## Last Time

## In our previous lab we proved that the compilers produce object code by translating human readable source code into machine readable code. Before a source code instruction can be executed a number things have to take place. This is where language compilers, like the COBOL compiler come in. Compilers create object code that takes advantage of the hardware’s instruction set architecture. In doing so, the compiler will create code that uses different instruction formats. These instructions can then be executed by the CPU execution units. But the execution unit needs CPU registers to do some its work and needs main memory as a staging area for your program’s instructions. Once your source code is translated to object code instructions, these instructions can be executed as follows:

* **Program Load -** your program’s load module or some piece of it must be retrieved from secondary storage (like disk) then loaded into the computer’s main memory (or RAM)
* **Fetch -** each instruction must be retrieved from memory by the CPU’s Fetch Unit via the fetch portion of the Fetch/Execution cycle and placed in the **Instruction Register**
* **Increment for next address -** **Instruction Address Register** is incremented by the current instruction’s length to calculate the address of the next instruction**.** In the System z this register is called the **Program Status Word** or PSW
* **Decode -** the instruction in the Instruction Register is decoded to determine which circuits to activate
* **Execution -** the decoded instruction is executed by the execution unit to activate the appropriate circuits
* **Write back-** the system clock has a set time period that waits for the bits to settle in i.e. not all bits are turned on or off at the exact same instant

## Out of Order Processing

To get higher performance from a CPU, computer designers have added circuitry to separate the stages of an instruction’s execution so that each stage can be worked on separately. This allows more than one instruction to be worked on at the same time. This expansion of circuitry led to the creation of instruction **pipelines**. Assuming a four stage instruction cycle, a two instruction pipeline might looks like this taking 5 clock units to execute these two instructions versus 8 clock units in a non-pipelined architecture.

**Fetch**

**Decode**

**Execute**

**Writeback**

**Fetch**

**Decode**

**Execute**

**Writeback**

There is a downside to pipelining. If an instruction in the pipeline is a branch to an instruction not in the pipeline the CPU needs to purge the instructions already in the pipeline and refill it with instructions relevant to branch. We call this event **out of order processing**. As you would suspect out of order pressing can negate any pipelining performance improvements. To minimize out of order processing, programs should be written with few, well-designed branching operations.

**Input/Output (I/O)**

One of the major characteristics of enterprise-class problems is the need to handle large scale input and output operations. As a general rule, enterprise solutions need to process high volumes of data from files and databases. In the credit card example discussed in class, we discussed how challenging it is for a company like Visa to process 81.6 billion card transactions in 2013. This is only one example of the transaction volume enterprise applications need to handle, but large enterprises have many applications with similar processing characteristics.

To process large volumes of data, the data must be placed and transferred from/to high speed I/O devices, like disk. Data from disk (SSD and tape) is transferred in blocks, so it doesn’t make sense to execute a separate CPU instruction for each piece of data in a block. A block of data may, in some cases, contain hundreds or thousands of records, so one I/O operation can move hundreds or even thousands of records from a device to memory. To speed up I/O operations, computer designers, provide the capability to move blocks of data directly between I/O devices and memory. The data movement process is under the control of a special purpose processor called the **I/O module**, also called **I/O Control Unit**. Each I/O device has an I/O control unit.

When an application program requests data (a record) the data transfer process is initiated by the CPU using **Programmed I/O**, but the CPU is bypassed for the remainder of the transfer. The I/O module will notify the CPU with an **interrupt** when the transfer is complete. This technique for moving data is known as **Direct Memory Access** or **DMA**.

For DMA to take place, three conditions need to be met:

* You need a method to connect the I/O module and memory – usually a bus or a channel does this.
* The I/O module must be capable of reading from or writing to memory. That is, it must be able to load the **Memory Address Register** and read and write to the **Memory Data Register**.
* There must be a means of avoiding a conflict between the CPU and the I/O module. Therefore specific memory addresses can only be used by one I/O device at a time. These reserved memory locations are also referred to **buffers**.

The procedure below illustrates how the CPU initiates the DMA process of writing data stored in memory to a disk drive:

* I/O service program uses programmed I/O to send the required **four pieces of data** to the disk controlling I/O module: 1. location (address) of the block in memory, 2. location where the data is to be stored on the device, 3. size of the block and 4. direction of the transfer (read or write)
* I/O service program sends a ready message to the I/O module using programmed I/O.
* The DMA process now takes place outside the control of the CPU, I/O service program and the application program that requested the I/O.
* When the transfer is complete, the I/O module sends an interrupt to the CPU, where the interrupt handler returns control to the application program requesting the I/O.

There are various ways to interconnect the CPU, memory and I/O devices. All you need is the five basic components: CPU, Memory, I/O module(s), I/O device(s) and buses. Conceptually they look like this.

**I/O Device**

**I/O**

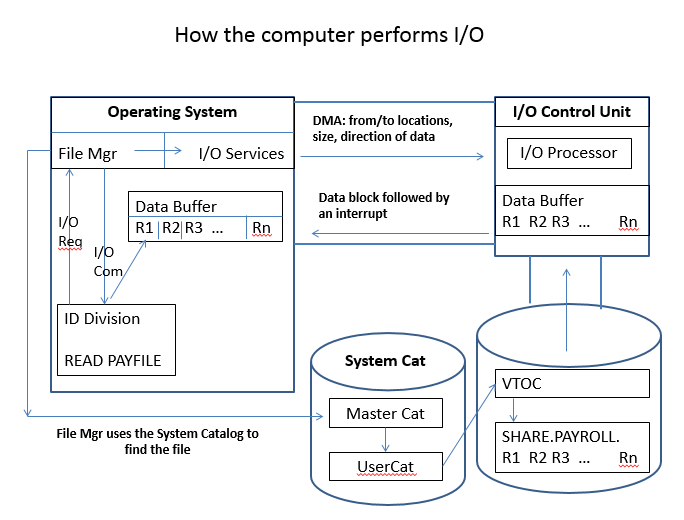
**Module**

## Memory

## CPU

**Buses**

Below is an illustration for how a COBOL program will find a typical Payroll file and return you each data block that contains the records you want to process. We’ll eventually cover all of these components though out in the semester.

****

**Learning Objectives**

After this lab exercise the student will be able to:

* Describe the major concepts embedded in superscalar processing: pipelining instructions, the instruction unit/execute unit model
* Explain the issues with handling out-of-order processing
* Describe input/output operations specifically direct memory access (DMA)

Use RDz and ISPF to create, compile and execute a COBOL program using out of order processing and input/output operations

**Resources - available via Blackboard**

# 1. COBOL programming - tutorials, lectures, exercises, examples: <http://www.csis.ul.ie/cobol/>

2. COBOL Reference Guide, <https://blackboard.syr.edu/bbcswebdav/pid-2619655-dt-content-rid-5650783_1/courses/24364.1131/COBOL%20Reference%20Guide.pdf>

3. JCL Workshop Content Area**Lab Exercise Setup**

In this lab you will use COBOL programs to demonstrate the use of both out of order processing and input and output operations.

1. You should already have the libraries needed to compile, link and run our COBOL programs we need to have places, or libraries, to save our **Source** and **Load** modules and the JCL to execute them. These PDS’ should have been created in a previous lab exercise. Using RDz’s Remote Systems Explorer look for the following PDS’.

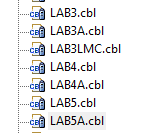
**SUSnnnn.GET239.COBOL** - for your COBOL source code

**SUSnnnn.GET239.JCL** - for your Job Control Language needed compile and execute your programs

**SUSnnnn.GET239.LOADLIB** - for your executable programs

**Out of Order Processing and File I/O**

1. In this lab we are going to write our next COBOL program that will demonstrate out of order processing and input/output operations on disk data sets. Also keep in mind very few programs are written from scratch. So in this lab you will be writing a new program but you will be using a previous lab program as your starting point. From the Remote Systems Explorer, right click on SUSnnnn.GET239.COBOL 🡪 New 🡪 Create Member. In the New Member dialog box name your member **LAB5** then click finish. In your GET239.COBOL library you should see your new LAB5 member:



1. In this lab we will start with last week’s **LAB4A**. Find it in your SUSnnnn.GET239.COBOL library. Double click on both LAB4A and LAB5. Both will open up in editor perspectives. LAB4A will look something like the one below and LAB5 will have nothing in it. So copy and paste LAB4A onto LAB5. No problem if yours looks a little different than mine. You can now close LAB4A. You have just cloned a new program.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**It is important to note that the new PROCEDURE DIVISION instructions needed for LAB5 will need to be coded in a logical order, not in the order explained in the this lab.**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* PROGRAM NAME : LAB4A \*

\* PURPOSE : This program introduces you to more COBOL \*

\* that will allow you to understand how the \*

\* CPU and memory work \*

\* WRITTEN BY : Dave D \*

\* DATE WRITTEN : September 2014 \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ID DIVISION.

PROGRAM-ID. LAB4A.

AUTHOR. Dave D

DATE-WRITTEN. September 2015.

DATE-COMPILED. CURRENT-DATE.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ENVIRONMENT DIVISION.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

DATA DIVISION.

FILE SECTION.

WORKING-STORAGE SECTION.

01 WS-NAMED-MEMORY-LOCATIONS.

\*\*\*\* EMPLOYEE FIELDS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

05 EMPLOYEE-NAME.

10 EMPLOYEE-FIRST-NAME PIC X(20).

10 EMPLOYEE-LAST-NAME PIC X(30).

05 EMPLOYEE-HIRE-DATE PIC X(8).

05 EMPLOYEE-HIRE-DATE2 REDEFINES EMPLOYEE-HIRE-DATE.

10 EMPLOYEE-HIRE-MM PIC 99.

10 EMPLOYEE-HIRE-DD PIC 99.

10 EMPLOYEE-HIRE-YY PIC 9(4).

\*\*\*\* TOTALS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

05 SERVICE-YEARS PIC 999 VALUE 0.

05 SERVICE-YEARSZ PIC Z99.

05 ANNUAL-SALARY PIC 9(7)V99 VALUE 0.

05 ANNUAL-SALARYZ PIC Z,ZZZ,ZZ9.99.

05 BI-WEEKLY-GROSS-PAY PIC 9(7)V99 VALUE 0.

05 BI-WEEKLY-GROSS-PAYZ PIC Z,ZZZ,ZZZ.99.

\*\*\*\* CURRENT DATE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

05 WS-CURRENT-DATE-DATA.

10 WS-CURRENT-DATE.

15 WS-CURRENT-YY PIC 9(4).

15 WS-CURRENT-MM PIC 99.

15 WS-CURRENT-DD PIC 99.

10 WS-CURRENT-TIME PIC 9(8).

10 WS-DIFF-FROM-GMT PIC S9(04).

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

PROCEDURE DIVISION.

MAIN-PROGRAM.

\*\*\*\* DISPLAY SPLASH PAGE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

DISPLAY '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'.

DISPLAY '\* GET239 Enterprise Technologies LAB4A \*'.

DISPLAY '\* <your name> \*'.

DISPLAY '\* <the current date> \*'.

DISPLAY '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'.

\*\*\*\* MOVE EMPLOYEE DATA TO BE DISPLAYED LATER \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

MOVE 'David' TO EMPLOYEE-FIRST-NAME.

MOVE 'Dischiave' TO EMPLOYEE-LAST-NAME.

MOVE 01011999 TO EMPLOYEE-HIRE-DATE.

\*\*\*\* Use the current date function to separate the MM,DD and YYYY

MOVE FUNCTION CURRENT-DATE TO WS-CURRENT-DATE-DATA.

\*\*\*\* MOVE Calculate years of service \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

SUBTRACT EMPLOYEE-HIRE-YY FROM WS-CURRENT-YY

GIVING SERVICE-YEARS.

MOVE 79000.13 to ANNUAL-SALARY.

DIVIDE ANNUAL-SALARY by 26 GIVING BI-WEEKLY-GROSS-PAY

ROUNDED.

MOVE SERVICE-YEARS TO SERVICE-YEARSZ.

MOVE ANNUAL-SALARY TO ANNUAL-SALARYZ.

MOVE BI-WEEKLY-GROSS-PAY TO BI-WEEKLY-GROSS-PAYZ.

\*\*\*\* DISPLAY EMPLOYEE DATA \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

DISPLAY 'Employee First Name: ' EMPLOYEE-FIRST-NAME.

DISPLAY 'Employee Last Name: ' EMPLOYEE-LAST-NAME.

DISPLAY 'Current Year : ' WS-CURRENT-YY.

DISPLAY 'Year Hired : ' EMPLOYEE-HIRE-YY.

DISPLAY 'Years of Service : ' SERVICE-YEARSZ.

DISPLAY 'Annual Salary : ' ANNUAL-SALARYZ.

DISPLAY 'Bi-weekly Amount : ' BI-WEEKLY-GROSS-PAYZ.

DISPLAY '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'.

DISPLAY '\* End of LAB4A \*'.

DISPLAY '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'.

END-PROGRAM.

STOP RUN.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* END OF THE PROGRAM.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

You now have a working program (assuming you got LAB4A to work) to use as a model for LAB5. In LAB5 we will be deleting most of the instructions from the **PROCEDURE DIVISION** in your old LAB4A and adding new instructions for out of order processing and input/output operations.

To demonstrate out of order processing we are going to create a loop where you will read all records from a **sequential** disk data set. The records in the disk data set are organized sequentially. This means that you need read and process the data set’s records starting at the beginning of the data set reading each record in the physical order they were recorded until you reach the end of the data set.

In COBOL we need to tell your program that it needs to expect a sequential data set. Essentially, we are setting up the input and output environment. We need to identify the file name we wish to process in the **ENVIRONMENT DIVISION**. So we are going to add the following code to your LAB5 program:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ENVIRONMENT DIVISION.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

**Notice that there is no period after CRIMEIN or REPORTOT**

SELECT CRIME-FILE-IN ASSIGN TO **CRIMEIN**

ORGANIZATION IS SEQUENTIAL.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

SELECT REPORT-OUT ASSIGN TO **REPORTOT**

ORGANIZATION IS SEQUENTIAL.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

The **FILE-CONTROL** paragraph of the **INPUT-OUTPUT SECTION** is where you **SELECT** (or identify) the file you wish to read. Here I called the file **CRIME-FILE-IN**. So later in the program I’m going to describe the records stored in this file. I then have to link this file to the physical file located on disk. So in my execute JCL I’m going to have a **DD** (Data Definition) statement that has a name of **CRIMEIN** as well as the name of the actual disk data set I plan to read. The name of **CRIMEIN** is arbitrary. I could have easily called it **INPUT** or **FILE1**.

//CRIMEIN DD DSN=SHARE.CHICAGO.CRIME09,DISP=SHR

Notice that my file organization is sequential. So I am planning to read each record in the order they were written to this data set. Also, you’ll notice that I’m creating an output dataset as well called **REPORT-OUT** and I’m pointing to a JCL DD statement whose DD name is **REPORTOT.**

//REPORTOT DD SYSOUT=\*

So in summary, we use the **ENVIRONMENT DIVISION** to tell the computer that we want two datasets. Both of which the records are recorded in sequential order. It is here we point our logical file references (CRIME-IN and REPORT-OUT) to the physical files referenced in the JCL. We still need to tell the computer that one file will be used as input and one will be used as output.

Next, in the **DATA DIVISION** is where we describe our data records. This is very similar to the way we described our variables in our WORKING-STORAGE SECTION. Only we will have a **FILE SECTION**. **The FILE SECTION has to appear in your program before the WORKING-STORAGE SECTION.** You do not need to type this code. Go to SHARE.GET239.COBOL(LAB5REC) and copy the contents into your program. It should look like the following. **Be sure not to change any of this code because it mirrors the records that are recorded on the file**. So changing it will give unpredictable results.

In the **FILE SECTION** of the **DATA DIVISION** you describe the data records you wish to process. We named the file, **CRIME-FILE-IN,** in our SELECT statement so we need to also name it CRIME-FILE-IN here in the FD or File Definition statement. It is after the FD that we describe our CRIME-RECORD. Since most records in large organization have already been defined you can just copy their description (or layout) into your program as needed.

FD CRIME-FILE-IN.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* READ ME: Do not change this record layout

\* Chicago City Crime record - Length 249

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

01 CC-CRIME-RECORD.

05 CC-CASE-NUMBER PIC X(8).

05 CC-DATE-TIME PIC X(15).

05 CC-ADDRESS PIC X(40).

05 CC-IUCR PIC X(4).

05 CC-PRIMARY-CRIME PIC X(30).

05 CC-CRIME-DESC PIC X(50).

**This is your input record layout**

05 CC-LOCATION PIC X(50).

05 CC-ARREST PIC X.

05 CC-DOMESTIC PIC X.

05 CC-BEAT PIC X(4).

05 CC-WARD PIC X(2).

05 CC-FBI-CODE PIC X(3).

05 CC-X-COORD PIC X(7).

05 CC-Y-COORD PIC X(7).

05 CC-YEAR PIC X(4).

05 CC-LAT PIC X(11).

05 CC-LONG PIC X(12).

\*\*\*\*\*\*\* End of Chicago City Crime Record \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

FD REPORT-OUT.

01 REPORT-REC.

05 RP-CASE-NUMBER PIC X(8).

05 PIC X.

**This is your output record layout**

05 RP-DATE-TIME PIC X(15).

05 PIC X.

05 RP-ADDRESS PIC X(40).

05 PIC X.

05 RP-IUCR PIC X(4).

05 PIC X.

05 RP-PRIMARY-CRIME PIC X(30).

05 PIC X.

05 RP-CRIME-DESC PIC X(50).

05 PIC X.

05 RP-LOCATION PIC X(50).

05 PIC X.

05 RP-ARREST PIC X.

05 PIC X.

05 RP-DOMESTIC PIC X.

05 PIC X.

05 RP-BEAT PIC X(4).

05 PIC X.

05 RP-WARD PIC X(2).

05 PIC X.

05 RP-FBI-CODE PIC X(3).

05 PIC X.

05 RP-X-COORD PIC X(7).

05 PIC X.

05 RP-Y-COORD PIC X(7).

05 PIC X.

05 RP-YEAR PIC X(4).

05 PIC X.

05 RP-LAT PIC X(11).

05 PIC X.

05 RP-LONG PIC X(12).

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

WORKING-STORAGE SECTION.

1. So far from the Environment Division, your program so far should look something like this:

ENVIRONMENT DIVISION.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

**Notice that the SELECT statement file name is the same as the FD file name**

SELECT CRIME-FILE-IN ASSIGN TO CRIMEIN

ORGANIZATION IS SEQUENTIAL.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

SELECT REPORT-OUT ASSIGN TO REPORTOT

ORGANIZATION IS SEQUENTIAL.

**Here you should see the file definition and the record description. Later you’ll see the connection to the physical file when we connect the PAYIN in the SELECT statement to the DD name in the JCL.**

**Notice the comments that state you should not add, change or delete any part of this record description since it maps directly to the data stored on disk. Doing so will yield unpredictable results.**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

DATA DIVISION.

FILE SECTION.

FD CRIME-FILE-IN.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* READ ME: Do not change this record layout

\* Chicago City Crime record - Length 249

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

01 CC-CRIME-RECORD.

05 CC-CASE-NUMBER PIC X(8).

05 CC-DATE-TIME PIC X(15).

05 CC-ADDRESS PIC X(40).

05 CC-IUCR PIC X(4).

05 CC-PRIMARY-CRIME PIC X(30).

05 CC-CRIME-DESC PIC X(50).

05 CC-LOCATION PIC X(50).

05 CC-ARREST PIC X.

05 CC-DOMESTIC PIC X.

05 CC-BEAT PIC X(4).

05 CC-WARD PIC X(2).

05 CC-FBI-CODE PIC X(3).

05 CC-X-COORD PIC X(7).

05 CC-Y-COORD PIC X(7).

05 CC-YEAR PIC X(4).

05 CC-LAT PIC X(11).

05 CC-LONG PIC X(12).

\*\*\*\*\*\*\* End of Chicago City Crime Record \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

FD REPORT-OUT.

01 REPORT-REC.

05 RP-CASE-NUMBER PIC X(8).

05 PIC X.

05 RP-DATE-TIME PIC X(15).

05 PIC X.

05 RP-ADDRESS PIC X(40).

05 PIC X.

05 RP-IUCR PIC X(4).

05 PIC X.

05 RP-PRIMARY-CRIME PIC X(30).

05 PIC X.

05 RP-CRIME-DESC PIC X(50).

05 PIC X.

05 RP-LOCATION PIC X(50).

05 PIC X.

05 RP-ARREST PIC X.

05 PIC X.

05 RP-DOMESTIC PIC X.

05 PIC X.

05 RP-BEAT PIC X(4).

05 PIC X.

05 RP-WARD PIC X(2).

05 PIC X.

**Notice that the FD statements and record descriptions are inserted before the working storage section**

05 RP-FBI-CODE PIC X(3).

05 PIC X.

05 RP-X-COORD PIC X(7).

05 PIC X.

05 RP-Y-COORD PIC X(7).

05 PIC X.

05 RP-YEAR PIC X(4).

05 PIC X.

**When setting up loops for iterative (think out of order here) processing you need a switch that you can check to see when you are done reading a file. Here we can use an 88 level that is a special way to define values to your program. The 88 level name EOF which I am using for end of file can be tested using an IF or a PERFORM statement later in this lab**

05 RP-LAT PIC X(11).

05 PIC X.

05 RP-LONG PIC X(12).

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

WORKING-STORAGE SECTION.

01 WS-NAMED-MEMORY-LOCATIONS.

\*\*\*\* SWITCHES \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

05 EOF-SWITCH PIC X VALUE 'N'.

88 EOF VALUE 'Y'.

1. In the **PROCEDURE DIVISION** we will be making some changes that will allow you to read a sequential file and we’ll be using out of order processing to do so. There are a number of ways to change the “direction” of your program’s logic; that is to say, to change the sequence that the instructions are executed. There are a variety of different instructions in COBOL that can allow you to control the flow of your program. Since out of order processing can be handled a number of ways, we will start slowly by just introducing two instructions that can alter your program’s flow, the **PERFORM** and the **IF/THEN/ELSE** statements.

The first instruction we’ll discuss is the **PERFORM**. The PERFORM statement transfers control to the paragraph(s) specified by the paragraph name after the PERFORM reserved word. When the control reaches the end of the paragraph (or the beginning of a new paragraph) control is returned to the next statement following the PERFORM statement. Let’s look at a couple of examples:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

PROCEDURE DIVISION.

**Notice that I used EOF that I defined above as an 88 level. I initialized it to “N” so my program will keep performing the 300-READ-CRIME-FILE paragraph until I move a “Y” to the EOF-SWITCH that I defined above**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

MAIN-PROGRAM.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

PERFORM 100-DISPLAY-SPLASH-PAGE.

PERFORM 200-OPEN-FILES.

PERFORM 300-READ-CRIME-FILE UNTIL EOF.

PERFORM 400-DISPLAY-SUMMARY-TOTALS.

PERFORM 500-CLOSE-FILES.

STOP RUN.

END-MAIN-PROGRAM.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

100-DISPLAY-SPLASH-PAGE.

DISPLAY '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'.

DISPLAY '\* GET239 Enterprise Technologies LAB5 \*'.

DISPLAY '\* <your name here> \*'.

DISPLAY '\* <the date> \*'.

**I have to open my files before I can read or write to them. The open statement handles the housekeeping required to insure the integrity of the datasets.**

DISPLAY '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

200-OPEN-FILES.

OPEN INPUT CRIME-FILE-IN

OUTPUT REPORT-OUT.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

300-READ-CRIME-FILE.

**Here I set up the loop to read my crime records. I only want to select 10 records so I use the IF statement to do this. Notice that I’m also checking for end of file.**

**Lastly, notice only one period at the end of my END-IF**

READ CRIME-FILE-IN

AT END MOVE 'Y' TO EOF-SWITCH.

IF NOT EOF

ADD 1 TO CRIME-REC-READ

IF CRIME-REC-COUNT-IN < 10

ADD 1 TO CRIME-REC-COUNT-IN

PERFORM 310-PROCESS-CRIME-RECORDS

END-IF

END-IF. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

310-PROCESS-CRIME-RECORDS.

ADD 1 TO CRIME-REC-COUNT-IN.

MOVE CC-CASE-NUMBER TO RP-CASE-NUMBER.

MOVE CC-DATE-TIME TO RP-DATE-TIME.

MOVE CC-ADDRESS TO RP-ADDRESS.

**For each record I select I’m moving the data that I read from my input file to my output file which is actually my report.**

MOVE CC-IUCR TO RP-IUCR.

MOVE CC-PRIMARY-CRIME TO RP-PRIMARY-CRIME.

MOVE CC-CRIME-DESC TO RP-CRIME-DESC.

MOVE CC-LOCATION TO RP-LOCATION.

MOVE CC-ARREST TO RP-ARREST.

MOVE CC-DOMESTIC TO RP-DOMESTIC.

MOVE CC-BEAT TO RP-BEAT.

MOVE CC-WARD TO RP-WARD.

MOVE CC-FBI-CODE TO RP-FBI-CODE.

MOVE CC-X-COORD TO RP-X-COORD.

MOVE CC-Y-COORD TO RP-Y-COORD.

MOVE CC-YEAR TO RP-YEAR.

MOVE CC-LAT TO RP-LAT.

**Then I write the output record**

MOVE CC-LONG TO RP-LONG.

WRITE REPORT-REC.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Here I’m displaying my record counts for my summary totals**

400-DISPLAY-SUMMARY-TOTALS.

MOVE CRIME-REC-READ TO CRIME-REC-READZ.

DISPLAY 'Total Recs Read : ' CRIME-REC-READZ.

MOVE CRIME-REC-COUNT-IN TO CRIME-REC-COUNT-INZ.

DISPLAY 'Total Recs In : ' CRIME-REC-COUNT-INZ.

MOVE CRIME-REC-COUNT-OUT TO CRIME-REC-COUNT-OUTZ.

DISPLAY 'Total Recs Out : ' CRIME-REC-COUNT-OUTZ.

DISPLAY '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'.

DISPLAY '\* End of Crime Reformat Program \*'.

DISPLAY '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Lastly before I end my program I need to close my files so that other programs can used them.**

500-CLOSE-FILES.

CLOSE CRIME-FILE-IN

REPORT-OUT.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* END OF THE PROGRAM.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

In this example, PERFORM 100-DISPLAY-SPLASH-PAGE transfers control to the first instruction in the 100-DISPLAY SPLASH-PAGE paragraph, executes the five DISPLAY instructions until it detects the next paragraph, 200-OPEN-FILES where control is then transferred to the next statement following the PERFORM 100-DISPLAY-SPLASH-PAGE, which is the PERFORM 200-OPEN-FILES. The instructions in the 200-OPEN-FILES paragraph are then executed until the end 200-OPEN-FILES paragraph is detected. Control is then returned to the PERFORM 300-READ-CRIME-FILE UNTIL EOF statement is detected. I think you get the idea how the PERFORM statement works.

So think of the PERFORM statement as going somewhere else in your program to do something then returning to where you came from. It is a neat instruction for making use of re-usable chunks of code.

Also, you can add a condition to the PERFORM statement like we did in the PERFORM 300-READ-CRIME-FILE UNTIL EOF. The UNTIL EOF is a condition that will allow you to create a “loop” then stop looping until the UNTIL condition is satisfied. In this case until the CRIME-FILE reaches the end of file condition. You’ll notice we set the EOF-SWITCH to “Y” in the READ statement when reach the end of the CRIME-FILE.

1. In your LAB5 output be sure to first display the “Splash Page” that identifies the course, lab, and you. But instead of coding it first, figure out a way to execute it by using out of order processing. Use the code sample above to figure out how to do it.

100-DISPLAY-SPLASH-PAGE.

DISPLAY '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'.

DISPLAY '\* GET239 Enterprise Technologies LAB5 \*'.

DISPLAY '\* Dave D \*'.

DISPLAY '\* October 1, 2014 \*'.

DISPLAY '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'.

1. Before we can read or write to a file we must first open it with an **OPEN** statement. The OPEN statement handles a variety of tasks like making sure that the file is available to you, verifies that it is the correct file, making sure that the file isn’t locked for exclusive use by another user or user’s program. Also, when we are done processing we need to close the file with the **CLOSE** statement which updates the file statistics (last time accessed, who accessed it, record counts, block counts etc.) and makes the file available for others to use. The format for the OPEN and CLOSE is pretty simple. You’ll notice the OPEN statement has a qualifier where you can specify a file as **INPUT** only, **OUTPUT** only or **IO,** which is both. Here we want to open our PAYROLL-FILE as input only. That way we can’t accidently write new records to it or accidently change existing records.

**OPEN INPUT CRIME-FILE-IN. <**This needs to be executed before your first record read>

**CLOSE CRIME-FILE-IN. <**This needs to be executed after you have read your last record>

Using out of order processing figure out how to open your CRIME FILE as input. Refer to the code sample above. Keep in mind you’ll need to close this file later on, when it makes sense to do so.

1. Once we open the file we can now read the records. To read a file we use the **READ** statement. The format of a read looks like this:

**When the program detects end of file I use the AT END phrase to move a “Y” to the EOF-SWITCH that I described above**

**READ CRIME-FILE-IN**

**AT END MOVE 'Y' TO EOF-SWITCH.**

The READ reserved word is followed by the file name. It is the same name used in the **SELECT, FD,** **OPEN** and **CLOSE** statements. You’ll notice there is a clause that allows you to test for the end of the file. In this case, you’ll notice I set a switch called EOF-SWITCH which I can test later on. Here you see that when the program detects the end of the PAYROLL-FILE, I move “Y” to my end of file switch, named EOF-SWITCH, which is defined in the WORKING-STORAGE SECTION as follows:

**WORKING-STORAGE SECTION.**

**01 WS-NAMED-MEMORY-LOCATIONS.**

**05 EOF-SWITCH PIC X VALUE 'N'.**

**88 EOF VALUE 'Y'.**

Notice that I used an 88 Level which lets me use the name EOF when testing for the “Y” that signifies the end of file. See if you can determine where the read statement should go in your program. Be sure to execute it using out of order processing.

1. Another useful instruction that can change program flow is the **IF, THEN, ELSE** statement. The IF statement allows you to test the values of your named memory locations, values in input fields from records you read and conditions set by the computer like end of file or certain error conditions that could arise. The format of the IF statement is:

**IF** <field name> operators <field name or literal values>

**THEN <**instruction to be executed if the comparison is true>

**ELSE** <instruction to be executed if the comparison is false.>

For the operators you can use either the English words: **EQUAL, IS EQUAL, IS EQUAL TO, LESS THAN, GREATER THAN** or the mathematical symbols: =, <, >. You can also use the reserved word **NOT** in front of any operator. Here are some examples:

IF MAILBOX-95 IS LESS THAN 0

THEN MOVE 0 TO MAILBOX-95

ELSE ADD MAILBOX-95 TO MAILBOX-99.

**Is the same as:**

IF MAILBOX-95 < 0

THEN MOVE 0 TO MAILBOX-95

ELSE ADD MAILBOX-95 TO MAILBOX-99.

To test a variable defined as an 88 level like the 88 EOF above. You just use the 88 level name after the IF statement. Notice you don’t need any operators. So the PERFORM 700-END-PROGRAM instruction will execute when EOF-SWITCH is equal to “Y”

IF EOF

THEN PERFORM 700-END-PROGRAM.

IF NOT EOF

THEN PERFORM 500-PROCESS-PAYROLL-RECORDS

END-IF.

1. So now let’s put all of the pieces together. Add the code needed to process the records in the CRIME FILE. Remember you need to identify the file in the ENVIRONMENT DIVISION, describe the record in the DATA DIVISION, you need to open the file, read the file, process the records, display the results, close the file and end the program. When you’re done it should look like the program below.

1. So let’s take a look at the whole program. This program is a great example of both **out of order processing** and **input/output control**. The program consists of four perform statements that drive all the processing.

The first perform, PERFORM 100-DISPLAY-SPLASH-PAGE will display the information about the course, the student and the date.

**1**

The second perform opens the CRIME FILE

**2**

The third perform reads the CRIME FILE sequentially reading one record at a time until it gets to the end of the file. The cool thing is that within the 300-READ-CRIME-FILE paragraph is another perform statement that performs the

**3**

**4**

310-PROCESS-CRIME-RECORDS which counts the number of records processed then formats and writes the report records.

When the program reaches the end of the CRIME File it performs 400-DISPLAY-SUMMARY-TOTALS paragraph which displays a few of data from each record

**5**

500-CLOSE-FILES closes both your input and your output files. They are no longer available for your use until you reopen them.

**6**

PROCEDURE DIVISION.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

MAIN-PROGRAM.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**1**

PERFORM 100-DISPLAY-SPLASH-PAGE.

**2**

PERFORM 200-OPEN-FILES.

**3**

PERFORM 300-READ-CRIME-FILE UNTIL EOF.

**5**

PERFORM 400-DISPLAY-SUMMARY-TOTALS.

**6**

PERFORM 500-CLOSE-FILES.

STOP RUN.

END-MAIN-PROGRAM.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**1**

100-DISPLAY-SPLASH-PAGE.

DISPLAY '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'.

DISPLAY '\* GET239 Enterprise Technologies LAB5 \*'.

DISPLAY '\* <your name here> \*'.

DISPLAY '\* <the date> \*'.

DISPLAY '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**2**

200-OPEN-FILES.

OPEN INPUT CRIME-FILE-IN

OUTPUT REPORT-OUT.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**3**

300-READ-CRIME-FILE.

READ CRIME-FILE-IN

AT END MOVE 'Y' TO EOF-SWITCH.

IF NOT EOF

ADD 1 TO CRIME-REC-READ

IF CRIME-REC-COUNT-IN < 10

ADD 1 TO CRIME-REC-COUNT-IN

PERFORM 310-PROCESS-CRIME-RECORDS

END-IF

END-IF. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**4**

310-PROCESS-CRIME-RECORDS.

ADD 1 TO CRIME-REC-COUNT-OUT.

MOVE CC-CASE-NUMBER TO RP-CASE-NUMBER.

MOVE CC-DATE-TIME TO RP-DATE-TIME.

MOVE CC-ADDRESS TO RP-ADDRESS.

MOVE CC-IUCR TO RP-IUCR.

MOVE CC-PRIMARY-CRIME TO RP-PRIMARY-CRIME.

MOVE CC-CRIME-DESC TO RP-CRIME-DESC.

MOVE CC-LOCATION TO RP-LOCATION.

MOVE CC-ARREST TO RP-ARREST.

MOVE CC-DOMESTIC TO RP-DOMESTIC.

MOVE CC-BEAT TO RP-BEAT.

MOVE CC-WARD TO RP-WARD.

MOVE CC-FBI-CODE TO RP-FBI-CODE.

MOVE CC-X-COORD TO RP-X-COORD.

MOVE CC-Y-COORD TO RP-Y-COORD.

MOVE CC-YEAR TO RP-YEAR.

MOVE CC-LAT TO RP-LAT.

MOVE CC-LONG TO RP-LONG.

WRITE REPORT-REC.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**5**

400-DISPLAY-SUMMARY-TOTALS.

MOVE CRIME-REC-READ TO CRIME-REC-READZ.

DISPLAY 'Total Recs Read : ' CRIME-REC-READZ.

MOVE CRIME-REC-COUNT-IN TO CRIME-REC-COUNT-INZ.

DISPLAY 'Total Recs In : ' CRIME-REC-COUNT-INZ.

MOVE CRIME-REC-COUNT-OUT TO CRIME-REC-COUNT-OUTZ.

DISPLAY 'Total Recs Out : ' CRIME-REC-COUNT-OUTZ.

DISPLAY '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'.

DISPLAY '\* End of Crime Reformat Program \*'.

DISPLAY '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**6**

500-CLOSE-FILES.

CLOSE CRIME-FILE-IN

REPORT-OUT.

1. After you compile and link LAB5 you are ready to run it. You will need to change your LAB4 JCL. The only difference between running LAB4 and LAB5 is that LAB5 needs two additional JCL DD statements. One for each file. The input file specified by the DD name (data definition) **CRIMEIN**. CRIMEIN is the same name you used in your SELECT statement where you coded **SELECT CRIME-FILE-IN ASSIGN TO CRIMEIN.** CRIME-FILE-IN is the internal COBOL name that you as the programmer use to read the file and CRIMEIN is the link to the external name that allows you to process any file you want but using the same program. The actual file name is identified by the “DSN=” data set name parameter. In this case the physical file name is **SHARE.CHICAGO.CRIME09**. The second DD statement is to give your output report identified in your program’s SELECT statement as **REPORT-OUT ASSIGN TO REPORTOT.** This gives your program a place to write your output records.

//SUS0000T JOB (000000),'Dave D',

// NOTIFY=&SYSUID,

// TIME=(,5),

// LINES=500,

// CLASS=A

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\* CREATE A CRIME REPORT of 500 RECORDS FROM 2009 CHICAGO DATA

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//STEP01 EXEC PGM=LAB5

//STEPLIB DD DSN=&SYSUID..GET239.LOADLIB,

// DISP=SHR

//CRIMEIN DD DSN=SHARE.CHICAGO.CRIME09,DISP=SHR **🡨 You need this DD statement**

//REPORTOT DD SYSOUT=\* **🡨 You need this DD statement**

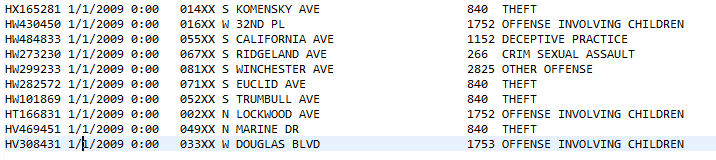
//SYSPRINT DD SYSOUT=\*

//SYSOUT DD SYSOUT=\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. After you compile your LAB5 program use the JCL above to execute it. You should get results like this:

**Crime Report**



**Summary Report**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* GET239 Enterprise Technologies LAB5 \*

\* <your name here> \*

\* <the date> \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Total Recs Read : 391,302

Total Recs In : 10

Total Recs Out : 10

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* End of Crime Record Selection Program \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## Please type your answers to the following questions and submit them with the requested screen prints.

1. **In your LAB5 program which COBOL instructions will cause out of order processing? (6)**

a. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

b. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

c. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

d. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

e. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

f. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **a. Why don’t you need to code the physical input file name in your program? (2) b. Why is it a bad idea to “hard code” the file name in your program? (2)**

a.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

b.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **For 5 extra credit points, use ISPF 🡪Option 3.4 🡪DSname Level 🡪 SHARE.CHICAGO.CRIME09 using the metadata for the crime file how many blocks did your program have to read to retrieve all the records on the file?**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **New Year’s Day No Arrest Report.**

**The new Police Chief for the City of Chicago wants a report for all the crimes committed on New Year’s Day from midnight to 3 am where there were no arrests made. Write a COBOL program named LAB5A to do the following tasks: (10 points)**

1. Display a splash page that has the following items:

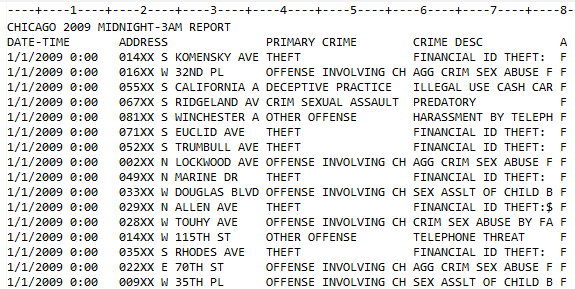
* GET239 Enterprise Technologies LAB5A
* Your name (First & Last)
* Date
* Surround the splash page with asterisks

1. Use the crime record file descriptions from SHARE.GET239.COBOL(LAB5REC)
2. Read the 2009 crime file (SHARE.CHICAGO.CRIME09) sequentially.
3. Count the total number of crime records on the file for 2009.
4. Select for reporting only crimes committed on **January 1, 2009 between (and including) the hours of midnight and 3 am for crimes where no arrests were made (CC-ARREST = “F”)** The time is represented in military time, 0 – 23:59, where 0 is midnight.
5. Count the number of records selected for processing i.e. only for those that meet the criteria identified in letter “e” above.
6. Produce a “detail report” with column headings and show only the date, time, address, primary crime and arrest code fields and count each detail line written to this report.
7. After you have processed all of the records, display the total number of crime records read, total number crime records selected in and written out and the trailer “End of Chicago Midnight to 3 am Report”
8. Submit with the answers to the questions above:

* Your LAB5A source code
* Screen print from your LOADLIB showing your LAB5A load module.
* Output reports from the LAB5A execution that proves it works, be sure to verify your results. See next page for format and counts.

Your output should look like this:

**Detail Report – Notice the headings and that there are no arrests (Arrests = F)**



**Summary Report – with splash page and trailer indicating the end of the report**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Splash page**

\* GET239 Enterprise Technologies LAB5 \*

\* <your name here> \*

\* <the date> \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Record counts**

Total Recs Read : 391,302

Total Recs In : 518

Total Recs Out : 518

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Trailer**

\* End of Chicago Midnight to 3 am Report \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*