CSCI 465/680-J9 **Assignment 2 – JCL and Program Fetch** **Fall 2019**
50 points

The primary purpose of this assignment is to give you some experience in writing and documenting simple JCL. Program fetch is what we call the process of "fetching" a program object (now referred to by IBM as a program object) from an existing load library and executing it. This is what happens in the real world after a program is developed, tested thoroughly, and approved for promotion into production.

It would be ridiculous if we had to compile a COBOL program, assemble an Assembly language program, or, for that matter, compile ANY program every time we need to execute it. Instead, the program is compiled (or assembled) and "link-edited or linkage-edited" by the Linkage Editor (a process now referred to by IBM as being "bound" by the Binder).

If all goes well, the Binder produces a so-called "permanent" program object which is stored as a member with the same name as the program itself (COBOL PROGRAM-ID or Assembler CSECT) in what we call a load library, i.e., a PDSE set up with parameters to hold only what we program objects, the mainframe's version of an executable, or .exe, file.

Once stored and ready for use at any time in the future, anyone with access to the appropriate load library can "fetch" the program object and execute it. As an example, the following excerpted line of JCL is "fetching" the COBOL Compiler:

```
//JSTEP01  EXEC PGM=IGYCRCTL,PARM=APOST
```

In most cases, you have to tell the system where to find the program object to which you refer with the PGM= keyword parameter. This is done by adding a STEPLIB DD card to each of your job steps. You can instead add a JOBLIB DD card just following the JOB card in your JCL. If you code the JOBLIB DD card, the system will first look in THAT PDSE for any fetched program objects referred to by any of your job steps.

By the way, the three common modules do not require a STEPLIB because they are so commonly used – hence we refer to them as the "common modules" – that a STEPLIB (or JOBLIB for any step fetching one of the common modules) is not necessary. For the programs ***you*** write from scratch, a STEPLIB will be necessary.

Programming Instructions

Write a complete job with two job steps. The first step will fetch and execute a program object named READDATA stored in your instructor's load library PDSE:

```
KC02322.CSCI465.LOADLIB
```

The second step will fetch and execute a program object named RPTDATA from the same load library.

The first step's program, READDATA, will read 80-byte records from a member named DATA2 in your instructor's PDS named:

```
KC02322.CSCI465.DATAFA19
```

and will write them out to a second data set defined with 80-byte records. Your JCL must indicate this second output data set as a temporary data set that will be passed to the second step of your JCL.

Note that, when reading data from a specific member of a PDS or PDSE, you refer to it as:

```
KC02322.CSCI465.DATAFA19(DATA2)
```

This is different than referring to a program object in a STEPLIB, though. You already indicate the name of the program object with PGM= on the EXEC card.

The second step's program, RPTDATA, will read 80-byte records from the temporary data set passed to it by the first step.

It will then write them out as "report records" to standard output as 133-byte records.

The input and output DD names for the first step are:

`RECSIN1` (used for input)
`RECSOUT1` (used for output – this should be an 80-byte LRECL temporary data set)

The input and output DD names for the second step are:

`RECSIN2` (used for input – this should refer to the 80-byte LRECL temporary data set from step 1)
`RECSOUT2` (used for output)

Also, add the following third DD card to ***each*** of the two steps. In the case of an abend, this will provide you a large dump of storage with a PSW and registers:

```
//SYSUDUMP DD SYSOUT=*
```

Documentation

Follow the coding and documentation guidelines for your JCL ***very*** carefully. They are found in Chapter 1 of the CSCI 465/680-J9 Course Notes.

Output

Your JCL should output the following lines, double spaced, although they will look single spaced in SDSF (Press F10 to scroll 1 column to the left in SDSF Status Queue so that you can verify the carriage control):

```
QUOTATIONS FROM TWO IMPORTANT AMERICAN HISTORICAL DOCUMENTS      FALL 2019

"We hold these truths to be self-evident, that all men are created

equal, that they are endowed by their Creator with certain unalienable

Rights, that among these are Life, Liberty and the pursuit of

Happiness." -- Declaration of Independence, 1776

"We the People of the United States, in Order to form a more perfect

Union, establish Justice, insure domestic Tranquility, provide for the
```

```
common defence, promote the general Welfare, and secure the Blessings

of Liberty to ourselves and our Posterity, do ordain and establish this

Constitution of the United States of America." -- Preamble to the U.S.

Constitution, 1789
```

Terminology

Note that there is use of both of the terms PDS (PDSs) and PDSE (PDSEs) in this document. Since implementation of the Enterprise COBOL Compiler, v. 5.1.0, at Marist in May 2015, it is necessary to use load libraries that are PDSEs (Partitioned Data Sets Extended) for all load libraries now. My load library, `KC02322.CSCI465.LOADLIB`, is a PDSE. The one you created in Assignment 1 for your own use is too!

Assignment Submission

When you are satisfied with your JCL, documentation, and report output, use mar_ftp.exe to download your output – the entire job and its report output – and submit the .txt file on the assignment's page on Blackboard.

Missing Output in Downloaded .txt File

Note that it is very important that, BEFORE you submit it, you open the .txt file to be sure that it is all there and formatted as expected. Be sure to do this throughout the entire semester.

Missing output is common. You will sometimes see all of the output in TSO/ISPF's SDSF facility but then it doesn't all show up in your .txt file. This often means that you have 1) forgotten the PARM=ASA on an assembly step or 2) you have bytes of garbage somewhere in your 133-byte output lines or they are not 133 bytes long.

To correct the second of these possible problems when you begin writing your own COBOL or Assembler programs, make sure that each of your output lines is defined as exactly 132 bytes in COBOL and that each length of bytes in between receiving fields is defined as spaces. For Assembly language programs, make sure that each of your output lines is defined as exactly 133 (including the first byte of carriage control) and that each length of bytes in between receiving fields is defined as spaces.