INFO3105 Week 4 Part 1

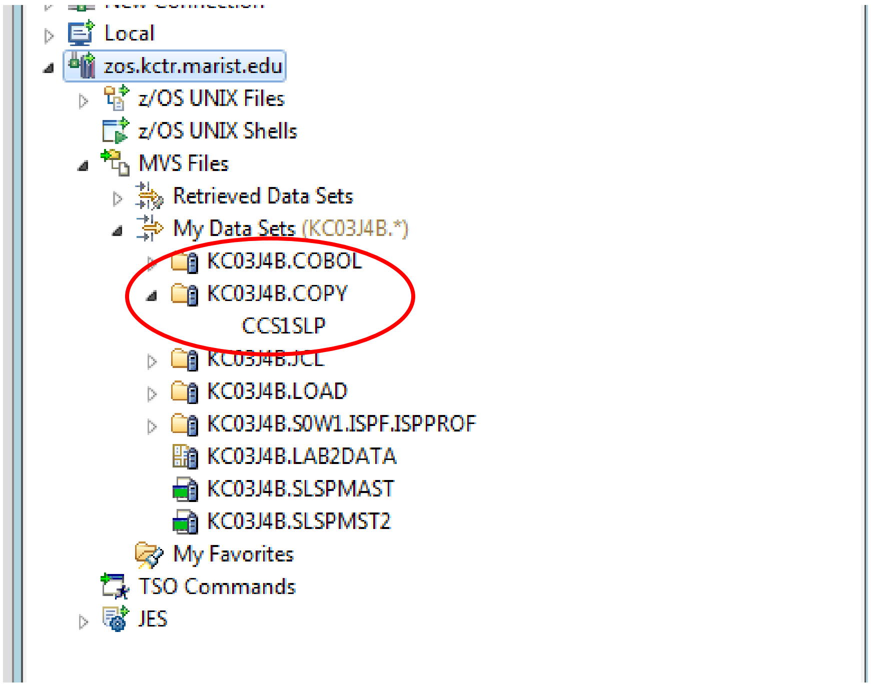
Review

- Summary Report Pseudocode
- Created/read in the SALESMAST or SLINPUT with salesperson data records (sorted by ??)
- Created 1st branch salesperson detail output / including calculations & Branch Totals

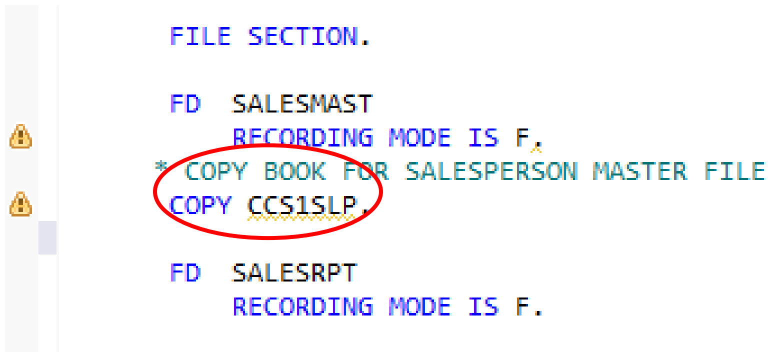
Copy Books

Mainframe programs rarely contain the actual code for master file layouts (anyone know why?). Instead they use an entity called a "**COPYBOOK**". Pages **316-317** of the text cover the usage of a copy books. We are going to place the contents for SALESMAST file definition/layout from last class in a copy book. To do so create a new PDS on the Marist host called COPY and create a new member for this Case1 called **CCS1SLP, t**hen place the file layout code for the sales person master file inside this member (Note: Remember that the file layout – the 01 must still be in column ?? or Area ? and the subsequent 05 level variables in column ?? or Area ? … :

```
                    …
       05 SALESPERSON-NO               PIC 9(5).
              …
       05 SALESPERSON-RETURN-SALES     PIC 9(4)V99.
       05 SALESPERSON-BRANCH-NO        PIC 9(3).
       05 SALESPERSON-COMM-RATE        PIC V9999.
```



And lastly add the **COPY** statement in the A margin of your code and remove the existing 01 for the Salesperson master file (ignore the warnings - the compile should still be able to find and copy the code in from the copybook – as long as you make the subsequent JCL changes):

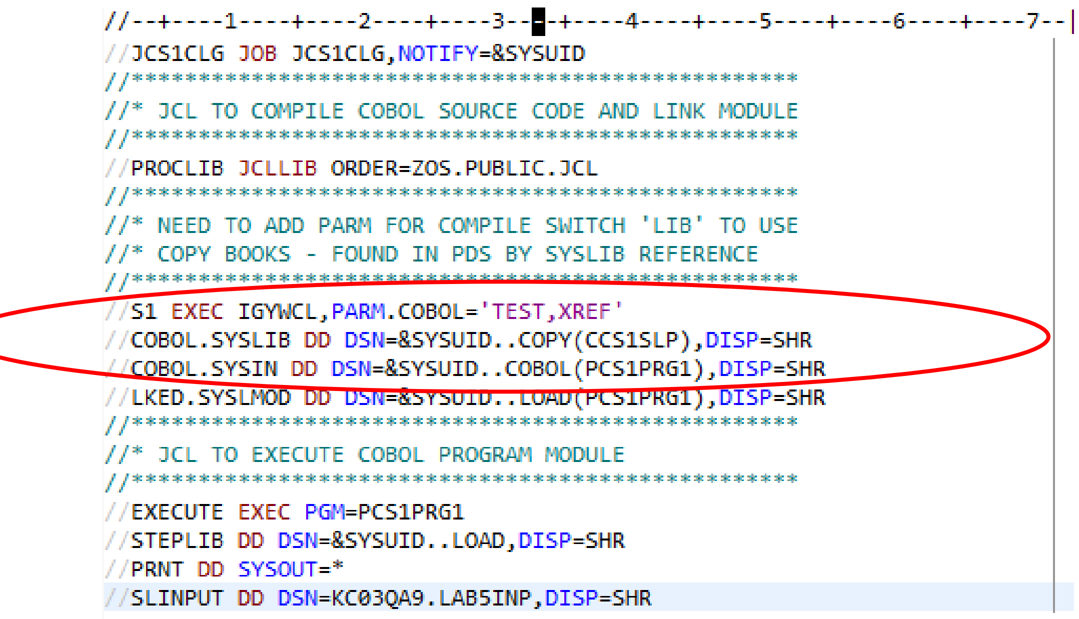


```
       FILE SECTION.

       FD  SALESMAST
           RECORDING MODE IS F.
      * COPY BOOK FOR SALESPERSON MASTER FILE
       COPY CCS1SLP.

       FD  SALESRPT
           RECORDING MODE IS F.
```

There are also a couple of modifications to be made to the JCL as well:



```
//--+----1----+----2----+----3----+----4----+----5----+----6----+----7--|
//JCS1CLG JOB JCS1CLG,NOTIFY=&SYSUID
//***************************************************
//* JCL TO COMPILE COBOL SOURCE CODE AND LINK MODULE
//***************************************************
//PROCLIB JCLLIB ORDER=ZOS.PUBLIC.JCL
//***************************************************
//* NEED TO ADD PARM FOR COMPILE SWITCH 'LIB' TO USE
//* COPY BOOKS - FOUND IN PDS BY SYSLIB REFERENCE
//***************************************************
//S1 EXEC IGYWCL,PARM.COBOL='TEST,XREF'
//COBOL.SYSLIB DD DSN=&SYSUID..COPY(CCS1SLP),DISP=SHR
//COBOL.SYSIN DD DSN=&SYSUID..COBOL(PCS1PRG1),DISP=SHR
//LKED.SYSLMOD DD DSN=&SYSUID..LOAD(PCS1PRG1),DISP=SHR
//***************************************************
//* JCL TO EXECUTE COBOL PROGRAM MODULE
//***************************************************
//EXECUTE EXEC PGM=PCS1PRG1
//STEPLIB DD DSN=&SYSUID..LOAD,DISP=SHR
//PRNT DD SYSOUT=*
//SLINPUT DD DSN=KC03QA9.LAB5INP,DISP=SHR
```

- We add the PARM option for the compiler to tell it to look for a logical name of SYSLIB for the copy book contents
- Then we add a DD for the SYSLIB logical name to point to our actual copy book PDS file at compile time
- Notice too I have already started to change my Program naming convention as per our preferred (like TD) programming naming conventions (this is because Host PDS file members - like program names - are maximum 8 characters long and they need to be unique across the whole company). Details for this are in next class notes (and are not required for this Lab7).

Using 88 Level Elements

When working with files, a common programming technique you will see involves elements defined with the special level of 88. (Notice the variable is the -Switch part that has the single character PIC X and the 88 levels are the boolean based representations of this field). An 88 level clause could be defined as follows:

```
WORKING-STORAGE SECTION.

01  WS-SWITCHES.
    05  WS-SALESMAST-EOF-SWITCH   PIC X      VALUE "N".
        88 WS-SALESMAST-EOF                  VALUE "Y".
        88 WS-SALESMAST-NOT-EOF              VALUE "N".
```

Then you would use these elements something like:

```
    IF WS-IN-STATUS NOT EQUAL "00"
        DISPLAY "FILE ERROR IN-STATUS = ", WS-IN-STATUS
    ELSE
        PERFORM C200-PROCESS-SALESPEOPLE
           UNTIL WS-SALESMAST-EOF
        CLOSE SALESMAST
              SALESRPT
    END-IF.
    STOP RUN.


C200-PROCESS-SALESPEOPLE.

    PERFORM R200-READ-SALESPERSON-RECORD.

    IF WS-SALESMAST-NOT-EOF
       PERFORM U200-SALESPERSON-CALCULATIONS
```

The 88 is like doing an in-line if statement on your switch variable, and makes the code a little easier to read & is therefore a standard. The above IF statement could have been written as:

```
IF WS-SALESMAST-EOF-SWITCH = “N”
```

And then modify the read routine to use the new 88 variable (and remove the INTO clause since you are now defining the input file in the FILE Section. INTO only allowed when reading into Working-Storage Section):

```
R200-READ-SALESPERSON-RECORD.

    READ SALESMAST
        AT END MOVE 'Y' TO
         WS-SALESMAST-EOF-SWITCH.
```

Working with Tables

We have one more topic to cover before we put the finishing requirements on our first case study. Most other programming languages call a group of like elements an array. COBOL calls them **Tables**. Chapter 10 covers tables extensively and shows a number of different ways to use tables. We'll keep things relatively simple and use a single level table with subscripts in the first case study. The section of the chapter that we'll be focusing on can be found on pages **275-284**. We're going to add the following table to our summary report program:

```
01 WS-RATING-VALUES.
   05   FILLER     PIC X(19) VALUE    "00000001999999    *".
   05   FILLER     PIC X(19) VALUE    "20000003999999   **".
   05   FILLER     PIC X(19) VALUE    "40000005999999  ***".
   05   FILLER     PIC X(19) VALUE    "60000007999999 ****".
   05   FILLER     PIC X(19) VALUE    "80000009999999*****".

01 WS-RATING-TABLE REDEFINES WS-RATING-VALUES.
   05 WS-RATING-GROUP OCCURS 5 TIMES.
      10   WS-SALES-RANGE-MIN-VALUE       PIC 9(5)V99.
      10   WS-SALES-RANGE-MAX-VALUE       PIC 9(5)V99.
      10   WS-STAR-RATING                 PIC X(5).
```

We'll give our salespeople a star rating based on their net sales. Their rating will be calculated by seeing what range their net sales falls into (assume no one can exceed $99,999.99 in sales for this table lookup…). For example if a salesperson has sold let's say $56,432.93, they would get a 3 star rating, sales of $15,234.22 would warrant a 1 star rating etc.

Now we've introduced a couple of new key words here, namely **REDEFINES** and **OCCURS**. The redefines clause is used to map the same memory for different uses. Here we have hardcoded the table elements as alphanumeric data, and then redefined them using the correct COBOL syntax. The occurs clause is used to define how many table elements we'll have; here we have set up 5 elements.

To access the table we will set up a subscript field that is defined as:

```
  05   WS-RATING-SUB                  PIC 9(2) COMP.
```

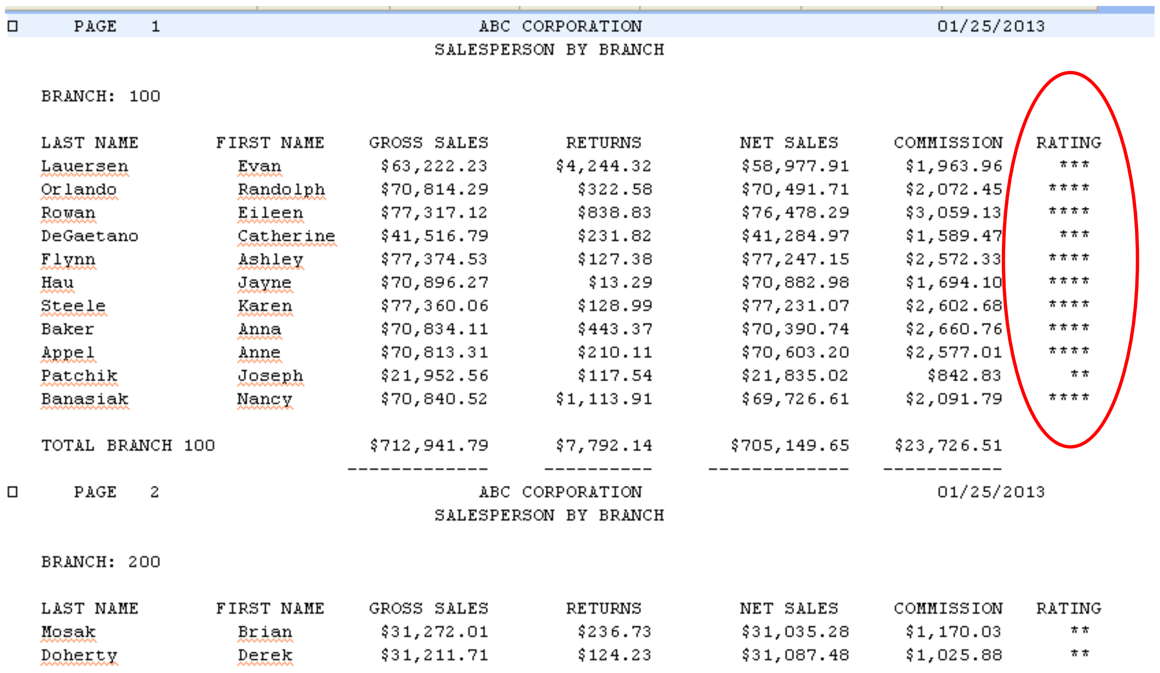
The **COMP** extension (short for COMPUTATIONAL) here means that this will be a binary field, we chose binary as it is the most efficient way to process subscripts (Note, TD has a policy that **all subscripts must be in binary format**, and should be defined multiples of two bytes).

Now how do we search the table? Once we have calculated our net sales, it's just a matter of looping through the table to see where the value fits in. In this case a **PERFORM VARYING** loop will be perfect for our purposes (an example of this can be found on **page 283** of the text):

```
PERFORM WITH TEST AFTER
  VARYING WS-RATING-SUB FROM 1 BY 1 UNTIL
   WS-NET-SALES > WS-SALES-RANGE-MIN-VALUE (WS-RATING-SUB)
   AND
   WS-NET-SALES < WS-SALES-RANGE-MAX-VALUE (WS-RATING-SUB)
   MOVE WS-STAR-RATING (WS-RATING-SUB) TO WS-SL-SLSP-RATING
END-PERFORM.
```

What we're doing here is basically comparing the net sales with the minimum and maximum values in each element of the table. When we come upon the case where net sales is greater than the minimum and less than the maximum, our subscript will give us the correct number for the WS-STAR-RATING element. The clause **WITH TEST AFTER** is important to our task if we do not add this to the perform varying we will get the wrong element. You can stick this perform varying code into the calculation routine you did earlier, just make sure it is done after calculating the net sales (WS-NET-SALES variable in the code above).

To complete the report, we just need to add a column to our detail line output to house the star rating:



```
□     PAGE    1                     ABC CORPORATION                          01/25/2013
                                 SALESPERSON BY BRANCH

   BRANCH: 100

   LAST NAME        FIRST NAME    GROSS SALES       RETURNS         NET SALES     COMMISSION    RATING
   Lauersen           Evan          $63,222.23      $4,244.32       $58,977.91     $1,963.96       ***
   Orlando            Randolph      $70,814.29        $322.58       $70,491.71     $2,072.45      ****
   Rowan              Eileen        $77,317.12        $838.83       $76,478.29     $3,059.13      ****
   DeGaetano          Catherine     $41,516.79        $231.82       $41,284.97     $1,589.47       ***
   Flynn              Ashley        $77,374.53        $127.38       $77,247.15     $2,572.33      ****
   Hau                Jayne         $70,896.27         $13.29       $70,882.98     $1,694.10      ****
   Steele             Karen         $77,360.06        $128.99       $77,231.07     $2,602.68      ****
   Baker              Anna          $70,834.11        $443.37       $70,390.74     $2,660.76      ****
   Appel              Anne          $70,813.31        $210.11       $70,603.20     $2,577.01      ****
   Patchik            Joseph        $21,952.56        $117.54       $21,835.02       $842.83        **
   Banasiak           Nancy         $70,840.52      $1,113.91       $69,726.61     $2,091.79      ****

   TOTAL BRANCH 100                $712,941.79      $7,792.14      $705,149.65    $23,726.51
                                 -------------     ----------    -------------   -----------
□     PAGE    2                     ABC CORPORATION                          01/25/2013
                                 SALESPERSON BY BRANCH

   BRANCH: 200

   LAST NAME        FIRST NAME    GROSS SALES       RETURNS         NET SALES     COMMISSION    RATING
   Mosak              Brian         $31,272.01        $236.73       $31,035.28     $1,170.03        **
   Doherty            Derek         $31,211.71        $124.23       $31,087.48     $1,025.88        **
```

Lab 7 - (2%)

- Add the COPY BOOK to your code
- Change your EOF logic to use an 88 level variable
- Add the table logic and new column to the report's detail area.
- Submit your **entire spool file** as a text file to the submissions dropbox (**please** no Zip/compressed files)
- … And : Read pages **202-212** of the text