Part 2: Practical Experience.

Approximate time to completion: five to ten hours.

Please Note:
You must complete Part 1 of the contest before beginning Part 2.
Contestants who do not complete Part 1 will not be judged in Part 2.
The object of this part of the contest is to get you acquainted with
some of the more practical aspects of being a Systems Programmer.

These challenges are modelled after tasks that are commonly
performed in z/OS shops.
The instructions are not nearly as detailed for this part of the contest -
you'll have to use some additional resources, along with your
new-found mainframe expertise, to complete the challenges.

The first 30 contestants to complete this section correctly each win a
Lenovo USB Hard Drive.

Don't forget to create a XWORD member to store your clues &
answers.

At the end of Part 2 You will have found 12 more crossword clues.

Good Luck!

Table of contents:
1. Prime your Part 2 data set
2. Run a REXX program with JCL
3. Debug broken JCL
4. Write your own ISPF panel
5. Trickier JCL error
6. Conversion Challenge
7. Unix on the mainframe
8. Write a REXX program (Calculate task)
9. Write a REXX program (Read/write a dataset)
10. XMIT all your hard work
11. What's next?

Before you begin, some places to go for help:

If you get stuck during the contest, or just want to find out more
about mainframes, check out these mainframe resources:
Introduction to the New Mainframe (US):

Introduction to the New Mainframe (RUS):

A new college textbook; great for introductory and overview information about the mainframe.

z/OS Library Centre (US):
http://www-03.ibm.com/systems/z/os/zos/bkserv/

Contains all of IBM's official technical documentation on z/OS, with hundreds upon hundreds of searchable books and documents. Everything you’d ever need to know about mainframes can be found here.

Message LookAt Facility (US):
http://www-03.ibm.com/systems/z/os/zos/bkserv/lookat/

If you receive an error message from the system, you can look up a detailed explanation of the error here.

Mainframe blog (US):
http://mainframe.typepad.com/

A mainframe blog that gives you a chance to interact with other mainframe enthusiasts.

z/OS Hot Topics newsletters (US):
http://www-03.ibm.com/systems/z/os/zos/bkserv/hot_topics.html

Keep up on the latest technical news about z/OS.

1. Prime your Part 2 data set

Background:
You're an old pro at allocating data sets by now. Refer to "Part 1: Introduction," if you need a refresher.

Your challenge:
Allocate a data set called 'RUS####.RUS####.ZOS.PART2' (where RUS#### is your userid you were allocated). The data set should be a PDSE with a primary allocation of 1 tracks and a secondary allocation of 1 track. This data set will hold all of your work throughout Part 2 of the contest. You'll submit it to IBM when you’ve finished all of the challenges. For the rest of the instructions, we'll refer to this data set as just ZOS.PART2 for short.

**Important:**
If you fail to use your user ID twice in the qualifier (name) for this new data set, your entry for Part 2 of the contest will not be judged! You MUST name the data set beginning with your user ID twice.

### 2. Run a REXX program with JCL

**Background:**
In Part 1: Breaking the ice, you ran a simple REXX program from the command panel. Now we are going to run that same REXX program in batch.
In z/OS, you have two different ways to do work: interactively in the foreground or non-interactively in the background. All of your work from Breaking the Ice was considered foreground work. You can do a lot of stuff in the foreground, but if a program takes any length of time to process, you can't do anything else with your session while it's running.
If you submit long-running programs in batch, however, you can still continue to use the foreground while your work cranks away in the background. In order to submit work to run in batch, you need to instruct z/OS through JCL (Job Control Language). You submit your JCL to JES (Job Entry Subsystem), and the system allocates the necessary resources for your job, and then executes the work. JCL is a very rich language, and it's very different from any other programming languages. JCL has been around for over 30 years; as you might imagine, JCL for just about every conceivable task has already been written.
Most systems programmers will find a JCL file (normally called a "job" in mainframe-speak) that does something very close to what they want to do, and then make a few little changes to it so that it fits their needs. You're going to do the same thing.

**Your challenge:**
There is a data set on the system called ZOS.CONTEST.JCL.D2008.
The member RUNREXX in this data set contains JCL that is very close to what you need. Copy this member into your own RUS####.PDS.CNTL data set.

To do this:
1. Open the ZOS.CONTEST.JCL.D2008 data set, then view the RUNREXX member by putting an V beside it.
2. Tell the system which text you would like to copy into the new data set member, you can do this using the cc tags you learned in Breaking the Ice. (you can also use c# beside the first line of JCL, where # is the number of lines you'd like to copy. If # is higher than the number of lines in the member, then c# will copy the entire member.)
3. On the command line, run the command:
   CREATE PDS.CNTL(RUNREXX). Remember, the system automatically uses your RUS#### ID as the high-level qualifier if you do not put single quotes around the data set name. This command will copy the text you specified using the "c#" or "cc" tags into your own RUS####.PDS.CNTL data set, with the member name of RUNREXX.

Once you've copied RUNREXX into your own data set, open it with the ISPF editor. Make these changes:
1. Change your_user_id to your RUS#### ID.
2. Change your_member_nam to the name that you called the PDSE member that you made in "Part 1: Breaking the ice." (In the examples for Part 1, the member was named SHANE.)

On the command line, enter: SUB (short for submit). Press Ctrl. This will send your job to JES for processing. Not so hard, right? But where do you find out what happened when your job ran? The answer is SDSF, which means Spool Display and Search Facility. To get to SDSF on this system, go to ISPF primary option S If you're not at the primary ISPF menu, don't forget that you can type =S From anywhere and jump right to it. At the main SDSF prompt, enter the command:
   OWNER RUS####. To see all the jobs in your held output queue, enter H ALL.

View your output, along with some system-generated output, by putting a ? beside your job (the JOBNAME specified in your JCL job was REXX) and pressing Ctrl. Put an S (for select) beside each piece here to see what happened when you ran your job. Now back out to the screen that lists REXX in the held output queue. Put an S beside REXX and press Ctrl. What you'll see now is the same output from when you specified ? - this time, though, it's all concatenated (linked together) into one display for easier viewing. After verifying that your job produced the same output that it did in "Part 1: Breaking the Ice," write out all of the output to a member of your ZOS.PART2 data set called REXXRUN. To do this, use the XDC command next to the job.
Make sure that you type XDC from next to the REXX job, not just one output piece - this will ensure that JESMSGLG, JESJCL, JESYSMSG, SYSTSPRT, and OUTDD are all sent as a single member to your ZOS.PART2 data set. From the XDC menu, correctly specify the data set name, member to use and disposition (OLD). Otherwise, you can accept the defaults and press Ctrl. You'll find a new member in your ZOS.PART2 data set. Go have a look. You just knocked out another challenge!

3. Debug broken JCL

Background:
As a systems programmer, you will be doing lots of work with JCL. If you make a simple JCL error, such as forgetting a comma or putting an X in column 72, your job will, in most cases, end very quickly, and the system will inform you of a JCL error.

Your challenge:
Fix a simple JCL error. Copy ZOS.CONTEST.JCL.D2008(BADJCL) to your own PDS.CNTL data set under a new member name of your choosing. Submit the job.
You'll get a JCL error. What gives? Check your output in SDSF and fix the error.
Hint: plug the message ID from JESYSMSG into the LookAt facility for some help (the message ID starts with IEF). The system you are currently using is z/OS, release level V1R6. After you fix the error and get the job to run correctly with no JCL errors, go back to SDSF to check your held output. You'll find a new piece of output. Once you verify that you got this new output, XDC all of the output from the job into your ZOS.PART2 data set under the member name FIXEDJCL. Again, be sure that you XDC from beside the JOBNAME (not the DDNAME), to ensure that your new member will contain JESMSGLG, JESJCL and JESYSMSG, as well as the new piece of output.

That's it!
You just debugged a JCL job.

You should also notice some more crossword clues.

4 Write your own ISPF panel

Background:
You can add a new icon on the desktop of your PC, and you can
customise your environment in z/OS as well. The panel interface of
ISPF can be
completely re-written to fit your needs. Most mainframe shops do this
as a first step of their customisation. In fact, the Zeus system that you
are using right now already has a degree of customisation to its
panels.

Your challenge:
Add the SDSF option to your own customised ISPF panel. Drop down
to the TSO READY prompt (=x in ISPF is the shortcut) and type
ex 'ZOS.CONTEST.CLIST(BLANKPNL)'. This will start a new,
customised ISPF main panel. This panel has been stripped down -
there is just enough
functionality there for you to work on a new panel of your own design.
You'll see that the Utility Menu - Option 3 - has also been reduced.
Note: If at any time you need to get back to the original ISPF main
panel (as you might if you break something in the course of
completing this challenge), you can do so by entering LOGON
RUS#### from the TSO READY prompt.

Type in the PANELID command on the command line - you'll see the
panel name
appear in the upper left corner. As you go into other menus (like
Option 3), you'll see that the panel ID changes. Browse around and
check out some panel IDs. You can find the code for the slimmed-
down ISPF and Utility panels in the PDS (Partitioned Data Set,
remember?) ZOS.CONTEST.PANELS - the panel IDs for the slimmed-
down ISPF and Utility panels correspond to member names in that
PDS. Have a look inside ZOS.CONTEST.CLIST(BLANKPNL).
You'll see a list of concatenated data sets - these are all of the data
sets that hold other panels that you might need while working in ISPF.
Often in z/OS, PDSs are concatenated so that they are processed
logically as one big PDS. When two data sets have the same member
name, the one that is first in the concatenation takes precedence.
You'll notice that in this CLIST (command list), ZOS.CONTEST.PANELS
is first in the list - it takes precedence over all the others. The original
ISPF main panel is in CENTER.ISPPLIB, which is further down the list.
Here's what you need to do:
1. Copy the ZOS.CONTEST.PANELS data set into your own data set
that you can modify. Copying an entire data set with all its members is
something you haven't done yet. Here's how you do it:
1. Find the Move/Copy Utility (ISRUMC1) from your ISPF main panel.
2. Specify the copy option, then fill in the data set name you are
copying.
3. On the next panel, specify the data set name you are copying to.
4. On the next panel (ISRMCALL), select Allocation Option 1.
5. On the next panel, put an "s" beside each member to specify that you want to copy both of them.
6. Verify that your new data set contains both members.

2. Copy ZOS.CONTEST.CLIST(BLANKPNL) into your own data set that you can modify.
3. Add an option to your copy of ZOS.CONTEST.PANELS to make SDSF an option from your slimmed-down ISPF panel. Your new panel must tell the user what option to enter to open SDSF -- "S" or "13" are both common conventions, though you can make it something different if you choose. SDSF must open correctly from your panel when the appropriate command ("S", "13" or whatever you choose) is entered.
4. Modify your copy of BLANKPNL so that your new panel takes precedence.
5. Execute your new CLIST from the TSO READY prompt to display your customised panel.
6. The judges will review your copy of BLANKPNL to ensure it is correct.

5. Trickier JCL error
Have a look at ZOS.CONTEST.JCL.D2008(BADJCL2). It doesn't run as is. There are a few errors in the JCL. Luckily, fixing JCL errors is your specialty! Track down the errors, fix it and use SDSF to save all of the correct output to a member named LASTJOB in your ZOS.PART2 data set. When it runs correctly, more crossword clues will be revealed. If you get stuck, start going through the JOB Card parameters one by one to see if you can figure out what each one of them means. Hint: Because you've stuck with it this far, you deserve a hint. Initiators are address spaces where batch jobs run, instead of the jobs running in your address space. For a job to execute, it needs an eligible initiator. Take a look at the INIT option in SDSF to find out more about the initiators on the system. We've said too much already! Good luck.

6. Conversion Challenge

Background:
When dealing with data in a computing environment the representation of that data will vary depending on that environment in which the data is being held. The data will typically be unreadable with the human eye as it has been encoded into a machine format relating to the hosting machine, the operating system and the country where the data exists. Extended Binary Coded Decimal Interchange Code
(EBCDIC) pronounced "eb-suh-dick." is used in IBM mainframes and some IBM midrange computers. It is an 8-bit code that represents a character.

Your challenge:
Is to convert the EBCDIC data representation in ZOS.CONTEST.JCL.D2008(DATA) into an alphanumeric character string. This data contains more crossword clues. Store the converted data as RUS####.RUS####.ZOS.PART2(DATA).

Note: There are many ways to achieve this task, I suspect there's a translator on the web.

7. Unix on the mainframe

Background:
z/OS has been supporting Unix processes since the early 90s. Many Unix programs have been ported to z/OS. Nearly all new work on the mainframe uses the Unix kernel and filesystem.

Your challenge:
There are several ways to invoke a Unix shell, using a TSO CMD or invoking from an ISPF panel are two ways.
From ISPF option 6 Enter OMVS to access the Unix shell, find where java is installed and what version it is; record this information in a file called /tmp/rusxxxx/javainfo (You can find the required Unix cmd using google no doubt ;-) )
To leave the Unix shell enter QUIT & press F2.
As a judge I don’t want to have to go searching the Unix filesystem and your z/OS datasets for your answers so I want you to copy /tmp/rusxxxx/javainfo to your ZOS.PART2 with a member name of JAVAINFO. Hint: the TSO OGET command is an easy way to do this.
You can get help on TSO commands by enter H cmd from ISPF option Where cmd is the command you want the syntax for.

8. Write a REXX program

Background:
In earlier tasks you’ve run REXX programs interactively (foreground processing) and in batch (background processing). In this task you get a chance to write some REXX. REXX is an interpreted language so there is no need to compile to run the code (though there is a compiler available as well), it’s fast to learn and very powerful and runs on just
about any platform but was developed on the mainframe at IBM Hursley. ZOS.CONTEST.JCL.D2008(REXX) contains a simple REXX program that you can use as a start for your own.

Your challenge:
Create your exec in RUS####.RUS####.ZOS.PART2(SHOPDAY) In this task write a REXX exec to calculate how many shopping days there are left until Christmas day 2009, from the day the exec is run.

Your exec should ask the user how many days a week they could shop, e.g I can't go shopping Saturday therefore I can only shop 6 days a week. The last shopping day is 24\textsuperscript{th} December Christmas eve.

The following book lists all the functions available to you. \textit{REXX Users Guide}.

http://publibfp.boulder.ibm.com/cgi-bin/bookmgr/BOOKS/ikj4c310/CCONTENT

Don't forget to test your code ! (Look at the trace cmd if your answer is not as expected it will help you debug your code).

\textbf{9. Write another REXX program}

Background:
In task 6 you converted some hex code into readable text. You probably found a converter on the web, but you could use REXX on the mainframe.

Your challenge:
Create a copy of ZOS.CONTEST.JCL.D2008(DATA) calling it RUS####.RUS####.ZOS.PART2(HEX) (Don't call it DATA as you will overwrite your existing member !). Your exec should READ member HEX, do the conversion and WRITE member CONVERT. To do this you will need to use a couple of REXX functions, EXECIO is used to read/write to a dataset, you can read all about it in the Rexx Users Guide URL in the previous task. Checkout ZOS.CONTEST.JCL.D2008(READMEM) for some code to get you started, this code will only READ a member.

You can run the sample code from ISPF 6 and entering :

```
exec 'ZOS.CONTEST.JCL.D2008(READMEM)'
'RUS####.ZOS.PART2(HEX)' exec
```
10. **Well done now submit your entry.**

Don't forget to ensure you have updated your XWORD member with the clues you've found and the answers. Keep a copy of this member on your own PC (Use cut & paste), you will need this data if you want to continue to Part3.

Send your part2 entry as follows from ISPF option 6 :

XMIT TSTMVS01.BABEYS DA(RUS###.ZOS.PART2)

11. **What next ?**

Part3 of the contest runs on a different machine so if you choose to enter part3 you must register your interest by sending an email to studmain@uk.ibm.com to be allocated a new userid for part3.

Here's a taster of what to expect in Part3 :

- Task 1 – Use of DB/2
- Task 2 - Cobol programming (Fix the performance problem)
- Task 3 - Dump reading – Learn how to read a z/OS dump
- Task 4 – CICS & MQ – Real world challenge
- Task 5 - Create a website to represent the Puzzle