CSCI 465/680-J9 **Assignment 3** **Fall 2019**

Three Common Modules

100 points

Overview

This assignment is in two parts, each worth 50 points. The first part consists of assembling and executing an Assembler program and the second consists of compiling and executing a COBOL program.

You will turn in two (2) text files of output from this assignment. One of the text files will be for Part A, the Assembler part, and the other will be for Part B, the COBOL part. This assignment's objective is to give you experience writing and running basic JCL jobs on the Marist mainframe.

Note: Although you will not be required to document either of the programs given to you, please review the JCL documentation guidelines in the course notes and document your own JCL adequately and appropriately. Inadequate documentation will result in deducted points. Be sure to include the double forward slashes by themselves at the bottom of each job file.

Part A - High-Level Assembler (50 points)

For Part A of the assignment, the source code of a simple complete Assembler program is provided for you as ASMPGM3.txt. This file is included in the assignment folder on Blackboard. Use the High-Level Assembler for the first step.

If you do not want to type or copy and paste the entire program into a member of your PDSE, you can open FileZilla, sign into your account on Marist, locate the folder on your machine into which you downloaded the ASMPGM3.txt file, right mouse-click on the file and rename it to remove the .txt extension. You can then click and drag it into your ASSIGNS PDSE on Marist and build the JCL ***around it*** in TSO/ISPF. (You can do the same for COBPGM3.txt for Part B!)

Your assignment for Part A is to create a complete JCL job stream to 1) assemble, 2) bind, and 3) fetch and execute this small Assembly Language program ***as an in-stream program***.

Programming Notes:

- The first step of this job will be the High-Level Assembler step, the second will be the Binder step, and the third will be a fetch and execute step.

- Be sure to add `PARM=ASA` to the EXEC card of the Assembler step. This parameter will be necessary ***throughout the semester*** on the Assembler steps of your JCL.

  Note: `PARM=ASA` tells the Assembler to produce the assembly listing using American National Standard (ANSI) printer-control characters. (If `NOASA` is specified the Assembler uses machine printer-control characters. We want the ASA version.)

- Write the object module from the Assembler step to a temporary data set that will be passed to the Binder step.

- Write the load module that comes out of the Binder step to **your own** PDSE named LOADLIB. The name of the load module member should be the exact same as the name of the program itself.

- Follow the Binder step with a third that fetches and executes your program object.

- Do not execute either the Binder step or the fetch and execute step unless the return code from all previous step(s) steps is zero (0).

- The name of the input file is the same as that for the COBOL program above:

  `KC02322.CSCI465.DATAFA19(DATA3)`

  The input DD card for this file for the Assembler program must be named `ASMIN`.

- The output DD card for the Assembler program must be named `ASMOUT`. This output should be written to standard output.

- The program you turn in should have correct output, and all three return codes should be 0000. The return codes are in the center of about lines 6, 7, and 8 of the JES2 Job Log (second printed page). You should see return codes of 0000 for the Assembler step, the Binder step, and the fetch and execute step.

Part B - COBOL (50 points)

For Part B of the assignment, the source code of a simple complete COBOL program is provided for you as COBPGM3.txt. This file is included in the same folder on Blackboard in which you found this assignment description. To get the file to Marist, use the FileZilla "trick" described above in Part A.

For this part, your assignment is to create a complete JCL job stream to 1) compile, 2) bind, and 3) fetch and execute this small COBOL program ***as an in-stream program***.

Programming Notes:

- The first step of both this job will be the COBOL Compiler step, the second will be the Binder step, and the third will be a fetch and execute step.

- Use the following COBOL compile parameters on the COBOL compile step:

  `PARM=APOST` See the course notes for an explanation of this parms.

- Write the object module from the COBOL Compiler step to a temporary data set that will be passed to the Binder step.

- Write the program object that comes out of the Binder step to your own PDSE named LOADLIB. The name of the program object member should be the exact same as the name of the program itself.

- Follow the Binder step with a third step that fetches and executes your program object.

- Do not execute either the Binder step or the fetch and execute step unless the return code from the above steps is zero (0).

- The name of the input file is the same as that for the COBOL program above:

  `KC02322.CSCI465.DATAFA19(DATA3)`

  The input DD card for this file for the Assembler program must be named `COBIN`.

- The output DD card for the Assembler program must be named `COBOUT`. This output should be written to standard output.

- The program you turn in should have correct output, and all three return codes should be 0000. The return codes are in the center of about lines 6, 7, and 8 of the JES2 Job Log (second printed page). You should see return codes of 0000 for the COBOL Compiler step, the Binder step, and the fetch and execute step.

IMPORTANT

Be sure to follow ALL of the JCL Documentation Standards as described in Chapter 1 in the CSCI 465/680-J9 Course Notes in both Part A and Part B! (Hint: Get one of the two jobs working and completely documented first, then copy it into another member of your ASSIGNS PDSE and change what needs to be changed in the JCL and the documentation.)

What To Turn In

Use mar_ftp.exe to get .txt file versions of the output down from the Marist output queue onto your PC. Submit the **two (2)** appropriately named .txt files for this assignment as instructed.