RPG IV Programming Advanced Workshop for IBM i

(Course code AS10)

Student Exercises

ERC 6.0

Trademarks

IBM® is a registered trademark of International Business Machines Corporation.

The following are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

AS/400®	Current®	DB™
DB2®	Integrated Language Environment®	iSeries®
i5/OS™	i5/OS®	Language Environment®
MQSeries®	OS/400®	Power Systems™
Power Systems Software™	Power®	Rational®
Redbooks®	RPG/400®	System i®
WebSphere®	400®	

Adobe is either a registered trademark or a trademark of Adobe Systems Incorporated in the United States, and/or other countries.

Pentium is a trademark or registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

Windows is a trademark of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Other product and service names might be trademarks of IBM or other companies.

September 2011 edition

Contents

Trademarks

The reader should recognize that the following terms, which appear in the content of this training document, are official trademarks of IBM or other companies:

IBM® is a registered trademark of International Business Machines Corporation.

The following are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

AS/400®	Current®	DB™
DB2®	Integrated Language Environment®	iSeries®
i5/OS™	i5/OS®	Language Environment®
MQSeries®	OS/400®	Power Systems™
Power Systems Software™	Power®	Rational®
Redbooks®	RPG/400®	System i®
WebSphere®	400®	

Adobe is either a registered trademark or a trademark of Adobe Systems Incorporated in the United States, and/or other countries.

Pentium is a trademark or registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

Windows is a trademark of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Other product and service names might be trademarks of IBM or other companies.

Exercises description

Exercise instructions: This section contains what it is you are to accomplish. There are no definitive details on how to perform the tasks. You are given the opportunity to work through the exercise given what you learned in the unit presentation.

Your instructor assigns you a team number, *nnn*. This team number is used for your userid, AS10*nnn*, and your library, AS10*nnn*. Your library contains all the objects you need to perform the exercises. At times, you are asked to copy objects from AS10*XXX*, which is the student master library.

When you are prompted to sign on to the i server, your userid is AS10*nnn* and your password is AS10. The password is set to expire, so enter a new one that you can remember easily. Please read the exercise instructions carefully as you proceed through the exercise.

Which editor to use: You can use the PC-based Remote Systems LPEX editor of Rational Developer for Power Systems, or you can use SEU as your editor. The Remote Systems LPEX editor is the recommended editor. It is straightforward and easy to use.

You can open the Remote Systems LPEX editor by **Start>All Programs>IBM Software Delivery Platform>IBM Rational Developer for Power Systems Software>IBM Rational Developer for Power Systems Software**

Your instructor can help you with any questions you have regarding the use of the Remote Systems LPEX editor or SEU.

Exercise 1. Subprocedures

What this exercise is about

This exercise provides an opportunity to code a subprocedure which can be used like an RPG IV built-in function (BIF). Your subprocedure receives input and then returns a value to the caller.

What you should be able to do

At the end of the lab, you should be able to:

- Code a subprocedure that returns a value to the caller
- Code a prototype for a subprocedure
- Code a procedure interface for a subprocedure
- Code a local variable for a subprocedure

Introduction

Given a complete loan payment calculator application, your task is to modify certain inline calculations to create subprocedures that return a value to the caller.

You are not required to write any new logic. All you are required to do is to move existing calculations to subprocedures. Then, you modify the current procedure so that it calls the subprocedures.

Exercise instructions

Step 1. Code a local subprocedure

In this portion of the exercise, you modify an existing program by moving some inline calculations into two subprocedures and writing the code necessary to call them. You code the procedure interfaces to define the parameters that are passed to the subprocedures.

__ 1. Create a new QRPGLESRC source member, **RATPER. RATPER**, a subprocedure, receives as parameters the annual interest rate and number of payments per year, returning the periodic interest rate.

__ 2. You do not have to code the prototype at this time. It is coded in a separate source member in a later step.

__ 3. Code the procedure interface for RATPER on D specs. At some point your complete subprocedure is an independent compile unit. The PI must precisely define all parameter attributes in the expected sequence that they are passed from the caller.

Use the following display file to help you to define your fields correctly:

```
A                                      INDARA
A                                      CA03(03)
A          R PAYFMT
A                                  1  2DATE
A                                      EDTCDE(Y)
A                                  1 29'Loan Payment Calculator'
A                                      DSPATR(HI)
A                                  1 61'System:'
A                                  1 70SYSNAME
A                                  2  2TIME
A                                  2 61'User:'
A                                  2 70USER
A                                  4  2'Type values, press Enter.'
A                                      COLOR(BLU)
A                                  6 18'Loan amount . . . . . .'
A            PRINCIPAL      9Y 2B  6 50EDTWRD(' ,   , 0 .  ')
A                                      TEXT('LOAN AMOUNT')
A                                      DSPATR(MDT)
A                                      COMP(GT .00)
A                                      CHECK(FE)
A                                  8 18'Annual interest % . . . . . .'
A            RATEPCANN      5Y 3B  8 50EDTWRD('0 .   ')
A                                      TEXT('ANNUAL INTEREST %')
A                                      DSPATR(MDT)
A                                      RANGE(.000 50.000)
A                                      CHECK(FE)
A                                 10 18'Payments per year . . . . . .'
A            NBRPAYYR       2Y 0B 10 50EDTWRD(' 0')
A                                      TEXT('NUMBER OF PAYMENTS PER YEA-
A                                      R')
A                                      DSPATR(MDT)
A                                      RANGE(1 52)
```

```
A                                      CHECK(FE)
A                                 12 18'Number of payments  . . . . . .'
A            NBRPAYTOT      4Y 0B 12 50EDTWRD(' ,   ')
A                                      TEXT('TOTAL NUMBER OF PAYMENTS')
A                                      DSPATR(MDT)
A                                      RANGE(1 1600)
A                                      CHECK(FE)
A                                 14 18'Periodic interest . . . . . .'
A            RATEPERIOD    13Y11O 14 50EDTCDE(4)
A                                      TEXT('DECIMAL INTEREST RATE PER-
A                                      IOD')
A                                 16 18'Periodic payment amount . . . .'
A            PAYMENTAMT    13Y 2O 16 50EDTWRD('  ,   ,   ,  0.  ')
A                                      TEXT('PAYMENT AMOUNT')
A                                      DSPATR(HI)
A            ERRMSG        40     21 35
A                                 22  2'F3=Exit'
A                                      COLOR(BLU)
```

__ 4. If you have not already done so, create the DSPF, **LOANPAYD**. (You should have created the display file as part of the Monitor Groups exercise.)

__ 5. Review **LOANPAYSP** as follows. A copy is in your library.

```
FLoanPayD  CF   E             WorkStn IndDS(LoanPDS)
D LoanPDS         DS
D  Exit                   3      3N

 /free
      ExFmt PayFmt;

      DoW NOT Exit;
          RatePeriod = ( RatePCAnn * 0.01 ) / NbrPayYr;
          PaymentAmt = (Principal*RatePeriod) /
                       (1-(1/((1+RatePeriod)**NbrPayTot)));
          ExFmt PayFmt;
      EndDo;

      *InLR = *On;
      Return;
 /End-free
```

__ 6. Code the calculations for the **RATPER** subprocedure that calculate the periodic interest rate for a loan. If you like, you can simply copy the calculation from your copy of LOANPAYSP to RATPER.

__ 7. Code the P specifications for your RATPER subprocedure.

__ 8. Code the PI for your RATPER subprocedure.

__ 9. Code a RETURN operation in your RATPER subprocedure.

__ 10. Exit and add an appropriate text description before saving your new source member.

Step 2. Code a subprocedure prototype

__ 1. Create new QRPGLESRC source member **RATPER_PR**. The **PR** suffix is an abbreviation for prototype. By coding a subprocedure prototype in a source member separate from the rest of the procedure, you can /COPY the prototype into modules that call your subprocedure, as well as modules that include it.

__ 2. Code statements in this member for a prototype for the RATPER subprocedure. Be sure that the PR statements match the requirements for the RATPER subprocedure. If necessary, refer to the PI in RATPER for help.

__ 3. Exit and add appropriate descriptive text before saving your new source member.

Step 3. Code another subprocedure and prototype

__ 1. Create another subprocedure member **PAYMNT** in your source file QRPGLESRC.

__ 2. This subprocedure receives the principal amount, rate per period, and total number of payments. It returns the actual payment amount. See the above program LOANPAYSP for the calculation of amount of the monthly payment. You can copy the calculation from your LOANPAYSP to PAYMNT.

__ 3. Exit and add an appropriate text description before saving your new source member.

__ 4. Now create another subprocedure prototype member **PAYMNT_PR** in your source file QRPGLESRC. This prototype describes the input to your PAYMNT subprocedure. If necessary, refer to the PI in PAYMNT for help.

Some programmers prefer to include all subprocedure prototypes in a single source member. This approach is acceptable, but you might find including so many unused prototypes unnecessarily cumbersome. Others prefer to reduce the number of *unreferenced* compiler messages by including prototypes for only the subprocedures that are to be included in the program. By coding each prototype in a separate source member, you can eliminate unnecessary prototypes.

__ 5. Exit and add appropriate descriptive text before saving your new source member. You have created code that can be used and reused to create local and exportable subprocedures.

Step 4. Include subprocedures in a main program

__ 1. Modify your copy of LOANPAYSP. You might want to make a backup copy of the program.

__ 2. Code statements necessary to perform the loan payment calculation using your two new subprocedures and the prototyped source members.

__ 3. Be sure that the rate and payment calculations are not performed within LOANPAYSP, but rather in your subprocedures.

__ 4. Code the necessary /**COPY** statements to include both prototype and subprocedure source members at the appropriate places in LOANPAYSP. This includes your prototypes and subprocedures in the compile unit.

__ 5. Exit and add appropriate descriptive text before saving your new source member.

__ 6. You should now have five new source members. LOANPAYSP is the member to be compiled. Using the /COPY compiler directive, it directs the compiler to include the other four source members at the appropriate points.

__ 7. Compiling program LOANPAYSP is a slightly different process. Compile your program specifying that the DFTACTGRP = *NO. If you do not specify this parameter as *NO, your compilation will fail.

__ 8. Test LOANPAYSP. Notice that this copy of the program does not have a monitor group in it; so you might experience an error if you do not enter a valid interest rate. If you like, add a monitor group to check for an interest rate error as you did earlier for LOANPAYLP. You need to include the monitor group in the appropriate subprocedure.

__ 9. If it operates correctly, you have created an RPG IV program using local subprocedures.

END OF LAB

Exercise 2. Creating ILE objects

What this exercise is about

This exercise familiarizes you with the commands that support modular programming in the Integrated Language Environment.

What you should be able to do

At the end of the lab, you should be able to:

- Create modules
- Create programs

Introduction

You are given the source code of an RPG IV procedure and a display file. Create a module and a program object from this source member.

Exercise Instructions

Step 1. Using Remote Systems LPEX Editor to create ILE objects

In this exercise you can use the Remote Systems LPEX Editor or PDM to create the objects. As always, you can switch between rsLPEX and 5250 emulation easily.

In your library, you notice several data files, **EMPMST**, **PRJMST**, and **RSNMST**. You also notice a display file, **MSTDSP**. These files are used by the RPG IV procedure, **PAYROLLG**.

If you choose to use 5250 emulation for the exercise, skip to the next step, “Using 5250 Emulation.”

__ 1. Open an edit session for your RPGLE source member **PAYROLLG**. We will use this member to reacquaint you with the various commands that can be executed from the Remote Systems LPEX Editor.

__ 2. From the editor's **Compile** menu, select **Compile Prompt**. Then select **CRTBNDRPG**.

__ 3. The Create RPG Module (CRTRPGMOD) window opens.

__ 4. Notice the generation severity level check value of 10. You could change it but we won’t.

__ 5. Check other parameters of the Create RPG Module (CRTRPGMOD) window. Specifically, notice the value of the **Debugging Views** parameter. You might need to set this to the level of debugging you need, for example ***ALL**.

__ 6. Click the **OK** button to submit your compile and close the window.

__ 7. You should see your compile messages in the lower portion of the window under the **Error List** tab.

__ 8. Switch to 5250 emulation. Look in your library for an object named **PAYROLLG**. Do you see an object type ***Module**?

__ 9. Enter option 5 next to the module to display module information. You can scroll down to view additional information for a specific item and press Enter to see different data. If you press Enter several times, for example, you notice that PAYROLLG is a procedure.

__ 10. Create a program from this module. Take option 26 in PDM and press Enter to view the CRTPGM command. Enter PAYROLLG as the program name. Notice that you can enter a plus sign (+) if this program contains more than a single module. The PAYROLLG program contains only one module, PAYROLLG.

__ 11. Check your messages that the command executed successfully.

__ 12. You should see your new *PGM object, **PAYROLLG**. You have just created an ILE module and an ILE program.

Step 2. Using 5250 Emulation

__ 1. In your library, find an RPGLE source member, PAYROLLG.

__ 2. Next to your member, enter option 15 and press F4. You see the display for the CRTRPGMOD command.

__ 3. Press Enter. Check your spool file to be sure that the module was created.

__ 4. Use WRKOBJPDM and look in your library for an object named **PAYROLLG**. Do you see an object type ***Module**?

__ 5. Enter option 5 next to the module to display module information. You can scroll down to view addition information for a specific item and press Enter to see different data. If you press Enter several times, for example, you will notice that PAYROLLG is a procedure.

__ 6. Create a program from this module. Take option 26 in PDM and press Enter to view the CRTPGM command. Enter PAYROLLG as the program name. Notice that you can enter a plus sign (+) if this program contains more than a single module. The PAYROLLG program will contain only one module, PAYROLLG.

__ 7. Check your messages that the command executed successfully.

__ 8. You should see your new *PGM object, **PAYROLLG**. You have just created an ILE module and an ILE program.

END OF LAB

Exercise 3. Bind by copy

What this exercise is about

This exercise provides an opportunity to use static binding, specifically bind by copy, wherein needed procedures are bound into an ILE program by copying the executable code from the modules containing the procedures to a program object.

What you should be able to do

At the end of the lab, you should be able to:

- Edit an RPG IV source member to change a dynamic call to a static call in the prototype
- Create RPG IV modules
- Create a multi-module ILE program using bind by copy
- Reuse existing tested modules in multiple ILE programs using bind by copy

Introduction

In this exercise, you modify your copy of VNRDLT and VNRSCHSPR to use a bound call rather than a dynamic call.

Exercise instructions

Step 1. Make copies of your existing source members

__ 1. Make copies of your **VnrDlt** and **VnrSchSPR** source members, naming them **VNRDLTPROC** and **VNRSCHMAIN**.

If you did not complete the subfile maintenance program **VnrSchSPR** in the earlier exercises, you can copy the sample solution from the course library:

Display File (DDS): **AS07V5LIB/QDDSSRC(VNDSCHS5S)** copy to **AS10nnn/QDDSSRC(VNDSCHS5S)**

RPG IV Programs: **AS07V5LIB/QRPGLESRC(VNRSCHSPR)** copy to **AS10nnn/QRPGLESRC(VNRSCHMAIN)** and **AS07V5LIB/QRPGLESRC(VNRDLTS)** copy to **AS10nnn/QRPGLESRC(VNRDLTPROC)**

__ 2. Previously, each was compiled as an individual program. **VnrSchSPR** calls **VnrDlt** using a dynamic call.

__ 3. What in the code of each program makes this a dynamic call?

_____

_____

List the changes that you have to make to each program:

In procedure VNRDLTPROC:

_____

_____

In procedure VNRSCHMAIN:

_____

_____

Step 2. Modify VNRDLTPROC and VNRSCHMAIN

__ 1. Using your documented changes above, modify the source members so that VNRSCHMAIN will use a bound call to call VNRDLTPROC.

Step 3. Create modules VNRDLTPROC and VNRSCHMAIN

__ 1. Compile each source member, creating modules.

__ 2. When you have compiled successfully, confirm that the two modules, **VNRDLTPROC** and **VNRSCHMAIN** have been created.

__ 3. Display the module information for each module. Does either module know about the other yet? _____

Step 4. Create VNRSCHMAIN *PGM

__ 1. Run the ILE CRTPGM command and prompt it. If you use LPEX, you should click **Actions** from the editor menu and select **Create Program**.

__ 2. For the program name, we use the same name as the *MODULE, VNRSCHMAIN.

__ 3. Notice that the modules to be included can be expanded using the plus sign (+). Enter a plus sign (+) and press Enter.

__ 4. Enter VNRDLTPROC and your library for the second module. Notice that more modules could be entered to be included in the VNRSCHMAIN program.

__ 5. Press F10 or in LPEX, look for the entry module box. In PDM, you see that the default is *FIRST. This means that the first module in the list is the PEP for the program. Which module is this? _____

__ 6. Press Enter to create the program.

Step 5. Test the program, VNRSCHMAIN

__ 1. As before, test that a delete option works.

__ 2. Remember to refresh your copy of the Vendor_PF file after testing, copying from the master copy in AS10XXX.

Step 6. Explore the *PGM, VNRSCHMAIN

__ 1. Find your new VNRSCHMAIN program.

__ 2. Use the DSPPGM command and prompt with F4. For the DETAIL parameter, specify *ALL. Press Enter.

Notice that the phrase `More` appears near the lower right corner of the panel, but you are also prompted to `Press Enter to continue`. To avoid missing any information, scroll your display forward through each section until `More` is replaced by `Bottom`. Then press Enter to advance to the next display to view other information.

__ 3. What is the program entry procedure module?

_____

__ 4. What is the program attribute?

_____

__ 5. What is the type of program?

_____

__ 6. Press Enter to go to the display that lists modules. How many modules are bound into this program?

_____

__ 7. The service programs are listed next. Explore the remaining information. How many service programs are bound to this program?

_____

END OF LAB

Exercise 4. Bind by reference

What this exercise is about

This exercise provides an opportunity to use static binding, specifically bind by reference, wherein one or more procedures needed by an ILE program are centrally contained in a service program, and bound by reference to the ILE program during the CRTPGM binding process.

What you should be able to do

At the end of the lab, you should be able to:

- Create a service program
- Bind by reference to modules in a service program object

Introduction

In the Bind by Copy exercise, module VNRDLTPROC was bound by copy into ILE program, VNRSCHMAIN.

It is often desirable to make commonly used modules available to the application by including them in a service program object. You now bind the VNRDLTPROC module into a new service program object, and then bind by reference to your new service program from the ILE procedure VNRSCHMAIN.

Reuse the modules from the Bind by copy exercise, and bind them together differently.

Exercise instructions

Step 1. Create service program, MySrvPgm

__ 1. Use the CRTSRVPGM command to create a service program that contains the module VNRDLTPROC. Specify the parameter **EXPORT(*ALL)**:

```
CRTSRVPGM   SRVPGM(MYSRVPGM)
            MODULE(VNRDLTPROC)
            EXPORT(*ALL)
```

Export indicates that all export capable symbols can be referenced beyond the scope of the object.

__ 2. What type of object did you create?

_____

Service programs usually contain more than one module. You just created a service program with only one module.

__ 3. What happens if you try to execute the command:

```
CALL MYSRVPGM
```

_____

A stand-alone service program cannot be called dynamically.

Step 2. Create and test a new program, VNRSCHREF

__ 1. Now create a new program that functions like VNRSCHMAIN in the Bind by copy exercise. Instead of binding the VNRDLTPROC module as we did in the bind by copy, all you need to do is bind the service programs that you just created. When you run CRTPGM, specify the program name as VNRSCHREF and bind the VNRSCHMAIN as you did before. You need to press F10 to see the parameter for binding of the service program.

__ 2. Test VNRSCHREF as you have before.

Step 3. Explore your service program and program object

__ 1. Run the DSPPGM command to explore the information available for the VNRSCHREF. Press the Enter key to move from display to display.

Which module is the program entry procedure? Why?

_____

__ 2. Press Enter and stop when you reach the Modules display. VNRDLTPROC was not specified in the CRTPGM command prompt. Rather, it was bound by reference to a service program.

__ 3. Press Enter and stop at the screen that displays information about service programs. Enter a 5 beside your MYSRVPGM and press Enter until you see the module VNRDLTPROC.

__ 4. Explore more. Make a note of any points of interest (Signature, Exports, and so on). Make a note of any questions and review them with the instructor and the rest of the class at the end of the exercise.

END OF LAB

Exercise 5. Using system APIs I

What this exercise is about

In this exercise, you code two programs and use three APIs. Using an API, you put data in a data queue that you created. Then, in the second program, you retrieve the data in the queue and display a program message that contains the data.

What you should be able to do

At the end of the lab, you should be able to:

- Use the QSNDDTAQ API to put information in a data queue
- Use the QRCVDTAQ API to retrieve information from a data queue
- Use the QMHSNDPM API to send a program message

Introduction

You use APIs in two programs.

Exercise instructions

Step 1: Create a data queue

The exercise requires that you perform three steps:

1. Create a data queue in your student library.
2. Write a program to out a specific message in your data queue.
3. Write another program to retrieve the message from the data queue and then send that message to your external message queue.

You will use three APIs to perform these tasks within your two programs.

__ 1. In your library, you will find a source member **DTAQPROTO** in QRPGLESRC. Review the source member, noticing that it has prototypes for two APIs. Write down the names of the APIs in the prototypes: _____

Study the prototypes carefully. The source member follows:

```
* Prototype for API QSNDDTAQ - Send To a Data Queue
D SndDtaQ         PR                  EXTPGM('QSNDDTAQ')
D  DataQueueNam                 10A   Const
D  DataQueueLib                 10A   Const
D  DataLength                    5P 0 Const
D  DataBuffer                32767A   Const Options(*Varsize)
 * Optional parameter group (Keyed DTAQ)
D  KeyLength                     3P 0 Const Options(*Nopass)
D  KeyBuffer                   256A   Const Options(*Nopass : *Varsize)
D  AsyncRqs                     10A   Const Options(*Nopass : *Varsize)
D  DataFrmJrn                   10A   Const Options(*Nopass)
 *
 * Prototype for API QRCVDTAQ - Received From a Data Queue
D RcvDtaQ         PR                  EXTPGM('QRCVDTAQ')
D  DataQueueNam                 10A   Const
D  DataQueueLib                 10A   Const
D  DataLength                    5P 0
D  DataBuffer                32767A         Options(*Varsize)
D  WaitTime                      5P 0 Const
 * Optional parameter group 1 (Keyed DTAQ)
D  KeyOrder                      2A   Const Options(*Nopass)
D  KeyLength                     3P 0 Const Options(*Nopass)
D  KeyBuffer                   256A         Options(*Nopass : *Varsize)
D  SndLength                     3P 0 Const Options(*Nopass)
D  SndBuffer                    44A         Options(*Nopass : *Varsize)
 * Optional parameter group 2
D  RemoveMsg                    10A   Const Options(*Nopass : *Omit)
D  RcvSize                       5P 0 Const Options(*Nopass : *Omit)
D  Error                     32767A         Options(*Nopass : *Varsize)
```

__ 2. Open your Web browser and open the link to the i Information Center:

http://publib.boulder.ibm.com/infocenter/iseries/v7r1m0

__ a. Expand the version in the left pane.

__ b. Expand **Programming**. Then, also in the left pane, expand **Application programming interfaces**.

__ c. Select **API finder**.

__ d. For each of the two APIs in the above prototype, enter the name of the API in the Find by name box.

__ e. For each of the APIs in the above prototype, carefully review the parameters in the documentation and make sure that you understand how the above prototypes were coded based on the API parameter definitions.

Step 2: Create a data queue

__ 3. Create a data queue in your library. Name your data queue, AS10*nnn*, where *nnn* is your assigned team number for labs. Make your data queue a maximum of 50. Take the defaults for the remaining parameters.

Step 3: Code the program to put data in your queue

__ 4. Code a program that uses the QSNDDTAQ API. The program should be named **SNDDTAQ**. You may copy the existing prototype into your program or code your own from scratch. Define any fields that the program will pass to the API. Assign values to the name of the data queue and the data that will be placed in the queue. The data to put in your queue should be:

```
This is a message from AS10nnn's Lab Exercise
```

Step 4: Code the program to retrieve the data in the queue and send a message

__ 5. Code a second program named **RCVSNDMSG**.

__ 6. This program requires prototypes for the QRCVDTAQ API and the QMHSNDPM API. You may use the DTAQPROTO source member for the QRCVDTAQ API, but you code the prototype for the send message API yourself. Use your lecture notes and the information center API finder to assist you. The message that your program sends contains the data that you loaded in your data queue. The message should be an information message sent to your external message queue.

__ 7. Define any program variables needed to pass as parameters to each API.

__ a. Hints:

- Call Stack Entry = *EXT
- Call Stack Counter = 0
- Error Code = 16A initialized to X'00'

__ 8. In your calculations, retrieve the contents of your data queue by calling the QRCVDTAQ API. Then send the data that your retrieved as a program message by calling the QMHSNDPM API.

Step 5: Compile and test your programs

__ 9. Compile both programs. Once they have successfully compiled, call **SNDDTAQ**. Then call **RCVSNDMSG**. You should see a message that states:

```
This is a message from AS10nnn's Lab Exercise
```

__ 10. Make any changes to your programs until you succeed. Ask your instructor for assistance as required.

End of exercise

Exercise 6. Using system APIs II

What this exercise is about

In this exercise, you code a program that prompts for an object name and displays owner information. You also use the API error handling data structure (DS), QUSEC.

What you should be able to do

At the end of the lab, you should be able to:

- Use the QUSROBJD API to retrieve object information
- Use the QUSEC data structure for basic error handling

Introduction

You use the QUSROBJD API and error handling.

Exercise instructions

Step 1: Review the display file

__ 1. In your library, you find a source member **DSPDOBJI** in QDDSSRC. Review the source member, noticing the record formats: OBJ_PROMPT, OBJ_DETAIL, FKEYS, and MSG.

Study the source member that follows:

```
A                                      INDARA
A                                      CA03(03 'Exit')
 **
A          R OBJ_PROMPT                OVERLAY
A                                    1  2USER
A                                    1 30'Display Object Description'
A                                       DSPATR(HI)
A                                       COLOR(WHT)
A                                    1 71SYSNAME
A                                    2 61DATE
A                                       EDTCDE(Y)
A                                    2 71TIME
A                                    4  2'Enter Object Name:'
A            OBJNAME       10A  B  4 25
A                                    5  2'Enter Object Type:'
A            OBJTYPE        7A  B  5 25
A                                    6  2'Enter Object Library:'
A            OBJLIB        10A  B  6 25
A          R OBJ_DETAIL                 OVERLAY
A                                    9  2'Object Owner:'
A            OBJOWNER      10A  O  9 25
A                                   10  2'Object Creation Date:'
A            OBJCRTDT      13A  O 10 25
A                                   11  2'Object Change Date:'
A            OBJCHGDT      13A  O 11 25
A          R FKEYS                      OVERLAY
A                                   20  7'Press Enter to continue'
A                                   21  7'F3=Exit' COLOR(BLU)
A          R MSG
A                                       OVERLAY
A  99                               15  1'Error Code:'
A  99        EXCEPTID       7A      15 15
A  99                               16  1'Exception Data:'
A  99        EXCDATA2      80A      18  1
A  99                                   DSPATR(HI)
```

__ 2. Open your Web browser and open the link to the i Information Center:

http://publib.boulder.ibm.com/infocenter/iseries/v7r1m0/

__ a. Find and select **Programming > Application Programming Interfaces**.

__ b. Select **API finder**.

__ c. Enter the name of the API, QUSROBJD, in the **Find by name** box.

__ d. Review the parameter definitions.

__ e. Scroll down, looking for the **Error code Parameter** link. Click the link.

__ f. Review the documentation for the error code parameter and read about the two formats available to you. Notice especially the input parameter(s) for each format. You will use format ERRC0100.

Step 2: Code the program

__ 3. Code a program that calls the QUSROBJD API. The program should be named **DSPROBJI**. Use the DSPF (you may improve it if you like) from the previous step above. Some further information follows:

__ a. Define the prototype for the QUSROBJD API. Use your lecture notes and the source member in QSYSINC to guide you in coding your prototype. Be certain to define the **QUSEC** parameter that will be passed to the data structure. Remember that this parameter is variable in size.

__ b. Use the OBJD0100 format for QUSROBJD. Define a DS using the member in QSYSINC and your lecture notes.

__ c. Define a DS to hold the contents of the **QUSEC** parameter. Assign a value to the bytes provided subfield that will cause **QUSEC** to handle any error. Review the API documentation for detail on this subfield.

__ d. Code your logic to display the message ID and the contents of the exception data in the display file when there is an exception based on the value of the bytes available subfield of the QUSEC DS. You can display as much as you like, modifying the display file as necessary. Notice the use of indicator 99 in the display file for an error.

__ e. When there is no exception, display the owner and date information as described in the display file.

__ f. Notice the use of F3 and indicator 03 for the exit key.

Step 3: Compile and test your program

__ 4. Compile your program.

__ 5. Test your program using a valid object name, such as:

```
Enter Object Name:     QDDSSRC
Enter Object Type:     *FILE
Enter Object Library:  AS10nnn
```

__ 6. Test your program using an invalid object name. Specify **QDSRC**.

__ 7. Next, change the initial value of the **Bytes Provided** field in your source program to zero, compile it, and test it by making the error in the previous step again. What happens? You will get an exception message!

__ 8. Make any changes to your program until you succeed in the test. Ask your instructor for assistance as required.

End of exercise

Exercise 7. Using conditional compiler directives

What this exercise is about

This lab covers the use of conditional directives.

What you should be able to do

At the end of the lab, you should be able to:

- Use /Define and /Undefine with a condition
- Use /If, /ElseIf and /Endif to selectively copy source code

Introduction

Compiler directives enable you to select what code is to be included in the compilation. One use would be to select which prototypes should be copied from a copy member. In this exercise, you will modify a copy member and a source procedure to copy only the prototypes needed for the procedure.

Exercise instructions

Step 1: Compile and test existing source

__ 1. In your library, you will find source members for the display file **LOANPAYD**. Compile the display file:

```
A*
A* Calc/display loan payment display file LOANPAYD
A*
A                                      INDARA
A                                      CA03(03)
A          R PAYFMT
A                                1  2DATE
A                                      EDTCDE(Y)
A                                1 29'Loan Payment Calculator'
A                                      DSPATR(HI)
A                                1 61'System:'
A                                1 70SYSNAME
A                                2  2TIME
A                                2 61'User:'
A                                2 70USER
A                                4  2'Type values, press Enter.'
A                                      COLOR(BLU)
A                                6 18'Loan amount . . . . . .'
A            PRINCIPAL      9Y 2B  6 50EDTWRD(' ,   , 0 .  ')
A                                      TEXT('LOAN AMOUNT')
A                                      DSPATR(MDT)
A                                      COMP(GT .00)
A                                      CHECK(FE)
A                                8 18'Annual interest % . . . . . .'
A            RATEPCANN      5Y 3B  8 50EDTWRD('0 .   ')
A                                      TEXT('ANNUAL INTEREST %')
A                                      DSPATR(MDT)
A                                      RANGE(.000 50.000)
A                                      CHECK(FE)
A                               10 18'Payments per year . . . . . .'
A            NBRPAYYR       2Y 0B 10 50EDTWRD(' 0')
A                                      TEXT('NUMBER OF PAYMENTS PER YEAR'
A                                      DSPATR(MDT)
A                                      RANGE(1 52)
A                                      CHECK(FE)
A                               12 18'Number of payments  . . . . . .
A            NBRPAYTOT      4Y 0B 12 50EDTWRD(' ,   ')
A                                      TEXT('TOTAL NUMBER OF PAYMENTS')
A                                      DSPATR(MDT)
A                                      RANGE(1 1600)
A                                      CHECK(FE)
A                               14 18'Periodic interest . . . . . .
A            RATEPERIOD    13Y11O 14 50EDTCDE(4)
A                                      TEXT('DECIMAL INTEREST RATE PER-
A                                      IOD')
A                               16 18'Periodic payment amount . . . .'
A            PAYMENTAMT    13Y 2O 16 50EDTWRD('  ,   ,   ,  0.  ')
```

```
A                                       TEXT('PAYMENT AMOUNT')
A                                       DSPATR(HI)
A            ERRMSG        40     21 35
A                                  22  2'F3=Exit'
A                                       COLOR(BLU)
```

__ 2. You will also find a copy of the **LOANPAY** procedure:

```
FLoanPayD  CF   E             WorkStn IndDS(LoanPDS)
D LoanPDS         DS
D  Exit                   3      3N
 /Copy prototypes
 /free
      ExFmt PayFmt;

      DoW NOT Exit;
          RatePeriod = Ratper(RatePCAnn:NbrPayYr);
          PaymentAmt = Paymnt(Principal:RatePeriod:NbrPayTot);

          ExFmt PayFmt;
      EndDo;

      *InLR = *On;
      Return;
 /end-free
 /Copy RatPer
 /Copy Paymnt
```

__ 3. **LOANPAY** contains subprocedures, **RatPer** and **PayMnt**. Review them in your library.

__ 4. Compile **LOANPAY,** specifying **DFTACTGRP *NO**. This procedure cannot run in the default activation group. It requires the facilities of ILE.

__ 5. Once your program has compiled, test it. You may use any values you want for the various fields.

Step 2: Add conditional compiler directives

__ 6. Review the compilation listing for **LOANPAY**. Notice that there are a number of prototypes that are not referenced in the listing. All prototypes are held in one source member, **PROTOTYPES**. You should be able to review your members, **PROTOTYPES** and **LOANPAY**, and determine which prototypes are not necessary in your program **LOANPAY**.

__ 7. Modify the source members to direct the compiler to copy only those prototypes that are referenced in the **LOANPAY** program.

__ a. Make copies of these members and modify them.

Step 3: Compile and test your modified program

__ 8. Review your compilation listing. When you look at it, you should see only the prototypes from the member **PROTOTYPE** that are needed in the procedure **LOANPAY**. The listing will also display when lines of code from the copy member were included or excluded. Look for this information.

__ 9. Test your program. Now there are no unreferenced prototypes.

End of exercise

Exercise 8. Using list APIs

What this exercise is about

This lab covers filling and retrieving object information in a user space.

What you should be able to do

At the end of the lab, you should be able to:

- Create a user space
- Fill the user space with object-related information
- Retrieve object information from the user space

Introduction

You are given three programs, used in lecture. You will modify these programs to retrieve information using a more detailed format.

Exercise instructions

Step 1: Review the source members given to you

__ 1. In your library, you will find three source members: **CRTUSPACE**, **FILLUSPACE**, and **RTVUSPACE**. Find them and review them. Copies are included below as well:

CRTUSPACE

```
D Message         C                   'Unable to create User Space'

D SpaceName       S             20A   Inz('APISPACE  *CURLIB')
D Attribute       S             10A   Inz('API_SPACE')
D Size            S             10I 0 Inz(5000)
D InitValue       S              1A   Inz('*')
D Authority       S             10A   Inz('*USE')
D Text            S             50A   Inz('AS10nnn User Space')
D Replace         S             10A   Inz('*YES')
D ErrorCode       DS
D  BytesAvl                     10I 0 Inz(%Size(ErrorCode))
D  BytesRet                     10I 0
D  MsgId                         7A
D  Reserved                      1A
D  MsgDta                       84A

D CreateUSpace    PR                  Extpgm('QUSCRTUS')
D                               20A   Const
D                               10A   Const
D                               10I 0 Const
D                                1A   Const
D                               10A   Const
D                               50A   Const
D                               10A   Const
D                              100A   Options(*Varsize)

 /Free
  CallP CreateUSpace(SpaceName : Attribute : Size : InitValue : Authority :
                     Text : Replace : Errorcode);
  If BytesRet <> 0;
     Dsply Message '*EXT';
  Endif;
  *InLR = *On;
 /End-free
```

FILLUSPACE:

```
D DspFileObj      PR                  ExtPgm('QUSLOBJ')
D APIUSSpace                    20A
D ObjFormat                      8A
D QualifObject                  20A
D ObjType                       10A
D Error                        100A   Options(*varsize)
  // Error Handling DS
D ErrorDS         DS
D  BytesAvl                     10I 0 Inz(%Size(ErrorDs))
D  BytesRet                     10I 0
D  MsgId                         7A
D  Reserved                      1A
D  MsgDta                       84A
D OBJL0100        DS
D  ObjectName                   10A
D  ObjectLib                    10A
D  ObjectTyp                    10A
 // Program variable definitions
D QualSpace       S             20A   Inz('APISPACE  *CURLIB')
D Format          S              8A   Inz('OBJL0100')
D QualObject      S             20A   Inz('*ALL      *CURLIB')
D ObjectType      S             10A   Inz('*FILE    ')
  //  (STEP 2) List all *FILE objects in Current Library
 /Free
  CallP  DspFileObj (QualSpace : Format : QualObject :
                     ObjectType : ErrorDS);
  *InLR = *on;
 /End-free
```

RTVUSPACE:

```
  // Prototype for QUSRTVUS Control information
D RtvUsrSpCtl     PR                  ExtPgm('QUSRTVUS')
D  SpaceName                    20A
D  StartPosition                10I 0
D  DataLength                   10I 0
D  ReceiverVar                 192A
D  Error                       100A   Options(*varsize)
  // Error Handling DS
D ErrorDS         DS
D  BytesAvl                     10I 0 Inz(%Size(ErrorDs))
D  BytesRet                     10I 0
D  MsgId                         7A
D  Reserve                       1A
D  MsgDta                       84A
  // Prototype for QUSRTVUS Data Information
D RtvUsrSpData    PR                  ExtPgm('QUSRTVUS')
D  SpaceName                    20A
D  StartPosition                10I 0
D  DataLength                   10I 0
D  ReceiverVar                  30A
D  Error                       100A   Options(*varsize)
  // INPUT parameters for QUSRTVUS
D QualSpace       S             20A   Inz('APISPACE  *CURLIB')
D StartPos        S             10I 0
D DataLength      S             10I 0
D Count           S                   Like(LstEntNo)
D
D Error           S             16A
  // OUTPUT parameters for QUSTRTVUS for return data
D Receiver1       DS
D  UserArea                     64
D  GenHdrSize                   10I 0
D  RelLevel                      4A
D  FormatUsed                    8A
D  APIUsed                      10A
D  CrtDatTim                    13A
D  InfoStatus                    1A
D  SizeSpace                    10I 0
D  IPSOffset                    10I 0
D  IPSSize                      10I 0
D  HdrOffset                    10I 0
D  HdrSize                      10I 0
D  LstOffset                    10I 0
D  LstSize                      10I 0
D  LstEntNo                     10I 0
D  LstEntSize                   10I 0
D  LstEntCCSD                   10I 0
D  CountryId                     2A
D  LanguageId                    3A
D  SubsetInd                     1A
D  Reserved                     42A
D
  // OUTPUT parameters - for return data (Format OBJL0100 List Entry)
```

```
D Receiver2       DS
D  ObjName                      10A
D  ObjLib                       10A
D  ObjType                      10A
D
  //  (STEP 3) Retrieve control information from User Space Generic Hea
 /Free
  StartPos = 1;
  DataLength = %Size(Receiver1);
  CallP RtvUsrSpCtl (QualSpace : StartPos : DataLength : Receiver1
                   : ErrorDS);

 // (STEP 4) Retrieve list entries from User Space
 StartPos = LstOffset + 1;
 DataLength = %Size(Receiver2);
 Count = 1;

 Dow Count <= LstEntNo;
     CallP RtvUsrSpData (QualSpace : StartPos : DataLength : Receiver2
                         Error);
   StartPos = StartPos + LstEntSize;
     Count = Count + 1;
     Dsply ObjName '*REQUESTER';
 Enddo;

 *InLR = *ON;
/End-free
```

Step 2: Modify the application

__ 2. For the program, **CRTUSPACE**, modify the variable named *Text* by replacing the *nnn* of AS10*nnn* with your assigned team number.

__ 3. Compile and execute the **CRTUSPACE** program. Check your library to make certain that the user space APISPACE was created.

__ 4. Next, you modify the **FILLUSPACE** and **RTVUSPACE** programs to use a more detailed format of the QUSLOBJ API.

__ 5. Modify the **FILLUSPACE** program:

__ a. Remove the OBJL0100 data structure in the **FILLUSPACE** program. Change the value of the *Format* variable to use the OBJL0200 format. Locate and review the format in the QUSLOBJ member of QRPGLESRC in QSYSINC. Doing this will acquaint you with the other format for data in the user space.

__ b. Compile and execute the **FILLUSPACE** program.

__ c. Confirm that the *USRSPC object contains information. Run the command:

```
DSPF '/QSYS.LIB/AS10nnn.LIB/APISPACE.USRSPC'
```

to review the contents of your APISPACE *USRSPC.

__ 6. If the User Space is empty, determine the problem with your **FILLSPACE** program and try again.

__ 7. If you want to start again, you should delete the user space using this command:

```
DLTUSRSPC APISPACE
```

__ 8. Modify the **RTVUSPACE** program:

__ a. Using your lecture notes and the information center, change the **RTVUSPACE** program to use pointers and the QUSPTRUS API rather than the QUSRTVUS API. Reference the Information Center for further detail about the parameters required for the QUSPTRUS API.

Modify the **RECEIVER2** data structure to accommodate additional subfields for OBJL0200 format.

__ b. Compile and execute the **RTVUSPACE** program.

__ c. Make further changes. Modify the DSPLY operation to present the Object Extended Attribute as well as the Object Name. You will need to define and employ a new variable plus build the message text using basic string handling functions.

__ 9. Check your message queue. You should see a list of all the objects in the course library.

End of exercise

Exercise 9. Using bindable CEE APIs

What this exercise is about

This lab covers using several ILE APIs. Specifically, you will modify an existing procedure by adding function using the APIs.

What you should be able to do

At the end of the lab, you should be able to:

- Use ILE CEE APIs in your applications
- Use the IBM i Information Center to find information about ILE CEE APIs

Introduction

In this exercise, you modify the given AGEDEMO program. You add more information on the display that shows the day of the week of the birth date.

Exercise instructions

Step 1: Explore the code that is provided

__ 1. In your QDDSSRC file, you should find:

__ a. **AGEINQ** DSPF Age demo + no. of days. Make a copy of it and name it **AGEINQX5**.

__ 2. In your QRPGLESRC file, you should find:

__ a. **AGEDEMO** RPGLE Ex 5- Age Demo with Subprocs inline. Make a copy of it and name it **AGEDEMOX5**.

Step 2: Create the objects

__ 3. Compile the QDDSSRC member above.

__ 4. Create the **AGEDEMOX5** program (this member calls the NbrDays subprocedure). Be sure to compile this program with the default activation group parameter (DFTACTGRP) set to ***NO**.

__ 5. Call the **AGEDEMOX5** program. You should see a display that prompts you for a birthdate and determines how many months since the last birthday and some other information. For this exercise, you will determine what day of the week the individual was born and display that on the screen.

Step 3: Decide what APIs to use

__ 6. In the lecture, we discussed several date and time APIs. There were others that we did not discuss, including an API to determine the day of the week.

__ 7. Open your browser and enter the following for the Web site:

http://publib.boulder.ibm.com/infocenter/iseries/v7r1m0

__ 8. On the left, expand **IBM i 7.1 Information Center**. Expand the **Programming** topic and then expand **Application Programming Interfaces.** Click **APIs by Category** so that you can see all of the API categories on the right.

__ 9. On the right, click the **ILE CEE** category. You are now reading the ILE CEE API documentation in the i Information Center. You may browse the information as you like or go directly to **Date and Time APIs** by clicking the link.

__ 10. If you look closely, you will see an API that returns the day of the week. What is it called? _____

__ 11. What are the parameters it expects and in what format should they be passed?

_____

_____

_____

__ 12. Now that you know what parameters are required, are there any other APIs that you will need to use in addition to the day of the week API? List the names of all APIs that apply.

_____

_____

_____

Hint: You need to pass a Lillian date.

Step 4: Modify your display file and RPG IV procedure

__ 13. In the DSPF, AGEINQX5, add the code to display the day of the week for the birthdate entered below the 'Month(s) since your last Birthday....' field. You may use any text you want to describe the output field that holds the day of the week of the birthdate entered on the screen. Also define the day of the week field. Name it anything you want.

__ 14. In your RPG IV procedure, **AGEDEMOX5**:

__ a. Add the logic to determine the day of the week to display in the display file.

__ b. Define the prototypes for the APIs that you will use, as well as any other fields that you require in order to pass valid parameters to the APIs. You may omit passing the feedback parameter unless you want to include it.

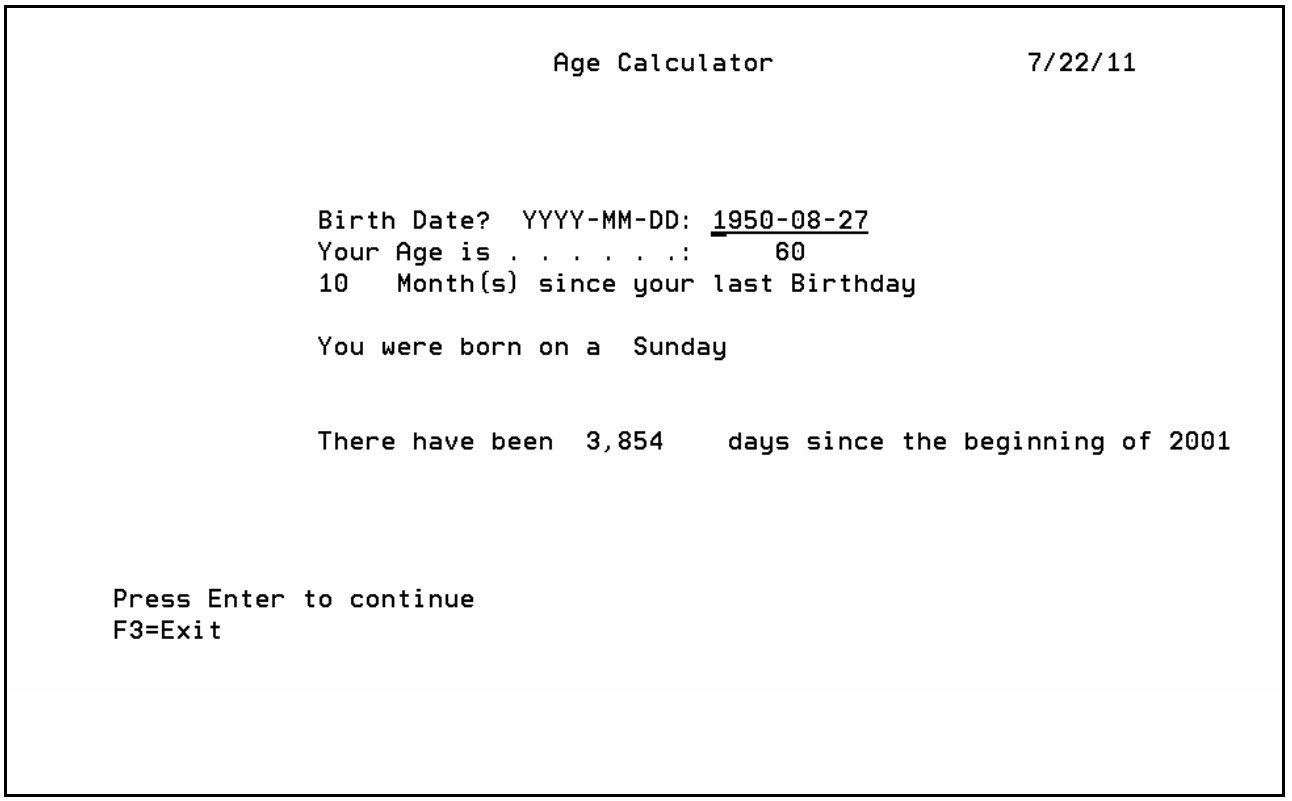
__ c. In order to display the day of the week, define an array that holds the day names, starting with Sunday.

Step 5: Compile and test

__ 15. Create a new DSPF from the modified AGEINQX5 source member.

__ 16. Create a new program from the modified procedure, AGEDEMOX5.

__ 17. Test your program. It should produce output similar to the following.



End of exercise

Exercise 10. Database triggers

What this exercise is about

This exercise provides an opportunity to code a trigger program and add it to a database file.

What you should be able to do

At the end of the lab, you should be able to:

- Write a trigger program in RPG IV
- Add a trigger to a database file

Introduction

In this exercise, you complete the coding for a trigger program, which prints a document to confirm that a customer order on file has been deleted.

You add the trigger to the ORDERHDR file as an *AFTER, *DELETE trigger. If successful, when a record is deleted from the ORDERHDR file, your trigger produces a document in your output queue that is meant to be FAXed to the customer.

Exercise instructions

Step 1: Understanding the data

__ 1. You are given the following information:

__ a. Field reference file (DICTIONARY) data description specifications (DDS):

```
     *
     *  FOR THE CUSTOMER FILE:
     *
A            CUSNBR          5A         TEXT('CUSTOMER NUMBER')
A                                       COLHDG('CUST' 'NO')
A            CUSNAM         20A         COLHDG('CUSTOMER' 'NAME')
A                                       TEXT('CUSTOMER NAME')
A            CUSTEL         15A         TEXT('CUSTOMER PHONE NUMBER')
A                                       COLHDG('CUSTOMER' 'PHONE' +
A                                       'NUMBER')
A            CUSFAX         15A         COLHDG('CUSTOMER' 'FAX' 'NUMBER')
A                                       TEXT('CUSTOMER FAX NUMBER')
A            CUSADR         20A         COLHDG('CUSTOMER' 'ADDRESS')
A                                       TEXT('CUSTOMER ADDRESS')
A            CUSCTY         20A         COLHDG('CUSTOMER' 'CITY')
A                                       TEXT('CUSTOMER CITY')
A            CUSZIP          5A         COLHDG('CUST' 'ZIP' 'CODE')
A                                       TEXT('CUSTOMER ZIP CODE')
A            CUSRCD         11  2       COLHDG('CUSTOMER' 'CREDIT' 'LIMIT')
A                                       TEXT('CUSTOMER CREDIT LIMIT')
A            CUSTOT         11  2       COLHDG('CUSTOMER' 'TOTAL' 'AMOUNT')

A                                       TEXT('CUSTOMER TOTAL AMOUNT')
     *
     *  FOR THE ORDERHDR FILE:
     *
A            ORHNBR          5A         TEXT('ORDER NUMBER')
A                                       COLHDG('ORDER' 'NO')
     *       CUSNBR
A            ORHDTE           L         TEXT('ORDER DATE'  )
A                                       COLHDG('ORDER' 'DATE')
A            ORHDLY           L         COLHDG('ORDER' 'DLVRY' 'DATE')
A                                       TEXT('ORDER DELIVERY DATE')
A            ORHTOT         11  2       COLHDG('ORDER' 'TOTAL')
A                                       TEXT('ORDER TOTAL')
A            SRNBR          10A         COLHDG('SALES' 'REP' 'NUMBER')
A                                       TEXT('SALES REP. NUMBER')
```

__ b. Customer master file (CUSTOMER) DDS:

```
     *
     *  FOR THE CUSTOMER FILE:
     *
    A                                      UNIQUE
    A                                      REF(DICTIONARY)
    A          R CUSREC
    A            CUSNBR    R
    A            CUSNAM    R
    A            CUSTEL    R
    A            CUSFAX    R
    A            CUSADR    R
    A            CUSCTY    R
    A            CUSZIP    R
    A            CUSRCD    R
    A            CUSTOT    R
```

__ c. Customer master file (CUSTOMER) data (First part of records):

CUST	CUSTOMER NO	NAME	CUSTOMER PHONE NUMBER	CUSTOMER FAX NUMBER
000001	00001	GREGORY HUMPHRIES	507-280-6570	507-286-4666
000002	00002	BRAD AUGUSTINE	918-622-8865	918-622-8765
000003	00003	JANICE ARMSTRONG	617-543-7373	617-543-7388
000004	00004	DOUG SHRAUGER	716-883-8627	716-883-8768
000005	00005	DONNA STOCKTON	415-883-3839	415-883-9466

__ d. Customer master file (CUSTOMER) data (Second part of records):

ADDRESS	CITY	ZIP CODE	CREDIT LIMIT
3605 WATSON BLVD.	BETHESDA, MD	55901	1,000.00
47 ELSMERE AVE.	ENDWELL, NY	09401	1,500.00
6996 WATERLOO ROAD	ITHACA, NY	20001	1,000.00
7 FOX CREEK RD.	SCRANTON, PA	45366	500.00
5 TACKAWANNA ST.	BUFFALO, NY	10020	1,000.00

__ e. Customer master file (CUSTOMER) data (Third part of records):

CUSTOMER TOTAL AMOUNT
837.50
27.50
475.00
72.40
.00

__ f. Order header file (ORDERHDR) DDS:

```
     *  FOR THE ORDERHDR FILE:
     *
    A                                      UNIQUE
    A                                      REF(DICTIONARY)
    A          R ORDHDRREC
    A            ORHNBR    R
    A            CUSNBR    R
    A            ORHDTE    R
    A            ORHDLY    R
    A            ORHTOT    R
    A            SRNBR     R
    A          K ORHNBR
```

__ g. Order header file (ORDERHDR) data:

	ORDER NO	CUST NO	ORDER DATE	ORDER DLVRY DATE	ORDER TOTAL	SALES REP NUMBER
000001	00001	00001	1993-01-01	1994-01-01	785.00	00006
000002	00002	00003	1994-08-19	1994-09-01	475.00	00007
000003	00003	00004	1994-09-01	1995-01-01	72.40	00006
000004	00004	00001	1994-08-31	1994-09-30	52.50	00007
000005	00005	00002	1994-08-31	1995-09-01	27.50	00005

__ h. Printer file (OEPFAX) DDS. You will need to create the printer file (PRTF):

```
A                                      REF(*LIBL/DICTIONARY)
A          R OEPFAX_FMT
A                                    2 54DATE EDTCDE(Y)
A                                    2 45'Date:  '
A                                    3 45'Time:  '
A                                    3 54TIME
A                                    5  5'To:  '
A            CUSNAM    R        O  5 12
A                                    5 45'FAX:'
A            CUSFAX    R        O  5 51
A                                   13  5'Order date:'
A            ORHDTE    R     A  O 13 19
A                                   14  5'Order delivery date:'
A            ORHDLY    R     A  O 14 27
A                                   15  5'Order total:'
 **
A            ORHTOT    R        O 15 19EDTWRD('   ,   ,  0.  CR')
A                                    8  5'From:  RPG Office Supply Company'
A                                   16  5'Please refer to order number'
A            ORHNBR    R        O 16 35
A                                   16 42'in all future correspondence'
A                                   18  5'Sincerely,'
A                                   21  5'John Doe'
A                                   22  5'818-555-1111'
A                                   10  5'Subject:  Order Deletion Confirmat-
A                                        ion'
A                                   12  5'This is to confirm the cancellatio-
A                                        n of your recent order.'
```

__ i. Source code for the fixed part of trigger parameter 1. This member is in your student library:

```
File QRPGLESRC, Member TRIGPLIST
D TrigPlist       DS
D*                                              Trigger Buffer
D FileName                      10A
D*                                              File Name
D LibName                       10A
D*                                              Library Name
D FileMember                    10A
D*                                              Member Name
D TrigEvent                      1A
D*                                              Trigger Event
D TrigTime                       1A
D*                                              Trigger Time
D CommitLockLvl                  1A
D*                                              Commit Lock Level
D Reserved1                      3A
D*                                              Reserved 1
D CCSID                         10I 0
D*                                              CCSID
D CurrentRRN                    10I 0
D*                                              Current Rrn
D Reserved2                      4A
D*                                              Reserved 2
D OldRecOffset                  10I 0
D*                                              Old Record Offset
D OldRecLen                     10I 0
D*                                              Old Record Len
D ORecNullOffset                10I 0
D*                                              Old Rec Null Map Offset
D ORecNullLength                10I 0
D*                                              Old Rec Null Map Length
D NewRecOffset                  10I 0
D*                                              New Record Offset
D NewRecLen                     10I 0
D*                                              New Record Len
D NRecNullOffset                10I 0
D*                                              New Rec Null Map Offset
D NRecNullLength                10I 0
D*                                              New Rec Null Map Length
D TrigBuffRes                   16A
D*                                              Reserved
```

Step 2: Refresh your database files

__ 2. If you want to replace the data in a modified file, call CPYF using the data in the corresponding file in library AS10XXX, the master student collection. You will need to temporarily disable the trigger that you added before you will be able to perform the CPYF using the **CHGPFTRG** command.

Step 3: Remove any existing triggers

__ 3. Determine if any trigger programs have already been added to your ORDERHDR file:

```
DSPFD FILE(AS10nnn/ORDERHDR) TYPE(*TRG)
```

where *nnn* is your team number.

__ 4. If any trigger programs have already been added to your ORDERHDR file, remove them with the **RMVPFTRG** command:

```
RMVPFTRG FILE(AS10nnn/ORDERHDR) TRGTIME(*ALL) TRGEVENT(*ALL)
```

Step 4: Understand the lab task

__ 5. If a customer order is deleted, you will print a document confirming the deleted order. This document will be FAXed later to the customer. You do not have to be concerned about FAXing the confirmation.

The following is a sample FAX that would be produced if order 00002 is deleted from ORDERHDR:

```
Date:     7/03/02
                                          Time:     8:11:51
To:    JANICE ARMSTRONG                   FAX:  617-543-7388
From:  RPG Office Supply Company
Subject:  Order Deletion Confirmation
This is to confirm the cancellation of your recent order.
Order date:   1994-08-19
Order delivery date:  1994-09-01
Order total:          475.00
Please refer to order number  00002  in all future correspondence.
Sincerely,
John Doe
818-555-1111
```

__ 6. The following sample shows the above sample document with the field names defined in the OEPFAX printer file substituted for the actual data:

```
Date:    07/03/02 (System)
                                          Time:    12:54:41 (System)
To:    (CUSNAM)                           FAX:  (CUSFAX)
From:  RPG Office Supply Company
Subject:  Order Deletion Confirmation
This is to confirm the cancellation of your recent order.
Order date:   (ORHDTE)
Order delivery date:  (ORHDLY)
Order total:           (ORHTOT)
Please refer to order number  (ORHNBR)  in all future correspondence.
Sincerely,
John Doe
818-555-1111
```

Step 5: Coding the trigger program

__ 7. Create the printer file OEPFAX in the your library AS10*nnn*. Use this file to print the FAX document in your trigger program.

__ 8. Write the trigger program OERFAXT. This program should be executed whenever a record in the ORDERHDR file is deleted.

__ 9. Use your lecture notes and the program that uses the pointer method as a guide when you write your trigger. We do not need to be concerned about the new record image, so you do not have to use a prefix in the record image data structure.

__ 10. You need to declare the customer file as the fax needs the customer name field from this file.

__ 11. A description of the logic of the program follows:

- __ a. The trigger determines the location of the (old) record image in the trigger buffer.
- __ b. Once this has been done, you should chain using the customer number field from the ORDERHDR file.
- __ c. Then, assuming a successful CHAIN, you should print the fax.
- __ d. Finally, end the trigger program. For each record that is deleted, the trigger will be called again.

Step 6: Compile and test your trigger program

__ 12. Compile your program, OERFAXT.

__ 13. Add your program **OERFAXT** as an *AFTER, *DELETE trigger to your data file, ORDERHDR, in your library. Use the command **ADDPFTRG**.

__ 14. Use the **DSPFD** command to confirm that the trigger has been added.

__ 15. Use DFU or SQL to delete a record from the ORDERHDR file.

__ 16. Check your output queue and verify that the information in the FAX is correct.

__ 17. If your trigger does not produce any output, check that the files contain data; if there are still problems, you can debug your trigger program. It will come up in debug mode even if the database manager fires it.

__ 18. Remove the trigger that you added to your ORDERHDR file:

```
RMVPFTRG FILE(AS10nnn/ORDERHDR) TRGTIME(*ALL) TRGEVENT(*ALL)
```

End of exercise

Exercise 11. Enhancing NOMAIN service program

What this exercise is about

In the exercise, you continue the process that we have been showing in the lecture unit. You add more subprocedures to the service program, recreate all impacted objects, and test the application, which is much more modular.

What you should be able to do

At the end of the lab, you should be able to:

- Write a NOMAIN procedure
- Create a service program containing NOMAIN procedures
- Create modules and ILE programs that use bind by reference to handle calls to subprocedures in a service program

Introduction

You find copies of all the code that was demonstrated in class. In addition, you add some existing subprocedures that were written in previous exercises to the NOMAIN procedure.

You place all prototypes in a single copy member, and, using conditional compiler directives, you copy only the prototypes referenced by your procedures to each procedure as you compile the module.

You re-create all objects and test all applications to make sure that everything still produces correct results.

Exercise instructions

Step 1: Explore the code that is provided

__ 1. In your QDDSSRC file, you should find:

__ a. AGEINQ DSPF Age demo and NbrDays

__ b. ITEMINQ2 DSPF Item inquiry with NbrDays

__ 2. In your QRPGLESRC file, you should find:

__ a. AGEDEMOMN RPGLE Ex 7- Age demo with subprocs in NOMAIN proc

__ b. ITEMINQ2 RPGLE Ex 7- Item inquiry with call to NbrDays

__ c. SUBPROCSNM RPGLE Ex 7- Subprocs in NOMAIN proc

__ 3. You also use other source members that you created in earlier exercises.

Step 2: Create the objects

__ 4. Compile the QDDSSRC members above.

__ 5. Create the AGEDEMOMN module (this member calls NbrDays but the subprocedure is in a NOMAIN procedure).

__ 6. Create the SubProcsNM module.

__ 7. Create a service program named **AS10*nnn***, where *nnn* is your team number. Specify SubProcsNM for the module and ***ALL** for the export parameter.

__ 8. Create a program, **AgeInqPGM**, that contains the AgeDemoMN module and binds by reference to your service program, **AS10*nnn***.

__ 9. Explore the objects you have created. If you are using the Programming Development Manager (PDM), put the number `5` beside a module, program, or service program object in order to display the details. As an alternative, you could use the **DSPMOD**, **DSPPGM,** and **DSPSRVPGM** commands respectively. When you use these commands, you can scroll for more information and press Enter to see different information. Use both. For the service program, specifically look for procedure exports and the service program signature. Make a note of the procedures exported:

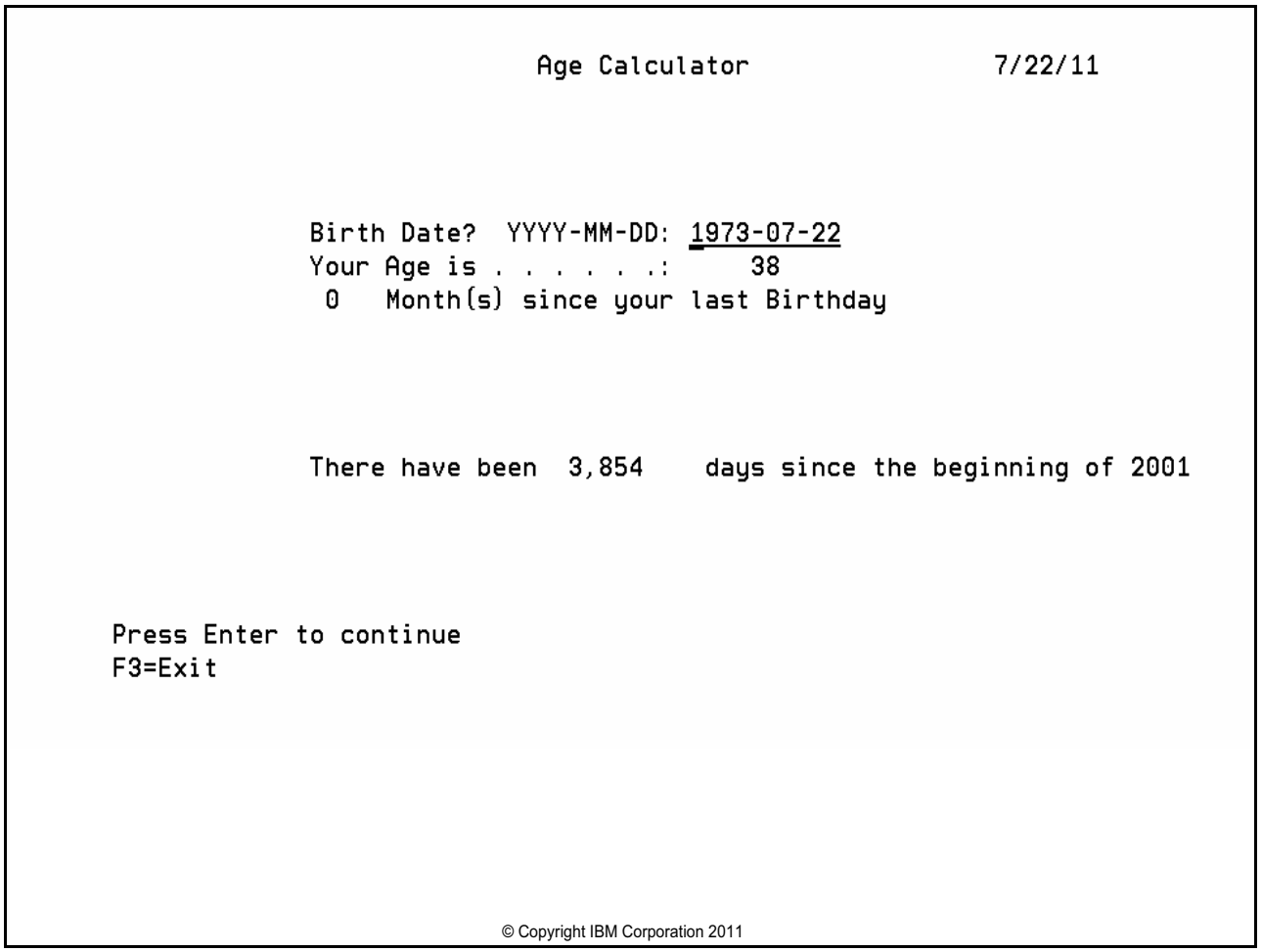
_____

_____

_____

Step 3: Test the application

__ 10. Call the **AgeInqPGM**. You should see something similar to this display:



__ 11. If you see different results, you should check for any errors you made in binding.

Step 4: Create more objects and test

__ 12. Create the module ItemInq2 that also calls **NbrDays**.

__ 13. Create a program, **ItemInq2** from the module and bind it to your service program, **AS10*nnn***.

__ 14. Call the **ItemInq2** program. You should see a display similar to this:

```
ITEMINQOV                    Item Inquiry                    7/22/11

              There have been  3,854    days since the beginning of 2001


              Item Number . . . . . .: 00000









Press Enter to continue
F3=Exit
```

Valid item numbers are 20001 - 20050

__ 15. If you have different results, check your module creation and binding parameters.

Step 5: Add more subprocedures to NOMAIN procedure

We have decided to place all subprocedures in a NOMAIN module. We will then organize NOMAIN modules by application. For now, we will start with some of the subprocedures that have been written to date and put them into a single module.

__ 16. In your QRPGLESRC file, find your copies of RATPER and PAYMNT subprocedures and copy them into the SubprocsNM NOMAIN source member.

__ 17. Remember to include prototypes for these subprocedures.

__ 18. Re-create the SubProcsNM module and then re-create your service program to contain this new module.

__ 19. Re-create the two programs that you created in the previous step to reference this new copy of your service program.

__ 20. Test the two programs that you have re-created. They should behave as they did before.

__ 21. Explore the objects you have created again. Use commands **DSPMOD**, **DSPSRVPGM**, and **DSPPGM**. For the service program, specifically look for procedure exports and the service program signature. Make a note of the procedures exported:

_____

_____

_____

Step 6: Modify prototypes

At this point, you have reorganized the NOMAIN subprocedure. In exercise 2, you used conditional directives to /COPY prototypes into calling programs. You used a member named PROTOTYPES.

__ 22. Modify your Prototypes source member to include the **NbrDays** prototype.

__ 23. Modify the procedures that call **NBRDAYS**, **ItemInq2,** and **AgeDemoMN** such that they copy from the Prototypes member only the prototype definitions that are required.

__ 24. Re-create all objects in the application that are impacted by this change. These are the AgeDemoMn and ItemINQ2 objects.

__ 25. Modify the source member SUBPROCSNM to use compiler directives and /COPY for its PROTOTYPES. Then recreate the *MODULE and finally the *SRVPGM **AS10*nnn***.

Step 7: Modify LOANPAY

__ 26. Edit your copy of LOANPAY. If you did not complete exercise 3, now modify it so that it uses conditional directives and the PROTOTYPES source member.

__ 27. Remember that RATPER and PAYMNT are now included in the SUBPROCSNM module that is included in your **AS10*nnn*** service program.

__ 28. Create your LOANPAY module and a new *PGM, **LOANPAYPGM**, that binds the service program.

__ 29. Test your **LOANPAYPGM** to make sure that it behaves as it did earlier.

End of exercise

Exercise 12. Using binding directories and binder language

What this exercise is about

This exercise provides an opportunity to create and use a binding directory. You create a binding directory and add entries for your modules and service program.

Subsequently, you reference your binding directory when creating ILE programs to simplify the number of modules that must be specified on the **CRTPGM** command.

Then, you use the binder language to handle exports of data and procedures.

What you should be able to do

At the end of the lab, you should be able to:

- Create a binding directory
- Reference a binding directory on the **CRTPGM** command to simplify ILE program creation and maintenance
- Add and remove binding directory entries
- Create and use the binder language

Introduction

Using a *binding directory*, you can reduce the number of modules that must be specified on the **CRTPGM** command to just one. You can also eliminate the requirement to specify any service programs.

Some programmers have compared a binding directory to a library search list. Each clearly serves a different purpose, but they can both be used to reduce keystrokes because their contents can be searched to help fill in the blanks.

The *binder language*, as you have seen, helps to manage exports from a service program. Not only that, it also helps simplify the process of application releases by minimizing the impact of changes on existing code.

Exercise instructions

Step 1: Create a binding directory

__ 1. Create a new binding directory MYBNDDIR in your library:

```
CRTBNDDIR  BNDDIR(AS10nnn/Mybnddir)
```

__ 2. Inspect your binding directory to confirm that it contains no entries:

```
WRKBNDDIRE  BNDDIR(AS10nnn/Mybnddir)
```

Step 2: Add entries to a binding directory

Modules and service programs need not exist before adding their names to a binding directory. You can add individual module names and service program names to a binding directory, or you can do a global add of either all module names or all service program names.

__ 3. Add entries for the names of all modules in your team library to your binding directory:

On the Work with Binding Directory Entries panel, type these entries on the top entry line:

```
                      Work with Binding Directory Entries

 Binding Directory:    MYBNDDIR       Library:    AS10V6LIB

 Type options, press Enter.
   1=Add   4=Remove

                                                    -------Creation--------
 Opt   Object       Type      Library      Activation   Date         Time
  1    *all______   *module   AS10nnn___   __________

   (No binding directory entries for this binding directory.)




                                                                      Bottom
 Parameters or command
 ===> _________________________________________________________________________
 F3=Exit   F4=Prompt   F9=Retrieve   F5=Refresh   F12=Cancel   F17=Top
 F18=Bottom
```

OR key the following command:

```
ADDBNDDIRE BNDDIR(MYBNDDIR) OBJ((AS10nnn/*ALL *MODULE))
```

__ 4. Inspect your binding directory to be sure that only the intended entries exist. If you added entries via the Work with Binding Directory Entries display, the added entries will appear automatically. If you used the **ADDBNDDIRE** command to add entries, use the following command to view them:

```
WRKBNDDIRE  BNDDIR(MYBNDDIR)
```

Your binding directory should now contain one entry for each module in your library (AS10*nnn*). If you find any other entries, remove them with option 4.

Step 3: Use your binding directory to create ILE programs

In the previous exercise, you created an ILE program, **LOANPAYPGM**. Now, you will create a new ILE program that will perform the same functions as **LOANPAYPGM**, but you will use a binding directory to create a new ILE program, **LOANBD**.

__ 5. In the previous step, you added all modules in your library to your MYBNDDIR. Verify that your binding directory contains the modules needed by LOANPAY, RATPER, and PAYMNT. Remember that RATPER and PAYMNT are now included in the SUBPROCSNM module.

__ 6. Create a new program, **LOANBD**, specifying the LOANPAY module and your binding directory.

__ 7. When you try to execute the **CRTPGM** command, you *may* see a message that the program was not created. Do not worry if you do not see:

```
Message . . . . :   Definition supplied multiple times for symbol 'NBRDAYS'.
Cause . . . . . :   Definition NBRDAYS was found to be exported from both
  *MODULE object AGEDEMO in library AS10nnn and *MODULE object SUBPROCSNM in
  library AS10nnn.
Recovery  . . . :   Try the Create Program (CRTPGM) or Create Service Program
  (CRTSRVPGM) command again, supplying only one of these objects, or try the
  CRTPGM or CRTSRVPGM command again, specifying one or both of OPTION(*DUPVAR)
  and OPTION(*DUPPROC).
```

__ 8. If you see the message, you will need to take appropriate action to create the program successfully. The purpose of this part of the exercise is not only to use a binding directory, but also to learn about additional features such as the duplicate export problem above.

Step 4: Test your program, LOANBD

__ 9. Test your program to make sure that it still runs as it did in the first exercise.

Step 5: Using the binder language for exports

You recall that you built a service program, **AS10*nnn***, in a previous exercise. You modify the service program using the binder language to export the procedure and data items needed rather than specifying ***ALL** for the **EXPORT** parameter of **CRTSRVPGM**.

__ 10. First, display your **AS10*nnn*** service program to determine what items need to be exported. What symbols need to be exported?

_____

__ 11. Create a new source member, NOMAINBD, in your source file, QSRVSRC.

__ 12. Code the binder source necessary to export the symbols you noted in the first part of this step.

Step 6: Create objects

__ 13. Create a new service program, **NOMAINBD**. Make sure that you include the SUBPROCSNM module and reference your binder language for any exports.

__ 14. Create a program, **AGEBIND**, that includes the main module, AGEDEMOMN and the service program you created.

Step 7: Test your program

__ 15. Test the program **AGEBIND** that you just created.

Step 8: Create objects for LOANPAY program

__ 16. Create a program, **LOANBIND**, that includes the main module, LOANPAY and the service program **NOMAINBD** that you created.

__ 17. If your program does not bind because all imports are not satisfied, check your binder language member, NOMAINBD.

__ 18. Try to create **LOANBIND** using the module LOANPAYAPI (LOANPAYAPI *MODULE exists in *LIBL; it is in course library AS10V3LIB, referencing service program **NOMAINBD**). Is it created successfully?

What happened? Why? What would you have to do to solve any problems that you encountered? Be prepared to discuss your observations with the class.

_____

_____

_____

_____

_____

_____

_____

_____

End of exercise

Exercise 13. Enhancing the condition handler

What this exercise is about

This lab covers using an ILE condition handler to manage errors.

What you should be able to do

At the end of the lab, you should be able to:

- Register an ILE condition handler
- Write a condition handler to handle errors
- Handle specific errors and indicate a resume action
- Percolate unanticipated errors

Introduction

In this exercise, you are given an existing procedure and its error handler. These are the examples that we used in the lecture. You modify both such that you handle a specific error, divide by zero, and percolate all other errors.

Exercise instructions

Step 1: Create objects and test existing application

__ 1. In your QRPGLESRC file, you should find two source members, CauseErr and CondHdlr.

__ 2. Review the code. You will notice that the CauseErr procedure has been modified since we used it in the lecture. It has a new error added to it (array index error). CondHdlr is unchanged.

__ 3. Create modules for CauseErr and CondHdlr and then create a program, **ILECond** where CauseErr is the entry procedure module.

__ 4. Test your program. Your MSGQ should contain the messages that follow:

```
DSPLY  Starting CauseErr
DSPLY  In CauseErr - causing Division error
DSPLY  Starting CondHdlr
DSPLY  Resume CauseErr
DSPLY  In CauseErr Subr - error detected
DSPLY  In CauseErr - causing Index error
DSPLY  Starting CondHdlr
DSPLY  Starting CondHdlr
DSPLY  Starting CondHdlr
DSPLY  Starting CondHdlr
DSPLY  In CauseErr Subr - error detected
```

__ 5. Check your joblog. It will contain the following messages:

```
Attempt made to divide by zero for fixed point operation.
Range of subscript value or character string error.
Receiver value too small to hold result.
Range of subscript value or character string error.
Receiver value too small to hold result.
```

__ 6. Explore the messages in the job log and note the message numbers below:

_____

_____

_____

_____

Step 2: Enhance the condition handler

__ 7. Make a copy of the handler, CondHdlr, and name it CondHdlrE.

__ 8. Modify your copy, CondHdlrE, and make the following changes:

__ a. Confirm that there is a pointer pointing to InToken.

__ b. Add the code to check for a specific divide by zero error. Use the joblog notes above to obtain the prefix and message number for the subfields in the CondToken DS. Note that CondMsgNo is a hex value in a character field. Set the error indicator on and set the action to resume.

__ c. For all other errors, simply set the action to percolate.

__ d. Add any messages and use the **Dsply** opcode to inform you of where you are in your handler. You may add as many messages as desired.

Step 3: Create objects and test

__ 9. Create your modules and a new program, **ILECONDE**.

__ 10. Call ILECONDE and check the MSGQ and joblog. Your MSGQ should look something like this:

```
DSPLY  Starting CauseErr
DSPLY  In CauseErr - causing Division error
DSPLY  Starting CondHdlr
DSPLY  Resume CauseErr
DSPLY  In CauseErr Subr - error detected
DSPLY  In CauseErr - causing Index error
DSPLY  Starting CondHdlr
DSPLY  Starting CondHdlr
```

Your joblog should look something like this:

```
Attempt made to divide by zero for fixed point operation.
Range of subscript value or character string error.
An array index is out of range (C G D F).
C
An array index is out of range (C G D F).
C
Application error.  MCH0603 unmonitored by ILECONDE at statement
  0000000045, instruction X'0000'.
```

__ 11. Make notes below about the behavior that you observed when testing your condition handler:

_____

_____

_____

_____

_____

End of exercise

Appendix A. Exercise Solutions

Exercise 1: Subprocedures

RPG IV: LOANPAYSP

```
FLoanPayD  CF   E             WorkStn IndDS(LoanPDS)
D LoanPDS         DS
D  Exit                   3      3N
 /Copy RatPer_PR
 /Copy Paymnt_PR

 /free
      ExFmt PayFmt;

      DoW NOT Exit;
          RatePeriod = Ratper(RatePCAnn:NbrPayYr);
          PaymentAmt = Paymnt(Principal:RatePeriod:NbrPayTot);

          ExFmt PayFmt;
      EndDo;

      *InLR = *On;
      Return;
 /end-free
 /Copy RatPer
 /Copy Paymnt
```

RPG IV: Paymnt

```
PPaymnt           B

** PAYMNT - Calc loan payment SUBPROCEDURE

DPaymnt           PI             9  2
DPrincipal                       9  2
DRatePeriod                     13 11
DNbrPayTot                       4  0

 /Free

  Return (Principal*RatePeriod) /
         (1-(1/((1+RatePeriod)**NbrPayTot)));

 /End-free
PPaymnt           E
```

RPG IV: Paymnt_PR

```
DPaymnt           PR             9  2

** Calc loan payment PROTOTYPE

DPrincipal                       9  2
```

```
DRatePeriod                     13 11
DNbrPayTot                       4  0
```

RPG IV: Ratper

```
PRatPer           B

** RATPER - Calc dec periodic interest rate SUBPROCEDURE

D                 PI            13 11
DRatePCAnn                       5  3
DNbrPayYr                        2  0

 /Free

  Return ( RatePCAnn * 0.01 ) / NbrPayYr;

 /End-free

PRatPer           E
```

RPG IV: RatPer_PR

```
DRatPer           PR            13 11

** RATPER - Calc dec periodic interest rate PROTOTYPE

DRatePCAnn                       5  3
DNbrPayYr                        2  0
```

Compilation Listing:

```
1 FLoanPayD  CF   E             WorkStn IndDS(LoanPDS)
    *------------------------------------------------------------------------
    *                             RPG name         External name
    * File name. . . . . . . . . :  LOANPAYD         AS07V2LIB/LOANPAYD
    * Record format(s) . . . . . :  PAYFMT           PAYFMT
    *------------------------------------------------------------------------
 2 D LoanPDS         DS
 3 D  Exit                    3      3N
 4  /Copy RatPer_PR
    *------------------------------------------------------------------------
    * RPG member name  . . . . . :  RATPER_PR
    * External name  . . . . . . :  AS07V2LIB/QRPGLESRC(RATPER_PR)
    * Last change  . . . . . . . :  07/31/03  14:58:09
    * Text 'description' . . . . :  Ex 15 - Calc periodic interest rate PR
    *------------------------------------------------------------------------
 5+
 6+DRatPer           PR            13 11
 7+
 8+** RATPER - Calc dec periodic interest rate PROTOTYPE
 9+
10+DRatePCAnn                      5  3
11+DNbrPayYr                       2  0
12+
```

```
13  /Copy Paymnt_PR
    *----------------------------------------------------------------
    * RPG member name  . . . . . :  PAYMNT_PR
    * External name  . . . . . :  AS07V2LIB/QRPGLESRC(PAYMNT_PR)
    * Last change  . . . . . :  07/31/03  14:58:09
    * Text 'description' . . . . :  Ex 16 - Calc loan payment PROTOTYPE
    *----------------------------------------------------------------
14+
15+DPaymnt          PR             9  2
16+
17+ ** Calc loan payment PROTOTYPE
18+
19+DPrincipal                      9  2
20+DRatePeriod                    13 11
21+DNbrPayTot                     4  0
22+
23
24  /free
25=IPAYFMT
    *----------------------------------------------------------------
    * RPG record format  . . . . :  PAYFMT
    * External format  . . . . . :  PAYFMT : AS07V2LIB/LOANPAYD
    *----------------------------------------------------------------
26=I                            S    1    9 2PRINCIPAL
27=I                            S   10   14 3RATEPCANN
28=I                            S   15   16 0NBRPAYYR
29=I                            S   17   20 0NBRPAYTOT
30      ExFmt PayFmt;
31
32      DoW NOT Exit;
33          RatePeriod = Ratper(RatePCAnn:NbrPayYr);
34          PaymentAmt = Paymnt(Principal:RatePeriod:NbrPayTot);
35
36          ExFmt PayFmt;
37      EndDo;
38
39      *InLR = *On;
40      Return;
41  /end-free
42  /Copy RatPer
    *----------------------------------------------------------------
    * RPG member name  . . . . . :  RATPER
    * External name  . . . . . :  AS07V2LIB/QRPGLESRC(RATPER)
    * Last change  . . . . . :  07/31/03  14:58:09
    * Text 'description' . . . . :  Ex 15 - Calc periodic interest rate SU
    *----------------------------------------------------------------
43=OPAYFMT
    *----------------------------------------------------------------
    * RPG record format  . . . . :  PAYFMT
    * External format  . . . . . :  PAYFMT : AS07V2LIB/LOANPAYD
    *----------------------------------------------------------------
44=O                        PRINCIPAL            9S ZONE      9,2
45=O                        RATEPCANN           14S ZONE      5,3
46=O                        NBRPAYYR            16S ZONE      2,0
47=O                        NBRPAYTOT           20S ZONE      4,0
48=O                        RATEPERIOD          33S ZONE     13,11
```

```
49=O                       PAYMENTAMT          46S ZONE     13,2
50=O                       ERRMSG              86A CHAR       40
51+PRatPer           B
52+
53+** RATPER - Calc dec periodic interest rate SUBPROCEDURE
54+
55+D                  PI            13 11
56+DRatePCAnn                       5  3
57+DNbrPayYr                        2  0
58+
59+ /Free
60+
61+  Return ( RatePCAnn * 0.01 ) / NbrPayYr;
62+
63+ /End-free
64+
65+PRatPer           E
66+
67   /Copy Paymnt
     *------------------------------------------------------------------
     * RPG member name  . . . . . :  PAYMNT
     * External name  . . . . . . :  AS07V2LIB/QRPGLESRC(PAYMNT)
     * Last change  . . . . . . . :  07/31/03  14:58:09
     * Text 'description' . . . . :  Ex 16 - Calc loan payment SUBPROCEDURE
     *------------------------------------------------------------------
68+PPaymnt           B
69+
70+** PAYMNT - Calc loan payment SUBPROCEDURE
71+
72+DPaymnt           PI             9  2
73+DPrincipal                      9  2
74+DRatePeriod                    13 11
75+DNbrPayTot                      4  0
76+
77+ /Free
78+
79+  Return (Principal*RatePeriod) /
80+         (1-(1/((1+RatePeriod)**NbrPayTot)));
81+
82+ /End-free
83+PPaymnt           E
```

Exercise 2: Creating ILE objects

No solution necessary.

Exercise 3: Bind by copy

Step 1 -3 The Extpgm keyword on the prototype.

VNRDLTPROC - in the prototype, change the ExtPGM keyword to ExtPROC and the name of the procedure to VNRDLTPROC rather than VNRDLT

VNRSCHMAIN - in the prototype, change the ExtPGM keyword to ExtPROC and the name of the procedure to VNRDLTPROC rather than VNRDLT

Step 3-3 The VNRSCHMAIN procedure lists the VNRDLTPROC Module in the Imported (unresolved) symbols display. The import request would be resolved when we run CRTPGM.

Step 4-5 PEP is in module VNRSUBMAIN.

Step 6-3 VNRSCHMAIN.

Step 6-4 RPGLE

Step 6-5 ILE

Step 6-6 two modules (VNRSCHMAIN and VNRDLTPROC)

Step 6-7 four; all system modules in QSYS

Prototype is modified in both VNRDLTPROC and VNRSCHMAIN:

```
D VnrDelete       PR                  ExtProc('VNRDLTPROC')
D  VndNbr                        5  0
```

CRTPGM:

Create Program (CRTPGM)

Type choices, press Enter.

```
Program  . . . . . . . . . . . . > VNRSCHMAIN    Name
  Library  . . . . . . . . . . . >   AS07nnLIB   Name, *CURLIB
Module . . . . . . . . . . . . . > VNRSCHMAIN    Name, generic*, *PGM, *ALL
  Library  . . . . . . . . . . . >   AS07nnLIB   Name, *LIBL, *CURLIB...
            + for more values > VNRDLTPROC
                              >   AS07nnLIB
```

```
Text 'description' . . . . . .   *ENTMODTXT

                        Additional Parameters

Program entry procedure module     *FIRST       Name, *FIRST, *ONLY, *PGM
  Library  . . . . . . . . . . .                  Name, *LIBL, *CURLIB...
```

Exercise 4: Bind by reference

Step 1 - 2 *SRVPGM

Step 1 - 3 Error message; you cannot call a service program.

Step 2 - 1

```
CRTPGM PGM(AS07nnLIB/VNRSCHREF)
       MODULE(AS07nnLIB/VNRSCHMAIN)
       BNDSRVPGM(AS07nnLIB/MYSRVPGM)
```

Step 3 - 1 VNRSCHMAIN is the PEP; it is the first (and only) module referenced.

Exercise 5 Using system APIs I

```
CRTDTAQ DTAQ(AS10V6LIB/AS10) MAXLEN(50)

D/Copy DtaQProto
 // Program variable definitions
D  DtaQName       S             10A
D  DtaQLib        S             10A   Inz('*CURLIB')
D  DataSnd        S             45A

 /Free
  DtaQName='AS10';
  DataSnd='This is a message from AS10V6''s Lab Exercise';

  SndDtaQ(DtaQName:DtaQLib:%Len(%TrimR(DataSnd)):DataSnd);
  *InlR = *on;
 /End-free

D/Copy DtaQProto
  // Prototype for QMHSNDPM
D SendPgmMsg      PR                  ExtPgm('QMHSNDPM')
D                                 7A
D                                20A
D                                45A
D                                10I 0
D                                10A
D                                10A
D                                10I 0
D                                 4A
D                                 6A
 // Program variable definitions for receive dataqueue
D  DtaQName       S             10A
D  DtaQLib        S             10A   Inz('*CURLIB')
D  Length         S              5P 0
D  Data           S             45A
D  Wait           S              5P 0 Inz(-1)
 // Program variables for Send Program Message
D ErrorMsgId      S              7A
D MsgFile         S             20A
D MsgData         S             45A
D MsgDtaLen       S             10I 0 Inz(%Len(MsgData))
D MsgType         S             10A   Inz('*INFO')
D CallStack       S             10A   Inz('*EXT')
D CallStackCtr    S             10I 0 Inz(0)
D MsgKey          S              4A
D ErrorCode       S             16A   Inz(X'00')

 /Free
  DtaQName='AS10';
  RcvDtaQ (DtaQName:DtaQLib:Length:Data:Wait);
  Msgdata = Data;
   // Send message that contains contents of my data queue
```

```
  SendPgmMsg (ErrorMsgId : MsgFile : MsgData : MsgDtaLen : MsgType:
       CallStack : CallStackCtr : MsgKey : ErrorCode);
  *InLR = *on;
 /End-free
```

Exercise 6 Using system APIs II

```
FDspDObjIS CF   E             WorkStn InDDS(DspIndic)

D DspIndic        DS
D   Exit                         1N   Overlay(DspIndic:03)
D   Error                        1N   Overlay(DspIndic:99)

D RtvObjD         Pr                  ExtPgm('QUSROBJD')
D ObjDta                        90A
D ObjDtaLen                     10I 0 Const
D FormatName                     8A   Const
D ObjectLib                     20A   Const
D ObjectType                    10A   Const
D ErrorInf                     200A   Options(*varsize)

  // Format OBJD0100 with received fields
DQUSD01DS         DS
D DataRet                       10I 0
D DataAvail                     10I 0
D ObjectName                    10A
D LibName                       10A
D ObjectType                    10A
D ReturnLib                     10A
D AuXStorPool                   10I 0
D ObjectOwn                     10A
D ObjectDomain                   2A
D CrtDatTime                    13A
D ChgDatTime                    13A
  // QUSEC Data Structure for error handling
DQusEC            DS
D BytesProv                     10I 0 Inz(200)
D BytesAvail                    10I 0
D ExceptID                       7A
D Reserved                       1A
D ExceptData                   200A
  // Program variable definitions
D ObjNameLib      S             20A
 /Free
  Write FKeys;  // Write to buffer
  Exfmt Obj_Prompt;

  DoW not Exit;
     ObjNameLib = ObjName + ObjLib;
  // Call QUSROBJD to retreive information about object entered by user
  CallP RtvObjD(QUSD01DS : %Len(QUSD01DS) :
        'OBJD0100' : ObjNameLib : ObjType : QusEC);

  If BytesAvail = 0;
     ObjOwner = ObjectOwn ;
     ObjCrtDt = CrtDatTime;
     ObjChgDt = ChgDatTime;
     Write Obj_Detail;
  Else;
     Error = *on;
```

```
     ExcData1 = ExceptID + %subst(ExceptData:1:40);
     ExcData2 = %subst(ExceptData:41:80);
  EndIf;
     Write Msg;
     Error = *off;
  Exfmt Obj_Prompt;

  Enddo;
    // F3 pressed; user wants to exit program
  *InLR = *on;
 /END-FREE
```

Exercise 7 Using conditional compiler directives

Prototypes:

```
 /If defined (DLTPROC)
  // Prototype for DLTPROC
D VnrDelete       PR                  ExtProc('DLTPROC')
D  VndNbr                        5  0
 /EndIf
 /If defined (DayofWeek)
  // Prototype for DayofWeek
D DayOfWeek       PR             1S 0
D                                 D
 /EndIf
 /If defined (LoanPay)
  // Prototype for loan payment
DPaymnt           PR             9  2

 ** Calc loan payment PROTOTYPE

DPrincipal                       9  2
DRatePeriod                     13 11
DNbrPayTot                       4  0
  // Prototype for periodic rate
DRatPer           PR            13 11

** RATPER - Calc dec periodic interest rate PROTOTYPE

DRatePCAnn                       5  3
DNbrPayYr                        2  0
 /EndIf
```

LOANPAY:

```
FLoanPayD  CF   E             WorkStn IndDS(LoanPDS)
D LoanPDS         DS
D  Exit                   3      3N
 /Define LoanPay
 /Copy prototypes
 /Undefine LoanPay
 /free
      ExFmt PayFmt;

      DoW NOT Exit;
          RatePeriod = Ratper(RatePCAnn:NbrPayYr);
          PaymentAmt = Paymnt(Principal:RatePeriod:NbrPayTot);

          ExFmt PayFmt;
      EndDo;

      *InLR = *On;
      Return;
 /end-free
 /Copy RatPer
 /Copy Paymnt
```

Compile listing extracts:

```
 4  /Define LoanPay
 5  /Copy prototypes
    *----------------------------------------------------------------------
    * RPG member name  . . . . . :  PROTOTYPES
    * External name  . . . . . . :  AS10V6LIB/QRPGLESRC(PROTOTYPES)
    * Last change  . . . . . . . :  05/23/09 10:22:17
    *----------------------------------------------------------------------
6+ /If defined (DLTPROC)
        LINES EXCLUDED: 3
 7+ /ElseIf defined (DayofWeek)
        LINES EXCLUDED: 3
 8+ /ElseIf defined (LoanPay)
 9+  // Prototype for loan payment
10+DPaymnt           PR             9  2
11+
12+ ** Calc loan payment PROTOTYPE
13+
14+DPrincipal                      9  2
15+DRatePeriod                   13 11
16+DNbrPayTot                     4  0
17+  // Prototype for periodic rate
18+DRatPer            PR         13 11
19+
20+** RATPER - Calc dec periodic interest rate PROTOTYPE
21+
22+DRatePCAnn                      5  3
23+DNbrPayYr                      2  0
24+ /EndIf
25  /Undefine LoanPay
26  /free
```

Exercise 8 Using list APIs

CRTUSPACE:

```
D Message         C                   'Unable to create User Space'
D
D SpaceName       S             20A   Inz('APISPACE  *CURLIB')
D Attribute       S             10A   Inz('API_SPACE')
D Size            S             10I 0 Inz(5000)
D InitValue       S              1A   Inz('*')
D Authority       S             10A   Inz('*USE')
D Text            S             50A   Inz('AS10INS User Space')
D Replace         S             10A   Inz('*YES')
D ErrorCode       DS
D  BytesAvl                     10I 0 Inz(%Size(ErrorCode))
D  BytesRet                     10I 0
D  MsgId                         7A
D  Reserved                      1A
D  MsgDta                       84A

D CreateUSpace    PR                  Extpgm('QUSCRTUS')
D                               20A   Const
D                               10A   Const
D                               10I 0 Const
D                                1A   Const
D                               10A   Const
D                               50A   Const
D                               10A   Const
D                              100A   Options(*Varsize)

 /Free
  CallP CreateUSpace(SpaceName : Attribute : Size : InitValue : Authority :
                     Text : Replace : Errorcode);
  If BytesRet <> 0;
     Dsply Message '*EXT';
  Endif;
  *InLR = *On;
 /End-free
```

FILLUSPACE:

```
D DspFileObj      PR                  ExtPgm('QUSLOBJ')
D APIUSSpace                    20A
D ObjFormat                      8A
D QualifObject                  20A
D ObjType                       10A
D Error                        100A   Options(*varsize)
  // Error Handling DS
D ErrorDS         DS
D  BytesAvl                     10I 0 Inz(%Size(ErrorDs))
D  BytesRet                     10I 0
D  MsgId                         7A
D  Reserve                       1A
D  MsgDta                       84A
D OBJL0200        DS
D ObjectName                    10A
D ObjectLib                     10A
D ObjectTyp                     10A
D InfoStatus                     1A
D ExtObjAttr                    10A
D TextDescript                  50A
D UserDefAttr                   10A
D Reserved                       7A
 // Program variable definitions
D QualSpace       S             20A   Inz('APISPACE  *CURLIB')
D Format          S              8A   Inz('OBJL0200')
D QualObject      S             20A   Inz('*ALL      *CURLIB')
D ObjectType      S             10A   Inz('*FILE    ')
  //  (STEP 2) List all *FILE objects in Current Library
 /Free
  CallP  DspFileObj (QualSpace : Format : QualObject :
                     ObjectType : ErrorDS);
  *InLR = *on;
 /End-free
```

RTVUSPACE:

```
// Prototype for QUSRTVUS Data Information
D RtvUsrSpData    PR                  ExtPgm('QUSRTVUS')
D  SpaceName                    20A
D  StartPosition                10I 0
D  DataLength                   10I 0
D  ReceiverVar                  30A
D  Error                        16A
  // INPUT parameters for QUSRTVUS
D QualSpace       S             20A   Inz('APISPACE  *CURLIB')
D Ptr1            S               *
D Ptr2            S               *
D Count           S                   Like(LstEntNo)
  // Field to hold DSPLY Output
D DspOut          S             30A
  // Decode of required elements of Generic Header
D Receiver1       DS                  Based(Ptr1)
D Receiver1       DS                  Based(Ptr1)
D   LstOffset                   10I 0 Overlay(Receiver1:125)
D   LstSize                     10I 0 Overlay(Receiver1:129)
D   LstEntNo                    10I 0 Overlay(Receiver1:133)
D   LstEntSize                  10I 0 Overlay(Receiver1:137)
  // Decode of Format OBJL0200
D Receiver2       DS                  Based(Ptr2)
D   ObjName                     10A
D   ObjLib                      10A
D   ObjType                     10A
D   InfoStatus                   1A
D   ExtObjAttr                  10A
D   Description                 50A
D   UserAttr                    10A
D                                7A
D
D RtvUsrSpcAddr   PR                  ExtPgm('QUSPTRUS')
D                               20A
D                                 *

 /Free
    // Retrieve control information from User Space Generic Header
    CallP RtvUsrSpcAddr(QualSpace:Ptr1);

    // Retrieve list entries from User Space
    Ptr2 = Ptr1 + LstOffset;
    For Count = 1 To LstEntNo;
      DspOut = ObjName + ' ' + ExtObjAttr;
      Dsply DspOut '*REQUESTER';
      Ptr2 = Ptr2 + LstEntSize;
    EndFor;

    *InLR = *ON;
    Return;
 /End-Free
```

Exercise 9 Using bindable CEE APIs

Step 3:

Part 10

CEEDYWK

Part 11

Lilian Date input 10I format; Day number output 10I format; Feeback (omissible)

Part 12

Need to call CEEDAYS to get the Lilian Day number

Sample Output:

```
                Birth Date?  YYYY-MM-DD: 1950-08-27
                Your Age is . . . . . .:     60
                10   Month(s) since your last Birthday

                You were born on a  Sunday


                There have been  3,854    days since the beginning of 2001




     Press Enter to continue
     F3=Exit
```

AgeINQX5 DSPF:

```
A                                      INDARA
A                                      CA03(03 'Exit')
 **
A          R HEADER
A                                 3 35'Age Calculator' COLOR(WHT)
A                                 3 64DATE  EDTCDE(Y)
 **
A          R PROMPT                    OVERLAY
A                                 8 20'Birth Date?  YYYY-MM-DD:'
A            BORN          10A  B  8 45
A  40                                  ERRMSG('No birth date entered - ner-
A                                      vous?' 40)
 **
A          R DETAIL                    OVERLAY
A                                 9 20'Your Age is . . . . . .:'
A            AGE            6  0O  9 45EDTCDE(L)
A                                10 25'Month(s) since your last Birthday'
A            MONTHS         2  0O 10 20EDTCDE(L)
A                                12 20'You were born on a'
A            DAYOFWK        9A  O 12 40
A                                15 20'There have been'
A            NODAYS         5Y 0O 15 36EDTCDE(1)
A                                15 46'days since the beginning of 2001'
A
 **
A          R FOOTER                    OVERLAY
A                                20  7'Press Enter to continue'
A                                21  7'F3=Exit'
A                                      COLOR(BLU)
```

AgeDemoX5 RPG IV:

```
FAgeInqX5 CF   E             Workstn IndDS(WKIND)
 // Prototypes for CEE APIs
D GetLilianDte    PR                  ExtProc('CEEDAYS')
D                                     Opdesc
D                                10A  Varying
D                                10A  Varying
D                                10I 0
D                                12A  Options(*Omit)

D GetDayOfWk      PR                  ExtProc('CEEDYWK')
D                                     Opdesc
D                                10I 0
D                                10I 0
D                                12A  Options(*Omit)

DNbrDays          PR             5  0
D BornVar         S              10A  Varying
D DateFmt         S              10A  INZ('YYYY-MM-DD')
D                                     Varying
D LDDayNo         S              10I 0
D DayBorn         S              10I 0
 // Day Array
D Days            S               9A  Dim(7) CtData PerRcd(4)
D WkInd           DS
D  Exit                    3      3N
D  BadDate                40     40N
D BornMonth       S               2  0
D CurrMonth       S               2  0



 /Free
  Write     Header;
  Write     Footer;
  Exfmt     Prompt;

  Dow NOT Exit;
  // Test for valid date value
      Test(DE) Born;
      If %error;
         BadDate = *On;
      Else;
  // Display details
         Age = %Diff(%date(*date):%date(Born):*y);
         Months = %Diff(%date(*date):%date(Born):*m) - Age*12;
  // Call NbrDays to determine # days since Jan. 1, 2000
         NoDays = NbrDays;

  // Determine day of week of birthdate
  BornVar = Born;
  CallP GetLilianDte (BornVar : DateFmt : LDDayNo : *Omit);
  CallP GetDayOfWk (LDDayNo: DayBorn : *Omit);
```

```
  DayOfWk = Days(DayBorn);
         Write Detail;
      EndIf;
  // Display prompt
      Exfmt Prompt;
  Enddo;

  *InLR = *On;
  Return;
 /End-free
 //************End of Main Procedure**************************
PNbrDays          B                   Export
DNbrDays          PI             5  0
DNbrDays          PR             5  0

 /FREE
  Return %Diff(%date(*date):D'2001-01-01':*D);
  *InLR = *on;
 /END-FREE
P                 E
**ctdata days
Sunday   Monday   Tuesday  Wednesday
Thursday Friday   Saturday
```

Exercise 10 Database triggers

OERFAXT solution:

```
H DatFmt(*mdy/)
FCUSTOMER  IF   E           K DISK
FOEPFAX    O    E             PRINTER

D TrigPlist       DS                  Based(TrigBPtr)
  //                                            Trigger Buffer
D FileName                  10A
  //                                            File Name
D LibName                   10A
  //                                            Library Name
D FileMember                10A
  //                                            Member Name
D TrigEvent                  1A
  //                                            Trigger Event
D TrigTime                   1A
  //                                            Trigger Time
D CommitLockLvl              1A
  //                                            Commit Lock Level
D Reserved1                  3A
  //                                            Reserved 1
D CCSID                     10I 0
  //                                            CCSID
D CurrentRRN                10I 0
  //                                            Current Rrn
D Reserved2                  4A
  //                                            Reserved 2
D OldRecOffset              10I 0
  //                                            Old Record Offset
D OldRecLen                 10I 0
  //                                            Old Record Len
D ORecNullOffset            10I 0
  //                                            Old Rec Null Map Offset
D ORecNullLength            10I 0
  //                                            Old Rec Null Map Length
D NewRecOffset              10I 0
  //                                            New Record Offset
D NewRecLen                 10I 0
  //                                            New Record Len
D NRecNullOffset            10I 0
  //                                            New Rec Null Map Offset
D NRecNullLength            10I 0
  //                                            New Rec Null Map Length
D TrigBuffRes               16A
  //                                            Reserved
  // Second trigger parameter (included for completeness; not used
D TBuffLen        S         10I 0
  // Define pointers to start of Trigger Buffer and to Record image
D TrigBPtr        S           *
D RecPtr          S           *
  // Prototype
D FaxTrig         PR              ExtPgm('OERFAXTS')
```

```
D                               1000A   Options(*Varsize)
D                                 10I 0 Const
  // Procedure Interface
D FaxTrig         PI
D   TrigBuff                    1000A   Options(*Varsize)
D   BuffLength                    10I 0 Const
  // Externally-described DS for record format
D OOrderRec     E DS                    ExtName(ORDERHDR)
D                                       Based(RecPtr)

 /Free
  // Get address of trigger buffer passed by DB Manager
  TrigBPtr = %Addr(TrigBuff);
  // Set Pointer to old record image in Trigger Buffer
  RecPtr   = TrigBPtr + OldRecOffset;
  // Get Customer Name field from Customer file - use CUSNBR field from
  // ORDERHDR file
  Chain CUSNBR Customer;
  // If record found (which it will be) write the FAX
  If %found(Customer);
    Write OEPFAX_FMT;
  EndIf;
  // End the program; trigger is invoked by DB manager once for each FAX
  *InLR = *On;
 /End-Free
```

Add trigger:

```
ADDPFTRG FILE(AS10nnn/ORDERHDR) TRGTIME(*AFTER) TRGEVENT(*DELETE)
PGM(AS10nnn/OERFAXT)
```

Display trigger information using DSPFD:

```
DSPFD FILE(AS10nnn/ORDERHDR) TYPE(*TRG)

   7/03/02               Display File Description
DSPFD Command Input
  File  . . . . . . . . . . . . . . . . . . . . : FILE       ORDERHDR
    Library . . . . . . . . . . . . . . . . . . :            AS10V6LIB
  Type of information . . . . . . . . . . . . . : TYPE       *TRG
  File attributes . . . . . . . . . . . . . . . : FILEATR    *ALL
  System  . . . . . . . . . . . . . . . . . . . : SYSTEM     *LCL
File Description Header
  File  . . . . . . . . . . . . . . . . . . . . : FILE       ORDERHDR
  Library . . . . . . . . . . . . . . . . . . . :            AS10V6LIB
  Type of file  . . . . . . . . . . . . . . . . :            Physical
  File type . . . . . . . . . . . . . . . . . . : FILETYPE   *DATA
  Auxiliary storage pool ID . . . . . . . . . . :            01
Trigger Description
  Trigger name  . . . . . . . . . . . . . . . . : TRG        QSYS_TRIG_AS10V6L
        IB__ORDERHDR___000001
    Trigger library . . . . . . . . . . . . . . :            AS10V6LIB
    Trigger state . . . . . . . . . . . . . . . : STATE      *ENABLED
    Trigger status  . . . . . . . . . . . . . . :            *OPERATIVE
    Trigger event . . . . . . . . . . . . . . . : TRGEVENT   *DELETE
    Trigger time  . . . . . . . . . . . . . . . : TRGTIME    *AFTER
    Allow repeated change . . . . . . . . . . . : ALWREPCHG  *NO
    Program Name  . . . . . . . . . . . . . . . : PGM        OERFAXTS
      Library . . . . . . . . . . . . . . . . . :            AS10V6LIB
    Program is threadsafe . . . . . . . . . . . : THDSAFE    *UNKNOWN
    Multithreaded job action  . . . . . . . . . : MLTTHDACN  *SYSVAL
    Trigger type  . . . . . . . . . . . . . . . :            *SYS
    Trigger orientation . . . . . . . . . . . . :            *ROW
    Trigger creation date and time  . . . . . . :            07/02/02  13:51:07
    Number of trigger update columns  . . . . . :                0
```

Exercise 11 Enhancing NOMAIN service program

Step 2 - Create the objects

```
CRTRPGMOD MODULE(AGEDEMOMN)

CRTRPGMOD MODULE(subprocSnm)

CRTSRVPGM SRVPGM(AS10nnn/AS10001) MODULE(AS10nnn/SUBPROCSNM)
EXPORT(*ALL) TEXT('AS10nnn Service Program - Ex 7')
CRTPGM PGM(AGEINQPGM) MODULE(AGEDEMOMN) BNDSRVPGM(AS10nnn/AS10001)
ACTGRP(*CALLER)
```

Part 9

```
The list of exports should be: NBRDAYS
```

Step 4 - Create more objects and test

```
CRTRPGMOD MODULE(ITEMINQ2)

CRTPGM PGM(ITEMINQ2) BNDSRVPGM(AS10nnn/AS10001) ACTGRP(*CALLER)
```

Step 5 - Add more subprocedures to NOMAIN procedure

Part 21

```
The list of exports should be: NBRDAYS,PAYMNT,RATPER
```

PROTOTYPES

```
 /If defined (NbrDays)
  // NbrDays
DNbrDays          PR             5  0
  // Prototype for loan payment
 /ElseIf defined (DLTPROC)
  // Prototype for DLTPROC
D VnrDelete       PR                  ExtProc('DLTPROC')
D  VndNbr                        5  0
 /ElseIf defined (DayofWeek)
  // Prototype for DayofWeek
D DayOfWeek       PR             1S 0
D                                 D
 /ElseIf defined (LoanPay)
  // Prototype for loan payment
DPaymnt           PR             9  2

 /ElseIf defined (DayofWeek)
  // Prototype for DayofWeek
D DayOfWeek       PR             1S 0
D                                 D
 /ElseIf defined (LoanPay)
  // Prototype for loan payment
DPaymnt           PR             9  2

 ** Calc loan payment PROTOTYPE

DPrincipal                       9  2
DRatePeriod                     13 11
DNbrPayTot                       4  0
  // Prototype for periodic rate
DRatPer           PR            13 11

** RATPER - Calc dec periodic interest rate PROTOTYPE

DPrincipal                       9  2
DRatePeriod                     13 11
DNbrPayTot                       4  0
  // Prototype for periodic rate
DRatPer           PR            13 11

** RATPER - Calc dec periodic interest rate PROTOTYPE

DRatePCAnn                       5  3
DNbrPayYr                        2  0
 /EndIf
```

AgeDemoMN

```
FAgeInq    CF   E             Workstn IndDS(WKIND)
D WkInd           DS
D  Exit                    3      3N
D  BadDate                40     40N
D BornMonth       S               2  0
D CurrMonth       S               2  0

 /Define NbrDays
 /Copy prototypes
 /Undefine NbrDays

 /Free
  Write     Header;
  Write     Footer;
  Exfmt     Prompt;

 /Copy prototypes
 /Undefine NbrDays

 /Free
  Write     Header;
  Write     Footer;
  Exfmt     Prompt;

  Dow NOT Exit;
  // Test for valid date value
      Test(DE) Born;
      If %error;
         BadDate = *On;
      Else;
  // Display details
         Age = %Diff(%date(*date):%date(Born):*y);
         Months = %Diff(%date(*date):%date(Born):*m) - Age*12;
  // Test for valid date value
      Test(DE) Born;
      If %error;
         BadDate = *On;
      Else;
  // Display details
         Age = %Diff(%date(*date):%date(Born):*y);
         Months = %Diff(%date(*date):%date(Born):*m) - Age*12;
  // Call NbrDays to determine # days since Jan. 1, 2000
         NoDays = NbrDays;
         Write Detail;
      EndIf;
  // Display prompt
      Exfmt Prompt;
  Enddo;

  *InLR = *On;
         NoDays = NbrDays;
         Write Detail;
      EndIf;
```

```
 // Display prompt
     Exfmt Prompt;
 Enddo;

 *InLR = *On;
 Return;
/End-free
```

ItemINQ2

```
 // Declare Files
FItem_PF   IF   E           K Disk
FItemInq2  CF   E             Workstn IndDS(WkstnInd)

 // Map indicators in DSPF to named indicators
D WkstnInd        DS
D  NotFound              40     40N
D  LowQty                30     30N
D  Exit                  03     03N

 /Define NbrDays
 /Copy prototypes
 /Undefine NbrDays

 /FREE
  PgmNam = 'ITEMINQOV';
D  Exit                  03     03N

 /Define NbrDays
 /Copy prototypes
 /Undefine NbrDays

 /FREE
  PgmNam = 'ITEMINQOV';
  // Call NbrDays to determine # days since Jan. 1, 2000
  NoDays = NbrDays;

  Write Header;  // Write to buffer
  Write Footer;  // Write to buffer
  Exfmt Prompt;  // Write to buffer; write buffer to display; enable re

  Dow NOT Exit;
    Chain ItmNbr Item_PF;
  NoDays = NbrDays;

  Write Header;  // Write to buffer
  Write Footer;  // Write to buffer
  Exfmt Prompt;  // Write to buffer; write buffer to display; enable re

  Dow NOT Exit;
    Chain ItmNbr Item_PF;
    NotFound = Not %found(Item_PF); // Set indicator for record not fou

    If %found(Item_PF); // Item Number valid?
      LowQty = (ItmQtyOH + ItmQtyOO) < 20; // Set Indicator for Qty < M
      Write Detail; // Write to buffer
    Endif;

    // Display prompt with error or not
    Exfmt Prompt; // Write to buffer; write buffer to display; enable r

    If %found(Item_PF); // Item Number valid?
      LowQty = (ItmQtyOH + ItmQtyOO) < 20; // Set Indicator for Qty < M
```

```
      Write Detail; // Write to buffer
    Endif;

    // Display prompt with error or not
    Exfmt Prompt; // Write to buffer; write buffer to display; enable r
  Enddo;
    // F3 pressed; user wants to exit program
  *InLR = *on;
 /END-FREE
```

LOANPAY

```
FLoanPayD  CF   E             WorkStn IndDS(LoanPDS)
D LoanPDS         DS
D  Exit                   3      3N
 /Define LoanPay
 /Copy prototypes
 /Undefine LoanPay
 /free
      ExFmt PayFmt;

      DoW NOT Exit;
          RatePeriod = Ratper(RatePCAnn:NbrPayYr);
          PaymentAmt = Paymnt(Principal:RatePeriod:NbrPayTot);

          ExFmt PayFmt;
      EndDo;


      *InLR = *On;
      Return;
 /end-free
```

Exercise 12 Using binding directories and binder language

Step 3

Part 6

```
CRTPGM PGM(AS10nnn/LOANBD) MODULE(AS10nnn/LOANPAYAPI) BNDDIR(AS10nnn/MYBNDDIR)
```

Part 7

```
CRTPGM PGM(AS10nnn/LOANBD) MODULE(AS10nnn/LOANPAY) BNDDIR(AS10nnn/MYBNDDIR)
OPTION(*DUPPROC)
```

Step 5

Part 10 - Export subprocedures, NBRDAYS, PAYMNT, and RATPER

NOMAINBD binder language

```
STRPGMEXP PGMLVL(*CURRENT)
          EXPORT SYMBOL(NBRDAYS)
          EXPORT SYMBOL(PAYMNT)
          EXPORT SYMBOL(RATPER)
ENDPGMEXP
```

Step 6

Part 13

```
CRTSRVPGM SRVPGM(NOMAINBD) MODULE(SUBPROCSNM) TEXT('Ex 8 - Use Binder Language')
```

Part 14

```
CRTPGM PGM(AGEBIND) MODULE(AGEDEMOMN) TEXT('Ex 8 -using binder language')
BNDSRVPGM(NOMAINBD)
```

Step 8

Part 16

```
CRTPGM PGM(LOANBIND) MODULE(LOANPAY) TEXT('Ex 8 -using binder language')
BNDSRVPGM(NOMAINBD)
```

Part 18

```
CRTPGM PGM(AS10nnn/LOANBIND) MODULE(*LIBL/LOANPAYAPI) BNDDIR(AS10nnn/MYBNDDIR)
```

Exercise 13 Enhancing the condition handler

Step 1:

```
CRTRPGMOD MODULE(CAUSEERR)
CRTRPGMOD MODULE(CONDHDLR)
CRTPGM PGM(ILECOND) MODULE(CAUSEERR CONDHDLR) ACTGRP(*CALLER)

CALL ILECOND
```

MsgQ:

```
DSPLY  Starting CauseErr
DSPLY  In CauseErr - causing Division error
DSPLY  Starting CondHdlr
DSPLY  Resume CauseErr
DSPLY  In CauseErr Subr - error detected
DSPLY  In CauseErr - causing Index error
DSPLY  Starting CondHdlr
DSPLY  Starting CondHdlr
DSPLY  Starting CondHdlr
DSPLY  Starting CondHdlr
DSPLY  In CauseErr Subr - error detected
```

Joblog:

```
Attempt made to divide by zero for fixed point operation. (MCH1211)
Range of subscript value or character string error.  (MCH0603)
Receiver value too small to hold result. (MCH1210)
Range of subscript value or character string error.
Receiver value too small to hold result.
```

Step 2:

CondHdlrE:

```
PCondHdlr         B                   Export
 //  CondHdlr Interface
D CondHdlr        PI
D  InToken                            Like(CondToken)
D  pErrInfo                      *
D  Action                      10I 0
D  OutToken                           Like(CondToken)

D Error           S               1N  Based(pErrInfo)

 // Action code values:
D Resume          C                   10
D Percolate       C                   20

 // Use Message to display progress to MSGQ
D Message         S              40A

 // Pointer InTokPoint for Condition Token DS
D InTokPoint      S                *

DCondToken        DS                  Based(InTokPoint)
D  CondSev                      5I 0
D  CondMsgNo                    2A
D                               1A
D  CondPrefix                   3A
D                               4A

 /Free
      InTokPoint = %Addr(Intoken);
                   Message = 'Starting CondHdlr';
                   Dsply Message '*REQUESTER';
      Select;
       When CondPrefix = 'MCH'
            and CondMsgNo = X'1211';
    // Set error flag on to indicate handled.
            Error = *on;
    // Set Action = Resume (10). Procedure with error will get
    // control at resume point.
            Action = Resume;

      Other;
            Action = Percolate;
      EndSl;

      Return;
 /End-Free
P                 E
```

Step 3:

```
CRTRPGMOD MODULE(CONDHDLRE)
CRTPGM PGM(ILECONDE) MODULE(CAUSEERR CONDHDLRE) ACTGRP(*CALLER)

CALL ILECONDE
```

MsgQ:

```
DSPLY  Starting CauseErr
DSPLY  In CauseErr - causing Division error
DSPLY  Starting CondHdlr
DSPLY  Resume CauseErr
DSPLY  In CauseErr Subr - error detected
DSPLY  In CauseErr - causing Index error
DSPLY  Starting CondHdlr
DSPLY  Starting CondHdlr
```

Joblog:

```
Attempt made to divide by zero for fixed point operation. (MCH1210)
Range of subscript value or character string error. (MCH0603)
An array index is out of range (C G D F). (RNQ0121)
C
An array index is out of range (C G D F).
C
Application error.  MCH0603 unmonitored by ILECONDE at statement
  0000000045, instruction X'0000'.
```