Net Express 4.0 exercises for *Murach's Mainframe COBOL*

The University Edition of Net Express 4.0 is an Integrated Development Environment (IDE) that you can use for developing COBOL programs on your own PC. To help you get the most from this IDE, this document provides exercises that are carefully coordinated with the first 16 chapters of *Murach's Mainframe COBOL* and especially designed for use with Net Express 4.0. Our goal is to help you learn more quickly and easily than you can with any other training package.

Before you start these exercises, you should read unit 1 of the *Net Express 4.0 Tutorial* that we've developed. And before you start the exercises for chapter 13, you should read unit 2. As you do the exercises, you'll be reminded of these reading recommendations.

Copyright ©2004 Mike Murach & Associates, Inc.
All rights reserved.
murachbooks@murach.com • www.murach.com

Before you start the exercises for this book

Before you start the exercises for Murach's Mainframe COBOL, you should do the following:

1. Install the University Edition of Net Express 4.0 on your PC. If you purchased this product separately, just follow the installation instructions that come with the CD. If you purchased this product as part of Net Express with .NET, follow the installation instructions for installing only Net Express 4.0. (You don't need to install Net Express with .NET.) Note that although the instructions tell you to insert CD-1 into your CD-ROM drive, Net Express is contained on a single CD in the front cover of the case. This is the only CD you will need.
2. Install the files we've provided to help you do the exercises.
    a. If you downloaded this Exercises document from our web site, the download includes all the other files you need, and they've all been installed on your system in folders that start with C:\Murach\Mainframe COBOL. In that case, you can skip to step 3.
    b. If you got this Exercises document as part of a class, your instructor may provide the additional files you need. Or, you can download the files from our web site. To do that, go to www.murach.com, and go to the page for *Murach's Mainframe COBOL*. Click the link for "FREE download of Net Express files." Then, download "All Net Express files." This will download one file named mcb2_netexpress.exe.

       Use the Windows Explorer to find the mcb2_netexpress.exe file. Then, double-click on this file and respond to the dialog boxes that follow. This installs the files in folders that start with C:\Murach\Mainframe COBOL.
3. Within the Mainframe COBOL folder, you'll find the Net Express tutorial in PDF format. Before you start the exercises, you should read unit 1 of this tutorial.
4. The download also includes starting source code for any exercises that require it, as well as the data files used by the exercises. During the installation process, these files are installed first in C:\Murach\Mainframe COBOL\Exercise starts\MCOBOL. Then, a batch file copies them to C:\MCOBOL. As you do the exercises, you'll work from the C:\MCOBOL folder. But if you make a mistake and want to restore a file to its original state, you can do that by copying the original from the Exercise starts\MCOBOL folder.
5. The MCOBOL folder contains subfolders with names like Chapter 01, Chapter 02, and Data. Each chapter folder contains the starting source code for the exercises for that chapter. These are also the folders in which you'll create your exercise solutions. The Data folder contains all of the data you'll need to run your exercise solutions.

Exercise 2-1 Get started with Net Express

This exercise helps you get started with Net Express by guiding you through some common operations.

Set the default folder

1. Start Net Express and click on the Continue button in the Welcome dialog box if it is displayed. Then, select the Customize IDE command in the Options menu.
2. Click on the Workspace tab in the dialog box that appears, enter c:\mcobol in the Default folder text box, and click on the OK button. We recommend that you use this folder as the top-level folder for all of the programs that you develop.

Start a new project from an existing source file

3. Use the File→New command to open the New dialog box, select Project, and click on the OK button. That opens the New Project dialog box.
4. If you did the first two steps in this exercise right, the folder in the New Project dialog box should be c:\mcobol. Then, select the Project from an existing application option, enter CALC2000 as the name of the project, add \chapter 02 to the end of the project folder (so it becomes c:\mcobol\chapter 2), and click on the Create button. That starts a Wizard that will step you through the process of adding an existing source file to the project.
5. Use the first step of the Wizard to add the file named CALC2000.CBL to the project. This file is in the c:\mcobol\chapter 02 folder. Then, complete the Wizard by clicking on the Next button twice and the Finish button. This opens a project window for the CALC2000 project that contains the CALC2000.CBL file and a CALC2000.INT file.

Experiment with the project and document windows

6. If necessary, click on the Maximize button for the project window to maximize it. Then, double-click on the CALC2000.CBL file. This opens the document window for that file. If necessary, maximize this window too so it replaces the project window.
7. To move back to the project window, click the project tab below the document window. Then, to move back to the document window, click its tab.
8. Press the Page Up or Page Down key to move through the program one screen at a time. Next, press the arrow keys to move one line or character at a time. Then, press the Ctrl key while you press the left or right arrow key to move from word to word in the program.
9. Note the colors that are used for the source code. At this point, COBOL keywords are in green, and all other code is in black. Then, click on the Rebuild button to compile the program, which should compile without error. Now, what color is used for variable names? What color is used for procedure names?
10. Move to the start of procedure 120. Next, hold down the Shift key as you press the right arrow key or End key to highlight the procedure name. Then, press the Delete key to delete the highlighted text. To restore that text, click on the Undo button in the toolbar or press Ctrl+Z.

11. Drag the mouse over two characters or lines so you can see that the mouse only lets you highlight complete lines. Next, drag the mouse over all the lines in procedure 120 including the procedure name. Then, press Ctrl+C to copy the lines, move the cursor to the start of procedure 110, and press Ctrl+V to paste the copied lines into the text. Last, undo the operation.

Use the Find and Replace command

12. Click on the Find Text button in the toolbar to display the Find and Replace window. Then, use the Fwd and Back buttons in this window to find each occurrence of FUTURE-VALUE. Next, use the All button to tag each occurrence, and use the Fwd and Back buttons to move from one tagged occurrence to another. Then, use the Toggle Compress button to compress and expand the source code. Last, click on the Compile Program button to remove the tags.
13. Use the Find Text command to find each occurrence of FUTURE-VALUE, but this time check the Whole word box. Then, click on the Show Toolbar View button and use the first two buttons in the Find toolbar to move from one occurrence to another. Last, click on the Show Dialog View button on the right of the Find toolbar to restore the Find and Replace window, and click on the Compile Program button to remove the tags.

Continue to experiment, then close the project

14. Review the commands in the File, Edit, Window, and Help menus, and use the Options→Edit command to review the edit options. Then, experiment on your own until you feel comfortable with the way that Net Express works.
15. Use the File→Close command with the document window active. Note that this closes the document window, but not the project window. Then, use the same command to close the project window, which closes the project too.

Exercise 2-2 Test the sales tax program

This exercise will guide you through the process of compiling and running the sales tax program that's presented in chapter 1.

Start a sales tax project and compile the program

1. Start a new project named CALC1000 in the c:\mcobol\chapter 02 folder. Then, add the source file named CALC1000 that's in the same folder to it. To do that, right-click in the left pane of the project window and select the Add files to project command.
2. Open the document window for the source file. If necessary, maximize it.
3. Click on the Rebuild button to compile the program. This should result in a clean compile.

Test the program

4. Click on the Run button to run the program. This displays the Start Animating dialog box, which shows that the version of your program that is being run is stored in a folder named DEBUG that has been created by Net Express. If you click on the check box, you can stop this dialog box from being displayed each time you run a program. But whether or not you do that, click on the OK button to continue. This opens the application output window, displays the results of the first Display statements, and waits for your response to the first Accept statement.
5. For the first test run, enter zero and press the Enter key. The program then displays the "END OF SESSION" message and terminates normally.
6. To run the program again, switch to the Net Express IDE and click on the Run button. For this test run, enter several values that will show you whether this program works correctly. When you're satisfied that it does, enter a zero to end the program.

Close the project

7. Switch to the project window by clicking on the project tab beneath the document window. Then, close the project window by clicking on its Close button. Note that this also closes the document window and the project.

Exercise 2-3 Correct compile-time and run-time errors

This exercise forces you to correct some compile-time errors and fix a bug in the sales tax program. This will give you an appreciation for what you have to do when you compile and test your own programs.

Correct the compile-time errors

1. Start a new project named CALC100X in the c:\mcobol\chapter 02 folder, and add the source file named CALC100X that's in the same folder to it. Then, open the document window for the source file and compile the program by clicking on the Build button. Oops! Several compile-time errors have been marked in the document window and listed in the output window.
2. Scroll to the first message in the output window: "Operand Y is not declared." Then, double-click on this message to jump to the error statement. If you want more information about the error, right-click on this statement and choose the Syntax Error Help command. This opens a Help window. When you're done reviewing it, close this window.
3. Can you correct these errors on your own? You should be able to without using the Help information for each error. At the least, try to correct these errors on your own before you read on.
4. The first error message says that the operand Y is not declared, which normally means that the variable isn't defined in the Data Division. In this case, though, Y is supposed to be a literal, so it should be coded as "Y".
5. The second message says that an Else phrase doesn't have a matching If phrase. The problem, though, is that there's a period at the end of the line before the Else. Since a period ends a statement, the compiler thinks the word Else is the first word in the next statement.
6. The third and fourth messages say that the variable named SALES-TAX isn't declared, and this time the messages are right. If you look in the Data Division, you can see that a variable named TAX-AMOUNT is defined, but there's no variable named SALES-TAX.
7. Correct these errors and recompile the program. This time, there shouldn't be any errors. But if you did something wrong, fix it and recompile until you get a clean compile.

Test the program and close the project

8. Run the program. Oops again! The program reaches the Stop Run statement and ends, but never accepts user entries and calculates the sales tax.
9. To fix this problem, think about what must be happening. Somehow the program thinks that the condition in the first Perform statement is true when the program starts. When you figure out what the problem is, fix it, recompile the program, and rerun it. This time, the program should work correctly.
10. Close the project.

Exercise 2-4 Test the future value program

This exercise will guide you through the process of compiling and testing the future value program that's presented in chapter 1. It will also show you how to use breakpoints, display the values of variables, and step through a program.

Compile and test the program

1. In exercise 2-1, you started a new project named CALC2000 in c:\mcobol\chapter 02. Use the File→Open command to open that project now, and open the document window for the source program.
2. Run the program. When the application output window is displayed, enter a value of zero for the first test run. This should end the program.
3. Run the program again. This time, enter 100, 1, and 10 as the values for the investment amount, number of years, and interest rate. Does the program display the correct future value (110.00)?
4. Continue testing the program by entering values that test the minimum and maximum values. Note that if you enter large values, you'll start to get inconsistent results. That's because the future-value field isn't large enough to hold the calculated results, so Net Express truncates them. When you're through experimenting, end the program.

Set a breakpoint and step through the program

5. Set a breakpoint on the Perform Until statement in procedure 100. Then, run the program. When the Accept statements are executed, enter a 1 to perform another calculation, and enter 100, 3, and 10 for investment amount, number of years, and interest rate.
6. When the breakpoint is reached, hold the mouse pointer over the year-counter variable name anywhere in the source code to see the data tip that's displayed for it. Next, double-click on that variable name and click on the Add to list button in the resulting dialog box to add that variable to the list at the bottom of the IDE. Repeat this for the number-of-years and future-value variables.
7. From the breakpoint, step through the program to see how the variable values change after each statement is executed and to see the sequence in which the statements are executed. When you're past the portion of the code that does the calculation, click on the Run button to continue without stopping. Then, enter 0 to end the program.
8. At this point, Net Express is still animating the program so it displays a dialog box that says the Stop Run statement has been encountered. To run the program again, use the Animate→Stop Animating command and click on the Run button. Note that the variable values are still displayed when the program enters break mode. Then, when you're through experimenting, remove the breakpoint and close the project.

Exercise 2-5 Start a new project for a new program

This exercise will guide you through the process of starting a new project for a new program. If, for example, you need to develop a program that's similar to the CALC2000 program, it makes sense to start the new program from a copy of the CALC2000 program.

Start the new project and add a copy of an old program to it

1. Use the New command in the File menu to start a new project. In the New Project dialog box, name the project PROJ1000, and specify c:\mcobol\projects as the folder for the project. Then, click on the Create button. When Net Express asks whether you want to create the folder, click the OK button. This opens the project window for the new project.
2. Right click in the project window and use the Add files command in the shortcut menu to display the Add files dialog box. Then, do the steps that follow to add a copy of the CALC1000.CBL file that's in c:\mcobol\chapter 02 to the project that you've just created. First, move to the c:\mcobol\chapter 02 folder, right-click on the CALC1000.CBL file, and select the Copy command. Second, move to the c:\mcobol\projects folder, right-click in the file area, and select the Paste command. Third, rename the file that you've just pasted to PROJ1000.CBL, select that file, and click on the Add button.
3. At this point, you can delete the lines of code that you don't need, modify the lines that need to be modified, and add the lines that need to be added. This is the fastest way to develop new COBOL programs.
4. Close the project.

Another way to start a new project with a copy of an old program

5. Open your Windows Explorer and navigate to the c:\mcobol\chapter 02 folder. Then, copy and rename the CALC2000.CBL file as follows: First, right-click on the file name and select the Copy command. Second, navigate to the c:\mcobol\projects folder, right-click in the file area, and select the Paste command. Third, rename the file that you've just copied to PROJ2000.CBL.
6. Go back to Net Express, start a new project named PROJ2000 in the c:\mcobol\projects folder, and add the file named PROJ2000 to it. At that point, you're ready to develop the new program from the old one.
7. Close the project.

Use the method that you like best whenever you create new programs

8. Pick the method you like best and use it whenever you create a new program or whenever one of the exercises asks you to do something like this:

   "Start a new project in c:\mcobol\chapter 03. Then, add a copy of the RPT1000 program that's in the same folder to the project after you rename the program RPT2000."
9. Go back to the Windows Explorer, and delete the projects folder.

Exercise 3-1 Test the sales report program

This exercise will guide you through the process of testing the sales report program in chapter 3 of *Murach's Mainframe COBOL*. It will also show you one way to review the input data and the report that's prepared from it. Later, when you read unit 2 of the Net Express 4.0 tutorial, you'll learn how you can use that IDE to review the data in files.

Compile and test the program

1. Start a new project named RPT1000 in c:\mcobol\chapter 03, and add the program named RPT1000 that's in the same folder to it.
2. Look at the system names in the Select statements for the customer master file and for the print file. There, you'll see that the system names for Net Express 4.0 are just the paths and a file names that identify the files with the convention that all data files have an extension of DAT and all print files have an extension of PRN.
3. Compile the program (it should have a clean compile). Then, run the compiled program (it should run to a normal termination).

Review the sales report and the test data

4. Start your word processor and open the print file named salesrpt.prn that's in the c:\mcobol\data folder. If the report looks like the one in figure 2-5 of the Net Express tutorial, it means the program worked correctly. If you want to print the file, delete the page break at the start of the file, adjust the font size and page setup (if necessary), and print. Then, close the file.
5. If necessary, you can also use your word processor or Notepad to review the test data for the program. To do that, open the file named custmast.dat in the c:\mcobol\data folder. Here, you can see that nothing separates one field from another or one record from another. The entire file is one long string of characters. When you're done reviewing the data, close the file.
6. To test whether page overflow works, change the LINES-ON-PAGE field in your program to a value of 10, recompile, and rerun. Then, look at the print file in your word processor to see whether the report prints with 10 lines per page. If it does, close the print file and reset the LINES-ON-PAGE value to 55.

Step through the program to see how it works

7. If you have any doubts about how this program works, use the skills you learned in unit 1 of the Net Express tutorial to step through the program from the start. Or use a breakpoint to step through any parts of the program that you don't understand. As you step, display the contents of the fields that are being operated upon.
8. When you're satisfied that you understand exactly how this program works, close the project.

Exercise 3-2 Compile and test a flawed program

To give you some practice with compiling and testing a report-preparation program, this exercise forces you to correct some compile-time errors and debug any run-time errors.

Correct the compile-time errors

1. Start a new project named RPT1000X in c:\mcobol\chapter 03, and add the program named RPT1000X that's in the same folder to it. Then, compile the program. Since we put a few errors in it, it doesn't compile cleanly.
2. Correct the errors and recompile until you get a clean compile. You should be able to do that without much trouble.

Test the program

3. Run the compiled program. It should end with a run-time error that indicates that the file can't be found. Then, when you return to the document window in break mode, the Open statement is highlighted. This indicates that there's either a problem with the Select statement or the file isn't where it's supposed to be.
4. Fix the problem, recompile, and rerun the program. This time, a run-time error message indicates that there's an illegal character in a numeric field in one of the Move statements in procedure 220. The clue here is that the statement moves a field from the customer master record to a numeric edited field. This indicates that there's either a problem with the data or with the FD statement or record description that defines the data, but you already know that the data is okay.
5. To fix the problem, you need to make sure that the FD and record description match up with the input specifications. When you find the problem, correct it, recompile, and rerun the program. This time the program should run to a normal termination.
6. Use your word processor to check the report output. At a glance, you can see that the report is double-spaced when it should be single-spaced. Don't stop there, though; do a thorough review of the report to make sure it works correctly. Check that the data values in each column are what you expected. Then, fix the problems, recompile, and retest until you've got everything working right.

Close the program

7. Close the project. Now, you've got two programs that work the same: RPT1000 and RPT1000X.

Exercise 3-3 Enhance the sales report program

To show you how easy it is to enhance a well-written program, this exercise guides you through the process of enhancing the sales report program.

Enhance the original program

1. Start a new project named RPT2000 in c:\mcobol\chapter 03. Then, add a copy of the RPT1000 program that's in the same folder to this project after you rename the program RPT2000.
2. Change the Program-ID in the Identification Division to RPT2000.
3. Make the enhancements to the program that are specified in figure 3-10 of the book. To learn the most from this exercise, try to make these enhancements without referring to the code in figures 3-11 and 3-12.
4. Compile and test the enhanced program until it works correctly.

Modify the enhanced program

5. Instead of printing a customer line on the report only when this year's YTD sales are greater than or equal to $10,000, modify the report so it prints a customer line when the change amount is greater than or equal to $5,000. Then, compile and test the program until that works right.
6. If that was easy enough to do, modify the report again so it prints a customer line only when the change percent is positive and greater than or equal to 25.0%. Then, compile and test the program until it works right.
7. Now, change the program back to the way it was in step 3 so it works as shown in figure 3-10.

Close the project

8. Close the project.

Exercise 4-1 Restructure the detail report program

This exercise guides you through the process of restructuring the enhanced version of the sales report program shown in figure 3-10 of the book. In particular, you need to add two Write modules to that program.

Design the detail report-preparation program of chapter 3

1. Start a new project named RPT2000 in c:\mcobol\chapter 04, and add a copy of the RPT2000 program that's in c:\mcobol\chapter 03 to it. This is the program that you created in exercise 3-3. Open the program, and notice that it consists of seven procedures (modules).
2. Create a structure chart for the RPT2000 program using its code as a guide.
3. Compare the structure chart you just created with the structure chart in figure 4-7. Then, modify the structure chart in figure 4-7 so it will create the RPT2000 report. To do that, you'll see that you just need to delete one module from the structure chart.

Modify the code for this program

4. Modify the code for RPT2000 so it conforms to the structure chart that you created in step 3. Be sure to rename or renumber procedures wherever appropriate. The code for the Write modules should be the same as it is in figure 4-16, which means that you need to add procedures 340 and 350 to the code and make some changes to procedure 320. For example, since module 350 counts the number of lines that are printed, you need to remove the line of code that counts customer lines from procedure 320.
5. Compile and test RPT2000 to make sure that it gives the same results as the original program.

Close the project

6. Close the project.

Exercise 4-2 Develop the summary report program

This exercise guides you through the development of the summary report program presented in figures 4-15 and 4-16 of the text. You'll start this program from the code for the report program that you created in exercise 4-1, and you'll create the new program in three phases. That should give you a better appreciation for top-down coding and testing.

Add the branch number to the heading and customer lines

1. Start a new project named RPT3000 in c:\mcobol\chapter 04. Then, add a copy of the RPT2000 program that's in the same folder to this project after you rename the program RPT3000.
2. Change the Program-ID in the Identification Division to RPT3000.
3. Modify the code in the Data Division and Procedure Division so the heading lines and customer line provide for the branch number. For this phase of testing, the program should print the branch number in every customer line. Also, be sure to change the report number in the second heading line to RPT3000.
4. Test and debug.

Print the branch summary lines

5. Add the Data Division and Procedure Division code for printing the branch summary lines. For this phase of testing, the program can still print the branch number in every customer line, so you don't have to add the code in module 320 that checks whether the branch number has changed. Also, modify the grand total line so the words GRAND TOTAL appear at the beginning of the line and two asterisks appear at the end of the line, as shown in figure 4-2.
6. Test and debug.

Print the branch number in just the first line of each group

7. Add the code to module 320 that checks for a new branch number and prints the branch number in just the first customer line of each group, not in every customer line.
8. Test and debug.

Close the project

9. Close the project.

Exercise 4-3 Modify the summary report program

In this exercise, you'll modify the summary report program that you created in exercise 4-2 so it prints only the branch summary lines, not the customer detail lines. The resulting report should look like this:

```
DATE:  06/24/2004           YEAR-TO-DATE SALES REPORT               PAGE:    1
TIME:  16:44                                                        RPT4000

BRANCH      SALES         SALES         CHANGE      CHANGE
 NUM       THIS YTD      LAST YTD       AMOUNT     PERCENT

  12      13,580.23     23,333.33      9,753.10-     41.8- *
  22      44,656.11      1,000.00     43,656.11     999.9  *
  34      25,921.47     64,812.88     38,891.41-     60.0- *
  47      36,367.91     32,009.24      4,358.67      13.6  *

TOTAL    120,525.72    121,155.45        629.73-      0.5- **
```

Modify the structure chart for the program

1. Review the structure chart in figure 4-7. Then, modify this structure chart so it's a proper design for a program that prepares the report shown above.

Code and test the changes

2. Start a new project named RPT4000 in c:\mcobol\chapter 04, and add a copy of the RPT3000 program that's in the same folder to it after you rename the program RPT4000. Then, change the Program-ID to RPT4000.
3. Code and test the required changes using the modified structure chart as a guide. Be sure to change the report number in the second heading line to RPT4000.
4. When your program works the way it's supposed to, close the project.

Exercise 4-4 Generate a structure listing

In this exercise, you'll use our LISTMODS program to generate a structure listing from the program that you developed in exercise 4-2.

Run LISTMODS

1. Start a new project named LISTMODS in c:\mcobol\chapter 04, and add the LISTMODS program that's in the same folder to it.
2. Modify the system name in the first Select statement in the Environment Division so it points to the program that you want to generate a structure listing for. If you don't change the system name, the program will generate a structure listing for the program you developed for exercise 4-2.
3. Compile and run the program. Then, review the printed output. The first page is the structure listing; the second page presents statistics that were derived from the program.

Close the project

4. Close the project.

Exercise 5-1 Improve the future value program

In this exercise, you'll modify the future value program of chapter 1 so it uses an inline Perform statement.

1. Start a new project named CALC2000 in c:\mcobol\chapter 05, and add the CALC2000 program that's in c:\mcobol\chapter 02 to it.
2. Modify the program so it uses an inline Perform Varying statement to do the future value calculation (see figure 5-11). Then, test the program to make sure that it still works correctly.
3. Close the project.

Exercise 5-2 Improve the summary report program

In this exercise, you'll improve the summary report program that you developed in exercise 4-2 by using some of the language that you learned in this chapter.

1. Start a new project named RPT3000 in c:\mcobol\chapter 05, and add the RPT3000 program that's in c:\mcobol\chapter 04 to it.
2. Define condition names for the first-record and end-of-file switches in the Data Division. Then, use the condition names in the Procedure Division to refer to the conditions, and use a Set to True statement to turn the end-of-file condition on. After you've made these changes, test them.
3. Rewrite the nested If statements in procedures 300 and 320 so they use Evaluate statements. Then, test the program to make sure that it still works correctly.
4. Close the project.

Exercise 5-3 Prepare a two-level summary report

In this exercise, you'll enhance the summary report program that you modified in exercise 5-2 so it prepares the two-level summary report shown in figure 5-18.

1. Start a new project named RPT5000 in c:\mcobol\chapter 05, and add a copy of the RPT3000 program that's in the same folder to it after you rename the program RPT5000. Then, change the Program-ID to RPT5000.
2. Enhance the code in the Data Division so it provides for the two-level report shown in figure 5-18.
3. Modify procedures 300, 320, and 360 in the Procedure Division so they provide for the two-level report, and add procedure 355, which will print the salesrep lines. Then, test the program to make sure that it works correctly.
4. Close the project.

Exercise 6-1 Modify the future value program

In this exercise, you'll apply some of the skills that you learned in this chapter as you modify the future value program that you prepared for chapter 5.

Open the project

1. Start a new project named CALC2000 in c:\mcobol\chapter 06, and add the CALC2000 program that's in c:\mcobol\chapter 05 to it.

Modify the program

2. Apply Packed-Decimal usage to the user-entries group and to the future-value and year-counter fields. Then, check to make sure that the Pictures for all of these fields have signs and an odd number of digits, which will improve both compile-time and run-time efficiency. If some don't, modify their pictures.
3. Change the Picture for the edited-future-value field so that it will be displayed with a floating dollar sign. Be sure to include one more dollar sign than there are digits to the left of the decimal point.
4. Redefine the edited-future-value field as a 13-byte alphanumeric field. Then, modify the code in procedure 100 so it checks if the future value that's calculated is greater than zero. If it's not, "Invalid" should be displayed instead of the future value amount.
5. Add a group named display-fields, and define a field in that group with a value that consists of 40 hyphens. Use the All figurative constant to specify this value. Then, modify procedure 100 so it displays this field instead of the literals that contain hyphens.

Compile and test the program

6. Compile and test the program until you're sure that it works correctly. (To test that it works correctly if the future value is less than or equal to zero, enter 0 or a negative value for the investment amount.)
7. Close the project.

Exercise 7-1 Calculate monthly payments

In this exercise, you'll develop a new program that gives you a chance to use one of the functions that is described in chapter 7.

1. Start a new project named CALC3000 in c:\mcobol\chapter 07, and add a copy of the CALC2000 program that's in c:\mcobol\chapter 06 to it after you rename the program CALC3000.
2. Modify the program so it calculates the monthly payment after the user enters the loan amount, interest rate, and number of months.

Exercise 8-1 Fix the Y2K problem in an elapsed days routine

In this exercise, you'll fix a Y2K problem in a routine that calculates the number of elapsed days between two Julian dates that have two-digit years.

1. Start a new project named DATE1100 in c:\mcobol\chapter 08, and add the DATE1100 program that's in the same folder to it.
2. Compile and run the program. For your first test, enter two Julian dates with years in the 90s (like 98100 and 99100). That should work. Then, enter one date with year 99 and another date with year 00 (like 99100 and 00100). That won't work correctly because there's a Y2K problem.
3. Modify the program so it gets Julian dates with four-digit years from the user. Then, make any other modifications that are necessary to fix the Y2K problem.
4. Test the program using the same data that you used in step 2. When you're sure that your modifications work correctly, close the project.

Exercise 8-2 Fix the Y2K problem in an elapsed days routine by using functions

In this exercise, you'll modify the program that you fixed in exercise 8-1 so it uses functions to calculate the number of elapsed days. As you will see, that's a better way to code this routine.

1. Start a new project named DATE1200 in c:\mcobol\chapter 08, and add a copy of the DATE1100 program that's in the same folder to it after you rename the program DATE1200. Then, change the Program-ID in the source code to DATE1200.
2. Replace the routine that calculates the number of elapsed days with one statement that uses the intrinsic functions for working with dates. Then, delete any unnecessary fields in the Data Division.
3. Test the program using the same data that you used in exercise 8-1. When you're sure that the program works correctly, close the project.

Exercise 8-3 Calculate age from a birth date with two-digit years

In this exercise, you'll write a routine that calculates the user's age after the user enters a month, day, and two-digit year of birth. To save time, you'll start from a program that gets the user entries.

1. Start a new project named DATE2100 in c:\mcobol\chapter 08, and add the DATE2100 program that's in the same folder to it.
2. Compile and run the program. As you can see, this program uses the Accept Date statement to get the current date with a four-digit year. This program also gets the birth month, day, and two-digit year from the user. However, it doesn't calculate the user's age.
3. Write the routine that calculates the user's age using the two-digit year for birth date. To do that, you need to use a windowing technique.
4. Compile and test the program. As you test, be sure to try month entries that are equal to the current month along with day entries that are before, equal to, and after the current day. Also, test to see whether your routine works for this birth date: April 3, 1929.
5. When you're satisfied that the program works correctly, close the project.

Exercise 8-4 Calculate age from a birth date with four-digit years

In this exercise, you'll modify the routine that you developed for exercise 8-3 so it uses four-digit years.

1. Start a new project named DATE2200 in c:\mcobol\chapter 08, and add a copy of the DATE2100 program that's in the same folder to it after you rename the program DATE2200. Then, change the Program-ID in the source code to DATE2200.
2. Modify the program so it gets the user entries with a four-digit year. Then, modify the routine that calculates the user's age so it uses the four-digit year for birth date.
3. Compile and test the program. As you test, be sure to try month entries that are equal to the current month along with day entries that are before, equal to, and after the current day.
4. When you're satisfied that the program works correctly, close the project.

Exercise 9-1 Edit numeric entries

In this exercise, you'll run a program that uses an editing routine that requires Inspect statements. Then, you'll modify the routine so it uses the Numval function to get the same result.

1. Start a new project named CHAR1500 in c:\mcobol\chapter 09, and add the CHAR1500 program that's in the same folder to it.
2. Compile and run the program. If you don't understand how it works, step through it and display the critical variables. Then, modify this program so it uses the Numval function to get the same result.
3. Change the picture of the user-entry field to X(10) and the picture of the edited-result field to S9(5)V99. Then, recompile and test again. This time, experiment with a variety of entries to see how well the Numval function works. When you're satisfied that you know what it can do, close the project.

Exercise 9-2 Unstring and string names

This exercise gives you a chance to use the Unstring and String statements. To start, you'll review a program that converts a name to first cap format. Then, you'll modify this program so it unstrings a full name in one format and strings it back together in a new format. The interactive session for this program should look something like this (the user entries are shaded):

```
----------------------------------------------
To end program, enter Y (or y).
n
----------------------------------------------
Enter last name, first name.
Example: Prince, Anne Marie
Murphy, Michael Ray
Result: Michael Ray Murphy
----------------------------------------------
To end program, enter Y (or y).
y
End of session.
```

Review the existing program

1. Start a new project named CHAR2500 in c:\mcobol\chapter 09, and add the CHAR2500 program that's in the same folder to it. This program converts a name from all uppercase letters to first cap format (see the second routine in figure 9-9).
2. Compile and run the program to see how it works. If you don't understand all of its code, step through it and display the critical variables.

Modify the program so it unstrings and strings names

3. Modify this program so it unstrings and strings names using an interactive session like the one shown above. Note that the last name that's entered by the user must be followed by a comma and a space for this to work. To start, assume that the last name will consist of just one word, but make this program work even if the user enters two or more names after the comma.
4. Modify the program so it works for last names that consist of two or more words like Van Erden. When you've got the program working the way you want it to, close the project.

Exercise 9-3 Edit phone numbers

This exercise gives you a chance to edit phone numbers that are entered with a variety of formats. To start, you'll review a program that edits a zip code field. Then, you'll modify this program so it edits a phone number. The interactive session for this program should look something like this (the user entries are shaded):

```
--------------------------------------------
To end program, enter Y (or y).
n
--------------------------------------------
Enter phone number.
Acceptable formats: (999) 999-9999
                     999 999 9999
                     999-999-9999
 555 123 4567
Edited phone number = 5551234567
Formatted phone number = (555) 123-4567
--------------------------------------------
To end program, enter Y (or y).
y
End of session.
```

Review the existing program

1. Start a new project named CHAR3500 in c:\mcobol\chapter 09, and add the CHAR3500 program that's in the same folder to it. This program edits a zip code (see the first routine in figure 9-9).
2. Compile and run the program to see how it works. If you don't understand all of its code, step through it and display the critical variables so you can see what's happening at each step.

Modify the program so it edits phone numbers

3. Modify this program so it edits the phone numbers that are entered by the user. These numbers can be entered with any of the formats shown above, but the edited phone number should have a picture of X(10) and this field should contain just ten digits when the editing is done. After the program edits each phone number, it should display it in its edited form as well as in the formatted style shown above. *Note that this program should work even if the user starts an entry with one or more spaces.*
4. Compile and test the program. When you've got the program working the way you want it to, close the project.

Exercise 10-1 Use a branch table to add branch names to a sales report

In this exercise, you'll enhance the summary report program that you created in chapter 5 by adding branch names to it so it looks something like this:

```
BRCH                     CUST                              SALES         SALES
NUM  BRANCH NAME          NUM   CUSTOMER NAME            THIS YTD      LAST YTD

 12  FORT WAYNE          11111  INFORMATION BUILDERS      1,234.56      1,111.11
                         12345  CAREER TRAINING CTR      12,345.67     22,222.22
                                        BRANCH TOTAL     13,580.23     23,333.33
 .
 .
 47  KANSAS CITY NW      12121  GENERAL SERVICES CO.     11,444.00     11,059.56
                         24680  INFO MANAGEMENT CO.      17,481.45     11,892.47
                         99999  DOLLAR SAVINGS BANK       5,059.00      4,621.95
                         76543  NATL MUSIC CORP.          2,383.46      4,435.26
                                        BRANCH TOTAL     36,367.91     32,009.24

                                         GRAND TOTAL    120,525.72    121,155.45
```

Here, the first two heading lines and the last two columns of the report are deleted so the report will fit on this page, but your report should include them.

The branch names are stored in a file named brchmast that's in sequence by branch number with this format:

```
    01  BRANCH-MASTER-RECORD.
        05  BM-BRANCH-NUMBER    PIC 9(2).
        05  BM-BRANCH-NAME      PIC X(18).
```

This program should load the records in this file into a table at the beginning of the program. Then, it should use search logic to get the branch name for a given branch number whenever necessary. If a branch number isn't found, the program should print "NO BRANCH RECORD" in the branch name column.

Enhance the structure chart for the program

1. Enhance the structure chart for the report program that's shown in figure 4-7 of the text so it includes modules for loading and searching the branch table.

Enhance the program

2. Start a new project named RPT8000 in c:\mcobol\chapter 10, and add a copy of the RPT3000 program that's in c:\mcobol\chapter 05 to it after you rename the program RPT8000. Change the Program-ID to RPT8000.
3. Modify the program so it loads the records in the branch master file into a branch table at the beginning of the program. The table should be defined so it uses subscripts and holds just the seven records in the branch master file.
4. Modify the remaining code so it prepares the report shown above. To make this manageable, you can do this in two parts. First, modify and test the program so it prints the branch name in each customer line. Then, modify and test the program so it prints the branch name in just the first line of each customer group.
5. When you've got the program working correctly, close the project.

Exercise 10-2 Modify the branch table code so it uses indexes and a sequential search

In this exercise, you'll modify the sales report program you created in exercise 10-1 so the branch table uses indexes instead of subscripts and so it uses the Search statement.

1. Start a new project named RPT8100 in c:\mcobol\chapter 10, and add a copy of the RPT8000 program that's in the same folder to it after you rename the program RPT8100. Then, change the Program-ID to RPT8100.
2. Modify the definition of the branch table so it includes an index. Next, modify the Procedure Division code so it uses the index instead of subscripts. Then, compile and test the program until it works correctly.
3. Change the code that searches the table so it uses a Search statement instead of a Perform Varying statement. Next, compile and test the program until this works correctly. Then, close the project.

Exercise 10-3 Change the sequential search to a binary search

In this exercise, you'll modify the sales report program you created in exercise 10-2 so it uses the Search All statement to search the branch table.

1. Start a new project named RPT8200 in c:\mcobol\chapter 10, and add a copy of the RPT8100 program that's in the same folder to it after you rename the program RPT8200. Then, change the Program-ID to RPT8200.
2. Modify the definition of the branch table to indicate that its entries are in ascending sequence by the branch-number field. Then, modify the Procedure Division code for searching the table so it uses the Search All statement.
3. Compile and test the program. When you're sure it works correctly, close the project.

Exercise 10-4 Change the branch table so it's variable-length

In this exercise, you'll modify the sales report program you created in exercise 10-3 so the branch table is defined as a variable-length table.

1. Start a new project named RPT8300 in c:\mcobol\chapter 10, and add a copy of the RPT8200 program that's in the same folder after you rename the program RPT8300. Then, change the Program-ID to RPT8300.
2. Modify the definition of the branch table so it can hold from 1 to 20 entries. Then, modify the code that loads the table so it saves the actual number of entries in the count field that the table definition refers to.
3. Compile and test the program. When you're sure it works correctly, close the project.

Exercise 10-5 Create a program that uses a two-level table

In this exercise, you'll create a program that loads, uses, and prints the two-level rate table that's presented in figure 10-6 in the text.

Load the table

1. Start a new project named TBL1000 in c:\mcobol\chapter 10, and add the TBL1000 program that's in the same folder to it. To save you time, this program contains the beginning code for your program. This includes most of the data definitions you'll need. You may want to print this code now so you can refer to it as you code the required procedures.
2. Write the code for procedure 100 and its subordinates so it loads the rate table from the rate table file, which contains one record for each age group. Figure 10-7 presents code that you can use as a guide, but note that you need to load both the low age and the high age for each age group. Note too the class group in the record description for this file is not defined with an index, but the age group and the class group in the rate table are defined with indexes. This means you have to use the Set statement to convert the subscript to an index somewhere in your routine.
3. Compile and test the loading routine. To make sure it works, you can add Display statements to the code that display each table record after it has been read and the entire table after it has been loaded. When you're sure that the routine works, remove the Display statements.

Search the table

4. Write the code for procedure 200 and its subordinates so it displays the proper rate after the user enters an age and class number. Note that the code for getting the age and class number from the user is already there. But you need to add the Search statements and logic that get the proper rate. For this coding, you can use the code in figure 10-14 as a guide, but your routine should search for an age that is greater than or equal to the low age and less than or equal to the high age. If the age or class number can't be found in the table, your routine should print an appropriate error message.
5. Compile and test the search routine until you're sure it works correctly.

Print the table

6. Write the code for procedure 300 and its subordinates so it prints just the data (no headings) for the table that you've loaded. To give you a start on this, the data definitions for the print line are already in working storage so the first printed line should look like this:

```
18-34    23.50    27.05    35.25    52.90
```

7. Compile and test the printing routine so you're sure it works correctly.

Close the project

8. When you've got everything working right, close the project. You should now have a solid appreciation for how multi-level tables are used.

Exercise 11-1 Create and use a copy member

In this exercise, you'll create and use a copy member for a customer master record.

1. Start a new project named RPT1100 in c:\mcobol\chapter 11, and add a copy of the RPT1000 program that's in c:\mcobol\chapter 03 to it after you rename the program RPT1100. Change its Program-ID to RPT1100.
2. Find the record description for the file named CUSTMAST, highlight it, and use the Copy command (Ctrl+C) to copy it to the clipboard. Next, open Notepad, and use the Paste command (Ctrl+V) to paste the contents of the clipboard into a new document. Then, save this file as CUSTMAST.CPY in the c:\mcobol\copy folder. You have just created a copy member.
3. Switch to the RPT1100 program, and delete the statements that you just copied to the copy member. In their place, code a Copy statement that refers to the copy member that you created in step 2.
4. Compile and run the program. It should work the same way that it did before.
5. When you're satisfied that the program works correctly, close the project.

Exercise 11-2 Create and use a subprogram

In this exercise, you'll create and use a subprogram that calculates the future value of an investment amount.

1. Start a new project named CALC2100 in c:\mcobol\chapter 11, and add a copy of the CALC2000 program that's in the same folder to it after you rename the program CALCFV. Change its Program-ID to CALCFV.
2. Modify this program so it becomes a subprogram that gets four fields passed to it: investment amount, number-of-years, yearly-interest-rate, and future value. The first three fields contain data that's entered by the user. The last field should receive the result of the calculation.
3. Compile the subprogram. If it doesn't compile cleanly, make the necessary changes until you get a clean compile. You have just created a subprogram.
4. Add another copy of the program named CALC2000 to the project after you rename the program CALC2100. Then, change its Program-ID to CALC2100.
5. Delete all the statements in the Procedure Division of CALC2100 that are used to calculate future value, and replace them with a Call statement that calls the CALCFV subprogram that you created in step 3. Then, delete all the statements in the Data Division that are no longer needed. Last, compile this calling program.
6. Use the Animate→Settings command to make sure that the calling program, not the subprogram, is targeted in the Start Animating at text box. If it isn't, change the targeted name from CALCFV to CALC2100. Then, close the dialog box.
7. Test and debug the calling program and the subprogram. If something isn't working right, set a breakpoint and step through the statements that follow, including the statements in the subprogram. This should allow you to pinpoint the problem.
8. When you've got the calling program and subprogram working correctly, close the project.

Exercise 11-3 Write a subprogram that edits a date field

In this exercise, you'll write a subprogram named EDITDATE that checks an eight-digit date field (YYYYMMDD) for validity. To be valid, the month should be a number from 1 through 12 and the day should be a valid number for the month (from 1 through 31 for January, from 1 through 29 for February, and so on).

If the date is valid, the subprogram should set a valid-date switch to Y; otherwise, it should set the switch to N. To use this program, the calling program should pass two fields to the subprogram in this sequence: (1) the eight-digit date field, and (2) the valid-date switch.

To test this program, you need to write a simple calling program that gets a date from the user and passes it to the subprogram. Then, if the valid-date switch is set to Y, this program should display a message like "Valid date." Otherwise, it should display a message like "Invalid date."

1. Start a new project named DATE3000 in c:\mcobol\chapter 11, and add a copy of the CALCFV program that's in the same folder to it after you rename the program EDITDATE. Next, change the Program-ID to EDITDATE, and modify the code so the subprogram does the function that's described above. Then, compile the subprogram, and close it.
2. Add a copy of the program named TESTEDIT that's in c:\mcobol\chapter 11 to the project. Then, modify this program so you can use it for testing the subprogram that you wrote in step 1. Compile this calling program.
3. Use the Animate→Settings command to make sure that the calling program, not the subprogram, is targeted in the Start Animating at text box. If it isn't, change the targeted name from EDITDATE to TESTEDIT. Then, close the dialog box.
4. Test and debug the calling program and subprogram.
5. When you're satisfied that the program and subprogram work correctly, close the project.

Before you start the exercises for section 3

Before you start the exercises for section 3, you should read unit 2 of the Net Express 4.0 tutorial. It presents the skills you need for working with all types of COBOL files. Later, when you do the exercises for chapters 14, 15, and 16, you can use unit 2 as a reference.

Exercise 13-1 Test the sequential update program

In this exercise, you'll test a version of the sequential update program that's presented in this chapter. That will give you a better idea of what testing this type of program involves.

Test the program with standard sequential files

1. Start a new project named SEQ1000 in c:\mcobol\chapter 13, and add the SEQ1000 program that's in the same folder to it.
2. Review the Select statements in SEQ1000. This is the program presented in figure 13-7, but the system names have all been changed so they're appropriate for PC files, and all of the files have been changed to line sequential organization.
3. Use your word processing program to print listings of the records in the files named rcttranl.dat and oldmastl.dat in the c:\mcobol\data folder. (The letter *l* indicates that these files contain line sequential data.) Although you could use the Data File Editor to display and print these files, it's just as easy to work with them in a word processor since they have line sequential organization.
4. Compile and run the program.
5. Use your word processing program to print listings of the records in the output files named newmastl.dat and errtranl.dat. Then, check the listings of the four files to see whether the updating is correct (not an easy process).
6. If you have any trouble understanding how this program works, set a breakpoint at the first statement in module 300 and run this program again. When the breakpoint is reached, step through the statements that implement the matching record logic. As you step, display the values in the control fields so you can see how they change. When you're satisfied that you understand how this program works, go on to the next step.

Add a Display statement to improve the testing process

7. Add a Display statement in procedure 340 that displays the entire master record each time a new master record is written. Then, compile and run the program again. This time, you can review the updated records in the screen display.
8. Find the errtranl.dat file in your Windows Explorer. Then, right-click on it and select Open With→ Notepad from the menu that's display to display the file in Notepad. Note that this file contains the two error transactions two or more times depending on how many times you ran the program. That's because this file is opened in extend mode, and the two error transactions are added to this file each time the program is tested.
9. When you're through experimenting, remove the Display statements, and close the project.

Exercise 13-2 Add branch names to a sales report

To give you a chance to use matching record logic in another type of program, this exercise has you enhance the summary report program that you created in chapter 5 by adding branch names to it so it looks something like this:

```
BRCH                        CUST                                SALES         SALES
NUM  BRANCH NAME             NUM   CUSTOMER NAME              THIS YTD      LAST YTD

 12  FORT WAYNE             11111  INFORMATION BUILDERS       1,234.56      1,111.11
                            12345  CAREER TRAINING CTR       12,345.67     22,222.22
                                           BRANCH TOTAL      13,580.23     23,333.33
 .
 .
 47  KANSAS CITY NW         12121  GENERAL SERVICES CO.      11,444.00     11,059.56
                            24680  INFO MANAGEMENT CO.       17,481.45     11,892.47
                            99999  DOLLAR SAVINGS BANK        5,059.00      4,621.95
                            76543  NATL MUSIC CORP.           2,383.46      4,435.26
                                           BRANCH TOTAL      36,367.91     32,009.24

                                            GRAND TOTAL     120,525.72    121,155.45
```

Here, the first two heading lines and the last two columns of the report are deleted so the report will fit on this page. Your report, however, should include them.

To get the branch names that are printed by this program, the program needs to read a file named brchmast that's in sequence by branch number with this format:

```
    01  BRANCH-MASTER-RECORD.
        05  BM-BRANCH-NUMBER    PIC 9(2).
        05  BM-BRANCH-NAME      PIC X(18).
```

To make this program work, you need to use matching record logic like the logic in the sequential update program. If the branch number in the customer record is matched by a branch record, the program should print the branch name in the first line of each branch group. Otherwise, the program should print "NO BRANCH RECORD" in the branch name column.

Enhance the structure chart for the program

1. Enhance the structure chart for the report program that's shown in figure 4-7 so it includes a module for reading a record from the branch master file.

Enhance the program

2. Start a new project named RPT6000 in c:\mcobol\chapter 13, and add a copy of the RPT3000 program that's in c:\mcobol\chapter 05 to it after you rename the program RPT6000.
3. Modify the program so it prepares the report shown above. To make this manageable, you can do this in two parts. First, modify and test the program so it prints the branch name in each customer line. Then, modify and test the program so it prints the branch name in just the first line of each customer group.
4. When you've got the program working correctly, close the project.

Exercise 13-3 Modify the sequential update program

In this exercise, you'll modify the sequential update program in figure 13-7 so it uses the logic of the sequential maintenance program in figure 13-11. That will give you a better idea of your design and coding alternatives. It will also give you a better appreciation of how the code in the sequential maintenance program works.

Design and modify the program

1. Start a new project named SEQ1100 in c:\mcobol\chapter 13, and add a copy of the SEQ1000 program that's in the same folder to it after you rename the program SEQ1100. Then, change its Program-ID to SEQ1100.
2. Create a structure chart for this program based on the structure chart for the sequential maintenance program in figure 13-10.
3. Modify the code in this program so it corresponds to the design that you created in step 2. As you code, use the program code in figure 13-11 as a guide. In particular, the logic in procedure 300 should be based on the settings of the need-transaction, need-master, and write-master switches.
4. Check the procedures that you've just created to make sure they turn the switches on and off in the same way that the program in figure 13-11 does. If you don't set the switches right, the program may get caught in a loop.
5. Modify the Select statements for all four files used by this program so that they work with record sequential instead of line sequential files. The names of these files are rcttran.dat, oldmast.dat, newmast.dat, and errtran.dat.
6. Use the Data File Editor to review the data in the rcttran and oldmast files. (The records in the rcttran file have a length of 23, and the records in the oldmast file have a length of 70.) Notice that, even though these are record sequential files, each record is displayed on a separate line. That makes the files easier to work with. When you're done reviewing these files, close them.

Compile and test the program

7. Compile and test the program. To be sure it works correctly, use the Data File editor to review the data in the newmast and errtran files. If the you run the program more than once, the errtran file may contain duplicates of the error transactions. In that case, use the Data File Editor to delete all of the records in this file. Then, run the program one more time and display the errtran file again to be sure the program worked correctly.
8. When you're sure that it works the way the previous version of this program worked, close the project.

Exercise 14-1 Run the indexed file creation program

In this exercise, you'll run the file creation program that's in figure 14-6. That will create an indexed inventory master file that you can use in exercise 14-2.

1. Start a new project named IND1000 in c:\mcobol\chapter 14, and add the IND1000 program that's in the same folder to it.
2. Compile and test the program. It should create an indexed inventory master file named invmasti.dat.
3. Use your Windows Explorer to find the file that you've just created in the c:\mcobol\data folder. Then, note that two files have been created by this program. Invmasti.dat contains the data, while invmasti.idx contains the index that lets you use random access with the file. That's the way Net Express implements an indexed file. On other systems, though, the data and index are stored in a single file.
4. Use the Data File Editor to display the invmasti.dat file. When you're done reviewing this data, close the file.
5. Close the file creation program, but note that you can rerun this program whenever you need to get a fresh copy of the inventory master file.

Exercise 14-2 Develop a random update program

In this exercise, you'll modify the sequential update program of chapter 13 so it updates the records in the inventory master file on a random basis. The logic is to read a receipt transaction record and update the master record with the same item number.

Create the program

1. Start a new project named IND3000 in c:\mcobol\chapter 14, and add a copy of the SEQ1000 program that's in the same folder to it after you rename the program IND3000. Then, change the Program-ID to IND3000.
2. Create a structure chart for the random update program based on the structure chart for the random maintenance program in figure 14-8.
3. Modify the IND3000 program so it randomly updates the indexed master file. As you work, use the chart that you created in step 2 and the code for the random maintenance program in figure 14-9 as a guide. For simplicity, the program can read a transaction record, apply the transaction when the master record is found, and rewrite the updated master. Change the Open statement for the errtran file so it's recreated each time the program is run.

Compile and test the program

4. Compile and run the program. To review the transaction records, open the rcttranl.dat file in the Data File Editor. Then, to find out whether the master records have been updated correctly, open invmasti.dat in the Data File Editor. That gives you an idea of whether the program worked right, but some of the data is hard to interpret.
5. As an alternative, add Display statements to the program that display each master record right after it has been read and right after each updated master record has been rewritten. These statements should be placed in the Read and Rewrite procedures. Now, compile and test the program again.
6. The trouble now is that the master file has been updated at least twice in steps 4 and 5 so it's hard to tell whether the on hand and on order fields have been updated correctly. To get a fresh copy of the indexed master file, run the IND1000 program again. Then, test the random update program again. This time you can tell for sure whether the program has worked correctly.
7. When you've got the program working right, close the project.

Exercise 14-3 Create an indexed customer master file

In this exercise, you'll run a file creation program that will create an indexed customer master file that you can use in exercises 14-4, 14-5, and 14-6.

1. Start a new project named IND5000 in c:\mcobol\chapter 14, and add the IND5000 program that's in the same folder to it.
2. Display the program, and review the Select statement for the indexed file. As you can see, it will create a file named custmsti.dat whose primary index is the customer number. In addition, the branch number is an alternate index with duplicates.
3. Also notice that the indexed file is accessed sequentially, which means that the sequential customer master file that's used as input must be in sequence by customer number. To accomplish that, this program includes a sort. If you aren't familiar with how sorting works, you may not understand some of the code in this program. But don't worry about that now.
4. Compile and test the program. To be sure it worked correctly, use the Data File Editor to display the indexed file. When you're done, close the file and the project.

Exercise 14-4 Retrieve records by primary key

In this exercise, you'll write an interactive program using Display and Accept statements that gets the data for customer records from the indexed customer master file that you created in exercise 14-3. For this program, the interactive session should look something like this (the user entries are shaded):

```
Enter a customer number.
Or, enter 00000 to end the program.
-----------------------------------------
11111
Number  Name                    YTD Sales
11111   INFORMATION BUILDERS     1,234.56
----------------------------------------
Enter a customer number.
Or, enter 00000 to end the program.
-----------------------------------------
00000
```

1. Start a new project named IND4100 in c:\mcobol\chapter 14, and add the IND4100 program that's in the same folder to it. This program contains code that will give you a good start on this program.
2. Modify the program so it does the required processing. If the user enters an invalid customer number, the program should print a message like: "No master record for customer number 00099."
3. Compile and test the program by entering both valid and invalid customer numbers. To make that easier to do, you can run the RPT3000 program that you created for exercise 4-2 and then print the summary report that's created by that program. Or, you can refer to the report in figure 4-2 of the book.
4. When you've got the program working right, close the project.

Exercise 14-5 Use alternate keys with duplicates

In this exercise, you'll modify the program that you created in exercise 14-4 so it displays the customer data for the branch number that the user enters. This time the interactive session should look something like this:

```
Enter a branch number.
Or, enter 99999 to end the program.
-----------------------------------------
12
Number  Name                    YTD Sales
11111   INFORMATION BUILDERS     1,234.56
12345   CAREER TRAINING CTR     12,345.67
----------------------------------------
Enter a branch number.
Or, enter 99 to end the program.
-----------------------------------------
99
```

1. Start a new project named IND4200 in c:\mcobol\chapter 14, and add a copy of the IND4100 program that's in the same folder to it after you rename the program IND4200. Then, change the Program-ID to IND4200.
2. Modify the program so it uses the branch number alternate key to get the data that the program requires. If the user enters an invalid branch number, the program should print a message like: "No customer records for branch 01." For a program like this, you can use either dynamic processing or skip-sequential processing, but use skip-sequential processing for this version of the program.
3. Compile and test until the program works correctly. Then, close the project.

Exercise 14-6 Use dynamic processing

In this exercise, you'll modify the program that you created in exercise 14-5 so it uses dynamic processing to get the same results.

1. Start a new project named IND4300 in c:\mcobol\chapter 14, and add a copy of the IND4200 program that you created in the last exercise to it after you rename the program IND4300. Then, change the Program-ID to IND4300.
2. Modify the program so it uses dynamic processing instead of skip-sequential processing. Use the file status code for the file so the program only reads the customer records that apply to each branch number that the user enters. Otherwise, the program should work the same way it did before.
3. Compile and test until the program works correctly. Then, close the project.

Exercise 15-1 Run the relative file creation program

In this exercise, you'll run a file creation program that will create a relative inventory master file that you can use in exercise 15-2.

1. Start a new project named REL4000 in c:\mcobol\chapter 15, and add the REL4000 program that's in the same folder to it. Then, compile and test the program. It should create a relative inventory master file named invmastr.dat.
2. Use the Data File Editor to display the file that you've just created. Be sure to select Relative for the file organization, and enter 70 for the record length. When you're done reviewing the data, close the file.
3. Close the file creation program, but note that you can rerun this program whenever you need to get a fresh copy of the inventory master file.

Exercise 15-2 Develop a random update program

In this exercise, you'll modify the random update program from chapter 14 so it works with the relative file you created in exercise 15-1.

1. Start a new project named REL3000 in c:\mcobol\chapter 15, and add a copy of the IND3000 program that's in c:\mcobol\chapter 14 to it after you rename the program REL3000. Then, change the Program-ID to REL3000.
2. Change the Select statement for the indexed inventory master file so it's appropriate for a relative file named invmastr that will be accessed randomly.
3. Make the necessary changes to the File Section, the Working-Storage Section, and the Procedure Division of this program so it works with the relative file. The relative record number should be calculated by subtracting 10000 from the item number field in the input file.
4. Compile, test, and debug the program until it works correctly. Then, close the project.

Exercise 16-1 Modify the sales report program to include a sort

In this exercise, you'll modify the two-level sales report program so it sorts the records in the customer file in customer number within salesrep number within branch number sequence before it prints the report. The specifications for this report are shown in figure 5-18.

Modify the structure chart

1. Modify the structure chart shown in figure 5-19 to include a sort function. To do that, you'll need to add another level that includes three modules that represent the main functions of this program: (1) get the input records; (2) sort the input records; and (3) print the report. To keep the changes to the code simple, number these modules 025, 050, and 075. Also, change the name of module 310 to indicate that it will perform a return function rather than a read function.

Modify the code

2. Start a new project named RPT7000 in c:\mcobol\chapter 16, and add the RPT7000 program that's in the same folder to it.
3. Add a Select statement, an SD statement, and a record description for the sort work file. Also, change the system name in the Select statement for the customer file so it gets data from a file named custmstu.dat, which isn't in the right processing sequence.
4. Add a Sort statement to module 000. This statement should include a Using clause to get the records from the customer master file, a Key clause to sort the records in the appropriate sequence, and an Output Procedure clause that names the procedure that prints the report.
5. Add the code for the three modules you added to the structure chart in step 1. The Sort module should be coded as a comment since its function is performed by the Sort statement. The Get module should also be coded as a comment since its function is performed by the Using clause of the Sort statement. The third module, which represents the output procedure, should include the code for printing the report that was originally in module 000.
6. Modify the code in the output procedure so it retrieves records from the sorted customer file instead of from the customer master file.

Compile and test the program

7. Compile and test the program. When you run the program, you should get an error message indicating that the customer file is already open. Because the Sort statement includes a Using clause instead of an input procedure, this statement will open and close the file automatically. So you can delete the Open and Close statements for this file. Make this change, then compile and test the program again. When you're sure that it works the way the previous version of this program worked, close the project.