GNU COBOL 01

Your First COBOL Program

Let's start with some basics. You might find that some of this revisits material you know, particularly if you already are a programmer. COBOL sets somewhat rigid requirements for the layout, contents, and order of a program. You must be familiar with these in order to program in COBOL.

Today's lesson covers the following topics:

- What is a computer?
- What is a program?
- What is a programming language?
- What is COBOL?
- The "Hello World" program.
- The parts of a COBOL program.
- The elements of code layout.
- Commenting a COBOL program.
- What is a shell program?

Today, you'll write your first program and learn the basic parts of a COBOL program.

What Is a Computer?

A computer is a machine that can add, subtract, multiply, divide, and perform other mathematical and logical functions on numbers. A computer recognizes some numbers as numbers, and it also can recognize words by translating the words into numbers. At one time, you could talk to a computer only in numbers, but modern computers can accept words on-screen, translate them into command numbers, and then execute them.

The heart of a computer is the machine that does the addition, subtraction, multiplication, and division, and also moves data from one location to another. This is called the *central processing unit* (CPU), because it processes the data. In personal computing, the computer itself frequently is named after the CPU. A computer that uses a CPU called the 80286 is sometimes called a 286. Machines that use the 80386 usually are sold as 386

machines, and those that use the 80486 CPU are called 486 machines. The arrival of 80586 chips heralded a name change. These are called Pentiums. Will 80686 chips be called Hexiums?

To be useful, the computer also must have a way to be given the numbers to process (the *input*) and a method of presenting results to the user (the *output*). The input to a computer usually is entered with a keyboard. Other input devices include bar code readers, optical character readers (OCR), scanners, and mice. The output from a computer usually is displayed on a monitor (or screen) or printed on paper or can be written out to a file on a disk.

To perform large calculations, the computer needs some place to store temporary or intermediate results. This temporary storage for the computer frequently is called *main memory*, *primary memory*, or *primary storage* and usually is stored inside the main box of the computer. Think of the memory storage area inside the computer as a giant scratch pad for programs. Early personal computers had as little as 4 kilobytes (4KB) of memory. (A *kilobyte* is a bit more than 1,000 bytes; a byte can store one character of data, such as A, X, or @.) The personal computer boom brought the price of memory down to the point that computers now commonly sell with a starting memory of 4 or 8 megabytes (8MB) and can be upgraded to 32MB or more. (A *megabyte* is about a million bytes.)

To save results so that they can be reused, the computer needs some place to store information on a permanent or long-term basis. The problem with main memory is that it needs to have power all the time. When a computer is switched off, the contents of main memory are lost. The computer needs something that will retain information even when power is switched off. This is the task of secondary storage. Secondary storage is permanent and continues to function even after power is gone. Secondary storage comes most commonly in the form of diskettes (or floppies), hard disks, and tapes. Data is recorded on diskettes, hard drives, and tapes in a manner similar to the way that music is stored on a music tape cassette. A CD-ROM (compact disc-read-only memory) is another type of secondary storage. It is most commonly used as a permanent storage device and contains data that cannot be modified; such as the text of encyclopedias and dictionaries, or a complete listing of businesses in a country. There are more expensive devices available that can write to a CD-ROM that can be used for the initial storage of the dictionary, encyclopedia, or whatever.

The central processing unit requires that any program to be run and any data to be processed must be in main memory. Whenever you run a program, it is loaded from the secondary storage device (disk) into main or primary memory before it is executed. Whenever you work on data, such as editing a file, the file is first loaded into main memory

from the disk drive (secondary storage). The editing is done directly in main memory and then must be saved back to disk. The central processing unit can neither execute a program directly from disk nor manipulate data directly on the disk.

Figures 1.1 through 1.4 illustrate the relationship between the CPU, main memory, and disk storage.

Figure 1.1.
When the user starts a word processing program, the CPU loads the program from a hard disk into memory.

In Figure 1.1, the user has typed a command or clicked a button to start a word processing program. The CPU locates the program on the disk, loads it into main memory, and then begins executing the instructions in the word processing program.

In Figure 1.2, the word processing program is running and the user types a command or clicks a button to load a document for editing. The CPU locates the document on the disk and loads it, ready to be edited, into main memory.

Figure 1.2.
When the user asks for a document to be edited, the CPU loads the document from a hard disk into memory.

In Figure 1.3, the word processing program is running and the user types a letter "A" to be added to the document. The CPU collects the letter typed at the keyboard and places it in the document in memory.

Figure 1.3.
When the user types the letter "A," the CPU collects the character from the keyboard and inserts it into main memory.

In Figure 1.4, the user has requested that the document be saved. The CPU collects all the memory containing the document and writes it to the hard disk.

Figure 1.4.
When the user asks for the document to be saved, it is pulled from memory and written back to the hard disk.

The CPU, input, output, main memory, and secondary storage all work together to form a computer.

What Is a Program?

A computer is an incredibly stupid device. It doesn't do anything unless and until it is told to do so.

When a computer is first switched on, the CPU starts looking through main memory for an instruction. You can think of it as being in a perpetual state of readiness.

The computer is designed so that a small portion of main memory is permanent; it retains its contents even when the power is switched off. This permanent memory is placed at the location where the CPU begins searching for its first instruction after powering up. Consequently, it finds this permanent instruction immediately. This permanent area of memory contains a sequence of instructions that the computer executes on power-up. The instructions look something like this:

- Test the monitor.
- Test the keyboard.
- Test all of the main memory and display the results on the monitor.
- Test any other devices that need to be tested, including disk drives.
- Load the *operating system* program from secondary storage (the disk) into main memory. The operating system is a master program that controls a computer's basic functions and allows other programs to access the computer's resources, such as the disk drives, printer, keyboard, and screen. (In practice, this step is a little more complicated, but the principle is correct.) For an MS-DOS-compatible computer, the operating system is MS-DOS (Microsoft Disk Operating System).
- Jump to the first, previously loaded instruction at the beginning of the MS-DOS operating system.

From this point on, the CPU is executing instructions within the MS-DOS operating system.

If you could read the first few instructions of the MS-DOS operating system in English, they might look something like what you see in Table 1.1.

Table 1.1. The first few instructions of the MS-DOS operating system as they might appear in English.

Instruction Number	Instruction	Comment

001	Display the prompt	Put the > prompt on the screen.
002	Wait for a keypress	
003	Was a key pressed?	
004	If not, GO TO 002	If no key was pressed, go back and try again.
005	Get the key value	
006	Save the key value	Store the value in main memory.
007	Display the key value	
008	Was the value (ENTER)?	Determine whether the Enter key was pressed, signaling the end of command input.
009	If not, GO TO 002	The user is still typing, so keep getting keypresses.
010	GO TO do the command	The user pressed Enter, so jump to the instruction that will try to execute the command. This part of the program is not shown.

The set of instructions is written in English to represent the steps of a program. The program is executed so quickly that you see no visible delay between typing the key and seeing it appear on-screen, even though the action of saving the key value occurs between the keystroke and the display.

What Is a Programming Language?

The CPU expects instructions to arrive as numeric codes. These numeric codes are not easily read by human beings. A *programming language* is a set of English-like instructions

that includes a set of rules (syntax) for putting the instructions together to create commands.

A translator changes the English-like commands into numeric codes that the computer can understand. The most common type of translator is a compiler. The *compiler* is a program that reads the English-like commands in a file and then creates another file containing computer-readable numeric codes or commands.

In the previous example, the CPU cannot understand the English-like instruction WAS A KEY PRESSED?, but a programming language might accept this as a valid command and translate it into codes that the CPU can recognize.

The term *program* is used loosely to refer to the actual application that is executed by the CPU, as well as the file of English-like commands originally written by the programmer before it was translated into the program that the CPU executes.

In strict terms, the English-like commands in a file are called *source code*, and the translated numeric codes placed in the output file are a runable program called *executable code*. The computer cannot directly execute source code as if it were a program. However, even experienced programmers will say, "I wrote a program to calculate the month-end balance." What they really mean is, "I wrote a source code file containing English-like commands that, when compiled, will produce an executable program file that, when run, will calculate the month-end balance." It is definitely easier to say, "I wrote a program."

What Is COBOL?

COBOL is a programming language especially aimed at solving business problems. You will see as you work through this book that COBOL solves a lot more than just business problems and can be used as a solution to many data processing problems.

New Term: *COBOL* is an acronym for Common Business Oriented Language.

NOTE: COBOL was developed by the Conference on Data Systems Languages (CODASYL), convened in 1959 by the Department of Defense. COBOL compilers became available in 1960, but they were not standardized. The American National Standards Institute (ANSI) standardized a version of COBOL in 1968. The language was revised and updated by ANSI in 1974 and again in 1985. These standards sometimes are called COBOL or COBOL-68, COBOL-74, or COBOL-85. Most compilers are now COBOL-85 standard, but there still are a few COBOL-74 versions out there. This book is written against the COBOL-85 standard, but you will have no trouble using a COBOL-74 compiler for any of the examples.

Because the year 2000 problem might be one of the reasons that you are studying this book, it is worth noting that legacy code could be written in any of the earlier COBOL standards. If you are taking this course to bring yourself up to speed for an update effort, you will need to learn the quirks and differences of the particular COBOL version that you will be working on. This book covers the core of COBOL-85 and will give you about 95% of any version of COBOL that you might work with. Throughout the book there are tips on differences that you might find in other versions of COBOL.

The future of COBOL is fairly bright. A new COBOL standard is being drafted even as I write this. This standard is intended to take COBOL into the future and certainly well beyond the year 2000. The amount of money that is being invested in correcting date problems in existing COBOL code is a sure indicator that no one is planning to dispose of COBOL in the near future.

Approximately 90% of all COBOL code runs in a character-based environment, which means that most COBOL screens are 80 columns wide by 24 or 25 characters high and do not contain graphics. Although there are versions of COBOL on the market that act in a Windows-like environment, this book is not intended to be a course in COBOL for Windows. This means that throughout the book, you will be running or executing your programs on an MS-DOS computer or in an MS-DOS window that has been opened on a Windows computer. The MS-DOS window is an 80 x 24 character window and represents the kind of display that you might see when coding COBOL on a mainframe or minicomputer.

The "Hello World" Program

The "Hello world" program has become almost trite. Writing a program that prints "Hello world" on-screen usually is the first program you learn in any language. Listing 1.1 is a basic COBOL program that will display "Hello world". The format of a COBOL program is covered in the following sections.

TYPE: Listing 1.1. "Hello world" in COBOL.

```
000100 IDENTIFICATION DIVISION.

000200 PROGRAM-ID. HELLO.

000300 ENVIRONMENT DIVISION.

000400 DATA DIVISION.

000500 PROCEDURE DIVISION.
```

```
000600

000700 PROGRAM-BEGIN.

000800     DISPLAY "Hello world".

000900

001000 PROGRAM-DONE.

001100     STOP RUN.
```

ANALYSIS: A COBOL program always contains four divisions. These four divisions always have the same names:

- IDENTIFICATION DIVISION
- ENVIRONMENT DIVISION
- DATA DIVISION
- PROCEDURE DIVISION

In Listing 1.1, lines 000100 and 000200 are the IDENTIFICATION DIVISION. This division is used to identify basic information about the program. In this example, the IDENTIFICATION DIVISION contains only the PROGRAM-ID, HELLO.

Line 000300 is the ENVIRONMENT DIVISION, which is used to identify the environment in which the program is running. Remember that COBOL is intended to run on many different types of machines, and this section is used to handle the differences between various computers. In this case, the program has no specialized machine requirements, so the ENVIRONMENT DIVISION is empty.

Line 000400 is the DATA DIVISION, which will contain any data that the program operates on. This program has no data, so the DATA DIVISION is empty.

Lines 000500 through 001100 are the PROCEDURE DIVISION. This is the meat of the program--the part that does the work intended by the programmer. The PROCEDURE DIVISION contains two paragraphs at line 000700 (PROGRAM-BEGIN) and line 001000 (PROGRAM-DONE). The term *paragraph* has a special definition in COBOL that will be covered a bit later in today's lesson. All the actual work in this program is done by line 000800.

If you have not yet installed your software, review Appendix C, "Editing, Compiling, and Linking," and the installation instructions in your software documentation; then complete the installation procedure. Make sure that you end up with a C> prompt under MS-DOS,

and create your working directories as described in Appendix C. Change to your working directory before you begin editing hello.cbl.

It is extremely important that you type (edit), compile, and, if necessary, link this program to produce a running program. If you can't get this program to run, the remainder of this book will be an exercise in theory with no practical application. If you are using some other editor and COBOL combination, consult your local documentation or someone who is experienced with the system. Don't pass over this exercise without getting it to run. It is the simplest program to edit, compile, link, and run.

Before you start typing, you should note a couple of things. The first six character positions of each line are used for line numbering. The seventh character position is always blank. The commands at lines 000800 and 001100 begin at position 12. Everything else starts at position 8.

The editor that comes with Micro Focus Personal COBOL actually skips positions 1 through 7 and leaves the cursor positioned at column 8. If you are using this editor, use the left arrow key to move to column 1 to type the line numbers and the lines. Columns 1 through 7 are traditionally used for line numbering in COBOL. Most larger systems will use line numbers. The practice of skipping or not requiring line numbers is common with PC COBOL compilers and very modern compilers, but at the moment it is the exception rather than the rule. This book uses line numbers in keeping with the practice of most larger systems, and to provide a convenient way to refer to lines of code when a program is being analyzed in the text.

It's preferable to use spaces instead of the Tab key while editing, and you should break the habit of using the Tab key. Many personal computer-based COBOL compilers can handle tab characters, but the language was not designed originally to allow tabs in the source code file, and their presence can cause trouble on larger machines.

If you are using Micro Focus Personal COBOL, start your editor by typing the following line and pressing Enter:

pcobol hello.cbl

The extension .cbl is the default extension for many MS-DOS based COBOL compilers. On the VAX minicomputer manufactured by Digital Equipment Corporation, the default extension is .COB, and you would use a command such as:

EDIT HELLO.COB

DO/DON'T:
DO type each line exactly as it appears in Listing 1.1. Review your work carefully.

DON'T make typing errors. Some of the typing errors that can cause serious problems when you are compiling include misspelling the name of a DIVISION (for example, INDENTIFICATION DIVISION), adding an unnecessary hyphen (for example, DATA-DIVISION), or omitting any of the periods. Note that everything ends with a period (.); in fact, line 000200 has two periods in it.

New Term: Checking source code created with an editor for typographical and other errors before compiling it is referred to as *desk-checking*.

New Term: COBOL compilers are prone to produce cascading errors. A *cascading error* is one or more errors (sometimes hundreds) generated by the compiler when the problem really is one simple error earlier in the program. Code containing a missing period can produce a stream of apparent errors on some compilers that can be traced back in the program to that single error. The missing period itself might not even be mentioned as an error by the compiler, but it causes later problems that do show up.

When you have completed your check, close and save the file. Under Micro Focus Personal COBOL, hold down the Alt key while pressing F4, and then release both keys and press Enter.

Now you are ready to compile your program. Under Micro Focus Personal COBOL, press F2 until check is displayed on the status line (the fifth line from the bottom of the screen) and press Enter.

New Term: Under Micro Focus Personal COBOL, the process of compiling is called *checking*. The Micro Focus Personal COBOL compiler actually is called the *checker*.

The program should compile with no errors or warnings. If there are errors or warnings, re-edit your source code file, hello.cbl, desk-check it (compare it to the example in the book), and locate the error. The compiler might tell you the line number at which the error occurred, and you can go straight to that line.

COBOL compilers are dependent on correct punctuation, so bad punctuation can sometimes confuse the compiler. If the compiler says you have an error on a particular line, but you can't seem to find it, look one or two lines earlier to check whether you left out a period or started a line in an incorrect column. You also might want to check Appendix D, "Handling Compiler Errors," to help you track down errors.

After the program compiles cleanly, you are ready to run it. Exit from the COBOL development environment, if you are in one. For Micro Focus Personal COBOL, press Esc. The bottom row of the screen will display the message "Exit from Personal COBOL." Press the Y key to exit.

To run the program, type cls and press Enter to clear the screen. Then type the following and press Enter:

pcobrun hello

The program runs and displays your "Hello World" text, along with some Micro Focus copyright information, as shown in the output that follows.

OUTPUT:

Personal COBOL version 2.0 from Micro Focus

PCOBRUN V2.0.02 Copyright (C) 1983-1993 Micro Focus Ltd.

Hello world

C:>

Congratulations, you've completed your first program. If you are not excited about this, try exclaiming "Wow! My first COBOL program!" a couple of times.

The output display is approximately what you should see on your screen under Microfocus Personal COBOL. Other versions of COBOL will produce different display arrangements and may or may not include copyright notices. The key point is that the message Hello world will be displayed on-screen.

If you did not get the expected results, do not despair. Go back to the beginning of this section and review all the work. Particularly review Appendix C and the installation instructions in your software documentation, and run the tests to make sure that your editor is installed correctly. Check all the spelling in the program and run it through the compiler (checker) again until there are no error messages.

The Parts of a COBOL Program

Recall that a COBOL program is made up of four mandatory divisions. They always appear in the program in the order shown in Listing 1.2.

TYPE: Listing 1.2. COBOL's four divisions.

```
000100 IDENTIFICATION DIVISION.
```

000200 ENVIRONMENT DIVISION.

000300 DATA DIVISION.

000400 PROCEDURE DIVISION.

The IDENTIFICATION DIVISION marks the beginning of a COBOL program. The name of the program, which you assign, will be entered as a statement in the IDENTIFICATION DIVISION (more on this in a moment).

The ENVIRONMENT DIVISION contains statements or commands to describe the physical environment in which the program is running. The main use of the ENVIRONMENT DIVISION is to describe the physical structure of files that will be used in the program. You won't be working with files in these early lessons, so for now this DIVISION will be little used.

The DATA DIVISION contains statements describing the data used by the program. The DATA DIVISION and the PROCEDURE DIVISION are the most important divisions in a COBOL program; they do 95 percent of the work. You will start working in the DATA DIVISION in Day 2, "Using Variables and Constants."

The PROCEDURE DIVISION contains the COBOL statements that the program will execute after the program starts running. The PROCEDURE DIVISION is the real workhorse of a COBOL program. Without a PROCEDURE DIVISION, you wouldn't have a program, because all the other divisions are used to create the environment and data that are used by the PROCEDURE DIVISION to actually do something.

You already have seen hello.cbl (in Listing 1.1). It contains no data, no environment, and only one significant statement, which is in the PROCEDURE DIVISION. However, without the DISPLAY "Hello" command, the program would do nothing at all.

Each DIVISION in a COBOL program is broken down into smaller units, like an outline. Briefly, a DIVISION can contain SECTIONs, a SECTION can contain paragraphs, and paragraphs can contain sentences. For the moment, you can ignore SECTIONs, which are introduced in Day 2. Think of a COBOL program as DIVISIONs containing paragraphs containing sentences.

The requirements for the contents of each different DIVISION can vary, but most compilers require that only two things be present in a COBOL program--other than the four divisions--in order to compile it:

- PROGRAM-ID
- STOP RUN

The PROGRAM-ID is a paragraph that must appear in the IDENTIFICATION DIVISION and is used to give the program a name.

There must also be one paragraph in the PROCEDURE DIVISION that contains the STOP RUN statement.

Listing 1.3 is an example of the smallest possible COBOL program that will compile and run on any COBOL compiler. It contains the PROGRAM-ID paragraph and only one paragraph in the PROCEDURE DIVISION.

The paragraph PROGRAM-DONE contains only one sentence, STOP RUN. This sentence causes the program to stop running when the sentence is executed. Most versions of COBOL require this explicit command as a way of identifying the point in the program where the program terminates.

TYPE: Listing 1.3. minimum.cbl, the irreducible minimum COBOL program.

```
000100 IDENTIFICATION DIVISION.

000200 PROGRAM-ID. MINIMUM.

000300 ENVIRONMENT DIVISION.

000400 DATA DIVISION.

000500 PROCEDURE DIVISION.

000600

000700 PROGRAM-DONE.

000800     STOP RUN.
```

OUTPUT:

Nothing!

ANALYSIS: Clearly, minimum.cbl does even less than hello.cbl. In fact, minimum.cbl does nothing except stop running as soon as it starts. Its only function is to illustrate the minimum syntax that the COBOL compiler will accept.

Of all the errors that you can make in typing a COBOL program, an incorrect DIVISION name is one of the hardest errors to locate. In one compiler that I tested, misspelling the name of the IDENTIFICATION DIVISION as INDENTIFICATION DIVISION caused the compiler to report that the PROCEDURE DIVISION was missing. This is a difficult error to

spot because the real problem was three divisions away, and everything about the PROCEDURE DIVISION was fine. It is important that the DIVISIONs be typed correctly.

Listing 1.4 is more useful than minimum.cbl. You will recognize some similarities to hello.cbl, but I have divided a couple of the lines to illustrate a few more things about COBOL.

TYPE: Listing 1.4. Three levels of COBOL grammar.

```
000100 IDENTIFICATION DIVISION.

000200 PROGRAM-ID. SENTNCES.

000300 ENVIRONMENT DIVISION.

000400 DATA DIVISION.

000500 PROCEDURE DIVISION.

000600

000700 PROGRAM-BEGIN.

000800     DISPLAY "This program contains four DIVISIONS,".

000900     DISPLAY "three PARAGRAPHS".

001000     DISPLAY "and four SENTENCES".

001100 PROGRAM-DONE.

001200     STOP RUN.
```

OUTPUT:

```
C>pcobrun comment

Personal COBOL version 2.0 from Micro Focus

PCOBRUN V2.0.02  Copyright (C) 1983-1993 Micro Focus Ltd.

This program contains four DIVISIONS,

three PARAGRAPHS

and four SENTENCES
```

ANALYSIS: Strictly speaking, the PROGRAM-ID, SENTNCES, is a sentence, but it has such a specialized role in a COBOL program (identifying the program) that it is not usually

considered to be a sentence. The program is named sentnces.cbl (with the word sentnces deliberately shortened) because some operating systems (especially MS-DOS) limit filenames to eight characters plus an extension, and many compilers limit program names to eight characters.

DO/DON'T:
DO match the filename and program name; for example, sentnces.cbl is the file name and SENTNCES is the PROGRAM-ID.

DON'T add confusion by using a PROGRAM-ID that is different from the filename.

I will stick with the use of eight or fewer characters for the names of programs and files throughout the text.

The sentnces.cbl program contains all four DIVISIONs, three paragraphs (PROGRAM-ID, PROGRAM-BEGIN, and PROGRAM-DONE), and four sentences (the three DISPLAY statements in PROGRAM-BEGIN and STOP RUN at line 001200).

The paragraph name, PROGRAM-ID, is a required paragraph name and must be typed exactly as PROGRAM-ID. The paragraph names PROGRAM-BEGIN and PROGRAM-DONE are names I assigned when I wrote the program. Any of the paragraphs in the PROCEDURE DIVISION are given names you assign. The two paragraphs could have been named DISPLAY-THE-INFORMATION and PROGRAM-ENDS-HERE.

All the special words in COBOL (such as PROGRAM-ID, DATA, DIVISION, STOP, and RUN), as well as the paragraph names and program name (such as SENTNCES, PROGRAM-BEGIN, and PROGRAM-DONE), are created using the uppercase letters of the alphabet A through Z, the digits 0 through 9, and the hyphen (-). The designers of COBOL chose to allow a hyphen as a way of improving the readability of COBOL words. PROGRAM-BEGIN is easier to read than PROGRAMBEGIN.

The designers of COBOL also allowed for blank lines, such as line 000600 in Listing 1.4. Blank lines mean nothing in COBOL and can be used to spread things out to make them more readable.

You should type, compile (and, if necessary, link), and run Listing 1.4. See Appendix C for details. You might need to review this appendix a couple of times before you are completely comfortable with each of the steps involved in editing, compiling, and running.

Listing 1.4 illustrates the line-by-line organization of a COBOL program. There also is a left-to-right organization that determines what can be placed in certain columns.

A COBOL source code file has five areas, extending from left to right across the page. The first six characters or columns of a line are called the *sequence number area*. This area is not processed by the compiler, or if it is processed, it provides you only with warnings that numbers are out of sequence (if they are).

Character position 7 is called the *indicator area*. This seventh position is usually blank. If an asterisk is placed in this column, everything else on that line is ignored by the compiler. This is used as a method to include comments in your source code file.

The four character positions 8 through 11 are called *Area A*. DIVISIONs and paragraphs (and SECTIONs) must start in Area A. It is good coding practice to start DIVISIONs, SECTIONs, and paragraph names at column 8 rather than some random place in Area A.

Character positions 12 through 72 are called *Area B*. Sentences must start and end within Area B. It is good coding practice to start sentences at column 12 rather than some random place in Area B.

COBOL was designed as an 80-column language, but there is no formal definition of character positions 73 through 80. This is called the *identification area* (which has nothing to do with the IDENTIFICATION DIVISION).

The identification area is left to the designer of the COBOL compiler to use as needed. COBOL editors on large computers usually allow you to define an eight-character modification code that is inserted into the identification area whenever a line is changed or a new line is added. If you add lines to an existing program or change existing lines, it could be useful to know which lines were changed. Modification codes can be used to track down where a particular change was made. Modification codes are especially useful in companies where many programmers can work on many files. It helps keep track of changes, when they occurred, and who made them.

Some special COBOL editors place a modification code automatically in positions 73 through 80. This method of marking lines as modified usually depends on a special editor set up for COBOL that inserts these codes automatically. You probably will never see modification codes using COBOL on a PC.

Listing 1.5 is comment.cbl, which really is sentnces.cbl with a comment included and some lines that have been tagged with modification codes. I have deliberately left some of the sequence numbers out, or put them in incorrect order. The compiler will compile this without an error, but it might generate a warning that lines are out of order or sequence

numbers are not consecutive. The compiler doesn't really care what is in the first six positions, but it might provide some warning information. Because of the width limits in a book, the example modification codes in Listing 1.5 do not actually start in column 73 as they must in a real COBOL example.

TYPE: Listing 1.5. The areas of a COBOL program.

```
Indicator area Area A Area B Identification area

000100 IDENTIFICATION DIVISION.

000200 PROGRAM-ID. COMMENT.

000300 ENVIRONMENT DIVISION.

    DATA DIVISION.                          MB072197

000500 PROCEDURE DIVISION.

000600

000700* This is a comment.                   MB072197

000800* This paragraph displays information about the program.    MB072197

000900 PROGRAM-BEGIN.

003700    DISPLAY "This program contains four DIVISIONS,".      MB072197

003800    DISPLAY "three PARAGRAPHS".                  MB072197

001000    DISPLAY "and four SENTENCES".

001100 PROGRAM-DONE.

001200    STOP RUN.
```

Note that the output is the same as sentnces.cbl:

OUTPUT:

```
C>pcobrun comment

Personal COBOL version 2.0 from Micro Focus

PCOBRUN V2.0.02  Copyright (C) 1983-1993 Micro Focus Ltd.

This program contains four DIVISIONS,

three PARAGRAPHS
```

and four SENTENCES

C>

ANALYSIS: As a historical note, the very first COBOL programs were written using punch cards. Each card carried one line of code that had been carefully entered using a keypunch machine (a kind of typewriter that punches holes in cards). The stack of punched cards was carried to the computer and fed into it using a card reader. An "out of sequence" warning was used to let you know that you probably had dropped the punch card deck and hadn't put them back together in the correct sequence. Compiler error messages also referred to line numbers, and locating an error was difficult without line numbers on the cards. PC COBOL compilers rarely give warnings about sequence.

Lines 000700 and 000800 contain an asterisk in column 7, the indicator area. Everything beyond the asterisk is ignored and can be used as a comment, as in the example.

DIVISIONs and paragraphs start in Area A but can extend into Area B.

Sentences begin and end in Area B. In Listing 1.5, sentences appear at lines 003700, 003800, 001000, and 001200.

What Is a Shell Program?

Because COBOL has a certain minimum amount of code that is required for all programs, it is a good practice to maintain a COBOL program that contains the minimum requirements.

New Term: A COBOL program that contains the minimum requirements usually is called a *shell program*, a *skeleton program*, or a *boilerplate program*.

If you copy this shell program to a new file before you start editing the new program, you can save yourself extra typing. Listing 1.6, cobshl01.cbl, is your first version of the COBOL shell program. As you progress through the days, you'll gradually add more and more pieces to the shell; eventually, you'll have a complete shell to use in all projects.

TYPE: Listing 1.6. Your first version of a COBOL shell.

000100 IDENTIFICATION DIVISION.

000200 PROGRAM-ID. COBSHL01.

000300 ENVIRONMENT DIVISION.

000400 DATA DIVISION.

```
000500 PROCEDURE DIVISION.

000600 PROGRAM-BEGIN.

000700

000800 PROGRAM-DONE.

000900     STOP RUN.
```

Type Listing 1.6 and compile it to ensure that everything in it is correct. You will use this shell, and versions of it, many times before you are through with these lessons.

As you will see in the next few days, PC-based COBOL compilers are much less stringent about following the rules described in today's lesson. However, COBOL is intended to be a portable language that allows code to be moved to other computers. Another computer probably will be using another COBOL compiler that might require compliance to all the strict rules of COBOL. When you start breaking the rules, you limit your code to a compiler that can handle the loose syntax, and this might make it extremely difficult to move your code to another machine or compiler.

Summary

Today's lesson introduced you to some computer and programming basics, including the following:

- A computer processes numbers and is made up of a central processing unit, input devices, output devices, main memory, and secondary storage.
- A computer can't do anything without a program.
- A program is a series of instructions that the central processing unit executes to process data, usually using some input data and providing some sort of output.
- A programming language is a method of writing a source code file in an English-like language that a human being can understand.
- A compiler translates the source code file into instructions that the central processing unit can understand and execute.
- A COBOL program contains four DIVISIONs: the IDENTIFICATION DIVISION, the ENVIRONMENT DIVISION, the DATA DIVISION, and the PROCEDURE DIVISION.
- A DIVISION can be broken down into paragraphs. Some divisions have required paragraphs, such as the PROGRAM-ID paragraph in the IDENTIFICATION DIVISION.

- The names of the paragraphs in the PROCEDURE DIVISION are assigned by the programmer.
- The work of a COBOL program is done in sentences that contain the commands of a program and appear within a paragraph.
- A COBOL program is written in 80-column format. The columns are divided into the following areas:
    - Columns 1 through 6 are the sequence area and can be used by the programmer for line numbering. Line numbering is optional.
    - Column 7 is the indicator area. An asterisk (*) in column 7 causes everything to the right of the asterisk to be treated as a comment.
    - Columns 8 through 11 are Area A, and columns 12 through 72 are Area B. DIVISION names, SECTION names, and paragraph names must begin in Area A, but they can extend into Area B. Sentences begin and end in Area B.
    - Columns 73 through 80 are undefined and are not processed by the COBOL compiler, but some COBOL editors use these columns to tag lines with modification codes.

Today, you also learned how to type, compile, and run several simple COBOL programs.

Q&A

Q Why is so much of COBOL in uppercase?

A COBOL was developed in the days when computer terminals and keypunch machines used only uppercase letters. The entire language was defined in terms of this all-uppercase state of affairs. You can display uppercase and lowercase messages, such as Hello world, because terminals now have uppercase and lowercase capability. However, the actual elements of COBOL--the DIVISIONs, the paragraph names such as PROGRAM-BEGIN, and the verbs such as DISPLAY--originally were designed in uppercase only.

Q What does a blank line in a program do?

A Nothing. The compiler skips blank lines. You can put blank lines anywhere you want in the program to improve readability.

Q Does COBOL use line numbers?

A Yes, but they are optional. Most compilers ignore them or can be set to ignore them. Some compilers on larger computers will process the line numbers and provide a warning if the line numbers are out of sequence, but this is not an error.

Q Can I put anything in a comment?

A Yes. As long as the asterisk appears at column 7, everything after the asterisk is ignored and has no effect on the compiler. You can write English sentences or gobbledygook, although the usual practice is to provide some information that describes what the program is doing, or why it is doing it.

You can do this on as many lines as necessary to complete the comment.

Q Will my comments appear in the program when it runs?

A No. Comments appear only in the source code file, and they will not be included in the compiled program. Comments are intended to document the source code. The computer can read only the compiled code, and the comments wouldn't mean anything to the computer, so they are not included in the resulting program.

Q Why do paragraph names have hyphens in them?

A Paragraph names in COBOL are limited to 30 characters and may only contain uppercase letters (A through Z), digits (0 through 9), and the hyphen (-). The hyphen is used to improve the readability of paragraphs and the names of data (which is covered in Day 2). Some modern compilers allow paragraph names, division names, and other elements of COBOL to be typed in lowercase.

Workshop

Quiz

1. What is the output of the following program?

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. BYEBYE.
000300 ENVIRONMENT DIVISION.
000400 DATA DIVISION.
```

```
000500 PROCEDURE DIVISION.
000600
000700 PROGRAM-BEGIN.
000800     DISPLAY "Bye bye birdie".
000900 PROGRAM-DONE.
001000     STOP RUN.
```

2. How many DIVISIONs are in the following program, byebye.cbl?

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. BYEBYE.
000300 ENVIRONMENT DIVISION.
000400 DATA DIVISION.
000500 PROCEDURE DIVISION.
000600
000700 PROGRAM-BEGIN.
000800     DISPLAY "Bye bye birdie".
000900 PROGRAM-DONE.
001000     STOP RUN.
```

3. How many paragraphs?

4. How many sentences?

5. What is wrong with the following, bad01.cbl?

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. BAD01.
000300 ENVIRONMENT DIVISION.
000400
000500 PROCEDURE DIVISION.
000600
```

```
000700 PROGRAM-BEGIN.
000800     DISPLAY "I'm bad!".
000900 PROGRAM-DONE.
001000     STOP RUN.
```

6. What is wrong with the following, bad02.cbl?

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. BAD02.
000300 ENVIRONMENT DIVISION.
000400 DATA DIVISION.
000500 PROCEDURE DIVISION.
000600
000700 PROGRAM-BEGIN.
000800 DISPLAY "I'm bad!".
000900 PROGRAM-DONE.
001000     STOP RUN.
```

Hint: Where are sentences supposed to begin?

Note: Some compilers might not give an error on compiling BAD02, but might only provide a warning.

7. What is wrong with the following, bad03.cbl?

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. BAD03.
000300 ENVIRONMENT DIVISION.
000400 DATA DIVISION.
000500 PROCEDURE DIVISION.
000600     This program displays a message.
```

```
000700 PROGRAM-BEGIN.
000800     DISPLAY "I'm really bad!".
000900 PROGRAM-DONE.
001000     STOP RUN.
```

Exercises

1. Modify the hello.cbl program to display I am a COBOL programmer.

Hint: Copy the hello.cbl program to iam.cbl:

copy hello.cbl iam.cbl

Then use your editor or pcobol to change the PROGRAM-ID and DISPLAY statements.

2. Use the computer to compile each of the bad examples (bad01.cbl, bad02.cbl, and bad03.cbl) to get a feel for the types of error messages that your compiler produces.

COBOL-85 compilers are much more relaxed than earlier standards. The ENVIRONMENT, DATA, and PROCEDURE DIVISIONs are optional under ANSI 85, although what a program would do without a PROCEDURE DIVISION is a bit of a mystery. The results of compiling these programs are interesting.

Both Micro Focus Personal COBOL and ACUCOBOL found nothing wrong with bad01.cbl. Apparently, if a program contains no data, it doesn't need to have a DATA DIVISION.

Micro Focus Personal COBOL handled bad02.cbl without a hiccup.

ACUCOBOL produced a warning that a sentence was starting in Area A.

Both Micro Focus Personal COBOL and ACUCOBOL generated errors on bad03.cbl and would not compile it.

3. Modify bad01.cbl so that it compiles without errors or warnings.

4. Modify bad02.cbl so that it compiles without errors or warnings.

5. Modify bad03.cbl so that it compiles without errors or warnings.