットが何であれ、浮動小数点例外は発生しません。

# ISRL

ISRL(*x*,*n*) は、論理的に *x* を右に *n* 桁シフトし、左側にゼロを埋め込んだ結果を返します。




***x***
式。 *n* は、計算タイプでなくてはなりません。

***n***
式。 *n* は、計算タイプでなくてはなりません。

結果の属性は以下のように決定されます。

- *x* が REAL FIXED BIN(p,0) および SIGNED の場合、結果は SIGNED REAL FIXED BIN(p,0) です。
- *x* が REAL FIXED BIN(p,0) および UNSIGNED の場合、結果は UNSIGNED REAL FIXED BIN(p,0) です。
- それ以外の場合、*x* は SIGNED REAL FIXED BIN(p,0) に変換され、その結果は同じ属性を持ちます。

*n* が負の場合、または *n* が M より大きい場合は、結果は定義されません。

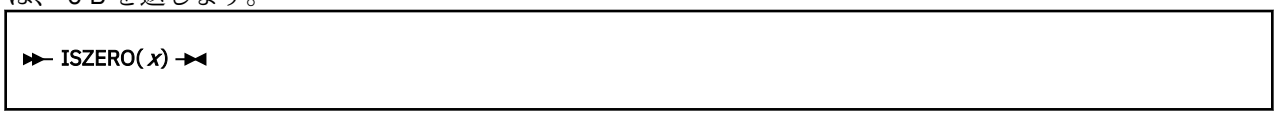
*x* が負でない場合は、ISRL(x,n) は LOWER2(x,n) と 等しく、 *x* が負の場合は、ISRL(x,n) は *n*=0 でなければ正になります。

**例**

```
isrl(+6,1)            /*  produces 3           */
isrl(-6,1)            /*  produces 2147483645  */
```

# ISZERO

ISZERO の呼び出し時に指定された引数がゼロである場合、ISZERO は '1'B を返します。 それ以外の場合は、'0'B を返します。



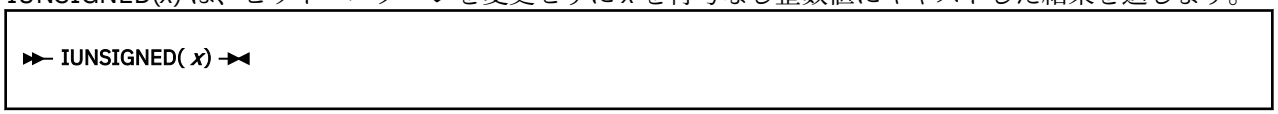

**x**
REAL FLOAT DECIMAL 式。

FLOAT(DFP) コンパイラー・オプションは有効でなければなりません。

引数のフォーマットが何であれ、浮動小数点例外は発生しません。

# IUNSIGNED

IUNSIGNED(*x*) は、ビット・パターンを変更せずに *x* を符号なし整数値にキャストした結果を返します。




***x***
式。 *n* は、計算タイプでなくてはなりません。

*x* が整数でない (つまり位取り係数ゼロの REAL FIXED BIN でない) 場合、*x* は REAL FIXED BIN(*p*,0) に変換されます。

IUNSIGNED( *x* ) は、整数 *x* に対して、*x* と同じビット・パターンを持つ値を返しますが、その値は属性 UNSIGNED FIXED BIN(*p*) を持ちます。

*x* が SIGNED である場合、*p* は以下のようになります。

- precision(*x*) = 7、15、31、または 63 の場合は、*p* = precision(*x*) + 1。 その他の場合は、*p* = precision(*x*)。
- *x* が UNSIGNED の場合は、*p* = precision(*x*)。

**例**

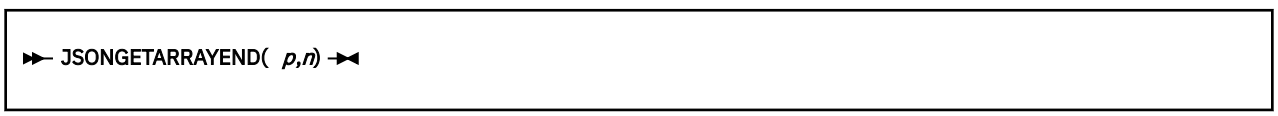
```
IUNSIGNED('ff_ff_ff_ff'xn) equals the largest UNSIGNED FIXED BIN(32) value.
```

# JSONGETARRAYEND

JSONGETARRAYEND(*p*,*n*) は、JSON テキスト内の次の文字 (空白は無視します) が右大括弧 ] であるかどうかを検査します。 この関数は、読み取られたバイト数に等しい *size_t* [1] 値を返します。

使用可能なバイト数 *n* がゼロより大きければ、JSONGETARRAYEND(*p*,*n*) は、右大括弧 ] をバッファーから読み取ろうとします。

- 空白文字の後の先頭文字が対象文字 (]) であれば、読み取られたバイト数は、その対象文字の 1 バイトと、その前の空白文字のバイト数を合計した値になります。
- 空白文字の後の先頭文字が対象文字 (]) でなければ、値としてゼロが戻されます。



**p**
読み取るバッファーのアドレスを指定するポインター。

**n**
バッファー内の使用可能バイト数を指定する *size_t* 値。

## JSONGETARRAYSTART

JSONGETARRAYSTART(*p*,*n*) は、JSON テキスト内の次の文字 (空白文字は無視) が左大括弧 [ であるかどうかを検査します。この関数は、読み取られたバイト数と等しい *size_t* [1] 値を戻します。

使用可能なバイト数 *n* がゼロより大きければ、JSONGETARRAYSTART(*p*,*n*) は、左大括弧 [ をバッファーから読み取ろうとします。

- 空白文字の後の先頭文字が対象文字 ([) であれば、読み取られたバイト数は、その対象文字の 1 バイトと、その前の空白文字のバイト数を合計した値になります。
- 空白文字の後の先頭文字が対象文字 ([) でなければ、値としてゼロが戻されます。

**p**
読み取るバッファーのアドレスを指定するポインター。

**n**
バッファー内の使用可能バイト数を指定する *size_t* 値。

## JSONGETCOLON

JSONGETCOLON(*p*,*n*) は、JSON テキスト内の次の文字 (空白は無視します) がコロンであるかどうかを検査します。 この関数は、読み取られたバイト数に等しい *size_t* [1] 値を返します。

使用可能なバイト数 *n* がゼロより大きければ、JSONGETCOLON(*p*,*n*) は、コロンをバッファーから読み取ろうとします。

- 空白文字の後の先頭文字が対象文字 (コロン) であれば、読み取られたバイト数は、その対象文字の 1 バイトと、その前の空白文字のバイト数を合計した値になります。
- 空白文字の後の先頭文字が対象文字でなければ、値としてゼロが戻されます。

**p**
読み取るバッファーのアドレスを指定するポインター。

**n**
バッファー内の使用可能バイト数を指定する *size_t* 値。

## JSONGETCOMMA

JSONGETCOMMA(*p*,*n*) は、JSON テキスト内の次の文字 (空白は無視します) がコンマであるかどうかを検査します。 この関数は、読み取られたバイト数に等しい *size_t* [1] 値を返します。

使用可能なバイト数 *n* がゼロより大きければ、JSONGETCOMMA(*p*,*n*) は、コンマをバッファーから読み取ろうとします。

- 空白文字の後の先頭文字が対象文字 (コンマ) であれば、読み取られたバイト数は、その対象文字の 1 バイトと、その前の空白文字のバイト数を合計した値になります。
- 空白文字の後の先頭文字が対象文字でなければ、値としてゼロが戻されます。

**p**
読み取るバッファーのアドレスを指定するポインター。

**n**
バッファー内の使用可能バイト数を指定する *size_t* 値。

## JSONGETMEMBER

JSONGETMEMBER は、JSON テキストからメンバー (名前/値のペア) を読み取ります。 この関数は、バッファーから読み取られたバイト数を示す *size_t* [1] 値を返します。

空白文字は JSON テキストのどの場所でも使用できますが、無視されます (ただし、読み取られたバイト総数には含められます)。

JSON テキストが無効の場合は、ERROR 条件が発生し、ONSUBCODE 組み込み関数によって無効文字の索引が戻されます。

JSONGETMEMBER の 3 番目の引数を省略すると、名前/値のペアが読み取られるだけになります。

3 番目の引数を構造体にする場合は、JSON テキスト内の名前がその構造体の中の名前と一致していなければなりません。 そうでない場合は、ERROR 条件が発生します。

**注 :** 構造体内のすべてのエレメントに名前/値のペアを指定する必要はありませんが、指定する名前の順序は構造体での順序と同じにする必要があります。

ターゲット内のいずれかの要素が CHARACTER 型になっている場合は、CODEPAGE コンパイラー・オプションに基づいて、JSON テキスト内の UTF-8 ソースからの変換が実行されます。

**p**
読み取るバッファーのアドレスを指定するポインター。

**n**
バッファー内の使用可能バイト数を指定する *size_t* 値。

**x**
変数参照。その名前/値のペアがバッファーから読み取られます。 その変数参照に以下の要素を組み込むことはできません。

- UNION
- 非計算要素
- GRAPHIC 要素
- COMPLEX 要素
- FIXED(p,q) 要素 (q < 0 または q > p)
- 名前のない要素

x は STRUCTURE タイプである可能性があります。

名前/値のペアは、JSON ストリングとしての変数名の後にコロン、その後に変数値、という形式でなければなりません。

### 例

バッファーに以下の JSON テキストが入っているとします。バッファー・アドレスが P、長さが N です。

```
{ "passes" : 3,
  "data" :
```

```
      [
           { "name" : "Mather",     "elevation" : 12100 }
         , { "name" : "Pinchot",    "elevation" : 12130 }
         , { "name" : "Glenn",      "elevation" : 11940 }
      ]
  }
```

JSON(CASE(ASIS)) オプションを指定して以下のコードをコンパイルすると、適切なサイズで構造体が割り振られ、その中に値が埋め込まれます。 その JSON コンパイラー・オプションが必要なのは、小文字の名前を受け入れるためです。

```
  dcl
    1 info based(q),
      2 count        fixed bin(31),
      2 data( passes refer(count) ),
        3 name       char(20) varying,
        3 elevation  fixed bin(31);

  read = 0;
  read += jsonGetObjectStart(p+read,n-read);
  read += jsonGetMember(p+read,n-read,passes);
  allocate info;
  read += jsonGetComma(p+read,n-read);
  read += jsonGetValue(p+read,n-read);
     read += jsonGetColon(p+read,n-read);
  read += jsonGetValue(p+read,n-read,data);
```

このコードは、バッファーにさらにデータが入っている場合でも、同じように正しく機能します。 次の例を参照してください。

```
{ "passes" : 5,
    "data" :
      [
           { "name" : "Muir",       "elevation" : 11980 }
         , { "name" : "Mather",     "elevation" : 12100 }
         , { "name" : "Pinchot",    "elevation" : 12130 }
         , { "name" : "Glenn",      "elevation" : 11940 }
         , { "name" : "Forester",   "elevation" : 13100 }
      ]
  }
```

## JSONGETOBJECTEND

JSONGETOBJECTEND(*p*,*n*) は、JSON テキスト内の次の文字 (空白は無視します) が右中括弧 } であるかどうかを検査します。 この関数は、読み取られたバイト数に等しい *size_t* [1] 値を返します。

使用可能なバイト数 *n* がゼロより大きければ、JSONGETOBJECTEND(*p*,*n*) は、右中括弧 } をバッファーから読み取ろうとします。

- 空白文字の後の先頭文字が対象文字 (}) であれば、読み取られたバイト数は、その対象文字の 1 バイトと、その前の空白文字のバイト数を合計した値になります。
- 空白文字の後の先頭文字が対象文字 (}) でなければ、値としてゼロが戻されます。

**p**
読み取るバッファーのアドレスを指定するポインター。

**n**
バッファー内の使用可能バイト数を指定する *size_t* 値。

## JSONGETOBJECTSTART

JSONGETOBJECTSTART(*p*,*n*) は、JSON テキスト内の次の文字 (空白は無視します) が左中括弧 { であるかどうかを検査します。この関数は、読み取られたバイト数に等しい *size_t* [1] 値を戻します。

使用可能なバイト数 *n* がゼロより大きければ、JSONGETOBJECTSTART(*p*,*n*) は、左中括弧 { をバッファーから読み取ろうとします。

- 空白文字の後の先頭文字が対象文字 ({) であれば、読み取られたバイト数は、その対象文字の 1 バイトと、その前の空白文字のバイト数を合計した値になります。
- 空白文字の後の先頭文字が対象文字 ({) でなければ、値としてゼロが戻されます。

**p**
読み取るバッファーのアドレスを指定するポインター。

**n**
バッファー内の使用可能バイト数を指定する *size_t* 値。

## JSONGETVALUE

JSONGETVALUE は、JSON テキストから値を読み取ります。 この関数は、バッファーから読み取られたバイト数を示す *size_t* [1] 値を返します。

空白文字は JSON テキストのどの場所でも使用できますが、無視されます (ただし、読み取られたバイト総数には含められます)。

JSON テキストが無効の場合は、ERROR 条件が発生し、ONSUBCODE 組み込み関数によって無効文字の索引が戻されます。

JSONGETVALUE の 3 番目の引数を省略すると、値が読み取られるだけになります。

3 番目の引数を構造体にする場合は、JSON テキスト内の名前がその構造体の中の名前と一致していなければなりません。 そうでない場合は、ERROR 条件が発生します。

**注 :** 構造体内のすべてのエレメントに名前/値のペアを指定する必要はありませんが、指定する名前の順序は構造体での順序と同じにする必要があります。

3 番目の引数を配列にする場合は、配列の値を省略できます。その場合、ターゲットの対応する要素は変更されません。

ターゲット内のいずれかの要素が CHARACTER 型になっている場合は、CODEPAGE コンパイラー・オプションに基づいて、JSON テキスト内の UTF-8 ソースからの変換が実行されます。

**p**
読み取るバッファーのアドレスを指定するポインター

**n**
バッファー内の使用可能バイト数を指定する *size_t* 値。

**x**
変数参照。その値がバッファーから読み取られます。

その変数参照に以下の要素を組み込むことはできません。

- UNION
- 非計算要素
- GRAPHIC 要素
- COMPLEX 要素
- FIXED(p,q) 要素 (q < 0 または q > p)
- 名前のない要素

x は STRUCTURE タイプである可能性があります。

### 例 1

以下のコードは、値 2、3、および 5 を配列 B に割り当てます。 返される値は、右大括弧 (]) の前の空白文字の数に 7 を加えたものです。

```
dcl b(3)    fixed bin;
dcl buffer  char(1000) var;
dcl p       pointer;
dcl n       fixed bin(31);


buffer = utf8( ' [ 2, 3, 5 ]' );
p = addrdata(buffer);
n = length(buffer);
read = jsonGetValue( p, n, b );
```

### 例 2

以下は、値 2 を B(1) に、3 を B(2) に割り当てて、B(3) を変更しないでおくコードです。 右大括弧 (]) の前の空白文字の数に 5 を足した値が戻されます。

```
dcl b(3)    fixed bin;
dcl buffer  char(1000) var;
dcl p       pointer;
dcl n       fixed bin(31);


buffer = utf8( ' [ 2, 3 ]' );
p = addrdata(buffer);
n = length(buffer);
read = jsonGetValue( p, n, b );
```

### 例 3

以下は、2 を C.D に 3 を C.E に割り当てるコードです。 13 以上の値が戻されます。

```
dcl 1 c, 2 d fixed bin, 2 e fixed bin;
dcl buffer  char(1000) var;
dcl p       pointer;
dcl n       fixed bin(31);

buffer = utf8( ' { "D" : 2, "E" : 3 } ' );
p = addrdata(buffer);
n = length(buffer);
read = jsonGetValue( p, n, c );
```

### 例 4

バッファーに以下の JSON テキストがあり、バッファー・アドレスが P その長さが N とします。

```
{ "PASSES" : 3,
  "DATA" :
    [
         { "NAME" : "Mather",  "ELEVATION" : 12100 }
       , { "NAME" : "Pinchot", "ELEVATION" : 12130 }
       , { "NAME" : "Glenn",   "ELEVATION" : 11940 }
    ]
}
```

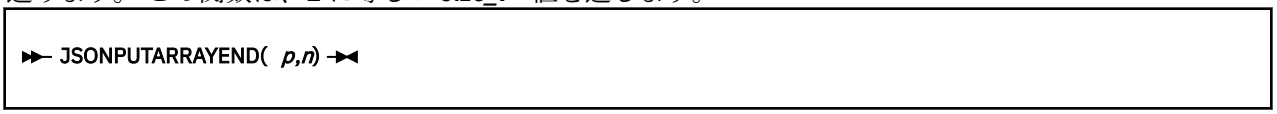
この場合、以下のコードで JSONGETVALUE を 1 回呼び出すと、構造体全体に値が埋め込まれます。

```
dcl
  1 info,
    2 passes       fixed bin(31),
    2 data(3),
      3 name       char(20) varying,
      3 elevation  fixed bin(31);
```

```
read = jsonGetValue( p, n, info );
```

## JSONPUTARRAYEND

JSONPUTARRAYEND(*p*,*n*) は、使用可能バイト数 *n* がゼロより大きい場合に右大括弧 ] をバッファーに書き込みます。 この関数は、1 に等しい *size_t* [1] 値を返します。

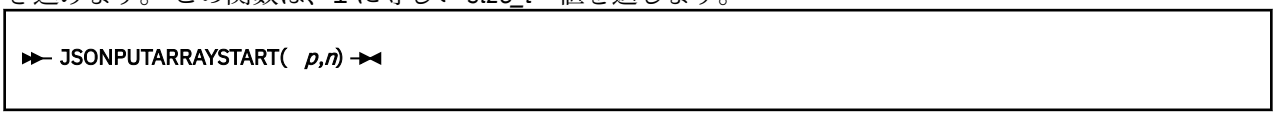



**p**
書き込むバッファーのアドレスを指定するポインター。

**n**
バッファー内の使用可能バイト数を指定する *size_t* 値。

## JSONPUTARRAYSTART

JSONPUTARRAYSTART(*p*,*n*) は、使用可能バイト数 *n* がゼロより大きい場合に左大括弧 [ をバッファーに書き込みます。 この関数は、1 に等しい *size_t* [1] 値を返します。

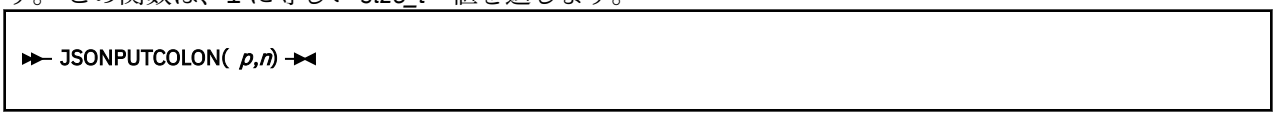



**p**
書き込むバッファーのアドレスを指定するポインター。

**n**
バッファー内の使用可能バイト数を指定する *size_t* 値。

## JSONPUTCOLON

JSONPUTCOLON(*p*,*n*) は、使用可能バイト数 *n* がゼロより大きい場合にコロンをバッファーに書き込みます。 この関数は、1 に等しい *size_t* [1] 値を返します。

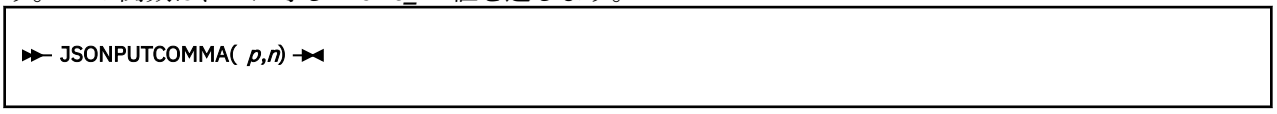



**p**
書き込むバッファーのアドレスを指定するポインター。

**n**
バッファー内の使用可能バイト数を指定する *size_t* 値。

## JSONPUTCOMMA

JSONPUTCOMMA(*p*,*n*) は、使用可能バイト数 *n* がゼロより大きい場合にコンマをバッファーに書き込みます。 この関数は、1 に等しい *size_t* [1] 値を返します。




**p**
書き込むバッファーのアドレスを指定するポインター。

**n**
バッファー内の使用可能バイト数を指定する *size_t*。

# JSONPUTMEMBER

JSONPUTMEMBER は、メンバー (名前/値のペア) を UTF-8 として JSON テキストに追加します。 この関数は、バッファーに書き込まれるバイト数を示す *size_t* [1] 値を返すか、指定されたバッファー・サイズがゼロであれば、書き込まれるすべての JSON テキストに必要となるバイト数を示す *size_t* 値を返します。

**p**
書き込むバッファーのアドレスを指定するポインター。

**n**
バッファー内の使用可能バイト数を指定する *size_t*。

**x**
変数参照。その値がバッファーに書き込まれます。 その変数参照に以下の要素を組み込むことはできません。

- UNION
- 非計算要素
- GRAPHIC 要素
- COMPLEX 要素
- FIXED(p,q) 要素 (q < 0 または q > p)
- 名前のない要素

x は STRUCTURE タイプである可能性があります。

**y**
名前を小文字、大文字、またはそのままのいずれで書き込むかを指定するオプション・パラメーター。

*y* は、値 LOWER、UPPER、または ASIS のいずれかを持つ文字定数でなければなりません。 これらの値は、大文字か小文字のいずれでも指定できます。 指定しない場合、デフォルトで JSON(CASE) オプションの値が使用されます。

## 例 1

```
dcl b(3)    fixed bin init(2,3,5);
dcl buffer  char(1000);
dcl p       pointer;
dcl n       fixed bin(31);

p = addr(buffer);
n = length(buffer);
written = jsonPutMember( p, n, b );
```

上記のコードによって、以下の UTF-8 ストリングをバッファーに書き込み、値 11 を変数 *written* に割り当てます。

```
"B":[2,3,5]
```

## 例 2

```
dcl 1 c, 2 d fixed bin init(2), 2 e fixed bin init(3);
dcl buffer  char(1000);
dcl p       pointer;
dcl n       fixed bin(31);

p = addr(buffer);
n = length(buffer);
written = jsonPutMember( p, n, c );
```

上記のコードによって、以下の UTF-8 ストリングをバッファーに書き込み、値 17 を変数 *written* に割り当てます。

```
"C":{"D":2,"E":3}
```

### 例 3

```
dcl 1 c(2), 2 d fixed bin init(2,3), 2 d fixed bin init(5,7);
dcl buffer  char(1000);
dcl p       pointer;
dcl n       fixed bin(31);

p = addr(buffer);
n = length(buffer);
written = jsonPutMember( p, n, c );
```

上記のコードによって、以下の UTF-8 ストリングをバッファーに書き込み、値 33 を変数 *written* に割り当てます。

```
"C":[{"D":2,"E":5},{"D":3,"E":7}]
```

### 例 4

```
dcl x       fixed bin(31) init(11);
dcl y       fixed bin(31) init(13);
dcl buffer  char(1000);
dcl p       pointer;
dcl n       fixed bin(31);

p = addr(buffer);
n = length(buffer);
written = 0;
written += jsonPutObjectStart( p+written, n-written );
written += jsonPutMember( p+written, n-written, x );
written += jsonPutComma( p+written, n-written );
written += jsonPutMember( p+written, n-written, y );
written += jsonPutObjectEnd( p+written, n-written );
```

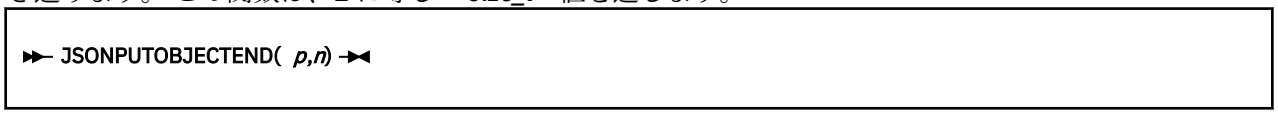
上記のコードによって、以下の UTF-8 ストリングをバッファーに書き込み、値 15 を変数 *written* に割り当てます。

```
{"X":11,"Y":13}
```

前の例とは異なり、このバッファーには完成した有効な JSON テキストが入ります。

## JSONPUTOBJECTEND

JSONPUTOBJECTEND(*p*,*n*) は、使用可能バイト数 *n* がゼロより大きい場合に右中括弧 } をバッファーに書き込みます。 この関数は、1 に等しい *size_t* [1] 値を返します。




**p**
書き込むバッファーのアドレスを指定するポインター。

**n**
バッファー内の使用可能バイト数を指定する *size_t*。

## JSONPUTOBJECTSTART

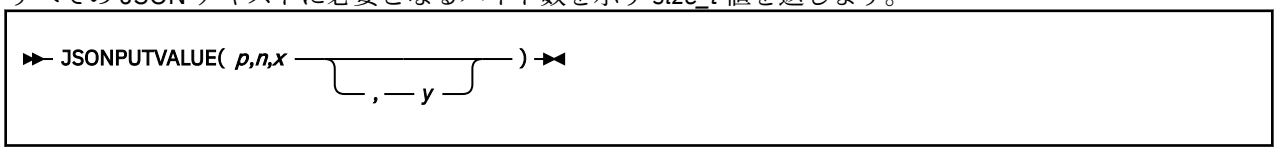
JSONPUTOBJECTSTART(*p*,*n*) は、使用可能バイト数 *n* がゼロより大きい場合に左中括弧 { をバッファーに書き込みます。 この関数は、1 に等しい *size_t* [1] 値を返します。

**p**
書き込むバッファーのアドレスを指定するポインター。

**n**
バッファー内の使用可能バイト数を指定する *size_t*。

## JSONPUTVALUE

JSONPUTVALUE は、値を UTF-8 として JSON テキストに追加します。 この関数は、バッファーに書き込まれるバイト数を示す *size_t*[1] 値を返すか、指定されたバッファー・サイズがゼロであれば、書き込まれるすべての JSON テキストに必要となるバイト数を示す *size_t* 値を返します。




**p**
書き込むバッファーのアドレスを指定するポインター。

**n**
バッファー内の使用可能バイト数を指定する *size_t*。

**x**
変数参照。その値がバッファーに書き込まれます。 その変数参照に以下の要素を組み込むことはできません。

- UNION
- 非計算要素
- GRAPHIC 要素
- COMPLEX 要素
- FIXED(p,q) 要素 (q < 0 または q > p)
- 名前のない要素

x は STRUCTURE タイプである可能性があります。

**y**
名前を小文字、大文字、またはそのままのいずれで書き込むかを指定するオプション・パラメーター。

*y* は、値 LOWER、UPPER、または ASIS のいずれかを持つ文字定数でなければなりません。 これらの値は、大文字か小文字のいずれでも指定できます。 指定しない場合、デフォルトで JSON(CASE) オプションの値が使用されます。

### 例 1

```
dcl b(3)    fixed bin init(2,3,5);
dcl buffer  char(1000);
dcl p       pointer;
dcl n       fixed bin(31);

p = addr(buffer);
n = length(buffer);
written = jsonPutValue( p, n, b );
```

上記のコードによって、以下の UTF-8 ストリングをバッファーに書き込み、値 7 を変数 *written* に割り当てます。

```
[2,3,5]
```

### 例 2

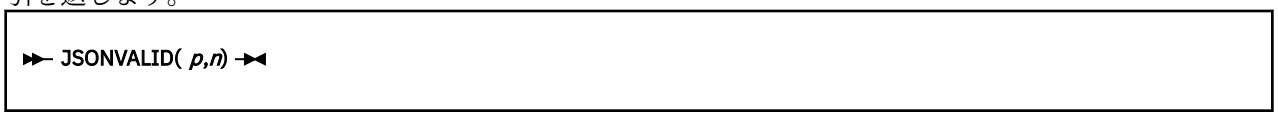
```
dcl 1 c, 2 d fixed bin init(2), 2 e fixed bin init(3);
dcl buffer  char(1000);
dcl p       pointer;
dcl n       fixed bin(31);

p = addr(buffer);
n = length(buffer);
written = jsonPutValue( p, n, c );
```

上記のコードによって、以下の UTF-8 ストリングをバッファーに書き込み、値 13 を変数 *written* に割り当てます。

```
{"D":2,"E":3}
```

## JSONVALID

JSONVALID は、バッファーに有効な JSON テキストが含まれているかどうかを判別します。 この関数は、JSON テキストが有効である場合はゼロの *size_t* [1] 値を返します。 その他の場合は、最初の無効バイトの索引を返します。

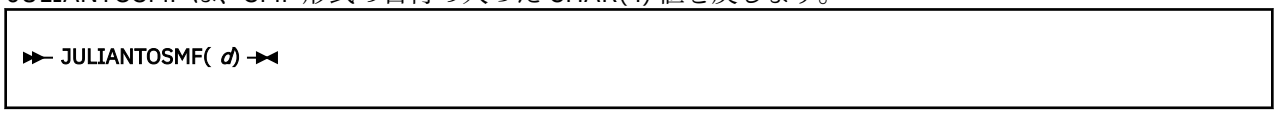



**p**
書き込むバッファーのアドレスを指定するポインター。

**n**
バッファー内の使用可能バイト数を指定する *size_t* 値。

## JULIANTOSMF

JULIANTOSMF は、SMF 形式の日付の入った CHAR(4) 値を戻します。




**d**
ユリウス形式 YYYYDDD の日付の入った CHAR(7) 変数。

## LBOUND

LBOUND は、*x* の次元 *y* の現行の下限を指定する FIXED BINARY 値で返します。

**x**
配列参照。 *x* の次元数は、*y* よりも小さくてはなりません。

**y**
*x* の特定の次元を指定する式。 必要な場合には、*y* は FIXED BINARY(31 ,0) に変換されます。 *y* は 1 以上でなければなりません。*y* が提供されない場合は、 デフォルトを 1 にします。

配列が 1 次の場合にのみ、*y* を省略できます。

CMPAT(V3) コンパイラー・オプションでは、LBOUND は FIXED BIN(63) 値を返します。 CMPAT(V2) および CMPAT(LE) の各コンパイラー・オプションでは、LBOUND は FIXED BIN(31) 値を返します。

## LBOUNDACROSS

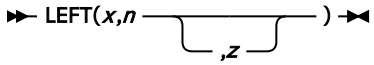
LBOUNDACROSS は、現行の下限の DIMACROSS 参照 *x* を指定する FIXED BINARY 値を戻します。

**x**
DIMACROSS 参照

CMPAT(V3) コンパイラー・オプションでは、LBOUNDACROSS は FIXED BIN(63) 値を返します。
CMPAT(V2) および CMPAT(LE) の各コンパイラー・オプションでは、LBOUNDACROSS は FIXED BIN(31) 値を返します。

## LEFT

LEFT は、ストリングの左端に、長さ *n* のストリング *x* を挿入し、 必要なら文字 *z* で右側を埋めた結果のストリングを戻します。




**x**
式。 *x* に、計算タイプ (必須) と文字タイプを保持している必要があります。 そうでない場合は、<synph><var>x</var></synph> は、CHARACTER に変換されます。

**n**
式。 *n* に、計算タイプ (必須) と文字タイプを保持している必要があります。 *n* に属性 FIXED BINARY(31,0) がない場合は、n はこの属性に変換されます。

**z**
式。 z が指定された場合、*z* は CHARACTER(1) NONVARYING タ イプでなければなりません。

**例**

```
dcl Source char value('One Hundred SCIDS Marks');
dcl Target char(30);

Target = left (Source, length(Target), '*');
           /* 'One Hundred SCIDS Marks*******'               */
```

*z* が省略される場合は、埋め込み文字にブランクが使われます。

## LENGTH

LENGTH は、*x* の現行の長さを指定するスケールのない REAL FIXED BINARY 値を戻します。

**x**
ストリング式、または LOCATES 属性および明示的な AREA 参照が指定された OFFSET 参照。 *x* が 2 進数の場合はビット・ストリングに変換され、それ以外の場 合は、文字ストリングに変換されます。

LENGTH 組み込み関数の例については、475 ページの『MAXLENGTH』を参照してください。

BIFPREC コンパイラー・オプションによって、戻される結果の精度が決まります。

LOCATES 属性および暗黙的な AREA 修飾が指定された OFFSET 参照に適用されるときには、以下のようになります。

- OFFSET 参照が非ヌルの場合、LENGTH は、位置指定されたデータのアドレスを返します。
- OFFSET 参照がヌルの場合、LENGTH は SYSNULL を返します。

## LINENO

LINENO は、*x* の現行の行数を指定するスケールのない REAL FIXED BINARY 値を戻します。

**x**
ファイル参照。

ファイルは、オープン状態で、PRINT 属性を持っている必要があります。 ファイルがオープンされていない場合、または PRINT 属性がない場合は、 0 が返されます。

BIFPREC コンパイラー・オプションによって、戻される結果の精度が決まります。

# LOCATION

LOCATION は、メンバー *x* を持つレベル 1 の構造体または共用体内にある *x* のバイト・ロケーションを示す FIXED BIN 値を返します。

**省略形:** LOC

**x**
構造体または共用体のメンバー名 *x* が構造体または共用体のメンバーでない場合は、値 0 が戻されます。 *x* に BIT 属性がある場合は、LOCATION によって戻される値は、 *x* を含むバイト位置です。

*x* は、添え字であってはいけません。

LOCATION は、制限付きの式で使用できます。 LOC(x) が次のいずれかに設定するのに使われる場合は、 *y* の前に *x* を宣言する必要があります。

- 定数エクステントを保持する必要がある変数 *y* のエクステント
- 定数値を保持する必要がある変数 *y* の値

CMPAT(V3) コンパイラー・オプションでは、LOCATION は FIXED BIN(63) 値を返します。 その他のすべての CMPAT オプションでは、FIXED BIN(31) 値を返します。

**例**

```
dcl 1 Table static,
      2 Tab2loc fixed bin(15) nonasgn init(loc(Tab2)),
                /* location is 0; gets initialized to 8  */
      2 Tab3loc fixed bin(15) nonasgn init(loc(Tab3)),
                /* location is 2; gets initialized to 808 */
      2 Length fixed bin nonasgn init(loc(End)),
                /* location is 4 */
      2 * fixed bin,
      2 Tab2(20,20)    fixed bin,
                /* location is 8 */
      2 Tab3(20,20)    fixed bin,
                /* location is 808 */

      2 F2_loc fixed bin nonasgn init(loc(F2)),
                /* location is 1608; gets initialized to  1612 */
      2 F2_bitloc fixed bin nonasgn init(bitloc(F2)),
                /* location is 1610; gets initialized to 1 */

      2 Flags,        /* location is 1612 */
        3 F1 bit(1),
        3 F2 bit(1),  /* bitlocation is 1 */
        3 F3 bit(1),
      2 Bits(16) bit, /* location is 1613 */
      2 End char(0);
```

# LOCNEWSPACE

LOCNEWSPACE(x, a) 組み込みサブルーチンは、*x* に関連付けられている LOCATES 属性で記述された変数タイプ用に *a* 内のスペースを割り振ります。

***x***
LOCATES 属性が指定された OFFSET 参照である必要があります。 *x* はスカラーである必要があります。

***a***
AREA 参照である必要があります。 *a* はスカラーである必要があります。

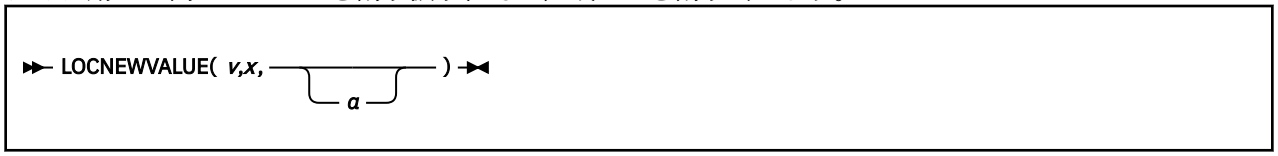
*a* が指定されなかった場合は、*x* の OFFSET 属性に AREA 参照が指定されている必要があり、LOCNEWSPACE がその区域内のスペースを割り振ります。

以下のコード・スニペットでは、 2 つの実行可能ステートメントは同等です。どちらのステートメントも、pool 領域から 32 バイトを割り振り、そのオフセットを name(1) に割り当てます。

```
          declare
              1 data based(data_ptr) unaligned,
                 2 actual_count fixed bin(31),
                 2 orderinfo(order_count refer( actual_count)),
                    3 name    offset(pool) locates(char(30) varying),
                    3 address offset(pool) locates(char(62) varying),
                 2 pool area(10_000);

       call locnewspace(name(1));
       call locnewspace(name(1), pool);
```

# LOCNEWVALUE

LOCNEWVALUE(v, x, a) 組み込みサブルーチンは、*x* に関連付けられている LOCATES 属性で記述された変数タイプ用に *a* 内のスペースを割り振り、その区域に *v* を割り当てます。




***V***
計算タイプおよびスカラーである必要があります。

***x***
LOCATES 属性が指定された OFFSET 参照である必要があります。 *x* はスカラーである必要があります。

***a***
AREA 参照である必要があります。 *a* はスカラーである必要があります。

*a* が指定されなかった場合は、*x* の OFFSET 属性に AREA 参照が指定されている必要があり、LOCNEWSPACE がその区域内のスペースを割り振ります。

以下の 3 つのステートメントは同等です。

- `call locnewvalue(v, x, a);`
- `call locnewspace(x, a);`
- `locval(x) = v;`

*x* の OFFSET 属性に AREA 属性が指定されている場合は、以下のステートメントが同等になります。

- `call locnewvalue(v, x);`
- `call locnewspace(x);`
- `locval(x) = v;`

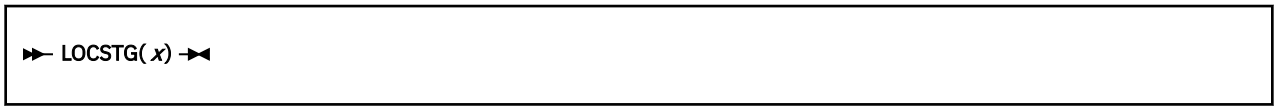
次のコード・スニペットでは、2 つの実行可能ステートメントが同等です。どちらのステートメントも、pool 区域内の 17 バイトを割り振り、そのオフセットを name(1) に割り当て、値 'Sherlock Holmes' を可変文字ストリングとして区域内のその位置に割り当てます。

```
declare
  1 data based(data_ptr) unaligned,
     2 actual_count fixed bin(31),
     2 orderinfo(order_count refer(actual_count)),
        3 name    offset(pool) locates(char(30) varying),
        3 address offset(pool) locates(char(62) varying),
     2 pool area(10_000);

call locnewvalue('Sherlock Holmes', name(1));
call locnewvalue('Sherlock Holmes', name(1), pool);
```

# LOCSTG

LOCSTG(*x*) は、LOCATES 属性を持つ *x* のすべての要素を保持するためにストレージに必要なバイト数を示す FIXED BIN 値を返します。

戻り値のタイプは CMPAT(V3) での FIXED BIN(63) です。 そうでない場合はタイプ FIXED BIN(31) です。




***x***

LOCATES 属性が指定された参照、または LOCATES 属性が指定されたサブエレメントを含む参照である必要があります。

**例**

以下の宣言では、参照 locstag(data) は値 96*actual_count を返します。

```
declare
   1 data based(data_ptr) unaligned,
       2 actual_count fixed bin(31),
       2 orderinfo(order_count refer( actual_count)),
          3 name     offset(pool) locates(char(30) varying),
          3 address offset(pool) locates(char(62) varying),
       2 pool area(10_000);
```

# LOCVAL

LOCVAL(*x*, *a*) は、*a* 区域内の *x* で指定されたオフセットにある値を返します。 値のタイプは、*x* の LOCATES 属性で指定されます。

***x***

LOCATES 属性が指定された OFFSET である必要があります。 区域 *a* に対する非ヌルの有効なオフセットである必要があります。

***a***

AREA 参照である必要があります。 *a* が指定されなかった場合は、*x* の OFFSET 属性に AREA 参照が指定されている必要があり、その区域からのオフセットであることが想定されます。

ADDR 組み込み関数の引数として LOCVAL 参照を使用しないでください。 そのような参照のアドレスを取得するには、POINTER 組み込み関数を、対応する OFFSET に適用します。

### 例

次の宣言では、locval(name(1)); と locval(name(1), pool); の 2 つの参照は同等です。 どちらの参照も、pool 内で name(1) に指定されたオフセットの位置にある可変値 char(30) を返します。

```
declare
    1 data based(data_ptr) unaligned,
       2 actual_count fixed bin(31),
       2 orderinfo(order_count refer( actual_count)),
          3 name     offset(pool) locates(char(30) varying),
          3 address offset(pool) locates(char(62) varying),
       2 pool area(10_000);
```

## LOG

LOG は、*x* の自然対数 (底 e の対数) の近似を浮動小数点 の値で戻します。 戻り値は、*x* の基数、モード、および精度になります。

**x**
式。 *x* は、ゼロより大きくなければなりません。

## LOGGAMMA

LOGGAMMA は、*x* のガンマ関数の対数の近似を浮動小数点の値で返します。

*x* のガンマは、次の式で与えられます。

$gamma(x) = \int_0^\infty (u^{x-1})(e^{-x})du$

LOGGAMMA は、*x* の基数、モード、および精度になります。

**x**
実数式。 *x* の値は、ゼロより大きくなければなりません。

## LOG2

LOG2 は、*x* の 2 進対数 (底 2 の対数) の近似を実数 の浮動小数点で戻します。 戻り値は、*x* の基数と精度になります。

**x**
実数式。 *x* の値は、ゼロより大きくなければなりません。

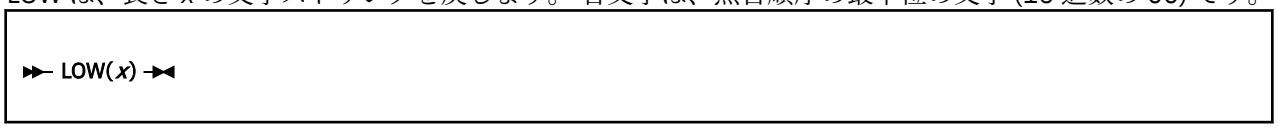
## LOG10

LOG10 は、*x* の常用対数 (底 10 の対数) の近似を実数 の浮動小数点で戻します。 戻り値は、*x* の基数と精度になります。

**x**
実数式。 この値はゼロより大きくなければなりません。

## LOW

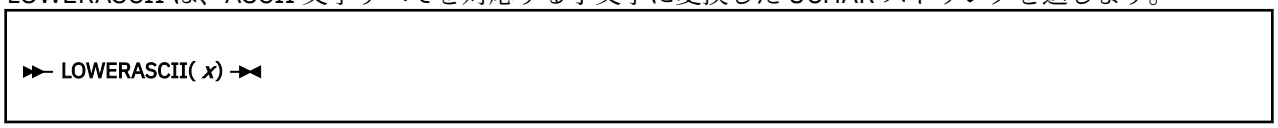
LOW は、長さ *x* の文字ストリングを戻します。 各文字は、照合順序の最下位の文字 (16 進数の 00) です。




**x**
式。 必要な場合には、*x* は正の実数固定小数点の 2 進数の値に変換されます。 *x* = 0 の場合は、結果はヌル文字ストリングです。

## LOWERASCII

LOWERASCII は、ASCII 文字すべてを対応する小文字に変換した UCHAR ストリングを返します。




**x**
式。 *x* は、UCHAR タイプを保持する必要があります。

LOWERASCII(*x*) は、TRANSLATE(*x, 'a...z', 'A...Z*) と同等です。

## LOWERCASE

LOWERCASE は、すべての文字を対応する小文字に変換した文字ストリングを返します。

**x**
式。 必要に応じ、*x* は文字に変換されます。

**c**
小文字にするコード・ページを指定する式。

LOWERCASE(*x*) は、TRANSLATE(*x, 'a...z', 'A...Z'*) と同等で、LOWERCASE(*x*, *c*) は TRANSLATE(*x, lowerc, upperc*) と同等です。 *lowerc* と *upperc* の値は、コード・ページ *c* の値によって決定されます。 LOWERCASE(*x*, *c*) を使用すると、英字「A...Z」が「a...z」に変換されるだけでなく、大文字の Ä-umlaut('4a'x) のような文字も小文字の ä-umlaut('c0'x) に変換されます。

例えば、Lower_01141 が次のように宣言されたとします。

```
dcl lower_01141 char
 value( (
              '8182838485868788'8991929394959697'x
           || '9899A2A3A4A5A6A7A8A9424445464748'x
           || '4951525354555657'586A708C8D8E9CC0'x
           || 'CBCDCECFD0DBDDDE'x
       ) );
```

また、Upper_01141 が次のように宣言されたとします。

```
dcl upper_01141 char
  value( (
              'C1C2C3C4C5C6C7C8C9D1D2D3D4D5D6D7'x
           || 'D8D9E2E3E4E5E6E7E8E9626465666768'x
           || '6971727374757677'78E080ACADAE9E4A'x
           || 'EBEDEEEF5AFBFDFE'x
       ) );
```

この場合、LOWERCASE(*x*, 1141 ) は、TRANSLATE( *x*, Lower_01141, Upper_01141 ) と同じになります。

付録に、サポートされている *c* の値の *lowerc* と *upperc* の値がリストされています。 詳しくは、『615 ページの『付録 A 制限値』』を参照してください。

## LOWERLATIN1

LOWERLATIN1 は、ASCII および Latin-1 補助文字すべてを対応する小文字に変換した UCHAR ストリングを返します。

**x**
式。 *x* は、UCHAR タイプを保持する必要があります。

DIAERESIS(ÿ) および SHARP S (ß) を持つ文字 Y は変更されません。

## LOWER2

LOWER2(*x*,*n*) は、次の値を戻します。

$$\mathrm{floor}(x * (2^{-n}))$$

**注 :** LOWER2(*x*,*n*) は、アセンブラー SRA(x,n) と同等です。

***x***
式。 *n* は、計算タイプでなくてはなりません。

**n**
式。 *n* は、計算タイプでなくてはなりません。

*x* が SIGNED REAL FIXED BIN(p,0) の場合、その結果は同じ属性です。 それ以外の場合、*x* は SIGNED REAL FIXED BIN(p,0) に変換され、その結果は同じ属性を持ちます。

*n* が負の場合、または *n* が M より大きい場合は、結果は定義されません。

**例**

```
lower2 (+6,1)                    /*  Produces 3   */
lower2 (-6,1)                    /*  Produces -3  */
lower2 (-7,1)                    /*  Produces -4  */
```

## MAINNAME

MAINNAME は、現行呼び出しスタックでの MAIN 関数の名前である CHARACTER ストリングを返します。

## 最大

MAX は、複数の式の集合から最大値を戻します。

**x および y**
式。

すべての引数は、実数でなければなりません。 結果は、引数の共通の基数およびスケールを持つ実数です。

引数が次の精度を使用した固定小数点の引数である場合は、次のようになります。

```
(p1,q1),(p2,q2),...,(pn,qn)
```

結果の精度は、次の式によって指定されます。

```
(min(N,max(p1-q1,p2-q2,...,pn-qn) + max(q1,q2,...,qn)),max(q1,q2,...,qn))
```

この場合 N は、使用できる最大の桁数です。

引数が次の精度を使用した浮動小数点である場合は、次のようになります。

```
p1,p2,p3,...pn
```

結果の精度は、次の式によって指定されます。

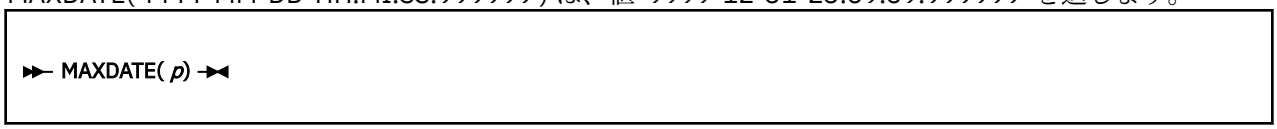
```
max(p1,p2,p3,...pn)
```

許可される引数 の最大数は 64 です。

# MAXDATE

MAXDATE は、指定された日付/時刻パターンに対応する最遅日付/時刻値を含む文字ストリングを返します。

MAXDATE('YYYY-MM-DD-HH.MI.SS.999999') は、値 '9999-12-31-23.59.59.999999' を返します。

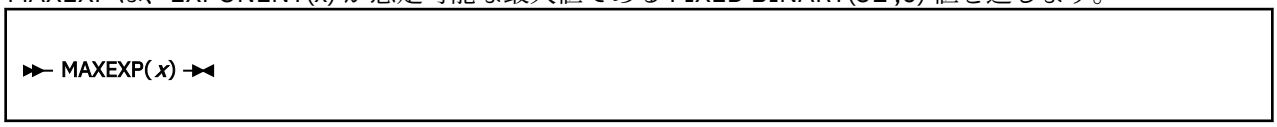



**p**
サポートされている日付/時刻パターンの 1 つを指定します。

許可される日付/時刻パターンを 382 ページの表 65 にリストしてあります。

# MAXEXP

MAXEXP は、EXPONENT(x) が想定可能な最大値である FIXED BINARY(31 ,0) 値を返します。




**x**
式。 *x* は、REAL 属性および FLOAT 属性をとる必要があります。

MAXEXP(x) は定数で、制限付き式に使用できます。

### 例 (Intel の値)

```
maxexp(x) = 128      for x float bin(p), p <= 21
maxexp(x) = 1024      for x float bin(p), 21 < p <= 53
maxexp(x) = 16384      for x float bin(p), 53 < p
```

```
maxexp(x) = 128      for x float dec(p), p <= 6
maxexp(x) = 1024      for x float dec(p), 6 < p <= 16
maxexp(x) = 16384      for x float dec(p), 16 < p
```

**例 (AIX の値)**

```
maxexp(x) = 128      for x float bin(p), p <= 21
maxexp(x) = 1024      for x float bin(p), 21 < p <= 53
maxexp(x) = 1024      for x float bin(p), 53 < p

maxexp(x) = 128      for x float dec(p), p <= 6
maxexp(x) = 1024      for x float dec(p), 6 < p <= 16
maxexp(x) = 1024      for x float dec(p), 16 < p
```

**例 (z/OS の 16 進値)**

```
maxexp(x) = 63      for x float bin(p), p <= 21
maxexp(x) = 63      for x float bin(p), 21 < p <= 53
maxexp(x) = 63      for x float bin(p), 53 < p

maxexp(x) = 63      for x float dec(p), p <= 6
maxexp(x) = 63      for x float dec(p), 6 < p <= 16
maxexp(x) = 63      for x float dec(p), 16 < p
```

**例 (z/OS の IEEE 2 進浮動小数点数値)**

```
maxexp(x) = 128     for x float bin(p), p <= 21
maxexp(x) = 1024    for x float bin(p), 21 < p <= 53
maxexp(x) = 16384   for x float bin(p), 53 < p

maxexp(x) = 128     for x float dec(p), p <= 6
maxexp(x) = 1024    for x float dec(p), 6 < p <= 16
maxexp(x) = 16384   for x float dec(p), 16 < p
```

**例 (z/OS の IEEE 10 進浮動小数点数値)**

```
maxexp(x) = 97      for x float dec(p), p <= 7
maxexp(x) = 385     for x float dec(p), 7 < p <= 16
maxexp(x) = 6145    for x float dec(p), 16 < p
```

# MAXVAL

MAXVAL は、その数値オペランドが取ることができる最大値を返します。

**x**
式。 *x* は、REAL 属性を持っていなければなりません。

MAXVAL(*x*) >= *x* および MINVAL(*x*) <= *x* は常に true です。

次の表は、*x* が FLOAT の場合、MAXVAL(*x*)、MINVAL(*x*)、および HUGE(*x*) の間の関係を示しています。

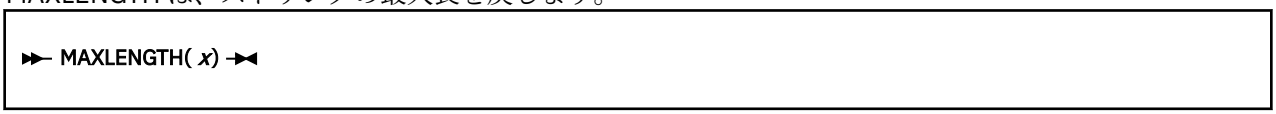
| 組み込み関数 | 同等 |
|---|---|
| MAXVAL(*x*) | HUGE(*x*) |
| MINVAL(*x*) | -HUGE(*x*) |

詳しくは、447 ページの『HUGE』および 546 ページの『TINY』を参照してください。

MAXVAL(*x*) は定数で、制限付き式に使用できます。

# MAXLENGTH

MAXLENGTH は、ストリングの最大長を戻します。




**x**
式。 *x* に、計算タイプ (必須) とストリング・タイプを保持している必要があります。 そうでないと、文字に変換されます。

**例**

```
dcl x char(20);
dcl y char(20) varying;

x, y = '';

x = copy( '*', length(x) );    /* fills x with '*'   */
y = copy( '*', length(y) );    /* leaves y unchanged */

x = copy( '-', maxlength(x) ); /* fills x with '-'   */
y = copy( '-', maxlength(y) ); /* fills y with '-'   */
```

*y* の最初の割り当ては、未変更のままになります。上記のコードの断片で使用される場合、*length(y)* は、ゼロを戻す (*y* は VARYING であり、事前に '' とセットされている) ためです。

しかし、*y* への 2 番目の割り当てでは、*maxlength(y)* が 20 (*y* の宣言された長さ) を戻すため、- 符号が 20 個入ります。

# MEMCONVERT

MEMCONVERT は、ソース・バッファー内のデータを指定のソース・コード・ページから指定のターゲット・コード・ページに変換し、結果をターゲット・バッファーに保管し、ターゲット・バッファーに書き込まれたバイト数を示す *size_t* [1] 値を返します。 また、変換で使用する手法を指定するオプション・パラメーター *t* もとります。

**p**
ターゲット・バッファーのアドレス。

**n**
ターゲット・バッファーの長さ。

**c**
ターゲット・コード・ページ。

**q**
ソース・バッファーのアドレス。

**m**
ソース・バッファーの長さ。

**d**
ソース・コード・ページ。

**t**
変換で使用する手法を指定する文字ストリングまたは変数。 *t* の長さは 8 以下です。

バッファー長は、負以外で、計算タイプでなければなりません。 バッファー長はタイプ *size_t* に変換されます。

いずれかのバッファー長がゼロの場合は、結果はゼロです。

コード・ページは、計算タイプでなければならず、タイプ FIXED BINARY (31,0) に変換されます。 コード・ページでは、有効かつサポートされているコード・ページを指定する必要があります。

# MEMCOLLAPSE

MEMCOLLAPSE は、指定された文字のすべての複数オカレンスを 1 つのオカレンスに置き換え、同時にその文字の先行インスタンスと末尾インスタンスもトリムして、ソース・バッファーの内容をターゲット・バッファーに充てんします。 ターゲット・バッファーに書き込まれるバイト数を示す *size_t* 値が返されます。

**p**
ターゲット・バッファーのアドレスを指定します。

**m**
ターゲット・バッファーの長さを指定します。計算タイプでなければならず、タイプ *size_t* に変換されます。

**q**
ソース・バッファーのアドレスを指定します。

**n**
ソース・バッファーの長さを指定します。 計算タイプでなければならず、タイプ *size_t* に変換されます。 これは、負でない値でなければなりません。

**z**
CHARACTER(1) NONVARYING タイプを持つ必要がある式。

**i**
オプションの式。これは計算式でなければならず、必要に応じて *size_t* に変換されます。 指定しない場合、*I* のデフォルト値は 1 です。 *I* < 1 の場合、デフォルト値 1 が使用されます。

戻り値は、以下のように、ターゲット・バッファーのアドレスまたはターゲット・バッファーのサイズによって異なります。

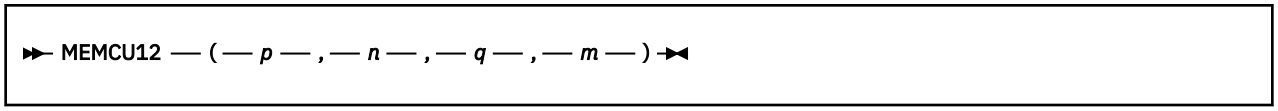
- ターゲット・バッファーのアドレスがゼロ (ヌル) の場合は、想定書き込みバイト数が返されます。
- ターゲット・バッファーの大きさが十分でなかった場合には、値 -1 が返されます。
- ターゲット・バッファーの大きさが十分であった場合には、バッファーに書き込まれたバイト数が返されます。
- ターゲット・バッファーには、ソース・バッファー内の *i* 番目の文字より前のすべての文字 (縮小されていない) と、その後に *n* 番目の位置以降からの、必要に応じてスクイーズおよびトリムされたすべての文字が含まれます。

**例**

```
dcl s  char(20);
dcl t  char(20);
dcl cx fixed bin(31);

s  = '...abc....def...gh..';
cx = memcollapse(sysnull(), 0, addr(s), stg(s), '.');
       /* cx = 10          */
cx = memcollapse(addr(t), stg(t), addr(s), stg(s), '.');
       /* cx = 10          */
       /* t = 'abc.def.gh' */
```

# MEMCU12

MEMCU12 は、ソース・バッファー内のデータを UTF-8 から UTF-16 に変換し、結果をターゲット・バッファーに保管し、ターゲット・バッファーに書き込まれたバイト数を示す *size_t* [1] 値を返します。




**p**
ターゲット・バッファーのアドレス。

**n**
ターゲット・バッファーの長さ。

**q**
ソース・バッファーのアドレス。

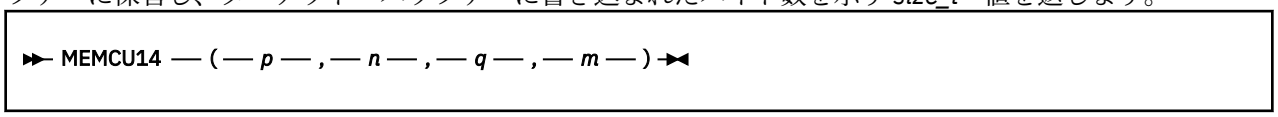
**m**
ソース・バッファーの長さ。

バッファー長は、負以外で、計算タイプでなければなりません。 バッファー長はタイプ *size_t* に変換されます。

ターゲット・バッファーが小さすぎる場合、または、ソース UTF-8 が無効な場合は、値 -1 が返されます。

## MEMCU14

MEMCU14 は、ソース・バッファー内のデータを UTF-8 から UTF-32 に変換し、結果をターゲット・バッファーに保管し、ターゲット・バッファーに書き込まれたバイト数を示す *size_t* [1] 値を返します。




**p**
ターゲット・バッファーのアドレス。

**n**
ターゲット・バッファーの長さ。

**q**
ソース・バッファーのアドレス。

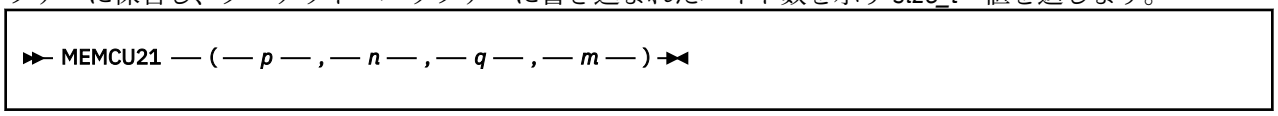
**m**
ソース・バッファーの長さ。

バッファー長は、負以外で、計算タイプでなければなりません。 バッファー長はタイプ *size_t* に変換されます。

ターゲット・バッファーが小さすぎる場合、または、ソース UTF-8 が無効な場合は、値 -1 が返されます。

## MEMCU21

MEMCU21 は、ソース・バッファー内のデータを UTF-16 から UTF-8 に変換し、結果をターゲット・バッファーに保管し、ターゲット・バッファーに書き込まれたバイト数を示す *size_t* [1] 値を返します。




**p**
ターゲット・バッファーのアドレス。

**n**
ターゲット・バッファーの長さ。

**q**
ソース・バッファーのアドレス。

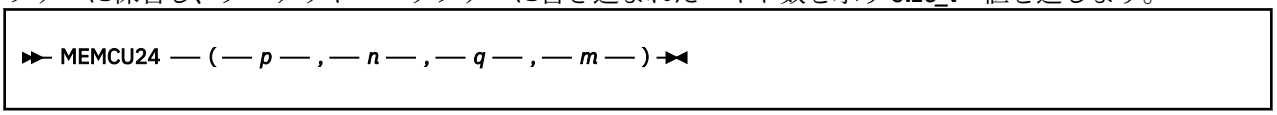
**m**
ソース・バッファーの長さ。

バッファー長は、負以外で、計算タイプでなければなりません。 バッファー長はタイプ *size_t* に変換されます。

ターゲット・バッファーが小さすぎる場合は、値 -1 が返されます。 ソースには有効な UTF-16 が入っていなければなりません。そうでない場合のこの関数の動作は指定されていません。

## MEMCU24

MEMCU24 は、ソース・バッファー内のデータを UTF-16 から UTF-32 に変換し、結果をターゲット・バッファーに保管し、ターゲット・バッファーに書き込まれたバイト数を示す *size_t*[1] 値を返します。




**p**
ターゲット・バッファーのアドレス。

**n**
ターゲット・バッファーの長さ。

**q**
ソース・バッファーのアドレス。

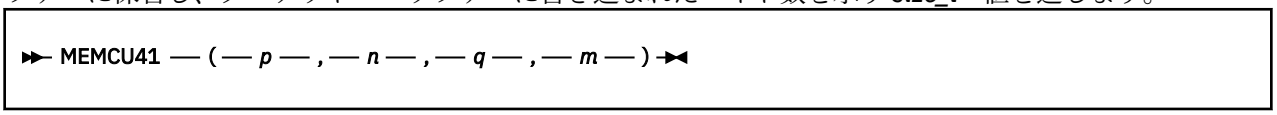
**m**
ソース・バッファーの長さ。

バッファー長は、負以外で、計算タイプでなければなりません。 バッファー長はタイプ *size_t* に変換されます。

ターゲット・バッファーが小さすぎる場合は、値 -1 が返されます。 ソースには有効な UTF-16 が入っていなければなりません。そうでない場合のこの関数の動作は指定されていません。

## MEMCU41

MEMCU41 は、ソース・バッファー内のデータを UTF-32 から UTF-8 に変換し、結果をターゲット・バッファーに保管し、ターゲット・バッファーに書き込まれたバイト数を示す *size_t*[1] 値を返します。




**p**
ターゲット・バッファーのアドレス。

**n**
ターゲット・バッファーの長さ。

**q**
ソース・バッファーのアドレス。

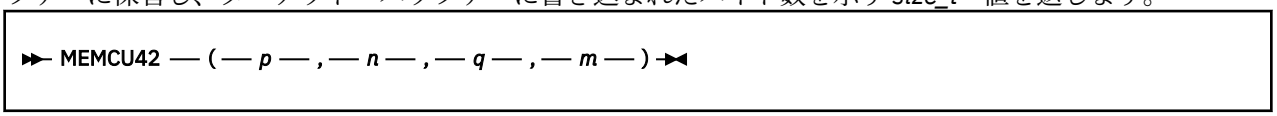
**m**
ソース・バッファーの長さ。

バッファー長は、負以外で、計算タイプでなければなりません。 バッファー長はタイプ *size_t* に変換されます。

ターゲット・バッファーが小さすぎる場合、または、ソース UTF-32 が無効な場合は、値 -1 が返されます。

## MEMCU42

MEMCU42 は、ソース・バッファー内のデータを UTF-32 から UTF-16 に変換し、結果をターゲット・バッファーに保管し、ターゲット・バッファーに書き込まれたバイト数を示す *size_t*[1] 値を返します。




**p**
ターゲット・バッファーのアドレス。

**n**
ターゲット・バッファーの長さ。

**q**
ソース・バッファーのアドレス。

**m**
ソース・バッファーの長さ。

バッファー長は、負以外で、計算タイプでなければなりません。 バッファー長はタイプ *size_t* に変換されます。

ターゲット・バッファーが小さすぎる場合、または、ソース UTF-32 が無効な場合は、値 -1 が返されます。

# MEMINDEX

MEMINDEX は、指定したサブストリングのバッファー内での開始位置を示す *size_t* [1] 値を返します。

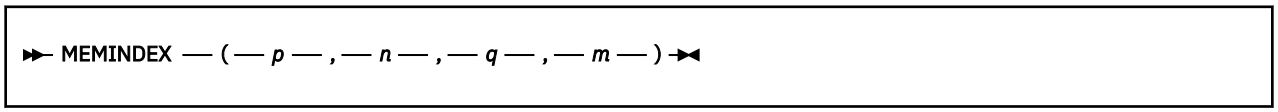
3 つの引数を使用する関数の構文は以下のとおりです。

**p**
検索されるバッファーのアドレス。

**n**
検索されるバッファーの長さ。

**x**
検索のターゲットとして使用するストリング式。

4 つの引数を使用する関数の構文は以下のとおりです。




**p**
検索される最初のバッファーのアドレス。

**n**
検索される最初のバッファーの長さ。

**q**
検索のターゲットとして使用する 2 番目のバッファーのアドレス。

**m**
検索のターゲットとして使用する 2 番目のバッファーの長さ。

バッファー長は、負以外で、計算タイプでなければなりません。 バッファー長はタイプ *size_t* に変換されます。

3 つの引数を使用する場合、ターゲットのストリング式は、タイプ CHARACTER (PICTURE を含む)、GRAPHIC、UCHAR、または WIDECHAR を保持している必要があります。 バッファー長は、そのストリング・タイプの単位の数として解釈されます。

4 つの引数を使用する場合は、バッファー長はバイト数を指定し、文字検索が 行われます。

VARYING、VARYING4、VARYINGZ のストリング *X* とストリング *Y* の場合、関数 MEMINDEX( ADDRDATA(X), LENGTH(X), Y ) は、INDEX( X, Y ) と同じ値を戻します。

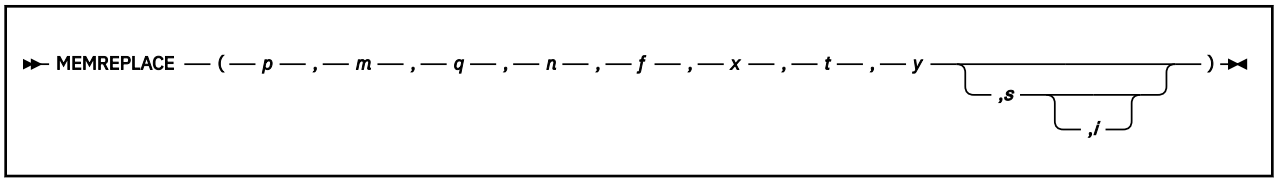
**例**

```
dcl cb(128*1024) char(1);
dcl wb(128*1024) widechar(1);
dcl pos fixed bin(31);
/* 128K bytes searched for the character string 'test' */
```

```
pos = memindex( addr(cb), stg(cb), 'test' );
/* 256K bytes searched for the string 'test' as widechar */
pos = memindex( addr(wb), stg(wb), wchar('test') );
```

# MEMREPLACE

MEMREPLACE は、指定された 3 番目のバッファーの 1 つ以上のオカレンスを 4 番目のバッファーで置き換えて、 ソース・バッファーの内容をターゲット・バッファーに充てんし、ターゲット・バッファーに書き込まれるバイト数を示す *size_t* 値を返します。




**p**
ターゲット・バッファーのアドレスを指定します。

**m**
ターゲット・バッファーの長さを指定します。 長さは負でない値でなければなりません。 計算タイプを保持していなければならず、*size_t* タイプに変換されます。

**q**
ソース・バッファーのアドレスを指定します。

**n**
ソース・バッファーの長さを指定します。 長さは負でない値でなければなりません。 計算タイプを保持していなければならず、*size_t* タイプに変換されます。

**f**
置き換えられるバイトを含むバッファーのアドレスを指定します。

**x**
バッファー f の長さをバイト単位で指定します。 長さは負でない値でなければなりません。 計算タイプを保持していなければならず、*size_t* タイプに変換されます。

**t**
バッファー p 内のバッファー f のバイトを置き換えるために使用されるバイトを含むバッファーのアドレスを指定します。

**y**
バッファー t の長さをバイト単位で指定します。 長さは負でない値でなければなりません。 計算タイプを保持していなければならず、*size_t* タイプに変換されます。

**s**
f および x で定義されたバッファーの検索を開始するソース・バッファー内の位置を指定するオプションの式。 計算タイプを保持していなければならず、*size_t* タイプに変換されます。 s のデフォルト値は 1 です。 s が 1 より小さい場合、または s が 1 + n より大きい場合は、ゼロ・バイトがターゲット・バッファーに書き込まれます。

**i**
f を t で置き換える必要がある最大回数を指定するオプションの式。 計算タイプを保持していなければならず、*size_t* タイプに変換されます。 i のデフォルト値は 1 です。i は負であってはなりません。 i の値が 0 の場合、ソース・バッファー内のすべての f のオカレンスが t に置き換えられます。

戻り値は、以下のように、ターゲット・バッファーのアドレスまたはターゲット・バッファーのサイズによって異なります。

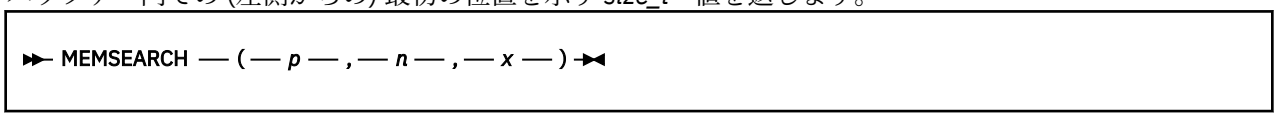
- ターゲット・バッファーのアドレスがゼロ (ヌル) の場合は、想定書き込みバイト数が返されます。
- ターゲット・バッファーの大きさが十分でなかった場合には、値 -1 が返されます。
- ターゲット・バッファーの大きさが十分であった場合には、バッファーに書き込まれたバイト数が返されます。

```
dcl ein char(50) var value('reserved from #date# till #date#.');
dcl aus char(80) var;
```

```
dcl cx fixed bin(31);

dcl f   char(6);
dcl t   char(10);

f = '#date#';
t = '2018/05/01';

cx = memreplace( addrdata(aus), maxlength(aus),
                 addrdata(ein), length(ein),
                 addrdata(f), length(f),
                 addrdata(t), length(t));
     /* cx = 37                                        */
     /* aus = 'reserved from 2018/05/01 till #date#.' */
cx = memreplace( addrdata(aus), maxlength(aus),
                 addrdata(ein), length(ein),
                 addrdata(f), length(f),
                 addrdata(t), length(t),16,1);
     /* cx = 37                                        */
     /* aus = 'reserved from #date# till 2018/05/01.' */
cx = memreplace( addrdata(aus), maxlength(aus),
                 addrdata(ein), length(ein),
                 addrdata(f), length(f),
                 addrdata(t), length(t),,0);
     /* cx = 41                                              */
     /* aus = 'reserved from 2018/05/01 till 2018/05/01.' */
```

# MEMSEARCH

MEMSEARCH は、 指定のストリングにある任意の文字、グラフィック、uchar、またはワイド文字がある、バッファー内での (左側からの) 最初の位置を示す *size_t* [1] 値を返します。




**p**
検索されるバッファーのアドレス。

**n**
検索されるバッファーの長さ

**x**
ストリング式

バッファー長は、負以外で、計算タイプでなければなりません。 バッファー長はタイプ *size_t* に変換されます。

ストリング式 *x* は、 タイプ CHARACTER (PICTURE を含む)、GRAPHIC、UCHAR、または WIDECHAR を保持している必要があります。 バッファー長は、そのストリング・タイプの単位の数として解釈されます。

アドレス *p* および長さ *n* は、*x* に存在する任意の文字、グラフィック、uchar、またはワイド文字を検索する「ストリング」を指定します。

バッファー長 *n* がゼロであるか、または *x* が ヌル・ストリングである場合は、結果はゼロです。

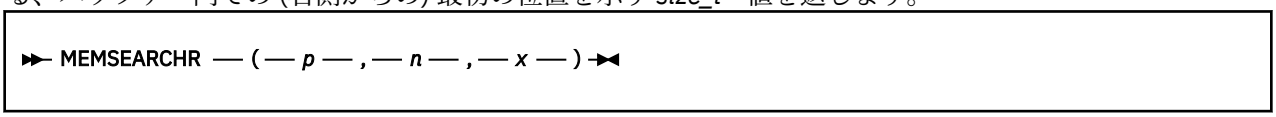
バッファーに *x* がない場合は、結果はゼロです。

**例**

```
dcl cb(128*1024) char(1);
dcl wb(128*1024) widechar(1);
dcl pos fixed bin(31);

/* 128K bytes searched from the left for a numeric */
pos = memsearch( addr(cb), stg(cb), '012345789' );

/* 256K bytes searched from the left for a widechar '0' or '1' */
pos = memsearch( addr(wb), stg(wb), '0030_0031'wx );
```

# MEMSEARCHR

MEMSEARCHR は、 指定のストリングにある任意の文字、グラフィック、uchar、またはワイド文字がある、バッファー内での (右側からの) 最初の位置を示す *size_t* [1] 値を返します。




**p**
検索されるバッファーのアドレス。

**n**
検索されるバッファーの長さ

**x**
ストリング式

バッファー長は、負以外で、計算タイプでなければなりません。 バッファー長はタイプ *size_t* に変換されます。

ストリング式 *x* は、 タイプ CHARACTER (PICTURE を含む)、GRAPHIC、UCHAR、または WIDECHAR を保持している必要があります。 バッファー長は、そのストリング・タイプの単位の数として解釈されます。

アドレス *p* および長さ *n* は、*x* に存在する任意の文字、グラフィック、uchar、またはワイド文字を検索する「ストリング」を指定します。

バッファー長 *n* がゼロであるか、または *x* が ヌル・ストリングである場合は、結果はゼロです。

バッファーに *x* がない場合は、結果はゼロです。

**例**

```
dcl cb(128*1024) char(1);
dcl wb(128*1024) widechar(1);
dcl pos fixed bin(31);

/* 128K bytes searched from the right for a numeric */
pos = memsearchr( addr(cb), stg(cb), '012345789' );

/* 256K bytes searched from the right for a widechar '0' or '1' */
pos = memsearchr( addr(wb), stg(wb), '0030_0031'wx );
```

# MEMSQUEEZE

MEMSQUEEZE は、指定された文字のすべての複数オカレンスを 1 つのオカレンスで置き換えて、ソース・バッファーの内容をターゲット・バッファーに充てんします。 ターゲット・バッファーに書き込まれるバイト数を示す *size_t* 値が返されます。

**p**
ターゲット・バッファーのアドレスを指定します。

**m**
ターゲット・バッファーの長さを指定します。計算タイプでなければならず、タイプ *size_t* に変換されます。

**q**
ソース・バッファーのアドレスを指定します。

**n**
ソース・バッファーの長さを指定します。 計算タイプでなければならず、タイプ *size_t* に変換されます。 これは、負でない値でなければなりません。

**z**
CHARACTER(1) NONVARYING タイプを持つ必要がある式。

**i**
オプションの式。これは計算式でなければならず、必要に応じて *size_t* に変換されます。 指定しない場合、*i* のデフォルト値は 1 です。 *i* < 1 の場合、デフォルト値 1 が使用されます。

戻り値は、以下のように、ターゲット・バッファーのアドレスまたはターゲット・バッファーのサイズによって異なります。

- ターゲット・バッファーのアドレスがゼロ (ヌル) の場合は、想定書き込みバイト数が返されます。
- ターゲット・バッファーの大きさが十分でなかった場合には、値 -1 が返されます。
- ターゲット・バッファーの大きさが十分であった場合には、バッファーに書き込まれたバイト数が返されます。
- ターゲット・バッファーには、ソース・バッファー内の *i* 番目の文字より前のすべての文字 (縮小されていない) と、その後に *n* 番目の位置以降からの、必要に応じてスクイーズおよびトリムされたすべての文字が含まれます。

**例**

```
dcl s  char(20);
dcl t  char(20);
dcl cx fixed bin(31);

s  = '...abc....def...gh..';
cx = memsqueeze(sysnull(), 0, addr(s), stg(s), '.');
      /* cx = 12         */
cx = memsqueeze(addr(t), stg(t), addr(s), stg(s), '.');
      /* cx = 12         */
      /* t = '.abc.def.gh.' */
```

# MEMVERIFY

MEMVERIFY は、指定のストリングにない (左側からの) 最初の文字、グラフィック、 uchar、またはワイド文字のバッファー内での位置を示す *size_t* [1] 値を返します。

**p**
検索されるバッファーのアドレス。

**n**
検索されるバッファーの長さ。

**x**
ストリング式。

バッファー長は、負以外で、計算タイプでなければなりません。 バッファー長はタイプ *size_t* に変換されます。

ストリング式 *x* は、 タイプ CHARACTER (PICTURE を含む)、GRAPHIC、UCHAR、または WIDECHAR を保持している必要があります。 バッファー長は、そのストリング・タイプの単位の数として解釈されます。

アドレス *p* および長さ *n* は、*x* に存在しない 任意の文字、グラフィック、uchar、またはワイド文字を検索する「ストリング」を指定します。

バッファー長 *n* がゼロであるか、または *x* が ヌル・ストリングである場合は、結果はゼロです。

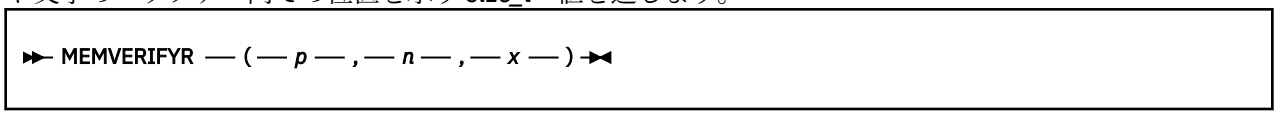
バッファー内のすべての文字、グラフィック、uchars、またはワイド文字が *x* に存在する場合、結果はゼロです。

**例**

```
 dcl cb(128*1024) char(1);
dcl wb(128*1024) widechar(1);
dcl pos fixed bin(31);

/* 128K bytes searched from the left for a non-numeric */
pos = memverify( addr(cb), stg(cb), '012345789' );

/* 256K bytes searched from the left for the a non-blank widechar */
pos = memverify( addr(wb), stg(wb), '0020'wx );
```

# MEMVERIFYR

MEMVERIFYR は、指定のストリングにない (右側からの) 最初の文字、グラフィック、 uchar、またはワイド文字のバッファー内での位置を示す *size_t* [1] 値を返します。




**p**
検索されるバッファーのアドレス。

**n**
検索されるバッファーの長さ。

**x**
ストリング式。

バッファー長は、負以外で、計算タイプでなければなりません。 バッファー長はタイプ *size_t* に変換されます。

ストリング式 *x* は、 タイプ CHARACTER (PICTURE を含む)、GRAPHIC、UCHAR、または WIDECHAR を保持している必要があります。 バッファー長は、そのストリング・タイプの単位の数として解釈されます。

アドレス *p* および長さ *n* は、*x* に存在しない 任意の文字、グラフィック、uchar、またはワイド文字を検索する「ストリング」を指定します。

バッファー長 *n* がゼロであるか、または *x* が ヌル・ストリングである場合は、結果はゼロです。

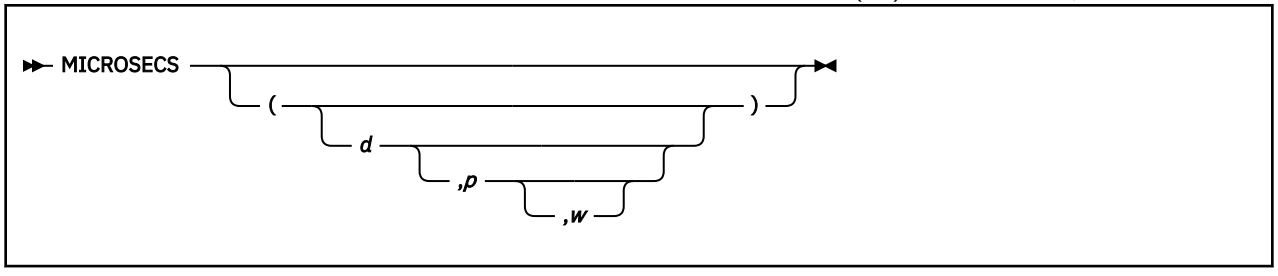
バッファー内のすべての文字、グラフィック、uchars、またはワイド文字が *x* に存在する場合、結果はゼロです。

**例**

```
dcl cb(128*1024) char(1);
 dcl wb(128*1024) widechar(1);
 dcl pos fixed bin(31);

 /* 128K bytes searched from the right for a non-numeric */
 pos = memverify( addr(cb), stg(cb), '012345789' );

 /* 256K bytes searched from the right for the a non-blank widechar */
 pos = memverify( addr(wb), stg(wb), '0020'wx );
```

# MICROSECS

MICROSECS は、日付 *d* に対応するマイクロ秒数である FIXED BINARY(63) 値を返します。



**d**
日付を表すストリング式を指定します。 ある場合、*d* は、パターン *p* で指定された日付/時刻を表す文字ストリングとして入力日付を指定します。 *d* を省略した場合は、TIMESTAMP() によって返された値が想定されます。

*d* に、計算タイプ (必須) と文字タイプを保持している必要があります。 そうでないと、文字に変換されます。

**p**
サポートされている日付/時刻パターンの 1 つを指定します。 *p* が省略されている場合は、TIMESTAMP パターン (つまり、「YYYY-MM-DD-HH.MI.SS.999999」) が想定されます。

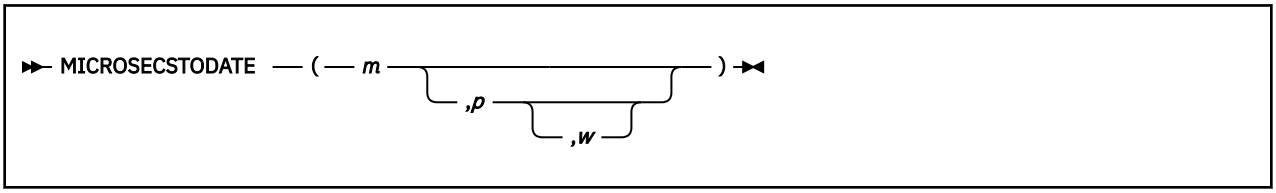
*p* に、計算タイプ (必須) と文字タイプを保持している必要があります。 そうでないと、文字に変換されます。

**w**
整数に変換できる式 (1950 など) を指定します。 負の場合、コードを実行するときに年の値から減算されるようにオフセットを指定します。 省略される場合は、*w* は、WINDOW コンパイル時オプションに指定された値に デフォルトで設定されます。

許可されるパターンを 382 ページの表 65 にリストしてあります。 リリアン形式の説明については、380 ページの『日付/時刻組み込み関数』 を参照してください。

## MICROSECSTODATE

MICROSECSTODATE は、指定された日付/時刻パターンの日付を含む、NONVARYING 文字ストリングを返します。 指定された日付/時刻パターンは、マイクロ秒数に対応しています。




**m**
マイクロ秒数 (リリアン形式) を指定します。 *m* は、計算タイプでなければなりません。また、必要な場合には、FIXED BIN(63) に変換されます。

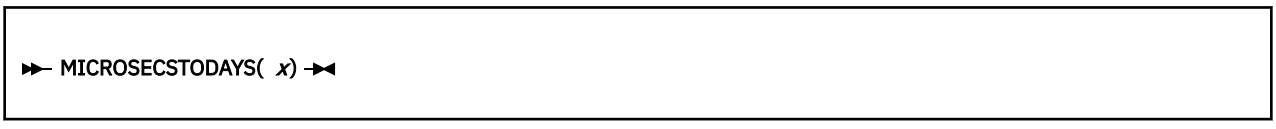
**p**
サポートされている日付/時刻パターンの 1 つを指定します。 *p* が省略されている場合は、TIMESTAMP パターン (つまり、「YYYY-MM-DD-HH.MI.SS.999999」) が想定されます。

**w**
整数に変換できる式 (1950 など) を指定します。 負の場合、コードを実行するときに年の値から減算されるようにオフセットを指定します。 省略される場合は、*w* は、WINDOW コンパイル時オプションに指定された値に デフォルトで設定されます。

許可されるパターンを 382 ページの表 65 にリストしてあります。 リリアン形式の説明については、380 ページの『日付/時刻組み込み関数』 を参照してください。

## MICROSECSTODAYS

MICROSECSTODAYS は、マイクロ秒数 *x* を日数に変換した数を表す FIXED BINARY(31) 値を返します。1 日に達しないマイクロ秒は無視されます。



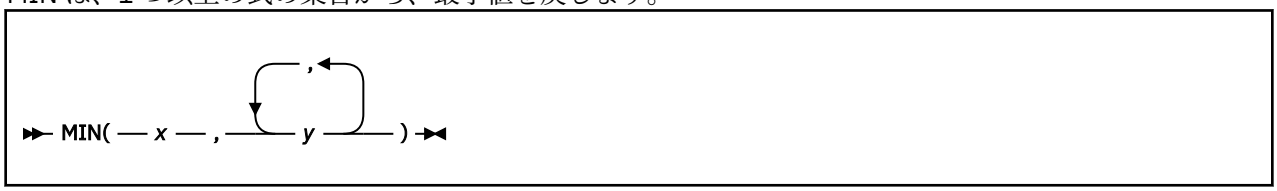

**x**
マイクロ秒数を指定する式。 *x* の値は、計算タイプでなければならず、必要に応じて FIXED BINARY(63) に変換されます。

MICROSECSTODAYS(x) は、*x*/(24*60*60*1_000_000) と同じです。

例えば、534 ページの『SECS』を参照してください。

## MIN

MIN は、1 つ以上の式の集合から、最小値を戻します。




**x および y**
式。

すべての引数は、実数でなければなりません。 結果は、引数の共通の基数およびスケールを持つ実数です。

結果の精度は、472 ページの『最大』の説明と同じです。

許可される引数 の最大数は 64 です。

## MINDATE

MINDATE は、指定された日付/時刻パターンに対応する最早日付/時刻値を含む文字ストリングを返します。

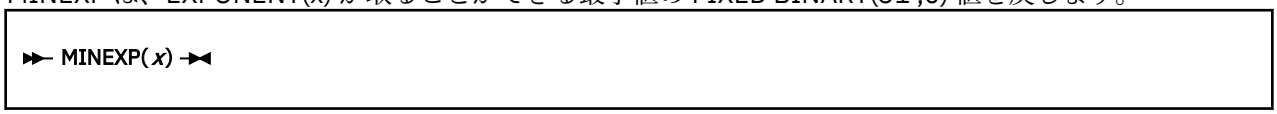
MINDATE('YYYY-MM-DD-HH.MI.SS.999999') は、NONULLDATE コンパイラー・オプションでは 値 '1582-10-14-00.00.00.000000' を返し、NULLDATE コンパイラー・オプション では '0001-01-01-00.00.00.000000' を返します。

**p**
サポートされている日付/時刻パターンの 1 つを指定します。

許可される日付/時刻パターンを 382 ページの表 65 にリストしてあります。

## MINEXP

MINEXP は、EXPONENT(x) が取ることができる最小値の FIXED BINARY(31 ,0) 値を戻します。




**x**
式。 *x* は、REAL 属性および FLOAT 属性をとる必要があります。

MINEXP(x) は定数で、制限付き式に使用できます。

**例 (Intel の値)**

```
minexp(x) = -125    for x float bin(p), p <= 21
minexp(x) = -1021    for x float bin(p), 21 < p <= 53
minexp(x) = -16831    for x float bin(p), 53 < p

minexp(x) = -125    for x float dec(p), p <= 6
minexp(x) = -1021    for x float dec(p), 6 < p <= 16
minexp(x) = -16831    for x float dec(p), 16 < p
```

**例 (AIX の値)**

```
minexp(x) = -125     for x float bin(p), p <= 21
minexp(x) = -1021     for x float bin(p), 21 < p <= 53
```

```
minexp(x) = -968     for x float bin(p), 53 < p

minexp(x) = -125     for x float dec(p), p <= 6
minexp(x) = -1021     for x float dec(p), 6 < p <= 16
minexp(x) = -968     for x float dec(p), 16 < p
```

**例 (z/OS の 16 進値)**

```
minexp(x) = -64    for x float bin(p), p <= 21
minexp(x) = -64    for x float bin(p), 21 < p <= 53
minexp(x) = -50    for x float bin(p), 53 < p

minexp(x) = -64    for x float dec(p), p <= 6
minexp(x) = -64    for x float dec(p), 6 < p <= 16
minexp(x) = -50    for x float dec(p), 16 < p
```

**例 (z/OS の IEEE 2 進浮動小数点数値)**

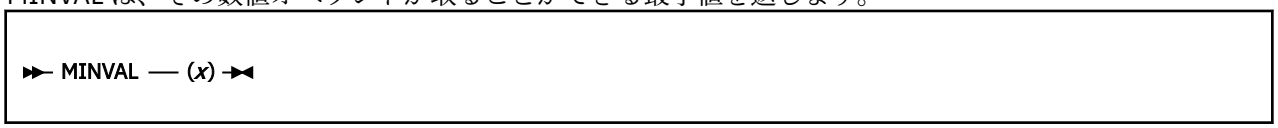
```
minexp(x) = -125    for x float bin(p), p <= 21
minexp(x) = -1021   for x float bin(p), 21 < p <= 53
minexp(x) = -16381  for x float bin(p), 53 < p

minexp(x) = -125    for x float dec(p), p <= 6
minexp(x) = -1021   for x float dec(p), 6 < p <= 16
minexp(x) = -16381  for x float dec(p), 16 < p
```

**例 (z/OS の IEEE 10 進浮動小数点数値)**

```
minexp(x) = -94       for x float dec(p), p <= 7
minexp(x) = -382      for x float dec(p), 7 < p <= 16
minexp(x) = -6142     for x float dec(p), 16 < p
```

# MINVAL

MINVAL は、その数値オペランドが取ることができる最小値を返します。




**x**
式。 *x* は、REAL 属性を持っていなければなりません。

MAXVAL(*x*) >= *x* および MINVAL(*x*) <= *x* は常に true です。

次の表は、*x* が FLOAT の場合、MAXVAL(*x*)、MINVAL(*x*)、および HUGE(*x*) の間の関係を示しています。

| 組み込み関数 | 同等 |
|---|---|
| MAXVAL(*x*) | HUGE(*x*) |
| MINVAL(*x*) | -HUGE(*x*) |

詳しくは、447 ページの『HUGE』および546 ページの『TINY』を参照してください。

MINVAL(*x*) は定数で、制限付き式に使用できます。

# MOD

MOD は、ある値で別の値を除算したときの剰余のモジュラー等価を返します。

MOD は、(x - R)/y = n のような負でない最小値 R を戻します。

*n* は、整数値です。 すなわち、R は *y* で割り切れるように、*x* から減算する 負でない最小値です。

**x**
実数式。

**y**
実数式。 *y* = 0 の場合は、ZERODIVIDE 条件が発生します。

結果 R は、引数と共通の基数およびスケールを持つ実数です。 結果が浮動小数点の場合、精度は *x* および *y* の精度より大きくなります。 結果が固定小数点の場合、精度は以下のようになります。

```
(min(n,p2-q2+max(q1,q2)),max(q1,q2))
```

(p1,q1) と (p2,q2) は、それぞれ *x* と *y* の精度であり、n には、FIXED DECIMAL の場合は N が入り、FIXED BINARY の場合は M が入ります。

*x* および *y* の位取り係数が異なる固定小数点の 場合は、R が計算される前に、小さい位取り係数の引数が大きい 位取り係数に変換されます。 変換が失敗すると、結果は不定になります。

結果の属性が FIXED BIN であり、1 つ以上のオペランドに属性 UNSIGNED FIXED BIN がある場合、以下の条件の両方が真でない限り、結果は SIGNED 属性を持ちます。

- すべてのオペランドが UNSIGNED FIXED BIN である。
- RULES(ANS) コンパイラー・オペランドが有効である。

上記のいずれかの条件が true でない場合、UNSIGNED オペランドはそれぞれ SIGNED に変換されます。 オペランドが大きすぎる場合は、変換によって以下が行われます。

- SIZE が有効である場合は SIZE 条件が発生します。
- SIZE が有効でない場合は負の値が生成されます。

### 例

次の例では、MOD 組み込み関数と REM 組み込み関数を対比しています。

```
rem( +10, +8 ) = 2
mod( +10, +8 ) = 2

rem( +10, -8 ) = 2
mod( +10, -8 ) = 2

rem( -10, +8 ) = -2
mod( -10, +8 ) = 6

rem( -10, -8 ) = -2
mod( -10, -8 ) = 6
```

### 関連情報

525 ページの『REM』
REM は、*x* を *y* で除算した剰余を戻します。

## MPSTR

MPSTR は、論理境界でストリングを切り捨て、混合文字ストリングを戻します。

例えば、MPSTR は、2 バイト文字をバイト間では切り捨てません。 戻されるストリングの長さは、式 *x* の長さ、または *y* で指定された値と等しくなります。 ストリングの処理は、式 *r* で選択された規則によって次のように判別されます。

**x**
文字ストリングの結果が出る式。 必要な場合には、*x* の値は文字に変換されます。

**r**
1 文字の結果が出る式。 式は GRAPHIC であることはできません。必要な場合には、文字に変換されます。

式 *r* は、ストリングの処理に使われる規則を指定します。 *r* で使われる文字およびその規則を次に示します。

**V または v**
混合ストリング *x* を検証し、混合ストリングを戻します。

**S または s**
ヌル DBCS ストリングを除去して新しいストリングを作成し、混合ストリングを戻します。

V および S の両方が指定される場合は、順番が指定された順番に関係なく、V は S より優先されます。

V を指定せずに S が指定された場合は、ストリング x は有効なストリングであると想定されます。 ストリングが無効の場合は、結果は不定になります。

**注 :** Intel および AIX では、パラメーター *r* は無視されます。

**y**
式。 必要に応じて、*y* は実数の固定小数点 2 進数値に変換されます。 *y* が省略される場合は、長さはタイプ変換の規則によって決められます。 *y* の値は、負であってはいけません。 *y* = 0 の場合は、結果はヌル文字ストリングです。 *y* が *x* を入れるのに必要な長さより大きい場合は、結 果はブランクで埋められます。 *y* が *x* を入れるのに必要な長さより小さい場合は、結果は右から 超過した文字を廃棄する (SBCS 文字の場合) か、必要な DBCS 文字 (2 バイト一組) を廃棄して切り捨てます。

## MULTIPLY

MULTIPLY は、*p* および *q* で指定された精度で、 *x* と *y* の積を返します。

結果の基数、スケール、およびモードは、PRECTYPE コンパイラー・オプションによって規則が変更されない限り、式の評価規則によって決まります。

**x および y**
式。

**p**
演算を通して維持する桁数を指定する制限付き式。

**q**
結果の位取り係数を指定する制限付き式。 固定小数点の場合に、*q* が省略されると、位取り係数 ゼロが想定されます。 結果が浮動小数点の場合は、*q* は省略します。

なお、FIXED DECIMAL に適用する際は、計算結果が指定した精度 *p* にとって大き過ぎても最大インプリメンテーション値未満である場合は、以下のようになります。

- SIZE が使用不可の場合は、FIXEDOVERFLOW 条件は発生せず、結果は切り捨てられます。
- SIZE が使用可能な場合には、SIZE 条件が発生します。

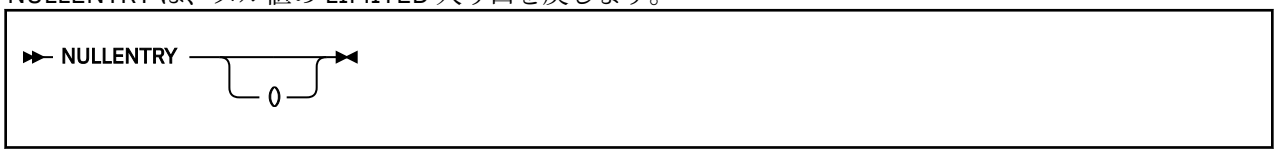
なお、上記テキストは、デフォルトでないコンパイラー・オプション DECIMAL(FOFLONMULT) が有効な場合、偽になります。 その場合、SIZE が使用不可で結果が大きすぎると、FIXEDOVERFLOW が発生します。

## NULL

NULL は、ヌル・ポインター値を戻します。 ヌル・ポインター値は、変数の生成を識別しません。 ヌル・ポインター値は、ハンドルに割り当てられ、ハンドルと比較されます。 ヌル・ポインター値は、組み込み関数の値のオフセット変数への割り当てによって OFFSET に変換されます。

## NULLENTRY

NULLENTRY は、ヌル値の LIMITED 入り口を戻します。




NULLENTRY は、他の入り口変数に割り当てたり、他の入り口変数と比較したりすることができます。

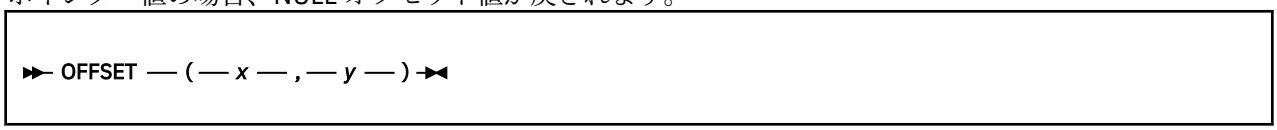
NULLENTRY を使用して、静的ストレージで入り口変数を初期化できます。

組み込み関数 PLISRTA、PLISRTB、PLISRTC、PLISRTD の引数の 1 つとして NULLENTRY を使用することはできません。

ENTRYADDR(NULLENTRY) は、SYSNULL と同じ値を戻します。

## OFFSET

OFFSET は、ポインター参照 *x* から派生し、領域 *y* を基準とする相対オフセット値を戻します。 x が NULL ポインター値の場合、NULL オフセット値が戻されます。



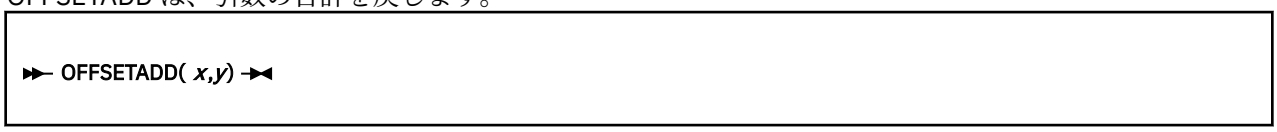

**x**
ポインター参照。 x は、区域 *y* 内で基底付き変数の生成を識別するか、ヌル・ポインター値でなければなりません。

**y**
区域参照。

*x* がエレメント参照の場合は、*y* は要素変数でなければなりません。

## OFFSETADD

OFFSETADD は、引数の合計を戻します。



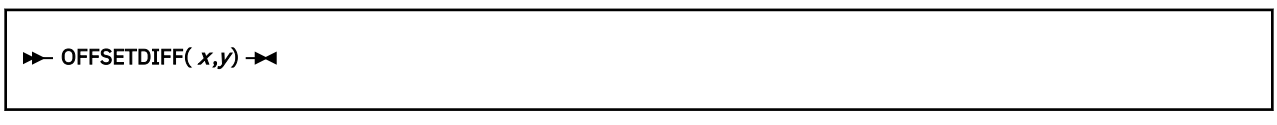

**x**
式。 *x* は、OFFSET として指定されなければなりません。

**y**
式。 *y* は計算タイプでなければならず、FIXED BINARY に変換されます。

## OFFSETDIFF

OFFSETDIFF は、引数間の算術差を表す FIXED BIN 値を戻します。

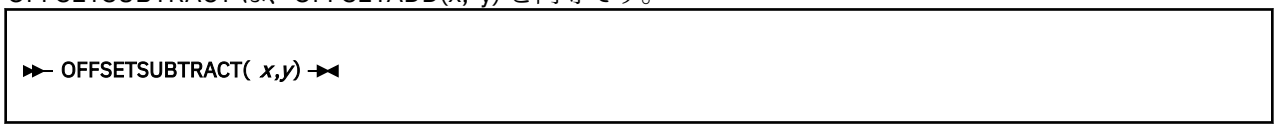
戻り値のタイプは OFFSETSIZE(4) での FIXED BIN(31) または OFFSETSIZE(8) での FIXED BIN(63) です。



**x および y**
式。 両方とも OFFSET として指定されなければなりません。

## OFFSETSUBTRACT

OFFSETSUBTRACT は、OFFSETADD(x,-y) と同等です。

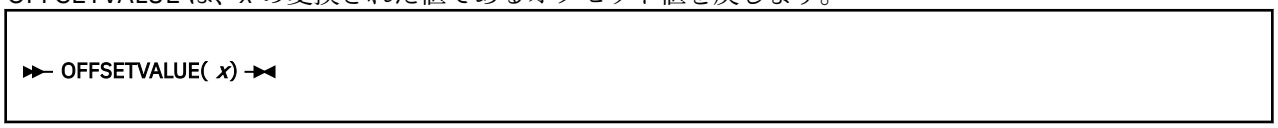



**x**
式。 *x* は、OFFSET として指定されなければなりません。

**y**
式。 *y* は計算タイプでなければならず、FIXED BINARY に変換されます。

## OFFSETVALUE

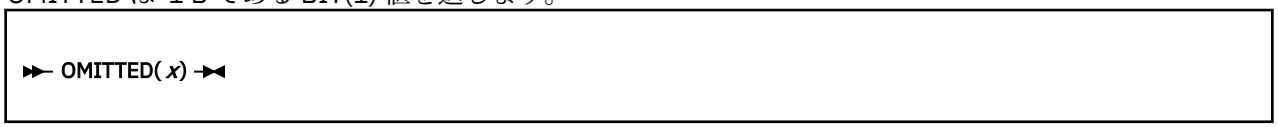
OFFSETVALUE は、*x* の変換された値であるオフセット値を戻します。




**x**
式。 *x* は計算タイプでなければならず、FIXED BINARY に変換されます。

## OMITTED

OMITTED が含まれているプロシージャーの呼び出しで、*x* という名前のパラメーターが省略される場合、OMITTED は '1'B である BIT(1) 値を返します。




**x**
BYADDR 属性付きのレベル -1 の添え字なしパラメーター。

**注 :** この引数は、呼び出し側コードの対応する ENTRY 宣言で OPTIONAL として宣言する必要があります。

## ONACTUAL

ONACTUAL は、ASSERTION 条件を発生させた ASSERT COMPARE ステートメントの「実際の」値である不変文字ストリングを返します。 GRAPHIC タイプまたは WIDECHAR タイプの式の場合は、ヌル・ストリングが返されます。

この関数は、ASSERTION 条件の ON ユニットのコンテキスト内、あるいは ASSERTION 条件の暗黙処置として発生した ERROR 条件または FINISH 条件の ON ユニットのコンテキスト内にあります。

コンテキスト以外で使用されている場合は、ヌル・ストリングが返されます。

## ONAREA

ONAREA は、AREA 条件が発生する AREA 参照の名前である値を持つ文字ストリングを返します。 参照に DBCS 名が含まれる場合、返されるストリングは混合文字ストリングです。 この関数は、AREA 条件、あるいは AREA 条件の暗黙処置として発生した ERROR 条件または FINISH 条件の、ON ユニット (またはその動的派生物のいずれか) のコンテキスト内にあります。

ONAREA 組み込み関数がコンテキスト外で使用された場合は、ヌル・ストリングが返されます。

AREA 参照が長すぎるか、複雑すぎる場合は、ヌル・ストリングが返されます。

## ONCHAR

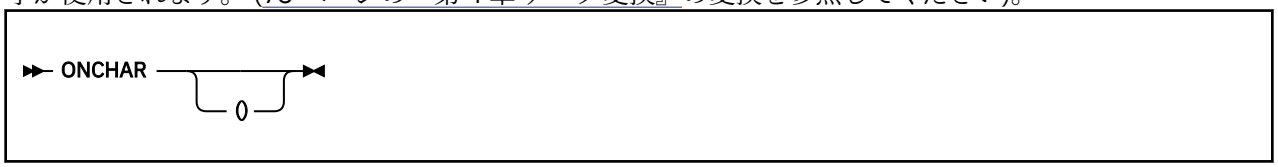
ONCHAR は、CONVERSION 条件が発生する原因となる文字を 含む character(1) ストリングを戻します。この関数は、CONVERSION 条件の ON ユニットの中か、CONVERSION 条件の暗黙処置として発生した ERROR 条件または FINISH 条件の ON ユニット (またはそれらの ON ユニットから動的に派生したもの) の中にあります。

ONCHAR 組み込み関数がコンテキスト以外で使われる場合は、ブランクが戻されま す。

## ONCHAR 疑似変数

ONCHAR 疑似変数は、ONCHAR 組み込み関数の現行値を設定します。

疑似変数に割り当てられたエレメント値は、長さ 1 の文字値に変換されます。 変換の再試行時に新しい文字が使用されます。 (73 ページの『第 4 章 データ変換』の変換を参照してください)。




疑似変数は、コンテキスト以外では使用してはいけません。

## ONCODE

ONCODE 組み込み関数は、最後の条件の原因によって決まる固定小数点の 2 進数 の値を提供します。

ONCODE は、特定の条件、例えば ERROR 条件を発生するさまざま な状況を区別するのに使用できます。検出された条件およびエラーに対応するコードについては、特定の条件を参照 してください。

ONCODE は、条件コードである実数の固定小数点の 2 進数の値を戻します。 ONCODE は、ON ユニットまたはその動的子孫のコンテキストにあります。 条件コードのすべての定義は「メッセージおよびコード」に記載されています。

コンテキスト以外で ONCODE が使われている場合は、ゼロが戻されます。

## ONCONDCOND

ONCONDCOND は、値が CONDITION 条件が発生する条件の名前である不変文字ストリングを戻します。

名前が DBCS 名の場合は、ONCONDCOND は混合文字ストリングとして戻されます。 ONCONDCOND は、次の状況のコンテキストにあります。

- CONDITION ON ユニットまたはそれらから動的に派生したもの

- CONDITION 条件をトラップする ANYCONDITION ON ユニット、または このような ON ユニットから動的に派生したもの

コンテキスト以外で ONCONDCOND が使われている場合は、 ヌル・ストリングが戻されます。

## ONCONDID

ONCONDID (ON 条件 ID の省略形) は、ON ユニットでハンドルされている条件を 識別する FIXED BINARY(31 ,0) 値を戻します。 ONCODE は、ON ユニットまたはそれらから動的に派生したものの 1 つのコンテキストにあります。

ONCONDID によって戻される値は、次の DECLARE ステートメントで指定されます。

```
  declare (   condid_area               value(1),
              condid_attention          value(2),
              condid_condition          value(3),
              condid_conversion         value(4),
              condid_endfile            value(5),
              condid_endpage            value(6),
              condid_error              value(7),
              condid_finish             value(8),
              condid_fixedoverflow      value(9),
              condid_invalidop          value(10),
              condid_key                value(11),
              condid_name               value(12),
              condid_overflow           value(13),
              condid_record             value(14),
              condid_size               value(15),
              condid_storage            value(16),
              condid_stringrange        value(17),
              condid_stringsize         value(18),
              condid_subscriptrange     value(19),
              condid_transmit           value(20),
              condid_undefinedfile      value(21),
              condid_underflow          value(22),
              condid_zerodivide         value(23),
              condid_assertion          value(24),
          ) fixed bin(31);
```

コンテキスト以外で ONCONDID が使われている場合は、ゼロが戻されます。

## ONCOUNT

ONCOUNT は、ON ユニットが入力されたときに、依然としてハンドルされる 条件の数を指定するスケールのない FIXED BINARY 値を戻します

ONCOUNT は、ON ユニットまたは ON ユニットから動的に派生したものの中にあります。 (345 ページの『複数条件』 を参照してください。)

コンテキスト以外で ONCOUNT が使われている場合は、ゼロが戻されます。

BIFPREC コンパイラー・オプションによって、戻される結果の精度が決まります。

## ONEXPECTED

ONEXPECTED は、ASSERTION 条件を発生させた ASSERT COMPARE ステートメントの「予期される」値である不変文字ストリングを返します。 GRAPHIC タイプまたは WIDECHAR タイプの式の場合は、ヌル・ストリングが返されます。

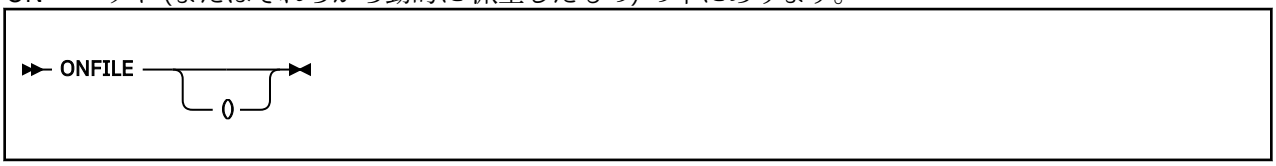
この関数は、ASSERTION 条件の ON ユニットのコンテキスト内、あるいは ASSERTION 条件の暗黙処置として発生した ERROR 条件または FINISH 条件の ON ユニットのコンテキスト内にあります。

コンテキスト以外で使用されている場合は、ヌル・ストリングが返されます。

## ONFILE

ONFILE は、値が入力条件または出力条件が発生するファイル名である文字ストリングを戻します。

名前が DBCS 名の場合は、ONFILE は混合文字ストリングとして戻されます。 また、ONFILE は、入力条件または出力条件か、入力条件または出力条件の暗黙処置として発生 した ERROR 条件または FINISH 条件の ON ユニット (またはそれらから動的に 派生したもの) の中にあります。

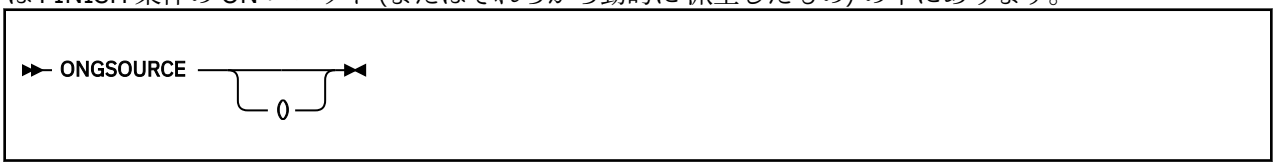



コンテキスト以外で ONFILE が使われている場合は、ヌル・ストリングが戻されます。

## ONGSOURCE

ONGSOURCE は、CONVERSION 条件が発生する原因となる DBCS 文字を含む漢字ストリングを戻します。

また、この関数は、CONVERSION 条件か、CONVERSION 条件の暗黙処置として発生 した ERROR 条件または FINISH 条件の ON ユニット (またはそれらから動的に 派生したもの) の中にあります。




ONGSOURCE 組み込み関数がコンテキスト以外で使われる場合は、 ヌル GRAPHIC ストリングが戻されます。

## ONGSOURCE 疑似変数

ONGSOURCE 疑似変数は、ONGSOURCE 組み込み関数の現行値を設定します。

疑似変数に割り当てられたエレメントの値は、グラフィックに変換されます。 再変換されるときには、このストリングが使われます。

疑似変数は、コンテキスト以外では使用してはいけません。

## ONHBOUND

ONHBOUND は、SUBSCRIPTRANGE が発生した配列のじょう上限を示す REAL FIXED BIN(63) 値を返します。

ONHBOUND がコンテキスト外で使用された場合は、ゼロが戻されます。

以下のコードが実行されると、ONHBOUND は 4 を返します。

```
          dcl a(3,2:4) fixed bin(31) init( (*) 0 );
          dcl jx       fixed bin(31);
          dcl value    fixed bin(31);

          jx = 5;
          (subrg): value = a(1,jx);
```

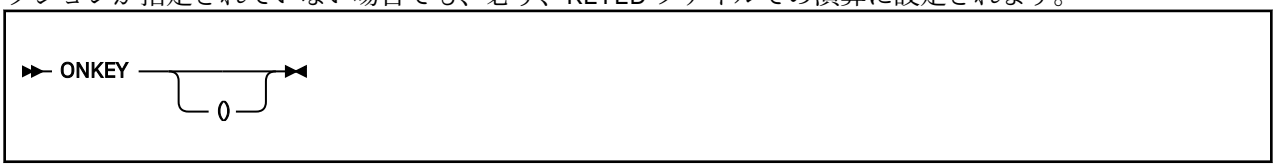
## ONKEY

ONKEY は、値が入出力条件が発生するレコードのキーである文字ストリングを戻します。

索引付きファイルの場合に、キーが GRAPHIC の場合は、ストリングは混合文字ストリングとして戻されます。 ONKEY は、次のコンテキストにあります。

- ON ユニットまたはその動的子孫
- 入出力条件 (ENDFILE 以外)
- 入出力条件の暗黙処置として発生した ERROR 条件または FINISH 条件

ONKEY は、条件が発生するステートメントで KEY オプション、KEYTO オプショ ン、または KEYFROM オプションが指定されていない場合でも、必ず、KEYED ファイルでの演算に設定されます。




ONKEY の指定結果を次に示します。

- 入出力条件 (ENDFILE 以外) の場合、または、ERROR 条件か FINISH 条件の暗黙処置として発生したこれらの条件の場合は、結果はエラーを発生する入出力ステートメントで記録されたキーの値です。
- 相対データ・セットの場合は、結果は相対レコード数の文字ストリング表 示です。 キーが間違えて指定された場合は、結果はソース・キーの最後の 8 文字です。 ソース・キーが 8 文字より小さい場合は、8 文字になるように右側がブランクで埋まります。 キーが正しく指定された場合は、文字ストリングは、必要ならば、左側がブランクで埋められた文字フォーマットの 相対レコード数で構成されます。
- 更新済みレコードのキーが入力レコードのキーと異なるときに、索引付きデータ・セットに 更新済みレコードを書き込む REWRITE ステートメントの場合は、結果は入力レコードの組み込み キーの値です。

コンテキスト以外で ONKEY が使われている場合は、ヌル・ストリングが戻されます。

## ONLBOUND

ONLBOUND は、SUBSCRIPTRANGE が発生した配列の下限を示す REAL FIXED BIN(63) 値を返します。

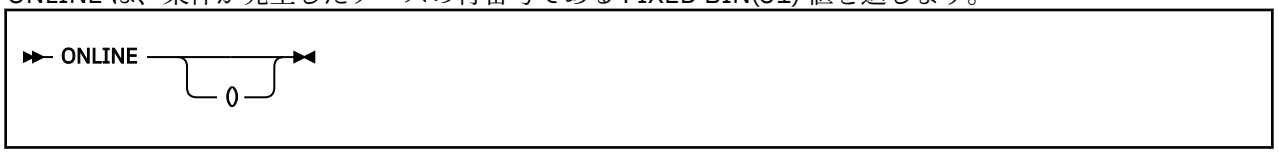
ONLBOUND がコンテキスト外で使用された場合は、ゼロが戻されます。

以下のコードが実行されると、ONLBOUND は 2 を返します。

```
          dcl a(3,2:4) fixed bin(31) init( (*) 0 );
          dcl jx       fixed bin(31);
          dcl value    fixed bin(31);

          jx = 5;
          (subrg): value = a(1,jx);
```

## ONLINE

ONLINE は、条件が発生したソースの行番号である FIXED BIN(31) 値を返します。




ソース・プログラムは、GONUMBER オプションを指定してコンパイルされている必要があり、Windows では /debug オプションでリンクされている必要もあります。

コンテキスト以外で ONLINE が使われている場合は、ゼロの値が戻されます。

## ONLOC

ONLOC は、ONPROC の同義語です。

コンテキスト以外で ONLOC が使われている場合は、ヌル・ストリングが戻されます。

**関連情報**

497 ページの『ONPROCEDURE』
ONPROCEDURE は、条件が発生したプロシージャーの名前を返します。

## ONOFFSET

ONOFFSET は、条件が発生したユーザー・プロシージャーの始め (または BEGIN ブロック) からのオフセットである FIXED BIN(31) 値を戻します。

コンテキスト以外で ONOFFSET が使われている場合は、ゼロの値が戻されます。

## ONOPERATOR

ONOPERATOR は、ASSERTION 条件を発生させた ASSERT COMPARE ステートメントの演算子を値として持つ CHAR(2) ストリングを返します。

ONOPERATOR 組み込み関数は、ASSERT COMPARE ステートメントにより発生した ASSERTION 条件の ON ユニットのコンテキスト内、 あるいは ASSERTION 条件の暗黙処置として発生した ERROR 条件または FINISH 条件の ON ユニットのコンテキスト内にあります。

ASSERT COMPARE ステートメントが ASSERTION 条件を発生させるが、その COMPARE 節で明示的に演算子を指定していない場合、 ONOPERATOR 組み込み関数は暗黙的な演算子値 'EQ' を返します。

ONOPERATOR 組み込み関数がコンテキスト外で使用された場合は、ヌル・ストリングが返されます。

## ONPACKAGE

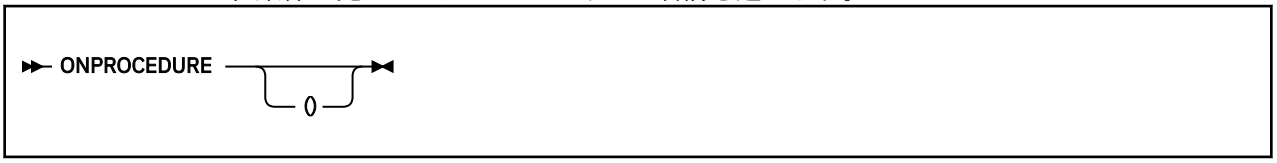
ONPACKAGE は、ASSERTION 条件を発生させた ASSERT ステートメントが呼び出されたパッケージ名を含む不変文字ストリングを返します。

この関数は、ASSERTION 条件の ON ユニットのコンテキスト内、あるいは ASSERTION 条件の暗黙処置として発生した ERROR 条件または FINISH 条件の ON ユニットのコンテキスト内にあります。

コンテキスト以外で使用されている場合は、ヌル・ストリングが返されます。

## ONPROCEDURE

ONPROCEDURE は、条件が発生したプロシージャーの名前を返します。




**省略形:** ONPROC

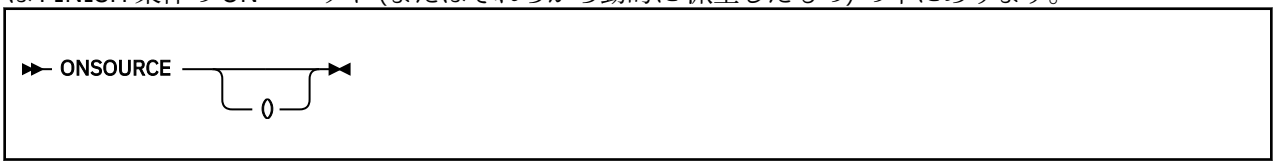
ONPROCEDURE は、CALL ステートメントまたは GOTO ステートメントに名前が表示されていても複数のラベル指定の左端の名前を常に戻します。 名前が DBCS 名の場合は、ONLOC は混合文字ストリングとして戻されます。 ONLOC は、ON ユニットまたはその動的子孫のコンテキストにあります。

コンテキスト以外で ONPROCEDURE が使われている場合は、ヌル・ストリングが戻されます。

## ONSOURCE

ONSOURCE は、値が CONVERSION 条件が発生したときに処理中のフィールドの内容である文字ストリングを戻します。

また、この関数は、CONVERSION 条件か、CONVERSION 条件の暗黙処置として発生 した ERROR 条件または FINISH 条件の ON ユニット (またはそれらから動的に 派生したもの) の中にあります。



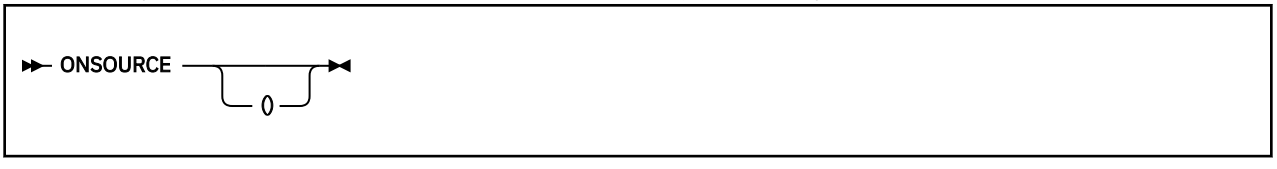

コンテキスト以外で ONSOURCE が使われている場合は、ヌル・ストリングが戻されます。

失敗した変換の中のソースが COMPLEX 値である場合、ONSOURCE() はその値の REAL 部分または IMAG 部分のみを表示します。

## ONSOURCE 疑似変数

ONSOURCE 疑似変数は、ONSOURCE 組み込み関数の現行値を設定します。

疑似変数に割り当てられたエレメントの値は、文字ストリングに変換されます。また、必要な場合には、CONVERSION 条件が発生したフィールドの長さと一致するように、右側がブランクで埋められるか切り捨てられます。 再変換されるときには、このストリングが使われます。



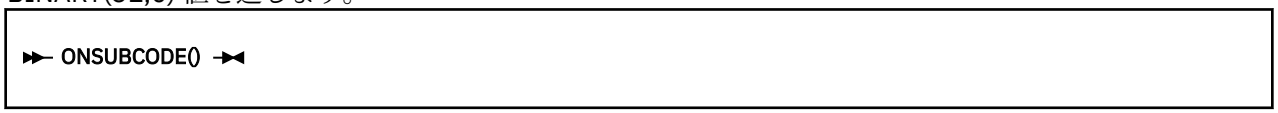
変換が再試行されるときは、疑似変数に割り当てられたストリングは、単一データ 項目として処理されます。 このため、エラーの訂正処理は、GET LIST ステートメントまたは GET DATA ステートメントの実行時に変換が発生するときに、複数のデータ項目を含むストリング を割り当てることはできません。 ストリングにブランクまたはコンマがあると、CONVERSION が再発生します。

疑似変数は、コンテキスト以外では使用してはいけません。

ONSOURCE が 2 進定数でない場合は、ONSOURCE 疑似変数を 1 に設定してはなりません。 例えば、ONSOURCE() が 'ERR' である場合、ONSOURCE() を '0'B に設定してはなりません。

## ONSUBCODE

ONSUBCODE は、発生した入出力エラー、JSON エラー、または変換エラーに関する詳細を提供する FIXED BINARY(31,0) 値を返します。



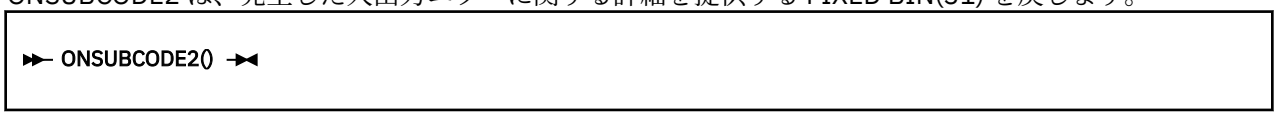

入出力エラーの場合、ONSUBCODE は、メッセージ IBM0236I および IBM0265I に関して文書化された SUBCODE1 値に対応します。 SUBCODE1 値は「メッセージおよびコード」で定義されています。

JSON 組み込み関数の場合、ERROR 条件が発生すると、ONSUBCODE は無効文字の索引を返します。

JSON または Unicode の CONVERSION 状態が発生した場合、ONSUBCODE は無効文字の索引を返します。

## ONSUBCODE2

ONSUBCODE2 は、発生した入出力エラーに関する詳細を提供する FIXED BIN(31) を戻します。




ONSUBCODE2 は、メッセージ IBM0236I および IBM0265I に関して文書化された SUBCODE2 値に対応します。 これらの SUBCODE2 値の定義は「メッセージおよびコード」に記載されています。

SUBCODE2 値は 8 つの 16 進数字 *xxxxyyyy* からなります。 *xxxx* はレジスター 15、*yyyy* は理由コードです。 戻りコードおよび理由コードは「*VSAM Macro Instructions*」に記載されています。

## ONSUBSCRIPT

ONSUBSCRIPT は、SUBSCRIPTRANGE を引き起こす無効な配列索引を示す REAL FIXED BIN(63) 値を返します。

ONSUBSCRIPT がコンテキスト以外で使用されると、ゼロが戻されます。

以下のコードが実行されると、ONSUBSCRIPT は 5 を戻します。

```
dcl a(3,2:4) fixed bin(31) init( (*) 0 );
dcl jx       fixed bin(31);
dcl value    fixed bin(31);

jx = 5;
(subrg): value = a(1,jx);
```

## ONTEXT

ONTEXT は、ASSERTION 条件を発生させた ASSERT ステートメントの TEXT 節の値を含む不変文字ストリングを返します。 ASSERT ステートメントに TEXT 節がなかった場合は、ヌル・ストリングが返されます。

この関数は、ASSERTION 条件の ON ユニットのコンテキスト内、あるいは ASSERTION 条件の暗黙処置として発生した ERROR 条件または FINISH 条件の ON ユニットのコンテキスト内にあります。

コンテキスト以外で使用されている場合は、ヌル・ストリングが返されます。

## ONUCHAR

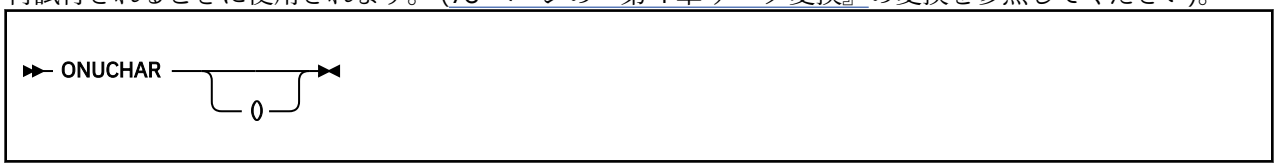
ONUCHAR は、CONVERSION 条件の原因となった UTF-8 データを含む UCHAR(1) ストリングを返します。この関数は、CONVERSION 条件の ON ユニットの中か、CONVERSION 条件の暗黙処置として発生した ERROR 条件または FINISH 条件の ON ユニット (またはそれらの ON ユニットから動的に派生したもの) の中にあります。

ONUCHAR 組み込み関数がコンテキスト以外で使われる場合は、UTF-8 ブランクが返されます。

## ONUCHAR 疑似変数

ONUCHAR 疑似変数は、ONUCHAR 組み込み関数の現行値を設定します。

疑似変数に割り当てられたエレメント値は、長さ 1 の UCHAR 値に変換されます。 この UCHAR は、変換が再試行されるときに使用されます。 (73 ページの『第 4 章 データ変換』の変換を参照してください)。




疑似変数は、コンテキスト以外では使用してはいけません。

## ONUSOURCE

ONUSOURCE は、その値が CONVERSION 条件の発生時に処理中であったフィールドの内容である UCHAR ストリングを返します。 また、この関数は、CONVERSION 条件か、CONVERSION 条件の暗黙処置として発生 した ERROR 条件または FINISH 条件の ON ユニット (またはそれらから動的に 派生したもの) の中にあります。

ONUSOURCE 組み込み関数がコンテキスト外で使用された場合は、ヌル・ストリングが返されます。

## ONUSOURCE 疑似変数

ONUSOURCE 疑似変数は、ONUSOURCE 組み込み関数の現行値を設定します。

疑似変数に割り当てられたエレメント値は、UCHAR ストリングに変換されます。 また、必要な場合には、CONVERSION 条件が発生したフィールドの長さと一致するように、右側が UCHAR ブランクで埋められるか切り捨てられます。 再変換されるときには、このストリングが使われます。 (73 ページの『第 4 章 データ変換』の変換を参照してください)。

疑似変数は、コンテキスト以外では使用してはいけません。

## ONWCHAR

ONWCHAR は、CONVERSION 条件が発生する原因となるワイド文字を 含む widechar(1) ストリングを戻します。

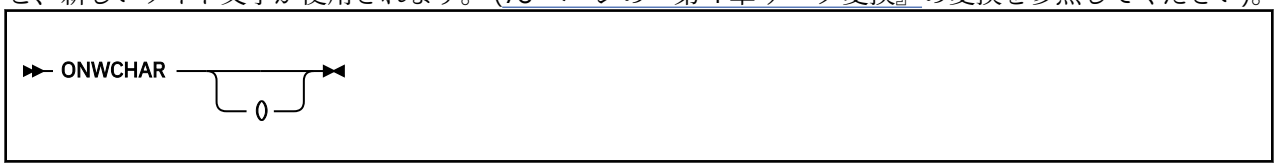
この関数は、CONVERSION 条件の ON ユニットの中か、CONVERSION 条件の暗黙処置として発生した ERROR 条件または FINISH 条件の ON ユニット (またはそれらの ON ユニットから動的に派生したもの) の中にあります。

ONWCHAR 組み込み関数がコンテキスト以外で使われる場合は、 ワイド文字ブランクが戻されます。

## ONWCHAR 疑似変数

ONWCHAR 疑似変数は、ONWCHAR 組み込み関数の現行値を設定します。

疑似変数に割り当てられたエレメント値は、長さ 1 のワイド文字値に変換されます。 変換が再試行されると、新しいワイド文字が使用されます。 (73 ページの『第 4 章 データ変換』の変換を参照してください)。

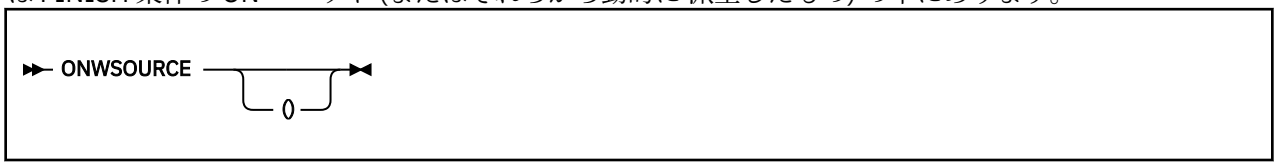



疑似変数は、コンテキスト以外では使用してはいけません。

## ONWSOURCE

ONWSOURCE は、その値が CONVERSION 条件の発生時に処理中であったフィールドの内容である WIDECHAR ストリングを返します。

また、この関数は、CONVERSION 条件か、CONVERSION 条件の暗黙処置として発生 した ERROR 条件または FINISH 条件の ON ユニット (またはそれらから動的に 派生したもの) の中にあります。




コンテキスト以外で ONWSOURCE が使われている場合は、ヌル・ストリングが戻されます。

## ONWSOURCE 疑似変数

ONWSOURCE 疑似変数は、ONWSOURCE 組み込み関数の現行値を設定します。

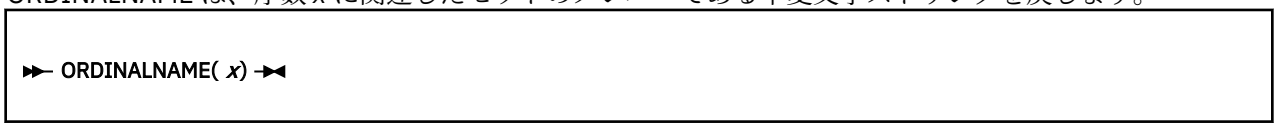
疑似変数に割り当てられたエレメントの値は、ワイド文字ストリングに変換されます。 また、必要な場合には、CONVERSION 条件が発生したフィールドの長さと一致するように、右側がワイド文字ブランクで埋められるか切り捨てられます。 再変換されるときには、このストリングが使われます。

変換が再試行されるときは、疑似変数に割り当てられたストリングは、単一データ 項目として処理されます。 このため、エラーの訂正処理は、GET LIST ステートメントまたは GET DATA ステートメントの実行時に変換が発生するときに、複数のデータ項目を含むストリング を割り当てることはできません。 ストリングにブランクまたはコンマがあると、CONVERSION が再発生します。

疑似変数は、コンテキスト以外では使用してはいけません。

## ORDINALNAME

ORDINALNAME は、序数 *x* に関連したセットのメンバーである不変文字ストリングを戻します。

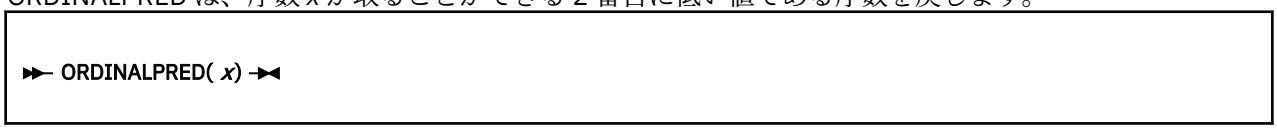



**x**
参照。 この関数には、序数タイプを保持している必要があります。

ORDINAL を計算式内で使用してはなりません。また、文字へ変換してもな りません。しかし、ORDINALNAME によって、ORDINAL に表示可能な値を取得する ことが可能なので、デバッグにおいて非常に役立ちます。

## ORDINALPRED

ORDINALPRED は、序数 *x* が取ることができる 2 番目に低い値である序数を戻します。




**x**
参照。 この関数には、序数タイプを保持している必要があります。

戻される序数には、序数 *x* と同じタイプを保持している必要があります。

## ORDINALSUCC

ORDINALSUCC は、序数 *x* が取ることができる 2 番目に高い値である序数を戻します。

**x**
参照。 この関数には、序数タイプを保持している必要があります。

戻される序数には、序数 *x* と同じタイプを保持している必要があります。

## PACKAGENAME

PACKAGENAME は、呼び出されたパッケージ名を含む不変文字ストリングを戻します。

現行のコンパイル単位にパッケージがない場合は、PACKAGENAME は最外部のプロシージャーの 名前を戻します。

# PAGENO

PAGENO は、ファイル *x* に関連する現行ページ番号である、スケールなし REAL FIXED BIN(31) 値を返します。

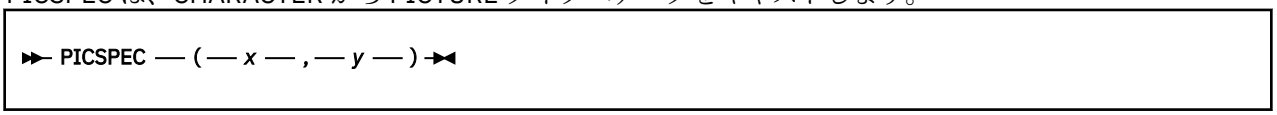
**x**
ファイル参照。 ファイルは、オープン状態で、PRINT 属性を持っている必要があります。

ファイルが PRINT ファイルでない場合は、ERROR 条件が発生します。

BIFPREC コンパイラー・オプションによって、戻される結果の精度が決まります。

# PICSPEC

PICSPEC は、CHARACTER から PICTURE タイプへデータをキャストします。




**x**
式。

**y**
ピクチャー指定。

式 *x* は、コンパイル時に既知となる長さの CHARACTER NONVARYING でなければなりません。

*y* は、最初の引数と同じ長さの外部表記を持つ、有効な PICTURE を指定する文字リテラルでなければなりません。

結果は、2 番目の引数によって指定された PICTURE タイプになります。

EDIT 組み込み関数とは異なり、変換は行われず、最初の引数がピクチャーに有効なデータを保持しているかを調べるチェックも行われません。

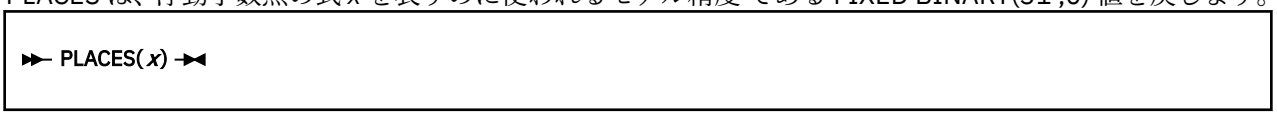
UNSPEC 組み込み関数のように、データの「タイプ」だけが変更されます。

そのため、例えば PICSPEC(x,'(5)9') では、*x* は CHAR(5) でなければなりませんが (ピクチャー指定 '(5)9' の長さは 4 文字であるものの、外部表記では 5 つの文字が必要なため)、*x* は実際に 5 つの数字を含んでいるかどうかチェックされることはありません。

N = N + PICSPEC(X,'(5)9') という記述によって *x* が CHAR から PIC'(5)9' に変換されることはありませんが (この変換ではライブラリー呼び出しが必要)、*x* の内容は、PIC'(5)9' であると宣言されたかのように扱われます。

# PLACES

PLACES は、浮動小数点の式 *x* を表すのに使われるモデル精度 である FIXED BINARY(31 ,0) 値を戻します。




**x**
式。 *x* は、REAL FLOAT として宣言しなければなりません。

PLACES(x) は定数で、制限付き式に使用できます。

### 例 (Intel の値)

```
places(x) = 24        for x float bin(p), p <= 21
places(x) = 53        for x float bin(p), 21 < p <= 53
places(x) = 64        for x float bin(p), 53 < p
```

```
places(x) = 24        for x float dec(p), p <= 6
places(x) = 53        for x float dec(p), 6 < p <= 16
places(x) = 64        for x float dec(p), 16 < p
```

**例 (AIX の値)**

```
places(x) = 024        for x float bin(p), p <= 21
places(x) = 053        for x float bin(p), 21 < p <= 53
places(x) = 106        for x float bin(p), 53 < p

places(x) = 024        for x float dec(p), p <= 6
places(x) = 053        for x float dec(p), 6 < p <= 16
places(x) = 106        for x float dec(p), 16 < p
```

**例 (z/OS の 16 進値)**

```
places(x) = 6         for x float bin(p), p <= 21
places(x) = 14        for x float bin(p), 21 < p <= 53
places(x) = 28        for x float bin(p), 53 < p

places(x) = 6         for x float dec(p), p <= 6
places(x) = 14        for x float dec(p), 6 < p <= 16
places(x) = 28        for x float dec(p), 16 < p
```

**例 (z/OS の IEEE 2 進浮動小数点数値)**

```
places(x) = 24        for x float bin(p), p <= 21
places(x) = 53        for x float bin(p), 21 < p <= 53
places(x) = 113       for x float bin(p), 53 < p

places(x) = 24        for x float dec(p), p <= 6
places(x) = 53        for x float dec(p), 6 < p <= 16
places(x) = 113       for x float dec(p), 16 < p
```

**例 (z/OS の IEEE 10 進浮動小数点数値)**

```
places(x) = 7         for x float dec(p), p <= 7
places(x) = 16        for x float dec(p), 7 < p <= 16
places(x) = 34        for x float dec(p), 16 < p
```

# PLIASCII

PLIASCII は、位置 *y* にある *z* バイトの EBCDIC 値を位置 *x* の ASCII 値に変換します。

位置 x および y にあるストレージは、それらが同じ位置を指定していない限り、重なり合ってはなりません。

**x および y**
タイプ POINTER または OFFSET を持つ式。 タイプが OFFSET の場合は、式は、AREA 属性によって宣言される OFFSET 変数でなければなりません。

**z**
式。 計算タイプでなければならず、タイプ *size_t* に変換されます。[1]

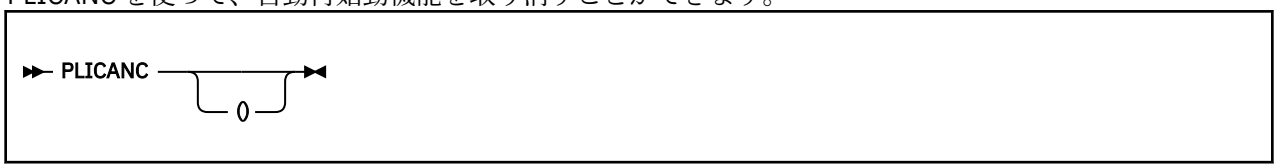
# PLIATTN

PLIATTN は、コード内のそのポイントで ATTENTION 条件を引き起こします。 これにより、コンパイラーがアテンション・ブレークポイントを挿入する場所に対する明示制御が提供されます。

INTERRUPT オプションには、このサブルーチンの呼び出しのために生成されたコードに対する効果はありません。

## PLICANC

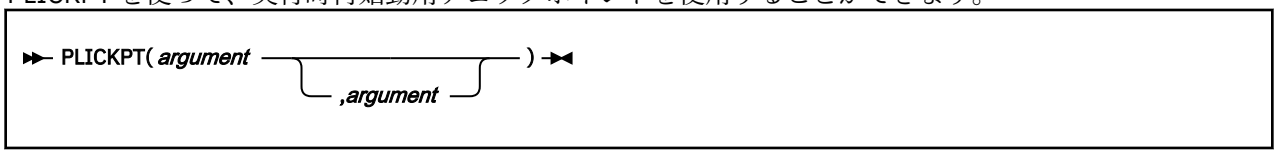
PLICANC を使って、自動再始動機能を取り消すことができます。




PLICANC の使用に関する情報については、「プログラミング・ガイド」を参照してください。

## PLICKPT

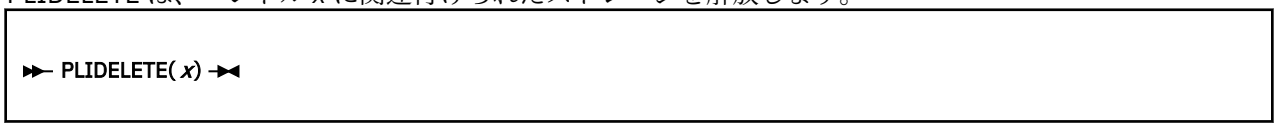
PLICKPT を使って、実行時再始動用チェックポイントを使用することができます。




PLICKPT の使用に関する情報については、「プログラミング・ガイド」を参照してください。

## PLIDELETE

PLIDELETE は、ハンドル *x* に関連付けられたストレージを解放します。



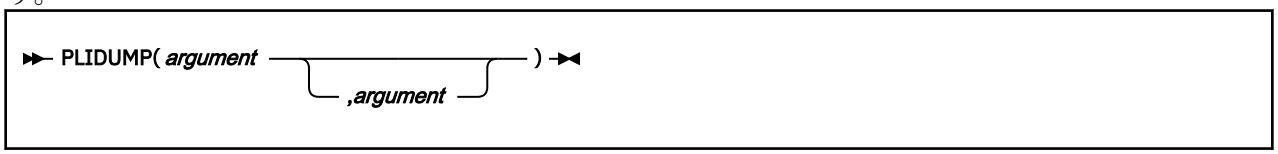

**x**
ハンドル式。

PLIDELETE(x) は、ハンドルに関連したストレージを解放するには最適な方法です。 このストレージは、通常、NEW タイプ付き関数によって取得されます。

CALL PLIDELETE(x) は、CALL PLIFREE(PTRVALUE(x)) と同等です。

## PLIDUMP

PLIDUMP により、プログラムで使われるストレージの選択部分の定様式ダンプを獲得することができます。




PLIDUMP の使用に関する詳細は、「プログラミング・ガイド」を参照してください。

## PLIEBCDIC

PLIEBCDIC は、位置 *y* にある *z* バイトの ASCII 値を位置 *x* の EBCDIC 値に変換します。

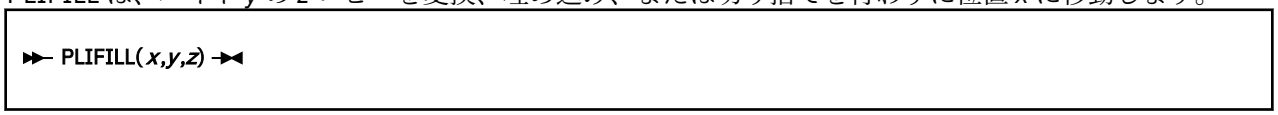
位置 x および y にあるストレージは、それらが同じ位置を指定していない限り、重なり合ってはなりません。

**x および y**
タイプ POINTER または OFFSET を持つ式。 タイプが OFFSET の場合は、式は、AREA 属性によって宣言される OFFSET 変数でなければなりません。

**z**
式。 計算タイプでなければならず、タイプ *size_t* に変換されます。[1]

## PLIFILL

PLIFILL は、バイト *y* の *z* コピーを変換、埋め込み、または切り捨てを行わずに位置 *x* に移動します。




**x**
式。 *x* は、POINTER または OFFSET として宣言されなければなりません。 これが OFFSET である場合、*x* は AREA 属性を使用して宣言する必要があります。

**y**
CHARACTER(1) NONVARYING として宣言されなければなりません。

**z**
式。 タイプ *size_t* [1] に変換されます。

**例**

```
dcl 1 Str1,
      2 B  fixed bin(31),
      2 C  pointer,
      2 * union,
        3 D  char(4),
        3 E  fixed bin(31),
        3 *,
          4 * char(3),
          4 F fixed bin(8) unsigned,
      2 * char(0)
           initial call plifill( addr(Str1), '00'x, stg(Str1) );
```

## PLIFREE

PLIFREE は、ALLOCATE 組み込み関数を使用して割り振られたポインター *p* に関連付けられたヒープ・ストレージを解放します。

**p**
ロケーター式。

PLIFREE は、ALLOCATE (ALLOC) の反対です。

## PLIMOVE

PLIMOVE は、位置 *y* の *z* ストレージの単位 (バイト) を変換、 埋め込み、または切り捨てを行わずに位置 *x* に移動します。

PLIOVER 組み込みサブルーチンと異なり、位置 *x* および *y* での ストレージは固有であると仮定されます。ストレージがオーバーラップする場合は、予期しない結果が発生することがあります。

**x および y**
POINTER または OFFSET として宣言された式。 タイプが OFFSET の場合は、*x* または *y* を AREA 属性 に よって宣言する必要があります。

**z**
式。 計算タイプでなければならず、タイプ *size_t* に変換されます。[1]

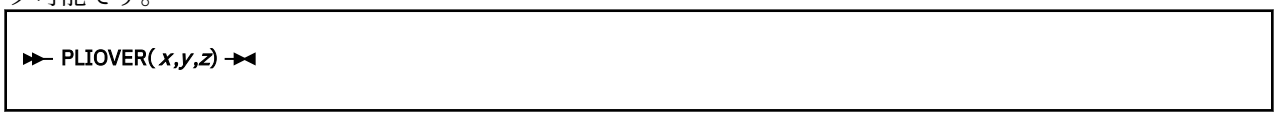
**例**

```
dcl 1 Str1,
      2 B  fixed bin(31),
      2 C  pointer,
      2 * union,
        3 D  char(4),
        3 E  fixed bin(31),
        3 *,
          4 * char(3),
          4 F fixed bin(8) unsigned,
      2 * char(0);
dcl 1 Template nonasgn static,
      2 * fixed bin(31) init(200),
      2 * pointer init(sysnull()),
      2 * char(4) init(''),
      2 * char(0);

call plimove(addr(Str1), addr(Template), stg(Str1));
```

**関連情報**
506 ページの『PLIOVER』
PLIOVER は、位置 *y* の *z* ストレージの単位 (バイト) を変換、 埋め込み、または切り捨てを行わずに位置 *x* に移動します。 PLIMOVE 組み込みサブルーチンと異なり、位置 *x* および *y* での ストレージはオーバーラップ可能です。

## PLIOVER

PLIOVER は、位置 *y* の *z* ストレージの単位 (バイト) を変換、 埋め込み、または切り捨てを行わずに位置 *x* に移動します。 PLIMOVE 組み込みサブルーチンと異なり、位置 *x* および *y* での ストレージはオーバーラップ可能です。



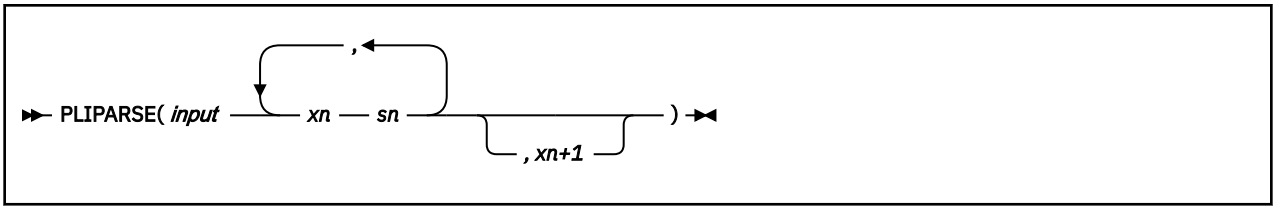

**x および y**
POINTER または OFFSET として宣言された式。 タイプが OFFSET の場合は、*x* または *y* を AREA 属性 に よって宣言する必要があります。

**z**
式。 計算タイプでなければならず、タイプ *size_t* に変換されます。[1]

**関連情報**
505 ページの『PLIMOVE』
PLIMOVE は、位置 *y* の *z* ストレージの単位 (バイト) を変換、 埋め込み、または切り捨てを行わずに位置 *x* に移動します。

## PLIPARSE

PLIPARSE は、文字ストリングをサブストリングに構文解析します。




引数は 3 個以上 64 個以下でなければなりません。

最初の引数は、解析される入力ストリングです (CHARACTER タイプの任意の式にすることができます)。

その入力ストリングの後の引数は、偶数個のペアで構成され、各ペアは以下のもので構成されます。

- 各ペアがターゲット参照 (または *) と区切り文字で構成される、偶数個のペア
- 参照 (または *) でなければならないオプションの最後のターゲット引数

最初の区切り文字の後に最初のターゲット引数が見つからないと、残りの入力が割り当てられ、残りのターゲット引数には引用符が割り当てられます。

* 以外のターゲット引数は、CHARACTER タイプで、ASSIGNABLE でなければなりません。

区切り文字はすべて CHARACTER タイプでなければなりません。

**Example 1**

次の場合、

```
    dcl x1    char(16) varying;
    dcl x2    char(16) varying;
    dcl x3    char(16) varying;
    dcl x4    char(16) varying;

    dcl s1    char value('KEY:');
    dcl s2    char value('--');
    dcl s3    char value('-');

    input = '31415KEY:0123--45678-9';

    call pliparse( input, x1, s1, x2, s2, x3, s3, x4  );
```

ターゲット引数は、以下の値になります。

```
    x1 = '31415';
    x2 = '0123';
    x3 = '45678';
    x4 = '9';
```

**例 2**

この例では、「input」は「--」がない上記の例と異なります。

次の場合、

```
    dcl x1    char(16) varying;
    dcl x2    char(16) varying;
    dcl x3    char(16) varying;
    dcl x4    char(16) varying;

    dcl s1    char value('KEY:');
    dcl s2    char value('--');
    dcl s3    char value('-');

    input = '31415KEY:0123-45678-9';

    call pliparse( input, x1, s1, x2, s2, x3, s3, x4  );
```

ターゲット引数は、以下の値になります。

```
    x1 = '31415';
    x2 = '0123-45678-9';
    x3 = '';
    x4 = '';
```

**例 3**

次の場合、

```
dcl x1    char(16) varying;
dcl x2    char(16) varying;
dcl x3    char(16) varying;
dcl x4    char(16) varying;

input = ' Alex  Bruno';

call pliparse( input, x1, ' ', x2, ' ', x3, ' ', x4  );
```

ターゲット引数は、以下の値になります。

```
x1 = '';
x2 = 'Alex';
x3 = '';
x4 = 'Bruno';
```

**例 4**

次の場合、

```
dcl x1    char(16) varying;
dcl x2    char(16) varying;
dcl x3    char(16) varying;
dcl x4    char(16) varying;
input = collapse( ' Alex  Bruno', ' ' );

call pliparse( input, x1, ' ', x2, ' ', x3, ' ', x4  );
```

ターゲット引数は、以下の値になります。

```
x1 = 'Alex';
x2 = 'Bruno';
x3 = '';
x4 = '';
```

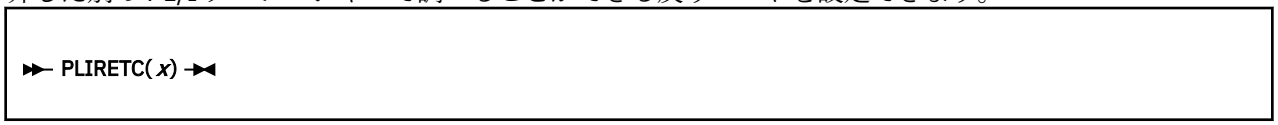
## PLIREST

PLIREST を使用すると、プログラム実行を再開できます。

PLIREST の使用に関する情報については、「プログラミング・ガイド」を参照してください。

## PLIRETC

PLIRETC を使用すると、この PL/I プログラムを呼び出したプログラム、 または PLIRETV 組み込み関数を介した別の PL/I プロシージャーで調べることができる戻りコードを設定できます。




**x**
FIXED BINARY(31 ,0) 戻りコードを指定する式。

## PLIRETV

PLIRETV は、PL/I の戻りコードである FIXED BINARY(31 ,0) 値を戻します。

PL/I の戻りコードの値は、CALL PLIRETC ステートメントで指定された最新の値です。

## PLISAXA

PLISAXA は、プログラムのバッファー内にある XML 文書の SAX スタイル構文解析を実行します。

**e**
イベント構造体。

**p**
構文解析イベントに戻されるポインター値または 「トークン」。

**x**
入力 XML が入っているバッファーのアドレス。

**n**
そのバッファーにあるデータのバイト数。 計算タイプでなければならず、タイプ *size_t* に変換されます。[1]

**c**
その XML のコード・ページの名称を指定する数値表現。

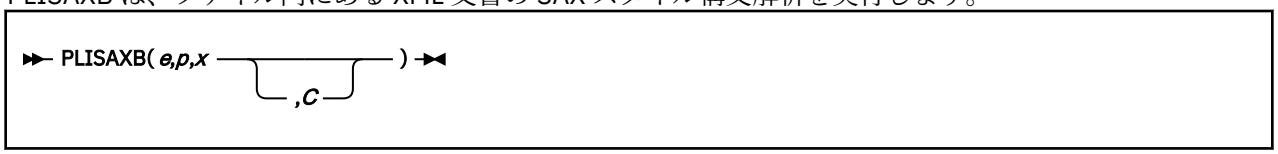
XML が CHARACTER VARYING ストリングまたは WIDECHAR VARYING ストリング に含まれている場合は、ADDRDATA 組み込み関数を使用して、 最初のデータ・バイトのアドレスを取得する必要があります。

また、XML が WIDECHAR ストリングに含まれている場合、 バイト数の値は LENGTH 組み込み関数によって戻される値の 2 倍になることに 注意してください。

詳しくは、「プログラミング・ガイド」を参照してください。

## PLISAXB

PLISAXB は、ファイル内にある XML 文書の SAX スタイル構文解析を実行します。




**e**
イベント構造体

**p**
構文解析イベントに戻されるポインター値または 「トークン」

**x**
入力ファイルを指定する文字ストリング式

**c**
その XML のコード・ページの名称を指定する数値表現

詳しくは、「プログラミング・ガイド」を参照してください。

## PLISAXC

PLISAXC は、プログラム内の 1 つ以上のバッファーにある XML 文書の SAX スタイル構文解析を実行します。

**e**
イベント構造体。

**p**
構文解析イベントに戻されるポインター値または「トークン」。

**x**
XML 文書があるバッファーのアドレス。

**n**
そのバッファーにあるデータのバイト数。 計算タイプでなければならず、タイプ *size_t* に変換されます。[1]

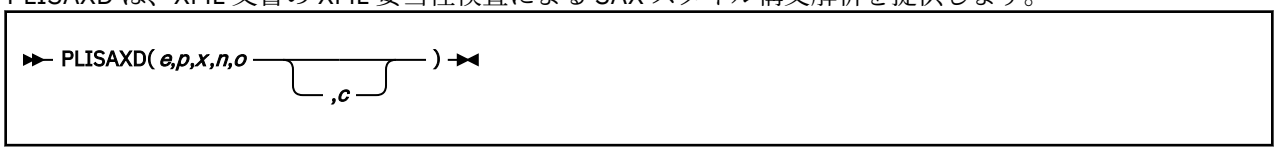
**c**
その XML 文書のコード・ページを指定する数値表現。

PLISAXC は、z/OS XML System Services パーサーを使用しており、z/OS でのみサポートされます。

詳しくは、「*Enterprise PL/I for z/OS* プログラミング・ガイド」を参照してください。

## PLISAXD

PLISAXD は、XML 文書の XML 妥当性検査による SAX スタイル構文解析を提供します。




**e**
イベント構造体。

**p**
構文解析イベントに戻されるポインター値または「トークン」。

**x**
XML 文書があるバッファーのアドレス。

**n**
そのバッファーにあるデータのバイト数。 計算タイプでなければならず、タイプ *size_t* に変換されます。[1]

**o**
Optimized Schema Representation (OSR) を含むバッファーのアドレス。

**c**
その XML 文書のコード・ページを指定する数値表現。

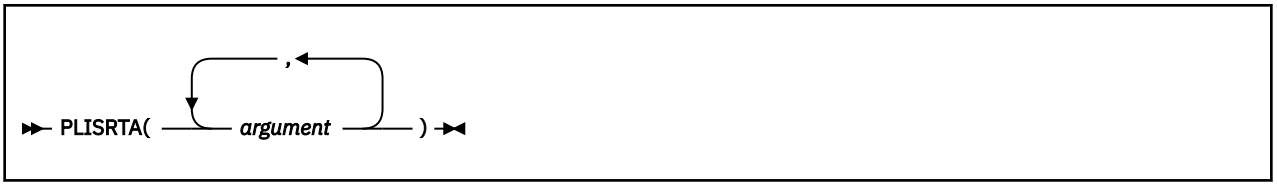
PLISAXD は、z/OS XML System Services パーサーを使用しており、z/OS でのみサポートされます。

詳しくは、「*Enterprise PL/I for z/OS* プログラミング・ガイド」の章『*PLISAXD XML* パーサーの使用』 を参照してください。

**注 :** OSR はスキーマのプリプロセス・バージョンです。 OSR について詳しくは、「*XML System Services User's Guide and Reference*」を参照してください。

## PLISRTA

PLISRTA は、入力ファイルをソートして、ソートされた出力ファイルを生成します。




詳しくは、プログラミング・ガイド を参照してください。

## PLISRTB

PLISRTB は、E15 PL/I 出口プロシージャーによって提供される入力レコードをソートし、 ソートされた出力ファイルを生成します。

詳しくは、プログラミング・ガイド を参照してください。

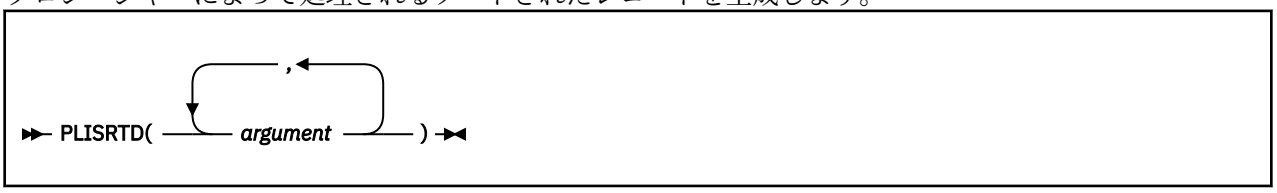
## PLISRTC

PLISRTC は、入力ファイルをソートして、E35 PL/I 出口プロシージャー によって処理されるソートされたレコードを生成します。

詳細については、「*Enterprise PL/I for z/OS* プログラミング・ガイド」を参照してください。

## PLISRTD

PLISRTD は、E15 PL/I 出口プロシージャーによって提供される入力レコードをソートして、E35 PL/I 出口プロシージャーによって処理されるソートされたレコードを生成します。




詳しくは、プログラミング・ガイド を参照してください。

## PLISTCK

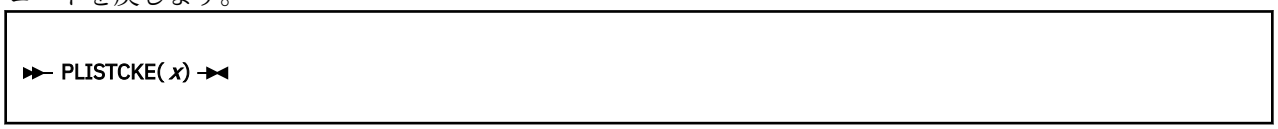
PLISTCK は、対応するストア・クロック・ハードウェア命令を生成し、その命令によって設定される条件コードを戻します。

**x**

REAL UNSIGNED FIXED BIN(64) の参照。 STCK 命令によって設定されます。 STCK 命令の詳細については、「*Principles of Operations*」を参照してください。

## PLISTCKE

PLISTCKE は、対応するストア・クロック・ハードウェア命令を生成し、その命令によって設定される条件コードを戻します。

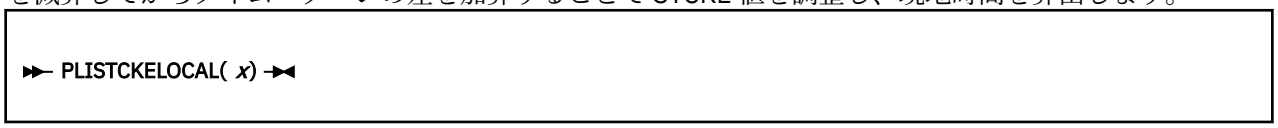



**x**

CHAR(16) NONVARYING の参照。 STCKE 命令によって設定されます。 STCKE 命令の詳細については、「*Principles of Operations*」を参照してください。

## PLISTCKELOCAL

PLISTCKELOCAL は、対応するストア・クロック・ハードウェア命令を生成し、STCKE 値からうるう秒の数を減算してからタイム・ゾーンの差を加算することで STCKE 値を調整し、現地時間を算出します。



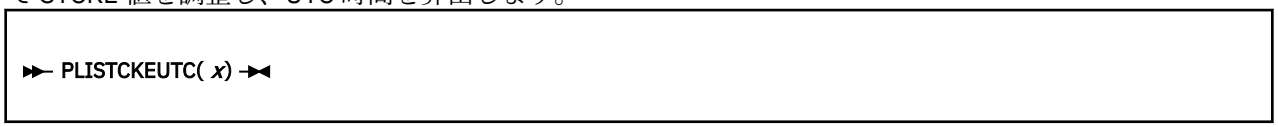

**x**

CHAR(16) NONVARYING の参照。

STCKE 命令によって設定されてから、調整されます。 STCKE 命令の詳細については、「*Principles of Operations*」を参照してください。

## PLISTCKEUTC

PLISTCKEUTC は、対応するストア・クロック・ハードウェア命令を生成し、うるう秒の数を減算することで STCKE 値を調整し、UTC 時間を算出します。



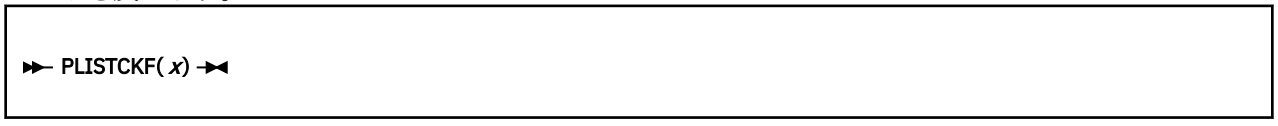

**x**

CHAR(16) NONVARYING の参照。

STCKE 命令によって設定されてから、調整されます。 STCKE 命令の詳細については、「*Principles of Operations*」を参照してください。

## PLISTCKF

PLISTCKF は、対応するストア・クロック・ハードウェア命令を生成し、その命令によって設定される条件コードを戻します。

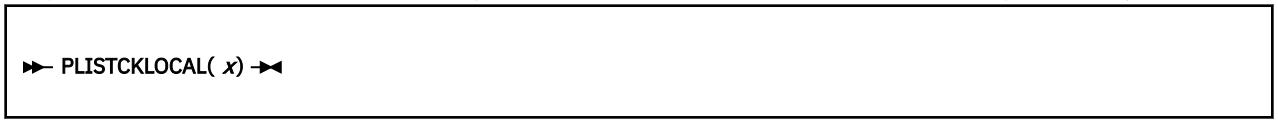



**x**

REAL UNSIGNED FIXED BIN(64) の参照。 STCKF 命令によって設定されます。 STCKF 命令の詳細については、「*Principles of Operations*」を参照してください。

## PLISTCKLOCAL

PLISTCKLOCAL は、対応するストア・クロック・ハードウェア命令を生成し、STCK 値からうるう秒の数を減算してからタイム・ゾーンの差を加算することで STCK 値を調整し、現地時間を算出します。



**x**

REAL UNSIGNED FIXED BINARY(64) の参照。

STCK 命令によって設定されてから、調整されます。 STCK 命令の詳細については、「*Principles of Operations*」を参照してください。

## PLISTCKP

PLISTCKP は、対応する Perform Timing Facility Function (PTFF) ハードウェア命令を生成し、その命令によって設定された条件コードを返します。

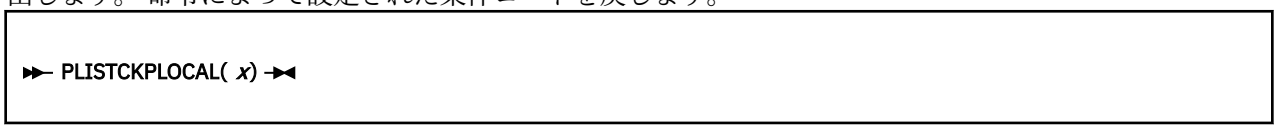
**x**

REAL UNSIGNED FIXED BIN(64) の参照。

これは PTFF 命令によって設定されます。 PTFF 命令の詳細については、「*Principles of Operations*」マニュアルを参照してください。

## PLISTCKPLOCAL

PLISTCKPLOCAL は、対応する Perform Timing Facility Function (PTFF) ハードウェア命令を生成し、PTFF 値からうるう秒の数を減算してからタイム・ゾーンの差を加算することで PTFF 値を調整し、現地時間を算出します。 命令によって設定された条件コードを戻します。



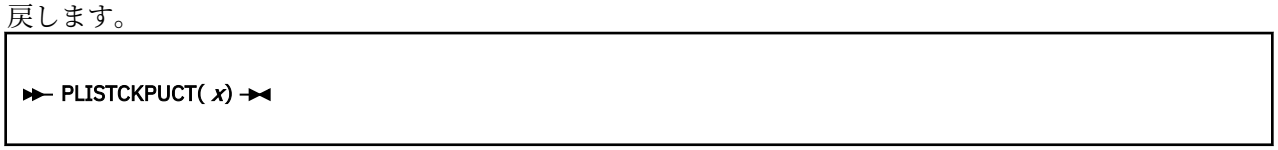

**x**

REAL UNSIGNED FIXED BIN(64) の参照。

これは PTFF 命令によって設定されてから、調整されます。 PTFF 命令の詳細については、「*Principles of Operations*」マニュアルを参照してください。

## PLISTCKPUCT

PLISTCKPUTC は、対応する Perform Timing Facility Function (PTFF) ハードウェア命令を生成し、うるう秒の数を減算することで PTFF 値を調整し、UTC 時刻を算出します。 命令によって設定された条件コードを戻します。




**x**

REAL UNSIGNED FIXED BIN(64) の参照。

これは PTFF 命令によって設定されてから、調整されます。 PTFF 命令の詳細については、「*Principles of Operations*」マニュアルを参照してください。

## PLISTCKUTC

PLISTCKUTC は、対応するストア・クロック・ハードウェア命令を生成し、うるう秒の数を減算することで STCK 値を調整し、UTC 時間を算出します。

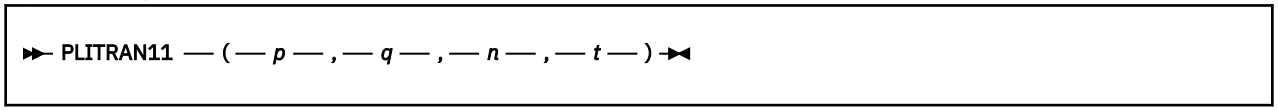
**x**

REAL UNSIGNED FIXED BINARY(64) の参照。

STCK 命令によって設定されてから、調整されます。 STCK 命令の詳細については、「*Principles of Operations*」を参照してください。

## PLITRAN11

PLITRAN11 は、ソース・バッファーの 1 バイト・データをターゲット・バッファーの 1 バイト・データに変換します。




**p**
ターゲット・バッファーのアドレス。

**q**
ソース・バッファーのアドレス。

**n**
ソース・バッファーの長さ。

**t**
256 バイト変換テーブルのアドレス。

バッファー長は、負以外で、計算タイプでなければなりません。 バッファー長はタイプ *size_t* に変換されます。[1]

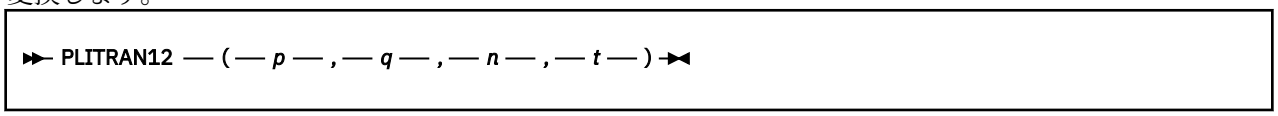
ターゲット・バッファーの大きさは、ソース・バッファー以上でなければなりません。

変換テーブルは、ダブルワード境界に位置合わせされなければなりません。 この位置合わせを強制する最も簡単な方法は、属性 ALIGNED(8) を表の宣言に追加することです。

z/OS では、PLITRAN11 は CU11 命令を使用してインライン・コードによって実装されます。

## PLITRAN12

PLITRAN12 は、ソース・バッファーの 1 バイト・データをターゲット・バッファーの 2 バイト・データに変換します。




**p**
ターゲット・バッファーのアドレス。

**q**
ソース・バッファーのアドレス。

**n**
ソース・バッファーの長さ。 バッファー長は、負以外で、計算タイプでなければなりません。 バッファー長はタイプ *size_t* に変換されます。[1]

**t**
512 バイト変換テーブルのアドレス。

ターゲット・バッファーの大きさは、ソース・バッファーの 2 倍以上でなければなりません。

変換テーブルは、ダブルワード境界に位置合わせされなければなりません。 この位置合わせを強制する最も簡単な方法は、属性 ALIGNED(8) を表の宣言に追加することです。

z/OS では、PLITRAN12 は CU12 命令を使用してインライン・コードによって実装されます。

例:

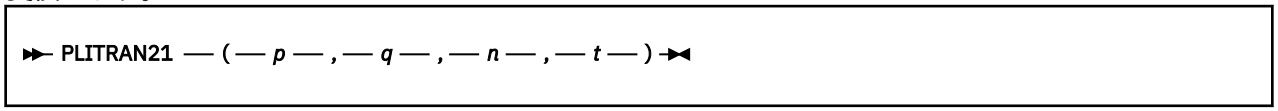
この表を使用して、バッファーを小文字の 16 進数に素早く変換できます。

```
           dcl
             1 lowerhex(16) static nonasgn aligned(8)
                            char(32) init (
```

```
                        /*   0 1 2 3 4 5 6 7 8 9 a b c d e f   */
                           '000102030405060708090a0b0c0d0e0f',
                           '101112131415161718191a1b1c1d1e1f',
                           '202122232425262728292a2b2c2d2e2f',
                           '303132333435363738393a3b3c3d3e3f',
                           '404142434445464748494a4b4c4d4e4f',
                           '505152535455565758595a5b5c5d5e5f',
                           '606162636465666768696a6b6c6d6e6f',
                           '707172737475767778797a7b7c7d7e7f',
                           '808182838485868788898a8b8c8d8e8f',
                           '909192939495969798999a9b9c9d9e9f',
                           'a0a1a2a3a4a5a6a7a8a9aaabacadaeaf',
                           'b0b1b2b3b4b5b6b7b8b9babbbcbdbebf',
                           'c0c1c2c3c4c5c6c7c8c9cacbcccdcecf',
                           'd0d1d2d3d4d5d6d7d8d9dadbdcdddedf',
                           'e0e1e2e3e4e5e6e7e8e9eaebecedeeef',
                           'f0f1f2f3f4f5f6f7f8f9fafbfcfdfeff'  );

          call plitran12( p, q, n, addr(lowerhex) );
```

## PLITRAN21

PLITRAN21 は、ソース・バッファーの 2 バイト・データをターゲット・バッファーの 1 バイト・データに変換します。




**p**
ターゲット・バッファーのアドレス。

**q**
ソース・バッファーのアドレス。

**n**
ソース・バッファーの長さ。 バッファー長は、負以外で、計算タイプでなければなりません。 バッファー長はタイプ *size_t* に変換されます。[1]

**t**
64 K バイト変換テーブルのアドレス。

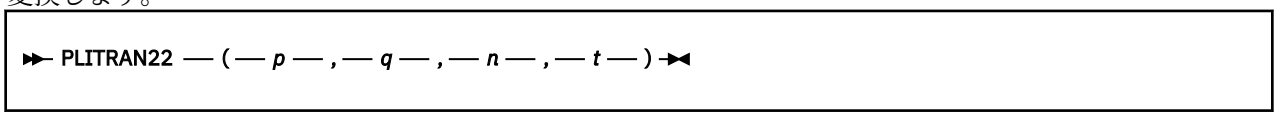
ターゲット・バッファーの大きさは、ソース・バッファーの半分以上でなければなりません。

変換テーブルは、ダブルワード境界に位置合わせされなければなりません。 この位置合わせを強制する最も簡単な方法は、属性 ALIGNED(8) を表の宣言に追加することです。

z/OS では、PLITRAN21 は CU21 命令を使用してインライン・コードによって実装されます。

## PLITRAN22

PLITRAN22 は、ソース・バッファーの 2 バイト・データをターゲット・バッファーの 2 バイト・データに変換します。




**p**
ターゲット・バッファーのアドレス。

**q**
ソース・バッファーのアドレス。

**n**
ソース・バッファーの長さ。 バッファー長は、負以外で、計算タイプでなければなりません。 バッファー長はタイプ *size_t* に変換されます。[1]

**t**
128 K バイト変換テーブルのアドレス。

ターゲット・バッファーの大きさは、ソース・バッファー以上でなければなりません。

変換テーブルは、ダブルワード境界に位置合わせされなければなりません。 この位置合わせを強制する最も簡単な方法は、属性 ALIGNED(8) を表の宣言に追加することです。

z/OS では、PLITRAN22 は CU22 命令を使用してインライン・コードによって実装されます。

## POINTER

POINTER は、オフセット参照 *x* で指定された世代を識別するポインター値を、*y* で指定された領域に戻します。 *x* が NULL オフセット値の場合は、NULL ポインター値が戻されます。

**省略形**: PTR

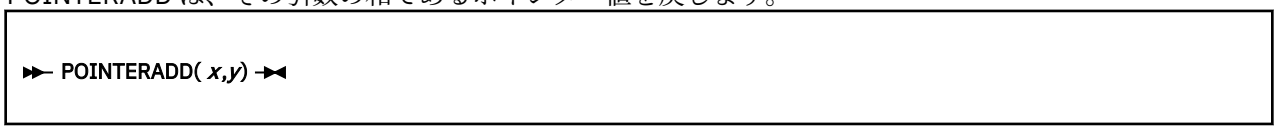
**x**
オフセット参照。 <synph><var>x</var></synph> は、ヌル・オフセット値です。 そうでない場合、*x* は基底付き変数の世代を識別する必要がありますが、必ずしも *y* にある必要はありません。 *y* にない場合、世代は *y* 内の世代と同等でなければなりません。

**y**
区域参照。

最後の生成の割り当てまで、それぞれ同じ回数だけ変数が割り当てられ、解放された場合は、異なる区域での基底付き変数の生成は同等です。

## POINTERADD

POINTERADD は、その引数の和であるポインター値を戻します。




**省略形**: PTRADD

**x**
ポインターの式。

**y**
計算タイプを保持する必要がある式。この式は、FIXED BINARY(31 ,0) に変換されます。

POINTERADD は、基底付き変数のロケーターとして使用できます。

POINTERADD は、減算されるオペランドの接頭部に負符号 (-) を付けることにより、減算に使用できます。

ポインターを増分するために、POINTERADD を使用する必要はありません。 整数の場合と同じようにポインターを増分させるだけです。 例えば、以下のように書き込む必要はありません。

```
p = pointeradd(p,2);
```

その代わりに、以下の同等ステートメントのどちらか 1 つを書き込むことができます。

```
p = p + 2;
p += 2;
```

しかし、POINTERADD は、以下の例にあるように、 ポインターのロケーション・オフセットでのストレージ参照を取り消す際に便利です。

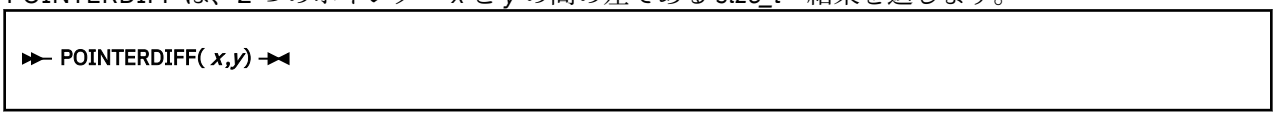
```
dcl x fixed bin(31), b based fixed bin(31);
x = pointeradd(p,2)->b;
```

PL/I のロケーターは参照でなければならないため、書き込みが できないことにご注意ください。

```
x = (p + 2)->b;
```

# POINTERDIFF

POINTERDIFF は、2 つのポインター *x* と *y* の間の差である *size_t* [1] 結果を返します。

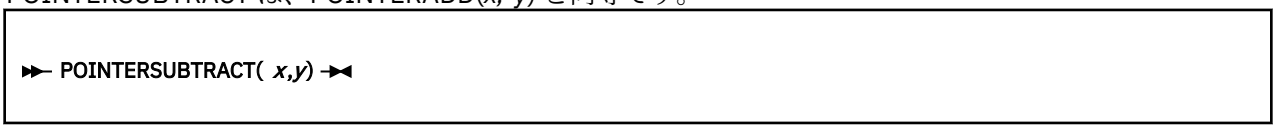



**省略形**: PTRDIFF

**x および y**
POINTER として宣言された式。

# POINTERSUBTRACT

POINTERSUBTRACT は、POINTERADD(*x,-y*) と同等です。




**省略形**: PTRSUBTRACT

**x**
ポインター式でなければなりません。

**y**
計算タイプを保持する必要がある式。この式は、FIXED BINARY(31 ,0) に変換されます。

# POINTERVALUE

POINTERVALUE は、変換された *x* の値であるポインター値を戻します。

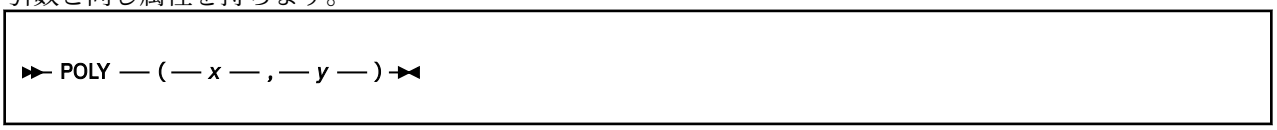
**省略形**: PTRVALUE

**x**
HANDLE 属性か計算タイプのどちらかを保持する必要がある式。 *x* に計算タイプがある場合は、FIXED BINARY(31 ,0) に変換されます。

POINTERVALUE(x) は、*x* が定数の場合に、静的ポインター変数を初期 化するのに使われます。

# POLY

POLY は、1 次元配列式 *x* から形成された多項式の近似である浮動小数点値を返します。 戻り値は、最初の引数と同じ属性を持ちます。




**x**
配列式。

**y**
要素式。

**x** は REAL FLOAT でなくてはなりません。また、必要に 応じて **y** は x の属性に変換されます。

x の下限が 0、上限が n である場合の結果は、係数が x によって与えられ、 y の次元を n とする古典多項式になります。 つまり、次のような結果になります。

```
      x(0) + x(1)*y + x(2)*y**2 + ... + x(n)*y**n
```

x の下限が m、上限が n である一般的なケースでは、 結果は次の多項式になります。

```
x(m) + x(m+1)*y + x(m+2)*y**2 + ... + x(n)*y**(n-m)
```

# POPCNT

POPCNT は、*x* の対応するバイトにおいて 1 になっているビットの数をバイトごとに保持する FIXED BIN 値を返します。

**x**
式。

*x* には、スケール係数がゼロである属性 REAL FIXED BIN が必要です。

結果の精度は、*x* と同じになります。

結果の属性は、*x* と同じ (UN)SIGNED 属性になります。

以下の POPCNT の使用例を参照してください。

- POPCNT( '01020304'xn ) は '01010201'xn を返します。
- POPCNT( '05060708'xn ) は '02020301'xn を返します。
- *x* の属性が FIXED BIN(31) の場合、ISRL(POPCNT(x)*'01010101'xn,24) は、x において 1 と等しいビットの数を返します。

z/OS の場合、POPCNT(*x*) 組み込み関数には ARCH レベル 9 以上が必要です。

# PRECVAL

PRECVAL は、数値表現の精度を提供する FIXED BINARY(31) 値を返します。

**x**
数値表現。

例えば、*x* が FIXED DEC(9,3) として宣言されている場合、PRECVAL(*x*) は 9 を返します。

# PRECISION

PRECISION は、*p* および *q* で指定された精度で *x* の値を返します。 戻り値の基数、モード、およびスケールは、 *x* のそれらと同じです。

**省略形**: PREC

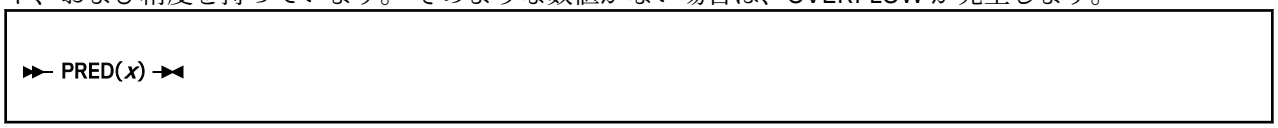
**x**
式。

**p**
制限付き式。 *p* は、式 *x* が変換後に持つ式の値である桁数を指定します。

**q**
制限付き式。 結果の位取り係数を指定します。 固定小数点の場合に、*q* が省略されると、位取り係数ゼロが想定されます。 結果が浮動小数点の場合は、*q* は省略します。

## PRED

PRED は、*x* より小さい表現可能な最大の数値である浮動小数点値を戻します。 これは、x の基数、モード、および精度を持っています。 そのような数値がない場合は、OVERFLOW が発生します。

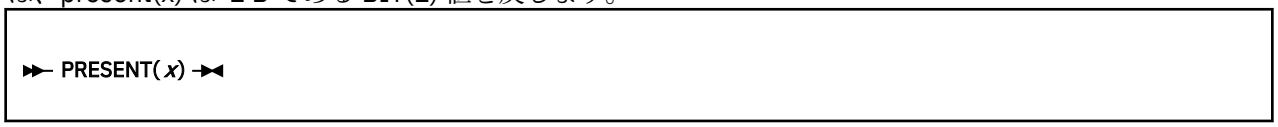



**x**
REAL FLOAT 式。

PRED(TINY(X)) は、ゼロを戻し、UNDERFLOW を発生させることはありません。

## PRESENT

PRESENT(x) が含まれているプロシージャーの呼び出しで、*x* という名前の パラメーターが存在する場合は、present(x) は '1'B である BIT(1) 値を戻します。

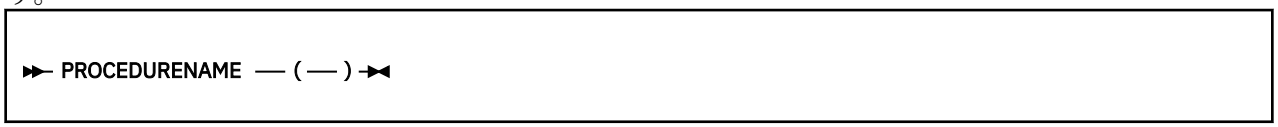



**x**
レベル 1 の添え字が付いていない BYADDR パラメーター。

**注 :** この引数は、呼び出し側コードの対応する ENTRY 宣言で OPTIONAL として宣言する必要があります。

## PROCEDURENAME

PROCEDURENAME は、この組み込み関数を呼び出すプロシージャー名を含む不変文字ストリングを戻します。

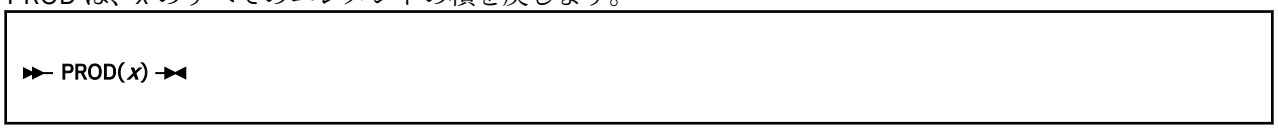



**省略形**: PROCNAME

PROCEDURENAME は、CALL ステートメントまたは GOTO ステートメントに名前が表示されていても複数のラベル指定の左端の名前を常に戻します。

## PROD

PROD は、*x* のすべてのエレメントの積を戻します。



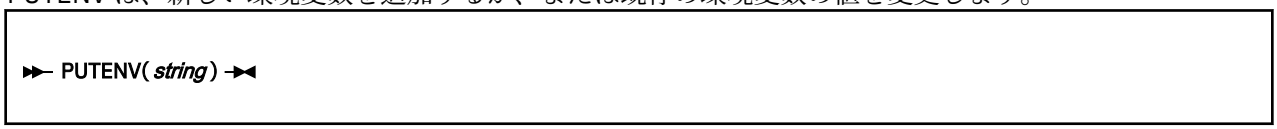

**x**
配列式。 *x* のエレメントがストリングの場合は、それらのエレメントは固定小数点の整数値に変 換されます。

*x* のエレメントが固定小数点の整数値またはストリングでない場合は、浮動小数 点に変換され、結果は浮動小数点になります。

結果は、*x* の精度になります。ただし、固定小数点の整数値 およびストリングの結果は精度 (n,0) の固定小数点になります。ここ で、*n* は許可される最大桁数です。 基数およびモードは、変換後の引数 *x* と一致します。

# PUTENV

PUTENV は、新しい環境変数を追加するか、または既存の環境変数の値を変更します。

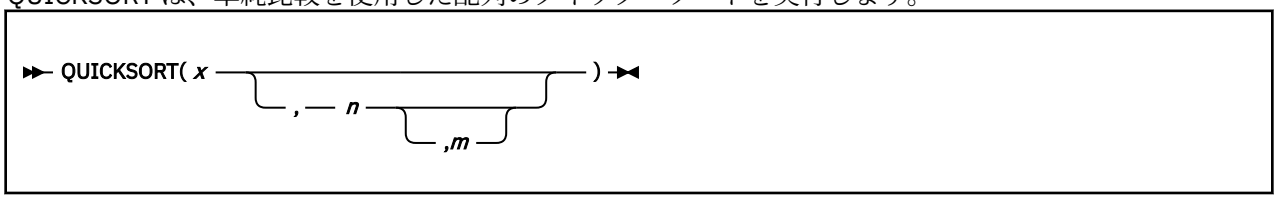



**ストリング**
フォーマット *envvarname=value* の文字ストリング。

成功した場合には、PUTENV は真 ('1'B) を戻し、そうでない場合には、偽 ('0'B) を戻します。

# QUICKSORT

QUICKSORT は、単純比較を使用した配列のクイック・ソートを実行します。




**x**
配列式。 *x* は、スカラーの 1 次元配列でなければなりません。 *x* が NONVARYING BIT の配列である場合は、位置合わせされている必要があります。

配列 *x* のエレメントは、以下のいずれかの条件を満たしている必要があります。

- COMPLEX ではなく、計算値でなければならない。
- POINTER でなければならない。
- HANDLE でなければならない。
- ORDINAL でなければならない。

**n**
調査される最初の配列エレメントの指標を指定する式。 デフォルトは LBOUND(x) になります。

**m**
調査される配列エレメントの数を指定する式。 カウントは *n* 番目から開始し、デフォルトは HBOUND(x) – n + 1 になります。

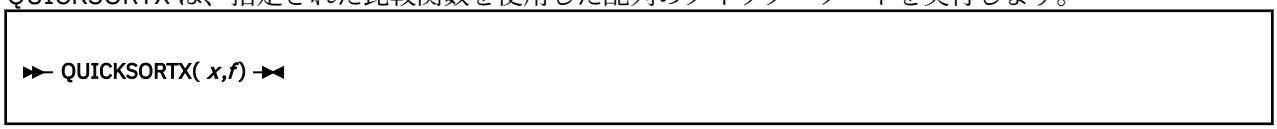
ソートされた配列エレメントは、単純比較の結果に基づいて昇順で格納されます。 2 つのエレメントが等しい 場合、ソートされた配列内でのそれらのエレメントの順序は不定です。

QUICKSORT は、*x* の内容をソート後のエレメントで上書きします。 エレメント j および k に対してクイック・ソートが完了すると、次のようになります。

j < k の場合、*x*(j) < = *x*(k)

# QUICKSORTX

QUICKSORTX は、指定された比較関数を使用した配列のクイック・ソートを実行します。




**x**
配列式。 *x* は、1 次元配列でなければなりません。 *x* が NONVARYING BIT の配列である場合は、位置合わせされている必要があります。

**f**
式。 必要なすべての比較を実行するために呼び出される関数を指定します。

関数 *f* は OPTLINK リンケージを持っている必要があり、配列 *x* から 2 つのエレメントのアドレスを保持する 2 つの POINTER BYVALUE 引数として渡されます。

関数 *f* は、属性 RETURNS( BYVALUE FIXED BINARY(31) ) を持ち、以下のように、値 -1 、0、または +1 のいずれかを返す必要があります。

- 最初の配列エレメントの値が 2 番目の配列エレメントの値より小さい場合、戻り値は -1 でなければなりません。
- 最初の配列エレメントの値が 2 番目の配列エレメントの値と等しい場合、戻り値は 0 でなければなりません。
- 最初の配列エレメントの値が 2 番目の配列エレメントの値より大きい場合、戻り値は +1 でなければなりません。

ソートされた配列エレメントは、比較関数の結果に基づいて昇順で格納されます。

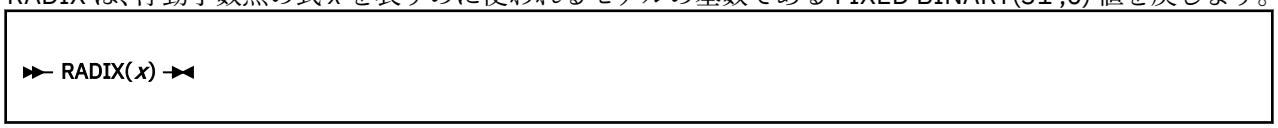
比較関数においてより大 とより小 のロジックを反転させれば、逆順でソートできます。 2 つのエレメントが等しい 場合、ソートされた配列内でのそれらのエレメントの順序は不定です。

QUICKSORTX は、*x* の内容をソート後のエレメントで上書きします。 エレメント j および k に対してクイック・ソートが完了すると、次のようになります。

if j < k, then*f*( addr(*x*(j)), addr(*x*(k)) ) < = 0

## RADIX

RADIX は、浮動小数点の式 *x* を表すのに使われるモデルの基数である FIXED BINARY(31 ,0) 値を戻します。




**x**
REAL FLOAT 式。

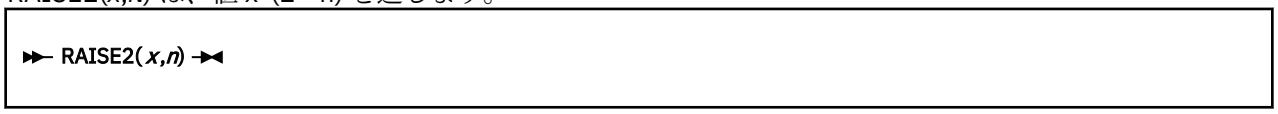
RADIX(x) は、*x* を表すために使用される浮動小数点フォーマットで決まります。 それは次のとおりです。

- *x* が IEEE 2 進浮動小数点フォーマットで保持されている場合は、2 になります。
- *x* が IEEE 10 進浮動小数点フォーマットで保持されている場合は、10 になります。
- *x* が z/OS 16 進フォーマットで保持されている場合は、16 になります。

RADIX(x) は、制限付き式に使用できます。

## RAISE2

RAISE2(*x*,*n*) は、値 x*(2**n) を返します。




***x***
式。 *n* は、計算タイプでなくてはなりません。

***n***
式。 *n* は、計算タイプでなくてはなりません。

*x* が REAL FIXED BIN(p,0) および SIGNED である場合、結果は SIGNED REAL FIXED BIN(r,0) です。ここで、p <= M1 の場合は r = M1、p > M1 の場合は r = M2 です。

*x* が REAL FIXED BIN(p,0) および UNSIGNED である場合、結果は UNSIGNED REAL FIXED BIN(r+1,0) です。ここで、p <= (M1+1) の場合は r = (M1+1)、p > (M1+1) の場合は r = (M2+1) です。

それ以外の場合、*x* は SIGNED REAL FIXED BIN(p,0) に変換され、その結果は上記と同じ属性を持ちます。

*n* が負の場合、または *n* が r より大きい場合、結果は未定義です。

**注 :** RAISE2(*x*,*n*) はアセンブラー SLA(*x*,*n*) と同等です。

**例**

```
raise2(6,1)                    /*  produces 12  */
```

# RANDOM

RANDOM は、与えられたシード *x* を使用して生成されたランダム数 FLOAT BINARY(53) を戻します。 *x* を省略すると、ランダム数は、シードを与えた 最後の RANDOM 呼び出しのシードに基づくか、シードを与えた RANDOM がこれ までに呼び出されていない場合は、デフォルトの初期シード 1 に基づいて生成されます。

**x**

式。 *x* に、計算タイプ (必須) および算術タイプを保持している必要があります。 *x* は数値で、実数でなければなりません。 *x* が指定された FIXED BINARY(31 ,0) でない場合は、x は変換されます。

0 < *x* < 2,147,483,646 でない場合は、ERROR 条件が発生します。

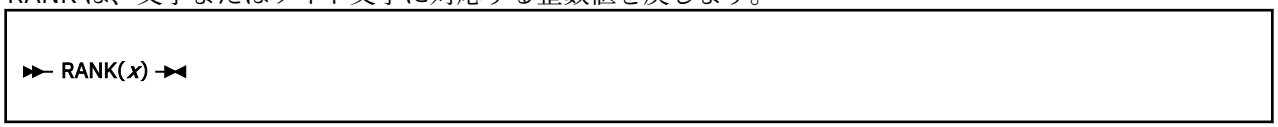
RANDOM によって生成される値は、0 < random(x) < 1 で 0 と 1 の間に均等に分散されます。 これらは、次のように乗算合同式法を使用して生成されます。

```
seed(x) = mod(950706376 * seed(x - 1), 2147483647)
random(x) = seed(x) / 2147483647
```

シードは、マルチスレッド化アプリケーションの各スレッド内ではなく、プログラム・レベルで維持されます。

# RANK

RANK は、文字またはワイド文字に対応する整数値を戻します。




**x**

属性 CHAR (1) NONVARYING または WCHAR (1) NONVARYING を 保持しなければなりません。

*x* が文字である場合、RANK(*x*) は `index(collate(),x)-1` として定義され、RANK は CHARVAL の逆になります。

*x*x がワイド文字である場合、ビッグ・エンディアン・フォーマットで格納された x を y と すると、RANK(*x*) は UNSPEC(*y*) に等しくなります。

# REAL

REAL は、*x* の実数部分を返します。 結果は、*x* の基数、位取り、および精度になります。

**x**

式。 *x* が実数の場合は、x は複素数に変換されます。

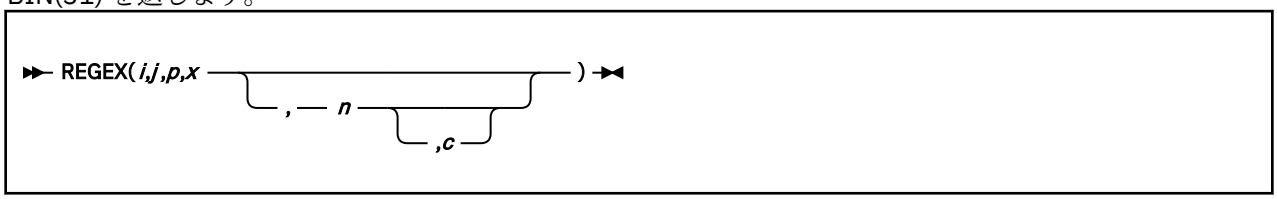
# REAL 疑似変数

REAL 疑似変数は、実数値または複合値の実数部分を *x* の実数部分に 割り当てます。

**x**
複素数参照。

# REGEX

REGEX は、あるストリングに対して指定された正規表現またはパターンと一致したことを示す FIXED BIN(31) を返します。




**i**
参照。 *i* は ASSIGNABLE でなければなりません。 パターンの一致が検出されると、正規表現 *p* の最初の一致の *x* 内のサブストリングの索引が割り当てられます。*i* は、スケール因数 0 を持つ REAL FIXED BIN でなければなりません。*i* はスカラーまたはスカラーの 1 次元配列のいずれかでなければなりません。

**j**
参照。 *j* は ASSIGNABLE でなければなりません。 パターンの一致が検出されると、正規表現 *p* の最初の一致の *x* 内のサブストリングの長さが割り当てられます。*j* は、スケール因数 0 を持つ REAL FIXED BIN でなければなりません。*j* はスカラーまたはスカラーの 1 次元配列のいずれかでなければなりません。

**p**
正規表現を保持するストリング。 パターン *p* は、CHARACTER タイプを保持していなければなりません。

パターン *p* は、(基本正規表現用の POSIX 標準ではなく) 拡張正規表現用の POSIX 標準に準拠している必要があります。 Wikipedia やその他の Web サイトに、正規表現についてわかりやすく説明されています。

**x**
ストリング。 正規表現 *p* との一致を見つけるために *x* が検索されます。 ストリング *x* は CHARACTER タイプでなければなりません。

**n**
式。 *n* は、検索を開始する *x* 内の位置を指定します。 *n* は計算タイプでなければならず、FIXED BINARY(31,0) に変換されます。 省略した場合、デフォルトで 1 になります。

**c**
制限付き式。 *c* は、*p* および *x* のコード・ページを指定します。 省略した場合、デフォルトで CODEPAGE コンパイラー・オプション内の値に設定されます。 省略しない場合は、*n* の値を指定する必要があります。

コード・ページは、計算タイプでなければならず、 FIXED BINARY (31,0) に変換されます。 コード・ページでは、有効かつサポートされているコード・ページを指定する必要があります。

*i* または *j* のいずれかが配列である場合は、以下のようになります

- 両方とも、境界と NATIVE タイプの size_t が一致する配列でなければなりません
- 各配列の最初のエレメントには、一致する式 (ある場合) の索引と長さが割り当てられます。
- 各配列の 2 番目以降のエレメントには、対応する一致する副次式 (ある場合) の索引と長さが割り当てられます。

文字 [、]、{、}、|、 ^、および $ は正規表現でよく使用される文字であり、エンコードされた異なる文字セットにさまざまなコード・ポイントがあります。 (暗黙的または明示的) コード・ページ値は、*p* および *x* の

コード・ページと正しく一致している必要があります。 そうでない場合は、パターンが無効と見なされるか、一致するものが見つからない可能性があります。

REGEX 組み込み関数の処理は、以下のステップで行われます。

1. *n* が 1 より小さい場合、または *n* が 1 +length(x) よりも大きい場合は、STRINGRANGE 条件 (割り込み可能な場合) が発生し、REGEX は値 1 を返します。
2. コード・ページ *c* と一致するロケールが存在しない場合、REGEX は値 -1 を返します。
3. ストリング *p* が有効な正規表現を指定していない場合、REGEX は 1 より大きい値を返します。
4. 正規表現 *p* のストリング *x* に一致するものがない場合、REGEX は値 1 を返し、索引 *i* と長さ *j* を 0 に設定します。 それ以外の場合、REGEX は値 0 を返し、正規表現 *p* に最初に一致する *x* 内のサブストリングに対応する索引 *i* と長さ *j* を設定します。

正規表現に基づく一致の検索では、大/小文字が区別されます。

**例**

例 1

*p* = "All(a|e)n" かつ *x* = "12Allan3Allen4Alan5Allan678" の場合、次のようになります

`regex( i, j, p, x )` は 0 を返し、*i* を 3、*j* を 5 に設定します (最初の "Allan" の一致が見つかったため)。
`regex( i, j, p, x, 4 )` は 0 を返し、*i* を 9、*j* を 5 に設定します ("Allen" の一致が見つかったため)。
`regex( i, j, p, x, 10 )` は 0 を返し、*i* を 20、*j* を 5 に設定します (2 番目の "Allan" の一致が見つかったため)。
`regex( i, j, p, x , 21 )` は 1 を返します (これ以上の一致がないため)。

また、先行する一致セットは、ストリング *x* をウォークスルーするためのオプションの 5 番目のパラメーターを使用する次のループで見つかった可能性もあります。

```
n = 1;
do loop;
  rc = regex( i, j, p, x, n );
  if rc <> 0 then leave;
  put skip list( substr( x, i, j ) );
  n = i + j;
end;
```

例 2

*p* = "[hc]+at" および *x* = "the cat in the hat" の場合、`regex( i, j, p, x, n )` は、*n* の値に応じて、"cat" または "hat" の一致を検索します。 ただし、*p* = "63"x || "hc" || "fc"x || "+at" であると、コード・ページ 1141 でも、このパターンは "[hc]+at" と表示されます。

- デフォルトのコード・ページ 1140 では、`regex( i, j, p, x, n )` は一致を検出しません。コード・ページ 1140 では、[ の 16 進値は "ba"x であり、] の 16 進値は "bb"x であるためです。
- ただし、`regex( i, j, p, x, n, 1141)` は、*n* の値に応じて、"cat" または "hat" の一致を検索します。

例 3

以下の場合、

```
pattern = '([a-zA-Z]+) * ([a-zA-Z]+) * ((([a-zA-Z1-9]+)\.){0,1}([a-zA-Z1-9]+))';
string = ' CREATE DATABASE TESTDB;';
rc = regex( a_index, a_length, pattern, string );
```

Then

```
a_index(2) and a_length(2) will give the index and length for CREATE
```

```
a_index(3) and a_length(3) will give the index and length for DATABASE
a_index(4) and a_length(4) will give the index and length for TESTDB
```

# REM

REM は、*x* を *y* で除算した剰余を戻します。

これは、次の式で計算できます。

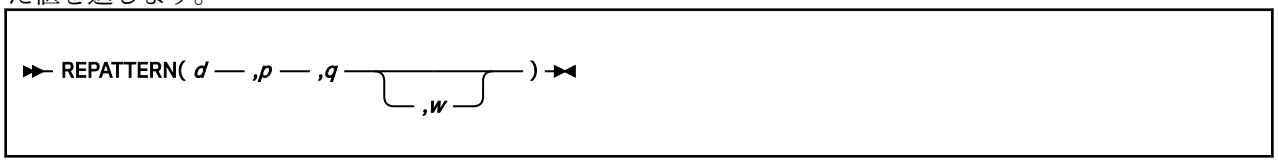
```
x - y * trunc(x/y)
```

**x および y**
式。 *x* および *y* は、計算タイプでなければなりません。 また、算術タイプでも構いません。

REM 組み込み関数と MOD 組み込み関数の対比の例について は、487 ページの『MOD』を参照してください。

# REPATTERN

REPATTERN は、1 つのパターンでは日付を保持している値をとり、2 番目のパターンでは日付に変換された値を返します。




**d**
日付を表すストリング式。 *d* の長さは、少なくともソース・パターン *q* の長さと同じでなければなりません。 *d* のほうが長い場合は、余分の文字を先行ブランクにしなければなりません。

*d* に、計算タイプ (必須) と文字タイプを保持している必要があります。 そうでないと、文字に変換されます。

**p**
ターゲット・パターン。サポートされている日付/時刻パターンの 1 つでなければなりません。

**q**
ソース・パターン。サポートされている日付/時刻パターンの 1 つでなければなりません。

**w**
整数に変換できる式 (1950 など) を指定します。 負の場合、コードを実行するときに年の値から減算されるようにオフセットを指定します。 省略される場合は、*w* は、WINDOW コンパイル時オプションに指定された値に デフォルトで設定されます。

戻り値は、属性 CHAR(m) NONVARYING を持っています。*m* はターゲット・パターン *p* の長さです。

許可されるパターンを 382 ページの表 65 にリストしてあります。 リリアン形式の説明については、380 ページの『日付/時刻組み込み関数』を参照してください。

REPATTERN 組み込み関数は、以下の両方が真である場合に、指定されたインライン変換を実行します。

- ソース・パターンとターゲット・パターンは DDD エレメントを使用しない。
- ソース・パターンにターゲットと同じ量の日付情報がある (つまり、ターゲットに年、月、日がある場合、ソースにも対応する情報が必要で、少なくともソース年の桁数がターゲットと同じである必要がある)。

例えば、以下のようになります。

- YYYYMMDD から DD.MM.YY の変換はインライン化されます。
- MM/DD/YYYY から YYMM への変換はインライン化されます。

- MMYY から YYYYMMDD への変換はインライン化されません。

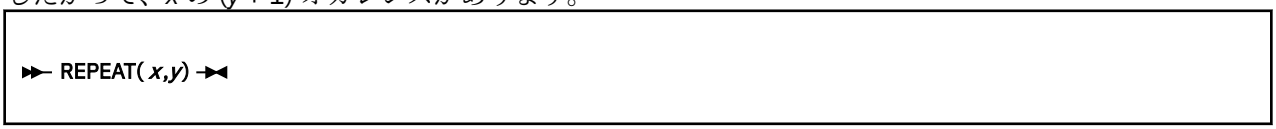
以下に、REPATTERN を使用して、年を 2 桁で表す日付パターンと 4 桁で表す日付パターンの間で変換を行う方法について、いくつかの例を示します。 ただし、両方のパターンが年の値を保持するのに同じ桁数を使用している場合でも、この組み込み関数を使用して、日付をあるサポートされたパターンから別のサポートされたパターンへ変換することができます。

REPATTERN('990101','YYYYMMDD','YYMMDD', 1950) returns '19990101'
REPATTERN('000101','YYYYMMDD','YYMMDD', 1950) returns '20000101'
REPATTERN('19990101','YYMMDD','YYYYMMDD', 1950) returns '990101'
REPATTERN('20000101','YYMMDD','YYYYMMDD', 1950) returns '000101'
REPATTERN('19490101','YYMMDD','YYYYMMDD', 1950) raises ERROR

## REPEAT

REPEAT は、*y* で指定された回数だけ連結した *x* で構成されるストリングを返します。

したがって、*x* の (*y* + 1) オカレンスがあります。




**x**
繰り返されるビット、文字、グラフィック、uchar、またはワイド文字の式。 *x* が算術の場合は、次の変換が発生します。

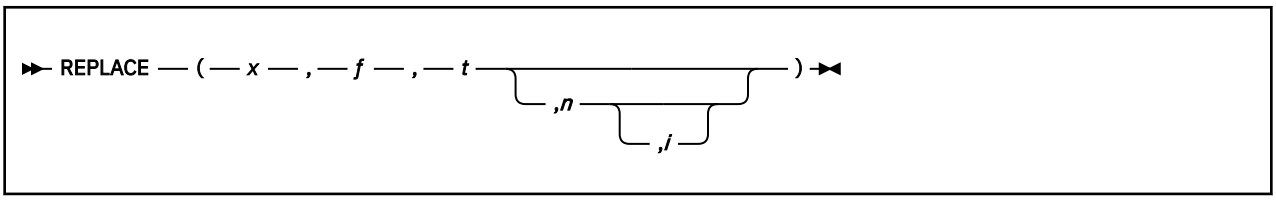
- 2 進数の場合は、*x* はビット・ストリングに変換されます。
- 10 進数の場合は、*x* は文字ストリングに変換されます。

**y**
式。 必要に応じて、*y* は実数の固定小数点 2 進数値に変換されます。

*y* がゼロまたは負の場合は、ストリング *x* が戻されます。 REPEAT 組み込み関数の例については、420 ページの『COPY』を参照してください。

## replace

REPLACE は、別のサブストリングに置き換えられたサブストリングのオカレンスが 1 つ以上あるストリングを返します。




**x**
サブストリング f のオカレンスをサブストリング t に置き換えるストリングを指定するストリング式。 x は CHARACTER タイプを保持していなければなりません。

**f**
ストリング x 内で置換されるサブストリングを指定する文字列式。 f は CHARACTER タイプでなければなりません。

**t**
ストリング x 内のサブストリング f を置換するために使用されるサブストリングを指定する文字列式。 t は CHARACTER タイプでなければなりません。

**n**
コンパイラーがサブストリング f の検索を開始する、ストリング x 内の場所を指定するオプションの式。 n は計算タイプでなければならず、FIXED BINARY (31,0) に変換されます。 n のデフォルト値は 1

です。 n が 1 より小さいか、または length(x) より大きい場合は、STRINGRANGE 条件が発生し (有効になっている場合)、結果はヌル文字ストリングになります。

**i**
サブストリング f をサブストリング t に置き換える必要がある最大回数を指定するオプションの式。 i は計算タイプでなければならず、FIXED BINARY(31,0) に変換されます。 i のデフォルト値は 1 です。 負でない値でなければなりません。 i が 0 の場合、 ストリング x 内のサブストリング f のすべてのオカレンスがサブストリング t に置き換えられます。

```
dcl ein char(50) var init( 'reserved from #date# till #date#.' );
dcl aus char(80) var;

dcl f   char(6);
dcl t   char(10);

f = '#date#';
t = '2018/05/01';

aus = replace( ein, f, t );
      /* 'reserved from 2018/05/01 till #date#.' */
aus = replace( ein, f, t, 16 );
      /* 'reserved from #date# till 2018/05/01.' */
aus = replace( ein, f, t, 1, 2 );
      /* 'reserved from 2018/05/01 till 2018/05/01.' */
aus = replace( ein, f, t, 16, 1 );
      /* 'reserved from #date# till 2018/05/01.' */
aus = replace( ein, f, t, 1, 0 );
      /* 'reserved from 2018/05/01 till 2018/05/01.' */
```

## REPLACEBY2

REPLACEBY2 は、*x* の中の一部の文字を文字のペアで置き換えること によって形成される、固定長ストリングを戻します。

**x**
置換できる文字があるかどうかを検索される文字式。

**y**
置換するペアの値を含む文字式。

**z**
置換される文字を含む文字式。

REPLACEBY2 は、*x* の各文字を次のように処理します。

*x* の中にある文字が *z* の中で検出されると、*y* にあ る *z* に対応する文字ペアが結果にコピーされます。それ以外の場合は、*x* の中の文字が結果にコピーされます。 *z* に複数の同一文字が入っている場合は、左端のオカレンスが使われます。

ストリング *y* は、ストリング *z* の 2 倍の長さでなければなりません。

例として、REPLACEBY2( 'Rätsel', 'aeoeuess', 'äöüß') はストリング 'Raetsel' を返します。

## REVERSE

REVERSE は、順序を反対にした *x* のエレメントを含む不変ストリングを戻します。

**x**
式。 *x* に、計算タイプ (必須) とストリング・タイプを保持している必要があります。 *x* にストリング・タイプがない場合は、 連結の規則に従って、ストリング (すなわち、数値から文字、ビット、グラフィック、uchar、またはワイド文字) に変換されます。

**例**

```
dcl Source char value('HARPO');
dcl Target char(length(Source));

Target = reverse (Source);     /* 'OPRAH' */
```

# RIGHT

RIGHT は、長さ *n* のストリング *x* の右端にストリング x を挿入し、必要な場合に左側が文字 *z* を埋めた結果のストリングを戻します。

*z* が省略される場合は、埋め込み文字にブランクが使われます。

**x**
式。 *x* に、計算タイプ (必須) を保持している必要があります。また、文字タ イプを保持することもできます。 そうでないと、文字に変換されます。

**n**
式。 *n* は計算タイプでなければならず、FIXED BINARY(31 ,0) に変換されます。

**z**
式。 z が指定された場合、*z* は CHARACTER(1) NONVARYING タ イプでなければなりません。

**例**

```
dcl Source char value('One Hundred SCIDS Marks');
dcl Target char(30);

Target = right (Source, length(Target), '*');
           /* '*******One Hundred SCIDS Marks'               */
```

# ROUND

ROUND は、*x* の値を *n* で指定された桁で丸めた値を返します。 結果は、*x* のモード、基数、およびスケールになります。

**x**
実数式。 *x* が負の場合は、絶対値が丸められてから、符号が戻ります。

**n**
オプションの符号付き整数。 丸める桁を指定します。

## FIXED の ROUND

FIXED の結果の精度は、次の式によって得られます。

```
(max(1,min(p-q+1+n,N)),n)
```

ここで、(*p*,*q*) は *x* の精度で、N は許可される桁数の最大数です。 したがって、*n* は結果の位取り係数を指定します。

*n* は、FIXED データの位取り係数の制限に従う必要が あります。 *n* が 0 より大きい場合は、小数点の右から (*n*) 番目の桁が丸め られます。 *n* がゼロまたは負の場合は、小数点の左から (1-*n*) 番目の桁が丸められます。

ROUND(*x*,*n*) では、*n* <= p は true でなければなりません。

結果の値は、次の式によって得られます。ここで、*x* が BINARY の場合は b = 2 で、 *x* が DECIMAL の場合は b = 10 です。

```
round(x,n) = sign(x)*(b-n)* floor(abs(x)* (bn) + 1/2)
```

したがって、次の例では値 6.67 が出力となります。

```
    dcl X fixed dec(5,4) init(6.6666);

    put skip list( round(X,2) );
```

## IEEE 10 進浮動小数点の ROUND

IEEE DECIMAL FLOAT の結果の精度は、ソースの引数の精度と同じです。

結果の値は、次の式によって得られます。ここで、b = 10 (=radix(x)) で、e = exponent(x) です。

```
round(x,n) = sign(x)*(b(e-n))* floor(abs(x)* (b(n-e)) + 1/2)
```

したがって、FLOAT(DFP) コンパイラー・オプションが有効である場合、以下の連続した 3.1415926d0 の丸めにより、次の値が算出されます。

```
    dcl x float dec(16) init( 3.1415926d0 );

    display( round(x,1) );  /* 3.000000000000000E+0000 */
    display( round(x,2) );  /* 3.100000000000000E+0000 */
    display( round(x,3) );  /* 3.140000000000000E+0000 */
    display( round(x,4) );  /* 3.142000000000000E+0000 */
    display( round(x,5) );  /* 3.141600000000000E+0000 */
    display( round(x,6) );  /* 3.141590000000000E+0000 */
```

## IEEE 2 進浮動小数点の ROUND

IEEE 2 進浮動小数点の結果の精度は、ソースの引数の精度と同じです。

コンパイラー・オプション USAGE(ROUND(IBM)) のもとでは、z/OS 上以外、結果の値はソースと同じになります。ここで、ソースがゼロでない場合は、結果は、ソースの右端ビットで定まることによって得られます。

コンパイラー・オプション USAGE(ROUND(ANS)) のもとでは、結果の値は次の式によって得られます。ここで、b = 2 (=radix(x)) で、e = exponent(x) です。

```
round(x,n) = sign(x)*(b(e-n))* floor(abs(x)* (b(n-e)) + 1/2)
```

USAGE(ROUND(ANS)) のもとでは、丸めは基数 2 の丸めであり、認識の甘いユーザーが予測したものと異なる結果が出ることがあります。 例えば、USAGE(ROUND(ANS)) でコンパイルされ、IEEE 2 進浮動小数点命令が使用された場合は、以下の連続した 3.1415926d0 の丸めにより、次の値が算出されます。

```
    dcl x float bin(53) init( 3.1415926d0 );

    display( round(x,1) );  /* 4.000000000000000E+0000 */
    display( round(x,2) );  /* 3.000000000000000E+0000 */
    display( round(x,3) );  /* 3.000000000000000E+0000 */
    display( round(x,4) );  /* 3.250000000000000E+0000 */
    display( round(x,5) );  /* 3.125000000000000E+0000 */
    display( round(x,6) );  /* 3.125000000000000E+0000 */
    display( round(x,7) );  /* 3.156250000000000E+0000 */
```

## IBM 16 進浮動小数点の ROUND

IBM 16 進浮動小数点の結果の精度は、ソースの引数の精度と同じです。

コンパイラー・オプション USAGE(ROUND(IBM)) のもとでは、z/OS 上以外、結果の値はソースと同じになります。ここで、ソースがゼロでない場合は、結果は、ソースの右端ビットで定まることによって得られます。

コンパイラー・オプション USAGE(ROUND(ANS)) のもとでは、結果の値は次の式によって得られます。ここで、b = 16 (=radix(x)) で、e = exponent(x) です。

```
round(x,n) = sign(x)*(b(e-n))* floor(abs(x)* (b(n-e)) + 1/2)
```

USAGE(ROUND(ANS)) のもとでは、丸めは基数 16 の丸めであり、認識の甘いユーザーが予測したものと異なる結果が出ることがあります。 例えば、USAGE(ROUND(ANS)) でコンパイルされ、IBM 16 進浮動小数点命令が使用された場合は、以下の連続した 3.1415926d0 の丸めにより、次の値が算出されます。

```
dcl x float bin(53) init( 3.1415926d0 );

display( round(x,1) );  /*  3.000000000000000E+00  */
display( round(x,2) );  /*  3.125000000000000E+00  */
display( round(x,3) );  /*  3.140625000000000E+00  */
display( round(x,4) );  /*  3.141601562500000E+00  */
display( round(x,5) );  /*  3.141586303710938E+00  */
display( round(x,6) );  /*  3.141592979431152E+00  */
```

## ROUNDAWAYFROMZERO

ROUNDAWAYFROMZERO は、ゼロから遠い方への丸めの規則に従って、*n* によって指定された桁で丸められた *x* の値を返します。 結果は、*x* のモード、基数、およびスケールになります。

**注 :** ROUNDAWAYFROMZERO 組み込み関数は、以前は ROUNDDEC という名前でした。

**x**
FIXED DECIMAL または DFP FLOAT の実数式。 *x* が負の場合は、絶対値が丸められてから、符号が戻ります。

**n**
丸めを実行する桁を指定する、オプションの符号付き整数。

*x* が FIXED DECIMAL または PICTURE FIXED DECIMAL の場合、ROUNDAWAYFROMZERO は ROUND と同じ結果を生成します。

*x* が FLOAT DECIMAL または PICTURE FLOAT DECIMAL で、FLOAT(DFP) コンパイラー・オプションが有効である場合、ROUNDAWAYFROMZERO は n 桁目ではなく、小数第 *n* 位で *x* を丸めます (ANSI 定義に従った ROUND 組み込み関数と同様)。 例えば、以下の連続した 3141.592653589793d0 の丸めにより、次の値が算出されます。

```
dcl x float dec(16) init( 3141.592653589793d0 );

display( fixed(roundawayfromzero(x,1),15,7) );  /* 3141.6000000 */
display( fixed(roundawayfromzero(x,2),15,7) );  /* 3141.5900000 */
display( fixed(roundawayfromzero(x,3),15,7) );  /* 3141.5930000 */
display( fixed(roundawayfromzero(x,4),15,7) );  /* 3141.5927000 */
display( fixed(roundawayfromzero(x,5),15,7) );  /* 3141.5926500 */
display( fixed(roundawayfromzero(x,6),15,7) );  /* 3141.5926540 */
display( fixed(roundawayfromzero(x,7),15,7) );  /* 3141.5926536 */
```

ROUNDAWAYFROMZERO は、CEIL、FLOOR、および TRUNC 組み込み関数を補完します。

- ROUNDAWAYFROMZERO(*x*,0) は、ゼロの逆方向に丸めます。
- CEIL(*x*) は、正の無限大方向に丸めます。
- FLOOR*x*) は、負の無限大方向に丸めます。
- TRUNC*x*,0) は、ゼロ方向に丸めます。

## ROUNDTOEVEN

ROUNDTOEVEN は、偶数への丸めの丸め規則に従って、*n* によって指定された桁で丸められた *x* の値を返します。

**x**
FIXED DECIMAL または DFP FLOAT の実数式。 *x* が負の場合、最も近い偶数値に丸められ、符号が復元されます。

**n**
丸めを実行する桁を指定する、オプションの符号付き整数。

ROUNDTOEVEN 組み込み関数は、基本的に ROUNDAWAYFROMZERO 組み込み関数と同じですが、ROUNDAWAYFROMZERO 関数では、ゼロから遠い方にタイが丸められる点が異なります。 例えば、ROUNDAWAYFROMZERO 関数では、24.5 は 25 に丸められ、-24.5 は -25 に丸められます。 しかし、ROUNDTOEVEN 関数では、23.5 と 24.5 は両方とも 24 に丸められ、-23.5 と -24.5 は両方とも -24 に丸められます。

## SAMEKEY

SAMEKEY は、アクセスされたレコードに同じキーを持つ別のレコードが続いているかどうかを示す、長さ 1 のビット・ストリングを戻します。

**x**
ファイル参照。 ファイルは、RECORD 属性をとる必要があります。

ファイル *x* での入出力操作の正常終了時、または RECORD 条件が発生する直前に、処理レコードの後に同じキーを持つ別のレコードが続いている場合は、SAMEKEY によってアクセスされる値が '1'B に設定されます。そうでない場合は、'0'B に設定されます。

SAMEKEY によってアクセスされる値は、次の場合にも '0'B に設定されます。

- RECORD 以外の条件が発生する入出力演算が、ファイルの位置が変更または 失われる原因になる場合
- ファイルがオープンされていない場合
- 現行カーソル位置がファイル内にない場合

## SCRUBOUT

SCRUBOUT は、2 番目のストリングからすべての文字を除去したストリングを返します。

**x**
ストリング *f* 内の文字が除去されるストリングを指定するストリング式。 *x* は CHARACTER タイプを保持していなければなりません。

**f**
*x* から削除される文字を指定するストリング式。 *f* は CHARACTER タイプでなければなりません。

**n**
コンパイラーがストリング *f* からの文字の検索を開始するストリング *x* 内の位置を指定するオプションの式。

*n* は計算タイプでなければならず、FIXED BINARY(31,0) に変換されます。 *n* のデフォルトは 1 です。

*n* が 1 より小さい場合、または length(*x*)+1 よりも大きい場合、 STRINGRANGE 条件 (割り込み可能な場合) が発生し、結果はヌル文字ストリングになります。

SCRUBOUT( *x*, '0123456789' ) は、*x* からすべての数字を除去します。

SCRUBOUT( *x*, '0123456789', 4 ) は、*x* から最初の 3 文字より後のすべての数字を除去します。

# SCALE

SCALE は、浮動小数点数を基数の整数乗で乗算します。

SCALE は、次の式に基づいて浮動小数点の値を戻します。

$$x*(\mathrm{radix}(x))^{n}$$

結果は、*x* の基数、モード、および精度になります。

**x**
REAL FLOAT 式。

**n**
式。 計算タイプでなければならず、FIXED BINARY(31,0) に変換されます。

# SCALEVAL

SCALEVAL は、数値表現の位取り係数を提供する FIXED BINARY(31) 値を返します。

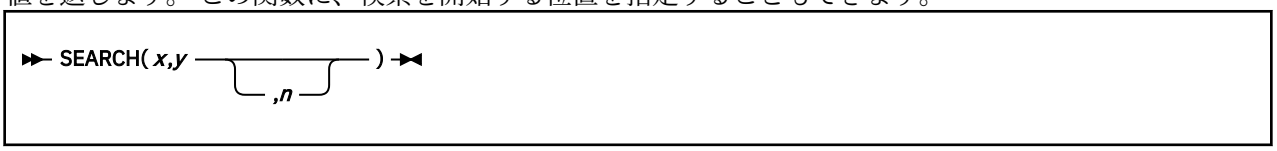
**x**
数値表現。

*x* が FLOAT の場合、返される値は 0 です。

例えば、*x* が FIXED DEC(9,3) として宣言されている場合、SCALEVAL(*x*) は 3 を返します。

# SEARCH

SEARCH は、別のストリングにある任意の 文字、ビット、グラフィック、uchar、またはワイド文字が指定のストリングにある場合、そのストリング内の最初の位置を指定するスケールのない REAL FIXED BINARY 値を返します。 この関数に、検索を開始する位置を指定することもできます。




**x および y**
式。 *x* は、ストリング *y* に表示される 文字、ビット、グラフィック、uchar、またはワイド文字を検索するストリングを指定します。

*x* または *y* はヌル・ストリングで、結果はゼロです。

*x* に *y* がない場合は、結果はゼロです。

**n**
式。 *n* は、検索を開始する *x* 内の位置を指定します。 計算タイプでなければならず、FIXED BINARY(31,0) に変換されます。

1 ≤ *n* ≤ LENGTH(*x*)+1 でない場合は、 STRINGRANGE 条件 (割り込み可能な場合) が発生します。 暗黙処置と通常の戻りは、結果ゼロになります。

BIFPREC コンパイラー・オプションによって、戻される結果の精度が決まります。

SEARCH は、数字のストリングにある区切り文字を検出するのに使われます。

SEARCH は、2 番目および 3 番目の引数がリテラル、VALUE 属性を使用して宣言された名前付き定数、または制限付き式のいずれかである場合に、最も機能を発揮します。

**例 1**

```
   dcl Source char value(' Our PL/I wields the Power ');
   dcl Pos fixed bin(31);

/* Find occurrences of any of the characters 'P','o',or 'w' in source *  /

   Pos = search (Source, 'Pow');              /*  returns 6 for the 'P'    */
   Pos = search (Source, 'Pow', Pos+1);    /*  returns 11 for the 'w'   */
   Pos = search (Source, 'Pow', Pos+1);    /*  returns 22 for the 'P'   */
   Pos = search (Source, 'Pow', Pos+1);    /*  returns 23 for the 'o'   */
   Pos = search (Source, 'Pow', Pos+1);    /*  returns 24 for the 'w'   */

   Pos = index (source, 'Pow',1);             /*  returns 22 for the 'Pow' */
```

上記の例では、SEARCH は 3 文字 (P, o, または w) のどれかが表れる位置 を戻します。 INDEX はストリング Pow が表れる位置を戻します。

**例 2**

```
   dcl Source char value (' 368,475;121.,856,478')
   dcl Delims char(3) init (',;.');             /* string of delimiters */
   dcl Number(5) char(3);
   dcl Start fixed bin(31);
   dcl End fixed bin(31);

   /* Extract the three-digit numbers from the source string */
   /* by searching for the delimiters                        */
   Start = verify (Source, ' ');
                     /* find start of first number */
   End   = search (Source, ',;.', Start );
                     /* find end of first number */
   if End = 0 then
      End = length (Source) + 1;
   Number(1) = substr (Source, Start, 3);        /* 368 */
   Start = verify (Source, Delims, End);
               /* find start of second number */
   End   = search (Source, Delims, Start );
   Number(2) = substr (Source, Start, 3);        /* 475 */
```

# SEARCHR

SEARCHR は、ストリングの 1 つのエレメントが別のストリング内で最初に現れる位置を検索しますが、検索は右から開始されます。

SEARCHR 関数は、次のこと以外は SEARCH 組み込み関数と同じ演算を行います。

- 検索は、右から左に行われる。
- *n* のデフォルトは LENGTH(*x*)。
- 0 ≤ *n* ≤ LENGTH(*x*) でないと、STRINGRANGE 条 件 (割り込み可能な場合) が発生します。 暗黙処置と通常の戻りは、結果ゼロになります。

BIFPREC コンパイラー・オプションによって、戻される結果の精度が決まります。

SEARCHR は、2 番目および 3 番目の引数がリテラル、VALUE 属性を使用して宣言された名前付き定数、または制限付き式のいずれかである場合に、最も機能を発揮します。

### 例

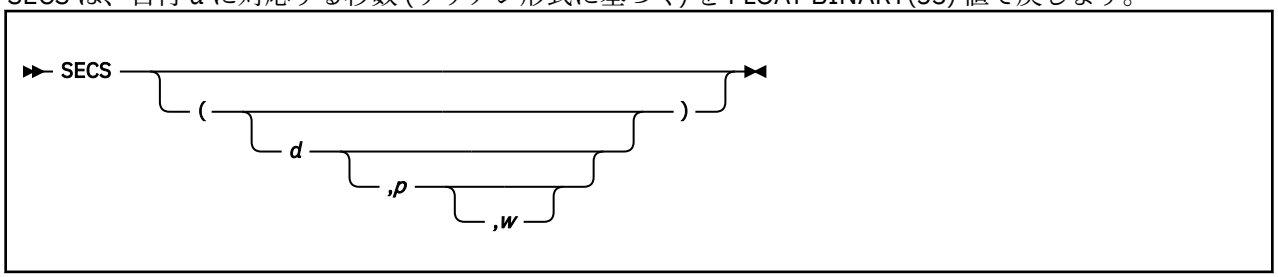
```
   dcl Source char value (' 555 Bailey Ave, San Jose, CA 95141, USA');
   dcl Digits char value ('0123456789');
   dcl (Start, End) fixed bin(31);
   dcl Num char(20) var;

      /*   Find last number (i.e., zip code)    */

   End   = searchr (Source, Digits);        /* returns 35 for the '1' */
   Start = verifyr (Source, Digits, End);   /* returns 30 for the ' ' */
   Num   = substr (Source, Start + 1, End - Start);  /* extract number */
```

### 関連情報

532 ページの『SEARCH』
SEARCH は、別のストリングにある任意の 文字、ビット、グラフィック、uchar、またはワイド文字が指定のストリングにある場合、そのストリング内の最初の位置を指定するスケールのない REAL FIXED BINARY 値を返します。 この関数に、検索を開始する位置を指定することもできます。

## SECS

SECS は、日付 *d* に対応する秒数 (リリアン形式に基づく) を FLOAT BINARY(53) 値で戻します。




**d**

日付を表すストリング式。 ある場合、*d* は、パターン *p* で指定された日付/時刻を表す文字ストリングとして入力日付を指定します。 *d* が欠落している場合は、DATETIME() であると想定されます。

*d* に、計算タイプ (必須) と文字タイプを保持している必要があります。 そうでないと、文字に変換されます。

**p**

サポートされている日付/時刻パターンの 1 つ。 *p* が省略される場合は、デフォルトの日 付/時刻のパターン「YYYYMMDDHHMISS999」が 想定されます。

*p* に、計算タイプ (必須) と文字タイプを保持している必要があります。 そうでないと、文字に変換されます。

**w**

整数に変換できる式 (1950 など) を指定します。 負の場合、コードを実行するときに年の値から減算されるようにオフセットを指定します。 省略される場合は、*w* は、WINDOW コンパイル時オプションに指定された値に デフォルトで設定されます。

許可されるパターンを 382 ページの表 65 にリストしてあります。 リリアン形式の説明については、380 ページの『日付/時刻組み込み関数』を参照してください。

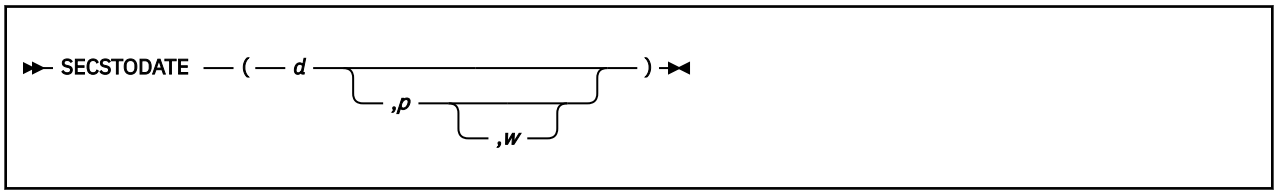
### 例

```
   dcl Dayname (7) char(9) var
static nonasgn init( 'Sunday',
                                                 'Monday',
                                                 'Tuesday',
                                                 'Wednesday',
                                                 'Thursday',
                                                 'Friday',
                                                 'Saturday');
   dcl Jul4_1776_Secs float bin(53);
   dcl Age_Tot_Secs pic 'Z,ZZZ,ZZZ,ZZZ,ZZ9';

   Jul4_1776_Secs = secs('17760704','YYYYMMDD');     /* seconds */
```

```
Age_Tot_Secs   = secs() - Jul4_1776_Secs;   /* seconds since */
display ('USA became independent on ' ||
          dayname(weekday(secstodays(Jul4_1776_Secs))) ||
         ', July 4, 1776 and at this very moment it has been ' ||
         Age_Tot_Secs, ||     ' seconds.');
```

## SECSTODATE

SECSTODATE は、*d* 秒 (リリアン形式に基づく) に対応する *p* で指定された 日付/時刻パターンを含む不変文字ストリングを戻します。




***d***
秒数 (リリアン形式) *d* は、計算タイプでなければなりません。また、必要な場合には、FLOAT BIN(53) に変換されます。

***p***
サポートされている日付/時刻パターンの 1 つ。 省略される場合、*p* は、デフォルトの日付/時刻パターン「YYYYMMDDHHMISS999」(DATETIME で 戻されるデフォルトのフォーマット) が想定されます。

***W***
整数に変換できる式 (1950 など) を指定します。 負の場合、コードを実行するときに年の値から減算されるようにオフセットを指定します。 省略される場合は、*w* は、WINDOW コンパイル時オプションに指定された値に デフォルトで設定されます。

許可されるパターンを 382 ページの表 65 にリストしてあります。 リリアン形式の説明については、380 ページの『日付/時刻組み込み関数』 を参照してください。

## SECSTODAYS

SECSTODAYS は、秒数 *x* を日数に変換した数を表す FIXED BINARY(31 ,0) 値を戻します。1 日に達しない秒は無視します。

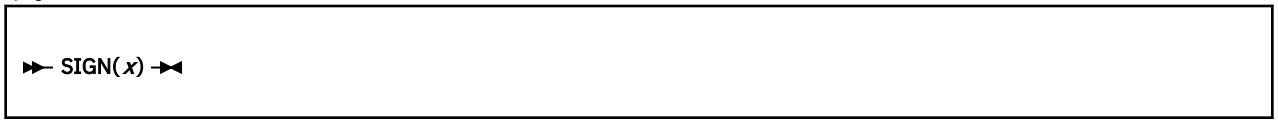
**x**
式。 *x* は、計算タイプ (必須) を保持し、FLOAT BINARY(53) である必要が あります。 そうでない場合は、<synph><var>x</var></synph> は、FLOAT BINARY(53) に変換されます。

SECSTODAYS(x) は、*x*/(24*60*60) と同じです。

例えば、534 ページの『SECS』 を参照してください。

## SIGN

SIGN は、*x* が正、ゼロ、または負であるかどうかを示す スケールのない REAL FIXED BINARY 値を戻します。




**x**
実数式。

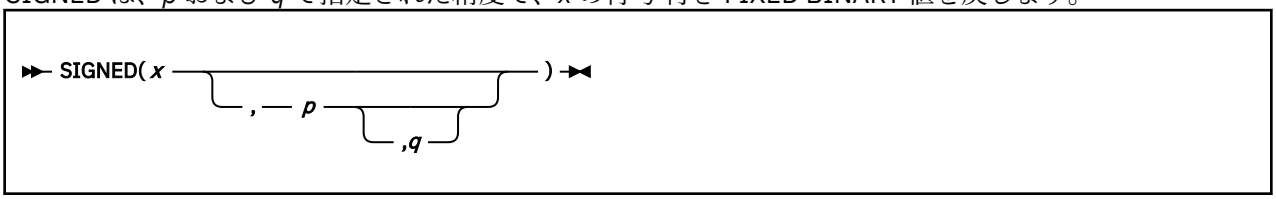
戻り値は、次の式で指定されます。

| x の値 | 返される値 |
|---|---|
| x > 0 | +1 |
| x = 0 | 0 |
| x < 0 | -1 |

BIFPREC コンパイラー・オプションによって、戻される結果の精度が決まります。

## SIGNED

SIGNED は、*p* および *q* で指定された精度で、*x* の符号 付き FIXED BINARY 値を戻します。




**x**
式。

**p**
演算を通して維持する桁数を指定する制限付き式。

**q**
結果の位取り係数を指定する制限付き式。 固定小数点の結果の場合は、*p* が指定され、*q* が省略され ると、位取り係数ゼロがデフォルトになります。

## SIN

SIN は、*x* のサインの近似値である浮動小数点値を返します。 戻り値は、*x* の基数、モード、および精度になります。

**x**
ラジアン値の式。

## SIND

SIND は、*x* のサインの近似値である実数の浮動小数点値を戻します。 戻り値は、*x* の基数と精度になります。

**x**
度で示される値を持つ実数式。

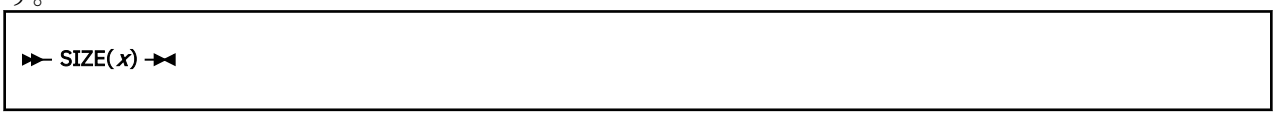
## SINH

SINH は、*x* の双曲線正弦の近似を表す浮動小数点の値を返します。 戻り値は、*x* の基数、モード、および精度になります。

**x**
ラジアン値の式。

# SIZE

SIZE は、変数 *x* に割り振られた実装環境定義ストレージ (バイト単位) を提供する FIXED BIN 値を返します。




**x**
任意のデータ・タイプ、データ編成、位置合わせ、およびストレージ・クラスの変数。 ただし、下のリストにあるものは除きます。

- 位置合わせされていない固定長ビット・ストリングである BASED 変数、DEFINED 変数、 パラメーター変数、添え字付き変数、構造変数、または共用体基底付き変数。
- 最初または最後の基本エレメントが位置合わせされていない固定長の ビット・ストリングである小構造体または共用体 (ただし、その小構造体を含んでいる 大構造体または共用体の最初または最後のエレメントになっている場合を除く)。
- BASED 属性、DEFINED 属性、またはパラメーター属性を持つ大構造体か共用体、またはパラメーターである大構造体か 共用体で、位置合わせされていない固定長のビット・ストリングを最初または最後のエレメントとするもの。
- 連結ストレージにない変数。

SIZE(x) によって戻される値は、次の状況で伝送される最大バイト数です。

```
declare F file record input
        environment(scalarvarying);
read file(F) into(x);
```

- *x* が可変長ストリングである場合、返される値には、ストリングの長さ接頭部、およびストリング最大長のバイト数が含まれます。
- *x* が区域である場合、返される値には、区域制御バイト、および区域の最大サイズが含まれます。
- *x* が区域または可変長ストリングを含む集合の場合は、戻り値に は、区域制御バイト、区域の最大サイズ、ストリングの長さの接頭辞、およびストリング の最大長でのバイト数が含まれます。

変数が割り振られていない場合、SIZE 組み込み関数を調節可能エクステントとともに、BASED 変数上で使用してはなりません。

CMPAT(V3) コンパイラー・オプションでは、SIZE は FIXED BIN(63) 値を返します。 その他のすべての CMPAT オプションでは、FIXED BIN(31) 値を返します。

割り振られたバイト数を無視して、変数が現在必要とするバイト数 を取得するには、CURRENTSIZE 組み込み関数を使用します。

*x* が BASED であり、REFER が使用されている場合に、以下の条件下では、コンパイラーは SIZE(*x*) のインライン・コードを生成します。

- *x* 内の、NONVARYING 属性および BIT 属性を持つエレメントは ALIGNED 属性を持っている。
- *x* 内のその他のすべてのエレメントは UNALIGNED 属性を持っている。

**例**

```
dcl Scids   char(17)         init('See you at SCIDS!') static;
dcl Vscids  char(20) varying init('See you at SCIDS!') static;
dcl Stg fixed bin(31);

Stg = storage   (Scids);           /* 17 bytes */
Stg = currentsize (Scids);         /* 17 bytes */
Stg = size (Vscids);               /* 22 bytes */
Stg = currentsize (Vscids);        /* 19 bytes */
Stg = size (Stg);                  /* 4  bytes */
Stg = currentsize (Stg);           /* 4  bytes */
```

**関連情報**

423 ページの『CURRENTSIZE』

CURRENTSIZE は、*x* によって要求される、実装環境定義ストレージを提供する FIXED BIN 値 (バイト単位) を返します。

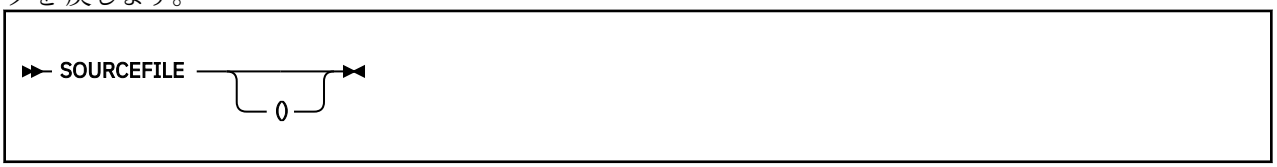
## SMFTOJULIAN

SMFTOJULIAN は、ユリウス形式 YYYYDDD の日付の入った CHAR(7) 値を戻します。

**d**
SMF 形式の日付の入った CHAR(4) 変数。

## SOURCEFILE

SOURCEFILE は、この関数が呼び出されるステートメントを含むファイルの名前をとる不変文字ストリングを 戻します。




SOURCEFILE 組み込み関数は、制限付き式に使用できます。

戻されるストリングは、システムによって異なります。これらのストリングは、トレースおよびデバッグの 目的にのみ使用する必要があります。

## SOURCELINE

SOURCELINE は、ステートメントが含まれているこのファイル内で、 この関数が呼び出されるステートメントの行番号である FIXED BINARY(31 ,0) 値を返します。 ステートメントが、いくつかのソース行に拡張している場合は、ステートメン トが開始する行の番号が戻されます。

SOURCELINE 組み込み関数は、制限付き式に使用できます。

## SQRT

SQRT は、*x* の正の平方根の近似を浮動小数点の値で返します。 戻り値は、*x* の基数、モード、および精度になります。

**x**
式。 *x* が実数の場合は、x はゼロより小さくてはいけません。

## SQRTF

SQRTF は SQRT と同様ですが、いくつかの違いがあります。

SQRTF と SQRT の違い:

- SQRTF は、ハードウェア・アーキテクチャーによって処理可能ならば、 結果をインラインで計算する。
- 引数は、実数でなければならない。

- 無効な引数は、ハードウェア例外を生成する。
- 結果の正確度は、ハードウェアによって設定される。

SQRTF 組み込み関数は DFP ではサポートされていない。

定義および構文については、538 ページの『SQRT』を参照してください。

## SQUEEZE

SQUEEZE は、オプションで指定された開始位置を使用して、1 つの文字のすべての複数オカレンスを 1 つのオカレンスに削減するストリングを返します。

**x**
ストリング式。 *x* は、*y* によって定義された文字のすべての複数オカレンスが 1 つのオカレンスに削減されるストリングを指定します。 *x* は、CHARACTER 属性を持っている必要があります。

**y**
CHARACTER(1) NONVARYING タイプを持つ必要がある式。

**n**
*y* の最初のオカレンスの検索を開始する *x* 内の位置を指定する式。 *n* は計算タイプでなければならず、タイプ *size_t* に変換されます。 *n* のデフォルトは 1 です。

- *n* < 1 の場合、デフォルト値 1 が使用されます。
- *n* > length(*x*) の場合、完全ストリング *x* が返されます。

**例**

```
dcl s1 char value( ' abc  :  def   gh  ' );
dcl s  char(20);

s = squeeze(s1, ' ', 1);
      /* ' abc : def gh ' */
s = squeeze(s1, ' ', index(s1,':'));
       /* ' abc  : def gh ' */
```

## STACKADDR

STACKADDR は、それが呼び出されたプロシージャー (または BEGIN ブロック) の動的保存域 (DSA) のアドレスを戻します。

STACKADDR 組み込み関数がストレージの変更に使用されると、予測不能な結果が生じることがあります。

## STCKETODATE

STCKETODATE は、(PLISTCKE により設定された) STCKE 値に対応する日付/時刻値を含む文字ストリングを返します。

**x**
STCKE 値を保持する CHAR(16) 値。

**p**
サポートされている日付/時刻パターンの 1 つを指定します。 *p* が省略されている場合は、TIMESTAMP パターン (つまり、「YYYY-MM-DD-HH.MI.SS.999999」) が想定されます。

許可されるパターンを 382 ページの表 65 にリストしてあります。 リリアン形式の説明については、380 ページの『日付/時刻組み込み関数』 を参照してください。

## STCKTODATE

STCKTODATE は、(PLISTCK により設定された) STCK 値に対応する日付/時刻値を含む文字ストリングを返します。

**x**
STCK 値を保持する UNSIGNED FIXED BIN(64) 値。

**p**
サポートされている日付/時刻パターンの 1 つを指定します。 *p* が省略されている場合は、TIMESTAMP パターン (つまり、「YYYY-MM-DD-HH.MI.SS.999999」) が想定されます。

許可されるパターンを 382 ページの表 65 にリストしてあります。 リリアン形式の説明については、380 ページの『日付/時刻組み込み関数』 を参照してください。

## STORAGE

STORAGE は、SIZE の同義語です。

**省略形**: STG

**注 :** USAGE(HEX(STG) は、USAGE(HEX(SIZE) の同義語として受け入れられます。

**関連情報**
537 ページの『SIZE』
SIZE は、変数 *x* に割り振られた実装環境定義ストレージ (バイト単位) を提供する FIXED BIN 値を返します。

## STRING

STRING は、*x* のすべてのエレメントを連結したストリングを戻します。

**x**
集合体参照またはエレメント参照。

STRING には、次の制限があります。

- 共用体を含む共用体または構造体には適用できない。
- スカラーに適用した場合は、スカラーはビット・ストリング、文字ストリング、ピクチャー文字ストリング、ピクチャー数値ストリング、漢字ストリング、uchar ストリング、またはワイド文字ストリングでなければなりません。
- 構造体に適用する場合は、構造体に埋め込みバイトがなく、構造体のエレメントは次の どちらかでなければなりません。

  - すべて位置合わせされていないビット・ストリング。
  - すべて文字ストリングで、それぞれ文字ストリング、ピクチャー付き列、またはピクチャー 付き数値ストリングのどれか。
  - すべて漢字ストリング。
  - すべての uchar ストリング
  - すべてワイド文字ストリング。
- 配列に適用する場合は、配列内のすべてのエレメントは、以前に説明した制限 を受けます。

戻されるストリングのタイプは、以下の例外 がありますが、基本エレメントの 1 つと同じタイプになります。

- 基本エレメントのいずれかが PICTURE である場合、戻されるタイプは CHARACTER タイプになります。
- 基本エレメントのいずれかが GRAPHIC タイプである場合、CHARACTER である必要がある STRINGOFGRAPHIC コンパイラー・オプションが指定されていなければ、戻されるタイプは GRAPHIC になります。

有効な STRING ターゲットを次に示します。

```
dcl
  1 A,
    2 B  bit(8),
    2 C  bit(2),
    2 D  bit(8);

dcl
  1 W,
    2 X  char(2),
    2 Y  pic'aa',
    2 Z  char(6);

dcl
  1 W,
    2 X  char(2),
    2 Y  pic'99',
    2 Z char(6);
```

無効な STRING ターゲットを次に示します。

```
dcl
  1 A,
    2 B  bit(8) aligned,
    2 C  bit(2),
    2 D  bit(8) aligned;
```

## STRING 疑似変数

STRING 疑似変数は、*x* がストリングのスカラーであるように、ストリングを *x* に割り当てます。 *x* の残りのストリングは、ブランクまたはゼロ・ビットで埋められるか、 可変長の場合は長さゼロが与えられます。

**x**
集合体参照またはエレメント参照。 *x* の各基本エレメントは、すべてビット・ストリングかすべて文字ストリングでなければなりません。

STRING 疑似変数は、コンテキスト以外では使用してはいけません。

また、疑似変数は、STRING 組み込み関数に制限されます。 制限の詳細については、540 ページの『STRING』 を参照してください。

## SUBSTR

SUBSTR は、サブストリングを戻します。このサブストリングは、*x* の *y* および *z* によって指定されます。

**x**
ストリング式。 サブストリングが抜き出されるストリングを指定する。 *x* がストリングでない場合は、文字に変換されます。

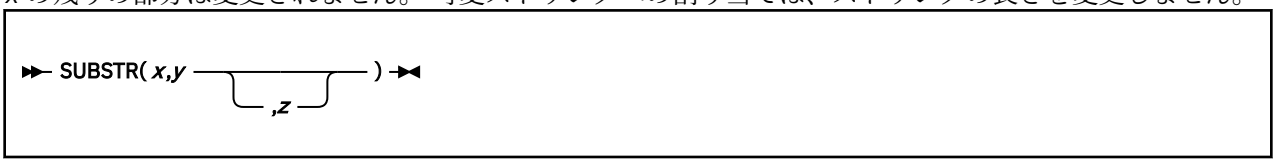
**y**
FIXED BINARY(31 ,0) に変換される式。 *y* は、*x* の中のサブストリングの開始位置を指定する。

**z**
FIXED BINARY(31 ,0) に変換される式。 *z* は、*x* 内のサブストリングの長さを指定します。 *z* がゼロの場合は、ヌル・ストリングが戻されます。 *z* が省略される場合は、戻されるサブストリングは *x* の位置 *y* から *x* の終わりまでです。

STRINGRANGE 条件が発生するのは、*z* が負の場合、または *y* および *z* の値が、*x* の現在の長さ内に完全に収まらないようなサブストリングである場合です。 *y* = LENGTH(*x*)+1 および *z* = 0 の場合は発生しません。 SUBSTR 組み込み関数の例については、 532 ページの『SEARCH』を参照してください。

## SUBSTR 疑似変数

SUBSTR 疑似変数は、*x* の *y* および *z* によって指定されたサブストリングにストリング値を割り当てます。 *x* の残りの部分は変更されません。 可変ストリングへの割り当ては、ストリングの長さを変更しません。




**x**
ストリング参照。 *x* は、数字であってはいけません。

**y**
式。 *x* のサブストリングの開始位置を指定する FIXED BINARY 値に 変換可能な *y* の式。

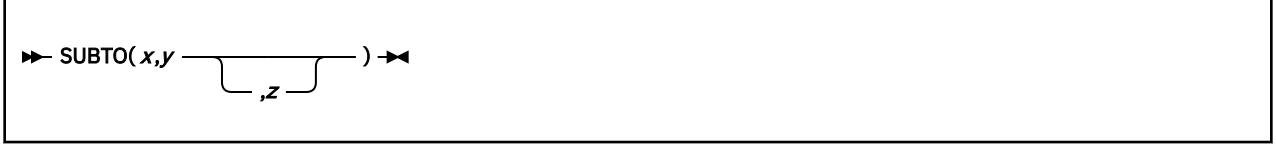
**z**
式。 *z* は、*x* 内のサブストリングの長さを指定します。 これは、実数固定小数点 2 進数値に変換することができます。 *z* がゼロの場合は、ヌル・ストリングが戻されます。 *z* が省略される場合は、戻されるサブストリングは *x* の位置 *y* から *x* の終わりまでです。

*y* および *z* は、*x* が配列である場合にのみ、配 列であることが可能です。

STRINGRANGE 条件が無効になっている場合は、SUBSTR(*x*,1,*z*) に対する割り当てによって、長さ *z* のストリングが、ADDRDATA(*x*) によって指定されたアドレスに割り当てられます。 *z* > MAXLENGTH(*x*) の場合、プログラムは無効であり、*x* の末尾を超えたデータは上書きされます。 例えば、*x* が CHAR(10) の場合に *z* の値が 15 だとすると、10 個のブランクが *x* まで書き込まれ、*x* を超えたストレージ上の 5 個のブランクは上書きされます。

## SUBTO

SUBTO は、*x* の、*y* および *z* で指定されたサブストリングを返します。




**x**
ストリング式。 サブストリングが抜き出されるストリングを指定する。 *x* がストリングでない場合は、文字に変換されます。

**y**
FIXED BINARY(31 ,0) に変換される式。 *y* は、*x* の中のサブストリングの開始位置を指定する。

**z**
FIXED BINARY(31,0) に変換される式。 *z* は、*x* 内のサブストリングの終了位置を指定します。 *z* = *y*-1 の場合、ヌル・ストリングが戻されます。 *z* が省略される場合は、戻されるサブストリングは *x* の位置 *y* から *x* の終わりまでです。

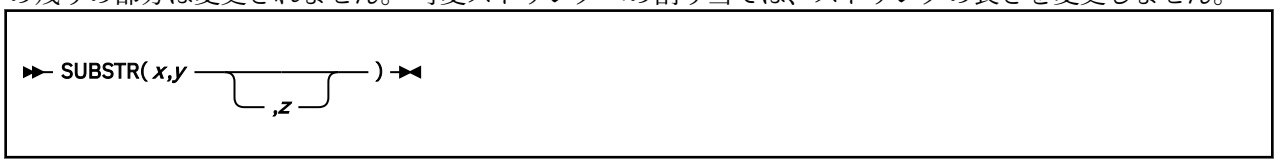
SUBTO(*x,y*) は SUBSTR(*x,y*) と同等で、SUBTO(*x,y,z*) は SUBSTR(*x,y,z-y+1*) と同等です。

STRINGRANGE 条件は、同等の SUBSTR 参照に対して STRINGRANGE 条件が発生する場合にのみ、SUBTO 参照に対して発生します。 特に、k = length(*x*) の場合、以下の 5 つの条件のいずれかで SUBTO(*x,y,z*) に対して STRINGRANGE が発生することを意味します。

```
y < 1
y > k+1
y = k+1 then unless z = k
z-y+1 < 0
z > k
```

## SUBTO 疑似変数

SUBTO 疑似変数は、*x* の *y* および *z* によって指定されたサブストリングにストリング値を割り当てます。 *x* の残りの部分は変更されません。 可変ストリングへの割り当ては、ストリングの長さを変更しません。




**x**
ストリング参照。 *x* は、数字であってはいけません。

**y**
*x* 内のサブストリングの開始位置を指定する FIXED BINARY 値に変換できる式。

**z**

FIXED BINARY(31,0) に変換される式。 *z* は、x 内のサブストリングの終了位置を指定します。 *z* = *y*-1 の場合、ヌル・ストリングが戻されます。 *z* を省略すると、割り当てられるサブストリングの位置は、*x* の最後まで *x* 内の y です。

SUBTO(*x,y*) は SUBSTR(*x,y*) と同等で、SUBTO(*x,y,z*) は SUBSTR(*x,y,z-y+1*) と同等です。

STRINGRANGE 条件が無効である場合、SUBTO(*x,y,z*) への割り当てでは、開始位置が y で終了位置が *z* のストリングが割り当てられます。 *z* > MAXLENGTH(*x*) である場合、プログラムは無効であり、*x* の終わりを超えるデータは上書きされます。

## SUBTRACT

SUBTRACT は、ADD(*x,-y,p,q*) と同等です。

引数の詳細については、398 ページの『追加』の引数の説明を参照してください。

## SUCC

SUCC は、*x* より大きい表現可能な最小の数値である浮動小数点値を返します。 これは、 *x* の基数、モード、および精度です。 そのような数値がない場合は、OVERFLOW 条件が発生します。

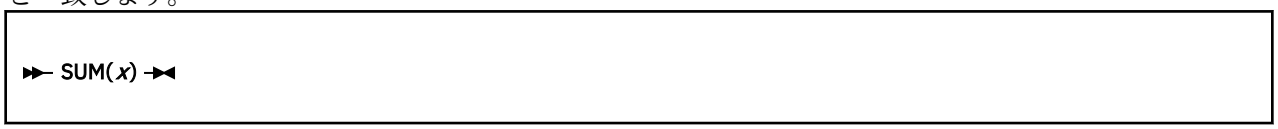
**x**
REAL FLOAT 式。

SUCC は、次の関係を満たしています。

```
pred(succ(x)) = x
succ(pred(x)) = x
succ(x)       = -pred(-x)
succ(0d0)     = tiny(0d0)
```

## SUM

SUM は、*x* のすべてのエレメントの和を戻します。 結果の基数、モード、およびスケールは、*x* のそれらと一致します。




**x**
配列式。 *x* のエレメントがストリングの場合は、それらのエレメントは固定小数点の整数値に変 換されます。

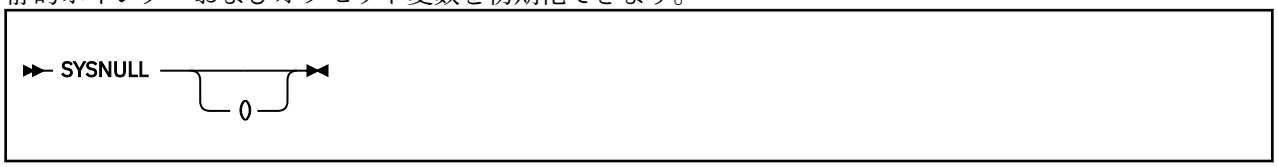
*x* のエレメントが固定小数点の場合は、結果の精度は (*N,q*) です。 N は、許可される桁の最大数で、*q* は *x* の位取り係数 です。

*x* のエレメントが浮動小数点の場合は、結果の精度は *x* と一致し ます。

## SYSNULL

SYSNULL は、システム・ヌル・ポインターの値を戻します。

SYSNULL をハンドルに割り当て、SYSNULL とハンドルを比較することができます。 SYSNULL を使用して静的ポインターおよびオフセット変数を初期化できます。




**注 :** NULL および SYSNULL を比較すると等しい場合があります。ただし、これらが等しいということに基づいてコーディングしないでください。

489 ページの『NULL』 も参照してください。

## SYSTEM

SYSTEM は、*x* に含まれているコマンドで呼び出されるときにコマンド・プロセッサーからの戻り値である FIXED BIN(31 ,0) 値を返します。

**x**
計算タイプと文字タイプが必要です。 そうでない場合は、*x* は文字に変換されます。

## TALLY

TALLY は、ストリング *y* がストリング *x* 内に現れる回数を示す FIXED BINARY(31 ,0) 結果を返します。

*x* 内に *y* が表示されない場合は、値 0 が戻されます。

**x および y**
ストリング式。

*x* と *y* の両方が計算タイプを保持していなければならず、 文字、ビット、グラフィック、uchar、またはワイド文字のタイプでなければなりません。

*x* または *y* はヌル・ストリングで、結果はゼロです。

**例**

```
TALLY ('We''ve got the Power!', 'power');      /* returns 0 */
TALLY ('We''ve got the Power!', 'Power');      /* returns 1 */
TALLY ('We''ve got the Power!', ' ');          /* returns 3 */
TALLY ('We''ve got the Power!', 'e');          /* returns 4 */
TALLY ('1001'B, '1'B);                         /* returns 2 */
```

## TAN

TAN は、*x* のタンジェントの近似値である浮動小数点値を返します。 戻り値は、*x* の基数、モード、および精度になります。

**x**
ラジアン値の式。

## TAND

TAND は、*x* の正接の近似を実数の浮動小数点の値で返します。 戻り値は、*x* の基数と精度になります。

**x**
度で示される値を持つ実数式。

## TANH

TANH は、*x* の双曲線正接の近似を浮動小数点の値で返します。 戻り値は、*x* の基数、モード、および精度になります。

**x**
ラジアン値の式。

## THREADID

THREADID (スレッド ID の短縮形) は、POINTER 値を戻します。この値は、接続されたスレッドのオペレーティング・システム・スレッド ID のアドレスです。

**x**
タスク参照。 *x* の値は、ATTACH ステートメントの THREAD オプションですでに設定されているはずです。

この組み込み関数によって戻される値を使用して、DosSetPriority などのシステム関数 (Windows の場合)、または posix 関数 (z/OS の場合) を呼び出すことができます。

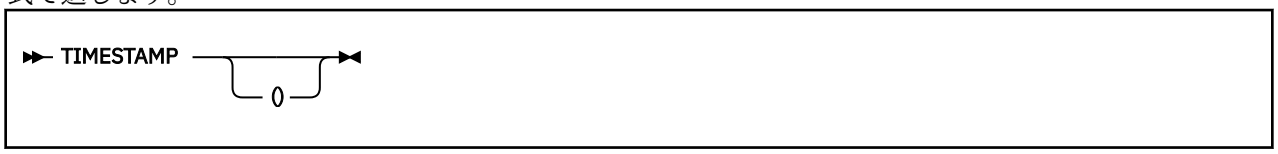
現在実行中のスレッドのシステム・スレッド ID を取得するには、そのスレッドが実行されているプラットフォームに適した関数を呼び出す必要があります。 そのため、Windows では GetCurrentThreadId、z/OS では pthread_self を呼び出してください。

## TIME

TIME は、*HHMISS*999 のフォーマットで、文字ストリング・タイム・スタンプを戻します。

## TIMESTAMP

TIMESTAMP は、現在日時を提供する CHAR(26) 文字ストリングを *YYYY-MM-DD-HH.MI.SS*.999999 の形式で返します。




## TINY

TINY は、*x* が取ることができる正の最小値を浮動小数点の値で戻します。 戻り値は、*x* の基数、モード、および精度になります。

**x**
REAL FLOAT 式。

TINY(*x*) は定数で、制限付き式に使用できます。

## TRANSLATE

TRANSLATE は、*x* と同じ長さの文字ストリングを戻します。

**x**
変換できる文字があるかどうかを検索される文字式。

**y**
文字の変換値を含む文字式。

**z**
変換される文字を含む文字式。 *z* が省略される場合は、デフォルトは collate() です。

TRANSLATE は、*x* の各文字を次のように処理します。

*x* の中にある文字が *z* に検出される場合は、*y* にあ る *z* に対応する文字が結果にコピーされます。それ以外の場合は、*x* の文字が結果にコピーされます。 *z* に複数の同一文字が入っている場合は、左端のオカレンスが使われます。

*y* は、*z* の長さと一致するように、右がブランクで埋められるか、切り捨てられます。

すべての算術引数またはビット引数は、文字に変換されます。

TRANSLATE は UCHAR データをサポートします。 ただし、*x* が UCHAR タイプを保持している場合、*z* を省略することはできません。

TRANSLATE は、GRAPHIC または WIDECHAR のデータをサポートしません。

TRANSLATE は、2 番目および 3 番目の引数がリテラル、VALUE 属性を使用して宣言された名前付き定数、または制限付き式のいずれかである場合に、最も機能を発揮します。

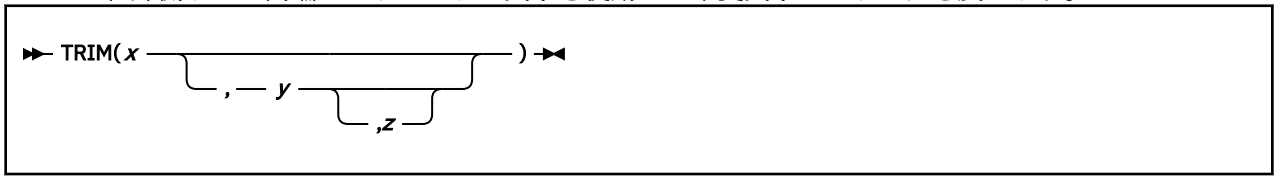
### 例

```
dcl source char value("Ein Raetsel gibt es nicht.");
dcl target char(length(source));
dcl (to   value ('ABCDEFGHIJKLMNOPQRSTUVWXYZ'),
     from value ('abcdefghijklmnopqrstuvwxyz')) char;

target = translate(source, to, from);
         /* "EIN RAETSEL GIBT ES NICHT." */
```

上記の例における TRANSLATE 組み込み関数と同じ目的で UPPERCASE を使用することもできます。 しかし、UPPERCASE 組み込み関数が規格英字のみを変換するのに対して、TRANSLATE は他の文字への変換に使用することができます。 例えば、「Raetsel」のスペルが ä-umlaut である場合、TRANSLATE は、ä-umlaut と Ä-umlaut をそれぞれ変換前のストリングと変換後のストリングに追加されていれば、ä-umlaut を Ä-umlaut に変換することができます。

## TRIM

TRIM は、片側または両端がトリムされた文字を使用した不変文字ストリングを戻します。




**x、y、および z**
式。

それぞれが、計算タイプおよび CHARACTER タイプまたは UCHAR タイプを保持している必要があります。 そうでない場合は、変換されます。

*x* は、*y* によって定義された文字が左からトリムされ、*z* に よって定義された文字が右からトリムされたストリングです。

*z* が省略される場合は、1 つのブランクを含む CHARACTER(1) NONVARYING ストリングをデフォルトにします。

*y* と *z* が両方とも省略される場合は、両方とも 1 つのブランクを含む CHAR(1) NONVARYING ストリングをデフォルトにします。

### 例

次の例で、TRIM 関数は以下のものを除去します。

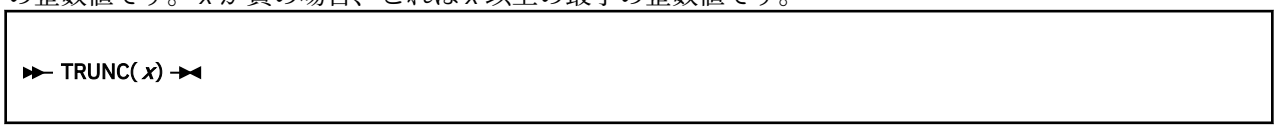
- ストリングの左側から、すべてのブランク。

- ストリングの右側から、すべてのブランクとすべてのアスタリスク。

```
dcl Source char value(" *** PL/I's got the Power!  ***  ");
dcl Target char(length(Source)) varying;

Target = trim(Source, ' ', '* ');
             /* "*** PL/I's got the Power!" */
```

# TRUNC

TRUNC は、*x* の切り捨てられた値である整数値を返します。 *x* が正または 0 の場合、これは *x* 以下の最大の整数値です。 *x* が負の場合、これは *x* 以上の最小の整数値です。




**x**
実数式。

結果の基数、モード、スケール、および精度は、*x* のそれらと一致します。 *x* が精度 (*p*,*q*) を使用した固定小数点のときは、結果 の精度は次の式で与えられます。

```
(min(N,max(p-q+1,1)),0)
```

この場合 N は、使用できる最大の桁数です。

# TYPE

TYPE は、ハンドル *x* によって位置付けされるタイプ付き構造体または共用体を戻します。

**x**
ハンドル。

TYPE(*x*) は、タイプ付き構造体 (または共用体) *x* の参照を解除し ます。 TYPE 組み込み関数の例については、548 ページの『TYPE 疑似変数』 を参照してください。

# TYPE 疑似変数

TYPE 疑似変数は、タイプ付き構造体または共用体をハンドル *x* によって位置付けられたストレージに割り当てます。

**x**
ハンドル。

定義済み構造体 *T* が与えられると、次の割り当てが有効です。

```
dcl P1 handle T;
dcl P2 handle T;
dcl D1 type T;
dcl D2 type T;

D1 = type(P2);          /* Assigns the storage located by P2 to D1 */
type(P1) = type(P2);
type(P1) = D2;          /* Assigns D2 to the storage located by P1 */
```

## UHIGH

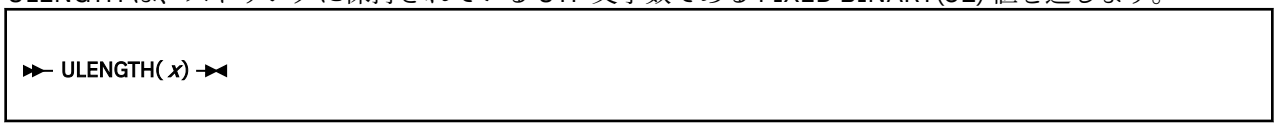
UHIGH は、各 UTF-8 データ項目が最も大きい UCHAR 値 ('F48FBFBF'ux) を持つ、長さ *x* の UCHAR ストリングを返します。

**x**
式。 *x* は、UCHAR タイプを保持する必要があります。

BYTELENGTH(UHIGH(*x*)) によって返される値は 4**x* に等しくなります。

## ULENGTH

ULENGTH は、ストリングに保持されている UTF 文字数である FIXED BINARY(31) 値を返します。




**x**
式。 *x* に、CHARACTER タイプまたは WIDECHAR タイプを保持する必要があります。

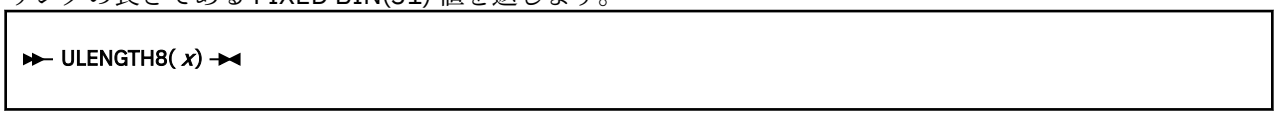
*x* が CHARACTER タイプである場合、ストリングには有効な UTF-8 データが入っている必要があります。 それ以外の場合、プログラムはエラーになります。

*x* が WIDECHAR タイプである場合、ストリングには有効な UTF-16 データが入っている必要があります。 それ以外の場合、プログラムはエラーになります。

ULENGTH は、それぞれ CHAR または WIDECHAR 引数に保持された UTF-8 または UTF-16 文字の数を返します。 ストリングが正規化されていない場合には文字数は返されません。 例えば、UTF-8 では、小文字の a ウムラウトは、ストリング 'c3_a4'x で正規化 (標準) 形式で表現するか、ストリング '61_cc_88'x として非正規化 (結合) 形式で表現できますが、ULENGTH は、ストリング 'c3_a4'x に対しては 1 を返し、ストリング '61_cc_88'x に対しては 2 を返します。

## ULENGTH8

ULENGTH8 は、ストリングに保持されている UTF 文字が UTF-8 に変換されている場合に必要な CHAR ストリングの長さである FIXED BIN(31) 値を返します。




**x**
式。 *x* に、CHARACTER タイプまたは WIDECHAR タイプを保持する必要があります。

*x* が CHARACTER タイプである場合、ULENGTH8 は LENGTH と同じになり、ストリングは有効な UTF-8 データかどうか、検査されません。

*x* が WIDECHAR タイプである場合、ストリングには有効な UTF-16 データが入っている必要があります。 また、ULENGTH8 は、*x* が UTF-16 から UTF-8 に変換されている場合に生じる CHAR ストリングの長さを返します。 有効な UTF-16 データがストリングに含まれていない場合、プログラムはエラーになります。

例えば、*x* が WIDECHAR ストリング '004B_00E4_0073_0065'wx である場合、ULENGTH8(*x*) は 5 を返します。

## ULENGTH16

ULENGTH16 は、ストリングに保持されている UTF 文字が UTF-16 に変換されている場合に必要な WIDECHAR ストリングの長さである FIXED BINARY(31) 値を返します。

**x**
式。 *x* に、CHARACTER タイプまたは WIDECHAR タイプを保持する必要があります。

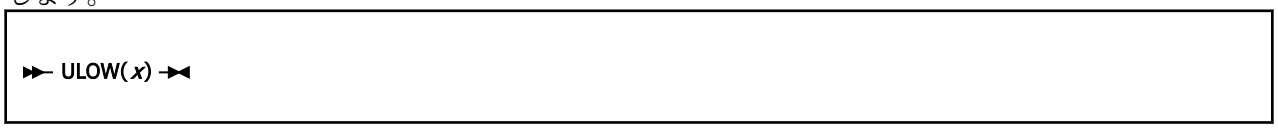
*x* が CHAR タイプである場合、ストリングには有効な UTF-8 データが入っている必要があります。 また、ULENGTH16 は、*x* が UTF-8 から UTF-16 に変換されている場合に生じる WIDECHAR ストリングの長さを返します。 有効な UTF-8 データがストリングに含まれていない場合、プログラムはエラーになります。

*x* が WIDECHAR タイプである場合、ULENGTH16 は LENGTH と同じになり、ストリングは有効な UTF-16 データかどうか、検査されません。

例えば、*x* が CHARACTER ストリング `'4b_c3_a4_73_65'x` である場合、ULENGTH16(*x*) は 4 を返します。

## ULOW

ULOW は、各 UTF-8 データ項目が最も小さい UCHAR 値 ('00'ux) を持つ、長さ *x* の UCHAR ストリングを返します。




**x**
式。 *x* は、UCHAR タイプを保持する必要があります。

BYTELENGTH(ULOW(*x*)) によって返される値は *x* に等しくなります。

## UNALLOCATED

UNALLOCATED は、指定されたポインター値が割り振り済みストレージの始まりであるかどうかを示す BIT(1) 値を返します。 この組み込み関数を使用するには、CHECK(STORAGE) コンパイル時オプションも指定しなければ なりません。

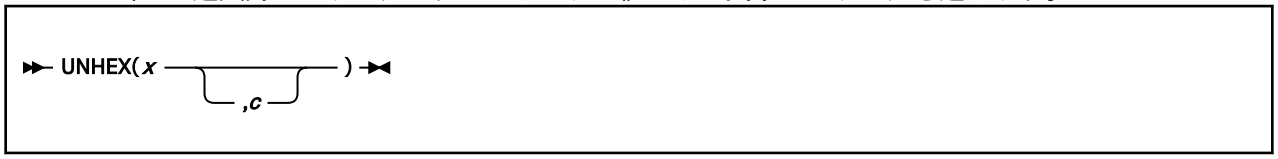
**p**
ポインターの式。

指定されたポインター値が、ALLOCATE ステートメントまたは ALLOCATE 組み込み関数によって取得したストレージの始まりでない 場合に、UNALLOCATED は BIT(1) 値 `'1'b` を返します。

なお、UNALLOCATED に渡されたポインターは、最も近いダブルワードに「切り捨て」られ、切り捨てられた値は、同様に切り捨てられた場合のすべての割り振り済みアドレスに対して比較されます。

## UNHEX

UNHEX は、16 進入力ストリングのデコードされた値である文字ストリングを返します。




**x**
式。CHARACTER タイプでなければなりません。

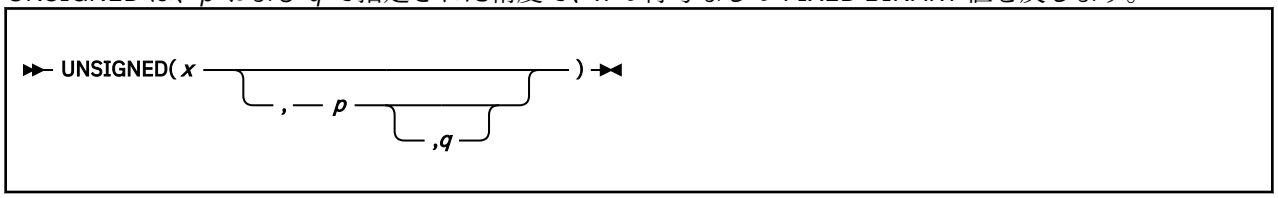
**c**
CHARACTER(1) NONVARYING タイプを持つ必要がある式。 指定した場合、この文字で *x* の文字が 8 文字ごとに区切られます。

UNHEX(*x*) は HEX(*y*) の逆であり、UNHEX(*x, c*) は HEX(*y, c*) の逆です。

*x* に 16 進以外の文字を含む場合は、CONVERSION 条件が発生します。

## UNSIGNED

UNSIGNED は、*p* および *q* で指定された精度で、*x* の符号なしの FIXED BINARY 値を戻します。




**x**
式。

**p**
整数。 演算を通して維持される桁数を指定します。

**q**
オプションの符号付き整数。 結果の位取り係数を指定します。 固定小数点の結果の場合は、*p* が指定され、*q* が省略され ると、位取り係数ゼロがデフォルトになります。

## UNSPEC

UNSPEC は、*x* の内部コードであるビット・ストリングを戻します。

**x**
スカラー、配列、構造体、または共用体の式。

UNSPEC 組み込み関数は、次の規則に従います。

- コンパイラー・オプション USAGE( UNSPEC(IBM)) の場合
  - 構造体参照と式の UNSPEC は使用できません。
  - 配列の UNSPEC 結果は、BIT の配列になります。
- コンパイラー・オプション USAGE( UNSPEC(ANS)) の場合
  - 集合体の場合、UNSPEC は埋め込みバイトまたはビットを 含まないものにだけ使用できます。
  - 結果は常に BIT スカラーです。 配列の UNSPEC 結果は、BIT の配列にはなりません。

**注 :** UNSPEC の使用により、プログラムの移植性に影響を与えることがあります。

戻されるビット・ストリングの長さは、551 ページの表 83 に示すように、 *x* の属性によって決まります。

表 *83. UNSPEC* によって戻されるビット・ストリングの長さ

| ビット・ストリング長 | *x* の属性 |
|---|---|
| 8 | SIGNED FIXED BINARY(p,q), 1 <= p <= 7<br>UNSIGNED FIXED BINARY(p,q), 1 <= p <= 8<br>ORDINAL SIGNED PRECISION(p), 1 <= p <= 7<br>ORDINAL UNSIGNED PRECISION(p), 1 <= p <= 8 |

表 *83. UNSPEC* によって戻されるビット・ストリングの長さ *(*続き*)*

| ビット・ストリング長 | *x* の属性 |
|---|---|
| 16 | SIGNED FIXED BINARY(p,q), 8 <= p <= 15<br>UNSIGNED FIXED BINARY(p,q), 9 <= p <= 16<br>ORDINAL SIGNED PRECISION(p), 8 <= p <= 15<br>ORDINAL UNSIGNED PRECISION(p), 9 <= p <= 16 |
| 32 | LP(32) での ENTRY LIMITED<br>SIGNED FIXED BINARY(p,q), 16 <= p <= 31<br>UNSIGNED FIXED BINARY(p,q), 17 <= p <= 32<br>ORDINAL SIGNED PRECISION(p), 16 <= p <= 31<br>ORDINAL UNSIGNED PRECISION(p), 17 <= p<= 32<br>FLOAT BINARY(p), 1 <= p <= 21<br>DFP でない場合は、FLOAT DECIMAL(p), 1 <= p <= 6<br>DFP である場合は、FLOAT DECIMAL(p), 1 <= p <= 7<br>OFFSETSIZE(4) での OFFSET<br>LP(32) での FILE 定数または変数<br>POINTER(32)<br>HANDLE(32) |
| 64 | LP(64) での ENTRY LIMITED<br>SIGNED FIXED BINARY(p), 31 < p<br>UNSIGNED FIXED BINARY(p), 32 < p<br>FLOAT BINARY(p), 21 < p < 53<br>DFP でない場合は、FLOAT DECIMAL(p), 7 <= p <= 16<br>DFP である場合は、FLOAT DECIMAL(p), 8 <= p <= 16<br>OFFSETSIZE(8) での OFFSET<br>LP(64) での FILE 定数または変数<br>LABEL 定数または変数<br>ENTRY 定数または変数<br>POINTER(64)<br>HANDLE (64) |
| 128 | FLOAT BINARY(p), 54 <= p<br>FLOAT DECIMAL(p), 17 <= p<br>TASK |
| n | BIT(n) |
| 8*n | CHARACTER(n)PICTURE (長さ *n* の 文字ストリング値付き) |
| 16*n | GRAPHIC(n)WIDECHAR(n) |
| 32*n | UCHAR (n) |
| 16+n | *n* が *x* の最大の長さの BIT(n) VARYING |
| 32+n | BIT(n) VARYING4 (*n* は *x* の最大長) |
| 16+(8*n) | *n* が *x* の最大の長さの CHARACTER(n) VARYING |
| 32+(8*n) | CHARACTER(n) VARYING4 (*n* は *x* の最大長) |
| 8+(8*n) | *n* が *x* の最大の長さの CHARACTER(n) VARYINGZ |
| 16+(16*n) | *n* が *x* の最大の長さの GRAPHIC(n) VARYING、*n* が *x* の最大の長さの WIDECHAR(n) VARYING |

表 *83. UNSPEC* によって戻されるビット・ストリングの長さ *(*続き*)*

| ビット・ストリング長 | *x* の属性 |
|---|---|
| 32+(16*n) | GRAPHIC(n) VARYING4 (*n* は *x* の最大長。WIDECHAR(n) VARYING4 (n は *x* の最大長) |
| 8+(16*n) | *n* が *x* の最大の長さの GRAPHIC(n) VARYINGZ、 *n* が *x* の最大の長さの WIDECHAR(n) VARYINGZ |
| 16+(32*n) | *n* が *x* の最大の長さの UCHAR(n) VARYING |
| 32+(32*n) | *n* が *x* の最大の長さの UCHAR(n) VARYING4 |
| 8+(32*n) | *n* が *x* の最大の長さの UCHAR(n) VARYINGZ |
| 8*(n+16) | LP(32) での AREA (n) |
| 8*(n+32) | LP(64) での AREA (n) |
| 8*FLOOR(n) | n = (p+2)/2 の FIXED DECIMAL (p,q) |

プログラム制御データの位置合わせおよびストレージ所要量は、サポートされるシス テムをまたがって変更することができます。

*x* が VARYING または VARYING4 のストリングの場合は、戻されるビット・ストリングにその長さの接頭部が組み込まれます。 *x* が区域の場合は、戻り値は制御情報を含みます。

## UNSPEC 疑似変数

UNSPEC 疑似変数は、直接、ビット値を *x* に割り当てます。 変換は行いません。

必要な場合には、ビット値は、*x* の長さに合うように、551 ページの表 83 に従って、右側に '0'B が埋め込まれます。

**x**
参照。

*x* が VARYING または VARYING4 のストリングの場合は、戻されるビット・ストリングにその長さの接頭部が組み込まれます。 *x* が区域の場合は、制御情報は、受信フィールドに含まれます。

疑似変数は、551 ページの『UNSPEC』 で説明されている UNSPEC 組み込み関数の規則に従います。

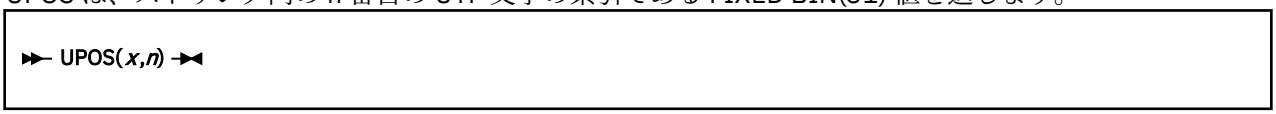
**注 :** UNSPEC の使用により、プログラムの移植性に影響を与えることがあります。

**例**

```
dcl 1 Str1 nonasgn static,
      2 * fixed bin(15) littleendian init(16),  /* '1000'X */
      2 * char init('33'x),
      2 * bit init('1'b),
      2 Ba(4) bit init('1'b, '0'b, '1'b, '0'b),
      2 B3 bit(3) init('111'b),
      2 * char(0);
dcl Bit_Str1 bit(size(Str1)*8);
dcl Bit_Ba   bit(dim(Ba)*length(Ba(1)));
dcl Bit_B3   bit(length(B3));

Bit_Ba   = unspec(Ba);   /* result is scalar '1010'B not an array */
Bit_B3   = unspec(B3);   /* '111'B                                 */
Bit_Str1 = unspec(Str1); /* '100033D7'B4 or
                            '100033'B4 || '11010111'B             */
```

# UPOS

UPOS は、ストリング内の n 番目の UTF 文字の索引である FIXED BIN(31) 値を返します。




**x**
CHARACTER または WIDECHAR タイプが必要な式。

**n**
計算タイプで、必要に応じて FIXED BIN(31) に変換される式。

*x* が CHARACTER タイプである場合、ストリングには有効な UTF-8 データが入っている必要があります。 それ以外の場合、プログラムはエラーになります。

*x* が WIDECHAR タイプである場合、ストリングには有効な UTF-16 データが入っている必要があります。 それ以外の場合、プログラムはエラーになります。

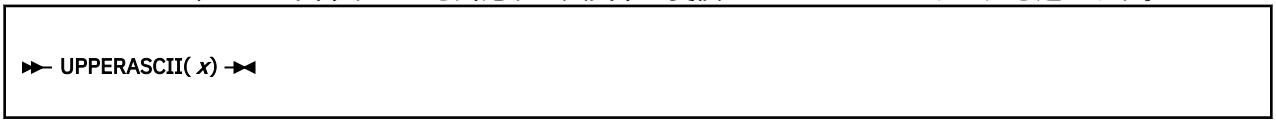
*n* が正ではない場合、または *n* が ULENGTH(*x*) より大きい場合は、ゼロが返されます。 それ以外の場合は、*x* が CHARACTER タイプの場合は、UPOS(*x,n*) は n 番目の UTF-8 文字が開始するバイトの位置を返し、*x* が WIDECHAR タイプの場合は、UPOS(*x,n*) は n 番目の UTF-16 文字が開始するワイド文字の位置を返します。

例えば、*x* が CHARACTER ストリング '4b_c3_a4_66_65_72'x である場合には、次のようになります。

- UPOS(*x*,1) は 1 を返します
- UPOS(*x*,2) は 2 を返します
- UPOS(*x*,3) は 4 を返します
- UPOS(*x*,4) は 5 を返します
- UPOS(*x*,5) は 6 を返します

# UPPERASCII

UPPERASCII は、ASCII 文字すべてを対応する大文字に変換した UCHAR ストリングを返します。




**x**
式。 *x* は、UCHAR タイプを保持する必要があります。

UPPERASCII(*x*) は、TRANSLATE(*x, 'A...Z', 'a...z'*) と同等です。

# UPPERCASE

UPPERCASE は、すべての文字を対応する大文字に変換した文字ストリングを返します。

**x**
式。 必要に応じ、*x* は文字に変換されます。

**c**
大文字に変換するコード・ページを指定する式。

UPPERCASE(*x*) は、TRANSLATE(*x, 'A...Z '、'a... z'*) と同等で、UPPERCASE(*x*, *c*) は TRANSLATE(*x, upperc, lowerc*) と同等です。 *upperc* と *lowerc* の値は、コード・ページ *c* の値によって決定されます。 UPPERCASE(*x*, *c*) を指定すると、英字「a...z」が「A...Z」に変換されるだけでなく、小文字の ä-umlaut('c0'x) のような文字も大文字の Ä-umlaut('4a'x) に変換されます。

例えば、Lower_01141 が次のように宣言されたとします。

```
dcl lower_01141 char
 value( (
              '8182838485868788'8991929394959697'x
           || '9899A2A3A4A5A6A7A8A9424445464748'x
           || '4951525354555657'586A708C8D8E9CC0'x
           || 'CBCDCECFD0DBDDDE'x
        ) );
```

また、Upper_01141 が次のように宣言されたとします。

```
dcl upper_01141 char
  value( (
              'C1C2C3C4C5C6C7C8C9D1D2D3D4D5D6D7'x
           || 'D8D9E2E3E4E5E6E7E8E9626465666768'x
           || '6971727374757677'78E080ACADAE9E4A'x
           || 'EBEDEEEF5AFBFDFE'x
        ) );
```

すると、UPPERCASE(*x*, 1141 ) は、TRANSLATE( *x*, Upper_01141, Lower_01141 ) と同じになります。

付録に、サポートされている *c* の値の *upperc* と *lowerc* の値がリストされています。 詳しくは、『615 ページの『付録 A 制限値』』を参照してください。

## UPPERLATIN1

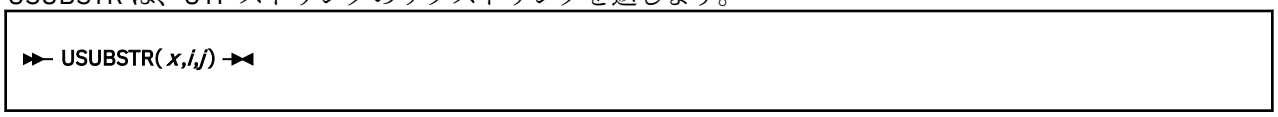
UPPERLATIN1 は、ASCII 文字および Latin-1 補足文字すべてを対応する大文字に変換した UCHAR ストリングを返します。

**x**
式。 x は、UCHAR タイプを保持する必要があります。

DIAERESIS(ÿ) および SHARP S(ß) の文字 Y は変更されません。

## USUBSTR

USUBSTR は、UTF ストリングのサブストリングを返します。




**x**
CHARACTER または WIDECHAR タイプが必要な式。

**i**
計算タイプで、必要に応じて FIXED BIN(31) に変換される式。

**j**
計算タイプで、必要に応じて FIXED BIN(31) に変換される式。

*x* が CHARACTER タイプである場合、ストリングには有効な UTF-8 データが入っている必要があります。 それ以外の場合、プログラムはエラーになります。

*x* が WIDECHAR タイプである場合、ストリングには有効な UTF-16 データが入っている必要があります。 それ以外の場合、プログラムはエラーになります。

以下の場合に、(STRINGRANGE 条件ではなく) ERROR 条件も発生します。

- *i* が 1 未満、または
- *j* がゼロ未満、または
- *i + j - 1* が ULENGTH(*x*) より大きい

*x* が CHARACTER タイプの場合は、USUBSTR(*x,i,j*) は、*i* 番目の UTF-8 文字で開始する *x* の *j* UTF-8 文字が入った CHARACTER ストリングを返します。

*x* が WIDECHAR タイプの場合は、USUBSTR(*x,i,j*) は、*i* 番目の UTF-16 文字で開始する *x* の *j* UTF-16 文字が入った WIDECHAR ストリングを返します。

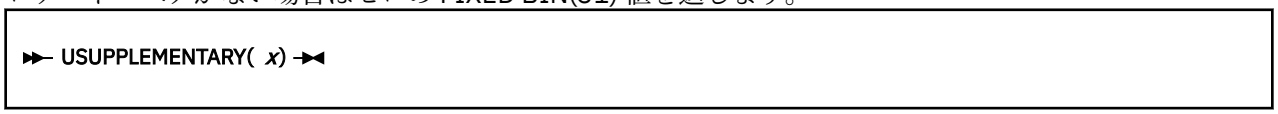
一般的に、USUBSTR(*x,i,j*) は SUBSTR(*x,i,j*) と同じではありません。

例えば、*x* が CHARACTER ストリング '4b_c3_a4_66_65_72'x である場合には、次のようになります。

- USUBSTR(*x*,1,2) は '4b_c3_a4'x を返します
- USUBSTR(*x*,2,1) は 'c3_a4'x を返します
- USUBSTR(*x*,2,2) は 'c3_a4_66'x を返します
- USUBSTR(*x*,3,2) は '66_65'x を返します

## USUPPLEMENTARY

USUPPLEMENTARY は、ストリング内の最初の UTF サロゲート・ペアの索引、またはストリングに UTF サロゲート・ペアがない場合はゼロの FIXED BIN(31) 値を返します。




**x**
CHARACTER または WIDECHAR タイプが必要な式。

*x* が CHARACTER タイプである場合、ストリングには有効な UTF-8 データが入っている必要があります。ただし、データの妥当性はチェックされません。 データが無効な場合は、ERROR 条件は発生せず、プログラムがエラー状態になり、この関数が戻す結果は予測不能になります。

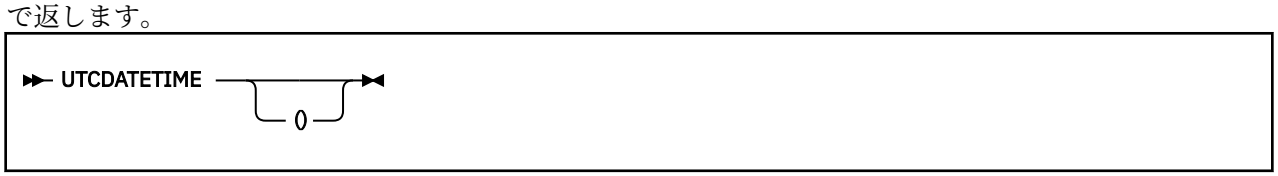
*x* が WIDECHAR タイプである場合、ストリングには有効な UTF-16 データが入っている必要があります。ただし、データの妥当性はチェックされません。 データが無効な場合は、ERROR 条件は発生せず、プログラムがエラー状態になり、この関数が戻す結果は予測不能になります。

例として、音楽のト音記号は、UTF-16 サロゲート・ペア 'D834_DD1E'wx で表され、したがって、以下のコードでは値 3 がリストされます。

```
dcl w  wchar(20) varying;
dcl jx fixed bin;

w = '0020_0020_D834_DD1E'wx

jx = usupplementary(w);

put skip list(jx);
```

## UTCDATETIME

UTCDATETIME は、現在の協定世界時 (UTC) を提供する文字ストリングをパターン *YYYYMMDDHHMISS*999 で返します。

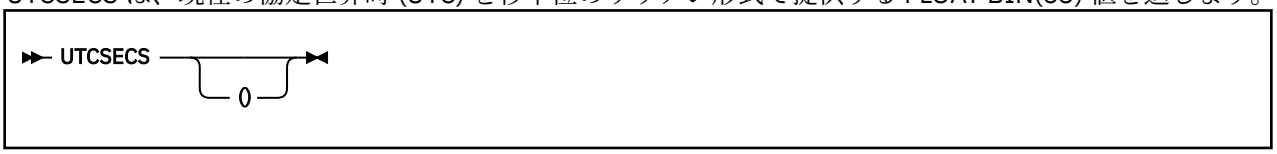



## UTCMICROSECS

UTCMICROSECS は、現在の UTC 時刻をマイクロ秒単位で提供する FIXED BINARY(63) 値を返します。

# UTCSECS

UTCSECS は、現在の協定世界時 (UTC) を秒単位のリリアン形式で提供する FLOAT BIN(53) 値を返します。




例えば、15 分単位の時間の個数を保持する変数を次のように定義します。

```
dcl qh fixed dec(5,2);
```

then

```
qh = 15*round(fixeddec((secs()-utcsecs())/900,7,2),0);
```

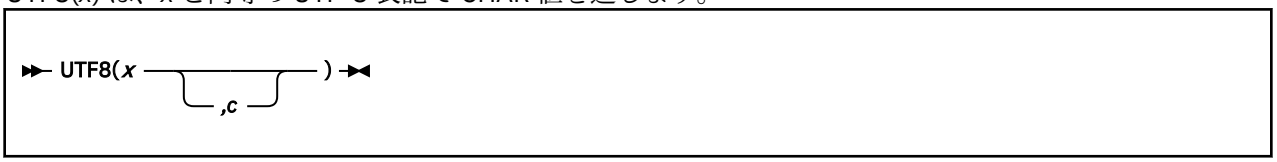
これにより、15 分単位の時間の個数としての UTC オフセットに設定されます。以下の式の場合、

```
edit((qh/60),'S99') || ':' || edit(rem(qh,60),'99')
```

通常のフォーマットの UTC オフセットを保持する char(6) ストリングになります。 例えば、カリフォルニアの場合は -08:00 で、ネパールの場合は +05:45 です。

# UTF8

UTF8(*x*) は、*x* と同等の UTF-8 表記で CHAR 値を返します。




**x**

式。FIXED、FLOAT、PICTURE、BIT、CHAR、または WIDECHAR のいずれかのタイプでなければなりません。

**c**

ソースのコード・ページを指定する制限付き式。 *x* が WIDECHAR 型を持つ場合は無視されます。

コード・ページは、計算タイプでなければならず、タイプ FIXED BINARY (31,0) に変換されます。 コード・ページでは、有効かつサポートされているコード・ページを指定する必要があります。

*x* が WIDECHAR 以外のタイプである場合、CODEPAGE オプションは、UTF-8 に変換されたときの *x* のコード・ページの値を指定します。

*x* が WIDECHAR タイプである場合は、*x* が UTF-16 を保持するものと仮定して UTF-8 への変換が行われます。

UTF8(*x*) は制限付き式で使用できます。 そのため、UTF8(*x*) を使用して UTF-8 リテラルを作成できます。

**注記 :**

- *x* が CHAR タイプである場合、UTF8(*x*) の長さが *x* の長さの 2 倍になる可能性があります。 *x* が WCHAR 型の場合、UTF8(*x*) の長さは *x* の長さの 3 倍になる可能性があります。 UTF8 の長さが CHAR の最大長を超える場合は、生成されたコードにより ERROR 条件が発生します。
- *x* が WCHAR タイプであり、無効な UTF-16 データを保持している場合は、生成されたコードで ERROR 条件が発生します。
- 例えば、UTF8('babb'x,1140) および UTF8('63fc'x,1141) は両方とも '5b5d'x を返します。 この理由は、コード・ページ 1140 では 'ba'x と 'bb'x が UTF-8 文字 '5b'x と '5d'x に相当しますが、コード・ページ 1141 では '63'x と 'fc'x が UTF-8 文字 '5b'x と '5d'x にマップするためです。

## UTF8STG

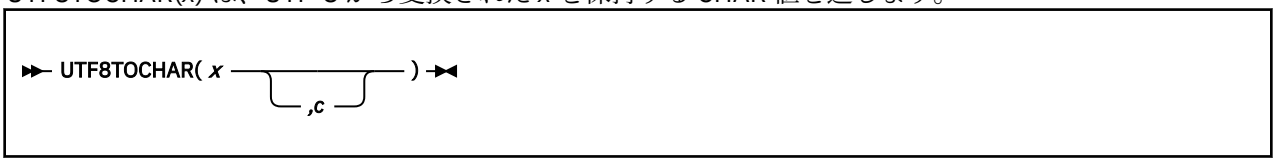
UTF8STG は、入力文字が有効な UTF-8 文字の始まりである場合に、存在しなければならないバイト数を示す FIXED BIN 値を返します。

**x**
入力文字を指定します。 *x* のタイプは CHAR(1) でなければなりません。

この関数は、文字が有効な UTF-8 文字の始まりにできない場合はゼロを返します。 例えば、文字に値 '80'x がある場合、UTF8STG はゼロを返します。

## UTF8TOCHAR

UTF8TOCHAR(*x*) は、UTF-8 から変換された *x* を保持する CHAR 値を返します。




**x**
式。CHAR タイプでなければなりません。

**c**
*x* のコード・ページを指定する制限付き式。

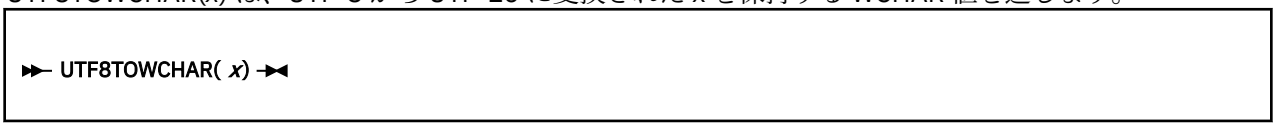
省略した場合、デフォルトで CODEPAGE コンパイラー・オプション内の値に設定されます。

指定された場合、コード・ページは、計算タイプでなければならず、FIXED BINARY (31,0) に変換されます。 コード・ページでは、有効かつサポートされているコード・ページを指定する必要があります。

**注 :** *x* が無効な UTF-8 データを保持している場合は、生成されたコードで ERROR 条件が起こります。

## UTF8TOWCHAR

UTF8TOWCHAR(*x*) は、UTF-8 から UTF-16 に変換された x を保持する WCHAR 値を返します。




***x***
式。CHAR タイプでなければなりません。 *x* は UTF-8 から UTF-16 に変換されます。

UTF8TOWCHAR(*x*) は制限付き式で使用できます。

**注 :** *x* が無効な UTF-8 データを保持している場合は、生成されたコードで ERROR 条件が起こります。

## UUID

UUID は、バージョン 5 フォーマットの汎用固有 ID である CHARACTER(36) ストリングを返します。

PL/I によって生成される UUID は、RFC 4122 のバージョン 5 フォーマットの UUID です。

UUID を構成する名前空間情報には、以下のものが含まれます。

1. システム情報:
   - CPU カウント
   - MVS 名

- ストレージ・サイズ
- 型式番号およびシリアル番号

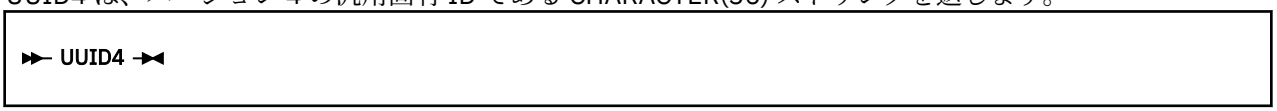
2. ランタイム 64 ビット TOD 値。
3. ジョブ名、または可能な場合、PID。

UUID は、上記の SHA1 ハッシュです。

USAGE(UUID(LOWER)) コンパイラー・オプションが有効である場合、返されるストリング内のすべての英字は小文字になります。 USAGE(UUID(UPPER)) コンパイラー・オプションが有効である場合、返されるストリング内のすべての英字は大文字になります。

## UUID4

UUID4 は、バージョン 4 の汎用固有 ID である CHARACTER(36) ストリングを返します。

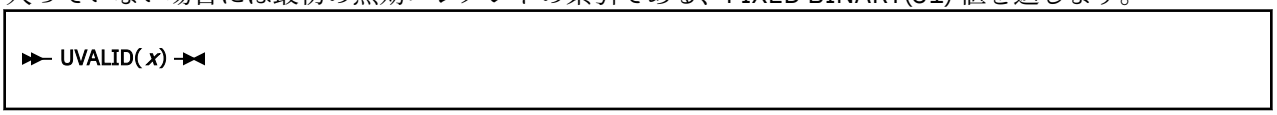



PL/I によって生成される UUID4 は、RFC 4122 のバージョン 4 の UUID です。 これは、真乱数または疑似乱数から UUID を生成するためのものです。

USAGE(UUID(LOWER)) コンパイラー・オプションが有効である場合、返されるストリング内のすべての英字は小文字になります。 USAGE(UUID(UPPER)) コンパイラー・オプションが有効である場合、返されるストリング内のすべての英字は大文字になります。

## UVALID

UVALID は、ストリングに有効な UTF データが入っている場合はゼロ、ストリングに有効な UTF データが入っていない場合には最初の無効エレメントの索引である、FIXED BINARY(31) 値を返します。




**x**
CHARACTER、UCHAR、WIDECHAR、または WIDEPIC タイプが必要な式。

*x* が CHARACTER タイプである場合、UVALID(*x*) は、ストリングに有効な UTF-8 データが入っていれば 0 を返します。 それ以外の場合は、最初の無効な UTF-8 データが開始している BYTE の索引を返します。

*x* が UCHAR タイプである場合、UVALID(*x*) は、ストリングに有効な UTF-8 データが入っていれば 0 を返します。 それ以外の場合は、最初の無効な UTF-8 データが開始している UCHAR の索引を返します。

*x* が WIDECHAR または WIDEPIC タイプである場合、UVALID(*x*) は、ストリングに有効な UTF-16 データが入っていれば 0 を返します。 それ以外の場合は、最初の無効な UTF-16 データが開始している WIDECHAR の索引を返します。

なお、UVALID は、(以下の規則に従って) ストリングに有効な UTF データが入っているかどうかを示します。 バイトが特定の文字を表現するために実際に割り振られているかどうかは示しません。

UTF-8 データの場合、バイトの妥当性は、その範囲によって以下のように異なります。

- '00'x から '7f'x: 有効
- '80'x から 'c1'x: 無効
- 'c2'x から 'df'x: 後ろ 2 番目のバイトで、そのバイトが '80'x から 'bf'x の範囲にある場合は有効
- 'e0'x から 'ef'x: 後ろにさらに 2 バイトあり、以下の場合に有効
  - 最初のバイトが 'e0'x で、2 番目と 3 番目のバイトがそれぞれ 'a0'x から 'bf'x、'80'x から 'bf'x の範囲にある場合
  - 最初のバイトが 'e1'x から 'ec'x の範囲にあり、2 番目と 3 番目のバイトが '80'x から 'bf'x の範囲にある場合

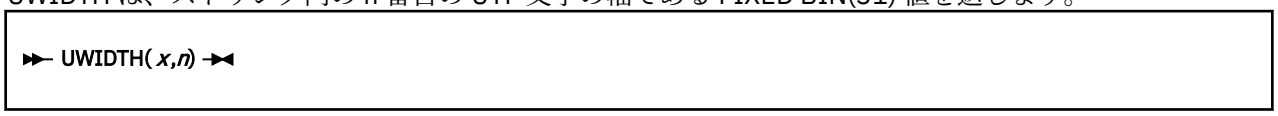
  - 最初のバイトが 'ed'x で、2 番目と 3 番目のバイトがそれぞれ '80'x から '9f'x、'80'x から 'bf'x の範囲にある場合
  - 最初のバイトが 'ee'x から 'ef'x の範囲にあり、2 番目と 3 番目のバイトが '80'x から 'bf'x の範囲にある場合
- 'f0'x から 'f4'x: 後ろにさらに 3 バイトあり、以下の場合に有効
  - 最初のバイトが 'f0'x で、2 番目、3 番目、および 4 番目のバイトがそれぞれ '90'x から 'bf'、'80'x から 'bf'x、'80'x から 'bf'x の範囲にある場合
  - 最初のバイトが 'f1'x から 'f3'x の範囲にあり、2 番目、3 番目、および 4 番目のバイトが '80'x から 'bf'x の範囲にある場合
  - 最初のバイトが 'f4'x で、2 番目、3 番目、および 4 番目のバイトがそれぞれ '80'x から '8f'x、'80'x から 'bf'x、'80'x から 'bf'x の範囲にある場合
- 'f5'x から 'ff'x: 無効

UTF-16 データの場合、ワイド文字の妥当性は、その範囲によって以下のように異なります。

- '0000'wx から '007f'wx: 有効。UTF-8 の場合は 1 バイト
- '0080'wx から '07ff'wx: 有効。UTF-8 の場合は 2 バイト
- '0800'wx から 'd7ff'wx: 有効。UTF-8 の場合は 3 バイト
- 'd800'wx から 'dbff'wx: 'dc00'wx 以上および 'dfff'wx 以下の値を持つ 2 番目のワイド文字が後ろにある場合は有効。 Unicode サロゲート・ペアであり、UTF-8 の場合は 4 バイト
- 'dc00'wx から 'dfff'wx: サロゲート・ペアの後半である場合にのみ有効
- 'e000'wx から 'ffff'wx: 有効。UTF-8 の場合は 3 バイト

## UWIDTH

UWIDTH は、ストリング内の n 番目の UTF 文字の幅である FIXED BIN(31) 値を返します。




**x**
CHARACTER または WIDECHAR タイプが必要な式。

**n**
計算タイプで、必要に応じて FIXED BIN(31) に変換される式。

*x* が CHARACTER タイプである場合、ストリングには有効な UTF-8 データが入っている必要があります。 それ以外の場合、プログラムはエラーになります。

*x* が WIDECHAR タイプである場合、ストリングには有効な UTF-16 データが入っている必要があります。 それ以外の場合、プログラムはエラーになります。

*n* が正ではない場合、または *n* が ULENGTH(*x*) より大きい場合は、ゼロが返されます。 それ以外の場合は、*x* が CHARACTER タイプの場合は、UWIDTH(*x,n*) は *n* 番目の UTF-8 文字の幅を返し、*x* が WIDECHAR タイプの場合は、UWIDTH(*x,n*) は *n* 番目の UTF-16 文字の幅を返します。

例えば、*x* が CHARACTER ストリング '4b_c3_a4_66_65_72'x である場合には、次のようになります。

- UWIDTH(*x*,1) は 1 を返します
- UWIDTH(*x*,2) は 2 を返します
- UWIDTH(*x*,3) は 1 を返します
- UWIDTH(*x*,4) は 1 を返します
- UWIDTH(*x*,5) は 1 を返します

## VALID

VALID は、参照の内容がそのデータ・タイプに有効であるかどうかを示す BIT(1) 値を返します。

**x**
PICTURE、FIXED DEC、または ORDINAL タイプのいずれかによる参照。

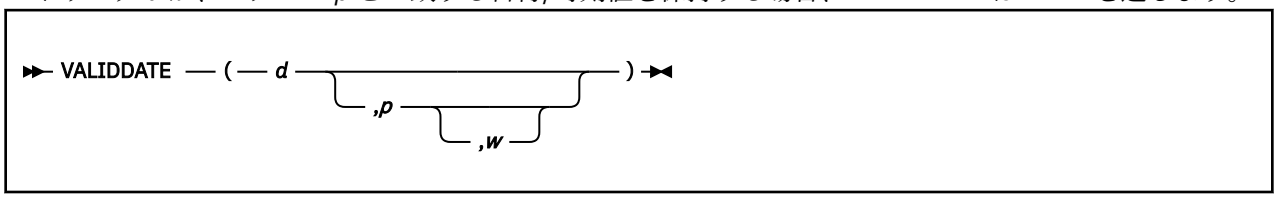
VALID(x) は、以下の場合に '1'b を返します。

- x が PICTURE または WIDEPIC であり、その内容が x のピクチャー指定に有効である。
- x が FIXED DECIMAL で、x のデータが有効なパック 10 進数データである。
- x が ORDINAL で、x のデータがその序数タイプに定義された値の 1 つである。

それ以外の場合は、'0'b を返します。

## VALIDDATE

ストリング *d* が、パターン *p* と一致する日付/時刻値を保持する場合、VALIDDATE は '1'B を返します。




**d**
日付を表すストリング式。

*d* には、パターン *p* に従って日時を表す文字ストリングとして入力日を指定します。

*d* に計算タイプ (必須) および文字タイプを保持する必要があります。 そうでない場合は、*d* は文字に変換されます。

**p**
サポートされている日付/時刻パターンの 1 つ。

存在する場合は、*d* の日付/時刻パターンを指定します。 *p* が欠落している場合は、'YYYYMMDDHHMISS999' のデフォルトの日付/時刻パターンであると想定されます。

*p* に計算タイプ (必須) および文字タイプを保持する必要があります。 そうでないと、文字に変換されます。

**w**
整数に変換できる式 (1950 など) を指定します。 負の場合、コードを実行するときに年の値から減算されるようにオフセットを指定します。 省略される場合は、*w* は、WINDOW コンパイル時オプションに指定された値に デフォルトで設定されます。

許可されるパターンを 382 ページの表 65 にリストしてあります。 リリアン形式の説明については、380 ページの『日付/時刻組み込み関数』を参照してください。

パターンに句読文字が含まれている場合、VALIDDATE はそれに一致する文字が入力ストリングに含まれているかを検査します。 例えばパターン YYYY-MM-DD の場合、VALIDDATE は 2019-03-14 を受け入れますが、2019.03.14 は受け入れません。

**例**

```
dcl duedate   char(8);
dcl (b1,b2)   bit(1);

duedate = '20190228';
b1 = validdate( duedate, 'YYYYMMDD' ); /* b1 = '1'b */

duedate = '02302019';
b2 = validdate( duedate, 'DDMMYYYY' ); /* b2 = '0'b */
```

# VALIDVALUE

VALIDVALUE は、式の値が変数の値セットのエレメントの 1 つと一致するかどうかを示す値を返します。

**x**
VALUELIST または VALUERANGE 属性を保持する必要がある参照。

**y**
*x* の値セットに対してテストされる式。 *x* に計算タイプが保持されている場合は、*y* も計算タイプを保持している必要があり、これは必要に応じて *x* と同じタイプに変換されます。

*x* に序数タイプが保持されている場合、*y* は同じ序数タイプを保持している必要があります。

*y* を省略すると、デフォルトの *x* になります。 VALIDVALUE(*x*) は、VALIDVALUE(*x*,*x*) と同等です。

VALIDVALUE は、以下の場合に、BIT(1) 値 '1'B を返します。

- *x* に VALUELIST 属性が保持され、その内容がそのリストのエレメントの 1 つである。
- *x* に VALUERANGE 属性が保持され、その内容がその範囲内である。

それ以外の場合は、'0'B を返します。

*x* に VALUERANGE 属性が保持されている場合、VALIDVALUE テストには 2 つのエンドポイント値が含まれます。 例えば、次のように宣言したとします。

```
dcl x  fixed dec(5) valuerange( 1900, 2100 );
```

次のテスト

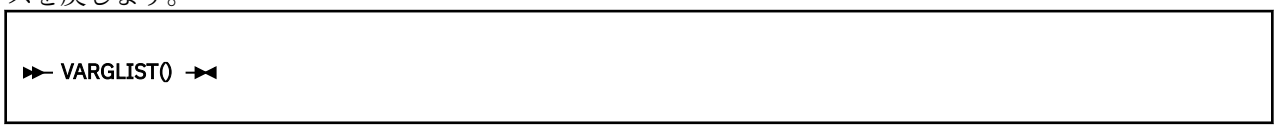
```
if validvalue( x ) then
```

この宣言は、次の宣言と同等です。

```
if (1900 <= x) & (x <= 2100) then
```

# VARGLIST

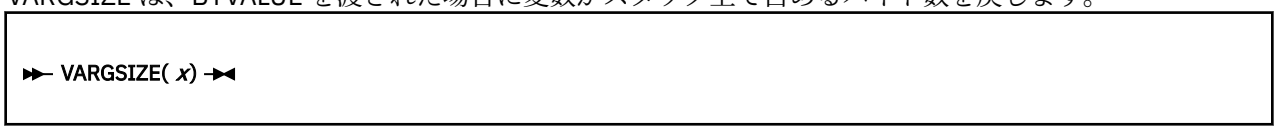
VARGLIST は、引数の変数番号を持つプロシージャーに渡された最初のオプション・パラメーターのアドレスを戻します。




VARGLIST 組み込み関数は、最後のパラメーターが LIST 属性をとるプロシージャーの中でのみ使用されます。

# VARGSIZE

VARGSIZE は、BYVALUE を渡された場合に変数がスタック上で占めるバイト数を戻します。




**x**
任意のデータ・タイプ、データ編成、位置合わせ、およびストレージ・クラスを持つ変数。ただし、下記のものは除きます。

*x* は、次のものであることはできません。

- 位置合わせされていない固定長ビット・ストリングである BASED 変数、DEFINED 変数、 パラメーター変数、添え字付き変数、構造変数、または共用体基底付き変数。
- 最初または最後の基本エレメントが位置合わせされていない固定長の ビット・ストリングである小構造体または共用体 (ただし、その小構造体を含んでいる 大構造体または共用体の最初または最後のエレメントになっている場合を除く)。
- BASED 属性、DEFINED 属性、またはパラメーター属性を持つ大構造体か共用体、またはパラメーターである大構造体か 共用体で、位置合わせされていない固定長のビット・ストリングを最初または最後のエレメントとするもの。
- 連結ストレージにない変数。

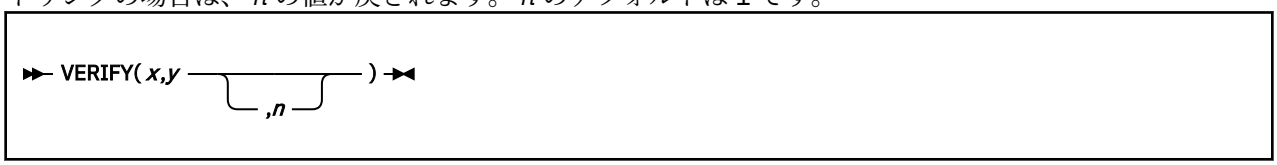
VARGSIZE(*x*) は、BYVALUE を渡された場合に *x* がスタック上で占めるバイト数を戻します。 この値は、少なくとも SIZE(*x*) と同じ大きさになります。 SIZE(*x*) によって戻される値が 4 バイトの倍数まで切り上げられる必要がある場合には この値はそれより大きくなります。

VARGSIZE は、最後のパラメーターが LIST 属性をとるプロシージャーの中でのみ 使用されなければなりません。

# VERIFY

VERIFY は、*y* にない左端の 文字、ビット、グラフィック、uchar、またはワイド文字 の *x* 内の位置を示すスケールなしの REAL FIXED BINARY 値を返します。 また、この関数を使用すると、処理を開始する *x* 内での位置を指定できます。

*x* のすべての文字、ビット、グラフィック、uchar、またはワイド文字が *y* にある場合は、値ゼロが返されます。 *x* がヌル・ストリングの場合は、値ゼロが戻されます。 *x* がヌル・ストリングでなく、*y* がヌル・ストリングの場合は、 *n* の値が戻されます。 *n* のデフォルトは 1 です。




**x**
ストリング式。

**y**
ストリング式。

**n**
*n* は、処理を開始する *x* 内の位置を指定します。 計算タイプでなければならず、FIXED BINARY(31,0) に変換されます。

1 ≤ *n* ≤ LENGTH(*x*) + 1 でない場合は、STRINGRANGE 条件 (割り込み可能な場合) が発生します。 暗黙処置と通常の戻りでは、結果は 0 になります。 *n* = LENGTH(*x*) + 1 の場合は、結果はゼロです。

BIFPREC コンパイラー・オプションによって、戻される結果の精度が決まります。

VERIFY は、2 番目および 3 番目の引数がリテラル、VALUE 属性を使用して宣言された名前付き定数、または制限付き式のいずれかである場合に、最も機能を発揮します。

**例**

```
X = '  a  b';         /* Two blanks in each space */
Y = ' ';              /* One blank                */
N = 1;
I = verify(X,Y,N);    /* I = 3 */

do while (I > 0);
  display ( 'Nonblank at position ' || trim(I) );
  N = I + 1;
  I = verify(X,Y,N);
end;
```

DO ループの最初のパススルーのあとで、N=4 で VERIFY(X,Y,N) は 6 を戻します。 2 回目のパスのあとでは、N=7 (LENGTH(*x*)+1) で VERIFY(X,Y,N) は 0 を戻し、ループが終了します。

他の VERIFY 組み込み関数の例については、532 ページの『SEARCH』を参照してください。

# VERIFYR

VERIFYR は、検証が右から左へ行われることを除き、VERIFY 組み込み関数と同じ演算を行います。

もう 1 つの相違点は、*n* のデフォルト値が LENGTH(*x*) であることです。

0 ≤ *n* ≤ LENGTH(*x*) でないと、STRINGRANGE 条 件 (割り込み可能な場合) が発生します。 *n* = 0 の場合は、結果はゼロです。

BIFPREC コンパイラー・オプションによって、戻される結果の精度が決まります。

VERIFYR は、2 番目および 3 番目の引数がリテラル、VALUE 属性を使用して宣言された名前付き定数、または制限付き式のいずれかである場合に、最も機能を発揮します。

引数の説明については、563 ページの『VERIFY』を参照してください。

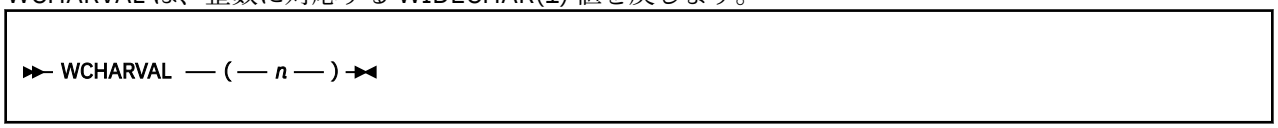
**例**

```
X = 'a  b  ';         /* Two blanks in each space */
Y = ' ';              /* One blank                */
N = length(X);        /* N = 6                 */
I = verifyr(X,Y,N);   /* I = 4                 */

do while (I > 0);
  display ( 'Nonblank at position ' || trim(I) );
  N = I - 1;
  I = verifyr(X,Y,N);
end;
```

DO ループの最初のパススルーの後、N= 3 および VERIFYR(X,Y,N) は 1 を戻します。 2 回目のパスの後、N= 0, VERIFYR(X,Y,N) は 0 を戻し、ループは終了します。 別の例については、533 ページの『SEARCHR』を参照してください。

# WCHARVAL

WCHARVAL は、整数に対応する WIDECHAR(1) 値を戻します。




**n**
式。必要な場合には、UNSIGNED FIXED BIN(16) に変換されます。

n がビッグ・エンディアン・フォーマットの場合、WCHARVAL(*n*) のビット値は *n* と同じですが (つまり、UNSPEC(WCHARVAL(*n*)) は UNSPEC(*n*) と等しい)、属性は WIDECHAR(1) です。

WCHARVAL は RANK の逆です (ワイド文字に適用した場合)。

# WEEKDAY

WEEKDAY は、日数 *x* が曜日に変換された FIXED BINARY(31,0) 値を返します。ここで、1=Sunday、2=Monday、. . .7=Saturday です。 *x* が指定されない場合は、今日の日付と見なされます。

**x**

式。 存在する場合は、*x* は、入力データを日数として指定します。 存在しない場合は、*x* は、DAYS() であると想定されます。

*x* がなく、今日の日付がシステムから利用できない場合は、結果 のゼロが戻されます。

*x* は、計算タイプでなければなりません。また、必要な場合には、 FIXED BINARY(31,0) に変換されます。

WEEKDAY の例については、534 ページの『SECS』を参照してください。

## WHIGH

WHIGH は、それぞれ最大のワイド文字値 (16 進 FFFF) を持つワイド文字で 構成される、長さ *x* のワイド文字ストリングを戻します。

**x**

式。 必要な場合には、*x* は正の実数固定小数点の 2 進数の値に変換されます。 *x* = 0 の場合は、結果はヌルのワイド文字ストリングです。

## WHEREDIFF

WHEREDIFF は、2 つのバッファーで異なる最初のバイトの索引を指定する size_t 値を返します。すべてのバイトが同じ場合はゼロを返します。

**x**

式。 POINTER 型または OFFSET 型でなければなりません。 OFFSET の場合は、式を AREA 修飾を使用して宣言する必要があります。

**y**

式。 POINTER 型または OFFSET 型でなければなりません。 OFFSET の場合は、式を AREA 修飾を使用して宣言する必要があります。

**z**

式。 これは size_t に変換されます。

2 つのバッファーが異なる場合、WHEREDIFF 組み込み関数は、異なる最初のバイトが 2 番目のバッファーの対応するバイトより大きいか小さいかを示しません。 バッファーの違いを知りたい場合は、代わりに COMPARE 組み込み関数を使用してください。

## WIDECHAR

WIDECHAR は、*y* で指定された長さでワイド文字値 *x* を戻します。

**省略形**: WCHAR

**x**
式。

*n* は、計算タイプでなくてはなりません。

*x* の値は、チェックされません。

**y**
式。 必要に応じて、y は実数の固定小数点 2 進数値に変換されます。

*y* が省略される場合は、長さはタイプ変換の規則によって決められます。

*y* は負であってはなりません。

*y* = 0 の場合は、結果はヌルのワイド文字ストリングです。

## WLOW

WLOW は、それぞれ最小のワイド文字値 (16 進 0000) を持つワイド文字で 構成される、長さ *x* のワイド文字ストリングを戻します。

**x**
式。 必要な場合には、*x* は正の実数固定小数点の 2 進数の値に変換されます。 *x* = 0 の場合は、結果はヌルのワイド文字ストリングです。

## WSCOLLAPSE

WSCOLLAPSE は、CHARACTER ソース・バッファー内のすべての空白文字を縮小した時にターゲット・バッファーに書き込まれるバイト数を示す *size_t* [1] 値を返します。

WSCOLLAPSE は、以下のいずれかの方法で空白文字を縮小します。

- ¥t¥f¥v¥n¥r の各文字をブランクに置き換えます。
- すべての先行ブランクおよび末尾ブランクをトリムします。
- 複数の内部ブランクを 1 つのブランクに縮小します。

**p**
ターゲット・バッファーのアドレスを指定します。

**m**
ターゲット・バッファーの長さを指定します。計算タイプでなければならず、タイプ *size_t* に変換されます。

**q**
ソース・バッファーのアドレスを指定します。

**n**
ソース・バッファーの長さを指定します。計算タイプでなければならず、タイプ *size_t* に変換されます。

ターゲット・バッファーのアドレスがゼロである場合は、書き込まれることが想定されたバイト数が返されます。 ターゲット・バッファーの大きさが十分でなかった場合には、値 -1 が返されます。 ターゲット・バッファーの大きさが十分であった場合には、バッファーに書き込まれたバイト数が返されます。

# WSCOLLAPSE16

WSCOLLAPSE16 は、UTF-16 としてエンコードされているソース・バッファー内のすべての空白文字を縮小します。 ターゲット・バッファーに書き込まれたバイト数を示す *size_t* [1] 値を返します。 WHITESPACECOLLAPSE は WSCOLLAPSE16 の非推奨同義語です。

WSCOLLAPSE16 は、以下のいずれかの方法で空白文字を縮小します。

- ¥t¥f¥v¥n¥r の各文字を UTF-16 のブランクに置き換えます。
- すべての先行ブランクおよび末尾ブランクをトリムします。
- 複数の内部ブランクを 1 つのブランクに縮小します。

**p**
ターゲット・バッファーのアドレスを指定します。

**m**
ターゲット・バッファーの長さを指定します。計算タイプでなければならず、タイプ *size_t* に変換されます。

**q**
ソース・バッファーのアドレスを指定します。

**n**
ソース・バッファーの長さを指定します。計算タイプでなければならず、タイプ *size_t* に変換されます。

ターゲット・バッファーのアドレスがゼロである場合は、書き込まれることが想定されたバイト数が返されます。 ターゲット・バッファーの大きさが十分でなかった場合には、値 -1 が返されます。 ターゲット・バッファーの大きさが十分であった場合には、バッファーに書き込まれたバイト数が返されます。

ソース・バッファーには UTF-16 データが保持される必要があります。

# WSREPLACE

WSREPLACE は、CHARACTER としてエンコードされたソース・バッファー内の \t、\f、\v、\n の各文字をブランクで置き換えます。 この関数は、ターゲット・バッファーに書き込まれたバイト数を示す *size_t* [1] 値を返します。

**p**
ターゲット・バッファーのアドレスを指定します。

**m**
ターゲット・バッファーの長さを指定します。計算タイプでなければならず、タイプ *size_t* に変換されます。

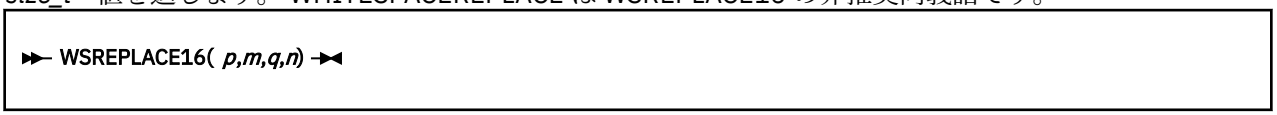
**q**
ソース・バッファーのアドレスを指定します。

**n**
ソース・バッファーの長さを指定します。計算タイプでなければならず、タイプ *size_t* に変換されます。

ターゲット・バッファーのアドレスがゼロである場合は、書き込まれることが想定されたバイト数が返されます。 ターゲット・バッファーの大きさが十分でなかった場合には、値 -1 が返されます。 ターゲット・バッファーの大きさが十分であった場合には、バッファーに書き込まれたバイト数が返されます。

# WSREPLACE16

WSREPLACE16 は、UTF-16 としてエンコードされたソース・バッファー内の \t、\f、\v、\n のすべての文字をブランクで置き換えます。 この関数は、ターゲット・バッファーに書き込まれたバイト数を示す *size_t* [1] 値を返します。 WHITESPACEREPLACE は WSREPLACE16 の非推奨同義語です。




**p**
ターゲット・バッファーのアドレスを指定します。

**m**
ターゲット・バッファーの長さを指定します。計算タイプでなければならず、タイプ *size_t* に変換されます。

**q**
ソース・バッファーのアドレスを指定します。

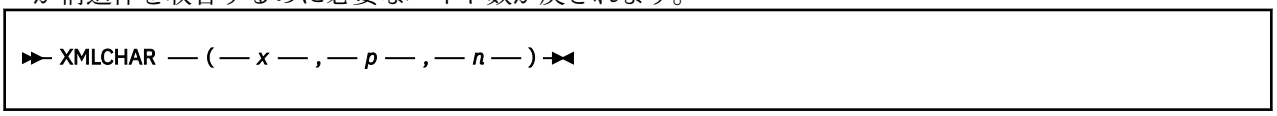
**n**
ソース・バッファーの長さを指定します。計算タイプでなければならず、タイプ *size_t* に変換されます。

ターゲット・バッファーのアドレスがゼロである場合は、書き込まれることが想定されたバイト数が返されます。 ターゲット・バッファーの大きさが十分でなかった場合には、値 -1 が返されます。 ターゲット・バッファーの大きさが十分であった場合には、バッファーに書き込まれたバイト数が返されます。

ソース・バッファーには UTF-16 データが保持される必要があります。

# XMLCHAR

XMLCHAR は、構造体のデータを XML としてバッファーにダンプします。 バッファーに書き込まれたバイト数を示す *size_t* [1] 値を返します。 バッファーが小さすぎる場合、構造体データは切り捨てられ、バッファーが構造体を収容するのに必要なバイト数が戻されます。




**x**
構造体または DEFINE STRUCTURE タイプへの参照。

参照 *x* は、次の規則に従っていなければなりません。

- これにはストリングや数値データなどの、計算可能なデータのみが含まれていなければなりません。ただし、GRAPHIC、UCHAR、WIDECHAR、または WIDEPIC エレメントを含めることはできません。
- 配列を含めることができますが、それ自体が配列である場合は、完全に添え字が付いていなければなりません。
- これには副構造体が含まれていることがありますが、含まれている副構造体は、名前の代わりにアスタリスク (*) を使用することはできません。 ただし、基本エレメントの名前としてアスタリスクを使用することはできますが、その場合ターゲット・バッファーに、名前が付けられていないエレメントは書き込まれません。
- *x* が構造体への参照である場合、DEFINE STRUCTURE タイプを含んでいてはなりません。

**p**
ターゲット・バッファーのアドレス。

**n**
ターゲット・バッファーの長さ。

バッファー長は、負以外で、計算タイプでなければなりません。 バッファー長はタイプ *size_t* に変換されます。

XML 出力が作成される場合は、以下の規則に従います。

- 変数に XMLCONTENT 属性がある場合、その変数はタグなしテキストとして表示されます。
- XMLATTR 属性を持つ変数がない場合、構造体内の各名前は、最初に「<」で囲まれて書き出されます。および "> "以降は" </"および ">" で囲まれています。
- 変数に XMLATTR 属性がある場合、フィールドはその含まれている構造の属性として表されます。
- 変数に XMLOMIT 属性がある場合、フィールドはヌル値があれば省略されます。
- 数値とビット・データは文字に変換されます。
- 先行ブランクと末尾ブランクは、可能な場合はいつでもトリムされます。

**注 :** デフォルトでは、生成された XML 出力内の変数名は、すべて大文字です。 XML コンパイラー・オプションの CASE(ASIS) サブオプションを使用すると、宣言されたとおりの大 / 小文字で名前を表示することを指定できます。

### XMLCHAR の使用例

この例は以下のコード・フラグメントに基づいています。

```
dcl buffer   char(800);
dcl written  fixed bin(31);
dcl next     pointer;
dcl left     fixed bin(31);
dcl
  1 a,
   2 a1,
     3 b1 char(8),
     3 b2 char(8),
   2 a2,
     3 c1 fixed bin,
     3 c2 fixed dec(5,1);

b1 = ' t1';
b2 = 't2';
c1 = 17;
c2 = -29;
next = addr(buffer);
left = stg(buffer);
written = xmlchar( a, next, left );
next += written;
left -= written;
```

以下のバイトがバッファーに書き込まれ、`written` は 72 に設定されます。

```
<A><A1><B1>t1</B1><B2>t2</B2></A1><A2><C1>17</C1><C2>-29.0</C2></A2></A>
```

### 関連情報

181 ページの『XML 関連属性』
XML 関連属性は、XML 属性の使用および XMLCHAR または XMLUCHAR 組み込み関数によって生成される制御出力を指定します。

## XMLSCRUB

XMLSCRUB は、CHARACTER ソース・バッファーを消し込みます。 ターゲット・バッファーに書き込まれたバイト数を示す *size_t*[1] 値を返します。

XMLSCRUB は、以下の方法で CHARACTER ソース・バッファーをクリーニングします。

- \t、\n、\r を除き、1 ブランクより小さい各文字をブランクに置き換えます。
- 復帰を  に置き換えます。
- 以下の文字を対応するストリングに置き換えます。

| 文字 | 文字列 |
|---|---|
| " | " |
| ' | ' |

| 文字 | 文字列 |
|---|---|
| & | & |
| < | < |
| > | > |

**p**
ターゲット・バッファーのアドレスを指定します。

**m**
ターゲット・バッファーの長さを指定します。計算タイプでなければならず、タイプ *size_t* に変換されます。

**q**
ソース・バッファーのアドレスを指定します。

**n**
ソース・バッファーの長さを指定します。計算タイプでなければならず、タイプ *size_t* に変換されます。

ターゲット・バッファーのアドレスがゼロである場合は、書き込まれることが想定されたバイト数が返されます。 ターゲット・バッファーの大きさが十分でなかった場合には、値 -1 が返されます。 ターゲット・バッファーの大きさが十分であった場合には、バッファーに書き込まれたバイト数が返されます。

## XMLSCRUB16

XMLSCRUB16 は、UTF-16 ソース・バッファーを消し込みます。 ターゲット・バッファーに書き込まれたバイト数を示す *size_t* [1] 値を返します。 XMLCLEAN は XMLSCRUB16 の非推奨同義語です。

XMLSCRUB16 は、以下の方法で UTF-16 ソース・バッファーをクリーニングします。

- 無効な各 UTF-16 を UTF-16 のブランクに置き換えます。
- 復帰を  に置き換えます。
- 以下の文字を対応するストリングに置き換えます。

| 文字 | 文字列 |
|---|---|
| " | " |
| ' | ' |
| & | & |
| < | < |
| > | > |

**p**
ターゲット・バッファーのアドレスを指定します。

**m**
ターゲット・バッファーの長さを指定します。計算タイプでなければならず、タイプ *size_t* に変換されます。

**q**
ソース・バッファーのアドレスを指定します。

**n**
ソース・バッファーの長さを指定します。計算タイプでなければならず、タイプ *size_t* に変換されます。

ターゲット・バッファーのアドレスがゼロである場合は、書き込まれることが想定されたバイト数が返されます。 ターゲット・バッファーの大きさが十分でなかった場合には、値 -1 が返されます。 ターゲット・バッファーの大きさが十分であった場合には、バッファーに書き込まれたバイト数が返されます。

ソース・バッファーには UTF-16 データが保持される必要があります。

# XMLUCHAR

XMLUCHAR は、XML としての構造体のデータを UTF-8 としてバッファーにダンプします。 バッファーに書き込まれたバイト数を示す *size_t*[1] 値を返します。 バッファーが小さすぎる場合、構造体データは切り捨てられ、バッファーが構造体を収容するのに必要なバイト数が戻されます。

**x**
構造体または DEFINE STRUCTURE タイプへの参照。

参照 *x* は、UCHAR エレメントを含むことができる点を除き、XMLCHAR と同じ規則に準拠していなければなりません。

**p**
ターゲット・バッファーのアドレス。

**n**
ターゲット・バッファーの長さ。

バッファー長は、負以外で、計算タイプでなければなりません。 バッファー長はタイプ *size_t* に変換されます。

XML 出力が作成されると、XMLUCHAR は XMLCHAR と同じ規則に従います。

**関連情報**
568 ページの『XMLCHAR』
XMLCHAR は、構造体のデータを XML としてバッファーにダンプします。 バッファーに書き込まれたバイト数を示す *size_t*[1] 値を返します。 バッファーが小さすぎる場合、構造体データは切り捨てられ、バッファーが構造体を収容するのに必要なバイト数が戻されます。

# Y4DATE

Y4DATE は、パターン `'YYMMDD'` を持つ日付の値を使用し、2 桁の年を 4 桁の年にした日付の値を返します。

**d**
日付を表すストリング式。

*d* に計算タイプ (必須) および文字タイプを保持する必要があります。 そうでない場合は、*d* は文字に変換されます。

**w**
整数に変換できる式 (1950 など) を指定します。 負の場合、コードを実行するときに年の値から減算されるようにオフセットを指定します。 省略される場合は、*w* は、WINDOW コンパイル時オプションに指定された値に デフォルトで設定されます。

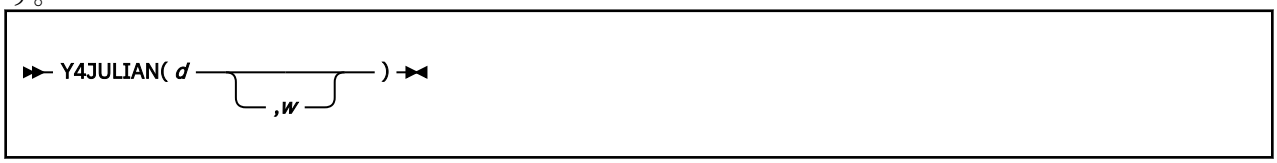
戻り値は、属性 CHAR(8) NONVARYING をとっており、次のように計算されます。

```
dcl y2 pic'99';
dcl y4 pic'9999';
dcl cc pic'99';

y2 = substr(d,1,2);
cc = w/100;

if y2 < mod(w,100) then
  y4 = 100*cc + 100 + y2;
else
  y4 = 100*cc + y2;

return( y4 || substr(d,3) );
```

Y4DATE('990101',1950) は '19990101' を戻します。
Y4DATE('000101',1950) は '20000101' を戻します。

## Y4JULIAN

Y4JULIAN は、パターン 'YYDDD' を持つ日付の値を使用し、2 桁の年を 4 桁の年にした日付の値を返します。




**d**

日付を表すストリング式。 *d* の長さは 5 以上でなければなりません。 5 より大きい場合、超過する文字は先行ブランクで形成しなければなりません。

*d* に計算タイプ (必須) および文字タイプを保持する必要があります。 そうでないと、文字に変換されます。

**w**

整数に変換できる式 (1950 など) を指定します。 負の場合、コードを実行するときに年の値から減算されるようにオフセットを指定します。 省略される場合は、*w* は、WINDOW コンパイル時オプションに指定された値に デフォルトで設定されます。

戻り値は、属性 CHAR(7) NONVARYING をとっており、次のように計算されます。

```
dcl y2 pic'99';
dcl y4 pic'9999';
dcl c  pic'99';

y2 = substr(d,1,2);
cc = w/100;

if y2 < mod(w,100) then
  y4 = 100*cc + 100 + y2;
else
  y4 = 100*cc + y2;

return( y4 || substr(d,3) );
```

Y4JULIAN('99001',1950) は '1999001' を戻します。
Y4JULIAN('00001',1950) は '2000001' を戻します。

## Y4YEAR

Y4YEAR は、パターン 'YY' を持つ日付の値を使用し、2 桁の年を 4 桁の年にした日付の値を返します。

**d**

日付を表すストリング式。 *d* の長さは 2 以上でなければなりません。 2 より大きい場合、超過する文字は先行ブランクで形成しなければなりません。

*d* に計算タイプ (必須) および文字タイプを保持する必要があります。 そうでないと、文字に変換されます。

**w**

整数に変換できる式 (1950 など) を指定します。 負の場合、コードを実行するときに年の値から減算されるようにオフセットを指定します。 省略される場合は、*w* は、WINDOW コンパイル時オプションに指定された値に デフォルトで設定されます。

戻り値は、属性 CHAR(4) NONVARYING を持っており、次のように計算されます。

```
dcl y2 pic'99';
dcl y4 pic'9999';
dcl c  pic'99';

y2 = d;
cc = w/100;

if y2 < mod(w,100) then
  y4 = 100*cc + 100 + y2;
else
  y4 = 100*cc + y2;

return( y4 );
```

Y4YEAR('99',1950) は '1999' を戻します。
Y4YEAR('00',1950) は '2000' を戻します。

# 第 19 章 タイプ付き関数

タイプ付き関数を使用して、定義されたタイプを操作できます。 この章では、タイプ付き関数について説明します。

タイプ付き関数は、次の方法で組み込み関数と区別します。

- 1 つ以上の引数が定義済みタイプである。
- それらを宣言することはできない。
- 引数は、( と ) との記号ではなく、(: と :) との複合記号で囲む。

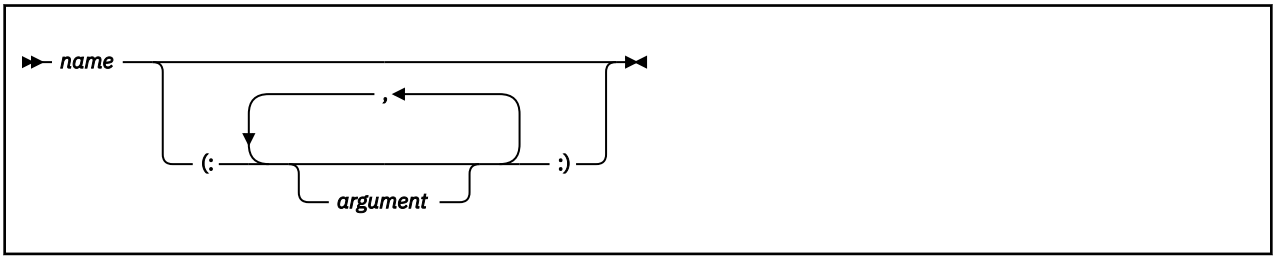
一般的に、次のようなフォーマットで記述します。

- 参照構文を示す見出し
- 戻り値の説明
- 引数の説明
- 関数の使用時のその他の修飾

## タイプ付き関数の呼び出し

タイプ関数を呼び出すには、タイプ関数の名前と、タイプ関数用の引数を指定します。

次の構文を使ってタイプ付き関数を呼び出します。




タイプ付き関数の引数は、区切り文字 (: および :) で囲みます。

## タイプ付き関数の引数の指定

タイプ付き関数の引数には、タイプ名 (別名、名前付き構造体および共用体、 序数) と、その他のデータ・タイプが可能です。

## タイプ付き関数の要旨

タイプ付き関数を、要旨とともにアルファベット順でリストします。

表 *84.* タイプ付き関数

| 機能 | 説明 |
|---|---|
| BIND | ポインターをタイプのハンドルに変換します。 |
| CAST | C の変換規則を使用して、式を指定のタイプに変換します。 |
| FIRST | 最初の値を序数セットに戻します。 |
| LAST | 最後の値を序数セットに戻します。 |
| 新規 | 構造体タイプのためにストレージを獲得し、 獲得されたストレージにハンドルを戻します。 |
| RESPEC | 式のビット・パターンを変更せずに、式の属性を指定のタイプに変更します。 |

© Copyright IBM Corp. 1999, 2022

表 84. タイプ付き関数 (続き)

| 機能 | 説明 |
|---|---|
| SIZE | タイプを表すために必要なストレージの量を戻します。 |
| VALUE | 対応する構造体型を持つ変数の初期化または代入を行います。 |

## BIND

BIND は、ポインター p を構造体タイプ t のハンドルに変換します。 BIND 関数は、タイプ付き構造体のメンバーのロケーターとして使用できます。

**t**
構造体タイプの名前

**p**
ポインターの式

## CAST

CAST は、C の変換規則を使用して、式 x をタイプ t に 変換します。

**t**
「C タイプ」のスカラーの名前

**x**
同じく「C タイプ」のスカラー式

サポートされる「C タイプ」は以下のとおりです。

- REAL FIXED BIN(p,0)
- REAL FIXED DEC(p,q) ここで、p >= q および q>= 0。
- NATIVE FLOAT
- ORDINAL
- POINTER または HANDLE
- LIMITED ENTRY

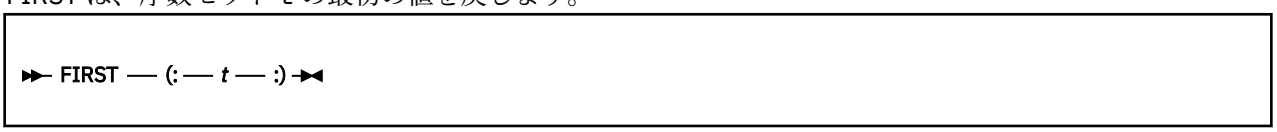
x が FLOAT または FIXED DEC の場合、t は FLOAT、FIXED、または ORDINAL でなければならず、t が FLOAT または FIXED DEC の場合、x は FLOAT、FIXED、または ORDINAL でなければなりません。

必要な変換はすべて、ANSI C 規則に従って行われます。 つまり、例えば CAST によって SIZE は発生せず、また負の値が UNSIGNED にキャストされた場合、結果は 大きな正数になります。

IEEE DFP は、CAST によってサポートされません。

## FIRST

FIRST は、序数セット t の最初の値を戻します。



**t**
序数タイプの名前

**例**

```
define ordinal Color ( Red,
                       Orange,
                       Yellow,
                       Green,
                       Blue,
                       Indigo,
                       Violet );

display (ordinalname( first(:Color:) ));  /* RED */
```

## LAST

LAST は、序数セット t の最後の値を戻します。

**t**
序数タイプの名前

**例**

```
define ordinal Color ( Red,
                       Orange,
                       Yellow,
                       Green,
                       Blue,
                       Indigo,
                       Violet );

display (ordinalname( last(:Color:) )); /* VIOLET */
```

## NEW

NEW は、構造体タイプ t のヒープ・ストレージを獲得し、獲 得したストレージにハンドルを戻します。

**t**
構造体タイプの名前

NEW(:t:) は、 BIND(: t, ALLOC( SIZE(:t:) ) :) と等価です。

## RESPEC

RESPEC は、式のビット値を変更せずに、 式 x の属性をタイプ t に変更します。

**t**
スカラー型の名前

**p**
スカラー式

x は t と同サイズでなければならず、x または t のいずれかが UNALIGNED BIT である場合は両方が UNALIGNED BIT でなければなりません (この場合、関数は何も行わないため、意味のないものになります)。

例えば、t が属性 LIMITED ENTRY を持つタイプである場合、 RESPEC( t, sysnull() ) は「ヌル」関数ポインターを戻します。

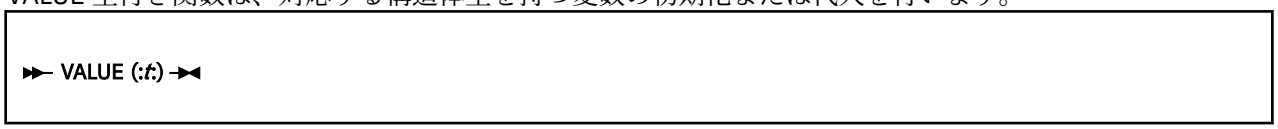
## SIZE

SIZE は、タイプ t で宣言された変数に必要なストレージ 必要量を戻します。

**t**
構造体タイプまたは共用体タイプの名前

## VALUE

VALUE 型付き関数は、対応する構造体型を持つ変数の初期化または代入を行います。




**t**
型付き構造体の名前。 VALUE 関数は、タイプ付き構造体 *t* のインスタンスをその初期値とともに返します。

VALUE 関数が部分的に初期化されている構造体型で使用されると、初期化されていないバイトおよびビットはゼロに設定されます。

VALUE 関数は、INITIAL 属性を持つ要素を含まない構造体型では使用できません。

VALUE 関数は、DEFINE STRUCTURE ステートメントの要素で、INIT 形式の INITIAL 属性と共に使用できます。 ただし、INIT CALL および INIT TO を、DEFINE STRUCTURE ステートメントの要素で VALUE 関数と共に使用することはできません。

以下の例は、VALUE 関数の使用方法を示しています。

```
define struct
  1 b,
     2 b1 fixed bin init(17),
     2 b2 fixed bin init(19);

 define struct
   1 c,
     2 c1 type b init( value(: b :) ),
     2 c2 fixed bin init(23);

 dcl x type c static init( value(: c :) );
 dcl y type c;

 y = value(: c :);
```

# 第 20 章 プリプロセッサーの機能

コンパイラーには、ソース・プログラムを変更できるように、MACRO プリプロセッサー が備わっています。 コンパイル時オプション MACRO または PP(MACRO) を指定すると、コンパイルの前にプリプロセッサーが実行されます。 MACRO プリプロセッサーは、プリプロセッサー入力を走査し、プリプロセッサー 出力を生成します。 プリプロセッサー出力は、コンパイラーへの入力として利用することができます。

この章では、読者が本書全体にわたって説明されている PL/I 言語をすでに理解していることを前提に、 プリプロセッサーを説明します。

プリプロセッサー入力 は、プリプロセッサー・ステートメント 、リスト制御ステートメント 、および入力テキスト の混在からなる文字ストリングです。

**プリプロセッサー・ステートメント**
プリプロセッサー・ステートメント [4] プリプロセッサー・スキャンによって検出されたときに実行されます (ただし、プリプロセッサー・プロシージャーは例外で、実行するには呼び出す必要があります)。 プリプロセッサー・ステートメント (プリプロセッサー・プロシージャー内のものを除く) の先頭にはパーセント記号 (%) が付いています。 オプションで、プランクを使用してステートメントの残りの部分からパーセント記号を分離できます。

プリプロセッサーは、プリプロセッサー・ステートメントを実行し、 それに応じて入力テキストを変更します。 プリプロセッサー・ステートメントを使用すれば、下記のいずれかの方法で 入力テキストの一部を変更することができます。

- 入力テキスト内の任意の ID (およびオプションの引数リスト) を テキストの任意のストリングに変更することができます。
- 入力テキストのどの部分を、プリプロセッサー出力の中にコピーするかを 指定することができます。
- ライブラリーに収められている文字のストリングを、プリプロセッサー 入力の中に組み込むことができます。

**リスト制御ステートメント**
リスト制御ステートメントは、プログラムの印刷リストのレイアウトを制御します。 これらのステートメントは、インソース・リスト (プリプロセッサー 入力) とソース・リスト (プリプロセッサー出力) の両方に 影響を与えます。 リスト制御ステートメントについては、193 ページの『第 8 章 ステートメントとディレクティブ』を参照してください。

**入力テキスト**
入力テキストは、プリプロセッサー・ステートメントでもリスト制御ステートメントでもないプリプロセッサー入力です。 入力テキストは PL/I 原始プログラムでもその他のテキストでも 構いませんが、その入力テキストは、プリプロセッサー走査による 入力テキストの処理に矛盾するものであってはなりません。

プリプロセッサーの出力 [5] 制御ステートメントのリストと出力テキストが混在している文字ストリングです。

**リスト制御ステートメント**
プリプロセッサー入力に含まれていて、走査されたリスト制御 ステートメントが、プリプロセッサー出力の中にコピーされます。

**出力テキスト**
走査され、おそらく変更された入力テキストが、プリプロセッサー出力に 組み入れられます。

プリプロセッサー入力、プリプロセッサー出力、またはその両方を印刷するか、データ・セットに書き込むための、コンパイル時オプションを指定できます。

---

[4] この説明では、わかりやすいように、プリプロセッサー・ステートメントを % 記号付きで示しています (ただし、プリプロセッサー・プロシージャーで使用される場合は、そのステートメントに % 記号はありません)。

[5] プリプロセッサー置換出力はフォーマット設定された形式で示されていますが、実行時に生成される実際の置換出力はフォーマット設定されていません。

© Copyright IBM Corp. 1999, 2022

# プリプロセッサー・オプション

MACRO または PP(MACRO) コンパイル時オプションを指定すると、 プリプロセッサーが呼び出されます。 また、プリプロセッサーだけに影響するコンパイラー・オプションを 指定することもできます。 一部のオプションは、プリプロセッサーの動作を大幅に変更する場合があります。

特に、以下のオプションに注意してください。

**CASE**
入力テキストを大文字に変換するかどうかを指定します。 このオプションには、次の 2 つのサブオプションがあります。

**ASIS**
入力テキストを「現状のまま」にするように指定します。

**UPPER**
入力テキストを大文字に変換するように指定します。

**DBCS**
プリプロセッサーが、テキストの置換時に DBCS を正規化するかどうかを指定します。 このオプションには、次の 2 つのサブオプションがあります。

**完全一致**
入力テキストは「現状のまま」です。プリプロセッサーは <kk.B> と <kk>B を別の名前として扱います。

**INEXACT**
入力テキストは「正規化」されます。プリプロセッサーは <kk.B> と <kk>B を同じ名前の 2 つのバージョンとして扱います。

**DEPRECATE**
エラー・メッセージを発行して非推奨にするマクロ・プロシージャーの使用箇所に、プリプロセッサーがフラグを立てるかどうかを指定します。 このオプションには、次の 1 つの式があります。

**ENTRY**
これは、*entry-name* という名前のマクロ・プロシージャーのすべての使用箇所にフラグを立てます。

**DEPRECATENEXT**
警告メッセージを発行して非推奨にするマクロ・プロシージャーの使用箇所に、プリプロセッサーがフラグを立てるかどうかを指定します。 このオプションには、次の 1 つの式があります。

**ENTRY**
これは、*entry-name* という名前のマクロ・プロシージャーのすべての使用箇所にフラグを立てます。

**固定**
FIXED 変数の処理方法を指定します。 このオプションには、次の 2 つのサブオプションがあります。

**BINARY**
FIXED 変数を BINARY として処理するように指定します。

**DECIMAL**
FIXED 変数を DECIMAL として処理するように指定します。

**ID**
インクルード・ディレクティブの名前を指定します。 最初の一続きの非ブランク文字としてのこのディレクティブで始まる行は、インクルード・ディレクティブとして扱われます。

**IGNORE | NOIGNORE**

**無視**
プリプロセッサーが特定のステートメントを無視することを指定します。 IGNORE には、1 つの必須サブオプション NOPRINT があります。 これは、MACRO プリプロセッサーで %NOPRINT ステートメントを無視する必要があることを指定します。

**NOIGNORE**
プリプロセッサーが %NOPRINT ステートメントを無視しないことを指定します。

**INCONLY | NOINCONLY**

**INCONLY**
プリプロセッサーで %INCLUDE および %XINCLUDE ステートメントのみを処理する必要があることを指定します。

**NOINCONLY**
プリプロセッサーですべてのプリプロセッサー・ステートメントを処理する必要があり、%INCLUDE および %XINCLUDE ステートメントのみではないことを指定します。

**NAMEPREFIX | NONAMEPREFIX**

**NAMEPREFIX**
プリプロセッサー・プロシージャーおよびプリプロセッサー変数の名前を、指定された文字で開始する必要があることを指定します。

**NONAMEPREFIX**
プリプロセッサー・プロシージャーおよびプリプロセッサー変数の名前を、特定の 1 文字で開始する必要がないことを指定します。

**RESCAN**
プリプロセッサーがテキストを再スキャンするときに、ID の大/小文字を扱う方法を指定します。 このオプションには、次の 2 つのサブオプションがあります。

**ASIS**
再スキャンは大文字と小文字を区別します。

**UPPER**
再スキャンは大文字と小文字を区別しません。

これらのオプションのデフォルトは、CASE(ASIS)、DBCS(INEXACT)、FIXED(DECIMAL)、NOIGNORE、NOINCONLY、NONAMEPREFIX、および RESCAN(ASIS) です。

これらのオプションの指定方法について詳しくは、 「プログラミング・ガイド」を参照してください。

# プリプロセッサー走査

プリプロセッサーは、プリプロセッサー入力の始まりから走査を開始し、 1 文字ずつ順に走査します。

デフォルトでは、CASE(UPPER) オプションが有効で、 プリプロセッサーは、入力に含まれている小文字 (コメントおよびストリング 定数内の小文字を除く) を大文字に変換します。 ただし、CASE(ASIS) サブオプションが有効になっている場合、 テキストは現状のままになります。

## プリプロセッサー・ステートメントの実行

プリプロセッサー・ステートメントは検出されたときに実行されます。

プリプロセッサー・ステートメントは以下のように使用できます。

- %DECLARE ステートメントを使用して、およびラベル接頭語として 書くことによって、プリプロセッサーを定義する。

  プリプロセッサー変数を明示的に宣言しなければ、 診断メッセージが出され、変数にはデフォルト属性 CHARACTER が 与えられます。 ただし、それ以後に実行される %ACTIVATE ステートメント内にその変数を 書かない限り、その変数は置き換えのために活動化されません。 変数は、プリプロセッサー・ステートメント内で参照できます。
- %DECLARE または %ACTIVATE ステートメントを使用して ID を 活動化することにより、置き換え処理を開始させる。これについては、582 ページの『入力テキストの実行』の項で説明します。
- %DEACTIVATE ステートメントを使用して ID を非活動化することにより、 置き換え処理を終了させる。
- %NOTE ステートメントを使用してコンパイラー・リスト内に メッセージを生成する。
- 文字ストリングをプリプロセッサー入力に入れる。
- %GOTO、%IF、%null、%DO、または %END ステートメントを使用してプリプロセッサー入力内の別の場所からプリプロセッサー走査を続行させる。

- %assignment ステートメント、または %DO ステートメントを使用してプリプロセッサー変数の 値を変更する。
- %PROCEDURE、%RETURN、および %END ステートメントを使用して プリプロセッサー・プロシージャーを定義する。 プリプロセッサー式の中の関数参照によって、プリプロセッサー・ プロシージャーを呼び出すことができます。あるいは、 関数プロシージャー名がアクティブであれば、プリプロセッサーによる 入力テキストの走査中に関数参照が検出されると、プリプロセッサー・ プロシージャーが呼び出されます。

## リスト制御ステートメントの実行

プリプロセッサー・プロシージャーに含まれていないリスト制御ステートメントは、 プリプロセッサー出力の中にコピーされます (各ステートメントごとに 1 行ずつ使われます)。

## 入力テキストの実行

アクティブ ID が新しい値で書き換えられた後の入力テキストが、プリプロセッサー出力の中にコピーされます。 無効文字 (文字定数またはコメントの一部) は、プリプロセッサー出力 内ではブランクで置き換えられます。

置換を決定するために、以下について入力テキストがスキャンされます。

- この PL/I 文字セットの一部ではない文字

  これらの文字は区切り文字として扱われ、その他の場合はこの出力に未変更のままコピーされます。

- PL/I 文字定数や PL/I コメント

  これらは変更されずに、プリプロセッサーによって入力テキストから プリプロセッサー出力に移されます。活動プリプロセッサー・ プロシージャーへの引数リスト内に書かれている場合は、この限りでは ありません。 ただし、この場合、入力マージンと出力マージンが異なっているときには、 ストリングやコメントが数行にまたがっていると、入力行数と出力行数が 一致しないことがあります。 出力は常に F フォーマットで行われ、そのマージンは 2 桁目と 72 桁目に 決まっているので、特に入力が V フォーマットであれば、上記のことが起こります。 このような場合、出力行の番号付けからも、入力行数と出力行数が 一致していないことが分かります。

- アクティブ ID

  新しい値で置き換えられる必要のある ID は、まず活動化 しておかなければ置き換えられません。 最初に、ID を %DECLARE ステートメントに挿入することで、その ID を活動化することができます。 %DEACTIVATE ステートメントを実行すれば、その ID を非活動化することが できます。その後、 %ACTIVATE ステートメントまたは %DECLARE ステートメントを実行すれば、その ID を再び活動化することができます。

  アクティブなプリプロセッサー変数の名前に一致する ID は、プリプロセッサー出力内ではその変数の値で置き換えられます。

  ID が、アクティブなプリプロセッサー関数 (プログラマー作成の関数でも 組み込み関数でもよい) の名前に一致したときは、そのプロシージャーが 呼び出されて、その呼び出しが、戻り値で置き換えられます。

  ID を活動化するとき、RESCAN オプションと NORESCAN オプションの どちらかを指定することができます。 NORESCAN オプションが適用されていると、値がただちにプリプロセッサー 出力の中に挿入されます。 RESCAN オプションが適用されていると、再走査が行われて、その値が テストされ、その値 (または値の一部) を別の値で書き換えることができるか どうかが判別されます。 別の値で置き換えられないときは、値がプリプロセッサー出力の中に 挿入されます。別の値で置き換えることができるときは、置き換えをそれ以上 できなくなるまで置き換え処理が続けられます。 このようにして、置き換え可能なものをすべて置き換えたあとにだけ、 プリプロセッサー出力への値の挿入が行われます。

置き換わる値には、% 記号、対になっていない引用符、対になっていない コメント区切り文字などが含まれていてはなりません。

プリプロセッサー・ステートメントは、通常のテキストとは別の行に存在している必要があります。 1 つの例外は、形式 %; で指定された場合のヌル・ステートメントです。 このようなヌル・ステートメントは、

置換テキストと通常のテキストを連結するために使用できます。 例えば、入力テキストが以下のようなものであるとします。

```
%dcl A char;
%A = 'B';

dcl A%C fixed bin(31);
```

プリプロセッサーは次の出力テキストを生成します。

```
dcl BC fixed bin(31);
```

プリプロセッサー入力内の最後の文字を超えて走査が試みられたときに、 走査が終了します。 その時点で、プリプロセッサー出力が完成しているので、コンパイルを 開始することができます。

## プリプロセッサー変数およびデータ要素

プリプロセッサー変数は、FIXED 属性、CHARACTER 属性、または INITIAL 属性を持つ %DECLARE ステートメントで指定します。 他の属性をプリプロセッサー変数に対して宣言することはできません。また、属性を繰り返してはなりません。 (ただし、プリプロセッサーでは他の属性を提供しています。) すべての変数は、STATIC ストレージ・クラスと同等のストレージを持ちます。

必須ではありませんが、%DECLARE 内の各変数に FIXED 属性または CHARACTER 属性を指定することは常にベスト・プラクティスです。

プリプロセッサー・データ・タイプはコード化算術データおよびストリング・データであり、以下のいずれかの属性を使用して宣言されます。

**固定**
FIXED 属性を指定して宣言したプリプロセッサー変数には、デフォルトで属性 DECIMAL(5,0) が与えられます。

FIXED(BINARY) マクロ・プリプロセッサー・オプションが有効になっている場合は、属性 BINARY(31,0) が与えられます。

いずれの場合も、初期値 0 が与えられます。

小数値はサポートされていません。

**CHARACTER**
CHARACTER 属性を付けて宣言したプリプロセッサー変数には、VARYING 属性が与えられます。

初期値「''」が与えられます。

**INITIAL**
INITIAL 属性を指定して宣言したプリプロセッサー変数は、指定された初期値を使用して割り当てられます。

プリプロセッサー・ビット変数はありませんが、 ビット定数を使用することはでき、比較演算子、連結演算子 (ビット・ オペランドとともに使用するとき)、*NOT* 演算子、および PARMSET 組み込み関数が実行されるとビット値が作られます。 %IF ステートメント内のプリプロセッサー式は、ビット値に変換されます。

プリプロセッサーによってサポートされる唯一の数値定数は、必要に応じて符号付きとなる、スケールなしの整数 (17 または -29 など) です。

プリプロセッサーでサポートされるストリング定数は文字ストリングおよびビット・ストリングのみであり、これらは 16 進表記を使用して (つまり X ストリングまたは BX ストリングとして) 指定できます。

ストリングの反復因数を使用することはできません。 ただし、COPY 組み込み関数を使用して定数を複製できます。

## プリプロセッサー参照とプリプロセッサー式

プリプロセッサー参照および式は、セクション『"式と参照"』で説明されているのと同じ方法で作成され、評価されます。

さらに、以下のコメントにも注意してください。

- プリプロセッサー式のオペランドとして指定できるのは、 プリプロセッサー変数、プリプロセッサー・プロシージャー参照、固定小数点 10 進定数、 ビット定数、文字定数、およびプリプロセッサー組み込み関数参照だけです。
- 配列を (複数のプロシージャー間で共用できるように) プリプロセッサー・プロシージャーの外部で宣言することはできますが、プロシージャーの外部から参照してはなりません (1 つの配列照会組み込み関数への最初の引数として参照する場合を除く)。
- 累乗演算記号 (**) は使えません。
- FIXED(DECIMAL) オプションを指定した場合
  - 算術演算の場合、精度 (5,0) の 10 進数演算だけが行われます。つまり、 各オペランドが精度 (5,0) の 10 進固定小数点整数値に変換された後で演算が 行われ、10 進固定小数点の演算結果も精度 (5,0) に変換されます。 例えば、3/5 という式は、0.6 ではなく 0 と評価されます。

    算術値に変換される文字値は、整数 (符号は省略可能) の形を していなければなりません。 ヌル・ストリングは 0 に変換されます。
  - 固定小数点値がビット値に変換されると、常に、長さが CEIL(3.32*5)、つまり 17 のストリングになります。
  - 固定小数点値を文字値に変換すると、結果は常に長さ 8 の ストリングになり、PL/I プログラム内で FIXED DEC(5,0) 値を CHARACTER に 変換した結果と同じ値が得られます。
- FIXED(BINARY) オプションを指定した場合
  - 算術演算の場合、精度 (31,0) の 2 進演算だけが行われます。つまり、 各オペランドが精度 (31,0) の 2 進固定小数点整数値に変換された後で演算が 行われ、2 進固定小数点の演算結果も精度 (31,0) に変換されます。 例えば、3/5 という式は、0.6 ではなく 0 と評価されます。

    算術値に変換される文字値は、整数 (符号は省略可能) の形を していなければなりません。 ヌル・ストリングは 0 に変換されます。
  - 固定小数点値がビット値に変換されると、常に、長さが 31 の ストリングになります。
  - 固定小数点値が文字値に変換されると、先行ブランクが 削除されるので、結果は可変長のストリングになります。

**関連情報**

51 ページの『式および参照』
この章では、式および参照のさまざまなタイプを説明しています。

## プリプロセッサー名の有効範囲

プリプロセッサー名の有効範囲は、その名前がどこで宣言されているかによって決まります。

プリプロセッサー・プロシージャー内で宣言されている名前の有効範囲は、そのプロシージャーです。 組み込まれたストリング内で宣言されている名前の有効範囲は、 そのストリングと、そのストリングが組み込まれた後で走査された すべての入力テキスト (ただし、同じ名前の宣言を含んでいる プリプロセッサー・プロシージャーは除く) です。 その他の名前の有効範囲は、プリプロセッサー入力全体です (ただし、 同じ名前の宣言を含んでいるプリプロセッサー・プロシージャーは除きます)。

## プリプロセッサー・プロシージャー

プリプロセッサー・プロシージャーは、プリプロセッサーで実行するアクションを指定するステートメントの集まりです。 プリプロセッサー・プロシージャーは、%PROCEDURE ステートメントと %END ステートメントで区切られます。

プロシージャーが RETURNS 属性を指定して定義されていない場合、プロシージャーには ANSWER ステートメントを使用できますが、RETURN ステートメントがあってはなりません。 逆に、プロシージャーが関数である場合は、 プロシージャーに少なくとも 1 つの RETURN ステートメントが 必要で、ANSWER ステートメントがあってはなりません。

プリプロセッサー・プロシージャー内で使用できるステートメントおよびグループは以下のとおりです。

- プリプロセッサー ANSWER ステートメント
- プリプロセッサー代入ステートメント
- プリプロセッサー CALL ステートメント
- プリプロセッサー DECLARE ステートメント
- プリプロセッサー DO グループ
- プリプロセッサー GO TO ステートメント

  (プリプロセッサー・プロシージャー内の GO TO ステートメントは、そのプロシージャーの外へ制御権を移動することはできません。)
- プリプロセッサー IF ステートメント
- プリプロセッサー ITERATE ステートメント
- プリプロセッサー LEAVE ステートメント
- プリプロセッサー・ヌル・ステートメント
- プリプロセッサー NOTE ステートメント
- プリプロセッサー RETURN ステートメント
- プリプロセッサー SELECT グループ

プリプロセッサー・プロシージャー内のプリプロセッサー・ステートメントの 頭には、% 記号を付けません。

プリプロセッサー・プロシージャーをネストすることはできません。 プリプロセッサー・プロシージャー内にプリプロセッサー ENTRY 宣言 があってはなりません。

プリプロセッサー・プロシージャーの入り口名と、そのプロシージャーへ 渡される引数リストをまとめて、関数参照 といいます。 プリプロセッサー式の中の関数参照によって、プリプロセッサー・ プロシージャーを呼び出すことができます。あるいは、 関数プロシージャー名がアクティブであれば、プリプロセッサーによる 入力テキストの走査中に関数参照が検出されると、プリプロセッサー・ プロシージャーが呼び出されます。 プリプロセッサー・プロシージャーの入り口名を %DECLARE ステートメントで 指定する必要はありません。

入り口名がアクティブであれば、そのプリプロセッサー・プロシージャーは呼び出される 前に走査される必要はありません。 ただし、プリプロセッサー入力内に存在する必要があります。 そうでないと、呼び出し点の前にストリングが組み込まれます。

プリプロセッサー出力では、関数参照とその引数リストが、 プリプロセッサー関数から返された値 (つまり、RETURN ステートメント 内のプリプロセッサー式の値) で書き換えられます。

## プリプロセッサー・プロシージャーの引数およびパラメーター

プロシージャー参照内の引数の個数と %PROCEDURE ステートメント内の パラメーターの個数とが同じである必要はありません。 引数は、パラメーター・リストとの突き合わせが行われる前に計算されます。

定位置引数の個数がパラメーターの個数よりも多い場合には、右側にある余分の引数が無視されます。 (引数が関数参照であるときは、その引数が最終的には無視される ものであっても、その関数が呼び出されて実行されます。) 関数参照によってセットされなかったパラメーターには、ゼロの値 (FIXED パラメーターの場合) または ヌル・ストリング (CHARACTER パラメーターの場合) が指定されます。

パラメーターを、関数参照によって複数回設定しないでください。 ただし、あるパラメーターの値を複数回指定した (例えば、定位置と キーワードの両方で) ときは、エラーと診断されますが、左端に設定した値が呼び出しで使われます。

関数参照がプリプロセッサー・ステートメント内にあるときは、通常の 方法で引数がパラメーターに関係付けられます。 106 ページの『プロシージャーへの引数の引き渡し』の項で説明したのと同じ方法で、仮引数が作られる ことがありますし、引数は対応するパラメーターの属性に変換されます。

関数参照が入力テキスト内に存在すると、仮引数が常に作成されます。 引数は、ストリングとして解釈されます。コンマまたは右括弧を区切り文字として使用してください。 ただし、対になっている括弧、対になっている単一引用符、または対に なっているコメント区切り文字の間にあるコンマや右括弧は、区切り文字の働きをしません。 例えば、位置引数のリスト (A(B,C),D) には、文字列 A(B,C) と文字列 D の 2 つの引

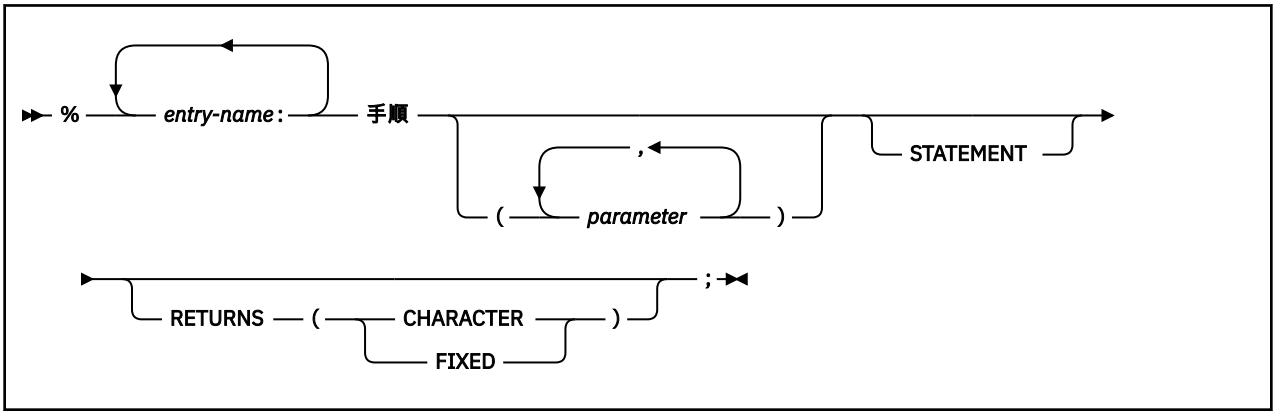
数があります。 引数内のブランク (先行ブランクと末尾ブランクを含む) は有効ですが、そのようなブランクが行の終わりまで続き、引用符またはコメント区切り文字で囲まれていない場合は、1 つのブランクに置き換えられます。

入力テキスト内で関数参照が検出されると、各引数が走査され、置き換えが 可能かどうか調べられます。この置換アクティビティーは、その関数に渡された引数の個数に影響しません。 置き換え処理によって引数内にコンマや括弧が入れられても、それらは 区切り文字とは見なされず、単に引数内の文字と見なされます。 キーワード呼び出しが使われている場合、そのキーワード自体は置換アクティビティーに適しません。 置き換えがすべて終わると、その結果の各引数が、その関数入り口名を持つ プリプロセッサー・プロシージャー・ステートメント内の対応する パラメーターの属性で指定されるタイプに変換されます。

## %PROCEDURE ステートメント

%PROCEDURE ステートメントは、%END ステートメントと対になっており、 プリプロセッサー・プロシージャーを区切る働きをします。

%PROCEDURE ステートメントの構文は以下のとおりです。




**省略形**: %PROC

**パラメーター**
関数プロシージャーのパラメーターを指定します。

**STATEMENT**
入力テキスト内で参照が行われ、STATEMENT オプションがある場合は、以下の規則が適用されます。

- 引数を定位置引数リスト内に指定することも、キーワード参照によって 指定することもできます。
- 参照の終わりは、セミコロンで示します。 置き換えが行われると、このセミコロンは保持されません。

例えば、`%FIND:PROC(A,B,C) STATEMENT...;` が先頭にあるプリプロセッサー・プロシージャーは、次の形式の参照によってプリプロセッサー式から呼び出す必要があります。

```
FIND(arg1,arg2,arg3)
```

関数参照が入力テキスト内にあるときは、下記のどの形式 (または類似の 形式) で参照しても、このプロシージャーを呼び出すことができ、すべて 同じ結果になります。

```
FIND(X,Y,Z);
FIND B(Y) C(Z) A(X);
FIND(X) C(Z) B(Y);
FIND(,Y,Z) A(X);
```

**RETURNS**
関数プロシージャーから返される値の属性を指定します。 以下の属性の 1 つを指定する必要があります。

- CHARACTER

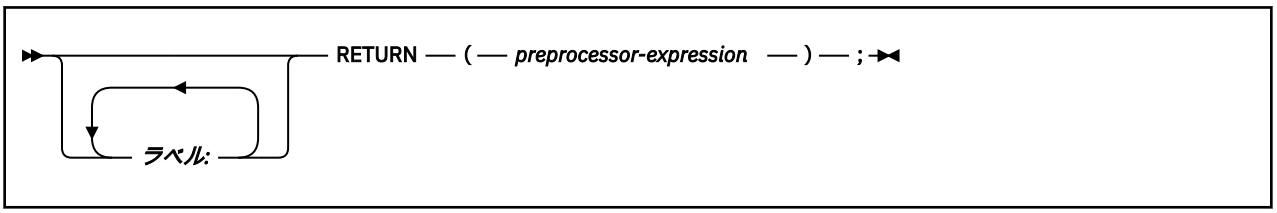
- FIXED

## プリプロセッサー RETURN ステートメント

プリプロセッサー RETURN ステートメントは、プロシージャーに RETURNS 属性がある場合に限り、プリプロセッサー・プロシージャー内でのみ使用できます。 このステートメントは、プリプロセッサー・プロシージャーの呼び出し点に値と制御を返します。

プリプロセッサー RETURN ステートメントでは先頭に % を使用できません。 RETURNS 属性を持つそれぞれのプリプロセッサー・プロシージャー内には、 1 つ以上の RETURN ステートメントがなければなりません。

プリプロセッサー関数プロシージャーから呼び出し点に返される値は、 そのプロシージャー内の RETURN ステートメント内のプリプロセッサー式 (preprocessor-expression) で 指定された値です。 プリプロセッサー RETURN ステートメントの構文は以下のとおりです。

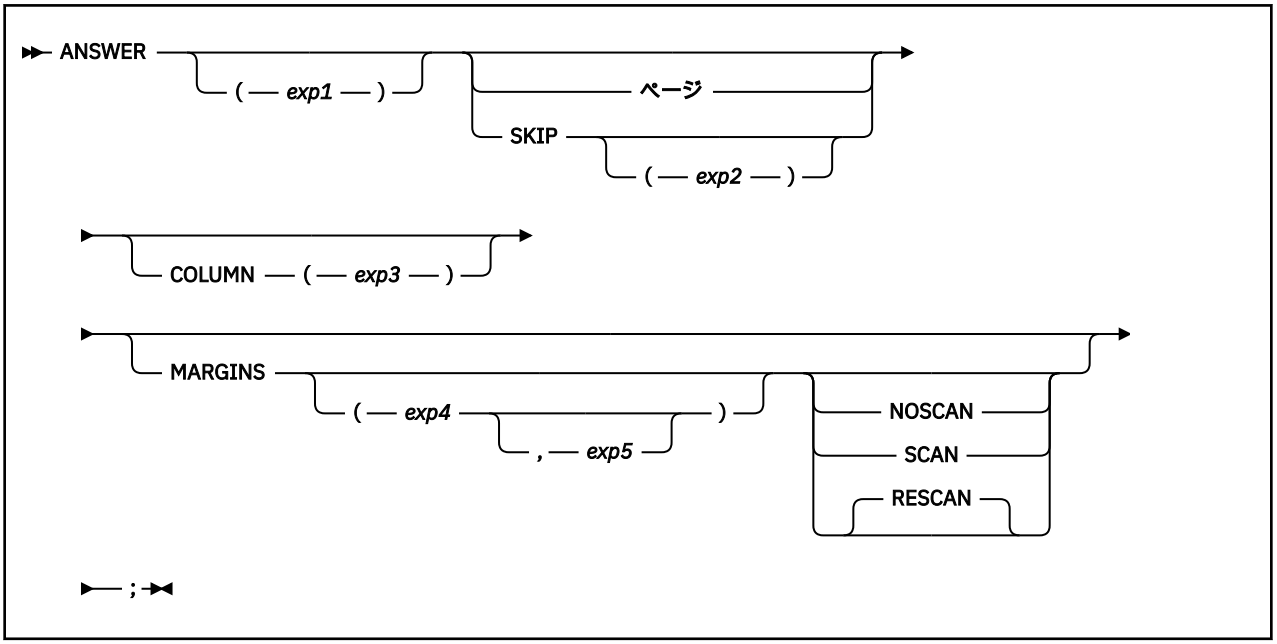



**preprocessor-expression**
この値は、%PROCEDURE ステートメント内に指定されている RETURNS 属性に合わせて変換されてから、呼び出し点に返されます。

## プリプロセッサー ANSWER ステートメント

プリプロセッサー ANSWER ステートメントは、 RETURNS 属性のないプリプロセッサー・プロシージャー内で だけ使用できます。

ANSWER ステートメントは、テキストを作成するか、またはほかの プリプロセッサー・プロシージャーを呼び出します。あるいは、その両方を行います。 応答されたテキストは、ソース・テキスト内のプリプロセッサー ・プロシージャーの呼び出しを置換します。 ANSWER ステートメントは 1 つのプリプロセッサー・プロシージャー内で何回でも使うことができます。




**省略形 :** ANSWER の場合は ANS、COLUMN の場合は COL、 MARGINS の場合は MAR です。

***exp1***
ANSWER テキストを表す文字式を表します。 ANSWER テキストは、単一文字ストリング定数か、または合成された プリプロセッサー式のいずれかにすることができます。

テキストが式の場合、式の計算は通常の方法で行われ、結果は単一文字ストリングに変換されます。

SCAN または RESCAN が有効な場合には、置換とプリプロセッサー・プロシージャー呼び出しのために、その文字ストリングが走査されます。 この置換はそのプリプロセッサー・プロシージャーの範囲内で行われ、応答 されたテキストが戻される範囲内で行われることは**ありません**。 それから、応答されたテキストがソース・テキスト内のプリプロセッサー呼び出しの点に挿入されます。 テキストがソース・テキストに戻されたあとは、いかなる置換アクティビティ ーが行われても、そのテキストが走査されることはありません。

ストリング内の置換アクティビティーは、ソース・テキストの走査と置換のときに用いられるのと同じ規則に従って行われます。 588 ページの『例』を参照してください。

**PAGE**
%PAGE ディレクティブを生成することによって、応答テキストを出力ソースの新規のページに強制的に入れるようにします。

**SKIP**
応答テキストを出力ソースの新規の行に強制的に入れるようにします。 *exp2* は、スキップされる行数を指定している算術式を表します。 *exp2* を省略した場合、デフォルトは 1 です。

**COLUMN**
ソース・プログラム行に応答テキストが入れられるときの開始カラムを指定します。 *exp3* の値は、 ソース・プログラム行内の 応答テキストの開始カラム番号を示す算術式を表します。

**MARGINS**
出力テキストが出力レコード内に置かれる場所を指定します。 *exp4* の値は、出力テキストの左マージンを示す算術式を表します。 *exp5* の値は、 出力テキストの右マージンを示す算術式を表します。

*exp5* に指定される値は、 MACLMAR (左マージン) 組み込み関数と MACRMAR (右マージン) 組み込み関数によって 戻される範囲内になければなりません。

ANSWER ステートメントの MARGINS オプションを指定しない場合の、 デフォルトの値は MARGINS(MACLMAR,MACRMAR) です。 オペランドを指定しないで MARGINS オプションを指定する場合は、 デフォルトの値は MARGINS(MACCOL,MACRMAR) です。

**RESCAN**
応答テキストに含まれるプリプロセッサー参照によって置き換えられる可能性があるため、応答テキストを再スキャンする必要があることを指定します。

**SCAN**
応答テキストに含まれるプリプロセッサー参照によって置き換えられる可能性があるため、応答テキストを再スキャンする必要があることを指定します。

**NOSCAN**
応答テキストを再スキャンしないことを指定します。

同じプリプロセッサー・プロシージャーの中で、式を指定した RETURN ステ ートメントと ANSWER ステートメントの両方を一緒に使うことはできません。

**注 :** 式の中でマクロが呼び出され、そのマクロが値を返さない場合、マクロ・プリプロセッサーは、マクロがヌル・ストリングを返したかのように動作します。

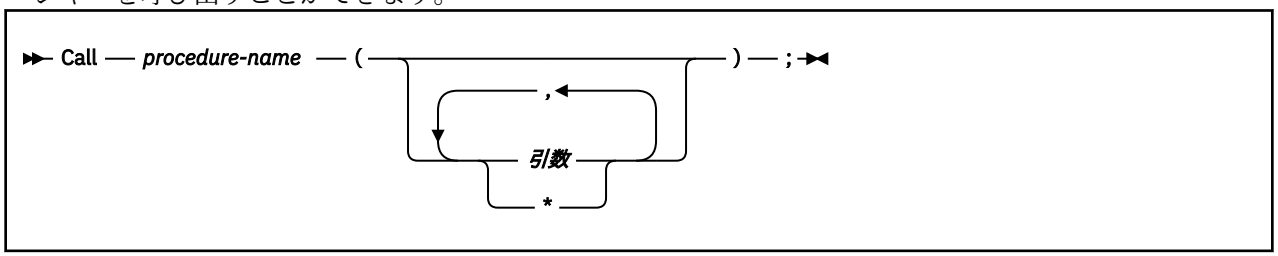
**例**

```
%dcl (Expression, Single_string) entry;
%dcl (Deactivated_macro, Statement_function) entry;
%dcl Deactivated_variable character;
%deact Deactivated_variable, Deactivated_macro;
%Deactivated_variable = '** value of deactivated variable **';

%Deactivated_macro: procedure returns( character );
   return( '** value of deactivated macro **' );
%end;
```

```
%Statement_function: procedure( key1 ) stmt returns( fixed );
   dcl key1 fixed;
   return( key1 + key1 );
%end;

%Expression: procedure;
   ANS( Counter ) skip;
   ANS( Deactivated_macro ) skip;
   ANS( Deactivated_variable ) skip;
   /* The following is invalid:                                  */
   /* ANS( Statement_function Key1(7););                         */
%end;

%Single_string: procedure;
   ANS( 'Counter' ) skip;
   ANS( 'Deactivated_macro' ) skip;
   ANS( 'Deactivated_variable' ) skip;
   ANS( 'Statement_function Key1( 7 );' ) skip;
%end;


Expression              /* Generates:                             */
                        /*   00001                                */
                        /*   ** value of deactivated macro **     */
                        /*   ** value of deactivated variable ** */

Single_string           /* Generates:                             */
                        /*   Counter                              */
                        /*   Deactivated_macro                    */
                        /*   Deactivated_variable                 */
                        /*         14                             */
```

## プリプロセッサー CALL ステートメント

プリプロセッサーの CALL ステートメントを使用すると、MACRO プロシージャーは、別の MACRO プロシージャーを呼び出すことができます。




***procedure-name***
呼び出すプロシージャーの名前を指定します。 プロシージャーには、RETURNS 属性または STATEMENT オプションが含まれていてはなりません。

**argument**
呼び出されたサブルーチンに渡されるエレメント、要素式、または集合を指定します。 106 ページの『プロシージャーへの引数の引き渡し』を参照してください。

# プリプロセッサー組み込み関数

プリプロセッサー組み込み関数 と呼ばれる 1 組の事前定義関数の 1 つを、関数参照で呼び出すことができます。 これらの組み込み関数の呼び出し方法は、プログラマー定義の関数の呼び出し方法と 同じですが、正しい数の引数を指定しなければならない点だけが異なります。

プリプロセッサー組み込み関数は以下のとおりです。

| | | | |
|---|---|---|---|
| COLLATE | INDEX | 最大 | シスパーム |
| COMMENT | LBOUND | MIN | SYSPOINTERSIZE |
| COMPILEDATE | LENGTH | PARMSET | SYSTEM |
| COMPILETIME | LOWERCASE | QUOTE | SYSVERSION |
| COPY | MACCOL | REPEAT | TRANSLATE |
| COUNTER | MACLMAR | SUBSTR | TRIM |
| DIMENSION | MACNAME | SYSDIMSIZE | UPPERCASE |
| HBOUND | MACRMAR | SYSOFFSETSIZE | VERIFY |

プリプロセッサーは、入力テキスト内のプリプロセッサー組み込み関数参照を、 その組み込み関数名がアクティブであるときにだけ、実行します。 %DECLARE ステートメントまたは %ACTIVATE ステートメントを使用すれば、組み込み関数を 活動化することができます。

プリプロセッサー・ステートメント内のプリプロセッサー組み込み関数名は、 別の意味を持つように宣言されていない限り、組み込み関数として常に アクティブになっています。

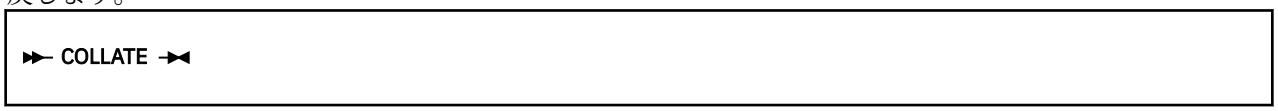
プリプロセッサー組み込み関数名をユーザー定義のプリプロセッサー・ プロシージャーの名前として使用している場合は、その名前を参照すると、 組み込み関数ではなくそのプロシージャーを参照することになります。 そのような場合に、あるプリプロセッサー・プロシージャー内でその 組み込み関数を使用するときは、BUILTIN 属性を指定してその ID を 宣言しておかなければなりません。

次のプリプロセッサー組み込み関数の場合、引数は不要であり、 ヌル引数を指定してはなりません。

| | | | |
|---|---|---|---|
| COLLATE | MACCOL | SYSDIMSIZE | SYSTEM |
| COMPILEDATE | MACLMAR | SYSOFFSETSIZE | SYSVERSION |
| COMPILETIME | MACNAME | シスパーム | |
| COUNTER | MACRMAR | SYSPOINTERSIZE | |

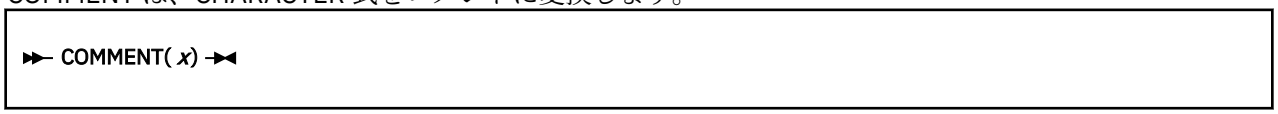
## COLLATE

COLLATE は、あり得る 256 の文字値を照合順序で 1 つずつ含む、 長さ 256 の CHARACTER ストリングを戻します。




## COMMENT

COMMENT は、CHARACTER 式をコメントに変換します。




**x**

コメントに変換される式。

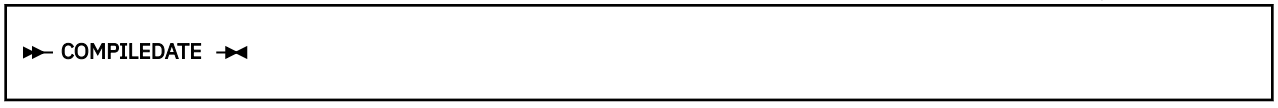
*x* のタイプは CHARACTER でなければならず、 それ以外の場合はこのタイプに変換されます。

*x* は /* と */ で囲まれます。

*x* に複合記号 /* または */ が含まれている場合、それらはそれぞれ /> および </ に置き換えられます。

## COMPILEDATE

COMPILEDATE は、コンパイル日時が入った 長さ 17 の CHARACTER ストリングを戻します。




COMPILEDATE から返されるストリングの形式は以下のとおりです。

**yyyy**
現在の年

**mm**
現在の月

**dd**
現在の日

**hh**
現在の時

**mm**
現在の分

**ss**
現在の秒

**ttt**
現在のミリ秒

時間帯と精度は、システムよって決まります。

### 例

次の例は、プログラムの実行中に、COMPILEDATE から返されたストリングを印刷する方法を示しています。

```
%DECLARE COMP_DATE CHAR;
%COMP_DATE=QUOTE(COMPILEDATE);
PUT EDIT (COMP_DATE) (A);
```

## COMPILETIME

COMPILETIME は、コンパイル日時が入った 長さ 18 の CHARACTER ストリングを戻します。

COMPILETIME から返されるストリングの形式は以下のとおりです。

**DD**
日付

**.**
ピリオド

**MMM**
月 (JAN、FEB、MAR などの形式)

**.**
ピリオド

**YY**
年

**b**
ブランク

**HH**
時

**.**
ピリオド

**MM**
分

**.**
ピリオド

**SS**
Second

「日」の先行ゼロはブランクで置き換えられます。その他の先行ゼロは 消去されません。

時刻機構が使用可能でない場合は、返されるストリングの最後の 8 文字は 00.00.00 となります。

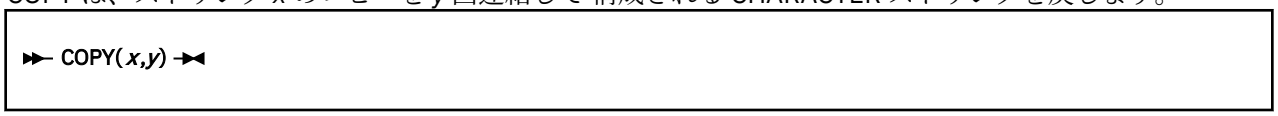
**例**

次の例は、プログラムの実行中に、COMPILETIME から返された ストリングを印刷する方法を示しています。

```
%DECLARE COMP_TIME CHAR;
%COMP_TIME=QUOTE(COMPILETIME);
PUT EDIT (COMP_TIME) (A);
```

# COPY

COPY は、ストリング *x* のコピーを *y* 回連結して 構成される CHARACTER ストリングを戻します。




**x**
式。

*x* のタイプは CHARACTER でなければならず、 それ以外の場合はこのタイプに変換されます。

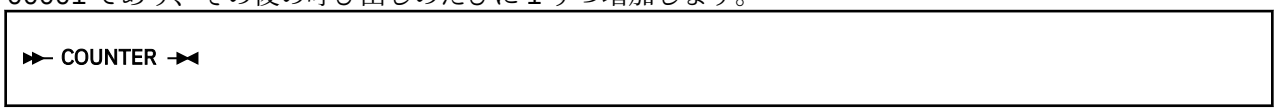
**y**
繰り返し回数を指定する式。 *y* のタイプは FIXED でなければならず、 それ以外の場合はこのタイプに変換されます。

*y* は負であってはなりません。

*y* がゼロの場合は、結果はヌル・ストリングです。

# COUNTER

COUNTER は、10 進数が入った長さ 5 の CHARACTER ストリングです。 最初の呼び出しで返される数値は 00001 であり、その後の呼び出しのたびに 1 ずつ増加します。

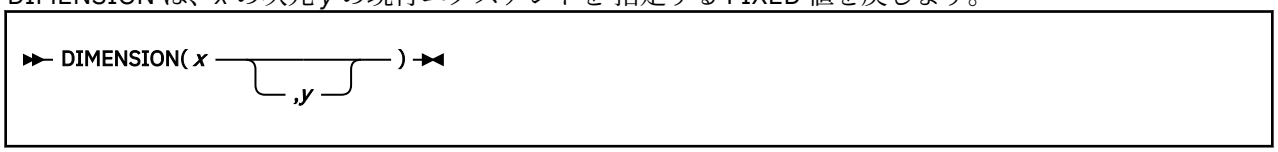



COUNTER が 99999 回呼び出されると、次回の呼び出し時に 診断メッセージが出され、00000 が返されます。 その直後の呼び出しは、最初の呼び出しとして扱われます。

COUNTER 組み込み関数は、固有の名前を生成したり、数を数えたりするときに 使用することができます。

# DIMENSION

DIMENSION は、*x* の次元 *y* の現行エクステントを 指定する FIXED 値を戻します。




**省略形**: DIM

**x**
配列参照。 *x* の次元数は、*y* よりも小さくてはなりません。

**y**
*x* の特定の次元を指定する式。

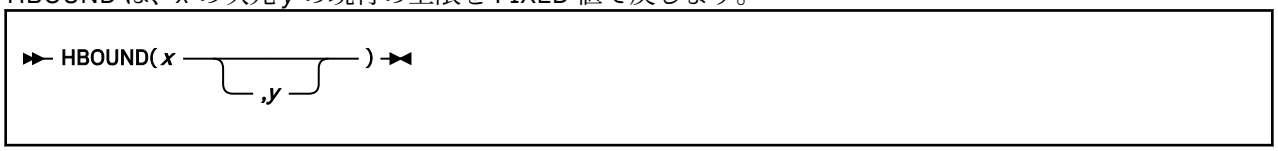
*y* のタイプは FIXED でなければならず、 それ以外の場合はこのタイプに変換されます。

*y* は 1 以上でなければなりません。 *y* が指定されていない場合、デフォルトは 1 です。

配列が 1 次の場合のみ、*y* を省略できます。

## HBOUND

HBOUND は、*x* の次元 *y* の現行の上限を FIXED 値で戻します。




**x**

配列参照。 *x* の次元数は、*y* よりも小さくてはなりません。

**y**

*x* の特定の次元を指定する式。

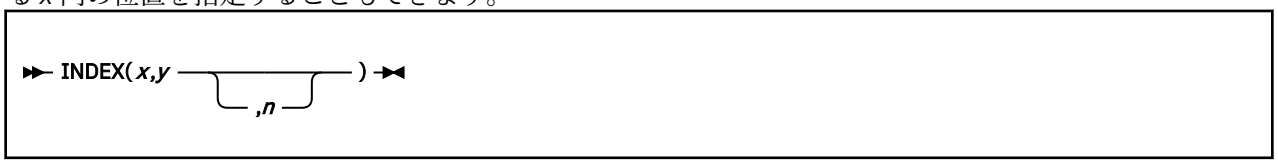
*y* のタイプは FIXED でなければならず、 それ以外の場合はこのタイプに変換されます。

*y* は 1 以上でなければなりません。 *y* が指定されていない場合、デフォルトは 1 です。

配列が 1 次の場合のみ、*y* を省略できます。

## INDEX

INDEX は、*y* と等しいサブストリングの *x* 内の開始位置を示す FIXED 値を返します。 また、処理を開始する *x* 内の位置を指定することもできます。




**x**

検索される式。

*x* のタイプは CHARACTER でなければならず、 それ以外の場合はこのタイプに変換されます。

**y**

検索のターゲット式。

*y* のタイプは CHARACTER でなければならず、 それ以外の場合はこのタイプに変換されます。

**n**

*n* は、処理を開始する *x* 内の位置を指定します。

*n* のタイプは FIXED でなければならず、 それ以外の場合はこのタイプに変換されます。

*x* 内で *y* が発生しない場合、または *x* か *y* のどちらかが長さゼロがある場合は、値ゼロが戻されます。

*n* は 0 より大きく、かつ 1 + LENGTH(*x*) を超えない ことが必要です。

*n* = LENGTH(*x*) + 1 の場合は、結果はゼロです。

## LBOUND

LBOUND は、*x* の次元 *y* の現行の下限を FIXED 値で戻します。

**x**

配列参照。 *x* の次元数は、*y* よりも小さくてはなりません。

**y**
*x* の特定の次元を指定する式。

*y* のタイプは FIXED でなければならず、 それ以外の場合はこのタイプに変換されます。

*y* は 1 以上でなければなりません。 *y* が指定されていない場合、デフォルトは 1 です。

配列が 1 次の場合のみ、*y* を省略できます。

## LENGTH

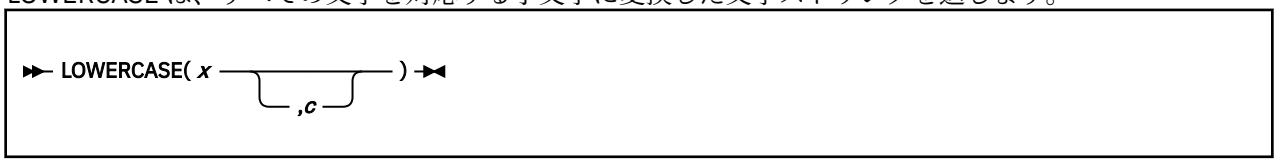
LENGTH は、指定の文字式の現在の長さを表す FIXED 値を戻します。

**x**
式。

*x* のタイプは CHARACTER でなければならず、 それ以外の場合はこのタイプに変換されます。

## LOWERCASE

LOWERCASE は、すべての文字を対応する小文字に変換した文字ストリングを返します。




**x**
式。

*x* のタイプは CHARACTER でなければならず、 それ以外の場合はこのタイプに変換されます。

**c**
式。 小文字にするコード・ページを指定します。

LOWERCASE(*x*) は、TRANSLATE(*x, 'a...z', 'A...Z'*) と同等で、LOWERCASE(*x*, *c*) は TRANSLATE(*x, lowerc, upperc*) と同等です。 *lowerc* と *upperc* の値は、コード・ページ *c* の値によって決定されます。 LOWERCASE(*x*, *c*) を使用すると、英字「A...Z」が「a...z」に変換されるだけでなく、大文字の Ä-umlaut('4a'x) のような文字も小文字の ä-umlaut('c0'x) に変換されます。

例えば、Lower_01141 は次のように宣言できます。

```
dcl lower_01141 char
 value( (
             '8182838485868788'8991929394959697'x
           || '9899A2A3A4A5A6A7A8A9424445464748'x
           || '4951525354555657'586A708C8D8E9CC0'x
           || 'CBCDCECFD0DBDDDE'x
       ) );
```

付録に、サポートされている *c* の値の *lowerc* と *upperc* の値がリストされています。 詳しくは、546 ページの『TRANSLATE』 および 615 ページの『付録 A 制限値』 を参照してください。

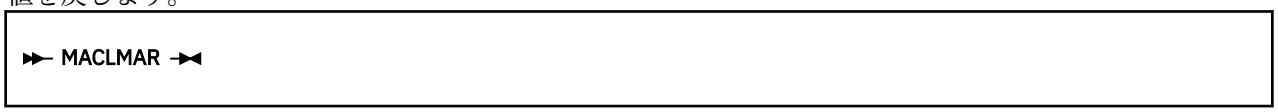
## MACCOL

MACCOL は、最外部のマクロ呼び出しが入っているソース・テキスト内で、 そのマクロ呼び出しが始まっているカラムを表す FIXED 値を戻します。

戻される値は、ネストされたマクロ呼び出しによって影響されません。

## MACLMAR

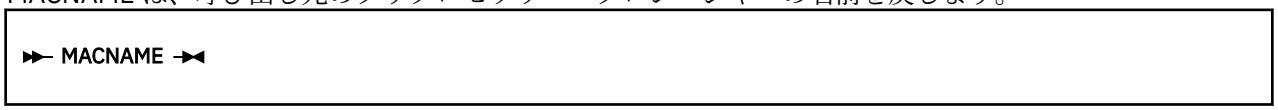
MACLMAR は、MARGINS コンパイラー・オプションにおける左ソース・マージンのカラム番号 を表す FIXED 値を戻します。




「プログラミング・ガイド」の MARGINS オプションに関する情報を参照してください。

## MACNAME

MACNAME は、呼び出し元のプリプロセッサー・プロシージャーの名前を戻します。




プリプロセッサー・プロシージャーの外部からの MACNAME の呼び出しは無効です。

## MACRMAR

MACRMAR は、MARGINS コンパイラー・オプションの右ソース・マージンのカラム番号を 表す FIXED 値を戻します。

「プログラミング・ガイド」の MARGINS オプションに関する情報を参照してください。

## MAX

MAX は、複数の式の集合から最大値を戻します。

**x および y**
式。

引数はすべて FIXED でなければならず、FIXED でないものは すべて FIXED に変換されます。

## MIN

MIN は、複数の式の集合から、最小値を戻します。

**x および y**
式。

引数はすべて FIXED でなければならず、FIXED でないものは すべて FIXED に変換されます。

## PARMSET

PARMSET は、指定のパラメーターがプロシージャーの 呼び出し時に設定されたかどうかを示す BIT 値を戻します。

**x**
そのプリプロセッサー・プロシージャーのパラメーターでなければなりません。

PARMSET 組み込み関数は、プリプロセッサー・プロシージャー内でだけ 使用することができます。

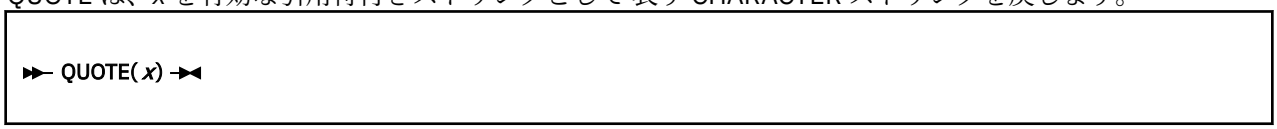
PARMSET から返される値は、そのプロシージャーを呼び出した関数参照に よってパラメーター *x* が明示的にセットされている場合はビット値 '1'B であり、セットされていない (つまり、プリプロセッサー式の中の関数参照で対応する引数が指定されなかったか、または入力テキスト内の 関数参照で対応する引数がヌル・ストリングであった) 場合は、ビット値 '0'B です。

対応する引数が関数参照内に指定されていても、その関数参照が別の プリプロセッサー・プロシージャー内にある場合は、PARMSET から '0'B が返されることがあります。次のようになります。

- その引数そのものが呼び出しプロシージャーのパラメーターでないときは、 値 '1'B が返されます。
- その引数が呼び出しプロシージャーのパラメーターであるときは、 呼び出しプロシージャーそのものが呼び出されたときの指定された パラメーターについての値が返されます。

## QUOTE

QUOTE は、*x* を有効な引用符付きストリングとして 表す CHARACTER ストリングを戻します。



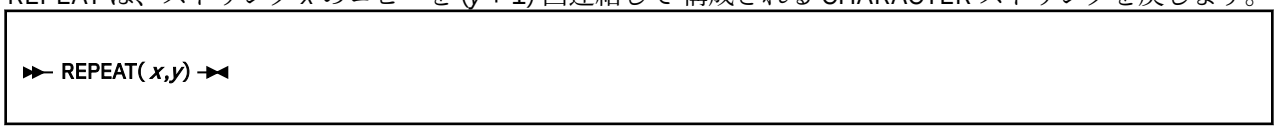

**x**
引用符付きストリングに変換される式。

*x* のタイプは CHARACTER でなければならず、 それ以外の場合はこのタイプに変換されます。

*x* に一重引用符が入っている場合には、 各引用符が、それぞれ 2 つずつ連続する一重引用符に置き換えられます。

## REPEAT

REPEAT は、ストリング *x* のコピーを (*y* + 1) 回連結して 構成される CHARACTER ストリングを戻します。




**x**
式。

*x* のタイプは CHARACTER でなければならず、 それ以外の場合はこのタイプに変換されます。

**y**
繰り返し回数を指定する式。 *y* のタイプは FIXED でなければならず、 それ以外の場合はこのタイプに変換されます。

*y* は負であってはなりません。

*y* がゼロの場合は、結果は *x* です (必要に応じ、 文字に変換される)。

## SUBSTR

SUBSTR は、サブストリングを戻します。このサブストリングは、*x* の *y* および *z* によって指定されます。

**x**
サブストリングが抜き出されるストリングを指定する式。

*x* のタイプは CHARACTER でなければならず、 それ以外の場合はこのタイプに変換されます。

**y**
*x* の中のサブストリングの開始位置を指定する式。

*y* のタイプは FIXED でなければならず、 それ以外の場合はこのタイプに変換されます。

**z**
*x* の中のサブストリングの長さを指定する式。

*z* のタイプは FIXED でなければならず、 それ以外の場合はこのタイプに変換されます。

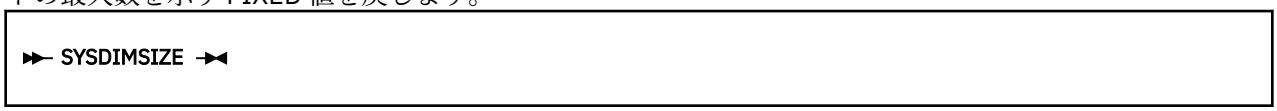
*z* がゼロの場合は、ヌル・ストリングが戻されます。 *z* が省略される場合は、戻されるサブストリングは *x* の位置 *y* から *x* の終わりまでです。

*z* は負でなければならず、*y* および *z* の値は、サブストリング全体が *x* の現在の長さに入るような値 でなければなりません。

*y* = LENGTH(*x*)+1 かつ *z* = 0 の場合は、ヌル・ストリングが返されます。

## SYSDIMSIZE

SYSDIMSIZE は、CMPAT コンパイラー・オプションの下で使用できる配列の索引を入れるのに必要なバイトの最大数を示す FIXED 値を戻します。

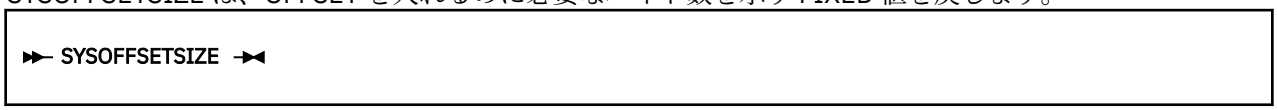



考えられる戻り値は以下のとおりです。

- CMPAT(V2) と CMPAT(LE) の場合は 4
- CMPAT(V3) の場合は 8

## SYSOFFSETSIZE

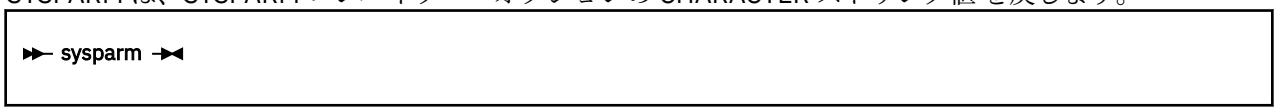
SYSOFFSETSIZE は、OFFSET を入れるのに必要なバイト数を示す FIXED 値を戻します。




現時点では、SYSOFFSETSIZE は 4 を戻します。

## シスパーム

SYSPARM は、SYSPARM コンパイラー・オプションの CHARACTER ストリング値 を戻します。




戻された値は大文字に変換されることはありません。コンパイラー・ オプションに指定されたとおりの値が戻されます。 「プログラミング・ガイド」の SYSPARM コンパイラー・オプションに関する情報を参照してください。

SYSPARM を使用すると、プログラムの外部情報はソース・プログラムを変更しないでアクセスできます。

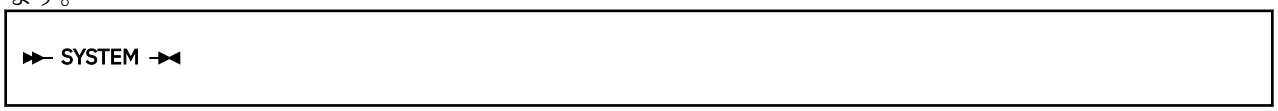
## SYSPOINTERSIZE

SYSPOINTERSIZE は、POINTER を入れるのに必要なバイト数を示す FIXED 値を戻します。

現時点では、SYSPOINTERSIZE は 4 を戻します。 しかし、LP(64) オプションでは、SYSPOINTERSIZE は 8 を戻します。

## SYSTEM

SYSTEM は、有効な SYSTEM コンパイラー・オプションの値が 入っている CHARACTER ストリングを戻します。




**注 :** 返される値には、先行ブランクと末尾ブランクが含まれている場合があります。 テストしやすくするために、TRIM 組み込み関数をその値に適用することができます。

「プログラミング・ガイド」の SYSTEM コンパイラー・オプションに関する情報を参照してください。

## SYSVERSION

SYSVERSION は、製品名、バージョン、リリース、およびモディフィケーショ ン・レベルを含む CHARACTER ストリングを戻します。

SYSVERSION から返される結果は、長さが 22 で、以下のいずれかの形式のストリングです。 各ストリングは、長さが 22 になるように、右側にブランクが埋め込まれます。

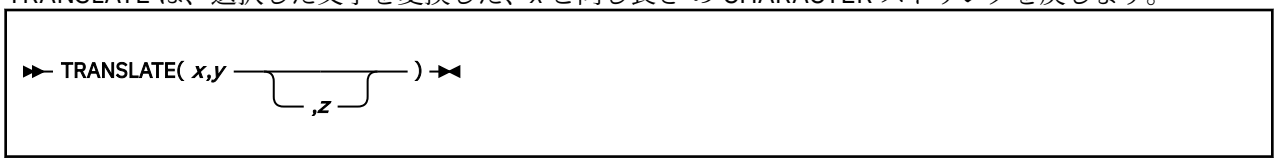
**z/OS の場合**
PL/I for z/OS V*x*.R*y*.M*z*

**AIX の場合**
PL/I for AIX *x.y*

**Windows の場合**
PL/I for Win* *x.y*

## TRANSLATE

TRANSLATE は、選択した文字を変換した、*x* と同じ長さ の CHARACTER ストリングを戻します。




**x**
変換できる文字を検索する対象を指定する式。

*x* のタイプは CHARACTER でなければならず、 それ以外の場合はこのタイプに変換されます。

**y**
文字の変換値を含む式。

*y* のタイプは CHARACTER でなければならず、 それ以外の場合はこのタイプに変換されます。

**z**
変換される文字を含む式。 *z* を省略した場合、デフォルトは COLLATE です。

*z* のタイプは CHARACTER でなければならず、 それ以外の場合はこのタイプに変換されます。

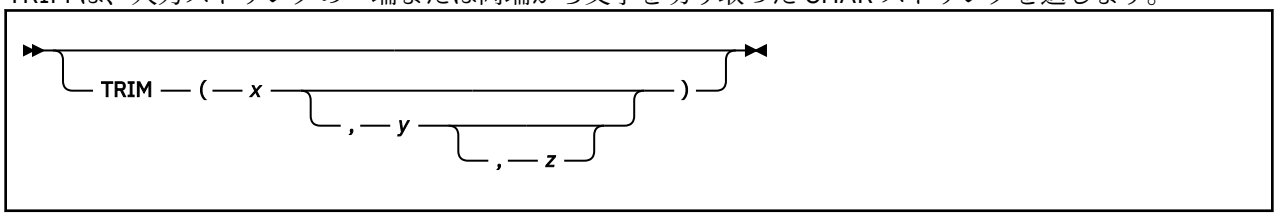
TRANSLATE は、*x* の各文字を次のように処理します。

*x* の中にある文字が *z* に検出される場合は、*y* にあ る *z* に対応する文字が結果にコピーされます。それ以外の場合は、*x* の文字が結果にコピーされます。 *z* に複数の同一文字が入っている場合は、左端のオカレンスが使われます。

*y* は、*z* の長さと一致するように、右がブランクで埋められるか、切り捨てられます。

## TRIM

TRIM は、入力ストリングの一端または両端から文字を切り取った CHAR ストリングを返します。




## UPPERCASE

UPPERCASE は、すべての文字を対応する大文字に変換した文字ストリングを返します。

**x**
式。 必要に応じ、*x* は文字に変換されます。

**c**
式。 大文字にするコード・ページを指定します。

UPPERCASE(*x*) は、TRANSLATE(*x, 'A...Z '、'a... z'*) と同等で、UPPERCASE(*x*, *c*) は TRANSLATE(*x, upperc, lowerc*) と同等です。 *upperc* と *lowerc* の値は、コード・ページ *c* の値によって決定されます。 UPPERCASE(*x*, *c*) を指定すると、英字「a...z」が「A...Z」に変換されるだけでなく、小文字の ä-umlaut('c0'x) のような文字も大文字の Ä-umlaut('4a'x) に変換されます

例えば、Upper_01141 は次のように宣言できます。

```
dcl upper_01141 char
  value( (
              'C1C2C3C4C5C6C7C8C9D1D2D3D4D5D6D7'x
           || 'D8D9E2E3E4E5E6E7E8E9626465666768'x
           || '6971727374757677'78E080ACADAE9E4A'x
           || 'EBEDEEEF5AFBFDFE'x
       ) );
```

付録に、サポートされている *c* の値の *upperc* と *lowerc* の値がリストされています。 詳しくは、546 ページの『TRANSLATE』 および 615 ページの『付録 A 制限値』 を参照してください。

## VERIFY

VERIFY は、*y* にない左端の文字の *x* 内での位置を示す FIXED 値を返します。 また、この関数を使用すると、処理を開始する *x* 内での位置を指定できます。

**x**
式。

*x* のタイプは CHARACTER でなければならず、 それ以外の場合はこのタイプに変換されます。

**y**
式。

*y* のタイプは CHARACTER でなければならず、 それ以外の場合はこのタイプに変換されます。

**n**
*n* は、処理を開始する *x* 内の位置を指定します。

*n* のタイプは FIXED でなければならず、 それ以外の場合はこのタイプに変換されます。

*x* のすべての文字が *y* にある場 合は、値ゼロが戻されます。 *x* がヌル・ストリングの場合は、値ゼロが戻されます。 *x* がヌル・ストリングでなく、*y* がヌル・ストリングの場合は、 *n* の値が戻されます。 *n* のデフォルトは 1 です。

*n* は 0 より大きく、かつ 1 + LENGTH(*x*) を超えない ことが必要です。

*n* = LENGTH(*x*) + 1 の場合は、結果はゼロです。

# プリプロセッサー・ステートメント

このセクションでは、プリプロセッサー・ステートメントをアルファベット順に示し、各ステートメントについて説明します。

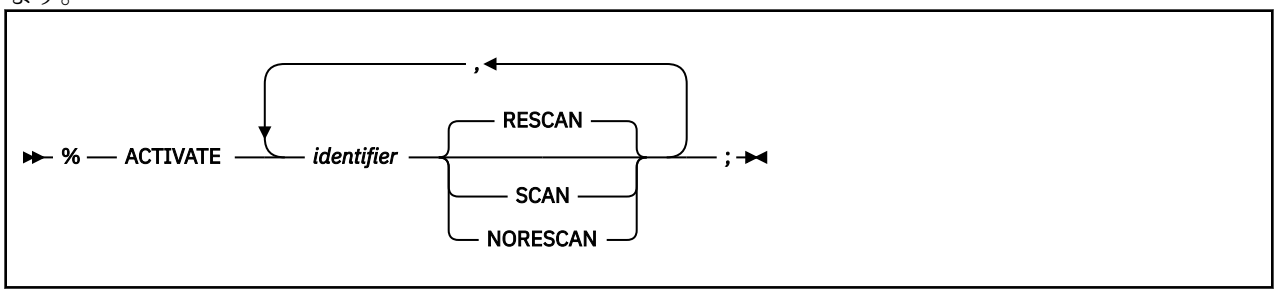
プリプロセッサー・ステートメント内でブランクを使用できるところで あればどこにでも、コメントを記入できます。 このようなコメントは、プリプロセッサー出力テキストには挿入されません。

すべてのプリプロセッサー・ステートメントにラベルを付けることができます。

%CONTROL ステートメントはサポートされません。 このキーワードを記述してもエラーにはなりませんが、無視されます。

## %ACTIVATE ステートメント

%ACTIVATE ステートメントは、ID をアクティブにして、それに置換適格性を与えます。 それ以後、その ID がアクティブになっている間は、入力テキスト内で その ID が検出されると、置き換え処置が開始されます。




**省略形:** %ACT

**識別子**
プリプロセッサー変数名、プリプロセッサー・プロシージャー名、 またはプリプロセッサー組み込み関数を指定します。

配列変数を参照する ID は指定できません。

**RESCAN**
結果が出力に入れられる前に、 すべての活動 ID を置換するのに必要な回数だけ この ID が置換されることを指定します。

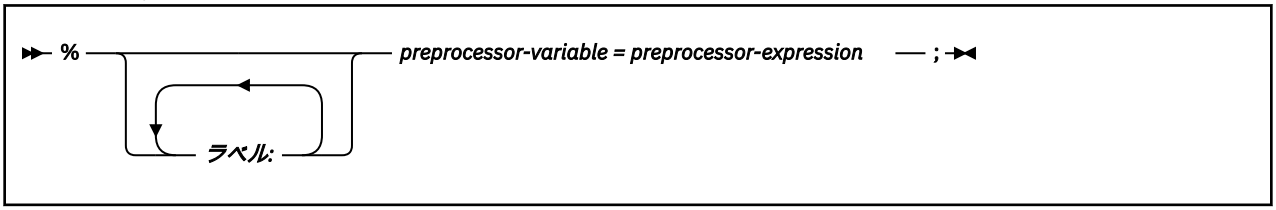
**SCAN**
結果が出力に入れられる前に、この ID が 1 回だけ置換されることを指定します。

**NORESCAN**
SCAN の同義語です。

すでにアクティブの ID に対して %ACTIVATE を使用しても 特に影響はありませんが、 走査状況が変わる可能性はあります。

## %assignment ステートメント

%assignment ステートメントは、プリプロセッサー式を評価して、その結果をプリプロセッサー変数に割り当てます。

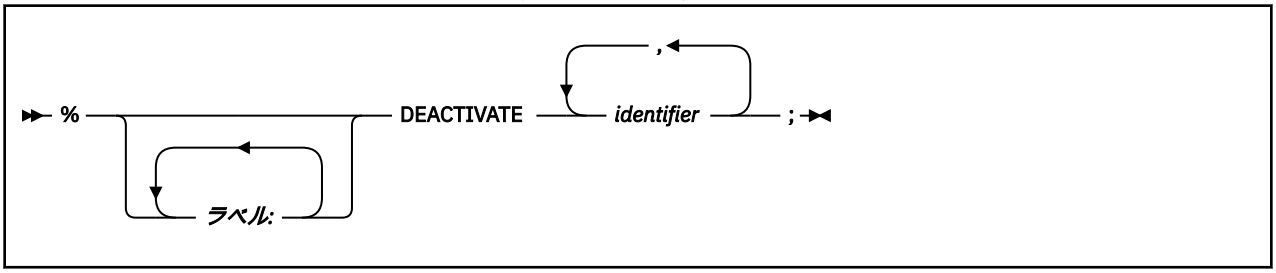



複合割り当てと多重割り当ては許可されていません。

割り当てのターゲットは、配列にはできませんが、配列要素にすることはできます。

## %DEACTIVATE ステートメント

%DEACTIVATE ステートメントは、ID を非活動化します。




**省略形**: %DEACT

**識別子**
プリプロセッサー変数名、プリプロセッサー・プロシージャー名、 またはプリプロセッサー組み込み関数名を指定します。

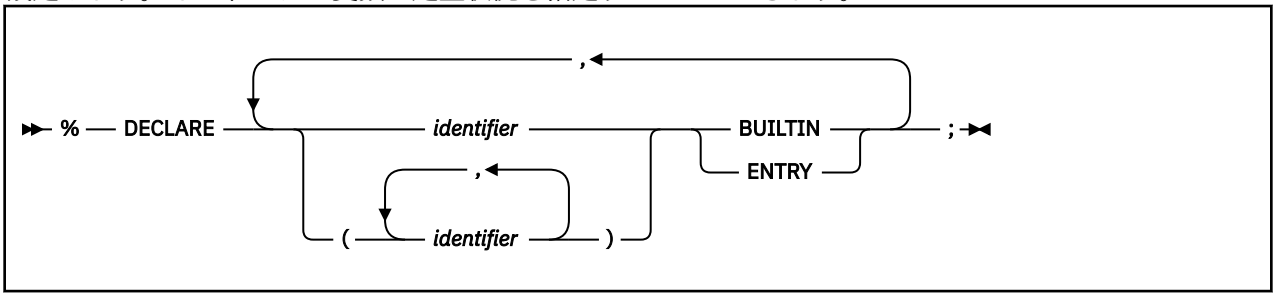
ID を非活動化すると、その ID は置換適格性を失いますが、 その値は失われません。 したがって、その ID を再活動化したとき、置き換え値を割り当てる必要は ありません。

ID を非活動化しても、後続のプリプロセッサー・ステートメント内で その ID に別の値を割り当てることができます。

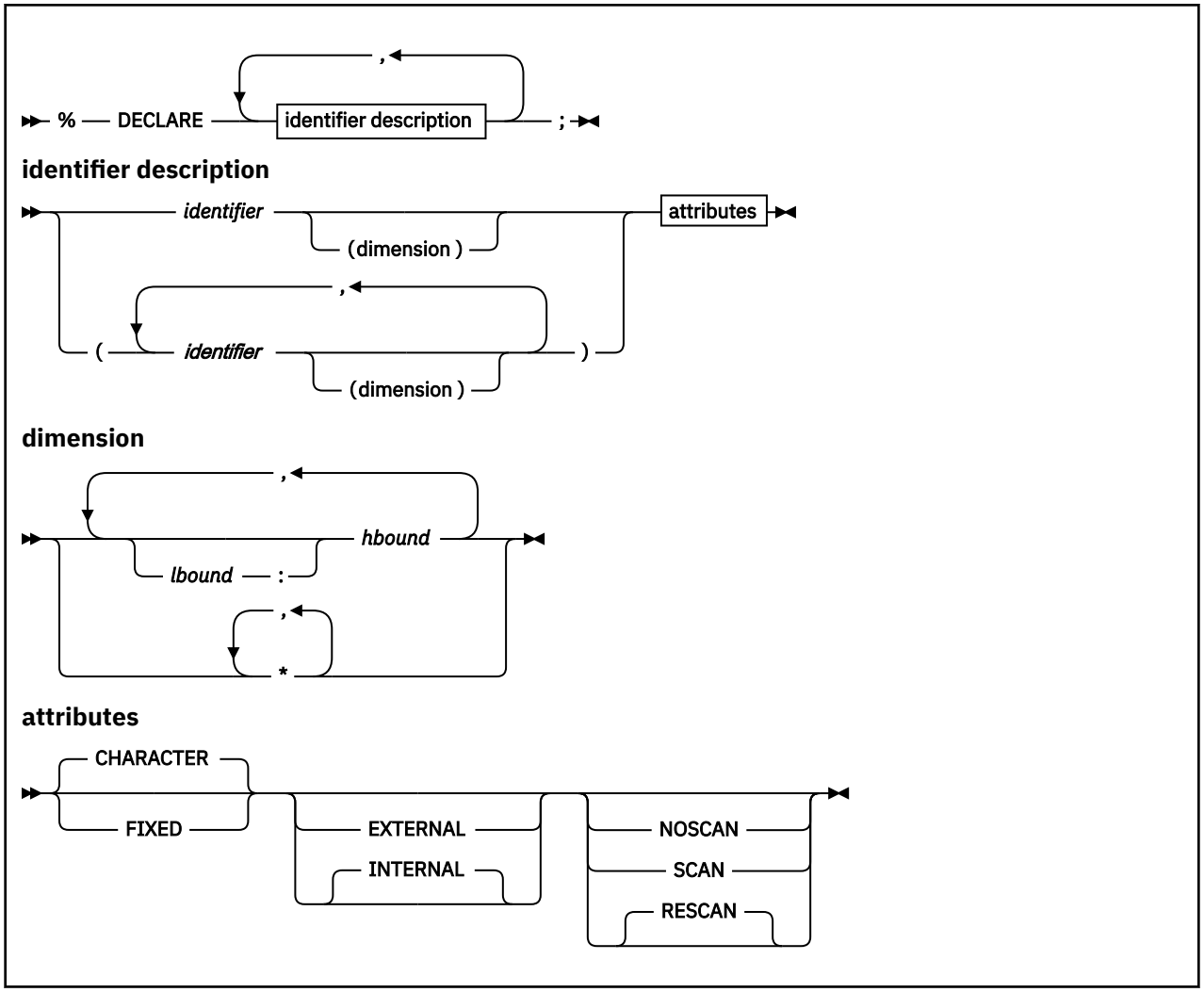
非活動化された ID の非活動化は無効です。

## %DECLARE ステートメント

%DECLARE ステートメントは、ID を マクロ変数、マクロ・プロシージャー、または組み込み関数として設定します。 また、マクロ変数の走査状況を指定することができます。




または




**省略形**: %DECLARE の場合は %DCL、CHARACTER の場合は CHAR、INTERNAL の場合は INT、EXTERNAL の場合は EXT です。

**identifier description**
マクロ機能 ID の名前と属性を指定します。

**BUILTIN**
この ID が同じ名前を持つプリプロセッサー組み込み関数であることを 指定します。

**CHARACTER**
この ID が、最大の長さを持たない可変長文字ストリングを 表していることを指定します。

**ENTRY**
この ID がプリプロセッサー・プロシージャーであることを指定します。

宣言により、入り口名が活動化されます。

プリプロセッサー・プロシージャーの入り口名を %PROCEDURE ステートメント のラベルとして書けば、その入り口名を明示的に宣言することができます。 ただし、このように明示宣言しただけでは、そのプリプロセッサー・ プロシージャー名は活動化されません。

**FIXED**
ID が整数を表すことを指定します。

(デフォルトの) FIXED(DECIMAL) オプションを指定した場合、 ID には属性 DECIMAL(5,0) も与えられます。

FIXED(BINARY) オプションを指定した場合、 ID には属性 BINARY(31,0) も与えられます。

**RESCAN**
この ID がアクティブであり、必要な回数だけ出力内で置換されることを指定します。

**SCAN**
この ID がアクティブであり、1 回だけ出力内で置換されることを指定します。

**NOSCAN**
この ID が非アクティブであり、出力内で置換されないことを指定します。

**ディメンション**
配列変数の次元指定です。 15 を超える次元は指定できません。

**注 :** 配列を (複数のプロシージャー間で共用できるように) プリプロセッサー・プロシージャーの外部で宣言することはできますが、プロシージャーの外部から参照してはなりません (1 つの配列照会組み込み関数への最初の引数として参照する場合を除く)。

***lbound***
この次元にとって必要な下限。 デフォルトは 1 です。

***hbound***
この次元にとって必要な上限。

配列境界は、整数定数、またはより一般的な式によって指定されます。 一般式は他のプリプロセッサー変数の値に依存していてはいけませんが、他の変数の境界など、他の属性に依存する場合があります。 例えば、次の宣言は示された順序で有効です。

```
     %dcl a(2) fixed;
     %dcl b(2+hbound(a)) fixed;
```

また、sysparm が整数定数の場合は、次の宣言も有効です。

```
%dcl c( sysparm() ) fixed;
```

**INTERNAL**
この属性は、プロシージャーの内部でだけ有効です。 プロシージャーの外部で指定すると、診断メッセージが出され、変数には EXTERNAL 属性が与えられます。

プロシージャーの外部で宣言された変数はすべて EXTERNAL であり、 プロシージャーの内部で宣言された変数はすべて INTERNAL です。

**EXTERNAL**
この属性は、プロシージャーの外部でだけ有効です。 プロシージャーの内部で指定すると、診断メッセージが出され、変数には INTERNAL 属性が与えられます。

## %DO ステートメント

%DO ステートメントは、%END ステートメントと対になっており、プリプロセッサー DO グループを区切る働きをします。また DO グループの反復実行を指定することもできます。

%DO ステートメントの構文については、207 ページの『DO ステートメント』を参照してください。

**注 :** 以下の例外を除き、DO ステートメントのすべての形式がサポートされています。

- UPTHRU と DOWNTHRU は受け入れられません。
- タイプ 3 DO ステートメントの specification は、複数回指定できません。

しかし、%DO ステートメントは、DO ステートメントでサポートされない追加フォーマット、%DO SKIP; ステートメントもサポートします。 このステートメントは、一致する %END ステートメントにより、すべてのコードを無視するようにします (したがって、コメントを含むコードをコメント化する方法として便利です)。

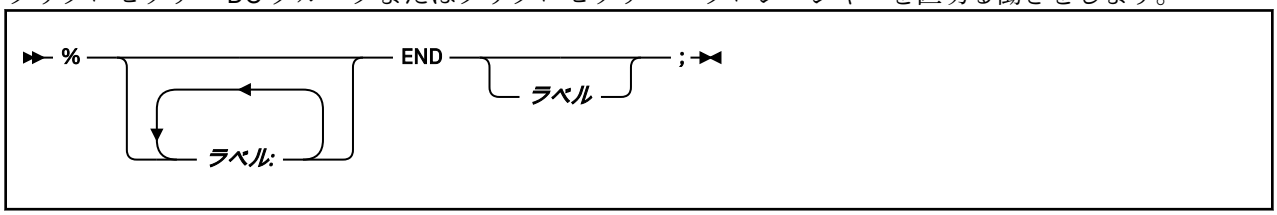
プリプロセッサー DO グループはネストさせることができます。

タイプ 3 のプリプロセッサー DO グループの中に制御権を移動することはできません。 ただし、その DO グループ内から呼び出されたプリプロセッサー・ プロシージャーから戻ることはできます。

プリプロセッサー・ステートメント、入力テキスト、およびリスト制御ステートメントを、 プリプロセッサーの DO グループ内に書くことができます。 プリプロセッサー・ステートメントは実行され、入力テキストは走査されて、 可能であれば置き換えられます。

## %END ステートメント

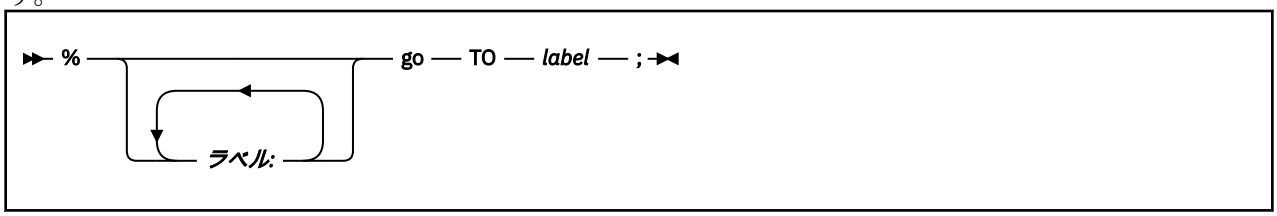
%END ステートメントは、%DO、%SELECT、または %PROCEDURE ステートメントと一緒に 使用され、プリプロセッサー DO グループまたはプリプロセッサー・ プロシージャーを区切る働きをします。




END の後ろのラベル (label) は、%PROCEDURE、%DO、または %SELECT ステートメントの ラベルでなければなりません。 複数の閉止も可能です。

## %GO TO ステートメント

%GO TO ステートメントを使用すると、プリプロセッサーは指定された ラベルの場所から走査を続行します。




**省略形**: %GOTO

GO TO のあとに続くラベルは、走査の移動先を指定します。 ここに指定するラベルは、プリプロセッサー・ステートメントのラベルで なければなりませんが、プリプロセッサー・プロシージャーのラベルで あってはなりません。

プリプロセッサー・プロシージャー内のプリプロセッサー GO TO ステートメントは、そのプロシージャーの外へ制御権を移動することはできません。 つまり、GO TO の後ろに指定するラベルは、そのプロシージャーに含まれている ラベルでなければなりません。

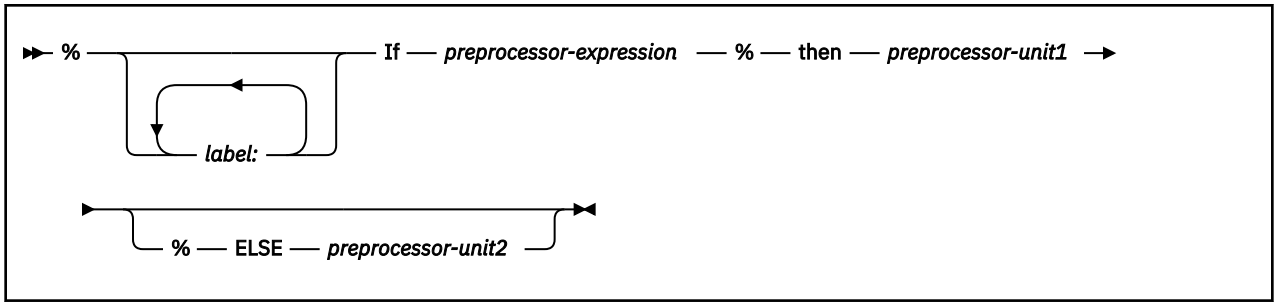
組み込み済みテキスト内の %GO TO ステートメントが制御権を移動することができる のは、同じ組み込みファイルの内部の点に限られています。 %GOTO ステートメントよりも、その %GOTO ステートメントのターゲット・ラベルが 先に来てはなりません。

**関連情報**

605 ページの『%INCLUDE ステートメント』
%INCLUDE が実行されると、そのステートメントで指定された外部テキストが、 プリプロセッサー入力の中に、%INCLUDE ステートメントの代わりとして 組み込まれます。 そのようなテキストがいったん組み込まれると、それは組み込み済み テキストと呼ばれます。そのテキストは、プリプロセッサー・ステートメント、 リスト制御ステートメント、および PL/I 原始ステートメントで 構成することができます。

## %IF ステートメント

%IF ステートメントは、プリプロセッサー式のビット値に従って、 走査の流れを制御します。




**preprocessor-expression**
この式が計算されて、ビット・ストリングに変換されます (変換 できないときはエラーになります)。 %IF ステートメント内の *preprocessor-expression* は短絡化されません。

**preprocessor-unit**
単一のプリプロセッサー・ステートメント (%DECLARE、%PROCEDURE、%END、%DO 以外のもの)、プリプロセッサー DO グループ、またはプリプロセッサー SELECT グループ。 それ以外の場合、記述は、221 ページの『IF ステートメント』で指定されている記述と同じです。

ストリング内のいずれかのビットの値が '1'B であれば、unit1 が実行され、 unit2 (ある場合) は無視されます。すべてのビットが '0'B であれば、 unit1 が無視され、unit2 (ある場合) が実行されます。

unit1 または unit2 のどちらかで %GO TO ステートメントまたは プリプロセッサー RETURN ステートメントを指定したことによって、 走査が別の場所から再開する場合を除けば、%IF ステートメントのすぐ後ろから走査が再開します。

%IF ステートメントは、221 ページの『IF ステートメント』の項で説明したのと同じ方法で ネストさせることができます。

## %INCLUDE ステートメント

%INCLUDE が実行されると、そのステートメントで指定された外部テキストが、 プリプロセッサー入力の中に、%INCLUDE ステートメントの代わりとして 組み込まれます。 そのようなテキストがいったん組み込まれると、それは組み込み済み テキストと呼ばれます。そのテキストは、プリプロセッサー・ステートメント、 リスト制御ステートメント、および PL/I 原始ステートメントで 構成することができます。

%INCLUDE ステートメントの構文については、223 ページの『%INCLUDE ディレクティブ』を参照してください。

*data set* と *member name* のペアはいずれも、ソース・プログラムに組み込まれる外部テキストを識別します。

走査は、包含されるテキストの最初の文字から続行されます。 包含されるテキストは、プリプロセッサー入力と同じ方法で走査されます。 ですから、包含されるテキストを、形成するプリプロセッサー出力の基にすることができます。

%INCLUDE ステートメントはネストすることができます。 つまり、組み込まれたテキストの中に、%INCLUDE ステートメントがあってもかまいません。

組み込み済みテキスト内の %GO TO ステートメントが制御権を移動することができる のは、同じ組み込みファイルの内部の点に限られています。 %GOTO ステートメントよりも、その %GOTO ステートメントのターゲット・ラベルが 先に来てはなりません。

組み込み済みテキスト内のプリプロセッサー・ステートメント、DO グループ、 SELECT グループ、およびプロシージャーは完結していなければなりません。 例えば、%IF ステートメントの半分が組み込みテキスト内にあり、残りの 半分がプリプロセッサー入力の別の部分にあるということは許されません。

プリプロセッサー入力と組み込み済みテキスト内に、%INCLUDE 以外には プリプロセッサー・ステートメントが含まれていない場合は、プリプロセッサーの実行を省略することができます。 (その場合、INCLUDE コンパイル時オプションを使用する必要があります。)

例えば、PAYRL がデータ・セット SYSLIB のメンバーであり、以下のテキスト (構造体宣言) を含むものとします。

```
DECLARE 1 PAYROLL,
          2 NAME,
            3 LAST   CHARACTER (30) VARYING,
            3 FIRST  CHARACTER (15) VARYING,
            3 MIDDLE CHARACTER (3) VARYING,
          2 CURR,
            3 (REGLAR, OVERTIME) FIXED DECIMAL (8,2),
          2 YTD LIKE CURR;
```

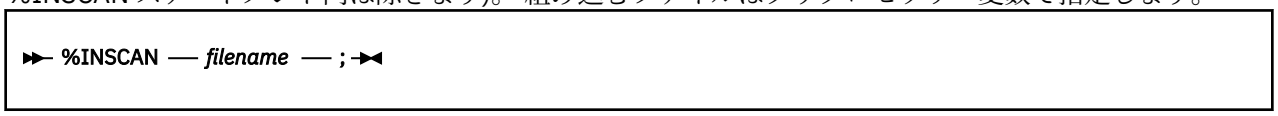
以下のプリプロセッサー・ステートメントにより、2 つの構造体宣言がプリプロセッサー出力テキスト内に生成されます。 それら 2 つの構造体宣言は名前だけが異なっており、それぞれ CUM_PAY と PAYROLL という名前です。

```
%DECLARE PAYROLL CHARACTER;
%PAYROLL='CUM_PAY';
%INCLUDE PAYRL;
%DEACTIVATE PAYROLL;
%INCLUDE PAYRL;
```

最初の %INCLUDE ステートメントが実行されると、PAYRL 内のテキストが プリプロセッサー入力の中に組み込まれます。 この組み込み済みテキスト内の ID PAYROLL がプリプロセッサー走査で検出される と、ID PAYROLL はアクティブのプリプロセッサー変数 PAYROLL の現行値 (つまり CUM_PAY) で 置き換えられます。 それ以上の包含されたテキストを走査しても、追加の置換は行われません。 次に、%DEACTIVATE ステートメントがプリプロセッサー走査で検出され、 プリプロセッサー変数 PAYROLL が非活動化されます。 2 番目の %INCLUDE ステートメントが実行されると、PAYRL 内のストリングが もう一度プリプロセッサー入力の中に組み込まれます。 しかし、今回は、この組み込み済みテキストが走査されても置き換えはまったく 行われません。

# %INSCAN ステートメント

%INCLUDE ステートメントと同様に、%INSCAN ステートメントもファイルを組み込みます (ただし、%INSCAN ステートメント内は除きます)。 組み込むファイルはプリプロセッサー変数で指定します。




**ファイル名**
組み込むファイルの名前を指定するプリプロセッサー式。

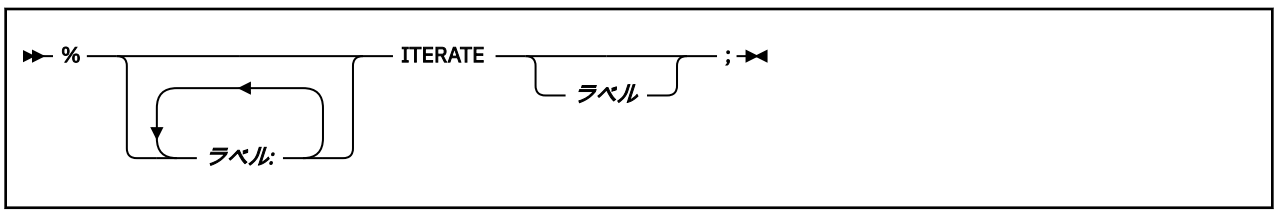
**例**

```
%dcl inname char;
%inname = 'oldform';
%inscan inname; /* includes the file "oldform" */
```

# %ITERATE ステートメント

%ITERATE ステートメントは、このステートメントを含む 反復 DO グループを区切る %END ステートメントに制御を移します。 現在行われている反復は完了され、次の反復が (必要であれば) 開始されます。

ITERATE ステートメントは、反復 DO グループが 非反復 DO グループを含んでいる場合には、 その非反復 DO グループ内で指定することができます。

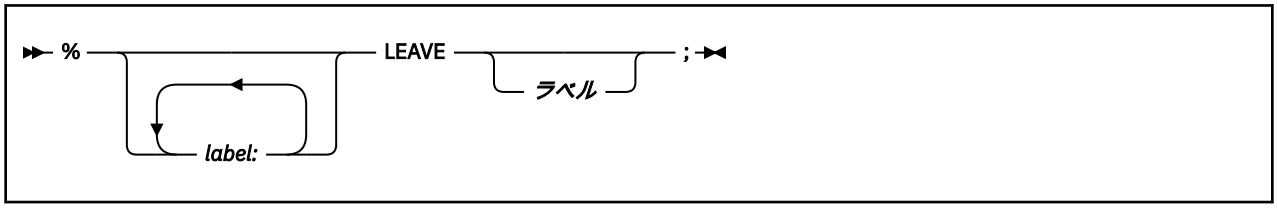


**label-constant**
このステートメントを含んでいる DO グループのラベルでなければなりません。 *label-constant* を省略すると、ITERATE ステートメントを含む最新の反復 DO グループの END ステートメントに制御が移ります。

## %LEAVE ステートメント

単純な DO グループに含まれているか、 または単純な DO グループを指定してある場合に、 %LEAVE ステートメントは、そのグループを終了します。 反復 DO グループに含まれているか、 またはそのようなグループを指定してある場合には、 %LEAVE ステートメントは、現在の繰り返しも含め、 そのグループのすべての繰り返しを終了します。

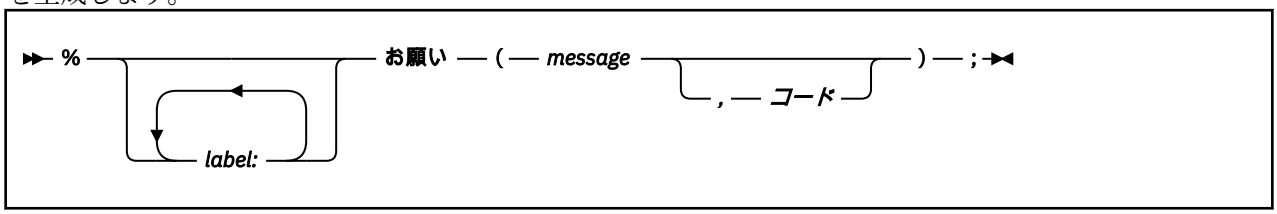
制御の流れは、DO グループが END ステートメントに達して終了した場合に通常移動するはずの位置に移動します。




**label-constant**
このステートメントを含んでいる DO グループのラベルでなければなりません。 ここで指定するラベルを持つ DO グループが終了します。 *label-constant* が省略されると、LEAVE ステートメントを含むグループが終了する DO グループです。

## %NOTE ステートメント

%NOTE ステートメントは、指定されたテキストと重大度コードを示す プリプロセッサー診断メッセージを生成します。




**message**
診断メッセージを必要とする値を持つ文字式。

**コード**
メッセージの重大度を表す値が入っている固定式。重大度は次のとおりです。

| コード | 重大度 |
|---|---|
| 0 | I |
| 4 | W |
| 8 | E |
| 12 | S |
| 16 | U |

*code* を省略すると、デフォルトとして 0 が設定されます。

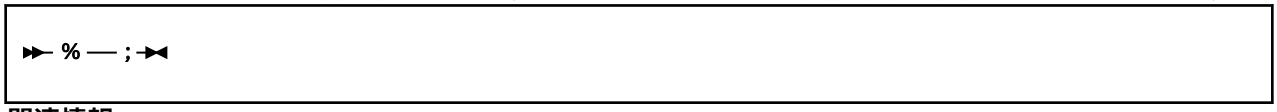
*code* の値が上記以外の値のときは、診断メッセージが生成され、 デフォルトの値が使われます。 値が 0 より小さいか 16 より大きい場合は、重大度 U がデフォルトです。 それ以外の場合は、次に低い重大度がデフォルトです。

生成されたメッセージは、他のプリプロセッサー・メッセージと一緒に ファイルされます。 特定のメッセージを続いて印刷するかどうかは、その重大度レベルと コンパイラー FLAG オプションの設定に依存します。

重大度 U のメッセージが生成されると、プリプロセスとコンパイル処理が直ちに終了します。 重大度 S、E、または W のメッセージが生成されたときにコンパイル処理が 終了するかどうかは、NOSYNTAX および NOCOMPILE コンパイル時オプションが どのようにセットされているかによって異なります。

## %null ステートメント

%null ステートメントは、何も行いません。またステートメントの実行順序を 変えることもしません。



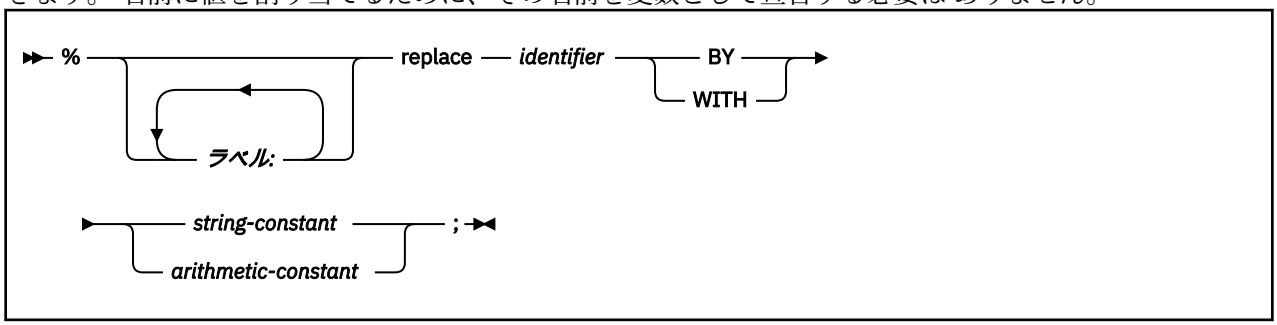

**関連情報**

586 ページの『%PROCEDURE ステートメント』
%PROCEDURE ステートメントは、%END ステートメントと対になっており、 プリプロセッサー・プロシージャーを区切る働きをします。

587 ページの『プリプロセッサー RETURN ステートメント』
プリプロセッサー RETURN ステートメントは、プロシージャーに RETURNS 属性がある場合に限り、プリプロセッサー・プロシージャー内でのみ使用できます。 このステートメントは、プリプロセッサー・プロシージャーの呼び出し点に値と制御を返します。

## %REPLACE ステートメント

%REPLACE ステートメントを使用すると、名前をストリング定数か 数値定数に直ちに置き換えることができます。 名前に値を割り当てるために、その名前を変数として宣言する必要は ありません。




**識別子**
置き換えられる名前。

**string-constant**
名前が宣言されていない場合、CHARACTER 属性が与えられます。

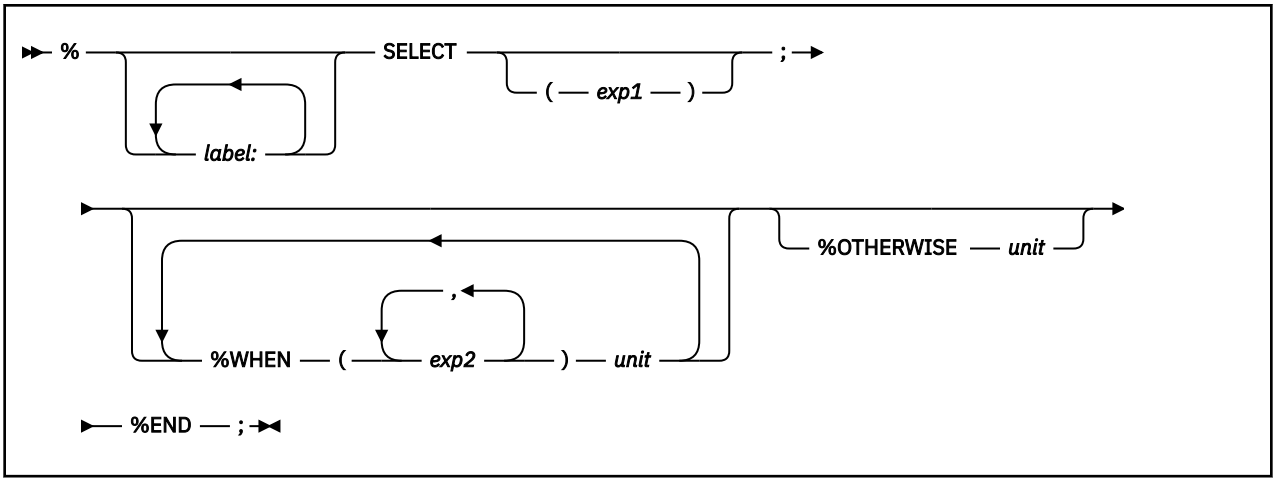
**arithmetic-constant**
名前が宣言されていない場合、FIXED 属性が与えられます。

FIXED(DEC) オプションが指定されている場合、値は FIXED DEC(5,0) に 変換されます。

FIXED(BIN) オプションが指定されている場合、値は FIXED BIN(31,0) に 変換されます。

## %SELECT ステートメント

%SELECT ステートメントは、%END ステートメントと対になっており、プリプロセッサー SELECT グループを区切る働きをします。




## %XINCLUDE ステートメント

%XINCLUDE ステートメントは %INCLUDE ステートメントと同じですが、ファイルがすでに存在している場合はファイルを組み込みません。

%XINCLUDE ステートメントの構文については、233 ページの『%XINCLUDE ステートメント』を参照してください。

## %XINSCAN ステートメント

%XINSCAN ステートメントは %INSCAN ステートメントと同じですが、ファイルがすでに存在している場合はファイルを組み込みません。

# プリプロセッサーの例

このトピックでは、プリプロセッサーの例をいくつか示します。

### 例 1

プリプロセッサー入力に以下のステートメントが含まれているとします。

```
%DECLARE A CHARACTER, B FIXED;
%A = 'B+C';
%B = 2;
X = A;
```

次のストリングがプリプロセッサー出力に挿入されます。

```
X = 2+C;
```

これらのプリプロセッサー・ステートメントは、デフォルトの値である RESCAN を指定して A と B を 活動化し、文字ストリング 'B+C' を A に割り当て、さらに定数 2 を B に割り当てます。

4 行目は入力テキストです。 A の現行値、つまり 'B+C' が A の代わりとしてプリプロセッサー出力の 中に挿入されます。 しかし、このストリングにはプリプロセッサー変数 B が含まれています。 B を再スキャンすると、プリプロセッサーはそれが活動化されたことを検出します。 ですから、プリプロセッサー出力内の B は値 2 に置き換わります。 プリプロセッサー変数 B の精度は、デフォルトの精度 (5,0) なので、 実際には 2 の前に 0 が 4 つ付いています。 ストリング 'B+C' 内の B がこの値で置き換えられるときに、この値は文字ストリングに変換されるので、2 の前にブランクが 7 つ付きます。

さらに再走査が行われ、2 をこれ以上置き換えられないことがわかると、 +C から走査が再開されます。これも置き換えることはできません。

上記の例で、プリプロセッサー変数 A がステートメント %ACTIVATE A NORESCAN; によって活動化された場合、プリプロセッサー出力は次のようになります。

```
X = B+C;
```

### 例 2

プリプロセッサー入力に以下のステートメントが含まれているとします。

```
%DECLARE I FIXED, T CHARACTER;
%DEACTIVATE I;
%I = 15;
%T = 'A(I)';
S  = I*T*3;
%I = I+5;
%ACTIVATE I;
%DEACTIVATE T;
R  = I*T*2
```

プリプロセッサー出力は次のようになります。 置換ブランクは表示されません。

```
S = I*A(I)*3;
R = 20*T*2;
```

### 例 3

この例は、プリプロセッサーの諸機能を使用して、DO グループの実行速度を上げる方法を示したものです。

DO グループの例を以下に示します。

```
DO I=1 TO 10;
Z(I)=X(I)+Y(I);
END;
```

以下のステートメントは同じことを行いますが、コンパイル済みプログラムの実行中に増分およびテストを行う必要はなくなります。

```
%DECLARE I FIXED;
%DO I = 1 TO 10;
Z(I)=X(I)+Y(I);
%END;
%DEACTIVATE I;
```

3 行目は入力テキストなので、置き換えをする必要があるかどうかを 調べるために走査されます。 この行が最初に走査されたときは、I の値は 1 であり、しかも I は 活動化されています。 その結果、次のストリングがプリプロセッサー出力に挿入されます。

```
Z(       1)=X(       1)+Y(       1);
```

1 の前には、それぞれブランクが 7 入ります。

I の値が増えて 10 になるまで、そのつど、入力テキストが走査され、 I があれば I の現行値で置き換えられます。 結果として、以下のストリングがプリプロセッサー出力に挿入されます。

```
Z(       1)=X(       1)+Y(       1);
Z(       2)=X(       2)+Y(       2);
          .
          .
          .
Z(      10)=X(      10)+Y(      10);
```

I の値が 11 に達すると、制御が %DEACTIVATE ステートメントに渡ります。

### 例 4

下記のプリプロセッサー入力内の VALUE は、'arg1(arg2)' の形をした文字ストリングを返すプリプロセッサー関数プロシージャーです。ただし、 *arg1* と *arg2* は、この関数に渡される引数を表しています。

```
DECLARE (Z(10), Q) FIXED;
%A='Z';
%ACTIVATE A, VALUE;
Q = 6 + VALUE(A,3);
%DECLARE A CHARACTER;
%VALUE: PROC(ARG1,ARG2) RETURNS(CHAR);
        DCL ARG1 CHAR, ARG2 FIXED;
        RETURN(ARG1‖'('‖ARG2‖')');
        %END VALUE;
```

4 行目が走査されるとき、A はアクティブになっているので置き換えに対して適格です。 VALUE もアクティブなので、4 行目で VALUE が参照されると、その名前のプリプロセッサー関数プロシージャーが起動します。

ただし、引数 A と 3 が VALUE に渡される前に、A は値 Z (前の 代入ステートメントで A に割り当てられた値) で置き換えられ、 3 は対応するパラメーターの属性に合わせて固定小数点フォーマットに変換されます。 引数が渡されると、VALUE はこれらの引数と括弧の連結演算を行い、 結果の値、つまりストリング Z(3) を呼び出し点に返します。 関数参照が戻り値で置き換えられ、その結果がプリプロセッサー出力に 挿入されます。 したがって、生成されるプリプロセッサー出力は以下のようになります。

```
DECLARE (Z(10),Q) FIXED;
Q = 6+Z(       3);
```

### 例 5

以下のように定義されたプリプロセッサー関数プロシージャー GEN は、パラメーター記述子リストに最大 99 のパラメーター記述子を持つ、最大 99 の入り口名に対して GENERIC 宣言を生成できます。 この例では、4 個だけが生成されます。

```
 %DCL GEN ENTRY;
 DCL A GEN (A,2,5,FIXED);
   %GEN: PROC(NAME,LOW,HIGH,ATTR) RETURNS (CHAR);
 DCL (NAME, SUFFIX, ATTR, STRING) CHAR, (LOW, HIGH, I, J) FIXED;
 STRING='GENERIC(';
 DO I=LOW TO HIGH;                      /* ENTRY NAME LOOP*/
    IF I>9 THEN
       SUFFIX=SUBSTR(I, 7, 2);
                                        /* 2 DIGIT SUFFIX*/
    ELSE SUFFIX=SUBSTR(I, 8, 1);
                                        /* 1 DIGIT SUFFIX*/
    STRING=STRING‖NAME‖SUFFIX‖' WHEN (';
    DO J=1 TO I;                        /* DESCRIPTOR LIST*/
       STRING=STRING‖ATTR;
       IF J<I                           /* ATTRIBUTE SEPARATOR*/
          THEN STRING=STRING‖',';
          ELSE STRING=STRING‖')';
                                        /* LIST SEPARATOR */
    END;
    IF I<HIGH THEN                      /* ENTRY NAME SEPARATOR*/
       STRING=STRING‖',';
    ELSE STRING=STRING‖')';
                                       /* END OF LIST /*
 END;
 RETURN (STRING)
 % END;
```

生成されるプリプロセッサー出力は以下のようになります。

```
 DCL A GENERIC(A2 WHEN (FIXED,FIXED),
               A3 WHEN (FIXED, FIXED, FIXED),
               A4 WHEN (FIXED, FIXED, FIXED, FIXED),
               A5 WHEN (FIXED, FIXED, FIXED, FIXED, FIXED));
```

### 例 6

この例では、下記のフォーマットのステートメントと同じ働きをする プリプロセッサー・プロシージャーを示します。

```
SEARCH TABLE(array) FOR(value)
USING(variable) AND(variable);
```

このステートメントは、配列添え字として指定した (またはデフォルトの) 変数 を使用して、指定した値を求めて 2 次元配列を探索するための ステートメントです。 このステートメントが実行されると、指定した値がどのエレメントに入っているかが 配列添え字変数によって示されます。 指定した値がどのエレメントにも入っていない場合は、両方の添え字変数が -22222 にセットされます。

このステートメントを実行するプリプロセッサー・プロシージャーは以下のようになります。

```
%SEARCH:
PROC(TABLE,FOR,USING,AND) STATEMENT RETURNS(CHARACTER);

  DECLARE(TABLE,FOR,USING,AND,LABL, DO1,DO2) CHARACTER,
      (PARMSET,COUNTER) BUILTIN;

  IF PARMSET(TABLE) & PARMSET(FOR) THEN;
  ELSE SERR:DO;
  NOTE ('MISSING OR INVALID ARGUMENT(S)'||'FOR ''SEARCH''',4);
  RETURN ('/*INVALID SEARCH STATEMENT*/');
  END;

  IF ¬PARMSET(USING) THEN
    USING='I';
  IF ¬PARMSET(AND)   THEN
    AND='J';
  IF USING = AND  THEN
    GO TO SERR;

  LABL='SL'||COUNTER;
  DO1=LABL||': DO '||USING||'=LBOUND('||TABLE||',1)
    TO HBOUND('||TABLE||',1);';
  DO2='DO '||AND||'=LBOUND('||TABLE||',2)
    TO HBOUND ('||TABLE||',2);';

  RETURN(DO1||DO2||'SELECT('||TABLE
    ||'('||USING||','||AND||'));
  WHEN('||FOR||') LEAVE '||LABL||';
  OTHER;
  END '||LABL||';
  IF '||AND||' > H  BOUND('||TABLE||',2) THEN
    '||USING||','||AND||.' = -22222;');
%END SEARCH;
```

この例では、PARMSET 組み込み関数を使用して、このプロシージャーが 呼び出されたときにどのパラメーターがセットされるかを判別しています。 USING がセットされていない場合、配列添え字付き変数 I のデフォルトが使用されます。 AND がセットされていないと、J が使用されます。 TABLE または FOR がセットされていないか、または呼び出しの結果として 両方の添え字が同じ変数になった場合は、プリプロセッサー診断メッセージが 出され、コメントがプリプロセッサー出力に返されます。

COUNTER 組み込み関数は、このプロシージャーから返されるプリプロセッサー 出力に固有のラベルを付けるために使われています。

プロシージャーは、キーワード引数、定位置引数、またはその両方の組み合わせを使用して呼び出すことができます。 次のようにプロシージャーを呼び出すと、同じ結果が生成されます。

```
SEARCH TABLE(LIST.NAME)
FOR('J.DOE') USING(I) AND(J);

SEARCH TABLE(LIST.NAME) FOR('J.DOE');

SEARCH(LIST.NAME) FOR('J.DOE');

SEARCH(LIST.NAME,'J.DOE');

SEARCH(,'J.DOE') TABLE(LIST.NAME);
```

これらの呼び出しによって返されるプリプロセッサー出力は以下のようになります。

```
SL00001:
DO I=LBOUND(LIST.NAME,1) TO HBOUND(LIST.NAME,1);
   DO J=LBOUND(LIST.NAME,2) TO HBOUND(LIST.NAME,2);
      SELECT(LIST.NAME(I,J));
   WHEN('J.DOE') LEAVE SL00001;
   OTHER;
END SL00001;
IF J > HBOUND(LIST.NAME,2) THEN
   I,J = -22222;
```

ラベル SL00001 は、最初の呼び出しの場合にだけ返されます。 それ以後、このプロシージャーを呼び出すたびに、新しい固有のラベルが 返されます。

# 付録 A 制限値

このトピックでは、PL/I 言語要素やマクロ機能言語要素に対する実装制限について、および UPPERCASE 組み込み関数と LOWERCASE 組み込み関数のサポート対象コード・ページ値に対する実装制限について簡単に説明します。

615 ページの表 85 に、PL/I 言語エレメントの設定制限値を要約しています。

表 *85.* 言語エレメントの制限

| 言語エレメント | 説明 | 制限 |
|---|---|---|
| 配列 | 配列の次元の最大数 | 15 |
| | 最小下限[注 1] | CMPAT(V3) の場合は $-2^{63}$、その他の場合は -2147483648 |
| | 最大上限[注 1] | CMPAT(V3) の場合は $+2^{63} - 1$、その他の場合は +2147483647 |
| **注 1 :** これらの限度の使用には注意が必要です。 例えば、A に最大上限が指定されていて、JX に属性 SIGNED FIXED BIN(31) が指定されている場合、ループ `DO JX = LBOUND(A) TO HBOUND(A)` は配列の最後のエレメントにヒットした後「ラップ」します。 TO の代わりに UPTHRU を使用した場合、ループはラップしません。 | | |
| 構造体 | 構造体内のレベルの最大数 | 15 |
| | 構造体内の最大レベル番号 | 255 |
| 演算精度 | FIXED DECIMAL の最大精度 | 31[注 2] |
| | FIXED BINARY の最大精度 | 63[注 3] |
| | FLOAT DECIMAL の最大精度 | 33[注 4] |
| | FLOAT BINARY の最大精度 | 109[注 5] |
| | FIXED データの最大位取り係数 | 127 |
| | FIXED データの最小位取り係数 | -128 |
| **注 2 :** これはコンパイル時オプション LIMITS(FIXEDDEC(31)) を指定した 場合に限り当てはまります。 デフォルトは 15 です。<br>**注 3 :** これはコンパイル時オプション LIMITS(FIXEDBIN(63)) を指定した 場合に限り当てはまります。 デフォルトは 31 です。<br>**注 4 :** Intel の FLOAT DECIMAL 最大精度は 18 です。 FLOAT(DFP) を指定すると、FLOAT DECIMAL 最大精度は 34 になります。<br>**注 5 :** Intel の FLOAT BINARY 最大精度は 64 です。 | | |
| ストリング変数/定数と AREA 変数/定数 | CHARACTER の最大の長さ | 32767 |
| | BIT の最大の長さ | 32767 |
| | GRAPHIC の最大の長さ | 16383 |
| | WIDECHAR の最大の長さ | 32767 |
| | AREA の最大サイズ | 2147483647 |

© Copyright IBM Corp. 1999, 2022

表 *85.* 言語エレメントの制限 *(*続き*)*

| 言語エレメント | 説明 | 制限 |
|---|---|---|
| **注 6 :** これらはデフォルトの制限値です。 LIMITS コンパイラー・オプションの STRING サブオプションの下では、ストリングの最大長は 128M になります。 | | |
| 組み込み関数 | IAND、IOR、MAX、および MIN 関数への引数の最大個数 | 64 |
| | ADD、BINARY、DECIMAL、DIVIDE、 FIXED、FLOAT、MULTIPLY、PRECISION、および SUBTRACT 関数内の精度 (p) の最大値 | 対応する算術精度の制限と同じ |
| | ADD、BINARY、DECIMAL、DIVIDE、 FIXED、MULTIPLY、PRECISION、および SUBTRACT 関数内のスケール (q) の最大値 | 対応する算術精度の制限と同じ |
| | CEIL、FLOOR、MAX、MIN、MOD、 ROUND、および TRUNC 関数内の桁 (N) の最大数 | 対応する算術精度の制限と同じ |
| プログラム・サイズ | ID の最大の長さ | 100 |
| | ステートメント・タイプが変更される前の字句単位 (キーワード、ID、区切り文字など) の最大数 | 511 |
| | ブロック内の DEFAULT ステートメントの最大数 | 31 |
| | %PUSH ステートメントの最大数 | 63 |
| | %INCLUDE ステートメントの最大数 | 4095 |
| | %INCLUDE ステートメントの最大ネスト | 2046 |
| | ソース・ファイル内の最大行数 | 1048575 |
| | ステートメントの最大数 | 16777215 |
| | ブロック内の LIKE 属性の最大数 | 63 |
| | データ・リスト内の出力式の最大数 | 60 |
| | データ・リスト内の反復 DO 指定の最大数 | 50 |
| | 位置合わせされていないビットを含まないデータ集合の最大サイズ | 2147483647 |
| | 位置合わせされていないビットを含むデータ集合の最大サイズ | 268435455 |
| | CALL または関数参照内の引数の最大数 | 255 |
| | プロシージャーの最大パラメーター数 | 4095 |
| | 分配された属性の最大ネスト | 15 |
| | BEGIN および PROCEDURE ステートメントの最大ネスト | 30 |
| | DO グループの最大ネスト | 49 |
| | IF ステートメントの最大ネスト | 49 |
| | SELECT ステートメントの最大ネスト | 49 |
| | 式の最大ネスト | 383 |
| | %NOTE メッセージの最大の長さ | 32767 |

表 *85.* 言語エレメントの制限 *(*続き*)*

| 言語エレメント | 説明 | 制限 |
|---|---|---|
| 各種 | 文字ピクチャー内のピクチャー文字の最大数 | 511 |
| | 数値ピクチャー内の最大バイト数 | 253 |
| | 数値ピクチャー内の数値ピクチャー文字の最大数 | 31 |
| | CHARACTER、X、BIT、BX、GRAPHIC、 GX、WX および M ストリング定数の外部表示の最大バイト数<br>外部表示には、すべての引用符とストリング接尾部が含まれます。 例えば、ストリング '01010110'B にはその外部指定内に 11 バイトありますが、内部指定には 1 バイトしかありません。 同様に、ストリング 'Ain''t Misbehavin''' にはその外部指定内に 21 バイトありますが、 内部表示には 17 バイトしかありません。 | 3072 |
| | KEYTO 文字ストリングの最大の長さ | 120 |
| | KEYTO グラフィックまたはワイド文字ストリングの最大長 | 60 |
| | KEY の最大の長さ | 32763 |
| | LINESIZE の最大行サイズ | F フォーマットあるいは U フォーマットでは 32,759、V フォーマットでは 32,751 |
| | LINESIZE の最小行サイズ | 1 |
| | PAGESIZE の最大ページ・サイズ | 32,767 |
| | PAGESIZE コンパイラー・オプションの最小ページ・サイズ | 1 |
| | DISPLAY 文字ストリングの最大サイズ | 126 |
| | 最大の DISPLAY 応答メッセージ | 72 バイト |
| | IEEE 正規化浮動小数点数値の範囲 | +3.30E-4932 から +1.21E+4932、0、-3.30E-4932 から -1.21E+4932 |
| | 16 進浮動小数点数値の範囲 | +10E-78 から +10E75、0、-10E-78 から +10E+75 |

また、617 ページの表 86 に、マクロ機能の言語エレメントの設定制限値を 要約しています。

表 *86.* マクロ機能の制限

| 言語エレメント | 説明 | 制限 |
|---|---|---|
| 配列 | 最大次元数 | 15 |
| | 下限の最小値 | -32768 |
| | 上限の最大値 | +32767 |

| 表 86. マクロ機能の制限 (続き) | | |
|---|---|---|
| **言語エレメント** | **説明** | **制限** |
| 演算 範囲 | 最大および最小の FIXED(DECIMAL) オプションによる FIXED 変数 | FIXED DECIMAL(5) ID と同じ |
| | 最大および最小の FIXED(BINARY) オプションによる FIXED 変数 | FIXED BINARY(31) ID と同じ |
| マクロ・プロシージャー | 最大ネスティング・レベル | 1 |
| 鍵 | キーワード・パラメーターの最大個数 | 4096 |
| ストリング結果 | 最大長 | 512K |

618 ページの表 87 は、*c* のサポートされる値の *lowerc* および *upperc* の値をリストします。 *c* は、大文字または小文字に変換されるコード・ページを示します。

表 *87. LOWERCASE* 組み込み関数および *UPPERCASE* 組み込み関数に対してサポートされているコード・ページ値

| 下限 | 上限 |
|---|---|
| `dcl lower_00037 char`<br>`  value( (`<br>`  '8182838485868788'x`<br>`  || '8991929394959697'x`<br>`  || '9899A2A3A4A5A6A7'x`<br>`  || 'A8A9424344454647'x`<br>`  || '4849515253545556'x`<br>`  || '5758708C8D8E9CCB'x`<br>`  || 'CCCDCECFDBDCDDDE'x`<br>`  ) );` | `dcl upper_00037 char`<br>`  value( (`<br>`  'C1C2C3C4C5C6C7C8'x`<br>`  || 'C9D1D2D3D4D5D6D7'x`<br>`  || 'D8D9E2E3E4E5E6E7'x`<br>`  || 'E8E9626364656667'x`<br>`  || '6869717273747576'x`<br>`  || '777880ACADAE9EEB'x`<br>`  || 'ECEDEEEFFBFCFDFE'x`<br>`  ) );` |
| `dcl lower_00273 char`<br>`  value( (`<br>`  '8182838485868788'x`<br>`  || '8991929394959697'x`<br>`  || '9899A2A3A4A5A6A7'x`<br>`  || 'A8A9424445464748'x`<br>`  || '4951525354555657'x`<br>`  || '586A708C8D8E9CC0'x`<br>`  || 'CBCDCECFD0DBDDDE'x`<br>`  ) );` | `dcl upper_00273 char`<br>`  value( (`<br>`  'C1C2C3C4C5C6C7C8'x`<br>`  || 'C9D1D2D3D4D5D6D7'x`<br>`  || 'D8D9E2E3E4E5E6E7'x`<br>`  || 'E8E9626465666768'x`<br>`  || '6971727374757677'x`<br>`  || '78E080ACADAE9E4A'x`<br>`  || 'EBEDEEEF5AFBFDFE'x`<br>`  ) );` |
| `dcl lower_00277 char`<br>`  value( (`<br>`  '8182838485868788'x`<br>`  || '8991929394959697'x`<br>`  || '9899A2A3A4A5A6A7'x`<br>`  || 'A8A9424344454648'x`<br>`  || '4951525354555657'x`<br>`  || '586A8C8D8EA1C0CB'x`<br>`  || 'CCCDCECFD0DBDDDE'x`<br>`  ) );` | `dcl upper_00277 char`<br>`  value( (`<br>`  'C1C2C3C4C5C6C7C8'x`<br>`  || 'C9D1D2D3D4D5D6D7'x`<br>`  || 'D8D9E2E3E4E5E6E7'x`<br>`  || 'E8E9626364656668'x`<br>`  || '6971727374757677'x`<br>`  || '787CACADAEFC7BEB'x`<br>`  || 'ECEDEEEF5BFBFDFE'x`<br>`  ) );` |
| `dcl lower_00278 char`<br>`  value( (`<br>`  '8182838485868788'x`<br>`  || '8991929394959697'x`<br>`  || '9899A2A3A4A5A6A7'x`<br>`  || 'A8A9424445464849'x`<br>`  || '525354555657586A'x`<br>`  || '70798C8D8E9CA1C0'x`<br>`  || 'CBCDCECFD0DBDDDE'x`<br>`  ) );` | `dcl upper_00278 char`<br>`  value( (`<br>`  'C1C2C3C4C5C6C7C8'x`<br>`  || 'C9D1D2D3D4D5D6D7'x`<br>`  || 'D8D9E2E3E4E5E6E7'x`<br>`  || 'E8E9626465666869'x`<br>`  || '7273747576777870'x`<br>`  || '80E0ACADAE9EFC7B'x`<br>`  || 'EBEDEEEF5BFBFDFE'x`<br>`  ) );` |

表 *87. LOWERCASE* 組み込み関数および *UPPERCASE* 組み込み関数に対してサポートされているコード・ページ値 *(*続き*)*

| 下限 | 上限 |
| --- | --- |
| dcl lower_00280 char<br>  value( (<br>  '8182838485868788'x<br>  \|\| '8991929394959697'x<br>  \|\| '9899A2A3A4A5A6A7'x<br>  \|\| 'A8A9424345464749'x<br>  \|\| '52535556575A6A70'x<br>  \|\| '798C8D8E9CA1C0CB'x<br>  \|\| 'CCCECFD0DBDCDEE0'x<br>  ) ); | dcl upper_00280 char<br>  value( (<br>  'C1C2C3C4C5C6C7C8'x<br>  \|\| 'C9D1D2D3D4D5D6D7'x<br>  \|\| 'D8D9E2E3E4E5E6E7'x<br>  \|\| 'E8E9626365666769'x<br>  \|\| '727375767771ED80'x<br>  \|\| 'FDACADAE9E7864EB'x<br>  \|\| 'ECEEEF74FBFCFE68'x<br>  ) ); |
| dcl lower_00284 char<br>  value( (<br>  '8182838485868788'x<br>  \|\| '8991929394959697'x<br>  \|\| '9899A2A3A4A5A6A7'x<br>  \|\| 'A8A9424344454647'x<br>  \|\| '4851525354555657'x<br>  \|\| '586A708C8D8E9CCB'x<br>  \|\| 'CCCDCECFDBDCDDDE'x<br>  ) ); | dcl upper_00284 char<br>  value( (<br>  'C1C2C3C4C5C6C7C8'x<br>  \|\| 'C9D1D2D3D4D5D6D7'x<br>  \|\| 'D8D9E2E3E4E5E6E7'x<br>  \|\| 'E8E9626364656667'x<br>  \|\| '6871727374757677'x<br>  \|\| '787B80ACADAE9EEB'x<br>  \|\| 'ECEDEEEFFBFCFDFE'x<br>  ) ); |
| dcl lower_00285 char<br>  value( (<br>  '8182838485868788'x<br>  \|\| '8991929394959697'x<br>  \|\| '9899A2A3A4A5A6A7'x<br>  \|\| 'A8A9424344454647'x<br>  \|\| '4849515253545556'x<br>  \|\| '5758708C8D8E9CCB'x<br>  \|\| 'CCCDCECFDBDCDDDE'x<br>  ) ); | dcl upper_00285 char<br>  value( (<br>  'C1C2C3C4C5C6C7C8'x<br>  \|\| 'C9D1D2D3D4D5D6D7'x<br>  \|\| 'D8D9E2E3E4E5E6E7'x<br>  \|\| 'E8E9626364656667'x<br>  \|\| '6869717273747576'x<br>  \|\| '777880ACADAE9EEB'x<br>  \|\| 'ECEDEEEFFBFCFDFE'x<br>  ) ); |
| dcl lower_00297 char<br>  value( (<br>  '8182838485868788'x<br>  \|\| '8991929394959697'x<br>  \|\| '9899A2A3A4A5A6A7'x<br>  \|\| 'A8A9424345464749'x<br>  \|\| '5253555657586A70'x<br>  \|\| '7C8C8D8E9CC0CBCC'x<br>  \|\| 'CDCECFD0DBDCDEE0'x<br>  ) ); | dcl upper_00297 char<br>  value( (<br>  'C1C2C3C4C5C6C7C8'x<br>  \|\| 'C9D1D2D3D4D5D6D7'x<br>  \|\| 'D8D9E2E3E4E5E6E7'x<br>  \|\| 'E8E9626365666769'x<br>  \|\| '727375767778FD80'x<br>  \|\| '64ACADAE9E71EBEC'x<br>  \|\| 'EDEEEF74FBFCFE68'x<br>  ) ); |
| dcl lower_00500 char<br>  value( (<br>  '8182838485868788'x<br>  \|\| '8991929394959697'x<br>  \|\| '9899A2A3A4A5A6A7'x<br>  \|\| 'A8A9424344454647'x<br>  \|\| '4849515253545556'x<br>  \|\| '5758708C8D8E9CCB'x<br>  \|\| 'CCCDCECFDBDCDDDE'x<br>  ) ); | dcl upper_00500 char<br>  value( (<br>  'C1C2C3C4C5C6C7C8'x<br>  \|\| 'C9D1D2D3D4D5D6D7'x<br>  \|\| 'D8D9E2E3E4E5E6E7'x<br>  \|\| 'E8E9626364656667'x<br>  \|\| '6869717273747576'x<br>  \|\| '777880ACADAE9EEB'x<br>  \|\| 'ECEDEEEFFBFCFDFE'x<br>  ) ); |

表 *87. LOWERCASE* 組み込み関数および *UPPERCASE* 組み込み関数に対してサポートされているコード・ページ値 *(*続き*)*

| 下限 | 上限 |
|---|---|
| dcl lower_00813 char<br>  value( (<br>  '6162636465666768'x<br>  \|\| '696A6B6C6D6E6F70'x<br>  \|\| '7172737475767778'x<br>  \|\| '797ADCDDDEDFE1E2'x<br>  \|\| 'E3E4E5E6E7E8E9EA'x<br>  \|\| 'EBECEDEEEFF0F1F3'x<br>  \|\| 'F4F5F6F7F8F9FAFB'x<br>  \|\| 'FCFDFE'x<br>  ) ); | dcl upper_00813 char<br>  value( (<br>  '4142434445464748'x<br>  \|\| '494A4B4C4D4E4F50'x<br>  \|\| '5152535455565758'x<br>  \|\| '595AB6B8B9BAC1C2'x<br>  \|\| 'C3C4C5C6C7C8C9CA'x<br>  \|\| 'CBCCCDCECFD0D1D3'x<br>  \|\| 'D4D5D6D7D8D9DADB'x<br>  \|\| 'BCBEBF'x<br>  ) ); |
| dcl lower_00819 char<br>  value( (<br>  '6162636465666768'x<br>  \|\| '696A6B6C6D6E6F70'x<br>  \|\| '7172737475767778'x<br>  \|\| '797AE0E1E2E3E4E5'x<br>  \|\| 'E6E7E8E9EAEBECED'x<br>  \|\| 'EEEFF0F1F2F3F4F5'x<br>  \|\| 'F6F8F9FAFBFCFDFE'x<br>  ) ); | dcl upper_00819 char<br>  value( (<br>  '4142434445464748'x<br>  \|\| '494A4B4C4D4E4F50'x<br>  \|\| '5152535455565758'x<br>  \|\| '595AC0C1C2C3C4C5'x<br>  \|\| 'C6C7C8C9CACBCCCD'x<br>  \|\| 'CECFD0D1D2D3D4D5'x<br>  \|\| 'D6D8D9DADBDCDDDE'x<br>  ) ); |
| dcl lower_00850 char<br>  value( (<br>  '6162636465666768'x<br>  \|\| '696A6B6C6D6E6F70'x<br>  \|\| '7172737475767778'x<br>  \|\| '797A818283848586'x<br>  \|\| '8788898A8B8C8D91'x<br>  \|\| '93949596979BA0A1'x<br>  \|\| 'A2A3A4C6D0E4E7EC'x<br>  ) ); | dcl upper_00850 char<br>  value( (<br>  '4142434445464748'x<br>  \|\| '494A4B4C4D4E4F50'x<br>  \|\| '5152535455565758'x<br>  \|\| '595A9A90B68EB78F'x<br>  \|\| '80D2D3D4D8D7DE92'x<br>  \|\| 'E299E3EAEB9DB5D6'x<br>  \|\| 'E0E9A5C7D1E5E8ED'x<br>  ) ); |
| dcl lower_00858 char<br>  value( (<br>  '6162636465666768'x<br>  \|\| '696A6B6C6D6E6F70'x<br>  \|\| '7172737475767778'x<br>  \|\| '797A818283848586'x<br>  \|\| '8788898A8B8C8D91'x<br>  \|\| '93949596979BA0A1'x<br>  \|\| 'A2A3A4C6D0E4E7EC'x<br>  ) ); | dcl upper_00858 char<br>  value( (<br>  '4142434445464748'x<br>  \|\| '494A4B4C4D4E4F50'x<br>  \|\| '5152535455565758'x<br>  \|\| '595A9A90B68EB78F'x<br>  \|\| '80D2D3D4D8D7DE92'x<br>  \|\| 'E299E3EAEB9DB5D6'x<br>  \|\| 'E0E9A5C7D1E5E8ED'x<br>  ) ); |
| dcl lower_00871 char<br>  value( (<br>  '8182838485868788'x<br>  \|\| '8991929394959697'x<br>  \|\| '9899A2A3A4A5A6A7'x<br>  \|\| 'A8A9424344454647'x<br>  \|\| '4849515253545556'x<br>  \|\| '575870798DA1C0CB'x<br>  \|\| 'CDCECFD0DBDCDDDE'x<br>  ) ); | dcl upper_00871 char<br>  value( (<br>  'C1C2C3C4C5C6C7C8'x<br>  \|\| 'C9D1D2D3D4D5D6D7'x<br>  \|\| 'D8D9E2E3E4E5E6E7'x<br>  \|\| 'E8E9626364656667'x<br>  \|\| '6869717273747576'x<br>  \|\| '7778807CAD5F4AEB'x<br>  \|\| 'EDEEEF5AFBFCFDFE'x<br>  ) ); |

表 87. *LOWERCASE* 組み込み関数および *UPPERCASE* 組み込み関数に対してサポートされているコード・ページ値 (続き)

| 下限 | 上限 |
| --- | --- |
| dcl lower_00920 char<br>  value( (<br>  '6162636465666768'x<br>  \|\| '696A6B6C6D6E6F70'x<br>  \|\| '7172737475767778'x<br>  \|\| '797AE0E1E2E3E4E5'x<br>  \|\| 'E6E7E8E9EAEBECED'x<br>  \|\| 'EEEFF0F1F2F3F4F5'x<br>  \|\| 'F6F8F9FAFBFCFE'x<br>  ) ); | dcl upper_00920 char<br>  value( (<br>  '4142434445464748'x<br>  \|\| '494A4B4C4D4E4F50'x<br>  \|\| '5152535455565758'x<br>  \|\| '595AC0C1C2C3C4C5'x<br>  \|\| 'C6C7C8C9CACBCCCD'x<br>  \|\| 'CECFD0D1D2D3D4D5'x<br>  \|\| 'D6D8D9DADBDCDE'x<br>  ) ); |
| dcl lower_01026 char<br>  value( (<br>  '8182838485868788'x<br>  \|\| '8991929394959697'x<br>  \|\| '9899A2A3A4A5A6A7'x<br>  \|\| 'A8A9424344454647'x<br>  \|\| '4951525354555657'x<br>  \|\| '586A709CA1C0CBCD'x<br>  \|\| 'CECFD0DBDDDEE0'x<br>  ) ); | dcl upper_01026 char<br>  value( (<br>  'C1C2C3C4C5C6C7C8'x<br>  \|\| 'C9D1D2D3D4D5D6D7'x<br>  \|\| 'D8D9E2E3E4E5E6E7'x<br>  \|\| 'E8E9626364656667'x<br>  \|\| '6971727374757677'x<br>  \|\| '787C809E7B4AEBED'x<br>  \|\| 'EEEF5AFBFDFE7F'x<br>  ) ); |
| dcl lower_01047 char<br>  value( (<br>  '8182838485868788'x<br>  \|\| '8991929394959697'x<br>  \|\| '9899A2A3A4A5A6A7'x<br>  \|\| 'A8A9424344454647'x<br>  \|\| '4849515253545556'x<br>  \|\| '5758708C8D8E9CCB'x<br>  \|\| 'CCCDCECFDBDCDDDE'x<br>  ) ); | dcl upper_01047 char<br>  value( (<br>  'C1C2C3C4C5C6C7C8'x<br>  \|\| 'C9D1D2D3D4D5D6D7'x<br>  \|\| 'D8D9E2E3E4E5E6E7'x<br>  \|\| 'E8E9626364656667'x<br>  \|\| '6869717273747576'x<br>  \|\| '777880ACBAAE9EEB'x<br>  \|\| 'ECEDEEEFFBFCFDFE'x<br>  ) ); |
| dcl lower_01140 char<br>  value( (<br>  '8182838485868788'x<br>  \|\| '8991929394959697'x<br>  \|\| '9899A2A3A4A5A6A7'x<br>  \|\| 'A8A9424344454647'x<br>  \|\| '4849515253545556'x<br>  \|\| '5758708C8D8E9CCB'x<br>  \|\| 'CCCDCECFDBDCDDDE'x<br>  ) ); | dcl upper_01140 char<br>  value( (<br>  'C1C2C3C4C5C6C7C8'x<br>  \|\| 'C9D1D2D3D4D5D6D7'x<br>  \|\| 'D8D9E2E3E4E5E6E7'x<br>  \|\| 'E8E9626364656667'x<br>  \|\| '6869717273747576'x<br>  \|\| '777880ACADAE9EEB'x<br>  \|\| 'ECEDEEEFFBFCFDFE'x<br>  ) ); |
| dcl lower_01141 char<br>  value( (<br>  '8182838485868788'x<br>  \|\| '8991929394959697'x<br>  \|\| '9899A2A3A4A5A6A7'x<br>  \|\| 'A8A9424445464748'x<br>  \|\| '4951525354555657'x<br>  \|\| '586A708C8D8E9CC0'x<br>  \|\| 'CBCDCECFD0DBDDDE'x<br>  ) ); | dcl upper_01141 char<br>  value( (<br>  'C1C2C3C4C5C6C7C8'x<br>  \|\| 'C9D1D2D3D4D5D6D7'x<br>  \|\| 'D8D9E2E3E4E5E6E7'x<br>  \|\| 'E8E9626465666768'x<br>  \|\| '6971727374757677'x<br>  \|\| '78E080ACADAE9E4A'x<br>  \|\| 'EBEDEEEF5AFBFDFE'x<br>  ) ); |
| dcl lower_01142 char<br>  value( (<br>  '8182838485868788'x<br>  \|\| '8991929394959697'x<br>  \|\| '9899A2A3A4A5A6A7'x<br>  \|\| 'A8A9424344454648'x<br>  \|\| '4951525354555657'x<br>  \|\| '586A8C8D8EA1C0CB'x<br>  \|\| 'CCCDCECFD0DBDDDE'x<br>  ) ); | dcl upper_01142 char<br>  value( (<br>  'C1C2C3C4C5C6C7C8'x<br>  \|\| 'C9D1D2D3D4D5D6D7'x<br>  \|\| 'D8D9E2E3E4E5E6E7'x<br>  \|\| 'E8E9626364656668'x<br>  \|\| '6971727374757677'x<br>  \|\| '787CACADAEFC7BEB'x<br>  \|\| 'ECEDEEEF5BFBFDFE'x<br>  ) ); |

表 87. *LOWERCASE* 組み込み関数および *UPPERCASE* 組み込み関数に対してサポートされているコード・ページ値 (続き)

| 下限 | 上限 |
|---|---|
| dcl lower_01143 char<br>  value( (<br>  '8182838485868788'x<br>  \|\| '8991929394959697'x<br>  \|\| '9899A2A3A4A5A6A7'x<br>  \|\| 'A8A9424445464849'x<br>  \|\| '525354555657586A'x<br>  \|\| '70798C8D8E9CA1C0'x<br>  \|\| 'CBCDCECFD0DBDDDE'x<br>  ) ); | dcl upper_01143 char<br>  value( (<br>  'C1C2C3C4C5C6C7C8'x<br>  \|\| 'C9D1D2D3D4D5D6D7'x<br>  \|\| 'D8D9E2E3E4E5E6E7'x<br>  \|\| 'E8E9626465666869'x<br>  \|\| '7273747577677787C'x<br>  \|\| '80E0ACADAE9EFC7B'x<br>  \|\| 'EBEDEEEF5BFBFDFE'x<br>  ) ); |
| dcl lower_01144 char<br>  value( (<br>  '8182838485868788'x<br>  \|\| '8991929394959697'x<br>  \|\| '9899A2A3A4A5A6A7'x<br>  \|\| 'A8A9424345464749'x<br>  \|\| '52535556575A6A70'x<br>  \|\| '798C8D8E9CA1C0CB'x<br>  \|\| 'CCCECFD0DBDCDEE0'x<br>  ) ); | dcl upper_01144 char<br>  value( (<br>  'C1C2C3C4C5C6C7C8'x<br>  \|\| 'C9D1D2D3D4D5D6D7'x<br>  \|\| 'D8D9E2E3E4E5E6E7'x<br>  \|\| 'E8E9626365666769'x<br>  \|\| '727375767771ED80'x<br>  \|\| 'FDACADAE9E7864EB'x<br>  \|\| 'ECEEEF74FBFCFE68'x<br>  ) ); |
| dcl lower_01145 char<br>  value( (<br>  '8182838485868788'x<br>  \|\| '8991929394959697'x<br>  \|\| '9899A2A3A4A5A6A7'x<br>  \|\| 'A8A9424344454647'x<br>  \|\| '4851525354555657'x<br>  \|\| '586A708C8D8E9CCB'x<br>  \|\| 'CCCDCECFDBDCDDDE'x<br>  ) ); | dcl upper_01145 char<br>  value( (<br>  'C1C2C3C4C5C6C7C8'x<br>  \|\| 'C9D1D2D3D4D5D6D7'x<br>  \|\| 'D8D9E2E3E4E5E6E7'x<br>  \|\| 'E8E9626364656667'x<br>  \|\| '6871727374757677'x<br>  \|\| '787B80ACADAE9EEB'x<br>  \|\| 'ECEDEEEFFBFCFDFE'x<br>  ) ); |
| dcl lower_01146 char<br>  value( (<br>  '8182838485868788'x<br>  \|\| '8991929394959697'x<br>  \|\| '9899A2A3A4A5A6A7'x<br>  \|\| 'A8A9424344454647'x<br>  \|\| '4849515253545556'x<br>  \|\| '5758708C8D8E9CCB'x<br>  \|\| 'CCCDCECFDBDCDDDE'x<br>  ) ); | dcl upper_01146 char<br>  value( (<br>  'C1C2C3C4C5C6C7C8'x<br>  \|\| 'C9D1D2D3D4D5D6D7'x<br>  \|\| 'D8D9E2E3E4E5E6E7'x<br>  \|\| 'E8E9626364656667'x<br>  \|\| '6869717273747576'x<br>  \|\| '777880ACADAE9EEB'x<br>  \|\| 'ECEDEEEFFBFCFDFE'x<br>  ) ); |
| dcl lower_01147 char<br>  value( (<br>  '8182838485868788'x<br>  \|\| '8991929394959697'x<br>  \|\| '9899A2A3A4A5A6A7'x<br>  \|\| 'A8A9424345464749'x<br>  \|\| '5253555657586A70'x<br>  \|\| '7C8C8D8E9CC0CBCC'x<br>  \|\| 'CDCECFD0DBDCDEE0'x<br>  ) ); | dcl upper_01147 char<br>  value( (<br>  'C1C2C3C4C5C6C7C8'x<br>  \|\| 'C9D1D2D3D4D5D6D7'x<br>  \|\| 'D8D9E2E3E4E5E6E7'x<br>  \|\| 'E8E9626365666769'x<br>  \|\| '727375767778FD80'x<br>  \|\| '64ACADAE9E71EBEC'x<br>  \|\| 'EDEEEF74FBFCFE68'x<br>  ) ); |
| dcl lower_01148 char<br>  value( (<br>  '8182838485868788'x<br>  \|\| '8991929394959697'x<br>  \|\| '9899A2A3A4A5A6A7'x<br>  \|\| 'A8A9424344454647'x<br>  \|\| '4849515253545556'x<br>  \|\| '5758708C8D8E9CCB'x<br>  \|\| 'CCCDCECFDBDCDDDE'x<br>  ) ); | dcl upper_01148 char<br>  value( (<br>  'C1C2C3C4C5C6C7C8'x<br>  \|\| 'C9D1D2D3D4D5D6D7'x<br>  \|\| 'D8D9E2E3E4E5E6E7'x<br>  \|\| 'E8E9626364656667'x<br>  \|\| '6869717273747576'x<br>  \|\| '777880ACADAE9EEB'x<br>  \|\| 'ECEDEEEFFBFCFDFE'x<br>  ) ); |

表 87. *LOWERCASE* 組み込み関数および *UPPERCASE* 組み込み関数に対してサポートされているコード・ページ値 (続き)

| 下限 | 上限 |
| --- | --- |
| dcl lower_01149 char<br>  value( (<br>  '8182838485868788'x<br>  \|\| '8991929394959697'x<br>  \|\| '9899A2A3A4A5A6A7'x<br>  \|\| 'A8A9424344454647'x<br>  \|\| '4849515253545556'x<br>  \|\| '575870798DA1C0CB'x<br>  \|\| 'CDCECFD0DBDCDDDE'x<br>  ) ); | dcl upper_01149 char<br>  value( (<br>  'C1C2C3C4C5C6C7C8'x<br>  \|\| 'C9D1D2D3D4D5D6D7'x<br>  \|\| 'D8D9E2E3E4E5E6E7'x<br>  \|\| 'E8E9626364656667'x<br>  \|\| '6869717273747576'x<br>  \|\| '7778807CAD5F4AEB'x<br>  \|\| 'EDEEEF5AFBFCFDFE'x<br>  ) ); |
| dcl lower_01155 char<br>  value( (<br>  '8182838485868788'x<br>  \|\| '8991929394959697'x<br>  \|\| '9899A2A3A4A5A6A7'x<br>  \|\| 'A8A9424344454647'x<br>  \|\| '4951525354555657'x<br>  \|\| '586A709CA1C0CBCD'x<br>  \|\| 'CECFD0DBDDDEE0'x<br>  ) ); | dcl upper_01155 char<br>  value( (<br>  'C1C2C3C4C5C6C7C8'x<br>  \|\| 'C9D1D2D3D4D5D6D7'x<br>  \|\| 'D8D9E2E3E4E5E6E7'x<br>  \|\| 'E8E9626364656667'x<br>  \|\| '6971727374757677'x<br>  \|\| '787C809E7B4AEBED'x<br>  \|\| 'EEEF5AFBFDFE7F'x<br>  ) ); |

# 特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものです。 本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。 日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。 本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。 これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の 製品、プログラム、またはサービスを使用することができます。 ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。 本書の提供は、お客様にこれらの特許権について 実施権を許諾することを意味するものではありません。 実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

IBM Corporation
J74/G4
IBM World Trade Asia Corporation
法務・知的財産
知的財産権ライセンス渉外

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
:NONE.
IBM Japan, Ltd.
<!--3-2-12, Roppongi, Minato-ku, Tokyo 106-8711-->

以下の保証は、国または地域の法律に沿わない場合は、適用されません。

インターナショナル・ビジネス・マシンズ・コーポレーション が提供
本書は、特定物として現存するままの状態で提供され、
第三者の権利の不侵害の保証、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任なしで提供されています。
「商品性」または「特定目的適合性」

国または地域によっては、法律の強行規定により、保証責任の制限が 禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。

本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。 IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、 改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものでは ありません。 それらの Web サイトにある資料は、この IBM 製品の資料の一部では ありません。それらの Web サイトは、お客様の責任でご使用ください。

# 商標

IBM、IBM ロゴ、および ibm.com® は、世界の多くの国で登録された International Business Machines Corporation の商標です。 世界中の多くの国で登録されています。 他の製品名およびサービス名等は、それぞれ IBM または各社の商標である場合があります。 現時点での IBM の商標リストについては、"著作権および商標情報" ( www.ibm.com/legal/copytrade) をご覧ください。

Intel および Pentium は、Intel Corporation の米国およびその他の国における商標です。

Java™ およびすべての Java 関連の商標およびロゴは Oracle やその関連会社の商標または登録商標です。

© Copyright IBM Corp. 1999, 2022

Microsoft、Windows、および Windows NT は、Microsoft Corporation の米国およびその他の国における商標です。

Pentium は Intel Corporation または子会社の米国およびその他の国における商標または登録商標です。

:NONE.

UNIX は The Open Group の米国およびその他の国における登録商標です。

<!-- 他の会社名、製品名およびサービス名などはそれぞれ各社の商標または登録商標です。-->

この情報をソフトコピーでご覧になっている場合は、写真やカラーの図表は 表示されない場合があります。

# 参照文献

## PL/I 資料

### Enterprise PL/I for z/OS

プログラミング・ガイド, GI13-5620
言語リファレンス, SC31-5716
メッセージとコード, GC31-5717
コンパイラーおよびランタイムマイグレーション・ガイド, GC31-5715

### PL/I ( MVS™ & VM 版)

「インストールとカストマイズ」、SC88-7221
「言語解説書」、SC88-7219
「コンパイル時メッセージおよびコード」、SC88-7224
「診断の手引き」、SC88-7223
「移行の手引き」、SC88-7220
「プログラミング・ガイド」、SC88-7218
「参照要約」、SX88-7011

### PL/I for AIX

「プログラミング・ガイド」、SA88-4427
「言語解説書」、SA88-4429
「メッセージおよびコード」、GA88-4430
「インストール・ガイド」、GA88-4428

## 関連資料

### Db2 for z/OS

「管理ガイド」、SC43-3430
「アプリケーション・プログラミングおよび *SQL* 解説書」、SC43-3444
「コマンド解説書」、SC43-3431
「メッセージ」、GC43-3438
「コード」、GC43-3442
「*SQL* 解説書」、SC43-3449
「*LOBs with Db2 for z/OS: Stronger and Faster*」、SG24-7270
Db2 for z/OS Product Documentation も参照してください

### DFSORT™

「アプリケーション・プログラミング・ガイド」、SA88-5607
「インストールおよびカスタマイズ」、SA88-7075

### IMS/ESAR

「アプリケーション・プログラミング: データベース管理プログラム」、SC88-7552
「*Application Programming: Database Manager Summary*」、SC26-8037

© Copyright IBM Corp. 1999, 2022

「アプリケーション・プログラミング: 設計の手引き」、SC88-7542
「アプリケーション・プログラミング: トランザクション管理プログラム」、SC88-7553
「*Application Programming: Transaction Manager Summary*」、SC26-8038
「アプリケーション・プログラミング: *EXEC DL/I* コマンド *(CICS* および *IMS™)*」、SC88-7554
「*Application Programming: EXEC DL/I Commands for CICS and IMS Summary*」、SC26-8036
「*IMS/ESA V6R1 Bookindex*」、GC27-1557

### TXSeries for Multiplatforms

「*Encina* 管理の手引き 第 *2* 巻：サーバー管理」、SD88-7403
「*Encina SFS Programming Guide*」、SC09-4483
TXSeries for Multiplatforms Knowledge Center を参照してください

### z/Architecture

「解説書」、SA88-8773
オンラインの *Principles of Operation* を参照してください

### z/OS 言語環境プログラム

「概念」、SC43-2659
「デバッグ・ガイド」、GA88-5524
「ランタイム・メッセージ」、SA88-7094
「カスタマイズ」、SA88-7096
「プログラミング・ガイド」、SA88-7098
「*64* ビット仮想アドレッシング・モード向けプログラミング・ガイド」、SA88-7101
「プログラミング・リファレンス」、SA88-7099
「ランタイム・アプリケーション マイグレーション・ガイド」、GC43-2660
「*Vendor Interfaces*」、SA38-0688
「言語間通信 アプリケーションの作成」、SC43-2661
z/OS Language Environment Knowledge Center も参照してください

### z/OS MVS

「*JCL* 解説書」、SA88-7091
「*JCL* ユーザーズ・ガイド」、SA88-5468
「システム・コマンド」、SA88-5490
z/OS MVS Knowledge Center を参照してください

### z/OS TSO/E

「コマンド解説書」、SA88-7049
「ユーザーズ・ガイド」、SA88-5454

### z/OS UNIX システム・サービス

「*z/OS UNIX System Services* コマンド解説書」、SA88-5452
「*z/OS UNIX System Services* プログラミング: アセンブラー呼び出し可能サービス 解説書」、SA88-7054
「*z/OS UNIX System Services* ユーザーズ・ガイド」、SA88-7053

### Unicode および文字表現

「*OS/390 Unicode* サポート: 変換サービスの使用法」、SD88-6163
「*z/OS Unicode* サービス ユーザーズ・ガイドおよびリファレンス」、SA88-7093

# 用語集

この用語集では、PL/I のすべてのプラットフォームおよびリリースについて用語を定義します。 本書では使用しない用語が含まれている場合があります。 該当する用語が見つからない場合は、本書の索引を調べるか、「*IBM Dictionary of Computing*」(SC20-1699) を参照してください。

**A**

**アクセス**
データを参照するかまたは取り出すこと。

**処置指定 (action specification)**
ON ステートメント内にある、ON ユニットまたは単一のキーワード SYSTEM。 該当する条件が発生すれば、2 つのうちいずれかがとるべき処置を指定する。

**活動化 (ブロックの) (activate (a block))**
ブロックの実行を開始すること。 プロシージャー・ブロックは、呼び出されるときに、活動化される。 開始ブロックが活動化するのは、分岐を含め、通常の制御の流れ内に現れたと きである。 パッケージを活動化することはできない。

**活動化 (プリプロセッサー変数またはプリプロセッサー・エントリー・ポイントの) (activate (a preprocessor variable or preprocessor entry point))**
マクロ機能 ID を、それに後続するソース・コード内で置換 可能にすること。 %ACTIVATE ステートメントは、プリプロセッサー変数や プリプロセッサー・エントリー・ポイントを活動化する。

**アクティブ**
活動化から終了にいたるまでのブロックの状態。 ソース・プログラム・テキスト中の対応する ID をプリプロセッサー変数や プリプロセッサー入り口名の値に置き換えることができるときの、 その変数や入り口の状態。 イベント変数が非同期操作に関連付けられている間に置かれている状態。 タスク変数に関連するタスクが付加されるときにタスク変数が 置かれている状態。 タスクが終了する前に置かれている状態。

**実際の起点 (actual origin (AO))**
配列または構造体内の最初の項目の位置。

**追加属性 (additive attribute)**
デフォルトがないファイル記述属性。必要に応じて、明示的に指定された別の属性によって明示的または暗黙的に指定する必要があります。 代替属性と対比。

**調節可能エクステント (adjustable extent)**
関連変数の世代によって異なることのある境界 (配列の)、 長さ (ストリングの)、またはサイズ (区域の)。 調節可能エクステントは、世代ごとに別々に評価される式またはアスタリスク (ただし、 基底付き変数の場合は REFER オプション) で指定される。 静的変数に使用することはできない。

**集合 (aggregate)**
「データ集合 *(data aggregate)*」を参照。

**集合式 (aggregate expression)**
配列式、構造式、または共用体式のこと。

**集合タイプ (aggregate type)**
どのデータ項目の場合も、それが構造体、共用体、または配列の いずれであるかの指定。

**割り振られた変数 (allocated variable)**
主記憶域が関連付けられ解放されない変数。

**割り振り**
変数用の主記憶域の予約。 割り振られた変数の世代。 PL/I ファイルを、システム・データ・セット、装置、またはファイルに 関連付けること。

**位置合わせ**
機械に依存する特定の境界 (例えば、フルワード境界 またはハーフワード境界) に関連付けて、データ項目を保管すること。

© Copyright IBM Corp. 1999, 2022

**英字 (alphabetic character)**
A から Z までの任意の英字と、#、$、@ (これらのグラフィック表記は国によって異なる 場合がある) の拡張英字。

**英数字 (alphameric character)**
英字または数字。

**代替属性 (alternative attribute)**
属性グループから選択するファイル記述属性。 何も指定しないと、デフォルトがとられる。 「追加属性 *(additive attribute)*」と対比。

**あいまい参照 (ambiguous reference)**
参照時点で認識されている名前をただ 1 つだけ識別するためには 修飾が不十分な参照。

**領域**
基底付き変数を割り振ることのできる、ストレージ中の部分。

**argument**
サブルーチンまたは機能の呼び出しの一部である引数リスト内 にある式。

**引数リスト (argument list)**
コンマで区切られ、入り口名定数、入り口名変数、総称名、または組み込み関数名に 続く、括弧で囲まれたゼロまたはそれ以上の引数のリスト。 そのリストは、エントリー・ポイントのパラメーター・リストである。

**算術比較 (arithmetic comparison)**
数値の比較。 「ビット比較 *(bit comparison)*」、「文字比較 *(character comparison)*」も参照。

**算術定数 (arithmetic constant)**
固定小数点定数または浮動小数点定数。 大部分の算術定数には符号を付けることができるが、符号は定数の 一部ではない。

**算術変換 (arithmetic conversion)**
ある 1 つの算術表現から別の表現に値を変換すること。

**算術データ**
基数、スケール、モード、および精度の特性を持つデータ。 コード化算術データとピクチャー数字データも含まれる。

**算術演算子**
接頭演算子の + と -、あるいは挿入演算子 + - * / ** のうちのいずれか。

**配列**
同じ属性を持ち、1 つ以上の次元別にグループ分けされた 1 つ以上のデータ・エレメントに、名前を付けて順番に並べた集合体。

**配列式 (array expression)**
評価されると値の配列が生成される式。

**構造体の配列 (array of structures)**
次元属性を構造体名に与えて指定される、順番に並べられた同一構造体の 集まり。

**配列変数**
同じ属性を持っていなければならないデータ項目の 集合を表す変数。 「構造変数 *(structure variable)*」と対比。

**ASCII**
情報交換用米国標準コード (American National Standard Code for Information Interchange)。

**代入 (assignment)**
値を変数に与える処理。

**非同期操作 (asynchronous operation)**
ステートメントの実行と、入出力操作が並行して行われること。 各種タスクに複数の制御の流れを使った、プロシージャーの 並行実行。

**タスクの生成 (attachment of a task)**
呼び込まれたプロシージャー (およびこれが呼び出す プロシージャー) を、呼び出しプロシージャーの実行と一緒に、非同期で 実行するために、プロシージャーを呼び出して別に制御の流れを確立 すること。

**アテンション**
タスクに割り込みが生じる原因となるような、タスクにとっては外部の事柄の発生。

**属性**
表明された特性を記述するのに名前と関連付けた記述特性。 式の計算の結果の特性を説明するために用いられる記述特性。

**自動ストレージ割り振り (automatic storage allocation)**
自動変数用のストレージ割り振り。

**自動変数 (automatic variable)**
ブロックの起動時に自動的にストレージを割り振られ、そのブロックの 終了時に自動的にそれを解除される変数。

**B**

**ベース**
算術値を表現するための数体系。

**基本エレメント (base element)**
それ自身は別の構造体や共用体ではない、構造体や共用体のメンバー。

**基本項目 (base item)**
定義変数を定義するための、自動、被制御、または静的変数、 またはパラメーター。

**基底付き参照 (based reference)**
基底付きストレージ・クラスを持った参照。

**基底付きストレージ割り振り (based storage allocation)**
基底付き変数用のストレージの割り振り。

**基底付き変数 (based variable)**
ストレージ・アドレスがロケーターによって与えられる変数。 同一変数の複数の世代をアクセスすることができる。 これは、ストレージ内の固定位置を識別しない。

**開始ブロック (begin-block)**
BEGIN ステートメントと END ステートメントによって区切られ、名前有効範囲を形成するステートメントの集まり。 開始ブロックの活動化は、条件が生じたために行われる (開始ブロックが ON ユニットへの処置指定である場合) か、または GOTO ステートメントの 結果の分岐を含め、通常の制御の流れを介して行われる。

**2 進数 (binary)**
0 と 1 が唯一の数表示である数体系。

**2 進数字 (binary digit)**
「ビット *(bit)*」を参照。

**2 進固定小数点値 (binary fixed-point value)**
2 進数字で構成され、オプションの 2 進小数点とオプションの符号を持った整数。 「*10* 進固定小数点値 *(decimal fixed-point value)*」と対比。

**2 進浮動小数点値 (binary floating-point value)**
2 進小数部と見なすことができる仮数と、2 の基数に対する整数指数と見なすことができる指数の形式での実数の近似値。 *10* 進浮動小数値と対比。

**bit**
0 または 1 です。 コンピューター・ストレージの最小スペース量。

**ビット比較 (bit comparison)**
2 進数字を、左から右へビットごとに比較すること。 「算術比較 *(arithmetic comparison)*」、「文字比較 *(character comparison)*」も参照。

**ビット・ストリング定数 (bit string constant)**
一連の 2 進数字で囲まれ、直後に接尾部 B が付きます。 文字定数と対比。 単一引用符に囲まれ、後に接尾部 B4 が付いた 16 進数字の連なり。

**ビット・ストリング (bit string)**
ゼロ以上のビットで構成されたストリング。

**ビット・ストリング演算子 (bit string operators)**
論理演算子 NOT と排他 OR (¬)、 AND (&)、および OR (|)。

**ビット値 (bit value)**
ビット・タイプを表す値。

**ブロック**
その中で宣言された名前の有効範囲と、その名前用のストレージ割り振りを 指定する、1 つの単位として処理される一連のステートメント。 ブロックとしては、パッケージ、プロシージャー、または開始ブロックの いずれもありえる。

**境界 (bounds)**
任意の配列次元の上限と下限。

**区切り文字 (break character)**
下線記号 (_)。 これは、ID の読みやすさを向上させるために使用できます。 例えば、変数を OLDINVENTORYTOTAL とする代わりに OLD_INVENTORY_TOTAL と記述できる。

**組み込み関数 (built in function)**
SQRT (平方根) のような、言語が提供する定義済み関数。

**組み込み関数参照 (built-in function reference)**
オプションの引数リストを持つ組み込み関数名。

**組み込み名 (built-in name)**
組み込みサブルーチンの入り口名。

**組み込みサブルーチン (built-in subroutine)**
コンパイル時に定義され、CALL ステートメントによって呼び出される 入り口名を持つサブルーチン。

**緩衝域**
レコードが入力時に読み込まれ、レコードが出力時に書き出される、 入出力操作に使用する中間記憶域。

## C

**call**
CALL ステートメントまたは CALL オプションを使用してサブルーチンを 呼び出すこと。

**文字比較 (character comparison)**
照合順序に従って、左から右へ文字単位で行われる比較。「算術比較 *(arithmetic comparison)*」、「ビット比較 *(bit comparison)*」も参照。

**文字ストリング定数 (character string constant)**
単一引用符で囲まれた一連の文字。例えば、'Shakespeare''s 'Hamlet:'' など。

**文字セット (character set)**
あらかじめ決められた文字の集まり。 *ASCII* および *EBCDIC* も参照してください。

**文字ストリング・ピクチャー・データ (character string picture data)**
文字値だけを持ったピクチャー・データ。 このタイプのピクチャー・データは、少なくとも 1 つの A または X ピクチャー指定文字を持っていなければならない。 「数値ピクチャー・データ *(numeric picture data)*」と対比。

**クローズ (ファイルの) (closing (of a file))**
ファイルをデータ・セットまたは装置と切り離すこと。

**コード化算術データ**
数値を表し、基数 (10 進数または 2 進数)、スケール (固定小数点または浮動小数点)、および精度 (個々に持ちうる桁数) を特徴とするデータ項目。 このデータは、変換しなくても、算術計算用に受け入れることのできる 形式で保管される。

**複合ネストの深さ (combined nesting depth)**
ネストの最も深いレベル。手順/開始/開始、実行、選択、および IF のレベルをカウントすることにより決定されます。THEN...プログラム内に ELSE ネストがあります。

**comment**
文書化のために使用され、/* および */ で区切られる、ゼロ以上 の文字数の文字ストリング。

**商用文字 (commercial character)**

- CR (貸方) ピクチャー指定文字。
- DB (借方) ピクチャー指定文字。

**比較演算子 (comparison operator)**
関係内の項目を相互に比較するよう指示するための算術、ストリング・ロケーター、 または論理関係で使用される演算子。 比較演算子は次のとおり。

= (に等しい)
> (より大)
< (より小)
>= (より大きいか等しい)
<= (より小か等しい)
¬= または <> (等しくない)
¬> (より大ではない)
¬< (より小ではない)

**コンパイル時 (間) (compile time)**
一般に、ソース・プログラムがオブジェクト・モジュールに 変換されている時間。 PL/I では、変更したい場合に、ソース・プログラムを変更して、 オブジェクト・プログラムに変換し終わるまでに経過する時間。

**コンパイラー・オプション (compiler options)**
コンパイルの特定の面を制御するために指定されるキーワード。例えば、 生成するオブジェクト・モジュールの特徴や、作成する印刷出力のタイプなどがある。

**複素数データ (complex data)**
おのおのの項目が実数部と虚数部で構成された算術データ。

**合成演算子 (composite operator)**
<=、**、および /* などの特殊文字を複数含む演算子。

**複合ステートメント (compound statement)**
他のステートメントが含まれているステートメント。 PL/I では、IF、ON、OTHERWISE、および WHEN だけが、複合ステートメントである。 「ステートメント本体 *(statement body)*」を参照。

**連結 (concatenation)**
2 つのストリングを指定順に結合し、元の 2 つのストリング の合計長に等しい長さを持った 1 つのストリングを作成する操作。 これは、演算子 || で指定する。

**condition**
エラー (オーバーフローなど) または予期される状況 (入力ファイルの 終わりなど) のいずれかの例外的な状態。 条件が発生する (検出される) と、その条件に対する規定のアクションが処理される。 「確立された処置 *(established action)*」および「暗黙処置 *(implicit action)*」も参照。

**条件名 (condition name)**
PL/I 定義またはプログラマー定義の条件の名前。

**条件接頭語 (condition prefix)**
ステートメントの接頭部として付けられる、括弧で囲まれた 1 つ以上の条件名のリスト。 条件接頭語は、指定した条件を使用可能にするか使用不能にするかを指定する。

**連結集合 (connected aggregate)**
エレメントが、間にデータ項目の入らない連続したストレージを占有する配列または構造体。 「非連結集合 *(nonconnected aggregate)*」と対比。

**連結参照 (connected reference)**
連結ストレージに対する参照。 プログラムを実行するには、ストレージが連結されていることが明らかでなければならない。

**連結ストレージ (connected storage)**
単一名を使って参照することができる諸項目の非中断かつ線形の 連なりの主記憶域。

**定数 (constant)**
名前が付いておらず、変更できない値を持った、算術または ストリング・データ項目。 VALUE 属性を指定して宣言された ID。 FILE 属性または ENTRY 属性を指定し、VARIABLE 属性を指定しないで宣言された ID。

**定数参照 (constant reference)**
対象として定数を持つ値参照。

**含まれているブロック、宣言、またはソース・テキスト (contained block, declaration, or source text)**
開始、プロシージャー、またはパッケージのブロック内の すべてのブロック、プロシージャー、ステートメント、宣言、または ソース・テキスト。 パッケージ、プロシージャー、および BEGIN ステートメントと それに対応する END ステートメント全体は、ブロック内には含まれて いない。

**収容ブロック (containing block)**
該当する宣言、ステートメント、プロシージャー、またはその他の ソース・テキストを収容している、パッケージ、プロシージャー、または開始ブロック。

**コンテキスト宣言 (contextual declaration)**
DECLARE ステートメントで明示的に宣言されていないが、その使用の前後関係から、特定の属性が ID に関連付けられるような ID の存在。

**制御文字**
特定コンテキスト内に存在することによって 制御機能が指定される、文字セット内の文字。 1 つの例としてファイルの終わり (EOF) マーカーがある。

**制御フォーマット項目 (control format item)**
ストリーム内または印刷ページの内での、あるデータ項目の位置付けを 指定するために、編集指示伝送の中で使用される指定。

**制御変数 (control variable)**
DO ステートメントの反復実行を制御するのに使用する変数。

**被制御パラメーター (controlled parameter)**
DECLARE ステートメント内で CONTROLLED 属性を指定される パラメーター。 これは、CONTROLLED 属性を持った引数としか関連付けることはできない。

**被制御ストレージ割り振り (controlled storage allocation)**
被制御変数用のストレージの割り振り。

**被制御変数 (controlled variable)**
現行世代にだけアクセスすることができ、ALLOCATE と FREE ステートメントによって割り振りと解放が制御される変数。

**制御セクション (control sections)**
オブジェクト・モジュール内のグループ化された機械命令。

**変換**
ある 1 つの表現法から、一組の特定属性に合うよう別の表現法に値を変換すること。 例えば、文字ストリングを FIXED BINARY (15,0) などの算術値に 変換すること。

**配列のクロス・セクション (cross section of an array)**
配列の少なくとも 1 つの次元のエクステントで表すことの できるエレメント。 配列参照内に添え字の代わりにアスタリスクがあれば、それはその次元 のエクステント全体を表す。

**現行世代 (current generation)**
変数名を参照して、現在使用できる自動変数または被制御変数の世代。

**D**

**data**
処理に適合した形式の情報または値の表現。

**データ集合 (data aggregate)**
異なったデータ項目の集まりであるデータ項目。

**データ属性 (data attribute)**
FIXED BINARY などの、データ項目が表すデータのタイプを指定するキーワード。

**データ・ディレクティブ伝送 (data-directed transmission)**
データを伝送するための、ストリーム指向伝送のタイプ。 これは代入ステートメントに似ており、`name = constant` という形式です。

**データ項目**
単一の名前付きデータ単位。

**データ・リスト (data list)**
ストリーム指向伝送における、GET および PUT ステートメント 内で使用するデータ項目を括弧で囲んだリスト。 「フォーマット・リスト *(format list)*」と対比。

**データ・セット**
単一のファイル名の参照によってアクセスすることができる、 プログラムの外部にあるデータの集まり。 参照されることが可能な装置。

**データ指定 (data specification)**
伝送モード (DATA、LIST、または EDIT) を指示し、さらに データ・リストと、編集ディレクティブ・モードの場合はフォーマット・リストを含む、 ストリーム指向伝送ステートメントの一部分。

**データ・ストリーム (data stream)**
ストリーム指向伝送でデータ・セットから、またはデータ・セット へ、文字形式のデータ・エレメントの連続ストリームとして転送されるデータ。

**データ伝送 (data transmission)**
データ・セットからプログラムへ、およびその逆に、データを 転送すること。

**データ・タイプ**
一連のデータ属性。

**DBCS**
文字セットにおいて、それぞれの文字は 2 つの連続するバイトで表される。

**非アクティブ (deactivated)**
ある ID の値で、ソース・プログラム・テキスト内のプリプロセッサー ID を置き換えることができない状態。 「アクティブ *(active)*」と対比。

**デバッグ (debugging)**
プログラムからバグを除去する処理。

**decimal**
0 から 9 までの数字を使った数体系。

**10 進ピクチャー文字 (decimal digit picture character)**
ピクチャー指定文字 9 のこと。

**10 進固定小数点定数 (decimal fixed-point constant)**
1 つ以上の 10 進数 (および任意で小数点を付けたもの) から成る定数。

**10 進固定小数点値 (decimal fixed-point value)**
小数点の想定位置を持つ 10 進数の連なりで構成される有理数。「*2* 進固定小数点値 *(binary fixed-point value)*」と対比。

**10 進浮動小数点定数**
10 進固定小数点定数から成る仮数と、3 桁を超えないオプションの 符号付き整定数が後に付いた文字 E から成る指数とで構成される値。

**10 進浮動小数点値 (decimal floating-point value)**
10 進小数部と考えることができる仮数形式の実数、 および 10 を底とする整数のべき乗と考えることができる指数の近似値。 「*2* 進浮動小数点値 *(binary floating-point value)*」と対比。

**10 進ピクチャー・データ (decimal picture data)**
「数値ピクチャー・データ *(numeric picture data)*」を参照。

**宣言 (declaration)**
ID を名前として確立し、その ID 用に一連の属性 を (部分的または全体的に) 指定すること。 特定名の属性のソース。

**デフォルト**
指定がされていないときに、とられる値、属性、またはオプション。

**定義された変数 (defined variable)**
指定された基底付き変数用の一部または全部のストレージに関連付けられる変数。

**区切る (delimit)**
1 つ以上の項目またはステートメントの前後を、文字またはキーワードで囲むこと。

**区切り文字**
すべてのコメントと、パーセント記号、括弧、コンマ、ピリオド、セミコロン、コロン、 割り当て記号、ブランク、ポインター、アスタリスク、および単一引用符。 これらは ID、定数、ピクチャー指定、iSUB、およびキーワードの 限界を定めるものとなる。

**記述子 (descriptor)**
区域サイズ、配列境界、またはストリング長 などの変数に関する情報を保持する制御ブロック。

**数字 (digit)**
0 から 9 までの文字の 1 つ。

**次元属性 (dimension attribute)**
配列の次元数を指定し、各次元の境界を示す属性。

**使用不可の (disabled)**
割り込みが発生せず、規定の処置も取られないような事態になった状態。

**DO グループ (do-group)**
DO ステートメントで区切られ、それに対応する END ステートメント で終了する、制御目的に使用される一連のステートメント。 「ブロック *(block)*」と対比。

**DO ループ (do-loop)**
「反復 *DO* グループ *(iterative do-group)*」を参照。

**仮引数 (dummy argument)**
参照でそのままの状態で渡すことのできない引数の値を保持するために、自動的に作成される一時ストレージ。

**ダンプ**
エラーの原因のトレースなどの、プログラムが使用するストレージの一部 または全部、または他のプログラム情報の印刷出力。

**E**

**EBCDIC**
拡張 2 進化 10 進コード (Extended Binary-Coded Decimal Interchange Code)。 8 ビットのコード化文字からなるコード化文字セット。

**編集ディレクティブ伝送 (edit-directed transmission)**
データが連続した文字ストリームとして中にあり、関連データ・ リストに対して行いたい編集を指定するにはフォーマット・リストを必要とするような タイプのストリーム指向伝送。

**エレメント**
配列などのデータ項目の集まりとは対照的な、単一のデータ項目。スカラー項目。

**要素式 (element expression)**
評価されるとエレメント値を生じる式。

**要素変数 (element variable)**
エレメントを表す変数。スカラー変数。

**エレメント名 (elementary name)**
「基本エレメント *(base element)*」を参照。

**使用可能 (enabled)**
条件により割り込みが生じて、該当する規定 ON ユニットが呼び出される条件の状態。

**end-of-step メッセージ (end-of-step message)**
ジョブ制御ステートメントとジョブ・スケジューラー・メッセージのリストに続いており、 各ステップの成功または失敗を示す戻りコードを含むメッセージ。

**入り口定数 (entry constant)**
PROCEDURE ステートメントのラベル接頭部 (入り口名)。 ENTRY 属性を指定し、VARIABLE 属性を指定しないで名前を宣言すること。

**入り口データ (entry data)**
プロシージャーへのエントリー・ポイントを表すデータ項目。

**入り口式 (entry expression)**
評価されると入り口名を生じるような式。

**入り口名 (entry name)**
ENTRY 属性を持つものとして明示的または内容に従って宣言された ID (ただし、VARIABLE 属性が与えられていない場合に限る)。または、ENTRY 属性を暗黙指定された入り口変数の値を持った ID。

**エントリー・ポイント (entry point)**
プロシージャー内の、そのプロシージャーを呼び出すことができるポイント。 *1* 次エントリー・ポイント および *2* 次エントリー・ポイント。

**入り口参照 (entry reference)**
入り口値を返す入り口定数、入り口変数参照、または関数参照。

**入り口変数 (entry variable)**
入り口値を割り当てる対象となりうる変数。 これは、ENTRY 属性と VARIABLE 属性を両方とも持っている必要がある。

**入り口値 (entry value)**
入り口定数または入り口変数によって表されるエントリー・ポイント。 入り口値には、その入り口定数に関連した活動化環境が含まれる。

**環境 (活動化の) (environment (of an activation))**
収容ブロック内で宣言されたデータに関して、呼び出されたブロックと関連し、 そのブロック内で使用される情報。

**環境 (ラベル定数の) (environment (of a label constant))**
ステートメント・ラベル定数への参照が適用されるブロックの個々の 活動化の識別情報。 この情報が決定されるのは、ステートメント・ラベル定数が、引数として 渡されたり、またはステートメント・ラベル変数に割り当てられ、 それが定数と一緒に渡されたり割り当てられたときである。

**確立された処置 (established action)**
条件が生じたときにとられる処置。 「暗黙の処置 *(implicit action)*」および「*ON* ステートメント処置 *(ON-statement action)*」も参照。

**エピローグ (epilogue)**
ブロックまたはタスクの終了時に自動的に生じる各種処理。

**評価 (evaluation)**
単一の値、値の配列、または値の構造化値へ式を換算すること。

**イベント**
状況および完了を、関連したイベント変数から決定することのできる プログラムの活動。

**イベント変数 (event variable)**
イベントと関連付けることができる EVENT 属性を持つ変数。 その値は、処置が完了したかどうか、および完了の状況を示す。

**明示宣言 (explicit declaration)**
ラベル接頭部として DECLARE ステートメント内、または パラメーター・リスト内に ID (名前) を出すこと。 「暗黙宣言 *(implicit declaration)*」と対比。

**指数文字 (exponent characters)**
以下のピクチャー指定文字のこと。

1. K および E。指数フィールドの先頭を示すため、浮動小数点 ピクチャー指定内で使用される文字。
2. F。10 進小数点をその想定位置から 右方向へ (正定数の場合) かまたは左方向へ (負定数の場合) 移動する ときに、小数部の桁数を示す整定数を使って指定される位取り係数文字。

**expression**
値、値の配列、または一連の構造化値セットを表すのに、プログラム内で使われる表記。 単独で使用される定数または参照、あるいは、定数または 参照あるいはその両方を演算子と組み合わせたもの。

**拡張英字 (extended alphabet)**
A から Z までの大文字、小文字の 英字、$、@、および #、または NAMES コンパイラー・オプションで指定されたもの。

**範囲**
配列の次元の境界、ストリング長、または区域サイズによって示される範囲。 この区域がターゲット区域に割り当てられる場合は、ターゲット区域のサイズ。

**外部名 (external name)**
有効範囲が必ずしも 1 つのブロックとその収容ブロックだけに 限定されない (EXTERNAL 属性を持つ) 名前。

**外部プロシージャー**
他のいずれのプロシージャーにも組み込まれないプロシージャー。 パッケージ内に入っていて同様にエクスポートされるレベル 2 の プロシージャー。

**外部シンボル (external symbol)**
それ自身が定義されている制御セクションを除く制御セクション内で参照できる名前。

**外部シンボル辞書 (External Symbol Dictionary (ESD))**
オブジェクト・モジュール内で使われるすべての外部シンボルの一覧表。

**特別言語文字**
英数字にも特殊文字にも分類されない文字 ($、@、および # など)。 このグループには、NAMES コンパイラー・オプションで指定された文字も 含まれる。

**F**

**属性分配 (factoring)**
1 つ以上の属性を、DECLARE ステートメント内の括弧で囲まれた名前リストに対して適用して、 複数の名前に共通する属性を反復する必要をなくすこと。

**フィールド (データ・ストリーム中の) (field (in the data stream))**
単一データまたはスペーシング・フォーマット項目によって、幅 (文字数) を定義 されるデータ・ストリームの部分。

**フィールド (ピクチャー指定の) (field (of a picture specification))**
任意の文字ストリング・ピクチャー指定、または固定小数点数を 記述した数字ピクチャー指定の部分 (または全部)。

**ファイル**
プログラムにおいて、単数または複数のデータ・セットを 名前付きで表現したもの。 ファイルは、オープンするごとに、単数または複数の データ・セットに関連付けられる。

**ファイル定数 (file constant)**
FILE 属性を指定し、VARIABLE 属性を指定しないで宣言された名前。

**ファイル記述属性 (file description attribute)**
各ファイル定数の個々の特性を記述したキーワード。 「代替属性 *(alternative attribute)*」と「追加属性 *(additive attribute)*」も参照。

**ファイル式 (file expression)**
評価されるとファイル・タイプを生じる式。

**ファイル名 (file name)**
ファイル用に宣言された名前。

**ファイル変数 (file variable)**
ファイル定数を割り当てることのできる変数。 この場合、ファイルは、FILE 属性と VARIABLE 属性を持って いなければならず、ファイル記述属性を持っていることはできない。

**固定小数点定数 (fixed-point constant)**
「算術定数 *(arithmetic constant)*」を参照。

**修正 (fix-up)**
コンパイル済みプログラムを実行可能にするために、 コンパイル時にエラーを検出したあとでコンパイラーが実行する解決手段。

**浮動小数点定数 (floating-point constant)**
「算術定数 *(arithmetic constant)*」を参照。

**制御の流れ (flow of control)**
実行の連なり。

**形式**
ストリーム内のデータ項目の表現法を記述したり (データ・フォーマット項目)、またはストリーム内のデータ項目の個々の位置決めを 記述する (制御フォーマット項目) ために編集ディレクティブ・データ伝送内で使用される仕様。

**フォーマット定数 (format constant)**
FORMAT ステートメントでのラベル接頭部。

**フォーマット・データ (format data)**
FORMAT 属性を指定された変数。

**フォーマット・ラベル (format label)**
FORMAT ステートメントでのラベル接頭部。

**フォーマット・リスト (format list)**
ストリーム指向伝送における外部メディアでのデータ項目のフォーマット を指定したリスト。 「データ・リスト *(data list)*」と対比。

**完全修飾名 (fully-qualified name)**
名前が参照するメンバーより上の階層順序内のすべての名前と、 そのメンバー自身の名前が組み込まれている名前。

**関数 (プロシージャー) (function (procedure))**
PROCEDURE ステートメント内に RETURNS オプションのある プロシージャー。 RETURNS 属性を指定して宣言された名前。 これは、関数参照内にその入り口名のうちの 1 つがあると呼び出され、 スカラー値を参照点に返す。 「サブルーチン *(subroutine)*」と対比。

**関数参照 (function reference)**
入り口定数または入り口変数のことで、このどちらも関数を 表さなければならないが、その後に空と考えられる引数リストが 続く。 「サブルーチン呼び出し *(subroutine call)*」と対比。

**g**

**世代 (変数の) (generation (of a variable))**
静的変数の割り振り、被制御変数または自動変数の特定の割り振り、 または基底付き変数の特定のロケーター修飾で、または定義された 変数かパラメーターで指示されるストレージ。

**総称記述子 (generic descriptor)**
GENERIC 属性内で使用する記述子。

**総称キー (generic key)**
キー・クラスを識別する文字ストリング。 そのストリングで始まるキーはすべて、そのクラスのメンバーである。 例えば、 「ABCD」、 「ABCE」、および「ABDF」、という記録済みキーは、すべて総称キー「A」および「AB」で識別されるクラスのメンバーであり、最初の 2 つは、「ABC」というクラスのメンバーでもある。そして、これら 3 つの記録済みキーは、 それぞれ「ABCD」、「ABCE」、「ABDF」というクラスの 固有のメンバーであると見なすことができる。

**総称名 (generic name)**
入り口名ファミリーの名前。 総称名への参照は、呼び出し点にある引数リスト内の引数の属性に一致するパラメーター記述子を持った入り口名によって置き換えられる。

**group**
より大きいプログラム単位に入っているステートメントの 集まり。 グループは、DO グループまたは選択グループのどちらかであるが、 ON ユニットとしての場合を除き、単一ステートメントを使用できるところでは常に使用することができる。

**時間**

**hex**
「*16* 進数字 *(hexadecimal digit)*」を参照。

**16 進数**
16 の基数を持った数体系。有効数は 0 から 9 の数字と、A は 10 を、F は 15 を 表す A から F までの文字。

**16 進数字 (hexadecimal digit)**
0 から 9 および A から F の数字の 1 つ。 A から F は、それぞれ 10 から 15 までの 10 進値を表します。

**I**

**識別子**
コメントや定数内に入ることがなく、前後に区切り文字を伴う文字のストリング。 ID の先頭文字は、26 個の英字、または特別言語文字 (ある場合) でなければならない。 その他の文字がある場合には、拡張英字、数字、区切り文字を追加して入れることができる。

**IEEE**
米国電気電子学会 (Institute of Electrical and Electronics Engineers)。

**暗黙的な (implicit)**
明示指定のないまま取られる処置。

**暗黙処置 (implicit action)**
使用可能な条件が生じたときに、その条件用に現在確立されている ON ユニットがない場合にとられる処置。 「*ON* ステートメント処置 *(ON-statement action)*」と対比。

**暗黙宣言 (implicit declaration)**
DECLARE ステートメント内で明示的に宣言されていないか、 または内容に従って宣言されていない名前。

**暗黙オープン**
OPEN ステートメント以外の入力ステートメントまたは出力ステートメント が原因で、ファイルがオープンされること。

**挿入演算子 (infix operator)**
2 つのオペランド間にある演算子。

**継承次元 (inherited dimension)**
構造体、共用体、またはエレメントでの、収容構造から派生する次元。 名前が配列ではないエレメントであれば、その次元 全体が継承次元で構成される。 名前が配列であるエレメントであれば、その次元は、継承次元および 明示的に宣言された次元で構成される。 1 つ以上の継承次元を持つ構造体を、非結合集合と呼ぶ。 「結合集合 *(connected aggregate)*」と対比。

**入出力 (input/output)**
補助メディアと主記憶装置との間でデータを転送すること。

**挿入点文字 (insertion point character)**
関連データを文字ストリングへ割り当てるときに、指示位置に 挿入されるピクチャー指定文字。 入力のときに P フォーマット項目内で使用される挿入文字は、検査の目的で用いられる。

**整数**
符号を付けるか付けないかは任意の、10 進または 2 進小数点のない一連の数字、または 一連のビット。 通常は、FIXED BINARY (p,0) または FIXED DECIMAL (p,0) と記述される、 符号を付けるか付けないかは任意の整数。

**規定境界 (integral boundary)**
そこでデータを位置合わせすることができる任意の 8 ビット単位のバイト・マルチアドレス。 通常はハーフワード、フルワード、 またはダブルワード (2、4、または 8 バイトの長さの整倍数) 境界である。

**介在配列 (interleaved array)**
非連結ストレージを参照する配列。

**介在添え字 (interleaved subscripts)**
添え字付き修飾参照の最下位レベル以外のレベルに存在する添え字。

**内部ブロック (internal block)**
ブロックの中に組み込まれている別のブロック。

**内部名 (internal name)**
名前が宣言されたブロック内のみで認識されている名前、また そのブロック内に入っているブロックの中でも認識されている可能性もある名前。

**内部プロシージャー (internal procedure)**
ブロックの中に組み込まれている別のプロシージャー。 「外部プロシージャー *(external procedure)*」と対比。

**interrupt**
条件やアテンションの発生の結果として、プログラムの制御の流れを変更すること。

**起動 (invocation)**
プロシージャーの活動化。

**起動する (invoke)**
プロシージャーを活動化すること。

**呼び出されたプロシージャー (invoked procedure)**
活動化されているプロシージャー。

**呼び出し側ブロック (invoking block)**
プロシージャーを活動化するブロック。

**反復因数 (iteration factor)**
INITIAL 属性指定において、特定の値を使って初期化されることになっている 配列の連続エレメント数を指定するための式。 フォーマット・リストにおける、特定のフォーマット項目またはフォーマット項目のリストを 連続して使用する回数を指定するための式。

**反復 DO グループ (iterative do-group)**
制御変数または WHILE や UNTIL オプション、またはこの両方 を指定した DO ステートメントを持つ DO グループ。

**K**

**鍵**
直接アクセス・データ・セット内のレコードを識別するデータ。 「ソース・キー *(source key)*」および「記録済みキー *(recorded key)*」を参照。

**キーワード**
PL/I において定義されたコンテキスト内で使用されると特定の意味を持つ ID。

**キーワード・ステートメント (keyword statement)**
ステートメントの機能を示すキーワードで始まる単純ステートメント。

**認識された (名前に関する用語) (known (applied to a name))**
宣言された意味で認識されること。 名前は、その有効範囲内で認識される。

**L**

**label**
ステートメントの接頭部として付く名前。 PROCEDURE ステートメント上の名前を、入り口定数と呼び、FORMAT ステートメント上の名前を、 フォーマット定数と呼ぶ。その他の種類のステートメント上の名前を、ラベル定数と呼ぶ。 LABEL 属性を持つデータ項目。

**ラベル定数 (label constant)**
ステートメント (PROCEDURE、ENTRY、FORMAT、または PACKAGE を除く) のラベル接頭語として書き込まれる名前。実行時に、そのラベル接頭語が 参照されれば、そのステートメントにプログラムの制御を渡す ことができる。

**ラベル・データ**
ラベル定数または、ラベル変数の値。

**ラベル接頭部 (label prefix)**
ステートメントの接頭部として付けられたラベル。

**ラベル変数 (label variable)**
LABEL 属性を指定して宣言された変数。 その値は、プログラム内でのラベル定数である。

**先行ゼロ (leading zeroes)**
算術値としては意味のないゼロ。 ある数値内で最初の非ゼロより左側にあるすべてのゼロ。

**レベル番号 (level number)**
DECLARE ステートメント中の名前の前に付く番号で、 構造体名の階層内のその相対位置を指定するもの。

**レベル 1 変数 (level-one variable)**
大構造体または共用体の名前。 構造体または共用体の中に含まれていない、添え字なし変数。

**字句単位の (lexically)**
単位を左から右への順序に扱うことに関連した用語。

**ライブラリー**
メンバーと呼ばれるその他のデータ・セットを保管するのに使用できる MVS 区分データ・セット 、または CMS MACLIB のこと。

**リスト・ディレクティブ**
ストリーム内のデータがブランクやコンマで区切られた定数になり、 フォーマット設定が自動的に行われるタイプのストリーム指向伝送。

**ロケーター**
変数のアドレスまたはその記述子を保持する制御ブロック。

**ロケーター/記述子 (locator/descriptor)**
その後に記述子の付いたロケーター。 ロケーターは、記述子のアドレスではなく、変数のアドレスを保持する。

**ロケーター修飾**
基底付き変数への参照において、その参照が参照している基底付き変数 の世代を指定するために、基底付き変数の左側に矢印で接続されているロケーター変数 または関数参照。 これは、暗黙参照であることもある。

**ロケーター値 (locator value)**
ストレージ・アドレスを識別する値か、またはストレージ・アドレスを識別するのに使用できる値。

**ロケーター変数 (locator variable)**
変数またはバッファーの主記憶域内の位置を識別する値を 持った変数。 これは、POINTER 属性または OFFSET 属性を 持つ。

**ロック・レコード (locked record)**
EXCLUSIVE DIRECT UPDATE ファイル内のレコードであって、1 つのタスクにだけ しか使用することはできず、そのレコードを使用しているタスクによって 解放されるまで、他のタスクからアクセスできないレコード。

**論理レベル (構造体または共用体メンバーの) (logical level (of a structure or union member))**
全レベル番号が直接順序になっているとき (あるレベル番号から次の レベル番号までの増分が 1 のとき) に、レベル番号で示される深さ。

**論理演算子 (logical operators)**
ビット・ストリング演算子 NOT と排他 OR (¬)、AND (&)、および OR (|)。

**ループ (loop)**
繰り返し実行される一連の命令。

**下限 (lower bound)**
配列次元の下限。

**m**

**主プロシージャー (main procedure)**
OPTIONS (MAIN) 属性を持った PROCEDURE ステートメントのある 外部プロシージャー。 このプロシージャーは、プログラム実行の 最初のステップで自動的に呼び出される。

**大構造 (major structure)**
レベル番号 1 を指定して宣言された名前を持つ構造。

**member**
構造体または共用体の中の、構造体、共用体、あるいはエレメントの名前。 ライブラリー内のデータ・セット。

**小構造 (minor structure)**
別の構造体または共用体の中に組み込まれている構造体。 小構造の名前は、1 よりも大きくかつ親構造体または 親共用体よりも大きいレベル番号を指定して宣言される。

**モード (算術データの) (mode (of arithmetic data))**
算術データの属性。 これは、実数 または複素数 のどちらかである。

**多重宣言 (multiple declaration)**
同一ブロックに対して内部であり、別の修飾を持たない同一 ID の複数宣言。 同一 ID の複数外部宣言。

**マルチプロセッシング (multiprocessing)**
複数のプログラムを同時に実行するために、複数の処理装置 を備えた計算機システムを使用すること。

**マルチプログラミング (multiprogramming)**
単一の処理装置を使って、複数のプログラムを並行して 処理するのに、計算機システムを使用すること。

**N**

**名前**
変数や定数にユーザーが与える ID。 コンテキスト中に現れる、キーワードではない ID。 場合によっては、ユーザー定義名とも呼ぶ。

**ネスト (nesting)**
次のものの発生。

- ブロック内にある別のブロック。
- グループ内にある別のグループ。
- THEN 文節または ELSE 文節内の IF ステートメント。
- 関数参照の引数としての関数参照。
- FORMAT ステートメントのフォーマット・リスト内のリモート・フォーマット項目。
- パラメーター記述子リスト内の別のパラメーター記述子リスト。
- 1 つ以上の属性が分配されている括弧で囲まれた名前リスト内の属性の指定。

**非結合ストレージ (nonconnected storage)**
非結合データ項目が占有するストレージ。 例えば、継承次元を持つ介在配列や構造体は、非結合ストレージ 内にある。

**ヌル・ロケーター値 (null locator value)**
内部記憶域内のどの位置も識別できない特殊ロケーター値。 これは、現在ロケーター変数がデータの世代を識別できないことを示すのに役立つ。

**ヌル・ステートメント (null statement)**
セミコロン記号 (;) のみが含まれているステートメント。 これは、アクションが実行されないことを示します。

**ヌル・ストリング (null string)**
長さゼロの文字ストリング、漢字ストリング、またはビット・ストリング。

**数字データ (numeric-character data)**
「*10* 進ピクチャー・データ *(decimal picture data)*」を参照。

**数値ピクチャー・データ (numeric picture data)**
算術値と文字値を持ったピクチャー・データ。 このタイプのピクチャー・データは、'A' または 'X' という文字を含むことはできない。

**o**

**オブジェクト**
単一名で参照されるデータの集まり。

**オフセット変数 (offset variable)**
OFFSET 属性を持ったロケーター変数のことであり、その値は、ストレージ内のある区域の先頭からの相対位置を識別する。

**オン条件 (ON-condition)**
PL/I プログラムにおける、プログラム割り込みの原因となりうる オカレンス。 予期しないエラーが検出されたり、予期できる出来事ではあるものの、 予期しない時にそれが起きたときに発生する。

**ON ステートメント処置 (ON-statement action)**
ある条件が生じたときに処置を取れるよう、条件に対して 明示的に設定された処置法。 プログラムの制御の流れ内で ON ステートメントが見つかると、とられる処置で、その条件に対する処置が設定される。 この処置は、ON ユニットが設定されたままであるか、RESIGNAL ステートメントで再設定されて条件が生じたときにとられる。 「暗黙の処置 *(implicit action)*」と対比。

**ON ユニット (ON-unit)**
該当する条件が起きたときに、とられるよう指定された処置。

**オープン (ファイルの) (opening (of a file))**
ファイルをデータ・セットに関連付けること。

**オペランド**
ID、定数、または式。式には演算子が、時には他のオペランドと ともに使用される。

**演算式 (operational expression)**
1 つ以上の演算子から成る式。

**演算子**
実行する演算を指定する記号。

**オプション**
ステートメントの実行や解釈に影響を及ぼすのに使われるステートメント中 の指定。

**P**

**パッケージ定数 (package constant)**
PACKAGE ステートメントのラベル接頭語。

**パック 10 進 (packed decimal)**
固定小数点 10 進データ項目の内部表現。

**埋め込み**
ストリングの長さを必要な長さまで拡張するために、ストリングの右側に 連結される、1 つ以上の文字、漢字、またはビット。 構造体または共用体の中に挿入される、1 つ以上のバイトまたはビット。 その構造、または共用体内の後続エレメントが正しい規定境界に位置合わせされるようにするためのもの。

**パラメーター**
PROCEDURE ステートメントの後に続くパラメーター・リスト中の名前。 そのプロシージャーが呼び出されれば、渡される引数を指定する。

**パラメーター記述子 (parameter descriptor)**
ENTRY 属性指定内でパラメーター用に指定される一連の属性。

**パラメーター記述子リスト (parameter descriptor list)**
ENTRY 属性指定内のすべてのパラメーター記述子のリスト。

**パラメーター・リスト (parameter list)**
コンマで区切られ、プロシージャー・ステートメント内の キーワード PROCEDURE の後に続くか、または ENTRY ステートメント内の キーワード ENTRY の後に続く、括弧で囲まれた 1 つ以上のパラメーターのリスト。 このリストは、呼び出し時に渡される引数リストと対応する。

**部分修飾名 (partially-qualified name)**
不完全な修飾名。 これには名前が参照する構造メンバーまたは共用体メンバーより上の階層順序内にある 名前のうちの全部ではない 1 つ以上の名前、およびそれ自身のメンバー名が 含まれる。

**ピクチャー・データ (picture data)**
文字形式で表された、数値データ、文字データ、またはそれらの混合。

**ピクチャー指定 (picture specification)**
PICTURE 属性を指定した宣言内で、または P フォーマット項目内でピクチャー文字を 使用して宣言されたデータ項目。

**ピクチャー指定文字 (picture specification character)**
ピクチャー指定で使用できるすべての文字。

**PL/I 文字セット (PL/I character set)**
PL/I のプログラム・エレメントを表現するために定義されている文字セット。

**PL/I プロンプター (PL/I prompter)**
PL/I コマンドのコマンド・プロセッサー・プログラムで、オペランドを調べ、コンパイラーに必要なデータ・セットを割り振る。

**呼び出し点 (point of invocation)**
呼び込まれたプロシージャーへの参照が現れる呼び込み側 ブロック内の地点。

**ポインター**
ストレージ内の位置を識別するための変数のタイプ。

**ポインター値 (pointer value)**
ポインター型を識別する値。

**ポインター変数 (pointer variable)**
ポインター値が入った POINTER 属性を持ったロケーター変数。

**精度 (precision)**
固定小数点データ項目内にある桁数またはビット数、または、 浮動小数点データ項目での最小確保有効数字 (指数は除く) の数。

**接頭部 (prefix)**
ステートメントの先頭に付けられるラベル、または 1 つ以上 の条件名の括弧で囲まれたリスト。

**接頭演算子 (prefix operator)**
オペランドの前に置かれ、そのオペランドにだけ適用される演算子。 接頭演算子には、プラス (+)、マイナス (-)、および not (¬) がある。

**プリプロセッサー**
コンパイルを実行する前に、ソース・プログラムを調べるためのプログラム。

**プリプロセッサー・ステートメント (preprocessor statement)**
プリプロセッサーがとる 処置を指定するために、ソース・プログラム内に入れる特殊ステートメント。これは、プリプロセッサーによって検出されると実行 される。

**1 次エントリー・ポイント (primary entry point)**
PROCEDURE ステートメントのラベル・リスト内の任意の名前によって識別される エントリー・ポイント。

**priority**
タスクに関連する値で、他のタスクに対するそのタスクの優先順位 (指名順 位) を指定する。

**問題データ (problem data)**
コード化算術データ、ビット・データ、文字データ、グラフィック・データ、およびピクチャー・データ。

**問題状態プログラム (problem-state program)**
オペレーティング・システムの問題プログラム状態内で 稼働するプログラム。 これには、入出力指示やその他の特権命令は入らない。

**プロシージャー**
PROCEDURE ステートメントと END ステートメントで 区切られたステートメントの集まり。 プロシージャーとはプログラムまたはプログラムの一部であり、名前の有効範囲を区切り、 そのプロシージャーまたは入り口名の 1 つへの参照によって活動化される。 「外部プロシージャー *(external procedure)*」および「内部プロシージャー *(internal procedure)*」も参照。

**プロシージャー参照 (procedure reference)**
入り口定数または入り口変数。 この後に引数リストを続けることができる。 プロシージャー参照は、CALL ステートメントや CALL オプションに入れることも、または 関数参照として使用することもできる。

**プログラム**
1 つ以上の外部プロシージャーまたはパッケージのセット。 外部プロシージャーのうちの 1 つは、PROCEDURE ステートメント内に OPTIONS(MAIN) 指定を 持っていなければならない。

**プログラム制御データ (program control data)**
PL/I プログラムの処理を制御するのに使用するための 区域、ロケーター、ラベル、フォーマット、項目、およびファイルのデータ。

**プロローグ (prologue)**
ブロックの起動時に自動的に生じる処理。

**疑似変数 (pseudovariable)**
ターゲット変数を指定するのに使用できるすべての組み込み関数の名前。 これは通常、代入ステートメントの左側にある。

## q

**修飾名 (qualified name)**
構造メンバーまたは共用体メンバーの階層順序。ピリオドで結合されていて、 構造体の中の名前を識別するのに使用される。 どの名前にも添え字を付けることができる。

## R

**範囲 (デフォルト指定の) (range (of a default specification))**
DEFAULT ステートメント内の属性を適用される ID またはパラメーター記述子のどちらか、またはこの両方のセット。

**record**
レコード単位入力または出力の操作における、伝送の 論理単位。 1 つ以上の関連データ項目の集まり。これらの項目には通常、それぞれ異なったデータ属性があり、 また通常は構造体か共用体の宣言で記述される。

**記録済みキー (recorded key)**
直接アクセス・データ・セット内でレコードを識別する 文字ストリングのことであり、そこでは文字ストリングそのものも データの一部として記録される。

**レコード単位データ伝送 (record-oriented data transmission)**
別々のレコードの形式でデータを伝送すること。 「ストリーム指向のデータ伝送 *(stream data transmission)*」と対比。

**再帰的プロシージャー (recursive procedure)**
そのプロシージャー自身からでも、または別のアクティブ・プロシージャー からでも呼び出すことのできるプロシージャー。

**再入可能プロシージャー (reentrant procedure)**
複数のタスク、スレッド、またはプロセスから同時に活動化でき、 しかもこれらのタスク、スレッド、およびプロセス間で相互に干渉が 生じないプロシージャー。

**REFER 式 (REFER expression)**
REFER というキーワードの前に付いた式。この式は、REFER オプションを含む基底付き変数が、ALLOCATE ステートメントまたは LOCATE ステートメントのいずれかによって割り振られるときの境界、長さ、または サイズとして使用される。

**REFER オブジェクト (REFER object)**
REFER オプション中の変数。メンバーの現行境界、長さ、またはサイズを保持しているか、 あるいは保持する予定のもの。 REFER オブジェクトは、同一構造体または共用体のメンバーでなければならない。 これは、ロケーター修飾したり添え字を付けてはならず、また REFER オプションを持ったメンバーの前になければならない。

**参照**
明示宣言を生じることになる 1 つのコンテキスト内以外の名前の出現。

**相対仮想起点 (relative virtual origin (RVO))**
配列の実際の原点から配列の仮想原点を引いたもの。

**リモート・フォーマット項目 (remote format item)**
R という文字の後に FORMAT ステートメントのラベル (括弧に囲まれている) のあるもの。 フォーマット指定ステートメントは、転送するデータのフォーマットを制御するために、 編集ディレクティブ・データ伝送ステートメントにより使用する。

**反復因数 (repetition factor)**
以下のものを指定する、括弧に入れられた符号なし整定数。

1. 後続するストリング定数を繰り返す回数。
2. 後続するピクチャー文字を繰り返す回数。

**反復指定 (repetitive specification)**
1 つ以上のデータ項目伝送の被制御反復を指定するための データ・リストの 1 エレメントであり、通常は配列と同時に使用する。

**制限付き式 (restricted expression)**
コンパイル時にコンパイラーによって評価されて定数を生じる式。 このような式のオペランドは、定数、指定した定数、および 制限付きの式になる。

**戻り値 (returned value)**
関数プロシージャーから返される値。

**RETURNS 記述子 (RETURNS descriptor)**
RETURNS 属性内と、PROCEDURE および ENTRY ステートメントの RETURNS オプション内 で使用する記述子。

## S

**スカラー変数 (scalar variable)**
構造、共用体、配列ではない変数。

**スケール (scale)**
1 つの数値表記体系であり、その算術値は固定小数点 または浮動小数点で表現される。

**位取り係数 (scale factor)**
固定小数点数内の小数桁数の指定。

**位取り係数 (scaling factor)**
「位取り係数 *(scale factor)*」を参照。

**有効範囲 (条件接頭語の) (scope (of a condition prefix))**
全体にわたって特定の条件接頭語が適用される、プログラムの部分。

**有効範囲 (宣言の) (scope (of a declaration or name))**
全体にわたって特定名が認識されているプログラムの部分。

**2 次エントリー・ポイント (secondary entry point)**
入り口ステートメントのラベル・リスト内の任意の名前によって識別されるエントリー・ポイント。

**選択グループ (select-group)**
SELECT ステートメントと END ステートメントで区切られたステートメントの連なり。

**選択文節 (selection clause)**
選択グループの WHEN 文節または OTHERWISE 文節。

**自己定義データ (self-defining data)**
プログラム実行時に決定され、集合のメンバー内に保管される 境界、長さ、およびサイズを持つデータ項目を含む集合。

**分離文字 (separator)**
「区切り文字 *(delimiter)*」を参照。

**シフト (shift)**
ストレージ内のデータを元の位置の左または右へ変更すること。

**シフトイン (shift-in)**
2 バイト・ストリングの終わりをコンパイラーに知らせるために 使用される記号。

**シフトアウト (shift-out)**
2 バイト・ストリングの先頭でコンパイラーにシグナルを送るために 使用される記号。

**符号および通貨記号 (sign and currency symbol characters)**
ピクチャー指定文字。 S、+、-、および $ (または < と > で囲まれたその他の通貨記号)。

**単純パラメーター (simple parameter)**
ストレージ・クラス属性が指定されていないパラメーター。 単純パラメーターはどのストレージ・クラスの引数も表すことができるが、 被制御引数の現行世代だけを表すことができる。

**単純ステートメント (simple statement)**
IF、ON、 WHEN、および OTHERWISE 以外のステートメント。

**source**
問題データに変換されるデータ項目。

**ソース・キー (source key)**
直接アクセス・データ・セット内で個々のレコードを 識別するため、レコード単位伝送ステートメント内で参照されるキー。

**ソース・プログラム**
ソース・プログラム・プロセッサー、およびコンパイラーへの入力となる プログラム。

**ソース変数 (source variable)**
他の演算に使用されるが、その演算で変更されることのない変数。「ターゲット変数 *(target variable)*」と対比。

**予備ファイル (spill file)**
一時作業ファイルとして使われる SYSUT1 という名前のデータ・セット。

**標準デフォルト (値) (standard default)**
属性またはオプションの指定がなく、適用できる DEFAULT ステートメント がない場合の、代替属性または代替オプション。

**標準ファイル (standard file)**
GET ステートメントや PUT ステートメントで FILE オプションまたは STRING オプションが ない場合に、PL/I が想定するファイル。 SYSIN が標準入力ファイルであり、SYSPRINT が標準出力ファイルである。

**標準システム処置 (standard system action)**
使用可能な条件のための ON ユニットがないときにその条件が発生した場合にとられる、言語で指定された処置。

**ステートメント**
キーワード、区切り文字、ID、演算子、および定数で構成され、セミコロン (;) で終了する PL/I ステートメント。 オプションで、条件接頭語リストとラベルのリストを持つことができます。「キーワード・ステートメント *(keyword statement)*」、「代入ステートメント *(assignment statement)*」、および「ヌル・ステートメント *(null statement)*」も参照。

**ステートメント本体 (statement body)**
ステートメント本体は、単純ステートメントまたは 複合ステートメントのどちらでもかまわない。

**ステートメント・ラベル (statement label)**
「ラベル定数 *(label constant)*」を参照。

**静的ストレージ割り振り (static storage allocation)**
静的変数用のストレージの割り振り。

**静的変数 (static variable)**
プログラム実行の開始前に割り振られ、その実行の継続時間中は その割り振りの変わらない変数。

**ストリーム指向データ伝送**
文字形式になった個々のデータ値の連続ストリームであるものとしてデータを 扱って、データを伝送すること。「レコード単位データ伝送 *(record-oriented data transmission)*」と対比。

**ストリング**
単一のデータ項目として処理される、連続した文字、グラフィックス、 またはビットの列。

**ストリング変数 (string variable)**
BIT、 CHARACTER、または GRAPHIC 属性を指定して宣言される変数。 この変数の値は、ビット・ストリング、文字ストリング、または漢字ストリング のいずれでもかまわない。

**構造**
必ずしも同じ属性を持たなくても差し支えないデータ項目の集まり。「配列 *(array)*」と対比。

**構造式 (structure expression)**
評価されると構造体の値セットを生成する式。

**配列の構造体 (structure of arrays)**
次元属性を持つ構造体。

**構造体メンバー (structure member)**
「メンバー *(member)*」を参照。

**構造化 (structuring)**
メンバー数、配置順、属性、および論理レベルによって表現される構造階層。

**サブルーチン**
PROCEDURE ステートメント内に RETURNS オプションのないプロシージャー。「関数 *(function)*」と対比。

**サブルーチン呼び出し (subroutine call)**
後に CALL ステートメント内にあるオプションの引数リストが付く、サブルーチンを表さなければならないエントリー参照。「関数参照 *(function reference)*」と対比。

**添え字**
配列の次元内の位置を指定するための要素式。 添え字がアスタリスクであれば、次元のすべてのエレメントを 指定する。

**添え字リスト (subscript list)**
括弧に入れられた、1 つ以上の添え字のリスト。配列の各々の 次元に対して 1 つの添え字が対応する。これらによって配列の単一エレメントまたはクロスセクションを一意的に識別する。

**サブタスク (subtask)**
特定のタスクによって生成されるタスク、または特定のタスクから最後に 生成されたタスクへの直接ライン内の任意のタスク。

**同期**
プログラムの順次実行での単一の制御の流れ。

**T**

**ターゲット**
データ項目 (ソース) が変換される属性。

**ターゲット参照 (target reference)**
受取側変数 (または受取側変数の一部) を指定する参照。

**ターゲット変数 (target variable)**
値が割り当てられる変数。

**タスク**
単一の制御の流れによる 1 つ以上のプロシージャーの実行。

**タスク名 (task name)**
タスク変数を参照するのに使用される ID。

**タスク変数 (task variable)**
TASK 属性を持ち、その値がタスクの相対優先順位を示す変数。

**終了 (ブロックの) (termination (of a block))**
ブロックの実行が終了して、RETURN ステートメントまたは END ステートメントによって、 制御がその起動側ブロックに戻るか、または GO TO ステートメントによって起動側ブロックまたは 他のアクティブ・ブロックに制御が渡ること。

**終了 (タスクの) (termination (of a task))**
タスクへの制御の流れを停止すること。

**切り捨て (truncation)**
ターゲット変数のストリング長や精度が限度を超えたときに、 データ項目の片方の端から 1 つ以上の数字文字、グラフィックス、 またはビットを除去すること。

**タイプ**
データの世代、値、または項目に対して適用される一連の データ属性とストレージ属性。

**U**

**未定義 (undefined)**
ユーザーが行ってはならないことを示す。 未定義機能が使用されると、PL/I 製品の個々のインプリメンテーションによって違った 結果が出る可能性がある。 このような場合、アプリケーション・プログラムはエラーとなる。

**共用体 (union)**
同一のストレージを占有し、相互にオーバーレイしたデータ・エレメントの集まり。 メンバーは構造体、共用体、基本変数、または配列のいずれであっても構わない。 それらは、同一の属性を持っていなくても構わない。

**配列の共用体 (union of arrays)**
次元属性を持った共用体。

**上限 (upper bound)**
配列次元の上限。

**V**

**値参照 (value reference)**
データ項目の値を得るのに使用する参照。

**variable**
データを参照するのに使用され、値を割り当てる対象となりうる 名前付きのエンティティー。 その属性は一定のままであるが、場合に応じてそれぞれ 異なる値を参照することができる。

**変数参照 (variable reference)**
変数全体またはその一部を指定する参照。

**仮想起点 (virtual origin (VO))**
すべてゼロの添え字を持った配列のエレメントを保持するための位置。 このようなエレメントが配列内になければ、仮想起点は本来それが保持されるべき 場所になる。

**z**

**ゼロ抑止文字 (zero-suppression characters)**
ピクチャー指定文字の Z と *。これは、対応する桁位置の ゼロを抑止し、それぞれをブランクまたはアスタリスクで置き換えるのに使用する。

# 索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

## [ア行]


アスタリスク
  算術演算での使用 56
  説明 328
  添え字としての使用 174
  ID として 6
アプリケーション 87
アレア
  変数の伝送 288
暗黙処置 (implicit action) 339
暗黙的な (implicit)
  解放
    基底付き変数の場合 249
    被制御変数の場合 240
  宣言 (declaration) 149
  ファイルのオープン 282
暗黙にロケーター修飾された参照 246
暗黙の日付
  代入 41
  比較 41
異常終了
  プロシージャー 98
  program 88
位置指定モード 293
移動モード 293
入り口参照の呼び出し 121
入り口参照呼び出し 121
入り口定数 111
入り口データ (entry data)
  参照の呼び出し 121
  説明 110
  総称入り口宣言 119
  属性
    分類 20, 21
    ENTRY 112
    GENERIC 119
    LIMITED 118
    LIST 116
    OPTIONAL 114
  直接入り口宣言 110
  定数 111
  変数 111
  GENERIC 119
引用符 (単一または二重)、ストリング・データを囲む 16
英字 1
英数字 2
エクステント (次元の) 171
エレメント
  スカラー 15
  代入 (assignment) 201
  パラメーター 108
  data 15
  DBCS の場合 12
  expression 52
エレメント (続き)
  program 1
  SBCS の場合 5
  variable 15
演算
  組み合わせ 68
  クラス 54
  算術 56
  接頭部 (prefix)
    説明 54
    例 70
  挿入演算子 54
  ハンドル 55
  比較
    説明 64
    例 66
  ポインター 55
  連結 (concatenation) 66
  論理 63
  bit 63
演算子
  算術
    使用 7
    説明 56
  使用 6
  ストリング 7
  挿入演算子
    ディスカッション 70
    ポインター式との併用 55
  比較 7
  論理 7
  bit 7
演算子の優先順位 68
演算式
  クラス 54
  説明 54
  定義 51
  データ・タイプの制限 54
  変換規則 54
  例 55
エントリー・ポイント 92
応答テキスト 588
オーバーパンチ・ピクチャー文字、I 334
オーバーパンチ・ピクチャー文字、R 334
オーバーパンチ・ピクチャー文字、T 334
大文字小文字の区別 5
オフセット・データ 252
オフセット変数 (offset variable) 244
オペランド
  定義 51
  変換 59


## [カ行]


介在添え字 (interleaved subscripts) 184
開始ブロック
  活動化 110

開始ブロック (続き)
終了 110
説明 109
例 109
外部プロシージャー
説明 92
動的ロード 100
各種演算の組み合わせ 68
確立された条件 339
確立された処置 (established action) 341
貸方 (CR) ピクチャー文字 334
下線 (_) 文字 16
仮想定数 24
括弧 6
活動化
開始ブロック (begin-block) 110
プロシージャー 97
ブロック 89
program 88
仮引数
説明 108
属性の導出 108
ルール 108
借方 (DB) ピクチャー文字 334
漢字ストリング定数 36
関数
組み込み 106
制限 104
説明 104
定義 104
プログラマー作成の 106
戻り 122
例 105
キーワード
定義 5
キーワード・ステートメント (keyword statement) 10
期間 6
記号、複合の 4
疑似変数
説明 53
宣言 373
要約 390, 391
呼び出し 373
ENTRYADDR 429
IMAG 449
ONCHAR 492
ONGSOURCE 494
ONSOURCE 497
ONUCHAR 499
ONUSOURCE 499
ONWCHAR 500
ONWSOURCE 500
REAL 522
STRING 541
SUBSTR 542
SUBTO 543
TYPE 548
UNSPEC 553
記述子のリスト 113
規定境界 (integral boundary) 158
基底付きストレージ
組み込み関数 243
説明 235
BASED 属性の構文 242
基底付き変数
組み込み関数 243
説明 242, 247
リストの入出力 256
ALLOCATE ステートメント 248
FREE ステートメント 249
基本名 174
境界 (bounds)
単純パラメーター (simple parameter) 95
被制御パラメーター (controlled parameter) 95
共用体
修飾名 176
説明 175
宣言 (declaration) 175
名前 175
配列のクロスセクション 184
例 176
レベル 176
UNION 属性、分類 22
共用体のレベル 176
区域
属性 251
代入 (assignment) 255
data 251
区切り文字 6
句読点のついた定数 16
組み合わせ配列、構造、および共用体 183
組み込み関数
基底付き変数 247
区域変数 255
固定
データ変換 74
最大 472
算術、要約 375
条件処理、要約 379
序数 144
序数処理データ、要約 389
数学、要約 386
ストリング処理、要約 393
ストレージ制御、要約 391
整数演算、要約 385
精度処理、要約 389, 390
宣言 373
追加 398
定義 106
データ変換の開始 74
での数学関数の正確度 375
入出力、要約 384
ヌル引数および 375
のカテゴリー 375
バイト 412
配列処理、要約 376
引数の集合 374
被制御変数
データ変換 74
日付/時刻、要約 380
ビット
データ変換 74
浮動小数点演算、要約 384
浮動小数点の照会、要約 383
プリプロセッサー 589
プリプロセッサー用 589
呼び出し 373
ABS 397

組み込み関数 (続き)
ACOS 398
ADDR 398
ADDRDATA 399
ALL 399
ALLCOMPARE 399
ALLOC31 400
ALLOCATE (ALLOC) 400
ALLOCATION (ALLOCN) 401
ALLOCNEXT 401
ALLOCSIZE 401
ANY 402
ASIN 402
ATAN 402
ATAND 402
ATANH 403
AUTOMATIC (AUTO) 403
AVAILABLEAREA 403
BASE64DECODE 404
BASE64DECODE16 405
BASE64DECODE8 404
BASE64ENCODE 405
BASE64ENCODE16 407
BASE64ENCODE8 406
BETWEEN 408
BETWEENEXCLUSIVE 408
BETWEENLEFTEXCLUSIVE 409
BETWEENRIGHTEXCLUSIVE 409
BINARY
データ変換 74
BINARY (BIN) 409
BINARYVALUE
序数での使用 144
ポインター式との併用 56
BINARYVALUE (BINVALUE) 410
BINSEARCH (BIN) 410
BINSEARCHX (BIN) 411
BITLOCATION (BITLOC) 412
BOOL 412
BYTELENGTH 413
CDS 413
CEIL 414
CENTERLEFT (CENTER) 414
CENTERRIGHT 415
CHAR 74
CHARACTER (CHAR) 415
CHARGRAPHIC (CHARG) 416
CHARVAL 417
CHECKSTG 417
CHECKSUM 418
CODEPAGE 418
COLLAPSE 418
COLLATE 418
COMPARE 419
COMPLEX (CPLX) 419
CONJG 420
COPY 420
COS 421
COSD 421
COSH 421
COUNT 421
CS 421
CURRENTSIZE (CSTG) 423
CURRENTSTORAGE 423
組み込み関数 (続き)
DATAFIELD 424
DATE 424
DATETIME 424
DAYS 424
DAYSTODATE 425
DAYSTOMICROSECS 426
DAYSTOSECS 426
DECIMAL
データ変換 74
DECIMAL (DEC) 426
DIMENSION (DIM) 427
DIVIDE 427
EDIT 427
EMPTY 428
ENDFILE 428
ENTRYADDR 429
EPSILON 429
ERF 429
ERFC 429
EXP 430
EXPONENT 430
FILEDDINT 430
FILEDDTEST 431
FILEDDWORD 431
FILEID 432
FILENEW 432
FILEOPEN 433
FILEREAD 433
FILESEEK 433
FILETELL 434
FILEWRITE 434
FIXEDBIN 435
FIXEDDEC 435
FLOAT
データ変換 74
FLOATBIN 436
FLOATDEC 437
FLOOR 437
FOLDEDFULLMATCH 438
FOLDEDSIMPLEMATCH 438
GAMMA 438
GETENV 439
GETJCLSYMBOL 439
GETSYSINT 439
GETSYSWORD 440
GRAPHIC
データ変換 74
GTCA 441
HANDLE 442
HBOUND 442
HBOUNDACROSS 442
HEX 443, 550
HEX8 444
HEXDECODE 445
HEXDECODE8 445
HEXIMAGE 446
HEXIMAGE8 446
HIGH 447
HUGE 447
IAND 447
ICLZ 447
IEOR 448
IFTHENELSE 448

組み込み関数 (続き)
IMAG
データ変換 74
INARRAY 449
INDEX 450
INDEXR 450
INDICATORS 451
INLIST 451
INOT 451
IOR 452
ISFINITE 452
ISIGNED 452
ISINF 453
ISJCLSYMBOL 453
ISLL 453
ISMAIN 454
ISNAN 454
ISNORMAL 454
ISRL 454
ISZERO 455
IUNSIGNED 455
JSON、要約 385
JSONGETARRAYEND 455
JSONGETARRAYSTART 456
JSONGETCOLON 456
JSONGETCOMMA 456
JSONGETMEMBER 457
JSONGETOBJECTEND 458
JSONGETOBJECTSTART 458
JSONGETVALUE 459
JSONPUTARRAYEND 461
JSONPUTARRAYSTART 461
JSONPUTCOLON 461
JSONPUTCOMMA 461
JSONPUTMEMBER 462
JSONPUTOBJECTEND 463
JSONPUTOBJECTSTART 463
JSONPUTVALUE 464
JSONVALID 465
JULIANTOSMF 465
LBOUND 465
LBOUNDACROSS 465
LEFT 466
LENGTH 466
LINENO 466
LOCATION (LOC) 467
LOCSTG 469
LOCVAL 469
LOG 470
LOG10 470
LOG2 470
LOGGAMMA 470
LOW 471
LOWER2 472
LOWERASCII 471
LOWERCASE 471
LOWERLATIN1 472
MAXDATE 473
MAXEXP 473
MAXLENGTH 475
MAXVAL 474
MEMCOLLAPSE 476
MEMCONVERT 475
MEMCU12 476
組み込み関数 (続き)
MEMCU14 477
MEMCU21 477
MEMCU24 478
MEMCU41 478
MEMCU42 478
MEMINDEX 479
MEMREPLACE 480
MEMSEARCH 481
MEMSEARCHR 482
MEMSQUEEZE 482
MEMVERIFY 483
MEMVERIFYR 484
MICROSECS 484
MICROSECSTODATE 485
MICROSECSTODAYS 485
MIN 486
MINDATE 486
MINEXP 486
MINVAL 487
MOD 487
MPSTR 488
MULTIPLY 489
NULL 489
NULLENTRY 490
NULLSTRPTR、 DEFAULT
ポインター式との併用 55
OFFSET 490
OFFSETADD 490
OFFSETDIFF 490
OFFSETSUBTRACT 491
OFFSETVALUE 491
OMITTED 491
ONACTUAL 491
ONAREA 491
ONCHAR 492
ONCODE 492
ONCONDCOND 492
ONCONDID 493
ONCOUNT 493
ONEXPECTED 494
ONFILE 494
ONGSOURCE 494
ONHBOUND 495
ONKEY 495
ONLBOUND 495
ONLINE 496
ONLOC 496
ONOFFSET 496
ONOPERATOR 496
ONPACKAGE 497
ONPROCEDURE 497
ONSOURCE 497
ONSUBCODE 498
ONSUBCODE2 498
ONSUBSCRIPT 498
ONTEXT 498
ONUCHAR 499
ONUSOURCE 499
ONWCHAR 500
ONWSOURCE 500
ORDINALNAME 501
ORDINALPRED 501
ORDINALSUCC 501

組み込み関数 (続き)
PACKAGENAME 501
PAGENO 502
PICSPEC 502
PLACES 502
PLIRETV 508
PLISTCK 511
PLISTCKE 512
PLISTCKELOCAL 512
PLISTCKEUTC 512
PLISTCKF 512
PLISTCKLOCAL 512
PLISTCKP 513
PLISTCKPLOCAL 513
PLISTCKPUCT 513
PLISTCKUTC 513
PLITRAN11 514
PLITRAN12 514
PLITRAN21 515
PLITRAN22 515
POINTER (PTR) 516
POINTERADD
ポインター演算との併用 55
POINTERADD (PTRADD) 516
POINTERDIFF (PTRDIFF) 517
POINTERSUBTRACT
(PTRSUBTRACT) 517
POINTERVALUE
使用 56
POINTERVALUE (PTRVALUE)
517
POLY 517
POPCNT 518
PRECISION
結果の計算 62
データ変換 74
PRECISION (PREC) 518
PRECVAL 518
PRED 519
PRESENT 519
PROCEDURENAME (PROCNAME)
519
PROD 519
PUTENV 520
QUICKSORT 520
QUICKSORTX 520
RADIX 521
RAISE2 521
RANDOM 522
RANK 522
REAL
データ変換 74
REGEX 523
REM 525
REPATTERN 525
REPEAT 526
replace 526
REVERSE 527
RIGHT 528
ROUND 528
ROUNDDEC 530
ROUNDTOEVEN 530
SAMEKEY 531
SCALE 532
組み込み関数 (続き)
SCALEVAL 532
SCRUBOUT 531
SEARCH 532
SEARCHR 533
SECS 534
SECSTODATE 535
SECSTODAYS 535
SIGN 535
SIGNED
データ変換 74
SIN 536
SIND 536
SINH 536
SIZE 537
SMFTOJULIAN 538
SOURCEFILE 538
SOURCELINE 538
SQRT 538
SQRTF 538
SQUEEZE 539
STACKADDR 539
STCKETODATE 539
STCKTODATE 540
STORAGE 540
STRING 540
SUBSTR 541
SUBTO 542
SUBTRACT 543
SUCC 543
SUM 544
SYSNULL 544
SYSTEM 544
TALLY 545
TAN 545
TAND 545
TANH 545
THREADID 545
TIME 546
TIMESTAMP 546
TINY 546
TRANSLATE 546
TRIM 547
TRUNC 548
TYPE 548
UHIGH 549
ULENGTH 549
ULENGTH16 549
ULENGTH8 549
ULOW 550
UNALLOCATED 550
UNSIGNED
データ変換 74
UNSPEC 551
UPOS 554
UPPERASCII 554
UPPERCASE 554
UPPERLATIN1 555
USUBSTR 555
USUPPLEMENTARY 556
UTCDATETIME 556
UTCMICROSECS 556
UTCSECS 557
UTF8 557

組み込み関数 (続き)
UTF8STG 558
UTF8TOCHAR 558
UTF8TOWCHAR 558
UUID 558
UUID4 559
UVALID 559
UWIDTH 560
VALID 560
VALIDDATE 561
VALIDVALUE 562
VARGLIST 562
VARGSIZE 562
VERIFY 563
VERIFYR 564
WCHARVAL 564
WEEKDAY 564
WHEREDIFF 565
WHIGH 565
WHITESPACECOLLAPSE 567
WHITESPACEREPLACE 568
WIDECHAR
データ変換 74
WIDECHAR (WCHAR) 565
WLOW 566
WSCOLLAPSE 566
WSCOLLAPSE16 567
WSREPLACE 567
WSREPLACE16 568
XMLCHAR 568
XMLCLEAN 570
XMLSCRUB16 570
XMLUCHAR 571
Y4DATE 571
Y4JULIAN 572
Y4YEAR 572
組み込み関数、その他
要約 387
組み込み関数および疑似変数の呼び出し 373
組み込み関数の宣言 373
組み込み疑似、要約 390
組み込みサブルーチン
宣言 373
定義 104
要約 395
呼び出し 374
LOCNEWSPACE 467
LOCNEWVALUE 468
PLIASCII 503
PLIATTN 503
PLICANC 504
PLICKPT 504
PLIDELETE 504
PLIDUMP 504
PLIEBCDIC 504
PLIFILL 505
PLIFREE 505
PLIMOVE 505
PLIOVER 506
PLIPARSE 506
PLIREST 508
PLIRETC 508
PLISAXA 509
PLISAXB 509
組み込みサブルーチン (続き)
PLISAXC 509
PLISAXD 510
PLISRTA 510
PLISRTB 511
PLISRTC 511
PLISRTD 511
組み込みサブルーチンの呼び出し 374
組み込み名
組み込み関数との併用 106
サブルーチンとの併用 104
位取り係数 (scaling factor)
説明 23
文字 337
グラフィック定数
構文 36
ストリング 287
説明 36
比較演算 65
グラフィック・データ
定数 (constant) 36
伝送 287
フォーマット項目 320
変換 85
GX (グラフィック 16 進) ストリング定数 36
グラフィック・データ、変換 (GRAPHIC) 441
グループ、ステートメントの 10
クロスセクション、配列の 174
計算条件
CONVERSION 351
FIXEDOVERFLOW 355
INVALIDOP 356
OVERFLOW 358
UNDERFLOW 364
ZERODIVIDE 365
計算タイプと序数タイプ 47
計算データ
ストリング・データ 17
説明 16
属性 16
変換 74
計算データ・タイプ
ストリング・データ
説明 29
ワイド文字 38
BIT 属性 29
CHARACTER 属性 29
graphic 36
GRAPHIC 属性 29
NONVARYING 属性 31
UCHAR 属性 29
VARYING 属性 31
VARYING4 属性 31
VARYINGZ 属性 31
WIDECHAR 属性 29
ストリングの反復因数 33
属性 22
BINARY 属性と DECIMAL 属性 23
REAL 属性と COMPLEX 属性 24
結果、算術演算の
ディスカッション 57
特殊な場合 62
FLOAT オペランド 58
言語間通信

言語間通信 (続き)
リンケージ
OPTLINK 130
SYSTEM 130
LINKAGE オプション 130
言語に固有のデフォルト
説明 165
定義 165
リストア 170
工業規格 xxxiv
構造式
範囲 71
構造体
修飾 141
修飾名 176
宣言 (declaration) 174
属性 22
組織の指定 174
代入 (assignment) 200, 201
タイプ付き
説明 137
ハンドル 139
HANDLE 組み込み関数 139
定義 137, 174
名前
基本 174
共用体の 175
小 174
説明 174
大 174
配列のクロスセクション 184
被制御 241
メンバー・エレメント 175
レベル
共用体の 176
共用体の最高位番号 176
共用体の最大番号 176
構造の最高位番号 175
構造の最大番号 175
説明 174
DEFINE STRUCTURE ステートメント 137
expression 52
INDFOR 属性 180
LIKE 属性 178
variable 178
構造体 ID 46
構造体タイプ、定義 137
構造体または共用体配列のクロスセクション 184
構造体マッピング
説明 184
対を作る順序の規則 185
例 187
1 つの対のマッピングを行うための規則 186
UNALIGNED 属性による影響 186
構文図の読み方 xxxi
コード化算術データ
構文 22
属性
省略形 23
タイプ 19, 21
変換ターゲット 78
10 進固定小数点 26
10 進浮動小数点 28
2 進固定小数点データ 25
コード化算術データ (続き)
2 進浮動小数点 27
BINARY 属性と DECIMAL 属性 23
FIXED および FLOAT 属性 23
PRECISION 属性 23
REAL 属性と COMPLEX 属性 24
固定小数点
フォーマット項目
説明 319
ピクチャー位取り係数の指定 337
10 進データ 26
2 進データ 25
固定情報
graphic 36
WX (ワイド文字) ストリング 38
コメント
説明 8
コロン記号 6
混合ストリング定数 37
混合データ 37
コンテキスト宣言 149
コンパイル単位 87
コンマ 6

## [サ行]

再帰
自動変数に及ぼす影響 99
属性 93, 110
再帰が自動変数に及ぼす影響 99
再帰的プロシージャー
自動変数に及ぼす影響 99
説明 99
属性の指定 99
例 99
サイズ
単純パラメーター (simple parameter) 95
被制御パラメーター (controlled parameter) 95
索引データ・セット 274
サブフィールド、数字データの 327
サブルーチン
組み込み 104
制限 103
戻り 122
例 103
PROCEDURE ステートメントで識別する 92
サブルーチン、組み込み
呼び出し 374
list 395–397
LOCNEWSPACE 467
LOCNEWVALUE 468
PLIASCII 503
PLIATTN 503
PLICANC 504
PLICKPT 504
PLIDELETE 504
PLIDUMP 504
PLIEBCDIC 504
PLIFILL 505
PLIFREE 505
PLIMOVE 505
PLIOVER 506
PLIPARSE 506
PLIREST 508

サブルーチン、組み込み (続き)
PLIRETC 508
PLISAXA 509
PLISAXB 509
PLISAXC 509
PLISAXD 510
PLISRTA 510
PLISRTB 511
PLISRTC 511
PLISRTD 511
算術演算
結果
ディスカッション 57
特殊な場合 62
FLOAT オペランド 58
RULES(ANS) のもとでの結果 59
説明 56
データ変換 56
算術演算子
使用 7
説明 56
算術演算での位取り 57
算術演算の結果、RULES (ANS) における 59
算術組み込み関数
最大 472
要約 375
ABS 397
CEIL 414
COMPLEX 419
CONJG 420
FLOOR 437
IMAG 449
MAXVAL 474
MIN 486
MINVAL 487
MOD 487
RANDOM 522
REAL 522
REM 525
ROUND 528
ROUNDAWAYFROMZERO 530
ROUNDTOEVEN 530
SIGN 535
TRUNC 548
算術データ
コード化 16
数値ピクチャー 16
算術ピクチャー指定
使用 39
説明 32
算術文字データ
使用 39
挿入編集文字 39
PICTURE データへの変換 81
参照
構文 51
説明 51
プリプロセッサー 583
ロケーター 245
時間のみのパターン 380
式
値の代入 203
エレメント 52
計算の順序 53
式 (続き)
構造 52
構文 51
式の中間結果 61
スカラー 52
制限付き
組み込み関数の適用 72
説明 71
例 72
説明 51
ターゲットの 53
タイプ 52
配列 70
プリプロセッサー 583
命令形式
クラス 54
定義 51
ディスカッション 54
式と参照の計算順序 68
式の計算順序 53
式の中間結果
ディスカッション 54
例 61
識別子
アスタリスク 6
キーワードの使用法 5
構造 46
スカラー 45
定義 5
プログラマー定義の名前 6
2 バイト文字の DBCS 11
DBCS 11
DBCS フォーマットの SBCS 11
自己定義データ (REFER オプション) 249
指数指定子 336
実行する際の制限事項 615
指定
データ・リスト項目の伝送 301
反復 298
編集ディレクティブ 306
リスト・ディレクティブ 310
指定文字 325
自動ストレージ
説明 235
AUTOMATIC 属性の構文 237
自動変数、再帰の影響 99
集合の割り当て 201
修飾
共用体 176
構造 176
説明
区切り文字としての使用 6
修飾された参照 176
終了
開始ブロック (begin-block) 110
スレッド 369
プロシージャー 98
ブロック 89, 218, 229
program 88
出力
定義 273
出力と入力
区域の 256
条件 340

出力と入力 (続き)
ディスカッション 273
主プロシージャー (main procedure)
引数の引き渡し 109
呼び出し 88
主プロシージャーの呼び出し 88
順序、計算の
式と参照の 68
条件
各種 340, 341
クラス 340, 341
計算の 340, 341
出力と入力 340
入出力 340, 341
プログラム・チェックアウト 340, 341
ANYCONDITION 347
AREA 348
ASSERTION 349
ATTENTION
説明 349
マルチスレッド化の 370
CONDITION 350
CONFORMANCE 351
CONVERSION 351
ENDFILE 353
ENDPAGE 353
ERROR 354
FINISH 355
FIXEDOVERFLOW 355
INVALIDOP 356
KEY 357
NAME 357
OPTIMIZATION のもとでの発生 341
OVERFLOW 358
RECORD 358
SIZE 359
status 340, 341
STORAGE 360
STRINGRANGE 361
STRINGSIZE 361
SUBCRIPTRANGE 362
TRANSMIT 362
UNDEFINEDFILE 363
UNDERFLOW 364
ZERODIVIDE 365
条件コード
ディスカッション 339
条件コード、ONCODE 組み込み関数との併用 492
条件処理
暗黙処置 (implicit action) 339
確立された処置 (established action) 339
条件接頭語の有効範囲 341
条件の割り込み可能化 339
条件の割り込み禁止化 339
説明 339
複数条件 345
マルチスレッド化 370
割り込み可能条件の確立 339
CONDITION 属性 346
ON ステートメント
確立されたアクションの有効範囲 343
構文 341
説明 341
動的に派生した ON ユニット 343
条件処理 (続き)
ON ステートメント (続き)
ヌル ON ユニット 342
ファイル変数のための ON ユニット 343
RESIGNAL ステートメント 345
REVERT ステートメント 344
SIGNAL ステートメント 345
条件処理組み込み関数
要約 379
DATAFIELD 424
ONACTUAL 491
ONAREA 491
ONCHAR 492
ONCODE 492
ONCONDCOND 492
ONCONDID 493
ONCOUNT 493
ONEXPECTED 494
ONFILE 494
ONGSOURCE 494
ONHBOUND 495
ONKEY 495
ONLBOUND 495
ONLINE 496
ONLOC 496
ONOFFSET 496
ONOPERATOR 496
ONPACKAGE 497
ONSOURCE 497
ONSUBSCRIPT 498
ONTEXT 498
ONUCHAR 499
ONUSOURCE 499
ONWCHAR 500
ONWSOURCE 500
UTF 処理組み込み関数
ONUSOURCE 499
条件処理組み込み関数 ONPROC
ONPROCEDURE 497
条件接頭語 (condition prefix)
構文 339
使用 339
説明 9
例 340
小構造体名 174
小数点と数字の指定子 327
商標 625
初期値
共用体の 265
STATIC 変数での 270
初期化
基底付き変数と被制御変数 271
共用体 270
自動変数 271
静的変数 270
配列変数 269
序数
許容属性 143
組み込み関数 144
説明 136
定義 136
配列と併用 144
例 137
DEFINE ORDINAL ステートメント 136

序数 (続き)
DO ループの例 144
DOWNTHRU の使用 217
options 136
ORDINAL 属性 141
PRECISION 属性 137
SIGNED 属性 137
UNSIGNED 属性 137
VALUE 属性 137
序数処理組み込み関数
要約 389
list 144
ORDINALNAME 501
ORDINALPRED 501
ORDINALSUCC 501
序数データ、属性、分類 21
処理モード
移動 293
説明 293
locate 293
処理リスト 256
数学組み込み関数
正確度 375
要約 386
ACOS 398
ASIN 402
ATAN 402
ATAND 402
ATANH 403
COS 421
COSD 421
COSH 421
ERF 429
ERFC 429
EXP 430
GAMMA 438
LOG 470
LOG10 470
LOG2 470
LOGGAMMA 470
SIN 536
SIND 536
SINH 536
SQRT 538
SQRTF 538
TAN 545
TAND 545
TANH 545
数学組み込み関数の正確度 375
数字データ
サブフィールド 327
定義 39
ピクチャー指定子 326
フィールド 327
変換 81
数字ピクチャー項目
説明 325
ディスカッション 327
スカラー ID 45
スタックに入れる 99
ステートメント
お勧めするコーディング 8
構文 8
ステートメント・エレメント
ステートメント・エレメント (続き)
DBCS の場合 12
SBCS の場合 5
ストリーム指向データ伝送
定義 273
リスト・ディレクティブ 295
ストリング演算子 () 7
ストリング・オーバーレイ定義 264
ストリング処理組み込み関数
ビット 411
要約 393
BOOL 412
CENTERLEFT 414
CENTERRIGHT 415
CHARACTER 415
CHARGRAPHIC 416
COLLAPSE 418
COPY 420
EDIT 427
GRAPHIC 441
HIGH 447
INDEX 450
INDEXR 450
LEFT 466
LENGTH 466
LOW 471
LOWERASCII 471
LOWERCASE 471
LOWERLATIN1 472
MAXLENGTH 475
MPSTR 488
REGEX 523
REPEAT 526
replace 526
REVERSE 527
RIGHT 528
SCRUBOUT 531
SEARCH 532
SEARCHR 533
SQUEEZE 539
SUBSTR 541
SUBTO 542
TALLY 545
TRANSLATE 546
TRIM 547
UHIGH 549
ULENGTH 549
ULENGTH16 549
ULENGTH8 549
ULOW 550
UPOS 554
UPPERASCII 554
UPPERCASE 554
UPPERLATIN1 555
USUBSTR 555
USUPPLEMENTARY 556
UTF8 557
UTF8STG 558
UTF8TOCHAR 558
UTF8TOWCHAR 558
UVALID 559
UWIDTH 560
VERIFY 563
VERIFYR 564

ストリング処理組み込み関数 (続き)
WHIGH 565
WIDECHAR 565
WLOW 566
ストリング・データ
引用符 16
可変長の伝送 287
属性
省略形 30
長さの指定 30
分類 19
定義 17
反復因数 (repetition factor) 33, 269
ミックスされた 37
文字データ 32
bit 34
BIT 属性 29
CHARACTER 属性 29
graphic 36
GRAPHIC 属性 29
NONVARYING 属性 31
PICTURE 属性 32
VARYING 属性 31
VARYING4 属性 31
VARYINGZ 属性 31
WIDEPIC 属性 32
ストリング内の引用符 16
ストレージ制御組み込み関数
要約 391
ADDR 398
ADDRDATA 399
ALLOC31 400
ALLOCATE 400
ALLOCATION 401
ALLOCNEXT 401
ALLOCSIZE 401
AUTOMATIC 403
AVAILABLEAREA 403
BINARYVALUE 410
BITLOCATION 412
CHECKSTG 417
CURRENTSIZE 423
CURRENTSTORAGE 423
EMPTY 428
ENTRYADDR 429
HANDLE 442
LOCATION 467
LOCSTG 469
LOCVAL 469
NULL 489
NULLENTRY 490
OFFSET 490
OFFSETADD 490
OFFSETDIFF 490
OFFSETSUBTRACT 491
OFFSETVALUE 491
POINTER 516
POINTERADD 516
POINTERDIFF 517
POINTERSUBTRACT 517
POINTERVALUE 517
SIZE 537
STORAGE 540
SYSNULL 544
ストレージ制御組み込み関数 (続き)
SYSTEM 544
UNALLOCATED 550
スペース・フォーマット項目 323
スレッド
切り離し 370
作成 367
終了 369
使用 367
条件処理 370
待機 369
タスク変数 (task variable) 370
データの共用 371
取り消し 369
ファイルの共用 371
ATTACH ステートメント 368
ENVIRONMENT オプション 368
TASK 属性 370
THREAD オプション 368
TSTACK オプション 368
スレッド間でのファイル共用 371
スレッド間のデータの共用 371
制御ストレージ 235
制限、FETCH および RELEASE の
説明 100
制限事項 615
制限付き式
組み込み関数の適用 72
説明 71
例 72
整数
値 23
整数演算組み込み関数
要約 385
IAND 447
ICLZ 447
IEOR 448
INOT 451
IOR 452
ISIGNED 452
ISLL 453
ISRL 454
IUNSIGNED 455
LOWER2 472
RAISE2 521
静的ストレージ 235, 236
静的割り振り 235
精度処理組み込み関数
固定 434
追加 398
要約 389, 390
BINARY 409
DECIMAL 426
DIVIDE 427
FIXEDBIN 435
FIXEDDEC 435
FLOAT 436
FLOATBIN 436
FLOATDEC 437
MULTIPLY 489
PRECISION 518
PRECVAL 518
SCALEVAL 532
SIGNED 536

精度処理組み込み関数 (続き)
SUBTRACT 543
UNSIGNED 551
セット、データの 273
接頭部 (prefix)
condition
構文 339
使用 339
例 340
セミコロン 6
ゼロ置換文字 334
ゼロ抑制文字 328
宣言、DEFINE ALIAS、ステートメント 135
宣言、データの
説明 147
属性分配 149
宣言
暗黙的な (implicit) 149
コンテキスト 149
配列 170
明示 147
有効範囲
ディスカッション 150
例 151
INTERNAL 属性と EXTERNAL 属性での定義 152
DEFINE ORDINAL ステートメント 136
走査、プリプロセッサー 581
総称入り口宣言 119
総称記述子 (generic descriptor) 120
総称選択 120
総称名 (generic name) 119
相対データ・セット 274
相対ライン 323
挿入演算 54
挿入演算子と配列 70
挿入文字 329
添え字
介在 184
定義 173
配列の 173
添え字付きの修飾された参照 184
ソースからターゲットへの変換規則
コード化算術 78
固定小数点 10 進数 79
固定小数点 2 進数 79
算術文字 81
算術文字 PICTURE 81
数字 81
ビット 83
浮動小数点 10 進数 80
浮動小数点 2 進数 80
文字 82
ワイド文字 85
graphic 85
uchar 85
属性
共用体データ 22
組み合わせ 283
計算データ 16
形式
説明 44
構造体データ 22
コード化算術 19, 21
序数データ 21
属性 (続き)
ストリング・データ 19, 21
タスク・データ 21
ディスカッション 16
データ・タイプに応じた分類 18
デフォルトのデータ属性 165
名前付きのコード化算術式 19
名前付きのストリング・データ 19
配列データ 21
パラメーター 95
ビット 29
非データ 17
ファイル・データ 20, 21
複合 24
プログラム制御データ 17
ポインター 247
ラベル・データ 20, 21
領域 21
ロケーター・データ 21
ABNORMAL 258
ALIGNED
記憶位置合わせ要件 158
説明 158
ASSIGNABLE 257
AUTOMATIC 237
BASED 242
BIGENDIAN 258
BINARY 23
BUFFERED 279
BUILTIN
使用 104, 373
BYADDR 127
BYVALUE 127
CHARACTER
説明 29
CONDITION 346
connected 260
CONTROLLED 238
data
説明 16
list 17
DATE 40
DECIMAL 23
DEFINED 261
DIMACROSS 172
DIMENSION 171
DIRECT 279
ENTRY 112
ENVIRONMENT 280
EXTERNAL
使用 101
説明 152
FILE 275
FORCE 156
FORMAT
変数タイプによる分類 21
GENERIC 119
GRAPHIC 29
HANDLE 139
HEXADEC 259
IEEE 259
INDFOR 180
INITIAL 265
INPUT 278

属性 (続き)
INTERNAL 152
JSONNAME 183
JSONOMIT 183
JSONTRIMR 183
KEYED 280
label 43
LIKE 178
LIMITED 118
LIST 116
LITTLEENDIAN 258
LOCATES 253
NOINIT 181
NONASSIGNABLE 257
NONCONNECTED 260
NONVARYING 31
NORMAL 258
NULLINIT 181
OFFSET 252
OPTIONAL 114
OPTIONS 123
ORDINAL 141
OUTPUT 278
PARAMETER 95
PICTURE 32
POSITION 261
PRECISION 23
PRINT 312
REAL 24
RECORD 278
RECURSIVE 99
RESERVED 155
RETURNS 132
SEQUENTIAL 279
SIGNED
説明 24
データ・ストレージ所要量 25
STATIC 236
STREAM 278
SUPPRESS 157
TASK 370
TYPE 140
UNALIGNED
記憶位置合わせ要件 158
説明 158
UNBUFFERED 279
UNION 176
UNSIGNED
説明 24
データ・ストレージ所要量 25
UPDATE 278
VALUE 45
VALUELIST 47
VALUELISTFROM 47
VALUERANGE 48
VARIABLE 45
VARYING 31
VARYING4 31
VARYINGZ 31
WIDECHAR
説明 29
WIDEPIC 32
XMLATTR 181, 182
XMLCONTENT 182
属性 (続き)
XMLNAME 182
XMLOMIT 182
属性分配 149
その他の組み込み関数
バイト 412
要約 387
ALLCOMPARE 399
BETWEEN 408
BETWEENEXCLUSIVE 408
BETWEENLEFTEXCLUSIVE 409
BETWEENRIGHTEXCLUSIVE 409
BINSEARCH 410
BINSEARCHX 411
BYTELENGTH 413
CHARVAL 417
COLLATE 418
FOLDEDFULLMATCH 438
FOLDEDSIMPLEMATCH 438
GETENV 439
GETJCLSYMBOL 439
GETSYSINT 439
GETSYSWORD 440
GTCA 441
HEX 443, 550
HEX8 444
IFTHENELSE 448
INDICATORS 451
INLIST 451
ISJCLSYMBOL 453
OMITTED 491
PACKAGENAME 501
PLIRETV 508
POPCNT 518
PRESENT 519
PROCEDURENAME 519
RANK 522
SOURCEFILE 538
SOURCELINE 538
STACKADDR 539
STRING 540
UNSPEC 551
UUID 558
UUID4 559
VALID 560
VALIDVALUE 562
WCHARVAL 564
WHEREDIFF 565
その他の条件
ANYCONDITION 347
AREA 348
ASSERTION 349
ATTENTION 349
CONDITION 350
CONFORMANCE 351
ERROR 354
FINISH 355
STORAGE 360

## ［タ行］

大構造体名 174
代数比較演算子 65
代替属性

代替属性 (続き)
定義 275
BUFFERED および UNBUFFERED 279
INPUT、OUTPUT、および UPDATE 278
RECORD と STREAM 278
SEQUENTIAL と DIRECT 279
代入
エレメント 201
区域 255
構造 200, 201
構造体割り当てのための BY DIMACROSS の使用 204
構造体割り当てのための BY NAME の使用 203
式値 203
集合 (aggregate) 201
配列
集合の割り当て 202
ターゲット変数 200
複合 199
複数 203
代入ステートメント
説明 197
ターゲット変数の要件 199
定義 10
BY DIMACROSS オプション 198
BY NAME オプション 198
代入ステートメントの BY DIMACROSS オプション
構造体割り当てで指定された場合 203
説明 198
タイプ
修飾 141
説明 140
タイプ付き関数 146
定義 135
ハンドル 139
変数 140
DEFINE STRUCTURE ステートメント 137
HANDLE 組み込み関数 139
タイプ付き関数
ディスカッション 575
引数 575
BIND 576
CAST 576
FIRST 576
LAST 577
list 575
NEW 577
RESPEC 577
SIZE 578
VALUE 578
タイプ付き関数の呼び出し 575
タイプ付きの名前 135
タイプ付き変数、宣言
修飾 141
ハンドル 139
タイプ定義、記述 135
タスク・データ、属性、分類 21
タスク変数 (task variable) 370
ダブルワード、データの位置合わせでの 158
単純
オーバーレイ定義 261, 264
ストリング・オーバーレイ定義 264
単純 95
単純定義 262
定義 261, 262
単純 (続き)
パラメーター
境界、長さ、およびサイズ 95
被制御 95
iSUB 定義 263
単純ステートメント (simple statement) 9
調節可能エクステント 241
直接入り口宣言 110
追加属性
定義 275
ENVIRONMENT 280
KEYED 280
通貨記号
説明 332
定義 331
通知、条件の 345
定数
虚数 24
名前付き 45
ファイル 275
文字ストリング 34
10 進固定小数点 27
10 進浮動小数点 28
2 進固定小数点 25
2 進浮動小数点 27
B3 (16 進ビット) ストリング 35
B4 (16 進ビット) ストリング 35
bit 34
BX (16 進ビット) ストリング 35
character 33
entry
使用 111
説明 111
GX (グラフィック) ストリング 36
label 43
M (混合) ストリング 37
UX (UCHAR) ストリング 38
XN (2 進 16 進数) 26
XU (2 進 16 進数) 26
ディレクティブ
印刷 (%P) 227
ページ (%P) 227
*PROCESS 228
*PROCINC 228
%INCLUDE 223
%LINE 225
%NOPRINT 225
%NOTE 225
%POP 227
%PROCESS 227
%PROCINC 228
%PUSH 228
%SKIP 232
データ
文字 32
データ・エレメント
属性 15
定数
引用符 16
句読法 16
名前付き 16
ディスカッション 15
データ項目 15
プリプロセッサー 583

データ型
計算の 16
ストリング・データ
UCHAR 37
ディスカッション 16
データ項目
定義 15
複合 24
モード 24
expression 52
データ項目のモード 24
データ指定オプション、ストリーム入出力の
説明 297
定義 295
データ・ディレクティブ 302
伝送されるデータ 287
データ・セット
記憶 274
索引付き 274
タイプ 274
データの伝送 273
領域の 274
連続 274
relative 274
データ宣言
暗黙的な (implicit) 149
共用体 175
言語に固有の属性デフォルト 165
構造体 174
説明 147
配列 170
明示 147
データ・ディレクティブ・データ伝送 295
データ・ディレクティブのデータ指定
ディスカッション 302
GET ステートメントの使用法 303
PUT ステートメントの使用法 305
データ伝送 (data transmission)
位置合わせされていないビット・ストリング 287
可変長ストリング 287
漢字ストリング 287
区域変数 288
出力 273
ストリーム指向 295
ストリーム指向ステートメント
取得 295
タイプ 3 の DO グループ 298
ディスカッション 295
PUT 296
説明 287
データ集合 287
データ・ディレクティブ 295
データ・リスト項目 301
編集ディレクティブ 295
レコード単位 287
レコード単位ステートメント
ディスカッション 288
DELETE 289
LOCATE 289
READ 288
REWRITE 289
WRITE 288
input 273
TRANSMIT 条件 362
データ伝送ステートメント・オプション
ディスカッション 296
明細 299
COPY 296
FILE 299
PAGE 299
SKIP 299
STRING 300
データの位置合わせ
ストレージ・アドレス 158
ディスカッション 158
ALIGNED 属性と UNALIGNED 属性の使用法 158
データの位置合わせ属性 158
データ変換
エラー 86
組み込み関数での開始 74
計算データ 74
算術値の精度 75
算術演算での 56
算術データからビット・ストリングへの変換、例 85
算術データから文字ストリングへの変換、例 86
ストリングの長さ 74
説明 73
ソースからターゲットへの変換規則 77
変換エラー 86
モード 75
ルール 74
デフォルト、属性の
言語に固有の 165
言語に固有のデフォルト属性の復元 170
説明 165
データ属性 165
DEFAULT ステートメント 166
伝送、データの 273
等号 6
動的に派生した ON ユニット 343
動的ロード、外部プロシージャーの
FETCH ステートメント 100
RELEASE ステートメント 100
動的割り振り 235
特別言語文字 2
取り消し
スレッド 369

## [ナ行]

内部、定義 151
内部プロシージャー (internal procedure) 92
名前
プリプロセッサー 584
名前、タイプ 135
名前付き定数 45
名前付きのコード化算術式の属性 19
名前付きのストリング・データ属性 19
名前付きの定数、説明 16
名前の認識 147
入出力組み込み関数
要約 384, 385
COUNT 421
ENDFILE 428
FILEDDINT 430
FILEDDTEST 431
FILEDDWORD 431
FILEID 432

入出力組み込み関数 (続き)
FILENEW 432
FILEOPEN 433
FILEREAD 433
FILESEEK 433
FILETELL 434
FILEWRITE 434
LINENO 466
ONSUBCODE 498
ONSUBCODE2 498
PAGENO 502
SAMEKEY 531
ヌル ON ユニット 342
ヌル・ステートメント (null statement)
説明 226
定義 10
ヌル引数、組み込み関数での使用 375

## [ハ行]

ハーフワード 158
排他的 OR 演算子 63
バイト、定義 158
配列
境界 (bounds) 171
クロスセクション 174
構造体と共用体 183
次元属性 171, 172
接頭演算子と 70
挿入演算子と 70
添え字 173
属性 21
代入 (assignment) 200, 202
定義 170
配列同士の演算 71
配列とエレメントの演算 70
範囲 171
例 172
expression
説明 52, 70
例 53
targets 200
variable 170
配列式 (array expression)
説明 70
定義 52
例 52
配列処理組み込み関数
要約 376
ALL 399
ANY 402
DIMENSION 427
HBOUND 442
HBOUNDACROSS 442
INARRAY 449
LBOUND 465
LBOUNDACROSS 465
POLY 517
PROD 519
QUICKSORT 520
QUICKSORTX 520
SUM 544
配列の上限、獲得 (HBOUND) 442
配列の下限、獲得 (LBOUND) 465
配列変数 170
パッケージ 90
バッファー管理組み込み関数
BASE64DECODE 404
BASE64DECODE16 405
BASE64DECODE8 404
BASE64ENCODE 405
BASE64ENCODE16 407
BASE64ENCODE8 406
CHECKSUM 418
COMPARE 419
EBCDIC 405
HEXDECODE 445
HEXDECODE8 445
HEXIMAGE 446
HEXIMAGE8 446
MEMCOLLAPSE 476
MEMCONVERT 475
MEMCU12 476
MEMCU14 477
MEMCU21 477
MEMCU24 478
MEMCU41 478
MEMCU42 478
MEMINDEX 479
MEMREPLACE 480
MEMSEARCH 481
MEMSEARCHR 482
MEMSQUEEZE 482
MEMVERIFY 483
MEMVERIFYR 484
PICSPEC 502
PLITRAN11 514
PLITRAN12 514
PLITRAN21 515
PLITRAN22 515
WSCOLLAPSE 566
WSCOLLAPSE16 567
WSREPLACE 567
WSREPLACE16 568
XMLCHAR 568
XMLSCRUB16 570
XMLUCHAR 571
パラメーター記述子 (parameter descriptor)
範囲 114
パラメーター記述子リスト (parameter descriptor list) 113
パラメーター属性 95
パラメーターの付いた配列引数 96
範囲
パラメーター記述子 (parameter descriptor) 114
BASED 宣言 242
ハンドル演算 55
反復因数 (iteration factor) 267, 269, 306
反復因数 (repetition factor)
ストリングの場合 33, 269
ピクチャー文字の場合 325
ビット・データの 35
反復実行 (DO ステートメント) 208, 215
比較演算
オペランドの変換 65
序数データ 65
説明 64
代数 65
ビット 65

比較演算 (続き)
プログラム制御データ 65
ポインターおよびオフセット・データ 65
文字 65
例 66
ワイド文字 65
graphic 65
uchar 65
比較演算子 7
引数
受け渡し
主プロシージャーへ 109
プロシージャーへの 106
仮の
説明 108
属性の導出 108
ルール 108
指定 127, 374
引数およびパラメーターの関連 106
引数の受け渡し
主プロシージャーへ 109
ディスカッション 106
BYVALUE および BYADDR の使用 106
INONLY、INOUT、および OUTONLY の使用 107
引数の集合 374
ピクチャー指定文字
定義 325
9
数字の場合 327
文字データの場合 325
A 325
B 329
CR 334
DB 334
E 336
F 337
I 334
K 336
R 334
S 332
T 334
V
数字の場合 327
挿入 330
X 325
Y 334
Z 328
- 332
* 328
/ 329
+ 332
$ 332
ピクチャー・データ (picture data)
位取り係数 (scaling factor) 337
指定 32
数字データの指定子 326
反復因数 (repetition factor) 325
文字データの指定子 325
PICTURE 属性の構文 32
ピクチャー・フォーマット項目 321
非結合ストレージ (nonconnected storage) 174
被制御
構造体と共用体メンバー 241
パラメーター 95
被制御 (続き)
変数
説明 238
複数生成 240
ALLOCATE ステートメントの使用法 238
FREE ステートメントの使用法 240
storage 235, 238
日付/時刻組み込み関数
要約 380
リリアン形式 380
DATE 424
DATETIME 424
DAYS 424
DAYSTODATE 425
DAYSTOMICROSECS 426
DAYSTOSECS 426
JULIANTOSMF 465
MAXDATE 473
MICROSECS 484
MICROSECSTODATE 485
MICROSECSTODAYS 485
MINDATE 486
REPATTERN 525
SECS 534
SECSTODATE 535
SECSTODAYS 535
SMFTOJULIAN 538
STCKETODATE 539
STCKTODATE 540
TIME 546
TIMESTAMP 546
UTCDATETIME 556
UTCMICROSECS 556
UTCSECS 557
VALIDDATE 561
VARGLIST 562
VARGSIZE 562
WEEKDAY 564
Y4DATE 571
Y4JULIAN 572
Y4YEAR 572
ビット
変換
説明 76
ルール 83
ビット演算子
使用 63
例 64
ビット・ストリングの伝送、位置合わせされていない 287
ビット・データ
反復因数 (repetition factor) 35
ビット・フォーマット項目 315
非データ属性 17, 47, 48
標準 xxxiv
非連結ストレージ 174, 262
ファイル
暗黙オープン 282
オープンとクローズ 280
参照の指定 278
スレッド間での共用 371
宣言 (declaration) 274
属性 20
属性の説明 275
代替属性 275

ファイル (続き)
追加属性 (additive attribute) 275
定義 274
定数 (constant) 275
FILE 属性 275
PRINT 313
sysin 285
SYSPRINT 285
variable 277
ファイル・データ 21
ファイルのオープンとクローズ 280
フィールド 327
ブール演算子 63
フォーマット項目
説明 306
明細 321
A 315
B 315
C 316
COLUMN 316
E 317
F 319
G 320
L 321
P 321
PAGE 322
R 322
SKIP 322
V 323
X 323
フォーマット・データ (format data) 44
フォーマット表記規則 xxxi
復元、言語に固有のデフォルトの 170
複合
データ項目 24
フォーマット項目 316
複合記号 4
複合ステートメント (compound statement) 10
複合代入演算子 199
複数条件 345
複数世代、被制御変数の 240
複数割り当て 203
含まれている、定義 150
符号
固定的使用 333
異なる符号を持つ CR と DB の使用 334
数字データでの指定 332
浮動的使用 333
浮動小数点
データ変換 80
フォーマット項目 317
10 進データ 28
2 進データ 27
浮動小数点演算組み込み関数
要約 384
EXPONENT 430
PRED 519
SCALE 532
SUCC 543
浮動小数点の照会組み込み関数
要約 383
EPSILON 429
HUGE 447
ISFINITE 452
浮動小数点の照会組み込み関数 (続き)
ISINF 453
ISNAN 454
ISNORMAL 454
ISZERO 455
MAXEXP 473
MINEXP 486
PLACES 502
RADIX 521
TINY 546
浮動文字 333
ブランク
区切り文字としての使用 6
説明 7
プリプロセッサー
機能 579
組み込み関数 589
参照と式 583
出力 579
出力テキスト 579
ステートメント、リスト 600
走査
およびプリプロセッサー・ステートメント 581
およびリスト制御ステートメント 582
説明 581
入力テキスト 582
名前の有効範囲 584
入力テキスト 579, 582
プリプロセッサー 579
プロシージャー 584
変数とデータ・エレメント 583
リスト制御 579
リスト制御ステートメント 579
例 609
input 579
statements
説明 579
のリスト 581
%ACTIVATE 600
%assignment 601
%CALL 589
%DEACTIVATE 601
%DECLARE 601
%DO 603
%END 604
%GO TO 604
%IF 604
%INCLUDE 605
%INSCAN 606
%ITERATE 606
%LEAVE 607
%NOTE 607
%null 608
%REPLACE 608
%SELECT 608
%XINCLUDE 609
%XINSCAN 609
フルワード 158
プログラマー定義の名前 6
プログラム
エレメント
CALL ステートメント 121
ブロック
活動化 89

プログラム・エレメント
ステートメント
単純 9
ディスカッション 8
複合 10
説明 1
1 バイト文字セット (SBCS)
ステートメント・エレメント 5
ディスカッション 1
2 バイト文字セット (DBCS)
ステートメント・エレメント 12
ディスカッション 11
group 10
プログラム制御データ
使用 43
説明 17
タイプおよび属性 43
プログラム・チェックアウト条件
STRINGRANGE 361
STRINGSIZE 361
SUBSCRIPTRANGE 362
プログラムの正常終了 88
プログラムの編成 87
プログラム・ブロックの定義 87
プロシージャー
外部 92
活動化 97
再帰的 99
終了 98
制御の伝送 98
説明 92
属性の指定 95
動的ロード
規則 100
ディスカッション 100
FETCH ステートメントの使用法 101
RELEASE ステートメントの使用法 101
内部 92
引数の受け渡し
仮引数の使用 108
ディスカッション 106
BYVALUE および BYADDR の使用 106
INONLY、INOUT、および OUTONLY の使用 107
プリプロセッサー 584
メインへの引数の引き渡し 109
ブロック
開始 109
活動化 89
終了 89
説明 88
タイプ 88
パッケージ 90
プロシージャー 92
別名
定義 135
DEFINE ALIAS ステートメント 135
変換
エラー 86
オペランド 59
組み込み関数を使用した 74
算術値の精度 75
算術演算での 56
ストリングの長さ 74
ソースからターゲットへの変換規則 77
変換 (続き)
その他のデータ属性の 76
モード 75
連結演算 66
ロケーター・データの 245
data 73
変換エラー 86
変更の要約 xxxiv
編集ディレクティブ
データ伝送 (data transmission) 295
フォーマット項目 315
編集ディレクティブ・データ指定 306
変数
基底付き
識別 242
使用 247
構造 174
参照 15
自動 99
定義 15
ディスカッション 238
配列 170
被制御 238
複素数データ項目を表す 24
プリプロセッサー 583
ポインター 244, 247
entry 111
offset 244
targets 53
変数、タイプ付き 140
変数、ハンドルとしての 139
ポインター演算 55
ポインター記号 6
ポインター変数 (pointer variable) 244, 247

## [マ行]

マクロ機能組み込み関数
シスパーム 597
COLLATE 590
COMMENT 590
COMPILEDATE 590
COMPILETIME 591
COPY 592
COUNTER 592
DIMENSION 592
HBOUND 593
INDEX 593
LBOUND 593
LENGTH 594
MACCOL 594
MACLMAR 595
MACRMAR 595
MAX 595
MIN 595
PARMSET 595
QUOTE 596
REPEAT 596
SUBSTR 596
SYSDIMSIZE 597
SYSOFFSETSIZE 597
SYSPOINTERSIZE 598
SYSTEM 598
SYSVERSION 598

マクロ機能組み込み関数 (続き)
TRANSLATE 598
VERIFY 599
マルチスレッド化
条件処理 370
スレッド
切り離し 370
作成 367
終了 369
使用 367
待機 369
cancel 369
スレッド間のデータの共用 371
スレッド間のファイルの共用 371
説明 367
タスク変数 (task variable) 370
リンケージ要件 368
ATTACH ステートメント 368
options
ENVIRONMENT 368
THREAD 368
TSTACK 368
TASK 属性 370
THREADID 組み込み関数 370
マルチスレッド化、THREADID 組み込み関数の 545
マルチスレッド化機能 367
明示宣言 (explicit declaration) 147
明示的にロケーター修飾された参照 246
モード、処理の
移動 293
説明 293
locate 293
文字
英字 1
英数字 2
セット
1 バイト 1
2 バイト 11
ゼロ消去 328
挿入 329
定数 (constant) 33
特殊 3
特別言語 2
比較演算での使用 65
ピクチャー指定 (picture specification) 32
フォーマット項目 315
文字データ
説明 32
ピクチャー指定子 325
変換 76, 82
文字ストリング定数 (character string constant) 34
文字セット
ディスカッション 1
1 バイト
区切り文字と演算子 6
ステートメント・エレメント用 5
DBCS フォーマットの ID 11
ID 5
2 バイト
識別子 11
ステートメント・エレメント 12

## [ヤ行]

有効範囲
確立されたアクションの 343
条件接頭語の 341
プリプロセッサー名 584
ラベル宣言の 150
抑止文字 328
呼び出されたプロシージャー (invoked procedure) 97
呼び出し、タイプ付き関数の 575
呼び出し側ブロック (invoking block) 97
呼び出し規則
OPTLINK 130
SYSTEM 130
呼び出し点、プロシージャーの 97

## [ラ行]

ラベル定数 43
ラベル・データ
説明 43
属性 20, 21
リスト制御ステートメント 579
リスト・ディレクティブ
出力 312
データ指定 (data specification) 310
データ伝送 (data transmission) 295
GET ステートメント 311
input 311
PUT ステートメント 312
リモート・フォーマット項目 (remote format item) 322
領域
属性 21
入出力 256
EMPTY 組み込み関数 428
IN オプション付きの ALLOCATE ステートメント 248
領域データ・セット 274
リリアン形式 380
累乗演算、特殊な場合 62
レコード単位データ伝送 (record-oriented data transmission)
定義 273
ディスカッション 287
statements 288
レベル、構造体の
固有名の指定 176
説明 174
レベル番号 (構造体エレメントの) 185
連結 (concatenation)
演算 66
演算子 7
連結ストレージ (connected storage) 260
連続データ・セット 274
ロード・モジュール
説明 87
ファイル拡張子 87
ロケーター
参照 245
修飾 245
修飾子 6
修飾のレベル 247
パラメーター 108
変換 245
data

ロケーター (続き)
data (続き)
オフセット変数 (offset variable) 244
修飾 245
説明 244
属性 21
ポインター変数 (pointer variable) 244
論理演算子
使用 7
ディスカッション 63
論理レベル (構造体エレメントの) 185

## [ワ行]

ワイド文字ストリング定数 38
ワイド文字定数
比較演算 65
ワイド文字データ
変換 85
WX (ワイド文字 16 進) ストリング定数 38
割り当てステートメントの BY NAME オプション
構造体割り当てで指定された場合 202
構造体割り当てで指定されなかった場合 202
説明 198
割り込み可能条件 339
割り振り 235

## [数字]

1 バイト文字セット (SBCS)
英字 1
ステートメント・エレメント 5
ディスカッション 1
特別言語 2
10 進数字 2
16 進数字 (hexadecimal digit) 3
2 進数字 (binary digit) 3
10 進固定小数点定数 (decimal fixed-point constant) 27
10 進固定小数点データ
説明 26
変換 79
10 進数字 2
10 進浮動小数点定数 28
10 進浮動小数点データ
説明 28
変換 80
16 進 (X) の文字ストリング定数 34
16 進数字 (hexadecimal digit) 3
2 進固定小数点定数 26
2 進固定小数点データ
説明 25
変換 79
2 進数字 (binary digit) 3
2 進浮動小数点定数 27
2 進浮動小数点データ
説明 27
変換 80
2 バイト文字セット (DBCS)
グラフィック・データ内の 36
継続規則 13
ステートメント・エレメント 12
ソース・プログラム内での使用 11
ディスカッション 11
2 バイト文字セット (DBCS) (続き)
入出力ストリーム内のデータ 313
ID 11
9 ピクチャー指定文字
使用 327
文字データの場合 325

## A

A (ASCII) 文字定数
文字定数 33
A ピクチャー指定文字 325
A フォーマット項目 315
ABNORMAL 属性 258
ABS 組み込み関数 397
ACOS 組み込み関数 398
ADD 組み込み関数 398
ADDR 組み込み関数 398
ADDRDATA 組み込み関数 399
ALIGNED attribute
記憶位置合わせ要件 158
説明 158
example 164
ALL 組み込み関数 399
ALLCOMPARE 組み込み関数 399
ALLOC (ALLOCATE) ステートメント 238
ALLOC31
組み込み関数 (built in function)
構文 400
ALLOCATE (ALLOC)
組み込み関数 (built in function)
基底付き区域変数の場合 243
基底付き変数 243, 247
構文 400
ステートメント 238
ALLOCATION (ALLOCN) 組み込み関数 401
ALLOCNEXT 組み込み関数 401
ALLOCSIZE 組み込み関数 401
AMODE31 または AMODE64 オプション 127
ANSWER ステートメント
プリプロセッサー・プロシージャーでの使用 587
ANY 組み込み関数 402
ANYCONDITION 条件 347
AREA
condition 348
ASIN 組み込み関数 402
ASM (ASSEMBLER) オプション 127
ASSEMBLER (ASM) オプション 127
ASSERT ステートメント 193
ASSERTION
条件 (condition) 349
ASSERTION 条件接頭語 340, 341
ASSIGNABLE 属性 257
ATAN 組み込み関数 402
ATAND 組み込み関数 402
ATANH 組み込み関数 403
ATTACH ステートメント 368
ATTENTION (ATTN) 条件
説明 349
マルチスレッド化 370
attributes
ALIGNED
example 164
UNALIGNED

attributes (続き)
UNALIGNED (続き)
example 164
AUTOMATIC (AUTO) 組み込み関数 403
AUTOMATIC 組み込み関数
基底付き区域変数の場合 243
基底付き変数の場合 243, 247
AUTOMATIC、(AUTO) 属性 237
AUTOMATIC、(AUTOMATIC) 属性 237
AVAILABLEAREA 組み込み関数
区域変数の場合 255
構文 403

## B

B (挿入文字) 330
B フォーマット項目 315
B3 (16 進ビット) ビット・ストリング定数 35
B4 (16 進ビット) ビット・ストリング定数 35
BASE64DECODE 組み込み関数
構文 404
BASE64DECODE16 組み込み関数
構文 405
BASE64DECODE8 組み込み関数
構文 404
BASE64ENCODE 組み込み関数
構文 405
BASE64ENCODE16 組み込み関数
構文 407
BASE64ENCODE8 組み込み関数
構文 406
BASED 属性 242
BEGIN ステートメント
説明 109
有効な OPTIONS オプション 123
BETWEEN 組み込み関数 408
BETWEENEXCLUSIVE 組み込み関数 408
BETWEENLEFTEXCLUSIVE 組み込み関数 409
BETWEENRIGHTEXCLUSIVE 組み込み関数 409
BIGENDIAN 属性 258
BINARY (BIN) 組み込み関数 409
BINARY (BIN) 属性 23
BINARYVALUE (BINVALUE) 組み込み関数 410
BINARYVALUE 組み込み関数
序数の 144
ポインター式との併用 56
BIND タイプ付き関数 576
BINSEARCH (BIN) 組み込み関数 410
BINSEARCHX (BIN) 組み込み関数 411
bit
演算子
説明 7
ビット演算子での使用 63
定数 (constant) 34
data 34
BIT 組み込み関数 411
BIT 属性 29
BITLOCATION (BITLOC) 組み込み関数 412
BOOL 組み込み関数 412
BUF (BUFFERED) 属性 279
BUFFERED (BUF) 属性 279
BUILTIN 属性
組み込み関数の名前を宣言する 104
BX (16 進ビット) ビット・ストリング定数 35
BYADDR オプション 127
BYADDR 属性 127
BYTE 組み込み関数 412
BYTELENGTH 組み込み関数 413
BYVALUE オプション 127
BYVALUE 属性 127

## C

C フォーマット項目 316
CALL オプション、INITIAL 属性の 267
CALL ステートメント 121
CAST タイプ付き関数 576
CDS 組み込み関数 413
CEIL 組み込み関数 414
CELL、同義語 176
CENTERLEFT (CENTER) 組み込み関数 414
CENTERRIGHT 組み込み関数 415
CHARACTER (CHAR) 組み込み関数 415
CHARACTER (CHAR) 属性
説明 29
CHARGRAPHIC (CHARG) 組み込み関数 416
CHARGRAPHIC オプション 128
CHARVAL 組み込み関数 417
CHECKSTG 組み込み関数 417
CHECKSUM 組み込み関数 418
CLOSE ステートメント 284
CMPAT オプション 129
COBOL オプション 129
CODEPAGE 組み込み関数 418
COLLAPSE 組み込み関数 418
COLLATE 組み込み関数 418
COLLATE マクロ機能組み込み関数 590
COLUMN キーワード
ANSWER プリプロセッサー・ステートメントの 588
COLUMN フォーマット項目 316
COMMENT マクロ機能組み込み関数 590
COMPARE 組み込み関数 419
COMPILEDATE マクロ機能組み込み関数 590
COMPILETIME マクロ機能組み込み関数 591
COMPLEX (CPLX) 組み込み関数 419
COMPLEX (CPLX) 属性 24
COND (CONDITION) 条件 350
CONDITION (COND) 条件 350
CONDITION 属性 346
CONFORMANCE 条件 351
CONJG 組み込み関数 420
CONNECTED (CONN) 属性 260
CONTROLLED (CTL) 属性 238
CONV (CONVERSION) 条件 351
CONVERSION (CONV) 条件 351
CONVERSION 条件接頭語 340, 341
COPY オプション 296
COPY 組み込み関数 420
COPY マクロ機能組み込み関数 592
COS 組み込み関数 421
COSD 組み込み関数 421
COSH 組み込み関数 421
COUNT 組み込み関数 421
COUNTER マクロ機能組み込み関数 592
CS 組み込み関数 421
CTL (CONTROLLED) 属性 238
CURRENTSIZE 組み込み関数 423
CURRENTSTORAGE 組み込み関数 423

## D


data
- 位置合わせ 158
- エレメント 15, 583
- 区域 251
- 計算の 16
- 形式 44
- 項目 15
- 算術文字 39
- 指定 297
- 数字 326
- スレッド間での共用 371
- 属性 16
- タイプ 16
- 伝送 273
- ビット定数 34
- フォーマット項目 315
- プログラム制御
  - 説明 17
  - タイプおよび属性 43
- 変換
  - エラー 86
  - 組み込み関数を使用した 74
  - 算術演算での 56
  - 説明 73
  - ソースからターゲットへの変換規則 77
- ミックスされた 37
- 文字定数
  - A (ASCII) 文字定数 33
  - E (EBCDIC) 文字定数 34
- ラベル、言語ステートメントの 43
- ロケーター 244
- ワイド文字 38
- 10 進固定小数点 26
- 10 進浮動小数点 28
- 2 進固定小数点 25
- 2 進浮動小数点 27
- bit 34
- entry 110
- graphic 36
- label 43
- LABEL 属性
  - 有効な OPTIONS オプション 43
- OFFSET 252
- UCHAR 37

DATAFIELD 組み込み関数 424
DATE 組み込み関数 424
DATE 属性
- 説明 40

DATETIME 組み込み関数 424
DAYS 組み込み関数 424
DAYSTODATE 組み込み関数 425
DAYSTOMICROSECS 組み込み関数 426
DAYSTOSECS 組み込み関数 426
DBCS (2 バイト文字セット) 11
DBCS の継続規則 13
DCL (DECLARE) ステートメント
- 説明 148

DECIMAL (DEC) 組み込み関数 426
DECIMAL (DEC) 属性 23
DECLARE (DCL) ステートメント
- 説明 148

DEF (DEFINED) 属性 261
DEFAULT (DFT) ステートメント 166
DEFAULT 組み込み関数の NULLSTRPTR サブオプション
- ポインター式との併用 55

DEFAULT ステートメントの DESCRIPTORS オプション 167
DEFINE STRUCTURE (STRUCT) ステートメント 137
DEFINE ALIAS ステートメント 135
DEFINE ORDINAL ステートメント
- 説明 136
- options 136

DEFINED (DEF) 属性 261
DELAY ステートメント 206
DELETE ステートメント 289
DESCRIPTOR オプション 129
DETACH ステートメント 370
DFT (DEFAULT) ステートメント 166
digits
- および小数点の指定子 327
- 16 進数 3
- 2 進数 (binary) 3
- decimal 2

DIM (DIMENSION) 属性 171
DIMACROSS 属性 172
DIMACROSS 配列の上限、獲得 (HBOUNDACROSS) 442
DIMACROSS 配列の下限、獲得 (LBOUNDACROSS) 465
DIMENSION (DIM) 組み込み関数 427
DIMENSION (DIM) 属性 171
DIMENSION マクロ機能組み込み関数 592
DIRECT 属性 279
DISPLAY ステートメント 207
DIVIDE 組み込み関数 427
DLLEXTERNAL オプション 129
DLLINTERNAL オプション 129
DO グループ
- タイプ 3 の DO グループ 208, 211
- マクロ機能 207
- 例 215

DO ステートメント
- 説明 207
- 反復実行 208

DO ステートメントの BY オプション 209
DOWNTHRU オプション
- 序数での使用 217
- 説明 210
- タイプ 3 の DO 指定との併用 213
- 例 217


## E


E (EBCDIC) 文字定数
- 文字定数 34

E ピクチャー文字 336
E フォーマット項目 317
EDIT オプション 306
EDIT 組み込み関数 427
EMPTY 組み込み関数
- 区域変数の場合 255

END ステートメント
- 説明 218

ENDFILE 組み込み関数 428
ENDFILE 条件 353
ENDPAGE 条件 353
ENTRY ステートメント 94
ENTRY ステートメント、有効な OPTIONS オプション 124
ENTRY 属性

ENTRY 属性 (続き)
説明 112
有効な OPTIONS オプション 123
entry-constant
FETCH ステートメントとの併用 101
ENTRYADDR 疑似変数 429
ENTRYADDR 組み込み関数 429
ENV (ENVIRONMENT) 属性 280
ENVIRONMENT (ENV) 属性 280
ENVIRONMENT オプション 368
EPSILON 組み込み関数 429
ERF 組み込み関数 429
ERFC 組み込み関数 429
ERROR 条件
説明 354
プロシージャーの異常終了 98
EXE (ファイル拡張子) 87
EXIT ステートメント 98
EXP 組み込み関数 430
EXPONENT 組み込み関数 430
EXPORTS オプション 91
EXT (EXTERNAL) 属性 152
EXTERNAL (EXT) 属性
使用 101
説明 152

## F

F ピクチャー文字 337
F フォーマット項目 319
FETCH ステートメント
外部プロシージャーの動的ロード 100
制約事項 100
説明 101
FETCHABLE オプション 129
FILE オプション
ストリーム指向データ伝送の 288
説明 299
レコード単位データ伝送の 290
FILE 指定、OPEN ステートメントでの 281
FILE 属性 275
FILEDDINT 組み込み関数 430
FILEDDTEST 組み込み関数 431
FILEDDWORD 組み込み関数 431
FILEID 組み込み関数 432
FILENEW 組み込み関数 432
FILEOPEN 組み込み関数 433
FILEREAD 組み込み関数 433
FILESEEK 組み込み関数 433
FILETELL 組み込み関数 434
FILEWRITE 組み込み関数 434
FINISH 条件 355
FIRST タイプ付き関数 576
FIXED 組み込み関数 434
FIXED 属性
説明 23
FIXEDBIN 組み込み関数 435
FIXEDDEC 組み込み関数 435
FIXEDOVERFLOW (FOFL) 条件 355
FIXEDOVERFLOW 条件接頭語 340, 341
FLOAT 組み込み関数 436
FLOAT 属性 23
FLOATBIN 組み込み関数 436
FLOATDEC 組み込み関数 437
FLOOR 組み込み関数 437
FOFL (FIXEDOVERFLOW) 条件 355
FOLDEDFULLMATCH 組み込み関数 438
FOLDEDSIMPLEMATCH 組み込み関数 438
FORCE 属性 156
FORMAT ステートメント 309
FORMAT 属性
説明 44
変数タイプによる分類 21
FREE ステートメント
基底付き変数 249
被制御変数 240
IN オプション 249
FROM オプション、データ伝送ステートメントの 290
FROMALIEN オプション 130

## G

G フォーマット項目 320
GAMMA 組み込み関数 438
GENERIC 宣言の WHEN オプション 119
GENERIC 属性
説明 119
OTHERWISE オプションの使用法 119
GENERIC 属性の OTHERWISE オプション 119
GET STRING ステートメント 295
GET ステートメント
ストリング 308
データ・ディレクティブ 303
編集ディレクティブ 308
リスト・ディレクティブ 311
GETENV 組み込み関数 439
GETJCLSYMBOL 組み込み関数 439
GETSYSINT 組み込み関数 439
GETSYSWORD 組み込み関数 440
GO TO (GOTO) ステートメント
説明 220
GRAPHIC ENVIRONMENT オプション 36
GRAPHIC オプション 36
GRAPHIC 組み込み関数 441
GRAPHIC 属性 (G) 29
GRAPHIC の文字への変換 (CHARGRAPHIC) 417
GTCA 組み込み関数 441
GX (グラフィック 16 進) ストリング定数 36

## H

HANDLE 組み込み関数 442
HANDLE 組み込み関数のタイプ付き構造体 442
HANDLE 属性 139
HBOUND 組み込み関数 442
HBOUND マクロ機能組み込み関数 593
HBOUNDACROSS 組み込み関数 442
HEX 組み込み関数 443, 550
HEX8 組み込み関数 444
HEXADEC 属性 259
HEXDECODE 組み込み関数 445
HEXDECODE8 組み込み関数 445
HEXIMAGE 組み込み関数 446
HEXIMAGE8 組み込み関数 446
HIGH 組み込み関数 447
HUGE 組み込み関数 447

## I

I (オーバーパンチ) ピクチャー文字 334
IAND 組み込み関数 447
ICLZ 組み込み関数 447
IEEE 属性 259
IEOR 組み込み関数 448
IF ステートメント
    構文 221
    説明 221
IF ステートメントの ELSE 文節 221
IF ステートメントの THEN 文節 221
IFTHENELSE 組み込み関数 448
IGNORE オプション、データ伝送ステートメントの 291
IMAG 疑似変数 449
IMAG 組み込み関数 449
IN オプション
    ALLOCATE ステートメント 248
    FREE ステートメント 249
IN オプション、基底付き変数の FREE ステートメントに指定する 249
INARRAY 組み込み関数 449
INCLUDE ディレクティブ 223
INDEX 組み込み関数 450
INDEX マクロ機能組み込み関数 593
INDEXR 組み込み関数 450
INDFOR 属性 180
INDICATORS 組み込み関数 451
INITACROSS 268
INITIAL (INIT) 属性 265
INITIAL CALL 267
INITIAL TO 268
INLINE オプション 130
INLIST 組み込み関数 451
INOT 組み込み関数 451
input
    区域の 256
    条件
        ENDFILE 353
        ENDPAGE 353
        KEY 357
        NAME 357
        RECORD 358
        TRANSMIT 362
        UNDEFINEDFILE 363
    定義 273
    ディスカッション 273
INPUT 属性 278
INT (INTERNAL) 属性 152
INTERNAL (INT) 属性 152
INTO オプション、データ伝送ステートメントの 291
INVALIDOP 条件 356
INVALIDOP 条件接頭語 340, 341
IOR 組み込み関数 452
IRREDUCIBLE (IRRED) オプション 131
ISFINITE 組み込み関数 452
ISIGNED 組み込み関数 452
ISINF 組み込み関数 453
ISJCLSYMBOL 組み込み関数 453
ISLL 組み込み関数 453
ISMAIN 組み込み関数 454
ISNAN 組み込み関数 454
ISNORMAL 組み込み関数 454
ISRL 組み込み関数 454
iSUB
    定義 261, 263
    非連結 262
ISZERO 組み込み関数 455
ITERATE ステートメント 223
IUNSIGNED 組み込み関数 455

## J

JSON 組み込み関数
    要約 385
    JSONGETARRAYEND 455
    JSONGETARRAYSTART 456
    JSONGETCOLON 456
    JSONGETCOMMA 456
    JSONGETMEMBER 457
    JSONGETOBJECTEND 458
    JSONGETOBJECTSTART 458
    JSONGETVALUE 459
    JSONPUTARRAYEND 461
    JSONPUTARRAYSTART 461
    JSONPUTCOLON 461
    JSONPUTCOMMA 461
    JSONPUTMEMBER 462
    JSONPUTOBJECTEND 463
    JSONPUTOBJECTSTART 463
    JSONPUTVALUE 464
    JSONVALID 465
JSONGETARRAYEND 組み込み関数 455
JSONGETARRAYSTART 組み込み関数 456
JSONGETCOLON 組み込み関数 456
JSONGETCOMMA 組み込み関数 456
JSONGETMEMBER 組み込み関数 457
JSONGETOBJECTEND 組み込み関数 458
JSONGETOBJECTSTART 組み込み関数 458
JSONGETVALUE 組み込み関数 459
JSONNAME 属性 183
JSONOMIT 属性 183
JSONPUTARRAYEND 組み込み関数 461
JSONPUTARRAYSTART 組み込み関数 461
JSONPUTCOLON 組み込み関数 461
JSONPUTCOMMA 組み込み関数 461
JSONPUTMEMBER 組み込み関数 462
JSONPUTOBJECTEND 組み込み関数 463
JSONPUTOBJECTSTART 組み込み関数 463
JSONPUTVALUE 組み込み関数 464
JSONTRIMR 属性 183
JSONVALID 組み込み関数 465
JULIANTOSMF 組み込み関数 465

## K

K ピクチャー文字 336
KEY オプション、データ伝送ステートメントの 291
KEY 条件 357
KEYED 属性 280
KEYFROM オプション、データ伝送ステートメントの 291
KEYTO オプション、データ伝送ステートメントの 292

## L

L フォーマット項目 321
label 9

LABEL 属性
    説明 43
LAST タイプ付き関数 577
LBOUND 組み込み関数 465
LBOUND マクロ機能組み込み関数 593
LBOUNDACROSS 組み込み関数 465
LEAVE ステートメント 224
LEFT 組み込み関数 466
length
    単純パラメーター (simple parameter) 95
    被制御パラメーター (controlled parameter) 95
LENGTH 組み込み関数 466
LENGTH マクロ機能組み込み関数 594
LIKE 属性 178
LIMITED 属性
    説明 118
    例 118
LINE オプション 299
LINE ディレクティブ 225
LINE フォーマット項目 321
LINENO 組み込み関数 466
LINESIZE 指定、OPEN ステートメントでの 282
LINKAGE オプション 130
list
    処理 256
    双方向 257
    単一方向 257
    チェーン 256
    パラメーター記述子 (parameter descriptor) 113
LIST 属性
    説明 116
LITTLEENDIAN 属性 258
LOCATE ステートメント 289
LOCATION (LOC) 組み込み関数 467
LOCNEWSPACE 組み込みサブルーチン 467
LOCNEWVALUE 組み込みサブルーチン 468
LOCSTG 組み込み関数 469
LOCVAL 組み込み関数 469
LOG 組み込み関数 470
LOG10 組み込み関数 470
LOG2 組み込み関数 470
LOGGAMMA 組み込み関数 470
LOW 組み込み関数 471
LOWER2 組み込み関数 472
LOWERASCII 組み込み関数 471
LOWERCASE 組み込み関数 471
LOWERLATIN1 組み込み関数 472

## M

M
    FIXED BINARY、最大精度 74, 375
    LIMITS(FIXEDBIN(M1,M2)) 74, 375
M (混合) ストリング定数 37
MACCOL マクロ機能組み込み関数 594
MACLMAR マクロ機能組み込み関数 595
MACRMAR マクロ機能組み込み関数 595
MAIN オプション 130
MARGINS キーワード
    ANSWER プリプロセッサー・ステートメントの 588
MAX 組み込み関数 472
MAXDATE 組み込み関数 473
MAXEXP 組み込み関数 473
MAXLENGTH 組み込み関数 475
MAXVAL 組み込み関数 474
MEMCOLLAPSE 組み込み関数 476
MEMCONVERT 組み込み関数 475
MEMCU12 組み込み関数 476
MEMCU14 組み込み関数 477
MEMCU21 組み込み関数 477
MEMCU24 組み込み関数 478
MEMCU41 組み込み関数 478
MEMCU42 組み込み関数 478
MEMINDEX 組み込み関数 479
MEMREPLACE 組み込み関数 480
MEMSEARCH 組み込み関数 481
MEMSEARCHR 組み込み関数 482
MEMSQUEEZE 組み込み関数 482
MEMVERIFY 組み込み関数 483
MEMVERIFYR 組み込み関数 484
MICROSECS 組み込み関数 484
MICROSECSTODATE 組み込み関数 485
MICROSECSTODAYS 組み込み関数 485
MIN 組み込み関数 486
MINDATE 組み込み関数 486
MINEXP 組み込み関数 486
MINVAL 組み込み関数 487
MOD 組み込み関数 487
MPSTR 組み込み関数 488
MULTIPLY 組み込み関数 489

## N

N
    FIXED DECIMAL、最大精度 74, 375
    LIMITS(FIXEDDEC(N1,N2)) 74, 375
NAME 条件 357
NEW タイプ付き関数 577
NOCHARGRAPHIC オプション 128
NODESCRIPTOR オプション 129
NOEXECOPS オプション 131
NOINIT 属性 181
NOINLINE オプション 130
NOMAP オプション 131
NONASSIGNABLE 属性 257
NONCONNECTED (NONCONN) 属性 260
NONVARYING (NONVAR) 属性 31
NOPRINT ディレクティブ 225
NORESCAN オプション 601
NORMAL 属性 258
NOT 演算子 63
NOTE ディレクティブ 225
NULL 組み込み関数 489
NULLENTRY 組み込み関数 490
NULLINIT 属性 181

## O

OFFSET 組み込み関数 490
OFFSET 属性 252
OFFSETADD 組み込み関数 490
OFFSETDIFF 組み込み関数 490
OFFSETSUBTRACT 組み込み関数 491
OFFSETVALUE 組み込み関数 491
OFL (OVERFLOW) 条件 358
OMITTED 組み込み関数 491
ON ステートメント 341

ON ユニット
動的に派生した 343
ファイル変数の 343
有効範囲 343
null 342
ONACTUAL 組み込み関数 491
ONAREA 組み込み関数 491
ONCHAR 疑似変数 492
ONCHAR 組み込み関数 492
ONCODE 組み込み関数
使用 339
ONCONDCOND 組み込み関数 492
ONCONDID 組み込み関数 493
ONCOUNT 組み込み関数 493
ONEXPECTED 組み込み関数 494
ONFILE 組み込み関数 494
ONGSOURCE 疑似変数 494
ONGSOURCE 組み込み関数 494
ONHBOUND 組み込み関数 495
ONKEY 組み込み関数 495
ONLBOUND 組み込み関数 495
ONLINE 組み込み関数 496
ONLOC 組み込み関数 496
ONOFFSET 組み込み関数 496
ONOPERATOR 組み込み関数 496
ONPACKAGE 組み込み関数 497
ONPROCEDURE 組み込み関数 497
ONSOURCE 疑似変数 497
ONSOURCE 組み込み関数 497
ONSUBCODE 組み込み関数 498
ONSUBCODE2 組み込み関数 498
ONSUBSCRIPT 組み込み関数 498
ONTEXT 組み込み関数 498
ONUCHAR 疑似変数 499
ONUCHAR 組み込み関数 499
ONUSOURCE 疑似変数 499
ONUSOURCE 組み込み関数 499
ONWCHAR 疑似変数 500
ONWCHAR 組み込み関数 500
ONWSOURCE 疑似変数 500
ONWSOURCE 組み込み関数 500
OPEN ステートメント 281
OPTIMIZATION、条件の発生 341
OPTIONAL 属性 114
options
アセンブラー 127
値の指定 168
データ伝送ステートメントの 290, 296
DESCRIPTORS オプション 167
DLLEXTERNAL 129
DLLINTERNAL 129
EXPORTS 91
FETCHABLE 129
GRAPHIC 36
GRAPHIC ENVIRONMENT 36
NORESCAN 601
OPTIONS 123
RANGE 166
RECURSIVE 99
REPEAT 210
REPLY 207
RESCAN 600
RESERVES 91
RETURNS 132
options (続き)
SCAN 600
SET 101
SNAP 342
SYSTEM 342
TITLE 101
OPTIONS オプション
アセンブラー 127
構文 123
説明 123
特性リスト 123
AMODE31 または AMODE64 127
BEGIN ステートメント 123
BYADDR 127
BYVALUE 127
CHARGRAPHIC 128
CMPAT 129
COBOL 129
DESCRIPTOR 129
ENTRY 宣言 123
FROMALIEN 130
INLINE 130
IRREDUCIBLE 131
LINKAGE 130
MAIN 130
NOCHARGRAPHIC 128
NODESCRIPTOR 129
NOEXECOPS 131
NOINLINE 130
NOMAP 131
NORETURN 131
ORDER 131
PROCEDURE ステートメント 126
RECURSIVE 99
REDUCIBLE 131
REENTRANT 132
REORDER 131
RETCODE 132
RETURN ステートメント 131
WINMAIN 132
OPTIONS オプション、ENTRY ステートメント 124
OPTIONS 属性 123
ORDER
属性 93, 110
ORDER オプション 131
ORDINAL 属性 141
ORDINALNAME 組み込み関数 501
ORDINALPRED 組み込み関数 501
ORDINALSUCC 組み込み関数 501
OTHERWISE ステートメント
SELECT ステートメント内の 231
OUTPUT 属性 278
OVERFLOW (OFL) 条件 358
OVERFLOW 条件接頭語 340, 341

## P

P フォーマット項目 321
PACKAGE ステートメント
説明 90
有効な OPTIONS オプション 125
例 91
PACKAGENAME 組み込み関数 501
PAGE オプション 299

PAGE キーワード、ANSWER ステートメントの 588
PAGE ディレクティブ 227
PAGE フォーマット項目 322
PAGENO 組み込み関数 502
PAGESIZE 指定、OPEN ステートメントでの 282
parameters
  エレメント 108
  属性 95
  配列引数
    例 96
  引数 106
PARMSET マクロ機能組み込み関数 595
PICSPEC 組み込み関数 502
PICTURE (PIC) 属性 32
PL/I アプリケーション
  構造の図解 87
  説明 87
PLACES 組み込み関数 502
PLIASCII 組み込みサブルーチン 503
PLIATTN 組み込みサブルーチン 503
PLICANC 組み込みサブルーチン 504
PLICKPT 組み込みサブルーチン 504
PLIDELETE 組み込みサブルーチン 504
PLIDUMP 組み込みサブルーチン 504
PLIEBCDIC 組み込みサブルーチン 504
PLIFILL 組み込みサブルーチン 505
PLIFREE 組み込みサブルーチン
  基底付き変数の場合 247
PLIMOVE 組み込みサブルーチン 505
PLIOVER 組み込みサブルーチン 506
PLIPARSE 組み込みサブルーチン 506
PLIREST 組み込みサブルーチン 508
PLIRETC 組み込みサブルーチン 508
PLIRETV 組み込み関数 508
PLISAXA 組み込みサブルーチン 509
PLISAXB 組み込みサブルーチン 509
PLISAXC 組み込みサブルーチン 509
PLISAXD 組み込みサブルーチン 510
PLISRTA 組み込みサブルーチン 510
PLISRTB 組み込みサブルーチン 511
PLISRTC 組み込みサブルーチン 511
PLISRTD 組み込みサブルーチン 511
PLISTCK 組み込み関数 511
PLISTCKE 組み込み関数 512
PLISTCKELOCAL 組み込み関数 512
PLISTCKEUTC 組み込み関数 512
PLISTCKF 組み込み関数 512
PLISTCKLOCAL 組み込み関数 512
PLISTCKP 組み込み関数 513
PLISTCKPLOCAL 組み込み関数 513
PLISTCKPUCT 組み込み関数 513
PLISTCKUTC 組み込み関数 513
PLITRAN11 組み込み関数 514
PLITRAN12 組み込み関数 514
PLITRAN21 組み込み関数 515
PLITRAN22 組み込み関数 515
POINTER (PTR) 組み込み関数 516
POINTER (PTR) 属性 247
POINTERADD (PTRADD) 組み込み関数
  ポインター演算との併用 55
POINTERDIFF (PTRDIFF) 組み込み関数 517
POINTERSUBTRACT (PTRSUBTRACT) 組み込み関数 517
POINTERVALUE (PTRVALUE) 組み込み関数
  使用 56
POLY 組み込み関数 517
POP ディレクティブ 227
POPCNT 組み込み関数 518
POS (POSITION) 属性 261
POSITION (POS) 属性 261
POSITION 属性 265
PRECISION (PREC) 組み込み関数 518
PRECISION 組み込み関数
  使用 62
PRECISION 属性
  序数 137
  説明 23
PRECVAL 組み込み関数 518
PRED 組み込み関数 519
PRESENT 組み込み関数 519
PRINT 属性 312
PRINT ディレクティブ 227
PROC (PROCEDURE) ステートメント 93
PROCEDURE (PROC) ステートメント
  使用 88
  説明 93
  有効な OPTIONS 125
PROCEDURE ステートメント 126
PROCEDURENAME (PROCNAME) 組み込み関数 519
PROCESS ディレクティブ 227, 228
PROCINC ディレクティブ 228
PROD 組み込み関数 519
program
  エレメント
    入り口値 (entry value) 121
    入り口データ (entry data) 110
    入り口の呼び出し 121
    開始ブロック 109
    関数 104
    組み込み関数 106
    サブルーチン 102
    説明 1
    OPTIONS オプション 123
    RETURN ステートメント 122
  活動化 88
  構造 87
  サブルーチン
    定義 102
  終了 88
  定義 (PL/I の場合の) 87
  パッケージ 90
  プロシージャー 92
  ブロック
    説明 88
  編成 87
  RETURN 122
PTR (POINTER) 属性 247
PTRADD (POINTERADD) 組み込み関数
  ポインター演算との併用 55
PTRVALUE (POINTERVALUE) 組み込み関数
  使用 56
PUSH ディレクティブ 228
PUT ステートメント
  ストリング 308
  データ・ディレクティブ 305
  編集ディレクティブ 308
  リスト・ディレクティブ 312
  STREAM 出力 296
PUTENV 組み込み関数 520

## Q


QUALIFY ステートメント
    説明 229
QUICKSORT 組み込み関数 520
QUICKSORTX 組み込み関数 520
QUOTE マクロ機能組み込み関数 596


## R


R (オーバーパンチ) ピクチャー文字 334
R フォーマット項目 322
RADIX 組み込み関数 521
RAISE2 組み込み関数 521
RANDOM 組み込み関数 522
RANGE オプション 166
RANK 組み込み関数 522
READ ステートメント 288
REAL 疑似変数 522
REAL 組み込み関数 522
REAL 属性 24
RECORD 条件 358
RECORD 属性 278
RECURSIVE オプション 99
RECURSIVE 属性 99
REDUCIBLE (RED) オプション 131
REENTRANT オプション 132
REFER オプション
    説明 249
    AREA 属性での 251
REGEX 組み込み関数 523
REINIT ステートメント 230
RELEASE ステートメント
    外部プロシージャーの動的ロード 100
    制約事項 100
    説明 101
    例 102
REM 組み込み関数 525
REORDER オプション 131
REPATTERN 組み込み関数 525
REPEAT オプション 210
REPEAT 組み込み関数 526
REPEAT マクロ機能組み込み関数 596
REPLACE 組み込み関数 526
REPLY オプション 207
RESCAN オプション 600
RESERVED 属性 155
RESERVES オプション 91
RESIGNAL ステートメント 345
RESPEC タイプ付き関数 577
RETCODE オプション 132
RETURN ステートメント
    関数からの戻り 122
    サブルーチンとの併用 122
    使用 98
    説明 122
    プリプロセッサー・プロシージャーでの使用 587
RETURNS オプション
    説明 132
RETURNS 属性 132
REVERSE 組み込み関数 527
REVERT ステートメント 344
REWRITE ステートメント
    説明 289
RIGHT 組み込み関数 528
ROUND 組み込み関数 528
ROUNDAWAYFROMZERO 組み込み関数 530
ROUNDTOEVEN 組み込み関数 530


## S


S ピクチャー文字 332
SAMEKEY 組み込み関数 531
SCALARVARYING オプション 287
SCALE 組み込み関数 532
SCALEVAL 組み込み関数 532
SCAN オプション 600
SCRUBOUT 組み込み関数 531
SEARCH 組み込み関数 532
SEARCHR 組み込み関数 533
SECS 組み込み関数 534
SECSTODATE 組み込み関数 535
SECSTODAYS 組み込み関数 535
SELECT グループ 231
SELECT ステートメント
    説明 231
    例 232
SEQL (SEQUENTIAL) 属性 279
SEQUENTIAL (SEQL) 属性 279
SET オプション
    説明 248
    ポインター参照の指定 101
    ALLOCATE ステートメントの使用法 248
    LOCATE ステートメントの使用法 289
    READ ステートメントの使用法 288
SIGN 組み込み関数 535
SIGNAL ステートメント 345
SIGNED 組み込み関数 536
SIGNED 属性
    序数 137
    説明 24
    データ・ストレージ所要量 25
SIN 組み込み関数 536
SIND 組み込み関数 536
SINH 組み込み関数 536
SIZE 組み込み関数 537
SIZE 条件 359
SIZE 条件接頭語 340, 341
SIZE タイプ付き関数 578
size_t 373
SKIP オプション 299
SKIP キーワード、ANSWER ステートメントの 588
SKIP ディレクティブ 232
SKIP フォーマット項目 322
SMFTOJULIAN 組み込み関数 538
SNAP オプション、ON ステートメントの 342
SOURCEFILE 組み込み関数 538
SOURCELINE 組み込み関数 538
SQRT 組み込み関数 538
SQRTF 組み込み関数 538
SQUEEZE 組み込み関数 539
STACKADDR 組み込み関数 539
STATEMENT オプション 586
statements
    キーワード 10
    形式 309
    取得
        データ・ディレクティブ 303

statements (続き)
取得 (続き)
編集ディレクティブ 308
リスト・ディレクティブ 311
STREAM 入力 295
代入 (assignment) 10, 197
単純 9
ディスカッション 193
複合 10
ALLOCATE 248
ANSWER
プリプロセッサー・プロシージャーでの使用 587
ASSERT 193
ATTACH 368
BEGIN 109
CALL 121
CLOSE 284
DECLARE 148
DEFAULT 166
DEFINE ALIAS 135
DEFINE ORDINAL 136
DEFINE STRUCTURE 137
DELAY 206
DELETE 289
DETACH 370
DISPLAY 207
DO 207
END 218
ENTRY 94
EXIT 98
FETCH 100
FREE 240, 249
GET STRING 295
GO TO 220
group 10
IF 221
ITERATE 223
LEAVE 224
LOCATE 289
null 226
ON 341
OPEN 281
PACKAGE 90
PROCEDURE
主プロシージャーを呼び出すための使用 88
説明 93
PUT
データ・ディレクティブ 305
編集ディレクティブ 308
リスト・ディレクティブ 312
STREAM 出力 296
QUALIFY 229
READ 288
REINIT 230
RELEASE
制約事項 100
説明 101
動的にロードする外部 100
例 102
RESIGNAL 345
RETURN
関数からの戻り 122
構文 122
サブルーチンとの併用 122
statements (続き)
RETURN (続き)
使用 98
説明 122
プリプロセッサー・プロシージャーでの使用 587
REVERT 344
REWRITE
説明 289
SELECT
説明 231
例 232
SIGNAL 345
STOP
使用 98
WAIT 369
WRITE
説明 288
XDEFINE ALIAS 233
XDEFINE ORDINAL 234
%PROCEDURE 586
STATIC 属性
説明 236
INITIAL 属性との併用 270
STCKETODATE 組み込み関数 539
STCKTODATE 組み込み関数 540
STOP ステートメント
使用 98
storage
基底付き 242
自動 237
接続された 260
被制御 238
非連結 174
分類 235
割り振り 235
CONTROL 235
static 236
STORAGE 組み込み関数 540
STORAGE 条件 360
STREAM 属性 278
STRG (STRINGRANGE) 条件 361
STRING オプション
説明 300
GET ステートメントでの使用 295
PUT ステートメントでの使用 296
STRING 疑似変数 541
STRING 組み込み関数 540
STRINGRANGE (STRG) 条件 261, 361
STRINGRANGE 条件接頭語 340, 341
STRINGSIZE (STRZ) 条件 261, 361
STRINGSIZE 条件接頭語 340, 341
STRUCT (DEFINE STRUCTURE) ステートメント 137
STRZ (STRINGSIZE) 条件 361
SUBRG (SUBSCRIPTRANGE) 条件 261, 362
SUBSCRIPTRANGE (SUBRG) 条件 261, 362
SUBSCRIPTRANGE 条件接頭語 340, 341
SUBSTR 疑似変数 542
SUBSTR 組み込み関数 541
SUBSTR マクロ機能組み込み関数 596
SUBTO 疑似変数 543
SUBTO 組み込み関数 542
SUBTRACT 組み込み関数 543
SUCC 組み込み関数 543
SUM 組み込み関数 544

SUPPRESS 属性 157
SYSDIMSIZE マクロ機能組み込み関数 597
sysin 285
SYSNULL 組み込み関数 544
SYSOFFSETSIZE マクロ機能組み込み関数 597
SYSPARM マクロ機能組み込み関数 597
SYSPOINTERSIZE マクロ機能組み込み関数 598
SYSPRINT 285
SYSTEM オプション、ON ステートメントの 342
SYSTEM 組み込み関数 544
SYSTEM マクロ機能組み込み関数 598
SYSVERSION マクロ機能組み込み関数 598

## T

T (オーバーパンチ) ピクチャー文字 334
TALLY 組み込み関数 545
TAN 組み込み関数 545
TAND 組み込み関数 545
TANH 組み込み関数 545
targets
    疑似変数
        説明 53
    構造 200
    説明 53
    ターゲット変数の要件 199
    中間結果 54
    配列 200
    非計算 199
    変数 53
TASK 属性 370
THREAD オプション 368
THREADID 組み込み関数 545
TIME 組み込み関数 546
TIMESTAMP 組み込み関数 546
TINY 組み込み関数 546
TITLE オプション 101
TITLE 指定、OPEN ステートメントでの 282
TO オプション 209
TO オプション、INITIAL 属性の 268
TRANSLATE 組み込み関数 546
TRANSLATE マクロ機能組み込み関数 598
TRANSMIT 条件 362
TRIM 組み込み関数 547
TRUNC 組み込み関数 548
TSTACK オプション 368
TYPE 疑似変数 548
TYPE 組み込み関数 548
TYPE 属性 140

## U

UCHAR 413, 438
UCHAR VARYING 413
UCHAR ストリング定数 38
UCHAR 属性
    説明 29
uchar 定数
    比較演算 65
uchar データ
    変換 85
UCHAR データ
    UX (UCHAR 16 進) ストリング定数 38
UFL (UNDERFLOW) 条件 364
UHIGH 組み込み関数 549
ULENGTH 組み込み関数 549
ULENGTH16 組み込み関数 549
ULENGTH8 組み込み関数 549
ULOW 組み込み関数 550
UNALIGNED attribute
    example 164
UNALIGNED 属性
    記憶位置合わせ要件 158
    構造体マッピングへの影響 186
    説明および構文 158
UNALLOCATED 組み込み関数 550
UNBUF (UNBUFFERED) 属性 279
UNBUFFERED (UNBUF) 属性 279
UNDEFINEDFILE (UNDF) 条件 363
UNDERFLOW (UFL) 条件 364
UNDERFLOW 条件接頭語 340, 341
UNDF (UNDEFINEDFILE) 条件 363
UNION 属性 176
UNION、同義語 176
UNSIGNED 組み込み関数 551
UNSIGNED 属性
    序数 137
    説明 24
    データ・ストレージ所要量 25
UNSPEC 疑似変数 553
UNSPEC 組み込み関数 551
UNTIL オプション
    説明 209
    タイプ 2 の DO 指定との併用 210
UPDATE 属性 278
UPOS 組み込み関数 554
UPPERASCII 組み込み関数 554
UPPERCASE 組み込み関数 554
UPPERLATIN1 組み込み関数 555
UPTHRU オプション
    説明 209
    タイプ 3 の DO 指定との併用 213
    例 217
UPTHRU、序数との併用 217
USUBSTR 組み込み関数 555
USUPPLEMENTARY 組み込み関数 556
UTCDATETIME 組み込み関数 556
UTCMICROSECS 組み込み関数 556
UTCSECS 組み込み関数 557
UTF 処理組み込み関数
    LOWERASCII 471
    LOWERLATIN1 472
    ONUCHAR 499
    UHIGH 549
    ULENGTH 549
    ULENGTH16 549
    ULENGTH8 549
    ULOW 550
    UPOS 554
    UPPERASCII 554
    UPPERLATIN1 555
    USUBSTR 555
    USUPPLEMENTARY 556
    UTF8 557
    UTF8TOCHAR 558
    UTF8TOWCHAR 558
    UVALID 559

UTF 処理組み込み関数 (続き)
UWIDTH 560
UTF8 組み込み関数 557
UTF8STG 組み込み関数 558
UTF8TOCHAR 組み込み関数 558
UTF8TOWCHAR 組み込み関数 558
UUID 組み込み関数 558
UUID4 組み込み関数 559
UVALID 組み込み関数 559
UWIDTH 組み込み関数 560
UX (UCHAR 16 進) ストリング定数 38

## V

V ピクチャー指定文字 327
V フォーマット項目 323
VALID 組み込み関数 560
VALIDDATE 組み込み関数 561
VALIDVALUE 組み込み関数 562
VALUE 45
VALUE オプション 166, 168
VALUE 型付き関数 578
VALUE 属性
序数 137
説明 45
VALUELIST 属性 47
VALUELIST 属性の構文 47
VALUELISTFROM 属性 47
VALUERANGE 属性 48
VALUERANGE 属性の構文 48
VARGLIST 組み込み関数 562
VARGSIZE 組み込み関数 562
VARIABLE 属性 45
VARYING (VAR) 属性 31
VARYING4 属性 31
VARYINGZ (VARZ) 属性 31
VERIFY 組み込み関数 563
VERIFY マクロ機能組み込み関数 599
VERIFYR 組み込み関数 564

## W

WAIT ステートメント 369
WCHARVAL 組み込み関数 564
WEEKDAY 組み込み関数 564
WHEN ステートメント
説明 231
WHEREDIFF 組み込み関数 565
WHIGH 組み込み関数 565
WHILE オプション
説明 208
タイプ 2 の DO 指定との併用 210
WIDECHAR (WCHAR) 組み込み関数 565
WIDECHAR (WCHAR) 属性
説明 29
WIDEPIC 属性 32
WINMAIN オプション 132
WLOW 組み込み関数 566
WRITE ステートメント
説明 288
WSCOLLAPSE 組み込み関数 566
WSCOLLAPSE16 組み込み関数 567
WSREPLACE 組み込み関数 567
WSREPLACE16 組み込み関数 568
WX (ワイド文字 16 進) ストリング定数 38

## X

X (16 進) の文字ストリング定数 34
X ピクチャー指定文字 325
X フォーマット項目 323
XDEFINE ALIAS ステートメント 233
XDEFINE ORDINAL ステートメント 234
XMLATTR 属性 181, 182
XMLCHAR 組み込み関数 568
XMLCONTENT 属性 182
XMLNAME 属性 182
XMLOMIT 属性 182
XMLSCRUB16 組み込み関数 570
XMLUCHAR 組み込み関数 571
XN (2 進 16 進数) 定数 26
XU (2 進 16 進数) 定数 26

## Y

Y ゼロ置換ピクチャー文字 334
Y4DATE 組み込み関数 571
Y4JULIAN 組み込み関数 572
Y4YEAR 組み込み関数 572

## Z

Z ゼロ消去ピクチャー文字 328
ZDIV (ZERODIVIDE) 条件 365
ZERODIVIDE (ZDIV) 条件 365
ZERODIVIDE 条件接頭語 340, 341

## [特殊文字]

_ (下線、区切り)、ASCII および EBCDIC 値 3
- (減算)
演算子としての使用 7
算術演算での使用 56
ASCII および EBCDIC 値 3
-= (減算と代入)、複合シンボルの作成 4
-> (locator)
区切り文字としての使用 6
ロケーター修飾 245
-> (ロケーター)、複合シンボルの作成 4
, (分離文字)
区切り文字としての使用 6
ASCII および EBCDIC 値 3
; (ステートメント終了文字)
区切り文字としての使用 6
ASCII および EBCDIC 値 3
: (接頭部、次元、および範囲区切り記号)
使用 6
ASCII および EBCDIC 値 3
? (マクロ・トリガー文字)
ASCII および EBCDIC 値 3
. (名前修飾子、小数点)
区切り文字としての使用 6
ASCII および EBCDIC 値 3
' (引用符)
二重 3
’’ (囲み定数)

' ' (囲み定数) (続き)
ASCII および EBCDIC 値 3
" 二重引用符
ASCII および EBCDIC 値 3
( ) (囲み記号)
区切り文字としての使用 6
ASCII および EBCDIC 値 3
* (乗算)
演算子としての使用 7
算術演算での使用 56
ASCII および EBCDIC 値 3
* ゼロ消去ピクチャー文字 328
** (累乗演算)
演算子としての使用 7
算術演算での使用 56
複合記号の作成 4
**= (累乗および代入)、複合シンボルの作成 4
*/ (コメントの終わり)、複合シンボルの作成 4
*= (乗算および代入)、複合シンボルの作成 4
*PROCESS ディレクティブ 228
*PROCINC ディレクティブ 228
/ (除算)
演算子としての使用 7
算術演算での使用 56
ASCII および EBCDIC 値 3
/ (挿入文字) 329
/* (コメントの開始)、複合記号の作成 4
/* */ (コメント)
区切り文字としての使用 6
構文 8
/= (除算および代入)、複合シンボルの作成 4
& (AND 記号)
演算子としての使用 7
ASCII および EBCDIC 値 3
& (ビット演算子: AND) 63
&= (論理積および代入)、複合シンボルの作成 4
% (% ステートメント用)
区切り文字としての使用 6
ASCII および EBCDIC 値 3
% ディレクティブ
印刷 (%P) 227
ページ (%P) 227
%INCLUDE 223
%LINE 225
%NOPRINT 225
%NOTE 225
%POP 227
%PROCESS 227
%PROCINC 228
%PUSH 228
%SKIP 232
%ACTIVATE ステートメント 600
%assignment ステートメント 601
%CALL ステートメント 589
%DEACTIVATE ステートメント 601
%DECLARE ステートメント 601
%DO ステートメント 603
%END ステートメント 604
%GO TO ステートメント 604
%IF ステートメント 604
%IF ステートメントの ELSE 節 604
%IF ステートメントの THEN 文節 604
%INCLUDE ステートメント 605
%INCLUDE ディレクティブ 223
%INSCAN ステートメント 606
%ITERATE ステートメント 606
%LEAVE ステートメント 607
%LINE ディレクティブ 225
%NOPRINT ディレクティブ 225
%NOTE ステートメント 607
%NOTE ディレクティブ 225
%null ステートメント 608
%PAGE ディレクティブ 227
%POP ディレクティブ 227
%PRINT ディレクティブ 227
%PROCEDURE ステートメント 586
%PROCESS ディレクティブ 227
%PROCINC ディレクティブ 228
%PUSH ディレクティブ 228
%REPLACE ステートメント 608
%SELECT ステートメント 608
%SKIP ディレクティブ 232
%XINCLUDE ステートメント 609
%XINSCAN ステートメント 609
+ (加算)
演算子としての使用 7
算術演算での使用 56
ASCII および EBCDIC 値 3
+ (ピクチャー文字) 332
+= (加算と代入)、複合記号の作成 4
< (より小記号)
演算子としての使用 7
比較演算での使用 65
ASCII および EBCDIC 値 3
<= (「より小さいか等しい」記号)
演算子としての使用 7
比較演算での使用 65
= (等号)
演算子としての使用 7
区切り文字としての使用 6
比較演算での使用 65
ASCII および EBCDIC 値 3
> (より大記号)
演算子としての使用 7
ASCII および EBCDIC 値 3
>= (「以上」記号) 4
¬ (NOT 記号)
ASCII および EBCDIC 値 3
¬ (ビット演算子: NOT、XOR) 63
¬ (論理 NOT EOR 記号)
演算子としての使用 7
ASCII および EBCDIC 値 3
¬< (より小ではない記号)
演算子としての使用 7
説明 4
比較演算での使用 65
¬= (等しくない記号)
演算子としての使用 7
¬= または <> (「等しくない」記号)
説明 4
比較演算での使用 65
¬> (「以下」記号)
演算子としての使用 7
説明 4
比較演算での使用 65
| (ビット演算子: OR) 63
| (論理 OR 記号)
演算子としての使用 7

| (論理 OR 記号) (続き)
ASCII および EBCDIC 値 3
|= (論理和および代入)、複合シンボルの作成 4
|| (連結)
演算子としての使用 7
複合記号の作成 4
連結演算子での使用 66
||= (連結および代入)、複合シンボルの作成 4
$ (ピクチャー文字) 332

プログラム番号: 5655-PL6

SC31-5716-00