INFO3105 Week 4 Part 2

Review

- Copy Books
- Tables
- Subscripts

Finishing up Case 1

Now that we have most of the case study coded we'll be use **some** standards for our mainframe code that are specified by TD's mainframe programming group in a document called "**Cobol Standards and Guidelines**", we'll make these changes now and I'll be looking for adherence to these standards at evaluation time.

We'll start at the top of a typical program and work our way down through the various sections of our code.

Program Id. Programs should start with a **P** and then an application name and if there is more than one program that makes up the application some distinguishing characters. So in our case I'd like you to call the program **PCS1PRG1** (CS1 would be the application, PRG1 the first program).

Program Description should follow a layout similar to what is seen here, the text book calls this a **flower box** (see **page 669** of the text):

```
******************************************************************
*                 YOUR COMPANY NAME HERE                         *
*                                                                *
*               COBOL Z/OS PROGRAM DESCRIPTION                   *
******************************************************************
```

Program description.
Write 2 to 3 sentences briefly describing the purpose or function of the program.

```
* PROGRAM DESCRIPTION:                                           *
*                                                                *
```

Supply a complete list of the input and output files, used by the program. Delete the section that is not applicable.

```
* INPUT DD  NAME    FILE IDENTIFIER        FILE DESCRIPTION     *
* --------------    ---------------        ----------------     *
*                                                                *
*                                                                *
* OUTPUT DD  NAME   FILE IDENTIFIER        FILE DESCRIPTION     *
* ----------------  ---------------        ----------------     *
*                                                                *
*                                                                *
* COPYBOOKS               DESCRIPTION                            *
* ---------               -----------                            *
*                                                                *
******************************************************************
```

Program change log.

Include program change log here; acceptance test number, the programmer who made the change, the implementation date and a short narrative explaining the change. This info is critical for those on call :

```
*****************************************************************
*                    C H A N G E    L O G                       *
*****************************************************************
*                                                               *
* VER.  WRITTEN/CHANGED BY        IMPLEMENTATION                *
* NO.  (FULL 1ST AND LAST NAME)  DATE (MMM/DD/YYYY)             *
* ---- -----------------------  -----------------               *
*                                                               *
* DESCRIPTION OF CHANGE:                                        *
* ---------------------                                         *
*                                                               *
*****************************************************************
```

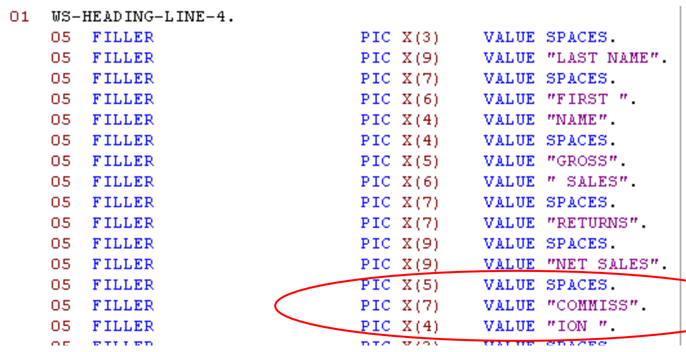
COPYBOOKS – are prefixed with a **C** then 3 chars for the application name so in our case it would be CCS1 then 3 distinguishing letters like SLP resulting in a copy book with the name **CCS1SLP**

JCL – prefixed with J
JCL follows similar naming convention for us it could be **JCS1CLG** (jcl for case1 compile link and go)

Moving on to the **Data Division** we want to where possible, **use value clauses** in the working-storage section instead of using moves in the procedure division to accomplish initialization.

Working Storage Fields
All fields in working storage should be prefixed with WS

Align all PIC clauses.
This one is purely cosmetic, and more than likely you will need to break things up over a number of lines, or the change the alignment from 1 section to another … for instance look at how the word commission is broken up below:



```
01  WS-HEADING-LINE-4.
    05  FILLER                    PIC X(3)     VALUE SPACES.
    05  FILLER                    PIC X(9)     VALUE "LAST NAME".
    05  FILLER                    PIC X(7)     VALUE SPACES.
    05  FILLER                    PIC X(6)     VALUE "FIRST ".
    05  FILLER                    PIC X(4)     VALUE "NAME".
    05  FILLER                    PIC X(4)     VALUE SPACES.
    05  FILLER                    PIC X(5)     VALUE "GROSS".
    05  FILLER                    PIC X(6)     VALUE " SALES".
    05  FILLER                    PIC X(7)     VALUE SPACES.
    05  FILLER                    PIC X(7)     VALUE "RETURNS".
    05  FILLER                    PIC X(9)     VALUE SPACES.
    05  FILLER                    PIC X(9)     VALUE "NET SALES".
    05  FILLER                    PIC X(5)     VALUE SPACES.
    05  FILLER                    PIC X(7)     VALUE "COMMISS".
    05  FILLER                    PIC X(4)     VALUE "ION ".
```

COMP-3 a.k.a Packed Decimal.

Avoid the use of **unpacked numeric items** (i.e.: PIC S999), as the compiler will generate instructions to pack them on any compare or move instruction. Use numeric display fields for display purposes only, never for computations.

One of the nuances of programming on the mainframe is its use of packed decimals. This originated from the need to save bytes of data wherever possible. A packed decimal representation stores two decimal digits in one byte. A packed decimal representation stores decimal digits in each "nibble" of a byte. Each byte has two nibbles, and each nibble is indicated by a hexadecimal digit. For example, the value 23 would be stored in two nibbles, using the hexadecimal digits 2 and 3. The sign indication is dependent on your operating environment. On an IBM mainframe, the sign is indicated by the last half of the last byte (*or high memory address*). For explicitly signed fields the "**C**" indicates **a positive** value and "**D**" indicates **a negative** value. For unsigned (*or implied positive*) fields the "F" indicates a positive value.

The mainframe can perform arithmetic functions on packed-decimal fields without having to convert the format. Storing numeric values in a packed-decimal format may save a significant amount of storage space. For example, on the mainframe the value 12,345 would be five (5) bytes in length *(i.e. x'F1F2F3F4F5')*. If the same information is stored in a packed-decimal (*i.e. USAGE IS COMP-3*) the field would be three (3) bytes in length *(i.e. x'12345F')*. Doesn't sound like much but if you're dealing with large programs, and thousands of them it adds up. So wherever possible change the Data Division's data types to COMP-3. Pages **206-211** of the text discusses using the COMP-3 notation

So if the working storage field is currently defined as:

```
    05  WS-PAGE-COUNT     PIC S9(3)  VALUE ZERO.
```

We want to change it to:

```
    05  WS-PAGE-COUNT     PIC S9(3)  COMP-3 VALUE ZERO.
```

And again what this does internally is store a number as: 123C (two bytes) instead of 1C2C3C (3 bytes) – The C here indicates it's a positive number. The textbook on **page 215** shows that the COMP-3 can be placed on individual items or at the group level.

See an example in FOL Content of Hex display of Packed Signed Positive 19,876.43 amount.

The following is a table that shows the actual field sizes (*for a COMP-3 or packed-decimal*) based on the number of digits specified in the picture clause.

# of Digits	Picture Clause	Field Size	Value	Positive *(Hex)*	Negative *(Hex)*
1	PIC S**9**	**1**	1	x'1C'	x'1D'
2	PIC S**99**	**2**	12	x'012C'	x'012D'
3	PIC S**999**	**2**	123	x'123C'	x'123D'
4	PIC S9(**4**)	**3**	1234	x'01234C'	x'01234D'
5	PIC S9(**5**)	**3**	12345	x'12345C'	x'12345D'
6	PIC S9(**6**)	**4**	123456	x'0123456C'	x'0123456D'
7	PIC S9(**7**)	**4**	1234567	x'1234567C'	x'1234567D'
8	PIC S9(**8**)	**5**	12345678	x'012345678C'	x'012345678D'
9	PIC S9(**9**)	**5**	123456789	x'123456789C'	x'123456789D'
10	PIC S9(**10**)	**6**	1234567890	x'01234567890C'	x'01234567890D'
11	PIC S9(**11**)	**6**	12345678901	x'12345678901C'	x'12345678901D'
12	PIC S9(**12**)	**7**	123456789012	x'0123456789012C'	x'0123456789012D'
13	PIC S9(**13**)	**7**	1234567890123	x'1234567890123C'	x'1234567890123D'
14	PIC S9(**14**)	**8**	12345678901234	x'012345678901234C'	x'012345678901234D'
15	PIC S9(**15**)	**8**	123456789012345	x'123456789012345C'	x'123456789012345D'
16	PIC S9(**16**)	**9**	1234567890123456	x'01234567890123456C'	x'01234567890123456D'
17	PIC S9(**17**)	**9**	12345678901234567	x'12345678901234567C'	x'12345678901234567D'
18	PIC S9(**18**)	**10**	123456789012345678	x'0123456789012345678C'	x'0123456789012345678D'

Using COMP Values

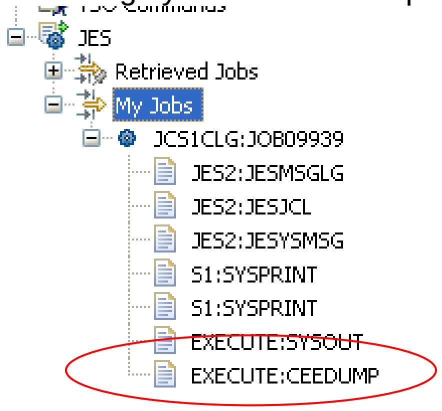
Like COMP-3 there are certain times where we want to be as efficient as possible when using table lookups, you are now to use COMP **for subscripts and indexes**. We looked at this last class and like COMP-3 you can designate at the individual item or group level. Make sure your subscript for our table lookup is COMP like this:

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

Note **COMP** fields should be defined in an **even** number of bytes and **COMP-3** fields should be defined in an **odd** number of bytes.

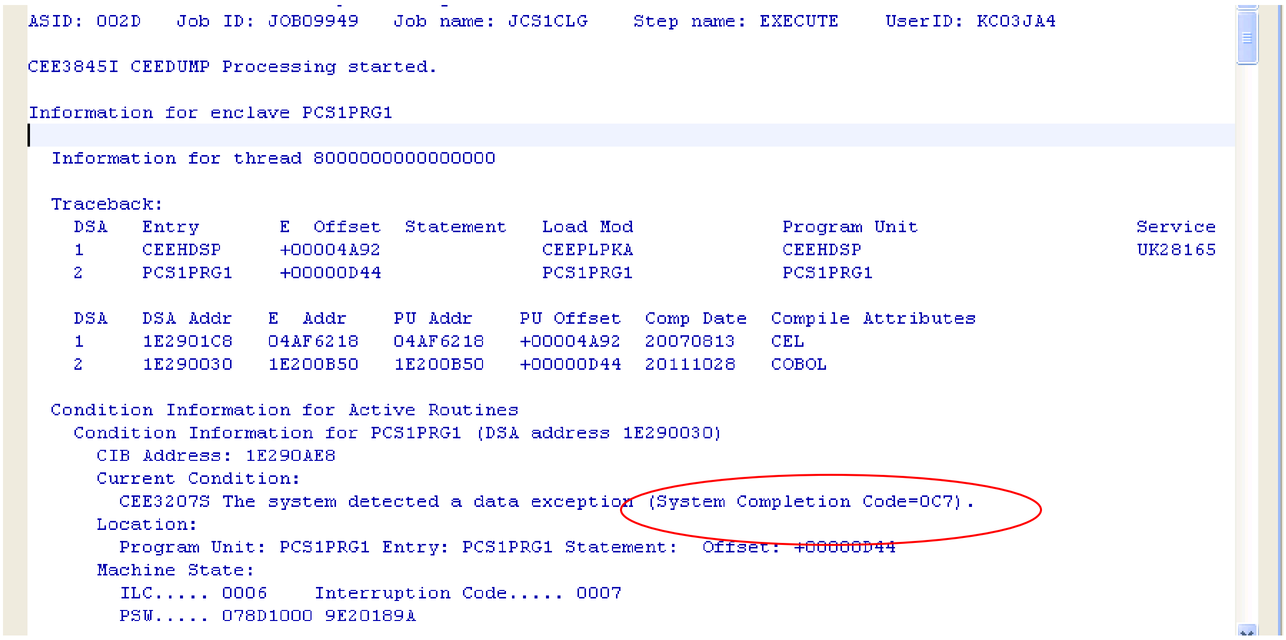
And now System Abends

With the introduction of the COMP-3 and COMP fields in the previous section you may start to see your program come to an Abnormal End (ABEND). Abends are fairly common for novice mainframe programmers and there are some things you can do to help their resolution. First off let's see what



one looks like:

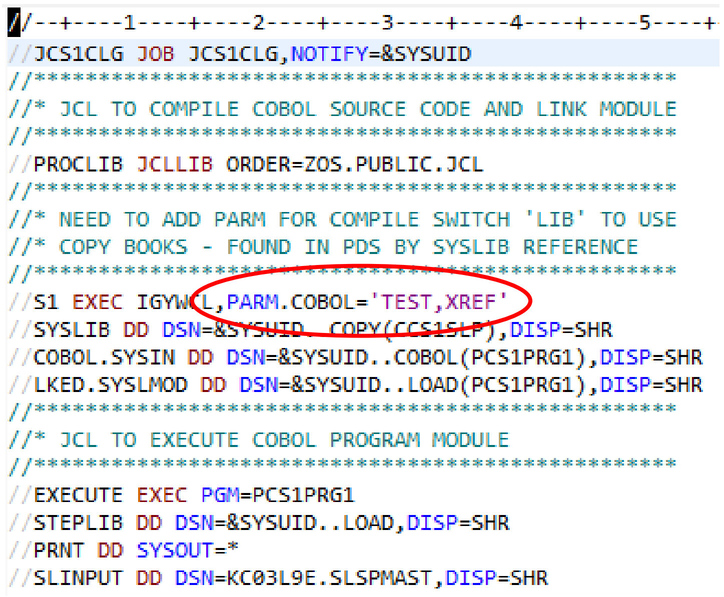
The first thing you will notice is that you didn't get any printed output but instead a CEEDUMP file. This indicates a problem and the system gives you a copy of the status of memory when it abended. Looking inside this file you will see something like:



```
ASID: 002D   Job ID: JOB09949   Job name: JCS1CLG    Step name: EXECUTE    UserID: KC03JA4

CEE3845I CEEDUMP Processing started.

Information for enclave PCS1PRG1

  Information for thread 8000000000000000

  Traceback:
    DSA   Entry       E  Offset   Statement   Load Mod              Program Unit                   Service
    1     CEEHDSP     +00004A92               CEEPLPKA              CEEHDSP                        UK28165
    2     PCS1PRG1    +00000D44               PCS1PRG1              PCS1PRG1

    DSA   DSA Addr   E  Addr    PU Addr    PU Offset  Comp Date   Compile Attributes
    1     1E2901C8   04AF6218   04AF6218   +00004A92  20070813    CEL
    2     1E290030   1E200B50   1E200B50   +00000D44  20111028    COBOL

  Condition Information for Active Routines
    Condition Information for PCS1PRG1 (DSA address 1E290030)
      CIB Address: 1E290AE8
      Current Condition:
        CEE3207S The system detected a data exception (System Completion Code=0C7).
      Location:
        Program Unit: PCS1PRG1 Entry: PCS1PRG1 Statement:   Offset: +00000D44
      Machine State:
        ILC..... 0006     Interruption Code..... 0007
        PSW..... 078D1000 9E20189A
```

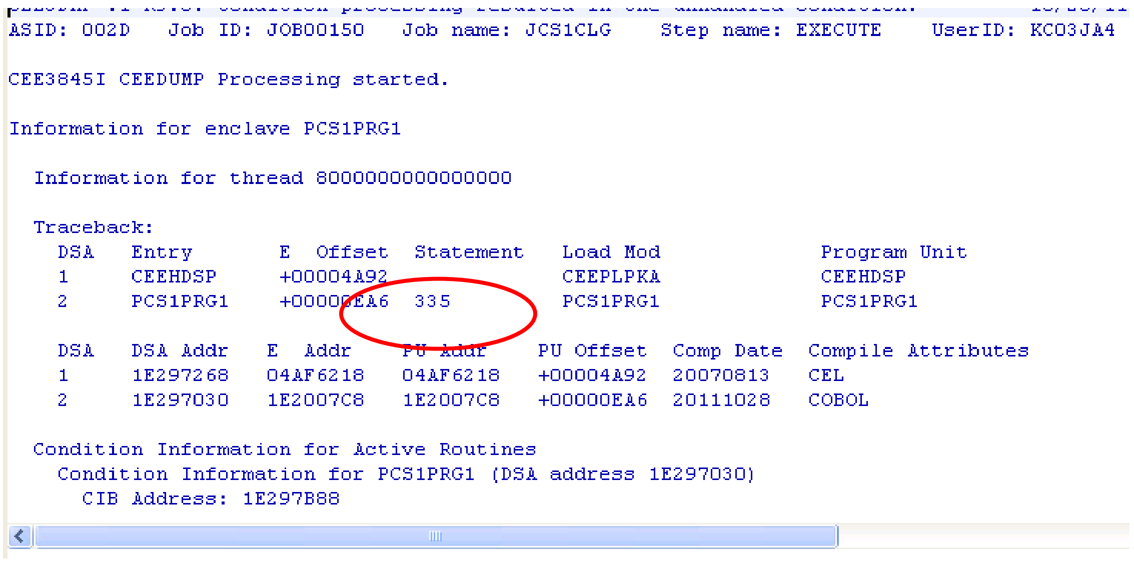
The only thing we have to go by is the **0C7** error message (see **page 545** of the text). This is a common **data exception** (called sock 7 by programmers) which typically you'll find that when putting some non-numeric data into a packed-decimal field. Now this tells us what the error is but it doesn't tell us where it occurred. We need to modify our JCL so we can get some more information from the dump to track it down.

We'll add 2 compiler options (TEST,XREF) to the JCL to tell the system to give us the referenced line number (Statement) that the program failed on, and then we can figure out what is going on:



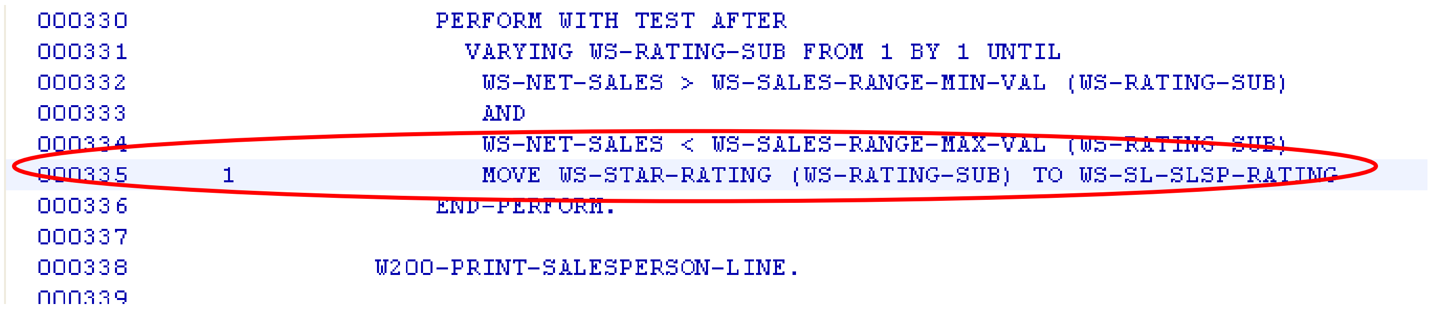
```
//--+----1----+----2----+----3----+----4----+----5----+
//JCS1CLG JOB JCS1CLG,NOTIFY=&SYSUID
//*****************************************************
//* JCL TO COMPILE COBOL SOURCE CODE AND LINK MODULE
//*****************************************************
//PROCLIB JCLLIB ORDER=ZOS.PUBLIC.JCL
//*****************************************************
//* 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'
//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=KC03L9E.SLSPMAST,DISP=SHR
```

Now when the program abends we'll get the statement from the listing that it abended on:



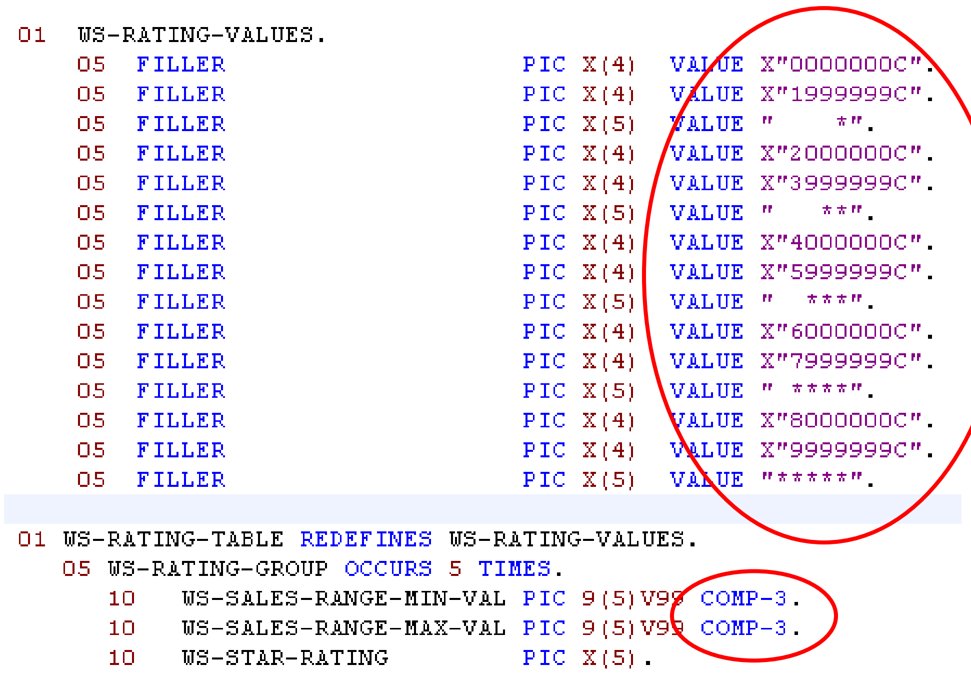
```
ASID: 002D   Job ID: JOB00150   Job name: JCS1CLG    Step name: EXECUTE    UserID: KC03JA4

CEE3845I CEEDUMP Processing started.

Information for enclave PCS1PRG1

  Information for thread 8000000000000000

  Traceback:
    DSA   Entry       E  Offset  Statement   Load Mod             Program Unit
    1     CEEHDSP     +00004A92              CEEPLPKA             CEEHDSP
    2     PCS1PRG1    +00000EA6  335         PCS1PRG1             PCS1PRG1

    DSA   DSA Addr   E  Addr    PU Addr    PU Offset  Comp Date  Compile Attributes
    1     1E297268   04AF6218   04AF6218   +00004A92  20070813   CEL
    2     1E297030   1E2007C8   1E2007C8   +00000EA6  20111028   COBOL

  Condition Information for Active Routines
    Condition Information for PCS1PRG1 (DSA address 1E297030)
      CIB Address: 1E297B88
```

Now we know that the program crashed on line 335. If we look at this in the Compile output / source listing we see the following:



```
000330                     PERFORM WITH TEST AFTER
000331                       VARYING WS-RATING-SUB FROM 1 BY 1 UNTIL
000332                         WS-NET-SALES > WS-SALES-RANGE-MIN-VAL (WS-RATING-SUB)
000333                         AND
000334                         WS-NET-SALES < WS-SALES-RANGE-MAX-VAL (WS-RATING-SUB)
000335         1               MOVE WS-STAR-RATING (WS-RATING-SUB) TO WS-SL-SLSP-RATING
000336                     END-PERFORM.
000337
000338                 W200-PRINT-SALESPERSON-LINE.
000339
```

From here we can see that I'm having a problem in my table, and upon further investigation I see when I changed the fields in my table to comp-3 as per our specs but didn't account for the data. To resolve this I needed to redo the data for the table (you'll have to do this as well, basically we're changing the data to packed literals in hex)



```
 01  WS-RATING-VALUES.
     05  FILLER                  PIC X(4)  VALUE X"0000000C".
     05  FILLER                  PIC X(4)  VALUE X"1999999C".
     05  FILLER                  PIC X(5)  VALUE "    *".
     05  FILLER                  PIC X(4)  VALUE X"2000000C".
     05  FILLER                  PIC X(4)  VALUE X"3999999C".
     05  FILLER                  PIC X(5)  VALUE "   **".
     05  FILLER                  PIC X(4)  VALUE X"4000000C".
     05  FILLER                  PIC X(4)  VALUE X"5999999C".
     05  FILLER                  PIC X(5)  VALUE "  ***".
     05  FILLER                  PIC X(4)  VALUE X"6000000C".
     05  FILLER                  PIC X(4)  VALUE X"7999999C".
     05  FILLER                  PIC X(5)  VALUE " ****".
     05  FILLER                  PIC X(4)  VALUE X"8000000C".
     05  FILLER                  PIC X(4)  VALUE X"9999999C".
     05  FILLER                  PIC X(5)  VALUE "*****".

 01 WS-RATING-TABLE REDEFINES WS-RATING-VALUES.
    05 WS-RATING-GROUP OCCURS 5 TIMES.
       10   WS-SALES-RANGE-MIN-VAL PIC 9(5)V99 COMP-3.
       10   WS-SALES-RANGE-MAX-VAL PIC 9(5)V99 COMP-3.
       10   WS-STAR-RATING         PIC X(5).
```

Procedure Division Notes.

- An **Initialization routine** (if required) should be performed at the beginning of the program to initialize any required working storage - eg. get current date, etc. (Note it is a best practice to initialize constants using VALUE clauses in working storage).
- Use a single OPEN statement for opening all the files.
- Explicitly close all open files.
- Whenever possible, perform the initial read in the initialization paragraph before entering the main processing loop
- Only use Periods (.) where they are required - at the end of a Paragraph name, at the end of a paragraph, and avoid periods on ALL other statements like MOVES and especially IF ELSE structures (use END-IF). NOTE : A period will END all open IF statements … so nested IF 's with a period may compile … but look at the Nesting Level # left of the Cobol code in the Compile Listing output Sysprint (Notice above the Star-rating PERFORM Until is a 1 level nested structure … & compare/contrast with your Control BREAK paragraph will be multiple nested levels deep as shown by the # beside the Compile Listing). This is one place to look if your control break logic is not producing the Record processing logic you expect
- For calculations, use COMPUTE rather than multiple ADD, SUBTRACT, MULTIPLY, and DIVIDE commands.
  - **Note**: Add the keyword ROUNDED when calculating the COMMISSION eg. WS-COMMISSION-EARNED **ROUNDED** =
  - **Also**: watch for the rounded behavior it can sometimes act differently than you expect.
- The following paragraph naming conventions and order must be followed:
  - The Mainline paragraph starts with A000-.
  - Initialization routine starts with B.
  - Main processing routine starts with C.
  - Termination processing start with T or E
  - Read routines start with R.
  - Write routines start with W.
  - Utility routines start with U.

Homework (no lab submission-but needed to prepare for midterm)

Work at making your case study COBOL code compliant with the rules above. You will need to finish making these changes in your code before the midterm Case coding class, since I will be giving you some relatively minor additional requirements to code in midterm practical class.

Remember you will be creating the case in 3 files with the names:

1. PCS1PRG1 – cobol source
2. CCS1SLP – copy book source – the jcl in the abend section shows the entry you need for your compile to locate the copy book (see the line that starts with SYSLIB)
3. JCS1CLG – jcl source

There will be some additions to your code for the midterm practical coding part, and you will be asked to compile, execute and view the output. See the file **Case 1 requirements** for detailed instructions.