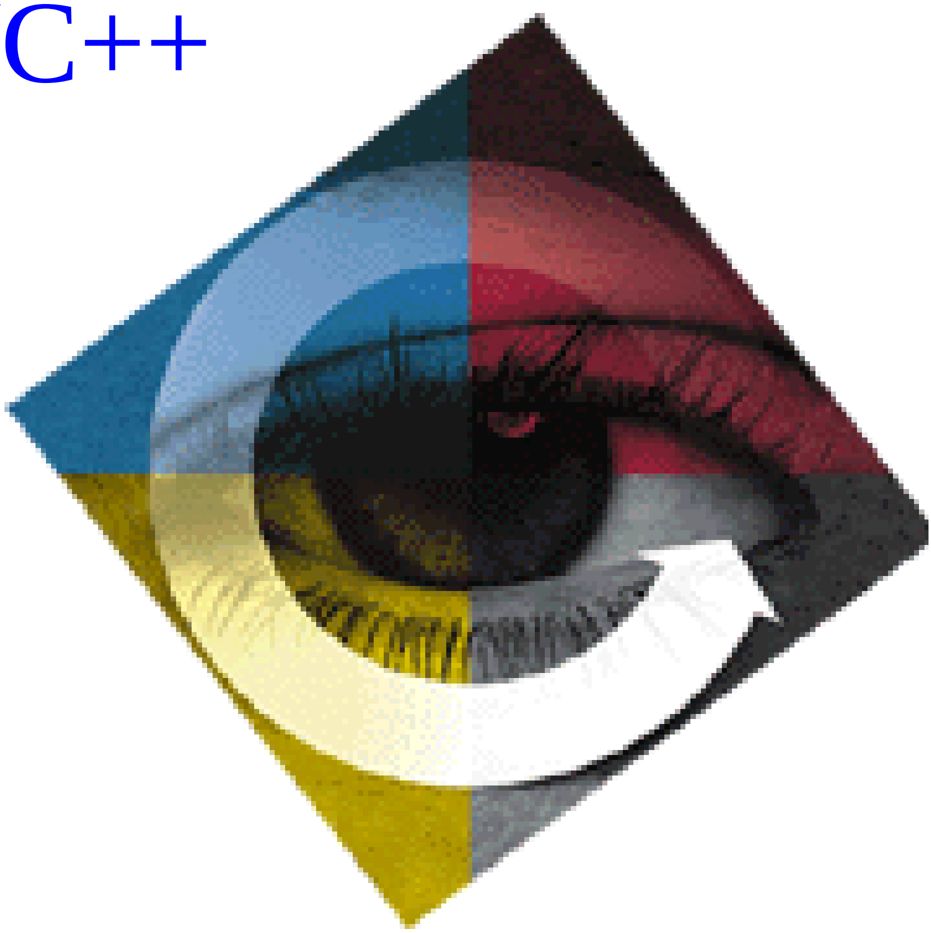
What is ILC and how to use it

COBOL, PL/I and C/C++



Tom Ross

SHARE Session: 8239

August, 2003

IBM Software

What is ILC: topics

- Introduction
- HLL parameter passing basics
- #pragma-less ILC for C/C++
- Optional parameters in 3 languages
- Language Environment ILC enhancements
  - Performance and CICS improvements for COBOL
- COBOL-PL/I ILC Migration

Introduction, what is ILC?

- **Interlanguage communication under Language Environment**
  - COBOL, PL/I and C/C++ are covered
  - Assembler is not a language in terms of ILC
  - Java not covered, it is a special case
    - Pure OO, no programs, no CALL statements, interoperability through CLASS inheritance
    - Java only runs in Unix Systems services, BPXPATCH, or special support of CICS and IMS
    - See Sessions 8246, 1:30 Thurs, OS/390 Java and COBOL,
      and 8247, 3:00 Thurs, OS/390 Java and PL/I
- **This presentation focuses on direct program CALLs**
  - Different rules, fewer restrictions under CICS with
    EXEC CICS LINK/XCTL
  - Same rules for CALLs under CICS

Introduction, why ILC?

- **When a subroutine you need is written in another language**
- **When programmers on a team want to use different programming languages**
- **To be avoided for retaining sanity**
  - Using one language is easiest to maintain
  - Often rewriting a subroutine to maintain single language is easier in long term
- **Sometimes ILC is unavoidable**
  - No choice to rewrite mandated subroutines
  - Calling TCP/IP and other services
  - Other vendor products

Introduction: CALL linkages

- **Static CALL means programs statically bound in same module**
- **Dynamic COBOL calls equivalent to FETCH in PL/I and C/C++**
- **DLL linkage: common facility for linkage between separately linked parts**
  - Many compilers now support DLL linkage:
    - Enterprise COBOL Version 3
    - Enterprise PL/I version 3,
    - COBOL for OS/390 & VM
    - C/C++ for OS/390 and z/OS
  - See session 8130, Wed 9:30 PM, Using Dynamic Link Libraries with LE!! Ooooops

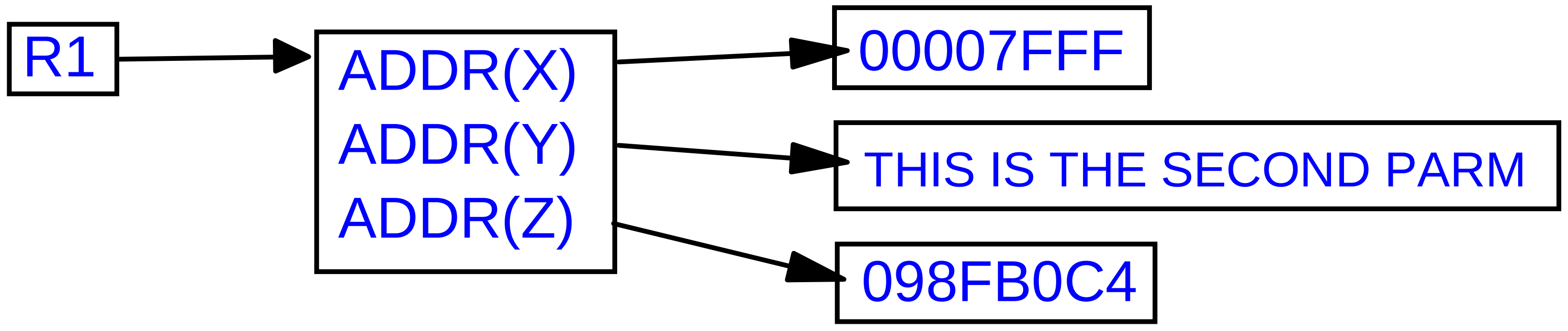
Introduction: HLL Parameter passing

- **Review 'Normal' S/390 parameter passing**
  - R1 points to 'Parameter List'
  - Parameter list is a list of pointers to parameters
- **Assembler and COBOL support 'normal' parameter passing**
- **PL/I is mostly normal except for operand descriptors**
  - Still pointed to by addresses in parm list
- **C/C++ introduce a new way**
  - Parameter values IN the parm list (sometimes)

Introduction: 'Normal' Parameter passing

- **77 X PIC S9(9) BINARY VALUE 32767.**
  77 Y PIC X(23) VALUE 'THIS IS THE SECOND PARM'.
  77 Z POINTER.

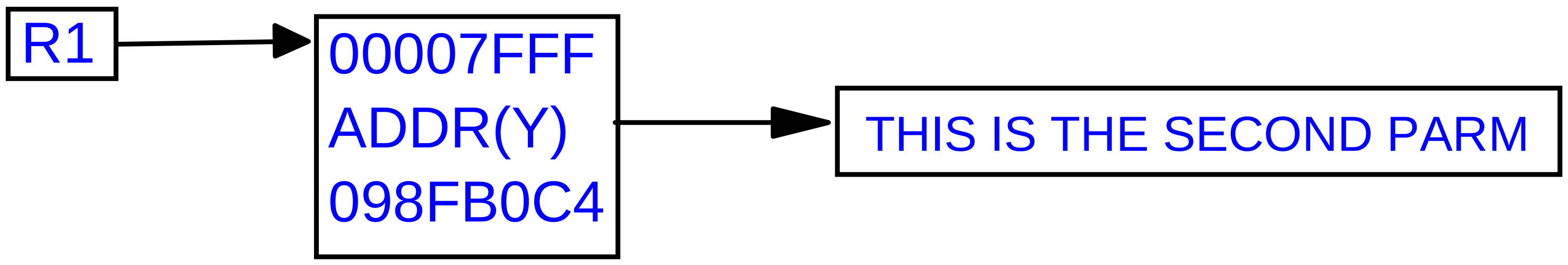
  SET Z TO ADDRESS OF Y.
  CALL 'SUB1' USING X, Y, Z



Introduction: 'C Style' Parameter passing

-
  ```
  { int x; char y(23); char *z;
     x = 32767;
     strcpy(y, "THIS IS THE SECOND PARM");
     z = y;

    sub1(x,y,z);
  }
  ```



Introduction: 'C Style' Parameter passing

- **What is a #pragma ?**
  - C compiler directive statement
  - In C++, the equivalent is EXTERN
  - EX: #pragma linkage(cobrtn,COBOL)
- **What does #pragma linkage COBOL do?**
  - Changes parameter passing to match 'normal 390' style
  - Changes writeable static processing for RENT programs to work for 'dynamic' ILC with older COBOL compilers
- **Example of C calling COBOL:**

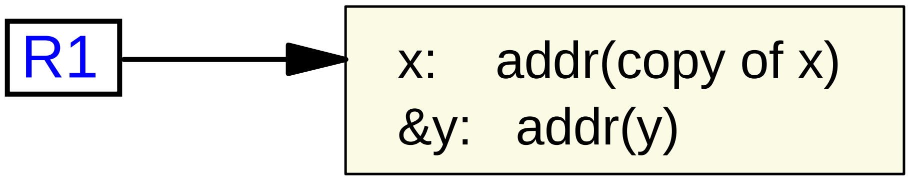
```
#pragma linkage (cobrtn,COBOL)
 { int x;
   x = 5;
  cobrtn(&x);
 }
```

Introduction: 'C Style' Parameter passing

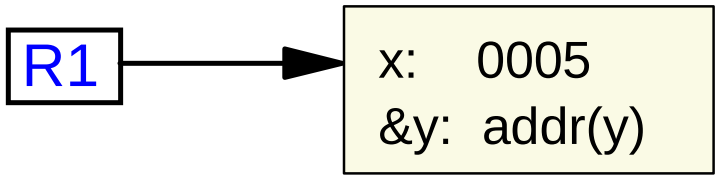
- **Example:**

```
#pragma linkage (cobrtn,COBOL)
 { int x,y;
   x = 5;
   y = 99;
  cobrtn(x,&y);   }
```

- **WITH #pragma COBOL, a ptr to x would be passed:**



- **WITHOUT #pragma COBOL, x is passed in parm list:**



extern-less ILC for COBOL w/C++

- Example:

```
extern "COBOL" {void cobsub (int, int*);}
int main()
{ int x,y;
   x = 5;
   y = 99;
  cobsub(x,&y); }
```

- With no extern, can't even call C or COBOL programs
  - 'Name mangling' is done to store information about function in function name
- Use extern "C" for ILC with C++ and COBOL
  - Turns off name mangling
  - Use 'C style' coding in COBOL programs
- Can still use extern "COBOL"
  - like #pragma linkage COBOL in C
  - Adds a level of indirection to parameters
  - Turns off name mangling
  - C callers would have to use COBOL style

C/C++ and COBOL parameter passing basics

- Choices for COBOL with C/C++:
  - Use COBOL-style linkage in all programs to be shared
    - Use #pragma in C code, EXTERN COBOL in C++
  - Use C-style linkage in all programs to be shared
    - #pragma-less in C code, EXTERN C in C++
  - Use different linkage for each combination
    - Makes sharing routines more difficult

C/C++ and COBOL parameter passing basics

- Now COBOL can also use C-style linkage conventions
  - USING BY VALUE, Z literals, RETURNING, and POINTERs
- Common mistake: passing strings between C and COBOL

```
*pass a null-terminated string to a C function
 CALL 'FOO' USING BY CONTENT Z'abc'.

int FOO(char* x)      ( same as char x[ ] )
 { printf("%s \n",x}
```

- Won't work, high order bit set in parameter list
  for 'abc#'
  - C does math on addresses

C/C++ and COBOL parameter passing basics

- **How to handle strings between C and COBOL?**
  - **Always use pointers to strings like C does**
- **A better way to pass strings between C and COBOL**

```
MOVE  Z'abc' TO X.
SET X-PTR TO ADDRESS OF X.
CALL 'FOO' USING BY VALUE X-PTR.


int FOO(char* x)
 { printf("%s \n",x}
```

- **Will work, high order bit not set in parameter list for X-PTR when USING BY VALUE**
- **No #pragma linkage COBOL required!**

#pragma-less ILC for COBOL and C/C++

- **Rules for #pragma-less ILC:**
  - **Use PROCEDURE DIVISION USING BY VALUE**
    - in COBOL programs receiving parms passed from C
  - **Use CALL xxx USING BY VALUE**
    - in COBOL programs for passing parms to C
  - **Use CALL xxx RETURNING**
    - in COBOL programs that CALL C functions
  - **Use PROCEDURE DIVISION RETURNING**
    - in COBOL programs that will be invoked as functions by C programs
  - **Dynamic CALL statements from COBOL to reentrant C**
    - Use COBOL for OS/390 & VM 2.2 or later
    - Use LE for OS/390 V2R9 or later
  - **Using fetch() in reentrant C to COBOL**
    - Use COBOL for OS/390 & VM 2.2 or later
    - Use LE for OS/390 V2R9 or later
  - **Can also use DLL support instead of fetch() or dynamic CALL**

#pragma-less ILC for COBOL and C/C++

- **C Function calls from COBOL:**

```
CALL 'getaddr' RETURNING X-PTR.
SET ADDRESS OF LS-ITEM TO X-PTR.
```

```
int getaddr (x_ptr pointer);
 { return x_ptr;}
```

#pragma-less ILC for COBOL and C/C++

- **COBOL Function calls from C:**

```
int callgetaddr (x_ptr pointer);
 { x_ptr = getaddr();}
```

```
PROGRAM-ID. GETADDR.

PROCEDURE DIVISION RETURNING X-PTR.

SET X-PTR TO ADDRESS OF WS-ITEM.
GOBACK.
```

ILC Parameter Passing - optional parms

- **COBOL, PL/I and C/C++ all support optional parms**
- **One common use is for LE callable services**
  - **Example: omitting the FC parameter**
- **Also allows for flexible use of common subroutines**
  - **Pass different numbers of parms when called for different purposes**
  - **This has been done in past, was never officially supported without using omitted parameters**

ILC Parameter Passing - optional parms

- **COBOL:**

```
Call 'sub1' Using PARM1, OMITTED, PARM3

Program-ID. SUB1.
Procedure Division Using RPARM1, RPARM2, RPARM3

   If ADDRESS OF RPARM2 = NULL Then
      Display 'No 2nd PARM was passed this time'
   Else
      Perform Process-Parm-2
   End-If
```

ILC Parameter Passing - optional parms

- **PL/I:**

```
CALL sub1 (PARM1, *, PARM3);


SUB1: PROCEDURE ( RPARM1, RPARM2, RPARM3);

   If ADDR(RPARM2) = SYSNULL() Then
      Display ('No 2nd PARM was passed this time')
   Else
      Call Process_Parm_2;
```

ILC Parameter Passing - optional parms

- C/C++:

```
sub1 (PARM1, NULL, PARM3);


void SUB1 ( RPARM1, RPARM2, RPARM3)

   { If (RPARM2 == NULL)
       printf ('No 2nd PARM was passed this time')
     Else
       Process_parm_2();
    }
```

COBOL-specific ILC enhancements

- **Performance**
  - **COBOL <-> PL/I ILC has 80% less overhead than pre-LE**
  - **COBOL <-> C/C++ ILC has 80% less overhead than pre-LE**
  - **C/C++ <-> PL/I ILC about equivalent to pre-LE**
- **CICS improvements**
  - **Under OS/VS COBOL, no CALLs allowed at all**
    - EXEC CICS LINK/XCTL only
  - **Under VS COBOL II, static and dynamic CALLs between VS COBOL II pgms**
    - EXEC CICS LINK/XCTL to other languages or OS/VS COBOL
  - **Under LE, static and dynamic CALLs to other languages**
    - All programs compiled with LE-conforming compilers
- **#pragma-less ILC for COBOL and C/C++**

COBOL:PL/I, COBOL:C ILC Migration

- **All interlanguage communications applications that have COBOL must be relinked before moving to LE**
  - **Can relink prior to moving using COBOL-PL/I, COBOL-C Migration Aid**
  - **PN69803 & PN69804 for OS PL/I V2**
- **OS/VS COBOL programs cannot call PL/I programs under LE**
  - **PL/I programs can't call OS/VS COBOL programs either**
  - **Must do source conversion and recompile with newer COBOL compiler**
- **PL/I multitasking applications have restrictions under LE**
  - **Can only call COBOL in one 'task' (thread)**
  - **Unless COBOL programs compiled with THREAD option!**
    - Enterprise COBOL for z/OS and OS/390 Version 3