

INFO3105 Week 10 Part 1

Review

- Start AIX
- KSDS Sequential file processing
- Using Evaluate for control break processing - different 1st record approach? (keep it consistent with your organizations standards!)

Calling Sub programs

This concept is covered on **pages 320-328** of the text. All mainframe shops use plenty of sub programs to do common routines. This is a simple example that shows the mechanics of how it works and then we'll look at an additional requirement for the 2nd case study. Here is an example of a Calling/Subprogram for you to practice, then we will add a subprogram to your Case2... Let's start with the source for a COBOL program called **CALLER**:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. CALLER.
*****
*   CALLER CODE FROM MURACH TEXT P. 321   *
*   PROGRAM CALLS SUB-PROGRAM CALCFV      *
*****
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 WS-USER-ENTRIES.
   05 WS-INVESTMENT-AMOUNT      PIC 9(5) .
   05 WS-NUMBER-OF-YEARS        PIC 99 .
   05 WS-YEARLY-INTEREST-RATE   PIC 99V9 .
01 WS-WORK-FIELDS.
   05 WS-FUTURE-VALUE          PIC 9(7)V99 .
   05 WS-EDITED-FUTURE-VALUE   PIC $$,$$$,$$$.$99 .
   05 WS-END-RTN               PIC X .
PROCEDURE DIVISION.
000-CALCULATE-FUTURE-VALUE.
    DISPLAY 'INVESTMENT VALUE:'.
    ACCEPT WS-INVESTMENT-AMOUNT.
    DISPLAY 'NUMBER OF YEARS'.
    ACCEPT WS-NUMBER-OF-YEARS.
    DISPLAY 'YEARLY INTEREST RATE'.
    ACCEPT WS-YEARLY-INTEREST-RATE.

    CALL 'CALCFV' USING WS-INVESTMENT-AMOUNT
                       WS-NUMBER-OF-YEARS
                       WS-YEARLY-INTEREST-RATE
                       WS-FUTURE-VALUE.

    MOVE WS-FUTURE-VALUE TO WS-EDITED-FUTURE-VALUE.
    DISPLAY 'FUTURE VALUE IS :', WS-EDITED-FUTURE-VALUE.
    ACCEPT WS-END-RTN.
STOP RUN.
```

We see that the syntax here calls the sub program **by reference** (programs share variables – which is the default in Cobol). It passes four variables to the sub program. The sub program is called

CALCFV. If we wanted to pass the variables by VALUE (programs have copies of the data) we would change the **USING** to **USING BY CONTENT**.

The sub program (CALCFV) looks like this:

```
IDENTIFICATION DIVISION.
*****
*   CODE FROM MURACH TEXT P. 323   *
*****

PROGRAM-ID. CALCFV.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  WORK-FIELDS.
    05  YEAR-COUNTER                PIC 999.

LINKAGE SECTION.
77  LS-INVESTMENT-AMOUNT            PIC 9(5) .
77  LS-NUMBER-OF-YEARS              PIC 99 .
77  LS-YEARLY-INTEREST-RATE         PIC 99V9 .
77  LS-FUTURE-VALUE                PIC 9(7)V99 .

PROCEDURE DIVISION USING LS-INVESTMENT-AMOUNT
                        LS-NUMBER-OF-YEARS
                        LS-YEARLY-INTEREST-RATE
                        LS-FUTURE-VALUE.

000-CALCULATE-FUTURE-VALUE.
    MOVE LS-INVESTMENT-AMOUNT TO LS-FUTURE-VALUE.
    MOVE 1 TO YEAR-COUNTER.
    PERFORM UNTIL YEAR-COUNTER > LS-NUMBER-OF-YEARS
        COMPUTE LS-FUTURE-VALUE ROUNDED =
            LS-FUTURE-VALUE +
            (LS-FUTURE-VALUE * LS-YEARLY-INTEREST-RATE / 100)
        ADD 1 TO YEAR-COUNTER
    END-PERFORM.

EXIT PROGRAM.
```

We see a couple of new items here, the first is the **LINKAGE SECTION**. The linkage section's job is to share the memory in the case of passing by reference or make a copy of the variables if using the USING BY CONTENT syntax. The other thing we see is the **USING** clause in the PROCEDURE DIVISION definition. This is just showing the variables that are housed in the LINKAGE SECTION. If we ran the program locally we'd see the following:

```
C:\ caller.exe
INVESTMENT VALUE:
20000
NUMBER OF YEARS
20
YEARLY INTEREST RATE
025
FUTURE VALUE IS :    $32,772.33
```

As we see the 20,000 over 20 years at 2.5% works out to the value **32,772.33**. The caller passes the variables over, the sub program does the calculation and then the caller displays the edited value.

Additional Notes for sub programs on the Host

To get our code functioning on to the Marist Host we would need to:

1. Create a **PCALCFV** COBOL member
 - a. Ensure the size of the Investment amount is 6 bytes
 - b. Change the rate from 99V9 **to 9V99** which is more in tune with today's rates
2. Create a **PCALLER** COBOL member
 - a. **REMOVE ALL ACCEPT** verbs from the program - we don't have access to the MARIST console
 - b. Add a **sequential input file** (called FVTERMS in the JCL below) to house the following input parameters The file should contain 11 bytes:
 - i. 1-6 Amount
 - ii. 7-8 # of years
 - iii. 9-11 interest rate

```
12500020350
05000015325
15000010225
07500025310
09999910299
```

- c.
 - d. Add additional code to process this file until EOF, this logic replaces what the original ACCEPTS were doing
 - e. Again, change the rate from 99V9 **to 9V99** which is more in tune with today's rates
 - f. Modify the DISPLAYs to be a bit more descriptive (see below)
3. Create a jcl member to **compile and link** PCALCFV - **JCLCALC**

4. Create another jcl member **compile, link and run** PCALLER – Replace my JCLRCLR to **J###CLR** where ### is the 3 characters in your KC03### account

```
//JCLRCLR JOB JCLRCLR,NOTIFY=&SYSUID
//*****
//* JCL TO COMPILE COBOL SOURCE CODE AND LINK MODULE
//*****
//PROCLIB JCLLIB ORDER=ZOS.PUBLIC.JCL
//COMPILE EXEC IGYWCL,PARM.COBOL='LIB,TEST,XREF'
//COBOL.SYSIN DD DSN=&SYSUID..COBOL(PCALLER),DISP=SHR
//LKED.SYSLMOD DD DSN=&SYSUID..LOAD(PCALLER),DISP=SHR
//LKED.SYSLIB DD DSN=&SYSUID..LOAD,DISP=SHR
//          DD DSN=CEE.SCEELKED,DISP=SHR
//RUNIT EXEC PGM=PCALLER
//STEPLIB DD DSN=&SYSUID..LOAD,DISP=SHR
//INPUT DD DSN=&SYSUID..FVTERMS,DISP=SHR
/*
```

The SYSLIB line is needed for the PCALLER compile to locate the PCALCFV object code. FVTERMS is the sequential file described above.

The screenshot shows a JCL job list on the left with a red circle around 'RUNIT:SYSOUT'. An arrow points from this circle to the output window on the right. The output window shows three files: 'PCALLER.cbl', 'KC03J4BJCLRCLR.JOB03696.D0000103.?.spool', and 'PCALCFV.cbl'. The output of the job is displayed in a table format with 7 columns.

	1	2	3	4	5	6	7
Future value of :	\$125,000.00	after 20 years	at 3.50%	is	\$248,723.63		
Future value of :	\$50,000.00	after 15 years	at 3.25%	is	\$80,783.14		
Future value of :	\$150,000.00	after 10 years	at 2.25%	is	\$187,380.53		
Future value of :	\$75,000.00	after 25 years	at 3.10%	is	\$160,889.56		
Future value of :	\$99,999.00	after 10 years	at 2.99%	is	\$134,259.86		

As we see the 125,000 over 20 years at 3.50% works out to the value **\$248,723.63**. The caller passes the variables over, the sub program does the calculation and then the caller displays out the edited value.

Calling a Sub Program in the Case Study – PCS2PRG3

Now that we have covered the mechanics of calling sub-programs we'll add one into the case study. Specifically we're going to remove the star rating logic and working storage variables from PCS2PRG2 and put some new ones in the new sub program.

To facilitate this we'll need to pass the department number and sales amount over to a new program called **PCS2PRG3** and receive the star string back in PCS2PRG2. So your call from PCS2PRG2 would look something like this:

```
* CALL TO SUB PROGRAM TO GET STAR RATING
CALL 'PCS2PRG3'
    USING  WS-DEPT-NO WS-NET-SALES WS-SL-SLSP-RATING.
```

After returning from the call, the variable WS-SL-SLSP-RATING should be loaded with the number of asterisks for the current salesperson. The PCS2PRG3 Linkage Section and Procedure Division layouts would be similar to:

```
LINKAGE SECTION.
77  LS-DEPT          PIC X(2) .
77  LS-SALES         PIC 9(7)V99  COMP-3 .
77  LS-RATING        PIC X(5) .

PROCEDURE DIVISION USING LS-DEPT LS-SALES LS-RATING.
```

We're also going to make this sub program (PCS2PRG3) use a **2 dimensional table**. I will put this into a text file on FOL to save you from typing 50 lines of code. You are to put this into a copy book and then use the copy book in PCS2PRG3. Here is how the table is laid out, note two dimensional tables are discussed on pages **286-288** of the text. We can see from the code below that 2 dimensions are coded by nesting an occurs within an occurs (first level 05 WS-DEPT-GROUP has 5 of the level 10 WS-DEPT-NO and WS-RATING-GROUP fields... and WS-RATING-GROUP has 5 ...):

```
01 WS-RATING-TABLE REDEFINES WS-DEPT-RATES.
05 WS-DEPT-GROUP OCCURS 5 TIMES.
    10 WS-DEPT-NO          PIC XX.
    10 WS-RATING-GROUP OCCURS 5 TIMES.
        15 WS-SALES-MIN-VAL PIC 9(5)V99  COMP-3 .
        15 WS-SALES-MAX-VAL PIC 9(5)V99  COMP-3 .
        15 WS-STAR-RATING  PIC X(5) .
```

To determine the correct star value, you have to **first locate the correct department** in the table (the first dimension) and **then locate the correct Net Sales** (the second dimension) to find the WS-STAR-RATING. See your code from Case1 rating table processing & modify as required.

When referencing the first dimension you use one subscript, such as:

WS-DEPT-NO (WS-DEPT-SUB)

When referencing variables in the second dimension you need to use 2 subscripts:

WS-SALES-MIN-VAL (WS-DEPT-SUB, WS-RATG-SUB)

You will need a standard compile and link jcl stream for the new PCS2PRG3 and then you will need **to change** the compile and link jcl for **PCS2PRG2** to include two linkage lines. The first one finds the sub program in your load pds, the second one is needed to locate the system module that provides the date to our program (we overrode the default SYSLIB). Don't forget to change the 1st line to reflect your # something like **JCS2###3** would be good:

```
!//JCS2CLP2 JOB JCS2CLP2,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.COBOLE='LIB,TEST,XREF'
//SYSLIB DD DSN=&SYSUID..COPY(CCS2SLSP),DISP=SHR
//COBOL.SYSIN DD DSN=&SYSUID..COBOL(PCS2PRG2),DISP=SHR
//LKED.SYSLMOD DD DSN=&SYSUID..LOAD(PCS2PRG2),DISP=SHR
//LKED.SYSLIB DD DSN=&SYSUID..LOAD,DISP=SHR
//          DD DSN=CEE.SCEELKED,DISP=SHR
/*
```

That's the last requirement for the 2nd case study!

Summary of Case 2 Requirements

- JCL
 - Compile and Link for PCS2PRG1 (Z)
 - Compile and Link for PCS2PRG2 (Z)
 - Compile and Link for PCS2PRG3 (Z)
 - JCS2LDRN (5 step job) (Z)
 - 1. Step 1 – Delete and Define KSDS

2. Step 2 – Load KSDS using DFSort
 3. Step 3 – Build AIX
 4. Step 4 – Execute PCS2PRG1
 5. Step 5 – Execute PCS2PRG2
 - Calls PCS2PRG3
- COBOL
 - PCS2PRG1 (transaction program)
 - PCS2PRG2 (double control break)
 - PCS2PRG3 (sub program from today using 2d table)
 - Data
 - Salesperson KSDS
 - Salesperson sequential file
 - Sales Transaction file
 - Copybooks
 - CCS2SLSP – Salesperson Master layout for KSDS
 - CCS2SLST – Transaction file layout
 - CCS2SLSW – Transaction fields for Working Storage
 - CCS2STAR – 2 dimensional star ratings for departments

Notes

- **Confirm your numbers are rounded**
- **Submit the entire output of the compile and link step for PCSPRG1, PCSPRG2, and PCS2PRG3 programs**
- **Submit the entire execution output from Marist for JCS2 execution**