

# 2

## How to compile and test a program with Personal COBOL

In this chapter, you'll learn how to enter, compile, test, and debug a program when you're using Micro Focus Personal COBOL on a PC. When you finish this chapter, you'll be able to develop programs on your own using that compiler.

In addition, this chapter will give you a good background for learning how to develop programs on other platforms or with other compilers. Even if you use a different compiler on the job or at your school, though, we recommend that you use Personal COBOL to learn on your own PC. It is an inexpensive product that is a terrific training tool.

If you do use a different compiler on the job or at your school, you will of course have to learn how to compile and test programs with that compiler too. If, for example, you're going to develop programs for an IBM mainframe, chapter 18 shows you how to compile and test programs on that platform. You can read that chapter any time after you complete this one.

<b>Introduction to Micro Focus Animator .....</b>	<b>50</b>
The Integrated Development Environment .....	50
The two operating modes .....	52
A basic procedure for developing programs with the Animator .....	54
How using Personal COBOL differs from developing production programs .....	56
<b>How to enter and edit a program .....</b>	<b>58</b>
How to start a new program .....	58
How to enter and edit code .....	60
Special features for working with code .....	62
How to set the default folder .....	64
How to print a source program .....	64
<b>How to compile and test a program .....</b>	<b>68</b>
How to compile a program .....	68
How to correct compile-time errors .....	70
How to test a program .....	72
How to correct run-time errors .....	74
<b>How to use the debugging features .....</b>	<b>78</b>
How to display and modify the values of variables .....	78
How to use breakpoints .....	80
How to step through a program .....	82
A summary of the other debugging features .....	84
<b>Perspective .....</b>	<b>87</b>

## Introduction to Micro Focus Animator

---

To develop a program using Micro Focus Personal COBOL, you use the Micro Focus Animator. This Animator works essentially the same way for all versions of Micro Focus COBOL, including the versions for developing PC, UNIX, and mainframe applications. As a result, you'll be able to transfer the skills that you learn in this chapter to any of the other Micro Focus COBOL products.

### The Integrated Development Environment

---

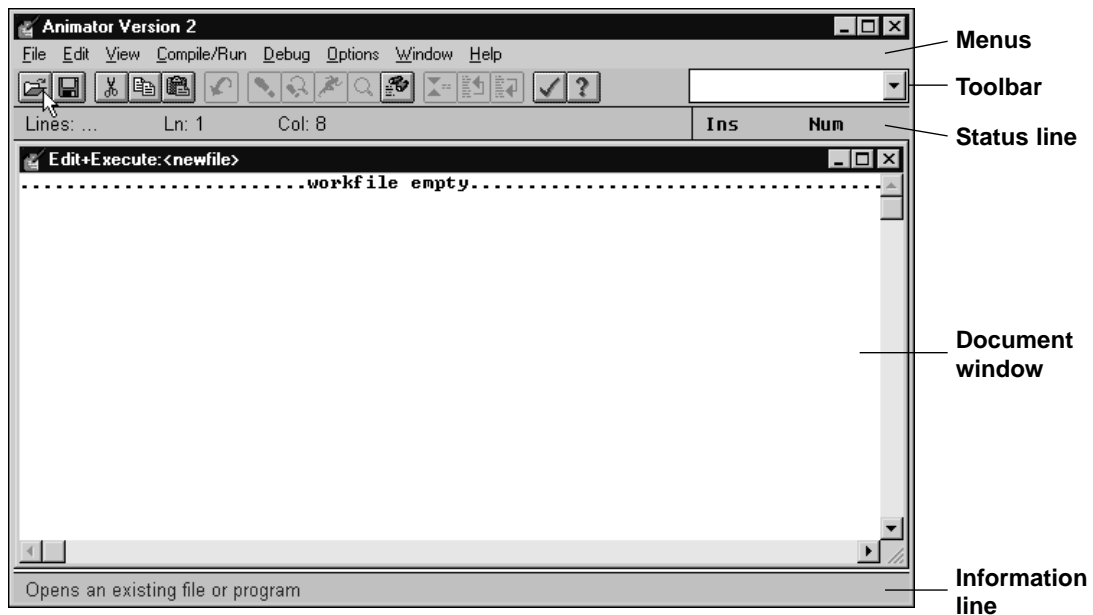
When you start Personal COBOL, an application window like the one shown in figure 2-1 is displayed. This is the *Integrated Development Environment*, or *IDE*, of Personal COBOL. The key component of this IDE is the Animator because you use it for entering, compiling, and testing code.

Within the application window are one or more *document windows*. By default, an empty document window is displayed when you start the Animator. However, you can open additional document windows to display existing program files or to create new files.

As in most Windows programs, you can use the menus in the Animator to issue commands. In the File menu, for example, you'll find commands for opening, saving, and printing files. You can also issue some of the most common commands by using the buttons in the toolbar. The first button on the toolbar, for example, lets you open an existing file.

Before you go on, you should know that Personal COBOL was written for Windows 3.1 systems. So if you're using a more recent operating system, such as Windows 95 or Windows 98, the Animator interface may seem somewhat dated. Nevertheless, you shouldn't have any trouble using Personal COBOL with one of these operating systems because this chapter presents everything you need to know.

## The starting window for the Micro Focus Animator



### Description

- When you first start the Animator, an empty *document window* is displayed where you can enter the code for a new program. Usually, though, you'll start a new program from an existing program as shown in figure 2-5.
- You can use the menus or the toolbar buttons to issue commands. To find out what a toolbar button does, place the mouse pointer over it and a description is displayed in the *information line* at the bottom of the window. The information line is also used to display status information about various operations.
- The *status line* indicates the number of lines in the current file and the current location of the cursor in that file. At the right side of the status line is the *key status area*, which indicates the status of the Insert, Caps Lock, and Num Lock keys.

Figure 2-1 The Integrated Development Environment for the Animator

## The two operating modes

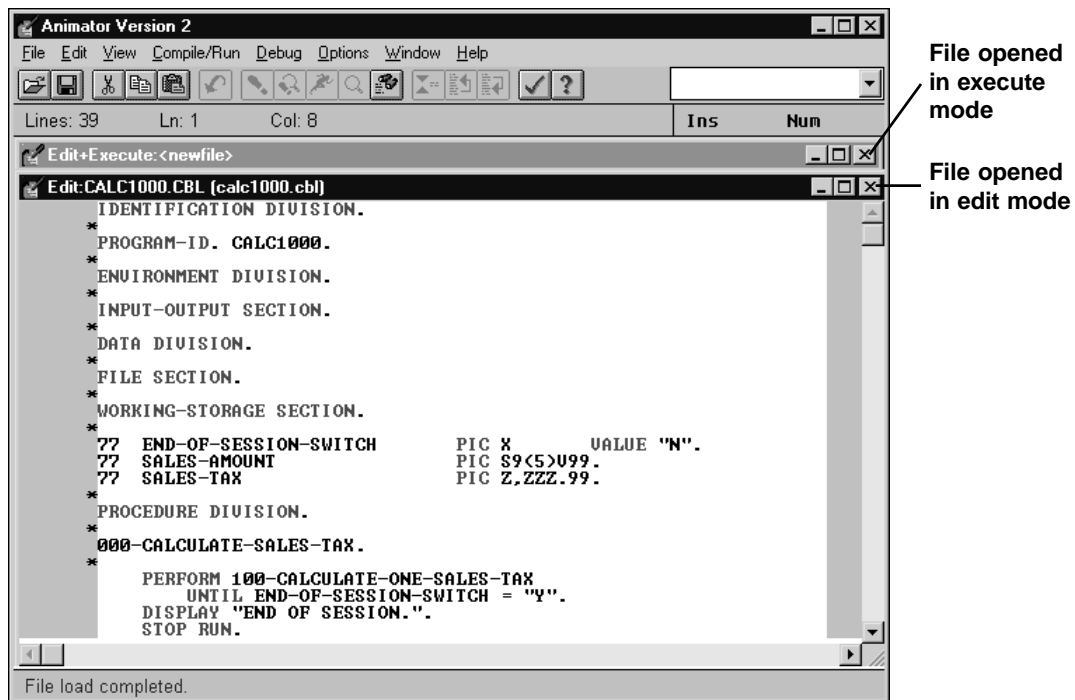
---

When you use the Animator, you can open a file in two different modes. When you use the Open for Edit command in the File menu, the file is opened in *edit mode*. When you use the Open for Execution command, the file is opened in *execute mode*.

In either case, the mode of the file is indicated by the title bar of its document window. In figure 2-2, for example, the active window is opened in edit mode. In contrast, the inactive window is in execute mode (this is the empty document window that's opened by default when you start Animator).

When you work with a file in edit mode, you can only enter, edit, and compile the code in the file. Then, after the program compiles with no errors, you need to open the file in execute mode so you can test it. In this mode, you have access to all of the debugging tools that the Animator provides. Although you can open up to 50 programs in edit mode, you can open only one document at a time in execute mode.

## The Animator with two open windows



## Description

- An Animator window can be opened in either *edit mode* or *execute mode*. The mode is indicated in the title bar of the window along with the name of the file that's displayed in the window.
- You use edit mode to enter and compile the code for a program. To open a program in edit mode, select the Open for Edit command from the File menu or click on the Open button in the toolbar.
- You use execute mode to test and debug a program after it has been compiled. To open a program in execute mode, select the Open for Execution command from the File menu.
- You can open up to 50 windows in edit mode, but you can open only one window in execute mode.
- If you select the New command from the File menu, a new document window is opened in edit mode.

Figure 2-2 The two operating modes of the Animator

## A basic procedure for developing programs with the Animator

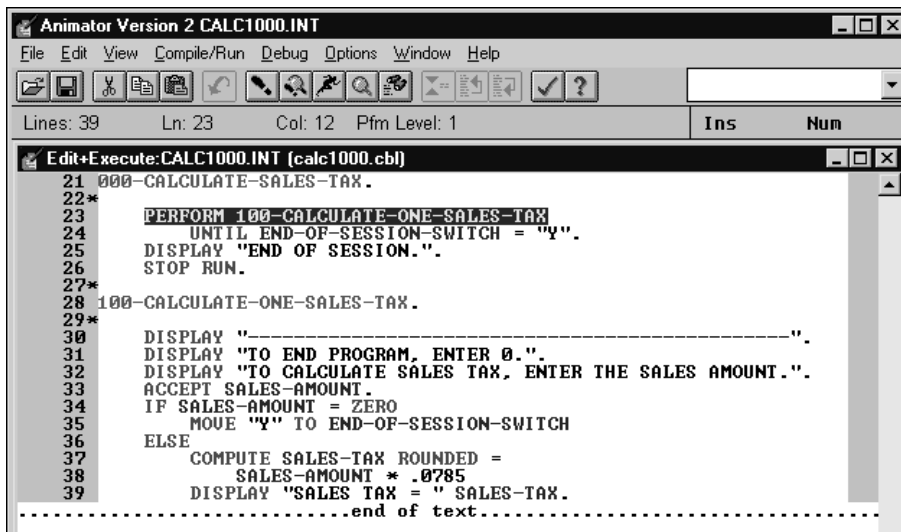
---

To develop a COBOL program with the Animator, you follow a procedure like the one outlined in figure 2-3. In step 1, you use edit mode to enter the *source code* for the program. This result is a *source program* that's stored in a *source file*. In step 2, you compile the source program to create an *intermediate file* that the Animator can execute.

At this point, you open the intermediate file in execute mode so you can do the next two steps. Then, in step 3, you *test* the program to see whether it works correctly. Here, your job is to test all possibilities until you're sure the program works correctly. For most programs, though, you'll find one or more errors. So in step 4, you *debug* the program, which means that you find and correct the causes of the errors.

As you develop COBOL programs, you'll run into three types of errors. A *compile-time error* occurs when the compiler can't convert a statement into machine language because its syntax is incorrect. Even if a program compiles without errors, though, the Animator may not be able to execute some of the statements in the program. In that case, a *run-time error* occurs. Finally, a *logical error* occurs when the program runs successfully, but the results aren't as expected.

## An intermediate file opened in execute mode



## A basic program development procedure

1. Use edit mode to enter the *source code* for the program, and save this *source program* in a file with *cbl* as the extension (the default). This file can be referred to as the *source file*.
2. Compile the program to check its syntax, and correct any errors that are detected. When the program compiles without errors, an *intermediate file* is created. This file has the same name as the source file, but has an *int* extension.
3. Open the intermediate file in execute mode, and run it to see whether it works. This is known as *testing* a program.
4. If the program doesn't work correctly, find the errors, correct them, recompile the program, and test the program again. This is known as *debugging* a program.

## Description

- The COBOL compiler compiles a program into machine language that the computer can understand. If a statement can't be compiled into machine language, a *compile-time error* occurs. See figure 2-10 for information on correcting compile-time errors.
- If a statement compiles cleanly but can't be executed, a *run-time error* occurs when you run the program. See figure 2-12 for information on correcting run-time errors.
- Even if a program runs without encountering any errors, the results of the program may not be as expected. This type of programming problem is referred to as a *logical error*.
- The Animator provides debugging tools that help you find and correct the run-time and logical errors.

Figure 2-3 A basic procedure for developing programs with the Animator

## How using Personal COBOL differs from developing production programs

---

When you use Personal COBOL, the Animator always controls the execution of the programs that you develop. As a result, you can't use Personal COBOL to develop *production programs* that are used on the job because those programs have to run by themselves.

To illustrate the difference, figure 2-4 shows how a production program is developed. After the programmer uses an interactive editor to create the source program, the programmer initiates a three-step procedure that's run by the computer. As you can see, the output of the second step is an *executable* that is run by itself in the third step. In contrast, Personal COBOL isn't able to produce an executable, so you always have to run your programs through the Animator.

In step 1 in this figure, the COBOL compiler compiles the source program into an *object module* (or *object program*). If the source program uses any Copy statements (see chapter 11), the compiler copies the source code from the related copy members into the program as part of this process. The compiler also produces output like a compiler listing and a list of the compile-time errors (often called *diagnostics*). Sometimes, this output is printed, but it is often reviewed on the monitor or screen without ever printing it.

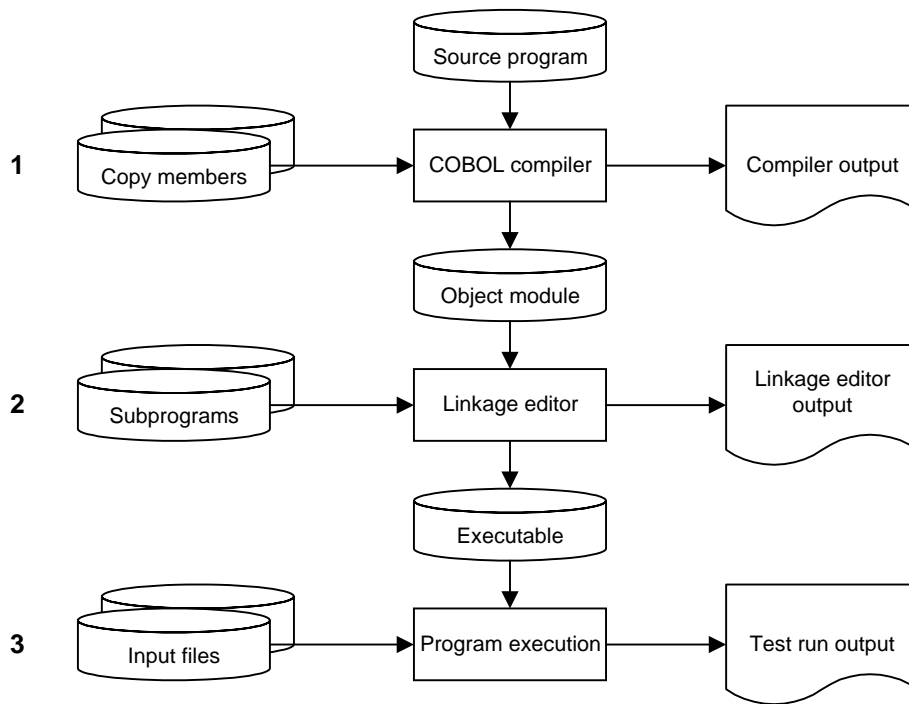
In step 2, the *linkage editor* program *link edits* (or *links*) the object module with any subprograms that the program requires into an executable. On most platforms, one or more system subprograms are required in this step. These subprograms do some of the specific types of processing that the program requires. In addition, the linkage editor can link the object module with user subprograms (see chapter 11). Although the linkage editor also produces some printed output in this step, programmers rarely need to refer to it.

In step 3, the executable version of your program is executed. This is the test run of your program. As a result, the program gets the input that it requires including keyboard or disk data, and the program produces the output it requires including display, disk, or printer data.

With minor variations in the terminology, this is the way a production program is prepared on all platforms: compile, link edit, and test. If you want to see how this is done on an IBM mainframe, please read chapter 18.



## The three-step procedure that's done by the computer



### Description

- Before the three-step procedure shown above can be run by the computer, the programmer enters the source program using an interactive editor like Micro Focus Animator.
- When the source program is ready, the programmer initiates a three-step procedure that's done by the computer. In step 1, the COBOL compiler compiles the source program into an *object module*. In step 2, the *linkage editor* link edits the object module into an *executable*. In step 3, the executable is run so the programmer can see whether the program works.
- During step 1, the COBOL compiler inserts the source code that's in the copy members that are referred to by any Copy statements in the program (see chapter 11). The compiler also produces output like a compiler listing or a list of any compile-time errors.
- During step 2, the linkage editor link edits the object program with any system subprograms or user subprograms that it needs (see chapter 11). The linkage editor also produces linkage editor output.
- Step 3 is the test run for the program. It gets whatever input the program specifies and produces whatever output the program specifies. In contrast to the way you test a program with Micro Focus Personal COBOL, this test run is independent of the development environment.

Figure 2-4 How a production program is compiled, link edited, and tested

## How to enter and edit a program

---

Now that you have a general idea of how to use the Animator to develop COBOL programs, you're ready to start working on your own programs. So the topics that follow show you how to enter and edit a program and how to use some of the Animator features that will save you time as you work.

### How to start a new program

---

If you want to start a new program, you can enter code into the empty document window that's displayed when you first start the Animator. Or, you can open a new document window in edit mode using the New command on the File menu. Because every COBOL program requires some of the same coding, though, you should rarely, if ever, start a new program from scratch.

Instead, you should start a new program from an old program that is similar to the one you're going to develop. At the least, the old program will have the required division and section headers in it. And it may have a lot of other code that you can modify for use in the new program.

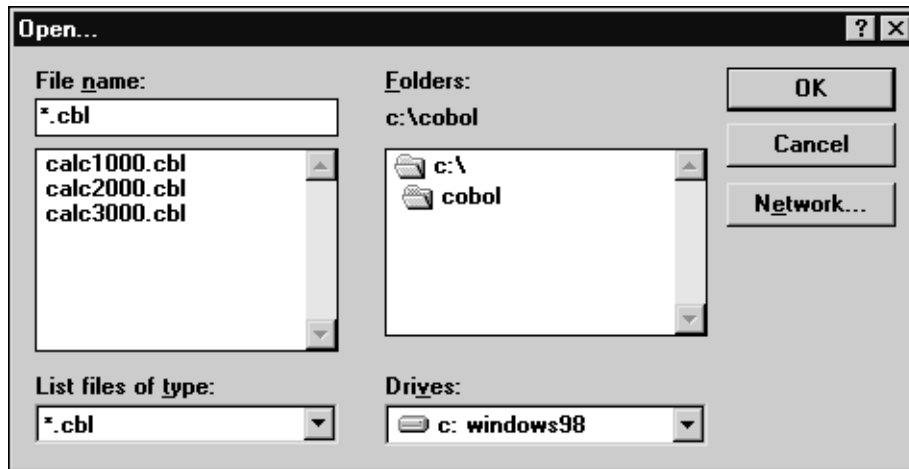
Figure 2-5 shows you how to start a new program from an old one with Personal COBOL. In step 1, when you use the Open for Edit command to open an old program, the Open dialog box is displayed. Since Personal COBOL originally was written for use with Windows 3.1, though, this dialog box uses the Windows 3.1 interface. In case you aren't familiar with this interface, this figure shows you how to use it to set the current folder to the one that you use for your programs. Later in this chapter, you'll learn how to change the default folder so it's displayed whenever you open a file.

As soon as you open the file, you should use the Save As command to save it with a new name. That way, you won't forget to do that later on and accidentally save the changes you make to the original file. Here again, the Save dialog box uses the Windows 3.1 interface, so you can change the current folder using the techniques shown in this figure. In addition, you have to use eight or fewer characters in the new file name with no intervening spaces because that's what Windows 3.1 required. This is true even if you're using Personal COBOL with Windows 95 or 98.

After you've saved the file, we recommend that you close it, then open it again to edit it. That's because of a quirk in the way Personal COBOL works. If you bypass this step, you can't compile the program immediately after editing it. Instead, the Animator closes the file and returns you to the old program you opened in step 1. So closing and re-opening the file as suggested in step 3 will save you some frustration later on.

Once you've saved the old program with the new name, you can delete the statements you don't need, modify the statements you do need, and add all the new statements that you need. Often, you can pick up dozens of statements from an old program when you start a new program this way. That's why this is one of the keys to increased programmer productivity.

## The Open dialog box



## How to start a new program from an old program

1. Use the Open for Edit command in the File menu to open an old program that is similar to the new one that you're going to develop.
2. Use the Save As command in the File menu to save the file with a new name that contains eight or fewer characters.
3. Close the file, then open it again for editing.
4. Delete the portions of the old program that you don't need.
5. Modify the portions of the old program that are appropriate for the new program.

## How to change the current folder in the Open or Save dialog box

1. To change the drive, choose a new drive from the Drives drop-down list.
2. To change the folder, double-click on the top-level folder in the Folders list box (c:\ in the example above) so all of its subfolders are displayed. Then, double-click on the first subfolder in the path that you want to establish, and continue double-clicking on subfolders until you reach the folder you want.

Figure 2-5 How to start a new program from an old program

## How to enter and edit code

---

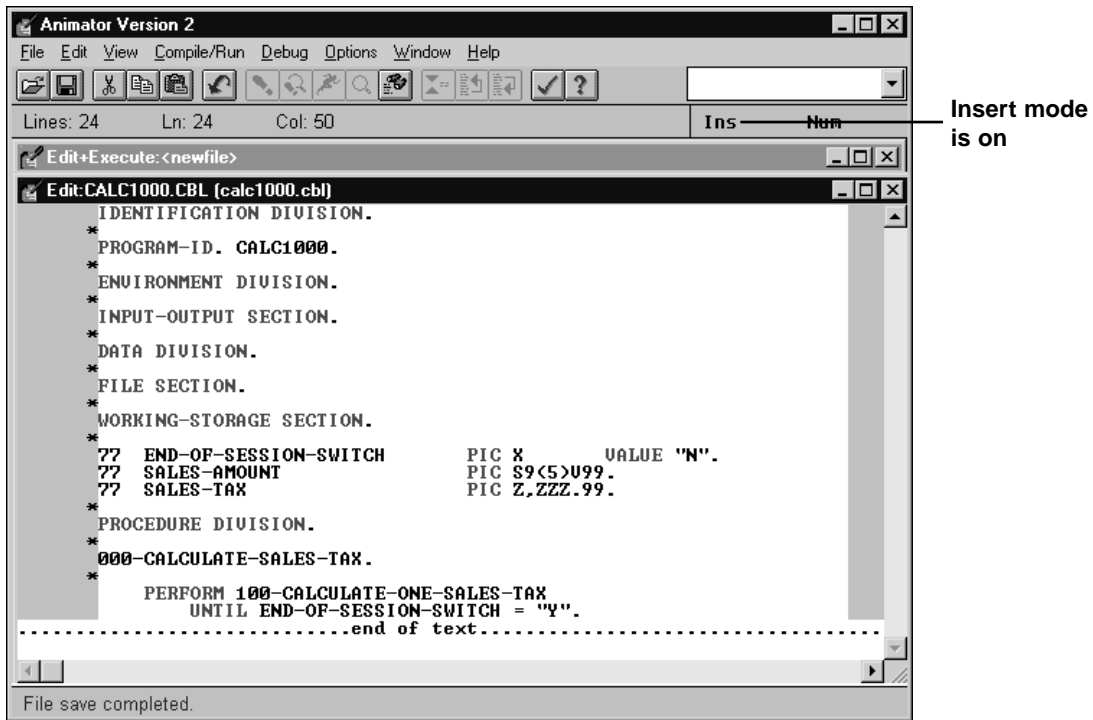
Before you learn how to enter code, you should know that you typically compose a program right at the keyboard. In other words, you don't write it down on paper before you enter it, although that used to be common practice. Today, interactive compilers make it easy to change program code on the fly, so it's more efficient to just enter the program code directly at the keyboard. Although this may take some practice at first, that's the way professional programmers work.

Figure 2-6 shows you how to enter COBOL code into the document window. Notice that the left and right margins are set the way you want them for a COBOL program because that's how the *COBOL profile* for the Animator is set. In addition, this profile provides for tab stops every fourth position so it's easy to align and indent your code.

If you've ever used a word processor, you shouldn't have any trouble using the Animator since the basic editing techniques are the same. The main difference is that you enter COBOL programs one line at a time, and you press the Enter key at the end of each line to start the next line. With a word processor, you have to press the Enter key only when you want to start a new paragraph.

Because this book assumes that you are already familiar with the way Windows programs work, it doesn't present detailed operational instructions for entering and editing code. As a result, this figure just summarizes some of the typical editing operations. Later, when you experiment with the Animator, you'll see that entering and editing a program isn't too much different from entering and editing a document, although the Animator features aren't quite as slick as those of a word processor.

## The Animator while a program is being entered into it



### Description

- The Animator uses colors to identify various elements of code. By default, the COBOL reserved words are displayed in green and the programmer entries are displayed in black.
- The *COBOL profile* is set so the white area of the document window represents positions 8 through 72 with tab stops at every fourth position. The gray areas represent positions 1-7 and 73-80.

### How to enter code

- To move to the next tab stop, press the Tab key. To move to column 7 so you can enter an asterisk into it, use the left arrow key or click on it with the mouse. To start a new line, press the Enter key.
- To turn insert mode on or off, press the Insert key. This controls whether you insert new code or type over existing code when you type.

### How to edit code and undo the last editing operation

- To delete one or more lines of code, highlight the lines and press the Delete key. To move or copy lines of code, highlight them and use the standard Windows Cut (Ctrl+X), Copy (Ctrl+C), and Paste (Ctrl+V) commands.
- Click on the Undo toolbar button to undo up to 100 operations (Ctrl+Z doesn't work).

Figure 2-6 How to enter and edit code

## Special features for working with code

---

Figure 2-7 presents some of the time-saving features you can use as you enter and edit your program. To start, this figure shows you how to display the *button bar* because its buttons are easier to use than the toolbar buttons. However, this bar isn't displayed by default. As a result, you need to do this procedure so the button bar is displayed in all of your coding sessions.

Once you've got the button bar displayed, you can use its buttons to start many of the common Animator functions. One of these buttons is the Find button, which can be used to start the Find feature. You can use this feature to find one or more occurrences of the text that you enter. If you check the Replace box in this dialog box, you can also enter the text for a find and replace operation.

When you click on the All button to find all the occurrences of the text, the lines that contain those occurrences are tagged and highlighted as shown in this figure. Then, you can move from one *tagged line* to another as described in this figure. You can also *compress* all of the tagged lines so only those lines appear in the document window, after which you can *expand* the tagged lines so all of the lines appear again.

If you find yourself having to scroll back and forth between two or more procedures or statements in a long program, you may want to mark them with a generic *tag* so you can move to them more easily. To do that, you can right-click on a line number and select the Tag command from the shortcut menu. Then, this tag appears in the left margin of the statement:

**Tag→**

Once the statements are tagged, you can move from one tag to another the same way that you move from one Find tag to another. You can also compress and expand these tags.

If more than one type of tag is displayed at the same time, you can move from one type of tag to another by placing the cursor in a line that has the type of tag you want to move to before you start the move operation. If, for example, both Tag and Find tags are displayed, you can move the cursor to a Tag line before you press the F8 key to move to the next Tag line. If you start a move operation when the cursor isn't in a tagged line, the Animator displays a small menu that lets you choose the type of tagged line that you want to move to.

Similarly, you can compress all the lines except those that have a certain kind of tag. To do that, place the cursor on a line that has that type of tag and click on the Compress button. Or, select the tag type from the shortcut menu that's displayed when you click on the Compress button.

The easiest way to remove all of the tags in a program is to compile the program, which you'll soon learn how to do. Otherwise, you can right-click on a generic tag and choose the Unset command in the shortcut menu to remove that tag. Or, to remove all Find tags, you can use the Clear Finds command in the Edit menu. Most of the time, though, you'll just compile the program to remove all tags.

## The Find dialog box and the highlighted lines that result from its command



### How to display the button bar

- Pull down the Options menu and choose the Configure Interface command. In the dialog box that's displayed, click on the Button Bar icon at the far left (the second one from the top) so its options are displayed. Then, check the Bar Visible option, click on the Save button, click OK on the message that's displayed, and click on the Close button.

### How to use the Find command to find and replace text

- To display its dialog box, click on the Find button and select the Text option. Then, after you enter the text for what you want to find and replace, you click on the buttons to set options, to find or replace one occurrence of the text, or to find or replace all occurrences. When you click on the All button, all of the lines that contain the occurrence are tagged.

### How to move to, compress, and expand tagged lines

- To move from one tagged line to another, click on the Previous or Next toolbar button or press the F7 or F8 key.
- To compress the display so only the tagged lines are shown, click on the Compress button (which then turns into an Expand button). To redisplay all the lines of code, click on the Expand button.

### How to remove tagged lines

- Compile the program as shown in figure 2-9. Or, to remove Find tags, choose the Clear Finds command in the Edit menu.

Figure 2-7 Special features for working with program code

As you experiment with the Animator, you'll discover that it has many other features. You'll also discover that you can start most features in more ways than figure 2-7 shows, including using regular menus, shortcut menus, toolbar buttons, and button bar buttons. This figure, though, presents some of the most useful features and the most efficient ways to work with them.

## How to set the default folder

---

By default, when you save a file, the Animator saves it in the same folder where Personal COBOL is installed. And when you open an existing file, the default is to look in that folder for the file. In most cases, though, you'll want to save the files you create in a different folder. Although you can switch to the appropriate folder every time you save a new file or open an existing file, it's easier to just change the default folder. Figure 2-8 shows you how to do that.

As you can see in this figure, you change the default folder from the Properties dialog box for Personal COBOL. The easiest way to display this dialog box is to locate Personal COBOL on the Windows Programs menu, right-click on it to display its shortcut menu, and choose the Properties command from that menu. Then, you can enter the path for the folder you want to use as the default in the Start In box. In this case, the default folder has been changed to C:\cobol. (Since this isn't case sensitive, you don't have to worry about the capitalization.)

## How to print a source program

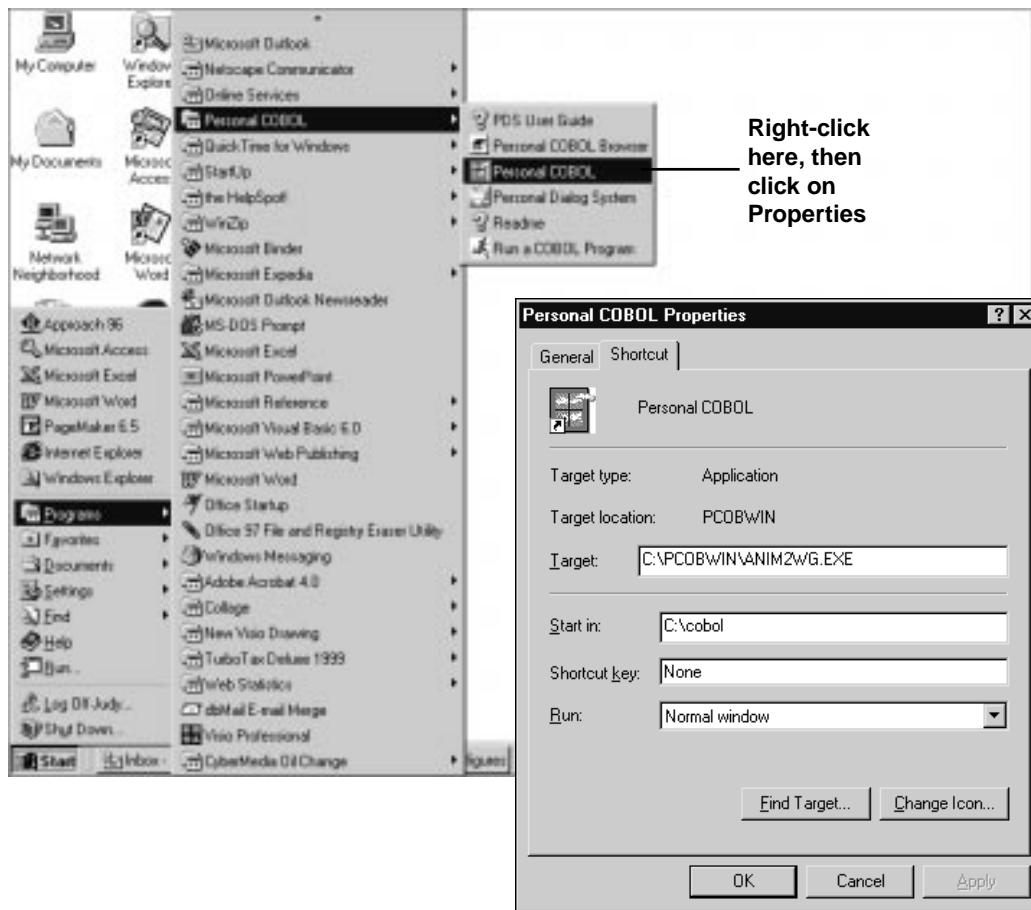
---

To print the source code for a program from the Animator, you can use the Print command in the File menu. This can be useful when you're debugging a long program or correcting its compile-time errors. Unfortunately, there is no way to print selected pages, so you have to print the entire program even if you make just a minor change to it. As you get used to working with programs on the screen, though, you'll find that you rarely need printed listings.

Another way to print a source program is to use a text editor like NotePad or a word processing program like Word. After you open the source file, you can change its formatting and print the entire program or selected portions. Often, this works better than using the Animator to print the program.



## The Properties dialog box for Personal COBOL and how to display it



### How to display the Properties dialog box

- Click on the Start button in the Windows taskbar and locate the Personal COBOL entry in the Programs menu. Right-click on the entry to display the shortcut menu, and choose the Properties command.

### How to change the default folder

- Change the Start In option in the Properties dialog box to the folder that will contain your program files and click on the OK button. The folder you specify will be displayed by default each time you open or save a file. This change will take effect the next time you run Personal COBOL.

Figure 2-8 How to set the default folder

## About the exercises in this book

If you're new to programming, the best way to master COBOL is to practice on a PC. That's why this book provides practice exercises that guide you through the process of developing COBOL programs. These exercises are designed for use with Micro Focus Personal COBOL because it's an inexpensive product that is an excellent training tool.

If you don't already have Micro Focus Personal COBOL, the last page in this book shows you how to order and install it. That page also shows you how to copy the programs and data from the CD ROM in this book to the C drive on your PC. Once you've installed Personal COBOL and copied the programs and data, you're ready to start the exercises.

### Exercise 2-1 Get started with the Animator

This exercise helps you get started with Animator by guiding you through some common operations. It also guides you through the process of setting the defaults so the Animator is easier to use.

#### Set the default folder for the Animator

1. Click on the Windows Start menu, locate the Personal COBOL entry in the Programs menu as shown in figure 2-8, and right-click on it. Then, click on the Properties option in the shortcut menu that appears.
2. In the dialog box that appears, enter `c:\cobol` in the Start In text box as shown in figure 2-8, and click on the OK button. We recommend that you use this folder for all of the programs that you develop.

#### Start Personal COBOL and review its buttons

3. Start Personal COBOL, maximize its window, and close the window for the new file.
4. Use your mouse to point to one of the toolbar buttons, and look at the description that appears in the information line at the bottom of the screen. Then, use this technique to review the other toolbar buttons.
5. Use the procedure in figure 2-7 to display the button bar. Then, move the mouse pointer to one of the buttons and note the description in the information line.

#### Start a new program from an old program

6. Click on the Open toolbar button to start the Open command. If you did steps 1 and 2 right, the folder should be set to `c:\cobol`, and several files should be listed. Then, double-click on `calc2000` to open that file.
7. Use the Save As command in the File menu to save the file as `int0000`. Then, close the file and open it for edit again (this is necessary due to a quirk in the way Personal COBOL works).

### Navigate through the program

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, click on the vertical scroll bar to move through the program.
9. Press the Ctrl key plus the left or right arrow key to move from word to word in the program. Then, press the Ctrl key plus the up or down arrow key to move from paragraph to paragraph.

### Edit the program

10. Press the Insert key and notice how this turns insert mode on or off as indicated by *Ins* in the status line. With insert mode off, move to the Program-ID paragraph and change the program name to INT0000.
11. Move to the start of procedure 120. Next, hold down the Shift key as you press the right arrow 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.
12. 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.
13. Repeat step 12, but cut the lines instead of copying them and use the toolbar buttons instead of the shortcut keys for cutting and pasting. Then, undo that operation.

### Use the Find command and generic tags

14. Use the Find command as shown in figure 2-7 to find each occurrence of FUTURE-VALUE. Next, use the Previous and Next toolbar buttons or the F7 and F8 keys to move from one occurrence to another. Then, compress and expand the source code. Last, use the Clear Finds command in the Edit menu to clear the find tags.
15. Repeat step 14 with the “Match whole words only” option on.
16. Tag the line in working storage that defines FUTURE-VALUE and the first line in procedure 120 that refers to this field. To do that, right click on each line number and select the Tag command from the shortcut menu. Next, use the F7 and F8 keys to move from one tag to another. Last, remove the tags by using the shortcut menus again.

### Continue to experiment, then close the program

17. Review the commands in the File, Edit, and View menus to see that you can also use these menus to start commands. Then, experiment on your own until you feel comfortable with the way the Animator works.
18. Save your changes and close the file using the Save button in the toolbar and the Close command in the File menu.

## How to compile and test a program

---

Once you've entered the source code for a program, you're ready to compile and test it. During these steps of program development, you'll find and correct any errors in your program.

### How to compile a program

---

Figure 2-9 shows how to compile a program. The easiest way to start the compile is to click on the Compile button in the button bar. Then, you respond to any dialog boxes that are displayed until the compile is finished.

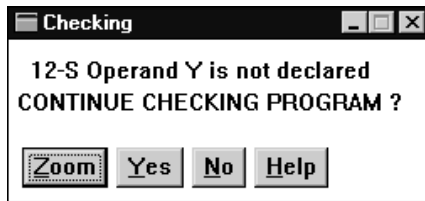
When you compile a program, the Animator checks the syntax of the code to be sure it's correct. When it's done, the Animator displays a window that lists all the errors it found as shown in the next figure. In addition, each line of code that contains an error is highlighted in the document window, and an Error> tag is placed in the left margin so the errors are easy to locate.

The first time you compile a program, sequence numbers are added in the left margin. These numbers are updated each time you compile the program so they're always in sequence. In addition, various elements of the code are displayed in different colors so they're easy to identify. For example, all the data names are displayed in red.





## The dialog box that's displayed when a compile-time error occurs



## The window that describes all the compile-time errors

### The four buttons in the Checking dialog box

Zoom	Completes the compile without displaying any additional errors
Yes	Completes the compile and displays additional errors
No	Cancels the compile
Help	Displays a description of the error

### Description

- When an error occurs during a compile, the Checking dialog box is displayed with a description of the error. Then, you usually click on the Zoom button so that dialog box isn't displayed any more but the compiler keeps on checking for errors. When the compiler finishes, the Syntax Errors window is displayed and the error lines are highlighted as shown in figure 2-9.
- If you double-click on an error in the Syntax Errors window, the cursor moves to the line of code that contains the error.
- To move from one error to another in the document window, click on the Previous or Next toolbar button or press the F7 or F8 key.
- To compress the display so only the error lines are displayed, click on the Compress button (which then changes to an Expand button). To expand the display, click on the Expand button.
- If you study the error messages and the source code, you should be able to figure out what's wrong with each highlighted statement. Then, you can correct the errors and compile again.
- If you need more information on an error, you can right-click on the highlighted statement and choose the Error Help command from the shortcut menu that's displayed.

---

Figure 2-10 How to correct compile-time errors

## How to test a program

---

Figure 2-11 shows you how to run a program so you can test whether it works right. If you've been working in edit mode and you've just gotten a clean compile, you first need to open the program for execution. When you do that, you'll notice that the first executable statement in the program is highlighted. Then, you can just click on the Run button to start the execution of the program.

When you run a program that uses Accept or Display statements, a *text window* like the one at the top of this figure is displayed. Then, whenever an Accept statement is executed, the program waits for your entry. After you enter a value and press the Enter key, the program continues.

As you will soon see, the text window that's used for an Accept or Display statement is small, dark, and hard to read. But if the program is executed on its own (not within the Animator), it will use the full display screen. As you learned earlier in this chapter, though, one of the limitations of Personal COBOL is that it doesn't let you create executables that can be run on their own. To do that, you have to use a more full-featured compiler like the ones you learned about in chapter 1.

When a program runs until the Stop Run statement is executed, it is a *normal program termination*. In that case, a dialog box like the one in this figure is displayed with a *return code* of zero. But that doesn't mean the program is correct. You still have to study the output of the program to make sure that it worked correctly. If it didn't, you need to debug the program.

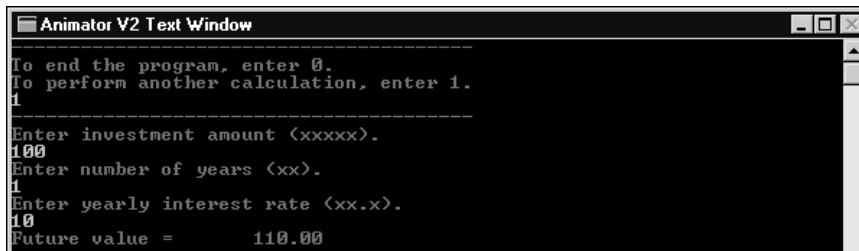
As you debug a program, you may need to run it two or more times to determine why it isn't working correctly. To re-run a program, though, you can't simply click on the Run button again. That's because when a program ends, whether it terminates normally or with a run-time error, the last statement that was executed is highlighted. If you click on the Run button at this point, the Animator will attempt to execute this statement again, which isn't what you want. Instead, you have to restart the program as described in this figure.

This figure also presents two techniques for cancelling the execution of a program. The first technique (pressing the Ctrl and the Break keys at the same time) is particularly useful for cancelling a program that's "caught in a loop." That can happen if a Perform Until statement is executing and the condition in the Until clause is never met. Unless you cancel out of the program, it will run indefinitely. The only time you can't use the Ctrl+Break key combination is if an Accept statement is being executed. Then, you have to use Ctrl+K.

Due to some bugs in Personal COBOL, cancelling a program doesn't always work the way you want it to. In some cases, you have to shut down Personal COBOL and restart it. In the worst cases, you have to shut down your entire system and restart it. Whenever possible, then, you should try to avoid logic errors that require cancelling a program.



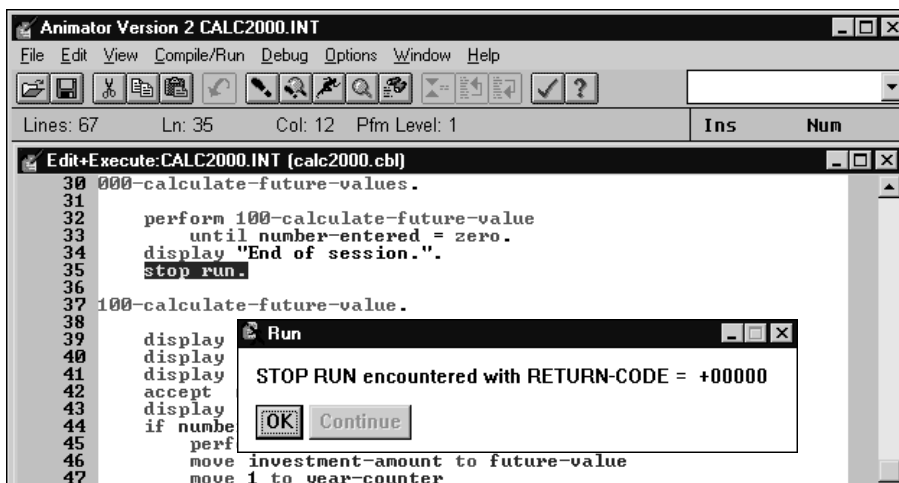
## The text window that's displayed when you run an interactive program



```

Animator V2 Text Window
-----
To end the program, enter 0.
To perform another calculation, enter 1.
1
-----
Enter investment amount (xxxxx).
100
Enter number of years (xx).
1
Enter yearly interest rate (xx.x).
10
Future value =      110.00
  
```

## The document window and the dialog box that's displayed when a program terminates normally



## How to start the execution of a program

- To run a program, open the compiled program (*int* extension) in execute mode. Then, click on the Run button.

## How to cancel the execution of a program

- You can press Ctrl+Break to cancel the execution of a program that's stuck in a loop or a program that displays a text window like the one shown above, unless an Accept statement is being executed. In that case, you can press Ctrl+K to cancel execution.

## How to restart the execution of a program

- When a program terminates normally, the Stop Run statement is highlighted, indicating that it was the last statement executed. To run the program again, use the Restart Application command in the Compile/Run menu to move the current execution point to the first executable statement. Then, click on the Run button.

## How to correct run-time errors

---

If a run-time error occurs during a test run, a dialog box like the one at the top of figure 2-12 is displayed. Then, when you click on the OK button in that box, you are returned to the Animator and the statement that caused the error is highlighted.

In the example in this figure, the Perform statement is highlighted. Because the message in the dialog box says that the problem is an illegal character in a numeric field, though, you can assume that the problem is with one of the variables in the Until clause of that statement. There, the values in the two variables are being compared numerically to see whether the first is greater than the second.

If you study the example, you can see that the statement before the highlighted one has an asterisk in column 7, so it's treated as a comment and ignored by the compiler. But this is the statement that sets the year-counter variable to 1. If this isn't done and a Value clause hasn't given it a starting value, the data will usually be invalid because it has the value of whatever was left in those storage positions by the last program.

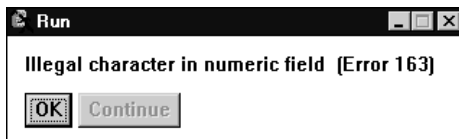
Invalid data in a numeric field is the most common cause of run-time errors in Personal COBOL. To help you figure out what caused the problem, you can display the current values of the variables. You can also set breakpoints and step through a program to see exactly what's happening as the program executes. You'll learn those debugging skills in the next topics.

When you know what the cause of the run-time error is, you can correct the error, recompile the program, and rerun it. When you recompile the program, it's automatically restarted so the first executable statement is highlighted. Then, you can just click on the Run button to execute it.

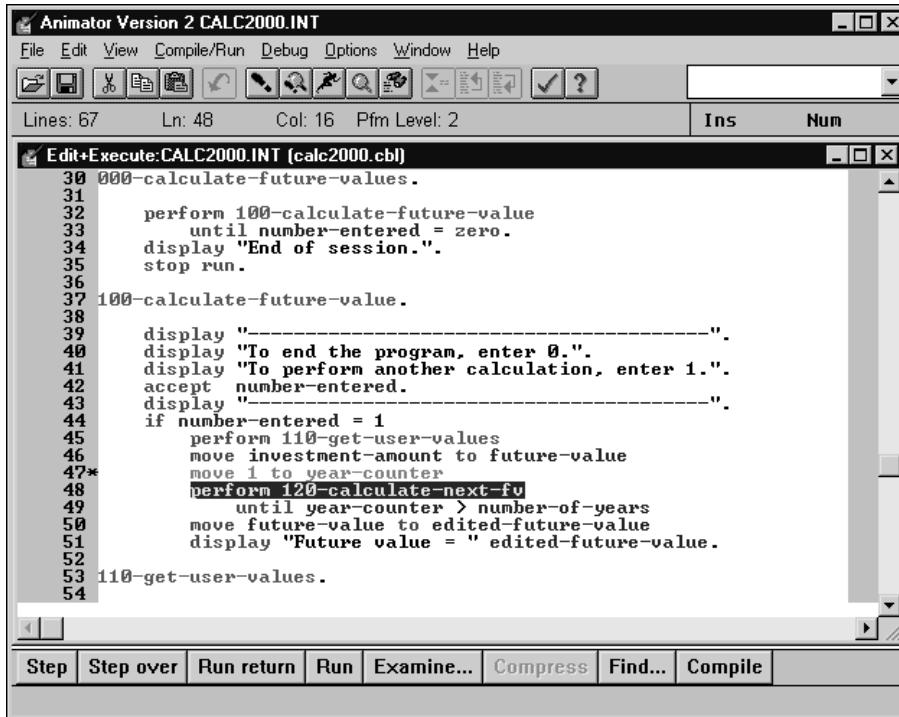
This figure also describes how Personal COBOL handles two conditions that usually cause run-time errors with other compilers. The first condition occurs when the result of an arithmetic operation is too large for the receiving field. In that case, Personal COBOL truncates the result. The second condition occurs when a Divide or Compute statement attempts to divide by zero. In that case, Personal COBOL treats the zero as a one.

Keep in mind that the results will be incorrect even though run-time errors won't occur for these conditions. That's why you need to check the results to make sure that neither one of these conditions led to an error. In contrast, if these conditions do cause run-time errors as they do on most other compilers, you don't have to worry that truncated or altered results will sneak into your program. Either way, though, you need to check the results to make sure they're accurate.

## A dialog box for a run-time error



## The Animator with the error statement highlighted



## Description

- When a run-time error occurs, the program is interrupted and a dialog box that contains an error message is displayed.
- When you click on the OK button in the dialog box, you are returned to the code in the Animator with the statement that caused the error highlighted. You then can correct the error, recompile the program, and test it again.
- The most common cause of a run-time error is invalid numeric data.

## How Personal COBOL handles two other types of run-time errors

- If an arithmetic operation has a result that is too large for the receiving field and the On Size Error clause hasn't been coded, the result is truncated instead of causing a run-time error.
- If a divide operation tries to divide by zero, the zero is treated as one instead of causing a run-time error.

Figure 2-12 How to correct run-time errors

## 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. This assumes that you've done exercise 2-1 so your default folder is set right and so the button bar is displayed.

### Open the sales tax program for editing and compile it

1. Start Personal COBOL, and close the window for the new file.
2. Choose the Open for Edit command in the File menu or on the toolbar, and open the file named calc1000 that's in the c:\cobol folder. If your defaults are set right, you shouldn't have to change the folder before you open the file.
3. Click on the Compile button. This should result in a clean compile.

### Open the program for execution and test it

4. Choose the Open for Execution command from the File menu, and open the file named calc1000.int. This is the intermediate version of the program that you just compiled. Note that you don't have to close the windows that are open for editing before you start this command.
5. Click on the Run button in the button bar to run the program in a text window.
6. For the first test run, enter zero and press the Enter key. A dialog box is then displayed that tells you that the program ended with a return code of zero. That means the program ended normally, so click on the OK button.
7. To run the program again, choose the Restart Application command from the Compile/Run menu. Then, click on the Run button again. 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 program

8. Close the sales tax program by clicking on the Close button in the upper right corner of the document window.

## 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. Open the program named calc100x for editing. Then, compile the program. When the Checking dialog box is displayed, click on the Zoom button to stop the compiler from displaying any more error messages.
2. Oops! There are several compile-time errors. These are the ones shown in figure 2-10. If you want more information about an error, right-click on the statement and choose the Error Help command.
3. Can you correct these errors on your own? You should be able to. At the least, try to correct them 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. 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

8. Open the calc100x.int program for execution, and run it. Oops again! The program ends without calculating 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.

### Close the program

10. Close the program.

## How to use the debugging features

---

When you test a program and a run-time error occurs or the output isn't what you expect it to be, it can be difficult to locate the source of the errors just by looking at the code. That's why Personal COBOL supplies you with debugging tools that save you time and frustration as you test your programs. The topics that follow present the best of these tools.

### How to display and modify the values of variables

---

When a program is interrupted by a run-time error, the Animator window is displayed in *break mode* and you can display the value of any variable by double-clicking on it. This opens a dialog box that shows the variable name and its value. In figure 2-13, for example, you can see that the value of the future-value variable at the time of the run-time error is 1000. You can also open a dialog box like this by clicking on the Examine button and entering the name of the variable you want to display in the resulting dialog box.

Another way to view the values of variables is to set up a *monitor/watch list* like the one shown in this figure. One way to do that is to click on the Add to List button in the dialog box for a single variable. In this example, two variables have been added to the Monitor/Watch List dialog box. In contrast to the dialog boxes for individual variables, this dialog box stays open from one execution of the program to another.

If you study the values in the monitor/watch list in this figure, you can see that the number-of-years variable has a valid value of 10, but year-counter has a value of:

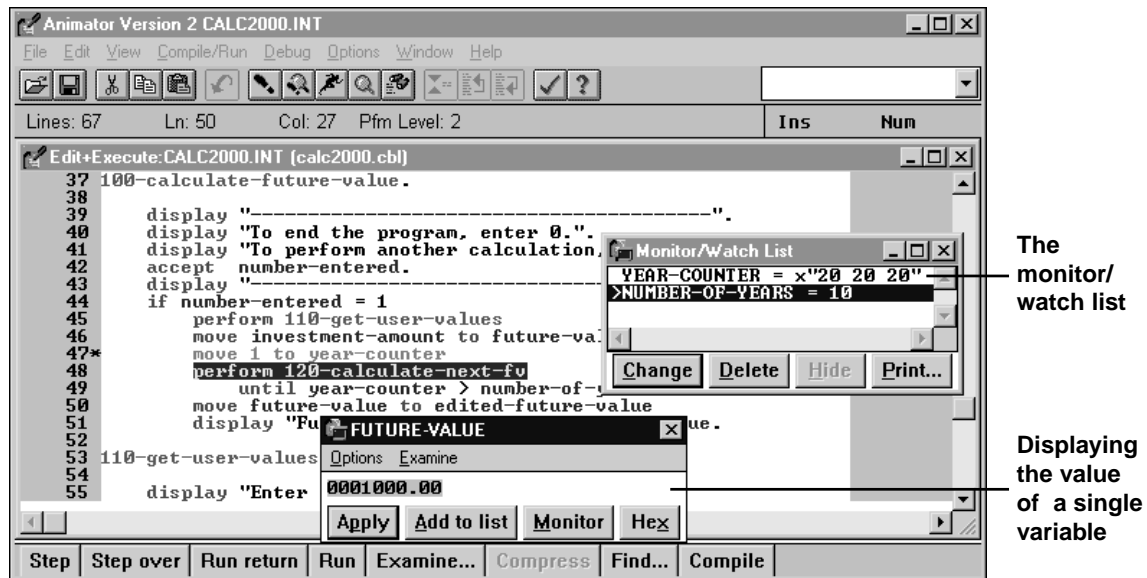
`x"20 20 20"`

Here, the *x* means that the value that follows is represented in *hexadecimal* (or *hex*) *code* with two hex digits for each byte of storage. If you know hex code, you know that hex 20 is the ASCII code for a blank, so the value in this numeric field is three blanks, which is an invalid numeric value.

In general, when you display the value of a numeric field, the Animator displays it in decimal if it is a valid numeric value. If it is invalid, though, the Animator displays it in hex. As a result, you don't need to know what the hex codes are to know when a field is invalid. Note, however, that you can display any characters in hex code by clicking on the Hex button in the display box. To learn more about hex codes, please refer to chapter 6.

If you want to change the value of a variable while a program is in break mode, you can do that from the dialog box that displays the variable's value. Just type over the value that's displayed and click on the Apply button. Or, if you're using a monitor/watch list, you can click on the Change button with the old value highlighted and then enter a new value. After you change the value to the one the one you want, you can click on the Run button to continue the test run with the new value.

## The Animator while the program is in break mode



## Two ways to display the value of any variable

- Double-click on the variable name. This opens a dialog box that displays the current value of that variable.
- Click on the Examine button. Then, enter the name of the variable whose value you want to display in the Examine dialog box, and click on OK.

## How to set up a monitor/watch list

- The easiest way to set up a *monitor/watch list* is to display the value of a variable and then click on the Add to List button. If the Monitor/Watch List dialog box isn't already displayed, it's opened and the variable is added to it. Otherwise, the variable is added to the existing list.

## How to change the value of a variable

- Display the value of the variable, then type the new value over the current value in the dialog box that's displayed and click on the Apply button.
- If the variable is displayed in the monitor/watch list, select the variable, click on the Change button, and enter the new value in the dialog box that follows.

## Why hexadecimal code is used for some values

- If a numeric field contains invalid data, the Animator displays it in *hexadecimal* (or *hex*) *code* as shown for the year-counter field above. This code is preceded by an *x*. Then, each pair of hex characters that follows represents the data for one byte of storage. One of the most common forms of invalid numeric data is hex 20, which represents one space. For more information about hex code, please refer to chapter 6.

Figure 2-13 How to display and modify the values of variables

## How to use breakpoints

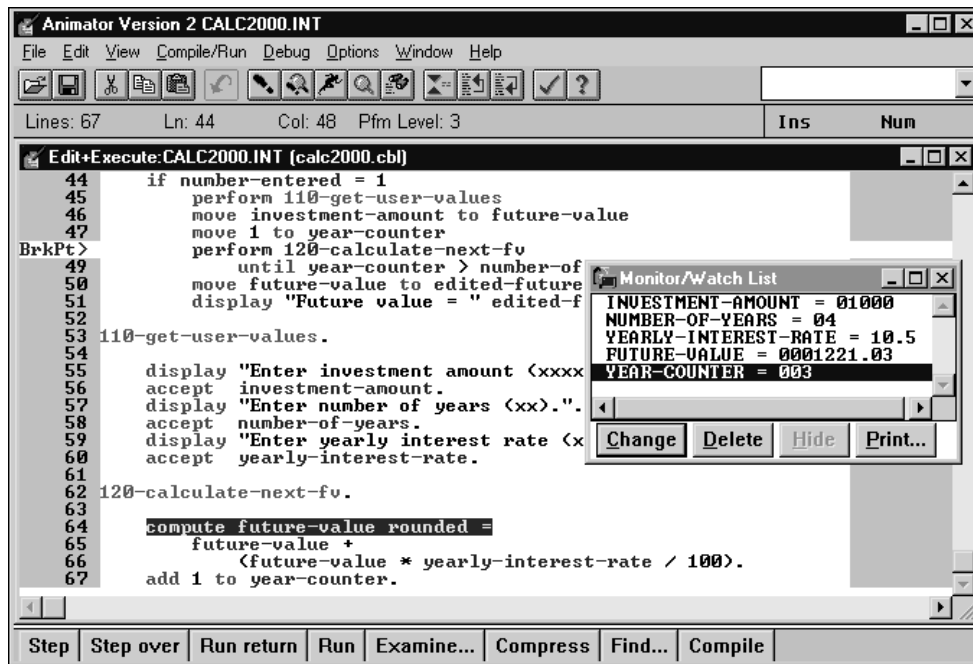
---

Another valuable debugging feature is the ability to set breakpoints. This is illustrated in figure 2-14. When you set a *breakpoint* on a statement, the program enters break mode each time it comes to that statement. In this figure, for example, a breakpoint has been set on the Perform Until statement, as you can see by the BrkPt> tag in the left margin.

While in break mode, you can display and change the values of variables. In this figure, the monitor/watch list is used to display the values of the five variables used by the program. This lets you analyze the way this program works so you can tell for sure whether it's working correctly. If necessary, you can set two or more breakpoints in a program so you can monitor the operation of a program at all of its critical points.



## The Animator when it reaches a breakpoint



### Description

- You can set a *breakpoint* on any executable COBOL statement or paragraph heading in the Procedure Division. Program execution stops when it reaches the statement you marked or the first statement in the paragraph you marked.
- When a program reaches a breakpoint, a dialog box is displayed. Then, you can click on Continue to continue program execution or on OK to display the document window in break mode so you can debug the program.
- In break mode, you can display and change the values of variables.

### How to set and remove breakpoints

- The easiest way to set or remove a breakpoint is to double-click in the margin to the left of a statement. Another way to do that is to right-click on the statement to display the shortcut menu and then select the Breakpoint or Unset command.
- To remove all breakpoints, choose the Clear all Breakpoints command from the Debug menu.

Figure 2-14 How to use breakpoints

## How to step through a program

---

From break mode or when you start a program, you can *step through the program* instead of running all the statements without intervention. Then, you can see how the values of the variables change as each statement or group of statements is executed. Figure 2-15 describes some of the techniques you can use to step through a program.

At the top of this figure, for example, you can see the result of clicking on the Step Over button after the program has reached the breakpoint on the Perform Until statement. This command causes the program to execute the performed procedure (in this case, procedure 120) without stopping. Then, it enters break mode again before the statement after the Perform Until statement is executed. In this case, the next statement is Move.

The simplest step technique is to step through the program one statement at a time. If, for example, you click on the Step button when the Move statement is highlighted, the Move statement is executed and the program enters break mode before the next statement is executed. This is illustrated by the examples in this figure. If you click on the Step button again, the program executes the next statement and enters break mode again. You can continue in this way until you're sure you understand how the program works. Whenever you want to run the rest of the program without stopping, you just click on the Run button.

Another step technique is to run the rest of the statements in the procedure that's currently executing without stopping. To do that, you use the Run Return command. You may want to do that if you enter a procedure that you already know works. Then, the program enters break mode again at the statement after the Perform statement that called the procedure.

The breakpoint and step features in combination with the features for displaying the values of variables can save you many hours of debugging time. In addition, these features are wonderful learning tools. By stepping through code that you don't understand and displaying the values of related variables, you can see exactly how a program works.

### The Animator after the Step Over command is executed from the breakpoint

```

37 100-calculate-future-value.
38
39     display "-----".
40     display "To end the program, enter 0.".
41     display "To perform another calculation, enter 1.".
42     accept number-entered.
43     display "-----".
44     if number-entered = 1
45         perform 110-get-user-values
46         move investment-amount to future-value
47         move 1 to year-counter
BrkPt>     perform 120-calculate-next-fv
49         until year-counter > number-of-years
50         move future-value to edited-future-value
51     display "Future value = " edited-future-value.
52

```

### The Animator after the Step command is executed from the screen above

```

37 100-calculate-future-value.
38
39     display "-----".
40     display "To end the program, enter 0.".
41     display "To perform another calculation, enter 1.".
42     accept number-entered.
43     display "-----".
44     if number-entered = 1
45         perform 110-get-user-values
46         move investment-amount to future-value
47         move 1 to year-counter
BrkPt>     perform 120-calculate-next-fv
49         until year-counter > number-of-years
50         move future-value to edited-future-value
51     display "Future value = " edited-future-value.
52

```

### How to step through a program

- To execute the current statement, click on the Step button. After the statement is executed, the program enters break mode at the next statement.
- To execute the statements in a performed procedure without stopping, click on the Step Over button. After the performed statements are executed, the program enters break mode at the next statement after the Perform statement.
- To execute the rest of the statements in the current procedure without stopping, click on the Run Return button. Then, the program enters break mode at the statement after the Perform statement that called the procedure.
- To execute the statements up to the statement that contains the cursor without stopping, choose the Run to Cursor command from the Compile/Run menu.
- To execute all the remaining statements in the program without stopping, click on the Run button.

Figure 2-15 How to step through a program

## A summary of the other debugging features

---

Figure 2-16 summarizes some of the other debugging features that are available with the Animator. Although you may never need to use any of these features, you may want to know what they are. If so, you can study this figure and the text that follows. Otherwise, you can skip to the next page.

In addition to the Step commands presented in figure 2-15, you can also use the Step All command. This command steps through all the statements in the program one at a time at the speed you specify. Note that the program doesn't enter break mode before it executes each statement. Instead, it executes at a reduced speed so you can see the values of the variables in the monitor/watch list as the program executes. You can also use the Step All dialog box that's displayed while the program is executing to change the execution speed or enter break mode.

The four Skip commands in this figure let you skip the execution of one or more statements. For example, you can use the Skip Statement command to skip the current statement, and you can use the Skip to Cursor command to skip all the statements between the current statement and the statement that contains the cursor.

The Set Advanced command lets you use some advanced features for setting breakpoints. For example, you can cause a program to enter break mode when the value of a variable changes or when a certain condition is met. These features can be useful for pinpointing the source of a logical error.

The Do Statement command lets you execute any valid COBOL statement while in break mode. After changing the value of a variable, for example, you may want to re-execute a statement that uses that variable. To do that, you just enter the statement into the dialog box that's displayed when you select the Do Statement command from the Debug menu.

You can use the last statement in this figure, Backtrack, to trace the execution of a program. To use this feature, you must turn backtracking on as described in this figure. Then, the Animator keeps a record of each statement that's executed, and you can use the Backtrack command to trace backwards through the program from the current statement. As you backtrack, the Animator simply highlights the previously executed statement.

### Some of the other debugging features you can use with the Animator

Menu	Command	Description
Compile/Run	Step All	Displays a dialog box that lets you step through all the statements in a program at a selected speed. You can use the Step All dialog box to select the speed, stop program execution, and continue execution without stepping.
	Skip Statement	Skips execution of the current statement. The current execution point moves to the next physical statement in the program.
	Skip Return	Skips execution of the remaining statements in the procedure currently being performed.
	Skip to Cursor	Moves the current execution point to the statement that contains the cursor, skipping the execution of any statements prior to that statement.
	Skip to Start	Moves the current execution point to the first executable statement of the program without initializing the data in working storage.
Debug	Set Advanced	Displays a dialog box that lets you set advanced breakpoints. These breakpoints can (1) cause program execution to stop when the value of a specified variable changes; (2) cause program execution to stop at the specified line when a condition is met; (3) cause program execution to stop at any line when a condition is met; (4) cause program execution to stop when a specified program is entered; and (5) cause a specified COBOL statement to be executed when the line containing the breakpoint is reached.
	Do Statement	Displays a dialog box that lets you execute any valid COBOL statement.
	Backtrack	Displays a dialog box that lets you trace the execution of the statements in a program. Before you can use this command, you must select the Backtrack On option from the dialog box that's displayed when you choose the Execute Options command from the Options menu.

### Additional help

- For more information on using these commands, see the online help for the Animator.

Figure 2-16 A summary of the other debugging features

## 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 a breakpoint, step through the program, and display the values of variables.

### Compile and test the program

1. Open the program named `calc2000` for editing. Then, compile the program. This should result in a clean compile.
2. Open the program for execution. Then, run the program. When the text window is displayed, enter a value of zero for this first test run. This should end the program.
3. Restart the program, and run it 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 you can test the minimum values easily, but 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 Personal COBOL 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 as shown in figure 2-14. Next, set up a Monitor/Watch List dialog box so it displays the values for the five variables shown in that figure. Now, you can watch these variables change as you step through the program.
6. Restart the program and run it. 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. When the breakpoint is reached, 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 you're interested in, click on the Run button to continue without stopping.
7. If you want to try that again, enter another set of variable values and step through the program when the breakpoint is reached. This should clear up any questions that you have about how this program works. When you're satisfied that you understand it completely, end the program.

### Close the program and end the Animator

8. Close the program. Then, exit from the Animator by clicking on the close button in the upper right corner of its window.

## Perspective

---

Now that you know how to compile and test a program using Micro Focus Personal COBOL, you should be able write simple interactive programs of your own. Then, in the next chapter, you will learn how to write programs that read files and prepare reports.

As you work with Personal COBOL, you should know that many mainframe programmers do most of their development work on PCs using the Animator for Micro Focus Workbench or Micro Focus Mainframe Express. After they compile and test their programs on PCs, they use the Workbench or Mainframe Express tools to upload their programs and data to the mainframe for final testing. Because Workbench and Mainframe Express on a PC provide a friendlier environment and better debugging tools than a mainframe, developing programs in this way can be much more efficient than developing them directly on the mainframe.

## Summary

---

- You use the Animator to develop programs for Micro Focus Personal COBOL on a PC. This is the same type of Animator that is used with the Micro Focus products for developing UNIX and mainframe programs.
- To enter and compile the initial code for a program, you open the program in *edit mode*. When the program compiles without errors, called a *clean compile*, you open the program in *execute mode* so you can *test* and *debug* it.
- A *compile-time error* occurs when the compiler can't convert a statement into machine language because its syntax is incorrect. A *run-time error* occurs when the Animator is unable to execute a statement. And a *logical error* occurs when the program runs successfully, but the results aren't what you expected.
- To help you debug a program, the Animator provides debugging tools. These tools let you set *breakpoints*, display and change the values of variables while the program is in *break mode*, and *step through a program* one statement at a time.

## Terms

---

Integrated Development Environment (IDE)	diagnostics
document window	linkage editor
information line	link edit
status line	link
key status area	COBOL profile
edit mode	button bar
execute mode	tagged line
source code	compressing lines
source program	expanding lines
source file	tag
intermediate file	clean compile
testing	text window
debugging	normal program termination
compile-time error	return code
run-time error	break mode
logical error	monitor/watch list
production program	hexadecimal code
executable	hex code
object module	breakpoint
object program	stepping through a program

## Objectives

---

- Given the specifications for a simple interactive program like the ones in chapter 1, use Micro Focus Personal COBOL to enter, compile, test, and debug the program.
- Describe the differences between testing and debugging.
- Describe the differences between compile-time, run-time, and logical errors.
- Explain why you should start your new programs from old programs.
- Explain how the debugging features can help you debug a program.