IBC233 Lab 3 (updated Fall 2014)

**Writing an interactive RPGLE screen program**

**Lab objectives:**

* **Create a display file (screen)**
* **Code an interactive RPGLE program using the display file**

**Lab Requirements:**

**Hand in the compile listing for MARKSRPG**

**Successfully run MARKSRPG**

Start an RDp session

Start a ‘Green Screen’ (emulator) Session.

**Using** **Rational Developer for Power Systems (RDp):**

**Part A**

**Objectives:**

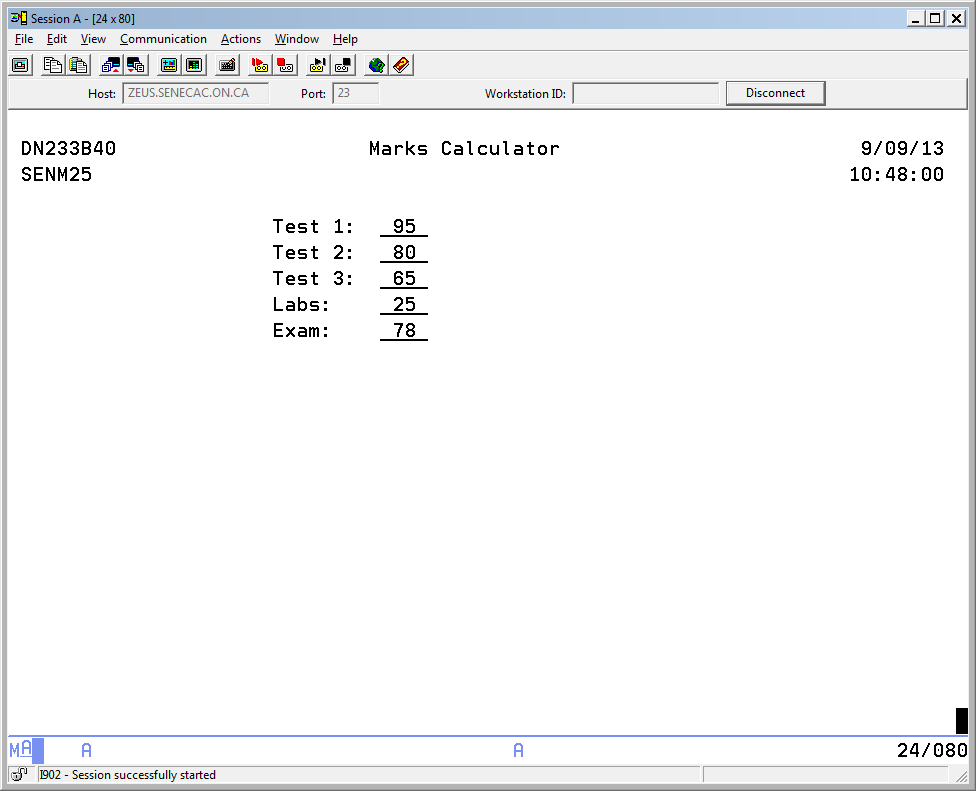
* **Use RDp Screen Designer to Create a Display File**

Display File screens in IBM i are very similar to forms you see on Web pages. Entering and compiling DDS code can produce these interactive screens. One option that may be used for entering DDS code is to use a GUI tool that allows you to visually select and enter what appears on the screen. Your selections are converted to DDS program code that is compiled to produce interactive screen records in a display file object. Those interactive screens are available to various program languages.

We will create a display file object called MARKSDSP that uses two overlapping screen records. RPGLE, CLLE and COBOL programs can interact with the screens in the MARKSDSP object. We will code an RPGLE program called MARKSRPG to view and interact with the screen records. The following shows a run of the RPGLE program.

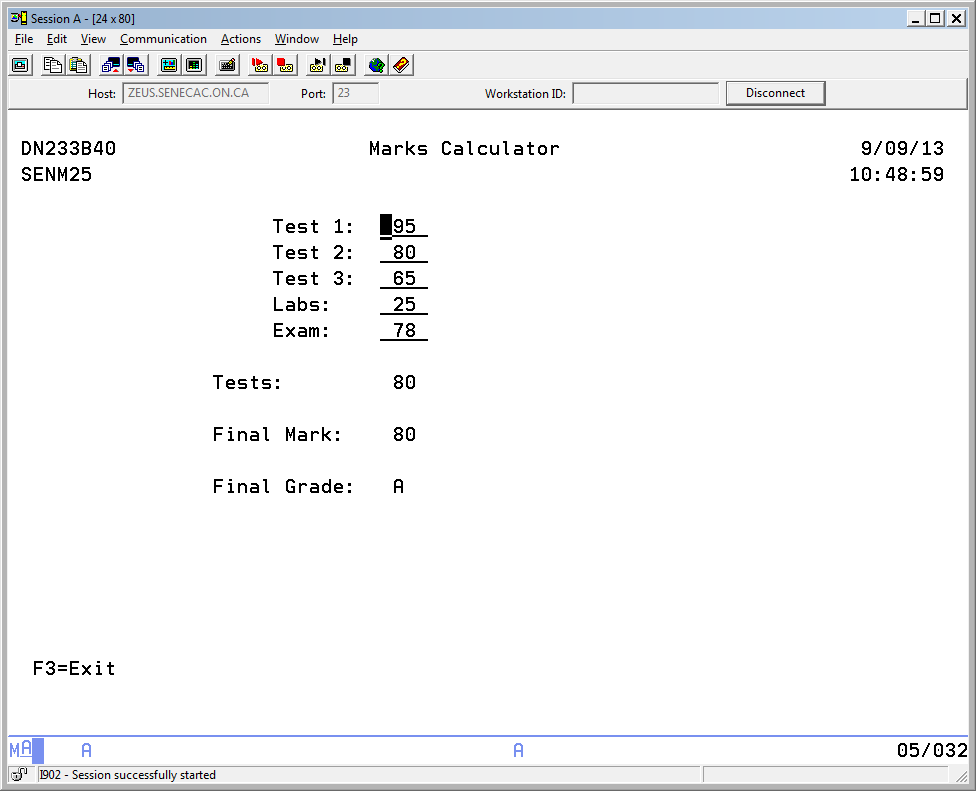
==>CALL MARKSRPG

Marks are entered on the first screen called RECORD1



ENTER is pressed

# A second screen called RECORD2 overlays the first screen



The user can either press F3 to exit the program or enter to check out other mark combinations. F3 is only available for use from RECORD2.

## Using Screen Designer to Produce a Display File

Source code for a Display File may be stored in a source file called **QDDSSRC**. You have already created this file in Lab 2 to store DDS code in the STUDENTS member that produced a physical file called STUDENTS.

DDS Stands for Data Description Specifications. The native IBM i language used to describe \*FILE objects.

Create a new member inside of QDDSSRC. The member name should be specified as MARKSDSP and the file type should be DSPF. After creating the member, close the MARKSDSP tab.   
  
Right click on your MARKSDSP member, click on Open With and select SCREEN DESIGNER.

You should still see a Remote Systems view on your upper right. The following should appear:

An outline view on the right.

A tab in the center that has your member name (MARKSDSP) with two boxes below the tab. One box has a title of Screens and the other box includes a Screen and Records tab

Below this a work area should appear with a ruler that counts from column 1 to column 100

A palette is available to the right of this work area and this will be used to select and place fields and constants on our screen record.

Immediately below this work are will be tabs that allow us to work with the GUI or to work with actual DDS code

Further down is a Source Prompter tab. We have already used a source prompter in lab 2 to enter RPGLE code and to enter DDS code for a physical file program.

A properties box on the lower left which can be used to enter useful information about records and fields.

|  |  |
| --- | --- |
|  | .  Left click on Standard Record in the Palette area and drag this over to the left work area.  Record1 should show up in several spots. In a right top window and in a Properties box on the lower left. This name can be changed to something else here. We will keep the default name. |

Create a date constant ( MM/DD/YY ) by clicking on Date Constant which is found in the palette underneath the Constants folder. Move your cursor over to the top left of your black grid screen and click once to place the date constant. As you move your cursor with the mouse, you should see the row and column numbers changing.

After you have placed this date, go back to the constants folder in the palette and use the arrows to scroll through other constant selections. We are using most of them. Refer to the finished first screen on a previous page to determine the placement of these constants and then add them to your work area.

Marks Calculator and Test 1, Test 2, … are text constants.

Click on Text Constant in the palette and then click to place this centred at the top of your screen.

You can change the default text of “Text constant” to “Marks Calculator” one of two ways. Type directly in the box provided. If this is awkward click on the word “Text Constant” in the properties box in the lower left corner.

Put in all the Text constants using the finished product screenshot.

We need to put in fields beside the test, lab and exam text constants.

Click on Named Field below the Fields folder in the palette. Place the named field beside the Test 1: constant.

This shows up as BBBBBBBBB on your work screen. Double click on the Properties box to specify information about this field. The field name that was provided for you is FIELD1. This can be easily changed in the box beside name.

Change the field name to TEST1.

The default properties for this field are set as 9 character Input/Output field. The usage reflects this in Properties with a setting of “Both”.

Make this a 3 character field by typing over the “9” that appears under Length on the right side of the properties view.

The usage for the field should remain as Both. This means it is input capable (the user can enter something into the field when it is presented on the screen record and it is output capable, the program or user can set the field contents and it will show up to the user the next time the screen is displayed.

What appears on your work screen with the following:

Usage set to INPUT \_\_\_\_\_\_\_\_\_\_\_

Usage set to OUTPUT \_\_\_\_\_\_\_\_\_\_\_

We want to make it a 3 digit numeric field with the field name of TEST1. In the Properties view change the type of field from Character to Zoned.

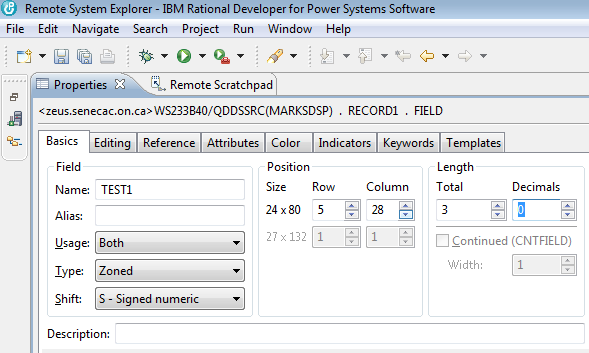
What appears on your work screen with the following

Usage set to INPUT \_\_\_\_\_\_\_\_\_\_\_

Usage set to OUTPUT \_\_\_\_\_\_\_\_\_\_\_

Usage set to BOTH \_\_\_\_\_\_\_\_\_\_\_

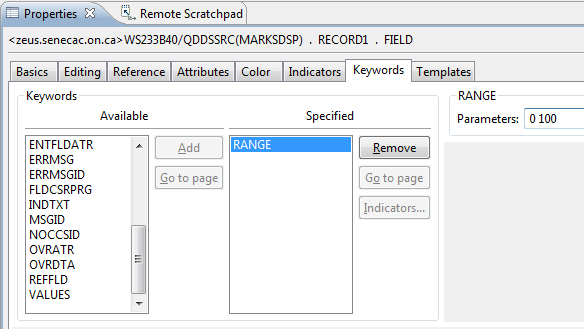
Shift appears below Type and should be set as Signed Numeric.



Click on the Keywords tab and look for the RANGE keyword from available keywords on the left hand side. When you click on RANGE, the Add button should become available. Click on this to add RANGE to the specified column box. The Range for a Test mark would be any marks from 0 to 100, Enter this information in the parameters box that should be available while you are working with the specified “RANGE” keyword. (0 100)

Remember that a Test may have a range of 0 to 100, but our Labs this semester are worth 30% of the final grade. They will have a range of 0 to 30.

A screenshot is provided for this.

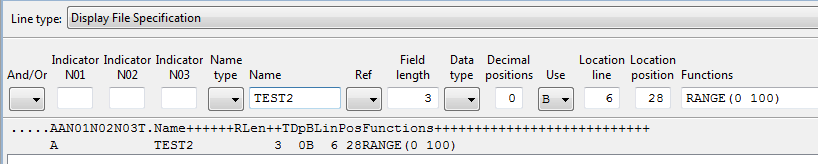


Double click on the Properties tab to get back to the multi view screen

Click on Named Field and place a second field beside a TEST2 text constant. There are three tabs showing midway down your screen. **Design Source** and **Preview.** Click on the Source tab and press F4.

Double click on the Source Prompter tab that appears at the bottom view on your screen. On my screen the tabs show up as Remote System Details, Tasks, Object Table, Commands Log, Error List and Source Prompter.

You want to enter the following code.

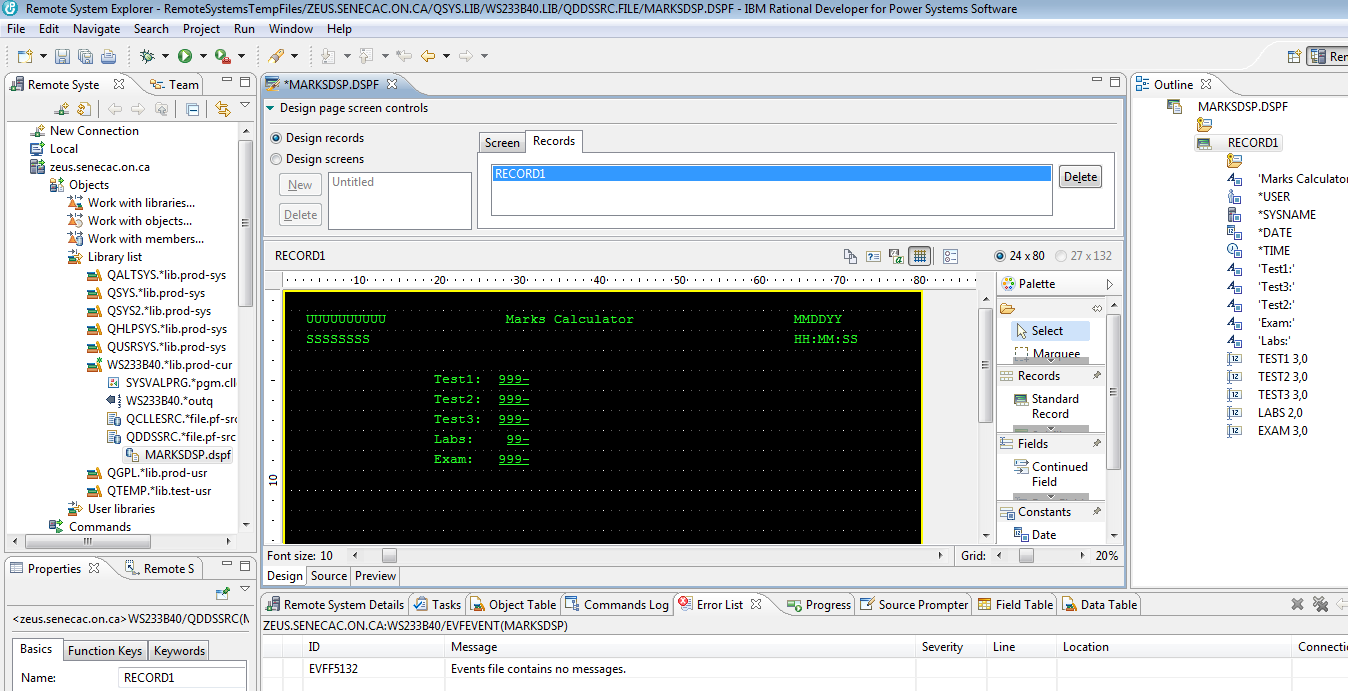


Notice that the RANGE(0 100) part can be added faster by typing in code than clicking on a number of things in the Properties box as we had to do with TEST1.

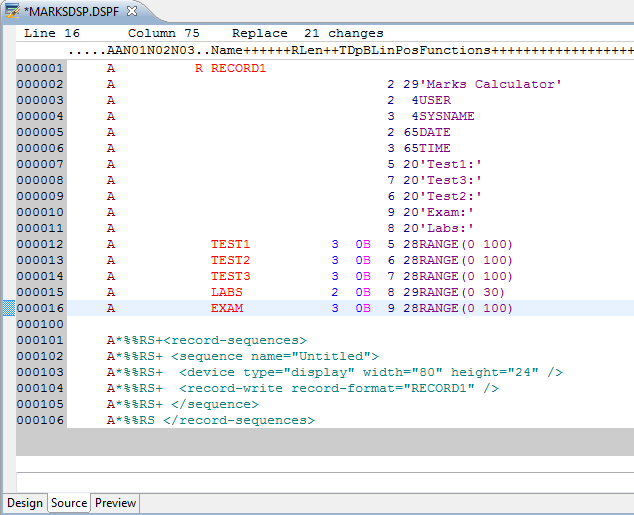
Notice the actual line of DDS code shows below where you made your entries. It starts with the letter A.

Enter the text and appropriate fields for the other two tests, labs and exam for this semester. You can also set your properties in the Properties box or use the Source Prompter to enter your code.

You can use the screenshots of the running program as a guide or the finished screen in Screen Designer is shown below. Remember to click on the Design tab to get back to the GUI interface.



DDS code has been generated for you. You can view and change this code. You have to be careful when changing it that your syntax is correct. In order to see the code, click on the **Source** tab. This is found just below your work screen. There are three tabs. **Design** (the one that provides a GUI for creating an interactive screen) **Source** (that allows you to view and change DDS code) and **Preview** (Discussed Later).



The length of the TEST1 field shows as “3S 0” which means a length of three, a type of “S” depicting zoned decimal and 0 decimal positions for this field. You can see at the top some help for the proper columns to make these sorts of entries. It is easier to use the properties box or the source prompter when entering this information. It is also possible to make direct entries as you get used to the code. Change the length of the TEST3 field from three to four and then click on the Design tab to see if this change is shown.

Go back to the Source tab and change it back to a length of three.

We have to be careful here. You could change the “RANGE” keyword to RBNGE” and introduce an error to your code and possibly confuse the Design Tab’s interpretation of the code.

According to the DDS code showing on this lab, what line and column (or position) number does my TEST1

appear on? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Your code can have this placed on a different line and position.

Go back to the Design tab and place a named field somewhere beside the TEST1 field. Check this out in the code by clicking on the Source tab.

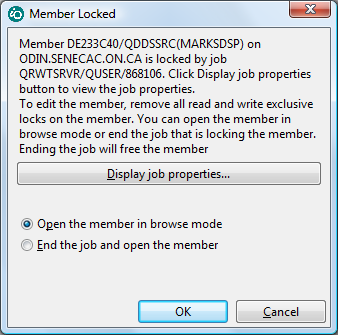
Notice that this new field appears seemingly out of order in the code. It appears at the bottom of the generated code. If you take a look at the line and position number it should show as the same line number you had for your TEST1 field.

If this bothers you, you could type an “M” where the line numbers for your code shows (Move). For me that would involve changing 001404 to M01404.

Then put your cursor on the line number for the TEST1 field and type an “A” (After). Try it.

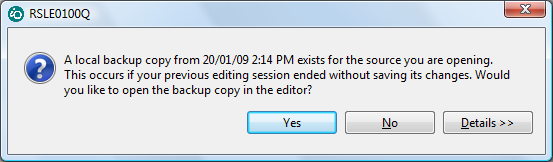
Click on File at the top right hand corner of your screen and select Save. This is a good periodic practice to recover from a crash or other problem.

If you do have a crash, you may restart and get the following message.



You can change the action to End the Job and open the member.

Your selection here depends on your situation.



If you had messed up the current session, you don’t want the local backup. If you have done a lot of work and not messed up the current session then you do want the local backup.

You are seeing this error dialogue because while doing this lab, my internet provider caused the connection to drop causing me to have to sign in again. This is less likely to happen to you at school, but may happen at home.

You are ready to provide the second screen record that will overlay the first screen record.

Click on Standard Record in the palette and drop it onto your work area. You should get a RECORD2 with a blank work area. You can toggle between RECORD2 and RECORD1 by clicking on the appropriate name at the top box in the screen. Try it.

Making sure all your entries are below the area for RECORD1 (in order for OVERLAY to work properly)

Enter:

Constants Field

Tests: a three digit zoned decimal output only field called TESTOVRALL

Final Mark: a three digit zoned decimal output only field called NUMGRADE

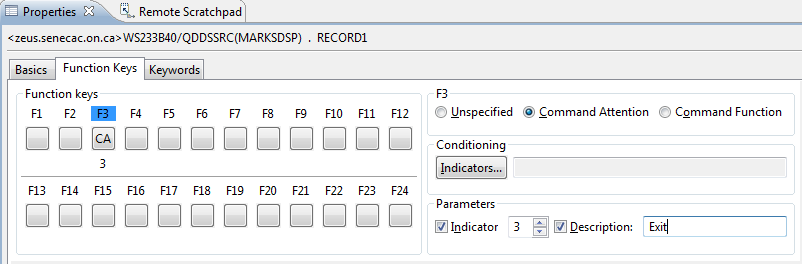
Final Grade: a two character output only field called LETGRADE

F3- Exit

**Specifying Function Keys**

We specified a constant which told the user to use F3 to exit the program. Constants are only text, they do not do anything! So, to make the F3 function key available to the user...

Making sure your focus is not on a constant or a field in the RECORD2 work area, double click on the properties view tab at the lower left of your screen and then select the Function Keys tab.



Click on F3 and on the right hand side select the Command Attention radio button.

Under Parameters click on Indicator and beside that change the number to show as a three.

Click on Description and type in EXIT.

Let’s review what you have done:

By clicking on F3 you have enabled the F3 key for RECORD2. That means when the user is looking at RECORD2 they can press F3. They can not press F4 or F5 because those keys have not been enabled.

The Enter key is by default enabled for the screen records. Now, the user can press F3 or Enter. Our program will exit when F3 is pressed and reshow RECORD1 if Enter is pressed.

Command Attention means that no data is passed back to the program. If you entered data into an input capable field, that data would not be available to the program. It is possible to allow an Exit from a program when the user presses F3 and take data entered into the input screen fields and have the program do something with that data before it terminates. Since Command Attention does not allow the program access to the input field data, you would probably select Command Function for that application.

The program needs to know if F3 or Enter was pressed. You have enabled a response indicator by checking of Indicator 3 under Parameters. There are 99 indicators available to you. They are like switches with an “On” or an “Off” setting. This logical field either has a ‘1’ or a ‘0’ in it.

The EXIT Description is a comment that is placed in your code.

What was the actual DDS code for all of this? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(You already know how to look this up)

Does this function key setting apply to RECORD1 and RECORD2 or just one of those records. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Add the OVERLAY keyword for RECORD2 by using the KEYWORDS tab when setting RECORD2 properties.

This will allow RECORD2 to show without wiping out RECORD1. If any of RECORD2’s fields or constants are on the same line as a RECORD1 field or constant, RECORD1 will still be wiped out.

Close your MARKSDSP tab and save your work. So far, we only have DDS code. We do not have a Display File Object that can be used by programs.

Right click on you closed Display file member (in the Remote Systems View) and compile it.

What is the command used to compile a display file DDS member? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

You should see feedback that the “Events file contains no messages” under the Error List Tab.

Click on the Commands Log Tab. You should see dialogue similar to the following:

SBMJOB CMD(CRTDSPF SRCFILE(DE233C40/QDDSSRC) SRCMBR(MARKSDSP) REPLACE(\*YES) OPTION(\*EVENTF) FILE(DE233C40/MARKSDSP)) JOBD(\*USRPRF)

Job 870076/DE233C40/QDFTJOBD submitted to job queue QBATCH in library QGPL.

CRTDSPF SRCFILE(DE233C40/QDDSSRC) SRCMBR(MARKSDSP) REPLACE(\*YES) OPTION(\*EVENTF) FILE(DE233C40/MARKSDSP)

**File MARKSDSP created in library DE233C40.**

If you don’t get this there may be an error introduced into the code when you were working with the Source tab.

There also may be trouble with your connection. If the **File MARKSDSP created in library** dialogue does not appear and no syntax errors show, then close up RDp, reopen it and then try the compile.

You can also compile this code using the green screen. Sign on to a Client Access session.

Type WRKMBRPDM QDDSSRC at the command line.

(This won’t work properly if you didn’t call your source physical file QDDSSRC)

Use option 14 beside the MARKSDSP member name.

Instead of typing WRKSPLF at the command line to view your listing, type in SP where you had typed in “14”.

Use option 5 and scroll up and down your listing or just type in “B” at the top to get to the bottom of your listing and look for the **File MARKSDSP created in library** message.

We are ready to code our program. Here is some help with entering the code.

Although a lot of help is provided, this help assumes that you made it to class and saw your instructor enter the code. If you missed that class, you should be able to get help from a lab aid or an instructor if you are stuck on any step. If you missed both the class and the lab, you should see an IBC233 tutor.

Add a new member called MARKSRPG with an RPGLE member type to your QRPGLESRC source physical file. It will contain your program code for an RPGLE program that uses the display file records you have just created.

Press F4 to get access to the source prompter and select the F:File Description Line Type.

What file name are we processing? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Enter that in the appropriate prompter box.

Also provide the following:

Means

File Type: C combined or input/output

File Designation: F full procedural (you will provide all the logic for the program)

File Format: E Externally described file Felds like TEST1 will have their definitions

brought in automatically from the Display file code.

You don’t have to define the fields in your program

Device WORKSTN A workstation file where input/output is through a display.

Apply this and double click on the MARKSRPG tab.

We are going to enter the rest of our code as Free Format RPG. To do this, we need to enter /FREE on the next line.

The slash must go in column 7. This can be determined by looking at the top area underneath the MARKSRPG tab. As you move your cursor, feedback should be provided about which column number you are in.

The code you need to enter for the main routine will be supplied for you. This version will not be perfect and may require a few adjustments in a later lab.

It is important to remember that each program statement needs to end with a semicolon. If you don’t do this, a syntax error may pop up when you go to the next line. The compiler will also reject an RPGLE statement that does not end with a ” ; ”

If a syntax error pops up it may not always appear immediately below the line that is missing the semicolon. Figure out where you need to make the adjustment and then press CTRL + F5 to get rid of the message that shows up in the LPEX editor.

Common PDM and LPEX editor commands:

A line can be deleted by typing a D in the first position of the line number and pressing enter.

A line can also be copied by typing a C at the line and a B or A at the line it is to be copied before or after

Multiple lines can be deleted by typing DD at the start of the block and DD at the end of the block

CC and MM also work with a block of lines

There are more modern ways of editing available with the LPEX editor.

Here is the code you need to enter.

/FREE

EXFMT RECORD1;

DOW NOT(\*IN03);

EXSR GETGRADE;

WRITE RECORD1;

EXFMT RECORD2;

IF \*IN03 = \*OFF;

EXSR CLEARMARKS;

EXFMT RECORD1;

ENDIF;

ENDDO;

\*INLR = \*ON;

RETURN;

EXFMT is a Read/Write operation

A write of the screen record to the display station and a pause while the user reads it. The user may fill in some fields if there are input capable fields on the screen. At minimum the user will press the return key. A read back to the program occurs and the next program statement is executed.

DOW / ENDDO is for looping. The test is done at the start of the loop and if the test is not passed, none of the loop statements are executed.

NOT(\*IN03) is a test of the response indicator 03 which was specified when developing MARKSDSP.

Where was that specified? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

You could have also said DOW (\*IN03 = ‘0’) or DOW (\*IN03 = \*OFF)

EXSR GETGRADE is sending control down to a subroutine called GETGRADE. This subroutine needs to appear after the RETURN statement and will determine the contents of RECORD2 output only fields.

Here is some sample code to use for this subroutine:

BEGSR GETGRADE;

LETGRADE = ‘F’;

TESTOVRALL = (TEST1 + TEST2 + TEST3) / 3;

NUMGRADE = 54;

ENDSR;

Notice that NUMGRADE and LETGRADE have not been determined properly in the code above. The lab mark has been entered as a mark out of 30 while the test and exam marks are entered out of 100. For these students the final mark is calculated by allowing 30% for labs, 35% for the tests and 35% for the final exam. The Numeric grade is incorrectly assigned a value of 54. That allows you to run the program and see if the screens are working properly. Once you have done this, the “NUMGRADE = ??” statement can be corrected to show the proper value.

Then figure out how to set the Letter Grade (LETGRADE).

A+ = 90 - 100 A = 80 – 89 B+ = 75 - 79 B = 70 – 74

C+ = 65 - 69 C = 60 – 64 D+ = 55 - 59 D = 50 - 54

F = 0 – 49

Case structure is a good choice here. Here is an example of that:

SELECT;

When Day = 1;

DayName = ‘Monday’;

When Day = 2;

DayName = ‘Tuesday’;

……

ENDSL;

If Day is a 1, then the first test would be done, the DayName would be set to ‘Monday’ and control would go to ENDSL. The test for Day = 2 would not be performed. If Day is a 7 then seven tests would be done and a DayName = ‘Sunday’ statement would be executed.

If an 8 or 9 was entered in Day you could include an Other clause in place of a When clause at the bottom of this structure before the ENDSL. Other is a catchall.

Other

DayName = ‘Invalid Day Number’

ENDSL

You need to figure out how to convert a number to a grade. This will be done after you get your program running with the incorrect way of determining the letter and numeric grades.

Do not test both the upper and lower limits of a range – you only need to test one or the other depending on your approach. Testing both limits is wasting CPU time. Not a big problem with this application, but may be a factor when processing millions of records.

Make sure you test this thoroughly when actually running your program.

WRITE RECORD1 In order to get the overlay RECORD1 is rewritten without a pause for the user (the READ back)

EXFMT RECORD2 – Overlays RECORD2 over RECORD1 by a write of RECORD2 and then a pause for a read back of RECORD2.

At this point the F3 key is available. The user can either press Enter and not get out of the loop or press F3 to eventually get out of the loop.

The ClearMarks subroutine is making sure that you reset the marks back to 0.

\*INLR = \*ON is the standard way of ending an RPGLE program. It translates as the Last Record indicator is set on. It really means that the program is ending and a copy of the program will not remain in memory when it is over. If this program had produced a spooled file, the spooled file would be closed and accessible. Forgetting this statement makes that spooled file difficult to display.

RETURN – control is returned to the operating system.

Use the sample code for the GETGRADE subroutine so you can see your program running. After the program is successfully running you can refine this subroutine. For now, your CLEARMARKS subroutine could say something like:

TEST1 = 0;

Compile MARKSRPG.

Fix any errors

Switch to your Client Access session and run MARKSRPG. This is done by typing

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ at the command line.

Enter marks for all the input fields and press enter.

When your results screen shows, can you still type numbers into the top screen fields? \_\_\_\_\_\_\_\_\_\_\_

We will have to fix this in a later lab. If we made those fields Input only they would disappear when REDCORD2 is being shown.

Test your input fields for valid marks. For example a TEST 1 mark cannot be 150, or “A” or –12.

Make sure you can exit your program by pressing the F3 key. A previous lab showed what do to if you can’t stop your program.

What do you need to do to stop this program if it won’t end?

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

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

Fix your program to produce the correct letter and numeric grades.

An instructor’s sample of this program is available to check yours against. It includes a feature where pressing the F4 key identifies it as a sample program and you will not duplicate this feature.

To run it change your current library to IBC233LIB and then invoke the program:

CHGCURLIB IBC233LIB

CALL MARKSRPG

After you have finished, you should reset your current library back to your own library.

What happens if you do not change the current library when running the instructor’s version of this program?

It will depend on the existence of MARKSDSP in your library.

Try running this program with your library as the current library after you have successfully created your version of MARKSDSP.

What happened?

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

**Part B**

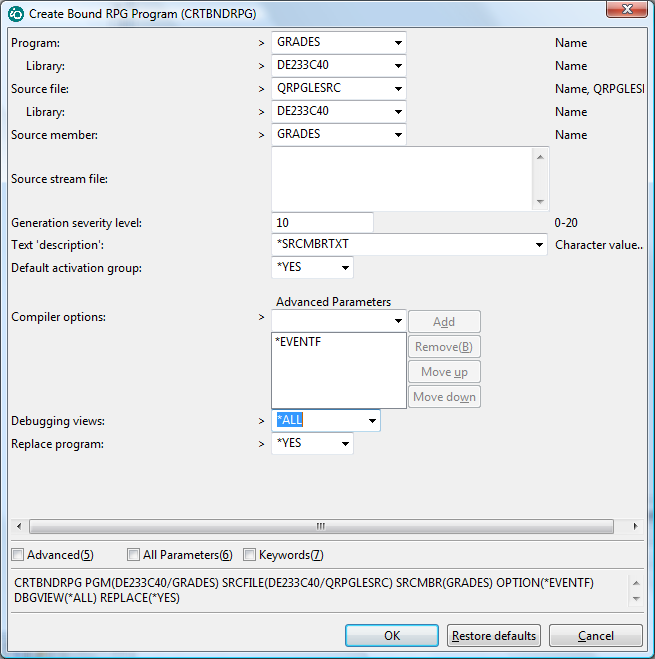
**Objectives:**

* **Use the Interactive Debugger (STRDBG)**
  + **Demonstrate stepping through a program**
  + **Demonstrate viewing the contents of a variable**

The **STaRt DeBuG** and it’s corresponding command **ENDDeBuG** are your best friends in this course. Learning to use this tool will save you hours of aggravation looking for logic errors in your assignments.

**Compile your program for use with debug**

Right click on your program source member and select Compile(Prompt)

****

Change the Debugging view to say \*ALL

What is the actual CL compile command being used. It shows at the bottom of your screen.

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

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

You can use this when compiling on the green screen. Option 14 can be prompted when you use it beside a program member name. You can page down to search for the debugging views.

You can also enter the command you wrote down at the command line without the “OPTION(\*EVENTF) keyword and value.

Pressing F9 can retrieve the command and the member name or the source file name can be changed depending on what RPGLE program you are compiling.

Remember – if your last compile does not allow debugging views, then you will not be able to use the debugger.

**Switch to a ‘Green Screen’ Session**

At the command line enter the STRDBG command and your program name. For example:

==>STRDBG MARKSRPG

You should see a screen that shows your program code. You can add a breakpoint to your program. We will add a breakpoint at the line where the test marks are added up. This is done by putting your cursor on a line and pressing F6. You can also press F6 a second time when the cursor is on the line to remove the breakpoint.

We will set a breakpoint here.

|  |  |
| --- | --- |
|  | Press F10 to exit from this screen so you can start your program.  Start your program  ==>Call MARKSRPG |

You are now back to the screen you saw before but the program is running in debug mode. You will see the first screen because we haven’t hit the breakpoint yet. Enter marks for this screen and then press enter.

We are now at the breakpoint. Put your cursor on the TESTOVRALL field and press F11 to display the variable.

If the variable is not showing on the screen, but you are curious to see it’s contents you can do this with the EVAL command at the command line.

Type EVAL LETGRADE at the command line. What shows? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

You can change a fields content while the program is running. Try the following at the command line:

EVAL LETGRADE = 'C'

Then just press F9 a few time to retrieve EVAL LETGRADE and run it.

Press F10 to step through one line of the program. What shows for TESTOVRALL now? \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Keep pressing F10 to get to the screen that shows the marks. This may overlay the debugger screen and be a little confusing. You can press enter to get out of this. Without overlay this would not be a problem.

Press F12 to resume the program and then get back to the spot where you have a breakpoint.

Press F13 to clear your breakpoint.

Resume your program with F12, get passed the problem overlay screen if it appears over you debug code by pressing enter.

Keep running the program until you see the RECORD1 and RECORD2 on your screen.

Press F3 to exit.

You can also exit from the breakpoint in the debugger by pressing F3 and your debug session will end.

When you are finished with your program, you need to end the debug session. To do this type in the following at the command line:

===>ENDDBG

Other useful debugger commands to check out

WATCH variable

ATT variable

SET

STEP 3

H at the debugger command line will give you access to debugger help

Lab 3 Summary

A display file is coded with DDS code in a member that has a DSPF type.

Once this DDS code has been successfully compiled a Display File object is produced that can be used by different programming languages that want to take advantage of the interactive screens allowing data entry and data display.

The easiest way to define how the display file screens or records will look is to use the Screen Designer GUI in RDi.

The GUI allows you to define field names, types, usage, editing, validations and optional indicators that allow viewing of the field.

You would use the GUI to provide this information and the appropriate line is entered as DDS code.

Here is a sample line with explanations

This is how it actually appears

Ind Field Name LT DU P Functions

A 90 TEST1 3S 0B 5 32RANGE(0 100)

Some separation is shown here to better highlight concepts – the line would not be entered this way.

Ind Field Name L T D U P Functions

A 90 TEST1 3 S 0 B 5 32 RANGE(0 100)

Indicator 90 has to be on in order to view this field called TEST1 on the screen. If indicator 90 is off, the TEST1 field will not be visible when the screen record is displayed. For our application it would not be a good idea to use an indicator with the TEST1 field.

L shows a field length of three. So at this point XXX could be entered

T or Type shows a field type of zoned decimal and D or Decimal shows 0 decimal positions. So now we cannot put X’s in the field. 999 would work.

U or Usage indicates if this field is for input only (I) , output only (O) or both input and output (B).

P or position on the screen shows that this field would reside on line 5 at column 32.

The RANGE function allows us to validate the field before it is passed to a program. The user can enter 0 to 100. They cannot enter a negative number or 101.

Here is some more partial DDS code with explanations

A R RECORD2

A CA03(03 'EXIT')

A OVERLAY

A 11 18'Tests:'

A 13 18'Final Mark:'

A 15 18'Final Grade:'

A 22 3'F3=Exit'

A TESTOVRALL 3Y 0O 11 32EDTCDE(1)

A NUMGRADE 3Y 0O 13 32EDTCDE(1)

A LETGRADE 2A O 15 33

This screen record is called RECORD2 and has some text and fields that are displayed.

At the record level we see CA03 – that enables the pressing of function key F3 when the screen record is displayed

(03 ‘Exit’) indicates when F3 is pressed indicator 03 will be turned on and the comment is ‘Exit’

OVERLAY is used to allow RECORD2 to overlay a screen record that is already being displayed. This only works of the first screen record does not use lines 11 – 25. You can see RECORD2 is using lines 11 to 22.

EDTCODE(1) can be used to make a numeric field show with commas, a decimal point and suppressed leading zeros

So 090 would show as 90. If the field was larger 0293334^33 would show as 293,334.33. The decimal point is not stored in the field but can be shown with proper editing. We used the “^” symbol to show where the placeholder for a decimal point is.

RPGLE

A Display file is declared in an RPGLE program with a file specification form

FMARKSDSP CF E WORKSTN

FileName Type Designation Format Device

FMARKSDSP C F E WORKSTN

Type can be (C)ombined for interactive screens, other options are (I)nput, (O)utput and (U)pdate

Designation can be (F)ull Procedural – that is the only one we will use this semester

Format can be (E)xternally Described – indicating that the program can bring in field names, types and sizes from the actual file. The programmer does not have to define these fields described in the file.

EXFMT – write a screen record from a display file, pause and when the user presses enter read back what has been inputted into the screen record fields. Sometimes there are no fields and the read back just acknowledges that the screen has been viewed and the user is letting the rest of the program proceed.

DOW - a loop where the test on entering the loop or repeating the loop is done at the start of the loop.

ENDDO

DOU – the loop code is executed at least once and the test on repeating the loop is done at the bottom.

WRITE – can be used to have a screen record display. There is no pause and the program continues. This is useful when showing a screen record and then overlaying a second record and then pausing to let the user look at both screen records.

EXSR GETGRADE – control goes from here to a subroutine at the bottom of the program. The named subroutine has a BEGSR and ENDSR to indicate what code is executed and then control goes back up and executes the next line after the EXSR line.

\*IN01 - \*IN99 – in RPGLE indicators are referred to by an asterisk and a number. There are 99 indicators available for use and their default setting is \*OFF or ‘0’ . They can be turned on during the program run and then checked to see if they are on or off to determine a course of action.

\*INLR = \*ON – these two statements are how you should end all your RPGLE programs.

RETURN The last record indicator is turned \*ON and RETURN is used to return control to the operating system.