GNU COBOL 02

Using Variables and Constants

The DATA DIVISION is a critical subject to understand in COBOL. Today's lesson covers the basics, but you will be returning to the DATA DIVISION again and again throughout this book.

Today, you learn about the following topics:

- **What is a constant?**
- **What is a variable?**
- **Defining numeric variables in COBOL.**
- **Naming variables in COBOL.**
- **More on using DISPLAY.**
- **Defining and using variables.**
- **Defining pictures.**
- **Using the MOVE verb.**
- **Formatting output.**
- **Tips on layout and punctuation.**
- **Continuation characters.**

What Is a Constant?

The data in COBOL programs falls into two broad categories: constants and variables.

New Term: A *constant* is a value that cannot be modified while the program is running.

You already have used constants in the hello.cbl program and examples in Day 1, "Your First COBOL Program." In Listing 1.1, the string "Hello world" is a constant.

In hello.cbl, there is no way to modify the display of "Hello world" without editing the program and recompiling it, as shown in Listing 2.1. This effectively creates a new program.

TYPE: Listing 2.1. Modifying a constant by editing and recompiling.

000100 IDENTIFICATION DIVISION.

000200 PROGRAM-ID. HELLO.

000300 ENVIRONMENT DIVISION.

000400 DATA DIVISION.

000500 PROCEDURE DIVISION.

000600

000700 PROGRAM-BEGIN.

000800   DISPLAY "I said, Hello world".

000900

001000 PROGRAM-DONE.

001100   STOP RUN.

If you want to display both messages, you have to use code similar to what you see in Listing 2.2; however, this merely uses two different constants in one program.

TYPE: Listing 2.2. Adding another constant.

000100 IDENTIFICATION DIVISION.

000200 PROGRAM-ID. HELOHELO.

000300 ENVIRONMENT DIVISION.

000400 DATA DIVISION.

000500 PROCEDURE DIVISION.

000600

000700 PROGRAM-BEGIN.

000800   DISPLAY "Hello world".

000900   DISPLAY "I said, Hello world".

001000 PROGRAM-DONE.

001100   STOP RUN.

Numeric constants can be used in a similar way. Listing 2.3 includes examples of numeric constants (55 and 12.607). Note the difference between character constants,

such as "Hello world", and numeric constants, such as 12.607. Character constants are enclosed in quotation marks. Numeric constants are not.

New Term: Character constants also are called *string constants*.

TYPE: Listing 2.3. String and numeric constants.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. CONST.
000300 ENVIRONMENT DIVISION.
000400 DATA DIVISION.
000500 PROCEDURE DIVISION.
000600
000700 PROGRAM-BEGIN.
000800     DISPLAY "Hello world".
000900     DISPLAY 55.
001000     DISPLAY 12.607.
001100 PROGRAM-DONE.
001200     STOP RUN.
```

Character constants can contain spaces, such as the space between the words in "Hello world". Without the double quotation marks at each end, the compiler would have trouble recognizing all of the character constant; it would not know where the constant ended. The following is a classic example of this problem:

```
DISPLAY THIS IS THE DISPLAY.
```

DISPLAY is a COBOL verb. In this example, the first occurrence of the word DISPLAY is the COBOL verb for displaying information. The compiler could become confused as to whether the second occurrence of the word DISPLAY is part of the message to be displayed or is another DISPLAY verb. To keep the compiler from having a nervous breakdown, the programmer is required to write the following:

```
DISPLAY "THIS IS THE DISPLAY".
```

Every popular programming language includes a requirement that character constants be enclosed in some sort of quotation marks or other signaling characters to indicate

the exact beginning and end of the character constant. COBOL uses double quotation marks at the beginning and end of the characters.

Numeric constants such as the following do not contain *white space*, and it is much easier for the compiler to recognize them as numbers:

DISPLAY 12.607.

Most popular languages do not require any special characters to signal a numeric constant.

New Term: *White space* is a general term to cover any blank characters. A space is white space, as is a tab, though you won't use tabs in COBOL programs. They are called white space because they print as white spaces on white paper when sent to a printer.

What Is a Variable?

Computers are capable of dealing with a large amount of data, and the data should be able to change while the computer is running. So, how do you change data? You use something called a variable.

New Term: A *variable* is a value that can be changed while the program is running.

When a variable is created in a program, an area of memory is set aside to hold values. In most programming languages, including COBOL, a variable is given a name. The name can be used in the program to refer to the value.

The value stored in memory can be modified while the program is running by using the variable name. You'll see some examples of using variables later in today's lesson; but you first must understand how to define a variable.

Defining Numeric Variables in COBOL

The following is an example of a COBOL numeric variable. Variable names use the same characters as paragraph names: A through Z, 0 through 9, and the hyphen (-). This is described more fully in the section "Naming Variables in COBOL," later in today's lesson.

```
001400 01  THE-NUMBER          PICTURE IS 9999.
```

A COBOL variable definition contains at least three parts:

- **The level number**
- **The name**

- **The PICTURE**

In the syntax, the level number is 01. For now, every variable you will be using will have a level number of 01. The level number 01 must be in Area A, columns 8 through 11. In the previous code fragment, the 01 starts in column 8.

The second part of the variable definition is the name of the variable and, in this case, is THE-NUMBER. This is the data name used to identify the variable. The data name is assigned by the programmer. The variable will be referred to by its data name, THE-NUMBER, anywhere in the program that the variable must be set or modified. The name of the variable must start in Area B, columns 12 through 72. In this example, THE-NUMBER starts in column 12.

The PICTURE defines two things about a variable: the size of the variable (the number of bytes used in memory for the value) and the type of data that can be stored in the variable. In this example, the picture 9999 indicates that four numeric characters can be stored in the variable named THE-NUMBER. Similarly, a variable with a PICTURE IS 99 could hold two numeric characters. The PICTURE IS clause and the actual picture 9999 must start somewhere in Area B, columns 12 through 72.

The PICTURE IS clause in the definition of a variable is the COBOL syntax for introducing the size of a variable and the type of data that a variable holds.

The 9999 in the picture does not indicate that the variable contains the value 9999. It indicates that the variable can be used for numeric values in the range 0 through 9,999. The 9999 picture indicates that four numeric digits can be stored in this variable. The picture 9999 will hold any of the values from 0 through 9,999. The values 17 and 6,489 will both fit in THE-NUMBER, but the value 65,413 is too large.

Look at Listing 2.4, add01.cbl, for the general format of the program now that it contains variables. Three variables are created in this program: FIRST-NUMBER, SECOND-NUMBER, and THE-RESULT. Each variable has the level number 01. The first two have pictures of 99 and will hold values ranging from 0 through 99. The third variable, THE-RESULT, has a picture of 999 and will hold a value of 0 through 999. Once again, the PICTURE IS clause does not set the value of the variable; it sets only the largest and smallest values that a variable can hold and the fact that the variable will hold numeric data.

TYPE: Listing 2.4. Using variables.

```
000100 IDENTIFICATION DIVISION.

000200 PROGRAM-ID. ADD01.
```

```
000300 ENVIRONMENT DIVISION.
000400 DATA DIVISION.
000500
000600 WORKING-STORAGE SECTION.
000700
000800 01  FIRST-NUMBER     PICTURE IS 99.
000900 01  SECOND-NUMBER    PICTURE IS 99.
001000 01  THE-RESULT       PICTURE IS 999.
001100
001200 PROCEDURE DIVISION.
001300
001400 PROGRAM-BEGIN.
001500
001600     DISPLAY "Enter the first number.".
001700
001800     ACCEPT FIRST-NUMBER.
001900
002000     DISPLAY "Enter the second number.".
002100
002200     ACCEPT SECOND-NUMBER.
002300
002400     COMPUTE THE-RESULT = FIRST-NUMBER + SECOND-NUMBER.
002500
002600     DISPLAY "The result is:".
002700     DISPLAY THE-RESULT.
002800
```

002900 PROGRAM-DONE.

003000 STOP RUN.

003100

Load and compile or type in with your editor and compile Listing 2.4, add01.CBL. When you run the program, you will be asked to enter the first number, as shown in the following output. Note that the final blank line 003100 in the listing has no effect on the program. You can leave it out if you wish.

OUTPUT:

C>pcobrun add01

Enter the first number.

First, type 97 and then press Enter. You will be asked for a second number, in a screen looking something like this:

C>pcobrun add01

Enter the first number.

97

Enter the second number.

Now type 33 and press Enter. The two numbers are added together and displayed:

C>pcobrun add01

Enter the first number.

97

Enter the second number.

33

The result is:

130

C>

ANALYSIS: Take a look at the DATA DIVISION. Here is your first example of a section, the WORKING-STORAGE SECTION. A SECTION in COBOL is created by typing a name, similar to a paragraph name, followed by one or more spaces, followed by the word SECTION and a period. SECTIONs in COBOL can be required or optional, depending on which DIVISION they are in. WORKING-STORAGE SECTION is a reserved name and a required section in the DATA DIVISION if your program uses any variables--and most programs do.

DO/DON'T:
DO precede the word SECTION with at least one space (WORKING-STORAGE SECTION).

DON'T precede SECTION with an extra hyphen (WORKING-STORAGE-SECTION).

Each of the variables is defined with a 01, a variable name, and a PICTURE. The PICTURE IS clauses are lined up on the right. There is no reason for this other than tidiness. As long as the PICTURE clause starts and ends in Area B, there are no other restrictions on alignment or position.

Now look at Listing 2.4 again, but this time from the perspective of a running program. In the DATA DIVISION, space is created for two variables with pictures of 99 and one variable with a picture of 999.

In the PROCEDURE DIVISION, a message is displayed for the user at line 001600, asking the user to enter the first variable. At line 001800, this value is accepted from the keyboard, using the ACCEPT verb. ACCEPT is a verb that causes the program to wait for input from the keyboard. Digits can be typed until the user presses Enter. When the user presses Enter, the value of the digits entered is moved to the variable named immediately after the ACCEPT verb.

When the program is running and encounters the sentence

ACCEPT FIRST-NUMBER

the computer stops and waits for the user to type. Whatever value is typed is stored in the two bytes of FIRST-NUMBER.

At line 002000, the user is asked for another number. This is accepted at line 002200 using the ACCEPT verb and stored in SECOND-NUMBER.

At line 002400, the COBOL COMPUTE verb is used to add the two values together. In the following statement, the values that have been stored at FIRST-NUMBER and SECOND-NUMBER are retrieved and added together, using + to perform the addition:

COMPUTE THE-RESULT = FIRST-NUMBER + SECOND-NUMBER.

The result of this addition is stored at THE-RESULT.

Finally, in lines 002600 and 002700, the result of the addition is displayed.

In this example, two of the variables--FIRST-NUMBER and SECOND-NUMBER--have been modified or "varied" by the user entering values at the keyboard. The third variable, THE-RESULT, was modified by the program. The program uses the COMPUTE statement to calculate a new value and store it in THE-RESULT. This is what variables are all about: the ability to vary their values while the program is running.

Some versions of COBOL (ACUCOBOL, for example) are picky about accepting data from the keyboard. If a field is defined with a PICTURE IS 99, you must enter two digits in response to an ACCEPT. To enter the number 3, you must enter 03. To enter 7 into a PICTURE 99999 field, you must enter 00007. ACUCOBOL includes an option to change this behavior, and I am informed that, as of their version 3.0 compiler, the ACCEPT verb behaves in a more relaxed manner, allowing numeric entry without the preceding zeroes. Other versions of COBOL, such as Micro Focus Personal COBOL, use this more relaxed approach to ACCEPT. If you want to enter a 3 into a PICTURE 999, just enter a 3 and press Enter. The COBOL language will correctly store 003 in the PICTURE 999.

For the time being, enter all the digits to avoid any problems. If a program complains of non-numeric data in a numeric field, you probably have not entered enough leading zeroes.

Type the program from Listing 2.4 into your computer and compile it. Run it several times, entering different values each time. You will see that the three variables truly are variable, because their values are determined while the program is running. You do not have to edit and recompile the program each time that you want to get a new result.

Take one more look at Listing 2.4, add01.cbl, for a quick review. Line numbers appear in columns 1 through 6, the sequence area. Comments start in the indicator area, and they start with an asterisk in that column. DIVISIONs, SECTIONs, paragraphs, and the

01 level number of a variable start in columns 8, Area A. Everything else starts and ends in Area B, usually at column 12.

Naming Variables in COBOL

COBOL variable names are similar to paragraph and section names because they can use any of the uppercase alphabet characters, the digits 0 through 9, and the hyphen (but not as a starting character). COBOL variable and paragraph names are limited to 30 characters. Table 2.1 provides some examples of valid and invalid variable names.

Table 2.1. Valid and invalid variable names.

Valid Name	Invalid Name	Explanation of Invalid Name
TOTAL-DOLLARS	TOTAL-$	Uses an invalid $ in the name
SUM-OF-COLUMNS	sum-of-columns	Uses lowercase letters
7-BY-5	7_BY_5	Uses the invalid _ character in the name
MINUS-RESULT	-RESULT	Starts with a hyphen
BOEING-707-SEATS	BOEING-707-MAXIMUM- SEATING-CAPACITY	Exceeds 30 characters

Some modern compilers have disposed of the uppercase requirement and will accept variable names such as sum-of-columns and 7-by-5, but it is a good practice to use uppercase because it makes your code more portable between COBOL compilers.

More on Using DISPLAY

The DISPLAY verb can be used to display more than one value at a time, like so:

DISPLAY "The result is " THE-RESULT.

It is used this way in Listing 2.5, add02.cbl. The only change in this program is that lines 002600 and 002700 have been combined into a single DISPLAY line, and the colon (:) in the message has been replaced with a space.

TYPE: Listing 2.5. Combining values in one DISPLAY statement.

000100 IDENTIFICATION DIVISION.

```
000200 PROGRAM-ID. ADD02.
000300 ENVIRONMENT DIVISION.
000400 DATA DIVISION.
000500
000600 WORKING-STORAGE SECTION.
000700
000800 01  FIRST-NUMBER     PICTURE IS 99.
000900 01  SECOND-NUMBER    PICTURE IS 99.
001000 01  THE-RESULT       PICTURE IS 999.
001100
001200 PROCEDURE DIVISION.
001300
001400 PROGRAM-BEGIN.
001500
001600    DISPLAY "Enter the first number.".
001700
001800    ACCEPT FIRST-NUMBER.
001900
002000    DISPLAY "Enter the second number.".
002100
002200    ACCEPT SECOND-NUMBER.
002300
002400    COMPUTE THE-RESULT = FIRST-NUMBER + SECOND-NUMBER.
002500
002600    DISPLAY "The result is " THE-RESULT.
002700
```

002800

002900 PROGRAM-DONE.

003000   STOP RUN.

003100

This is the output from a sample session with add02.cbl. The result line is combined into one line of display:

OUTPUT:

C>pcobrun add02

Enter the first number.

16

Enter the second number.

93

The result is 109

C>

Defining and Using Variables

When you define a variable in the WORKING-STORAGE SECTION of the DATA DIVISION, you are providing information for the compiler about the size of the variable and the type of data that can be stored in it.

A numeric variable is used to store numbers. The picture character used to represent a digit in a numeric variable is a 9, as in this example:

01 THE-NUMBER   PICTURE IS 99.

This description defines a variable named THE-NUMBER that can be used to hold a numeric variable that is two digits long; in other words, any value in the range of 0 through 99.

New Term: Variables that can hold character data are called alphanumeric variables.

Alphanumeric data contains one or more printable characters. Some examples of alphanumeric values are Hello, ??506^%$A, and 123-B707. An alphanumeric variable is defined in the same way as a numeric variable, except that the picture character used to represent one alphanumeric character is an X. The following syntax example defines an alphanumeric variable that can hold a word or message of no more than 10 characters:

```
001200 01 THE-MESSAGE    PICTURE IS XXXXXXXXXX.
```

An alphanumeric variable can also be used to hold numbers (such as storing 123 in a PICTURE IS XXX variable), but you will not be able to use the values as numbers. For example, you could display the PICTURE IS XXX variable containing 123, but you couldn't use the COMPUTE verb to add 1 to it.

In Listing 2.6, a modified version of hello.cbl named hello02.cbl illustrates the use of alphanumeric variables.

TYPE: Listing 2.6. Using an alphanumeric variable.

```
000100 IDENTIFICATION DIVISION.

000200 PROGRAM-ID. HELLO02.

000300 ENVIRONMENT DIVISION.

000400 DATA DIVISION.

000500

000600 WORKING-STORAGE SECTION.

000700

000800 01  THE-NAME     PICTURE IS XXXXXXXXXX.

000900

001000 PROCEDURE DIVISION.

001100

001200 PROGRAM-BEGIN.

001300

001400    DISPLAY "Enter someone's name.".

001500
```

001600 ACCEPT THE-NAME.

001700

001800 DISPLAY "Hello " THE-NAME.

001900

002000 PROGRAM-DONE.

002100 STOP RUN.

The following is an example of the output from hello02.cbl, using Erica as the name entered at the keyboard:

OUTPUT:

C>pcobrun hello02

Enter someone's name.

Erica

Hello Erica

C>

ANALYSIS: At line 001400, the user is asked to enter a name. At line 001600, the ACCEPT verb will cause the computer to accept input from the keyboard until the user presses Enter. Whatever is typed (up to 10 characters) will be stored in THE-NAME. THE-NAME then is displayed in a hello message.

Defining Pictures

So far, you've defined small variables, but you also can define longer ones in COBOL. Numeric variables can be as large as 18 digits:

01 THE-NUMBER PICTURE IS 999999999999999999.

Numeric variables are limited to 18 digits, but the length of alphanumeric variables is limited by the version of COBOL that you have. LPI COBOL has a limit of 32,767 characters. ACUCOBOL has a limit of 65,520 characters. The Professional Micro Focus COBOL compiler (big brother to Micro Focus Personal COBOL) has a limit of a whopping 256 million characters.

Defining long variables could become a tedious task if every X and 9 had to be spelled out explicitly; and typing in long strings of X or 9 could result in errors. In addition, having to type PICTURE IS for every variable can get tiring in large programs. Fortunately, COBOL allows some abbreviations that make the task less cumbersome.

The word IS in PICTURE IS is optional, and the word PICTURE can be abbreviated as PIC. This abbreviation is used so commonly that it is rare to see a program containing PICTURE IS:

```
01  THE-MESSAGE    PIC XXXXXX.
```

The second abbreviation is even more useful. The picture itself can be abbreviated by typing one picture character followed by the number of repetitions of that character in parentheses. Thus, PIC XXXXXX becomes PIC X(6), and PIC 99999 becomes PIC 9(5). The 18-digit number shown earlier becomes the following:

```
01  THE-NUMBER    PIC 9(18).
```

This works even when the repetition is one, so it is possible to describe PIC X as PIC X(1). When you are reading a listing, it sometimes is easier to determine the size of a variable quickly by scanning the values in parentheses. Some programmers make it a practice always to include the size in parentheses.

If you want to use the abbreviations to cut down on keystrokes, abbreviate anything exceeding a length of four. PIC XXXX and PIC X(4) require the same number of keystrokes, so for the sake of typing speed, it doesn't matter which you use. PIC X is faster to type than PIC X(1), but PIC X(5) is faster to type than PIC XXXXX.

You might find the use of parentheses is dictated by the style manual of the company that is using the program. If it is for your own use, pick the one that is more comfortable for you.

Introducing the MOVE Verb

The MOVE verb in COBOL is a general-purpose verb, used to store a value in a variable. The general syntax for MOVE is the following:

```
MOVE value TO variable.
```

In this syntax, variable must be a variable defined in the DATA DIVISION, and value can be another variable or a constant.

Here are some examples:

```
MOVE 12 TO THE-NUMBER.
```

MOVE ONE-NUMBER TO ANOTHER-NUMBER.

MOVE "XYZ" TO THE-MESSAGE.

MOVE is used to set a variable to a specific value. For example, if you're going to use the variable THE-COUNTER as a counter and you need the count to start at 1, you might use the following as one method of setting up the variable with a starting value:

MOVE 1 TO THE-COUNTER.

MOVE in COBOL does not move memory physically from one place to another. It copies values from the source variable and stores them in the target variable. Table 2.2 describes the effect of some different MOVE examples that move constants and variables into variables. All variables are assumed to be defined in the WORKING-STORAGE SECTION.

Table 2.2. Examples of the MOVE verb.

Command	Effect
MOVE 19 TO THE-NUMBER	Stores 19 in the variable THE-NUMBER, or sets THE-NUMBER to a value of 19
MOVE "Hello" TO THE-MESSAGE	Stores Hello in the variable THE-MESSAGE, or sets THE-MESSAGE to contain Hello
MOVE A-NUMBER TO THE-NUMBER	Locates the variable named A-NUMBER, gets the value stored there, and copies it or moves it to the variable named THE-NUMBER
MOVE THE-OLD-NAME TO THE-NEW-NAME	Locates the variable named THE-OLD-NAME, gets the value stored there, and copies it or moves it to the variable named THE-NEW-NAME

Listing 2.7 is a program designed solely to provide examples of the MOVE verb. It combines PICTURE abbreviations, multiple DISPLAY statements, and MOVE statements to display two messages, with message numbers, on the screen. This will give you a further idea of the uses and effects of MOVE.

TYPE: Listing 2.7. Using MOVE.

```
000100 IDENTIFICATION DIVISION.

000200 PROGRAM-ID. HELLO03.
```

```
000300 ENVIRONMENT DIVISION.
000400 DATA DIVISION.
000500
000600 WORKING-STORAGE SECTION.
000700
000800 01  THE-MESSAGE     PIC X(20).
000900 01  THE-NAME        PIC X(10).
001000 01  THE-NUMBER      PIC 99.
001100
001200 PROCEDURE DIVISION.
001300
001400 PROGRAM-BEGIN.
001500
001600    DISPLAY "Enter someone's name.".
001700
001800    ACCEPT THE-NAME.
001900
002000    MOVE "Hello" TO THE-MESSAGE.
002100
002200    MOVE 1 TO THE-NUMBER.
002300
002400    DISPLAY "Message "
002500            THE-NUMBER
002600            ": "
002700            THE-MESSAGE
002800            THE-NAME.
```

```
002900
003000    MOVE "Say Goodnight," TO THE-MESSAGE.
003100
003200    MOVE 2 TO THE-NUMBER.
003300
003400    DISPLAY "Message "
003500            THE-NUMBER
003600            ": "
003700            THE-MESSAGE
003800            THE-NAME.
003900
004000
004100 PROGRAM-DONE.
004200    STOP RUN.
004300
```

OUTPUT:

```
C>pcobrun hello03

Enter someone's name.
Gracie
Message 01: Hello           Gracie
Message 02: Say Goodnight,     Gracie

C>
```

ANALYSIS: Lines 000800, 000900, and 001000 contain abbreviated PICTURES. THE-MESSAGE is a 20-character alphanumeric field, and THE-NAME is a 10-character

alphanumeric field. The user is asked to enter a name, and this is accepted from the keyboard into THE-NAME at line 001800.

In lines 002000 and 002200, MOVE is used to move values to THE-MESSAGE and the THE-NUMBER. Lines 002400 through 002800 contain one long DISPLAY statement. Notice that this long statement ends with only one period, on line 002800. COBOL sentences can spread over more than one line as in this example, as long as they remain within Area B, which is columns 12 through 72. This DISPLAY creates one line of display information containing the values Message, THE-NUMBER, :, THE-MESSAGE, and THE-NAME, one after the other on a single line:

Message 01: Hello Charlie

Similar logic is repeated at lines 003000 through 003800, and a second line is displayed. See if you can guess how the output will appear before taking a look.

Note that the output from hello03.cbl is shown for an input name of Gracie. Listing 2.7 is a good program for practice. First type, edit, and compile hello03.cbl, and try it a couple of times. Then copy the program to hello04.cbl and edit it. Try different constants and display orders for the DISPLAY statements. Here are a couple of alternatives for the first DISPLAY statement:

DISPLAY "Line "

THE-NUMBER

"> "

THE-MESSAGE

THE-NAME.

DISPLAY THE-MESSAGE

THE-NAME

" was Number "

THE-NUMBER.

The following are sample output lines from these two formats:

Line 01> Hello Charlie

Hello Charlie was Number 01

One of the features of the MOVE verb is that it will pad a variable with spaces to the end if the value that is being moved into an alphanumeric field is too short to fill the field. This is convenient; it's almost always what you want. In the line MOVE "Hello" TO THE-MESSAGE, the first five characters of THE-MESSAGE are filled with Hello, and the remaining character positions are filled with spaces. (In Bonus Day 2, "Miscellaneous COBOL Syntax," you learn how to move a field while dropping the trailing space used for padding.)

MOVE pads numeric variables by filling them with zeroes to the left. Notice that a numeric value with a PIC 99 containing a value of 1 will display as 01 in Listing 2.7. This is because of the padding action of the MOVE verb.

Values that are too long are truncated. If THE-MESSAGE is defined as a PIC X(7), the line

MOVE "Hello world" to THE-MESSAGE

results in THE-MESSAGE containing Hello w and the rest of the value falling off the end.

Moving a value that is too large to a numeric variable results in a similar truncation, but on the left side. If THE-NUMBER is defined as a PIC 9999, the following line results in THE-NUMBER containing 1784:

MOVE 61784 TO THE-NUMBER

There isn't room for all five digits, so only the four digits on the right are picked up on the move.

Formatting Output

Listing 2.8 is another example of some of the principles you have learned. It displays three lines of a nursery rhyme involving some work that Jack had to do. The ADD verb (which is new in this listing) increments the value of THE-NUMBER as each line is displayed. In the DISPLAY statements, a space is used to separate the line number from the statement. Remember that the asterisk in column 7 is used to place a comment in the code.

TYPE: Listing 2.8. Using the ADD verb.

000100 IDENTIFICATION DIVISION.

000200 PROGRAM-ID. JACK01.

000300 ENVIRONMENT DIVISION.

```
000400 DATA DIVISION.
000500
000600 WORKING-STORAGE SECTION.
000700
000800 01  THE-MESSAGE     PIC X(50).
000900 01  THE-NUMBER      PIC 9(2).
001000
001100 PROCEDURE DIVISION.
001200 PROGRAM-BEGIN.
001300
001400* Set up and display line 1
001500     MOVE 1 TO THE-NUMBER.
001600     MOVE "Jack be nimble," TO THE-MESSAGE.
001700     DISPLAY THE-NUMBER " " THE-MESSAGE.
001800
001900* Set up and Display line 2
002000     ADD 1 TO THE-NUMBER.
002100     MOVE "Jack be quick," TO THE-MESSAGE.
002200     DISPLAY THE-NUMBER " " THE-MESSAGE.
002300
002400* Set up and display line 3
002500     ADD 1 TO THE-NUMBER.
002600     MOVE "Jack jump over the candlestick." TO THE-MESSAGE.
002700     DISPLAY THE-NUMBER " " THE-MESSAGE.
002800
002900 PROGRAM-DONE.
```

003000   STOP RUN.

003100

OUTPUT:

01 Jack be nimble,

02 Jack be quick,

03 Jack jump over the candlestick.

C>

C>

ANALYSIS: It is possible to use a variable as though it were a constant. In Listing 2.9, an additional variable, A-SPACE, is created. This variable is set to a value at the start of the program and then used in each message. The output of jack02.cbl should be identical to that of jack02.cbl.

New Term: Setting a variable to a starting value is called *initializing*.

TYPE: Listing 2.9. Another method of formatting.

000100 IDENTIFICATION DIVISION.

000200 PROGRAM-ID. JACK02.

000300 ENVIRONMENT DIVISION.

000400 DATA DIVISION.

000500

000600 WORKING-STORAGE SECTION.

000700

000800 01 THE-MESSAGE   PIC X(50).

000900 01 THE-NUMBER   PIC 9(2).

001000 01 A-SPACE   PIC X.

001100

001200 PROCEDURE DIVISION.

001300 PROGRAM-BEGIN.

```
001400
001500* Initialize the space variable
001600     MOVE " " TO A-SPACE.
001700
001800* Set up and display line 1
001900     MOVE 1 TO THE-NUMBER.
002000     MOVE "Jack be nimble," TO THE-MESSAGE.
002100     DISPLAY THE-NUMBER A-SPACE THE-MESSAGE.
002200
002300* Set up and Display line 2
002400     ADD 1 TO THE-NUMBER.
002500     MOVE "Jack be quick," TO THE-MESSAGE.
002600     DISPLAY THE-NUMBER A-SPACE THE-MESSAGE.
002700
002800* Set up and display line 3
002900     ADD 1 TO THE-NUMBER.
003000     MOVE "Jack jump over the candlestick." TO THE-MESSAGE.
003100     DISPLAY THE-NUMBER A-SPACE THE-MESSAGE.
003200
003300 PROGRAM-DONE.
003400     STOP RUN.
003500
```

Layout and Punctuation

As the programs you are working on get larger, it is a good idea to start paying attention to layout and other features of COBOL that can help make your code more readable.

Commas can be used in COBOL to separate items in a list:

DISPLAY THE-NUMBER, " ", THE-MESSAGE.

There are arguments for and against the use of commas. As far as the COBOL compiler is concerned, commas are optional; the compiler ignores them completely. The only use, therefore, is to improve readability. In a list of variables, the commas help to separate the elements when you are reading the code.

Serious problems result from mistyping a period for a comma. If the screen does not provide a clear display or the printer is printing the source code with a feeble ribbon, it is possible to mistake a comma for a period. A period is not optional and is a critical piece of COBOL syntax used to end sentences. The confusion of a comma for a period has caused some serious problems in programs, and it might be better to leave commas out unless there is a compelling reason to use them.

A sentence does not have to begin and end on one line. As long as it stays out of Area A (columns 8 through 11), a sentence can spread over multiple lines. Listing 2.7, hello03.cbl, uses this technique to clearly separate each value that is being displayed. Listing 2.10 is a version of jack02.cbl that spreads the DISPLAY sentence out in order to clarify what is being displayed. This will compile and run identically to jack02.cbl, but it is a little easier to read.

TYPE: Listing 2.10. Spreading out a sentence.

```
000100 IDENTIFICATION DIVISION.

000200 PROGRAM-ID. JACK03.

000300 ENVIRONMENT DIVISION.

000400 DATA DIVISION.

000500

000600 WORKING-STORAGE SECTION.

000700

000800 01  THE-MESSAGE     PIC X(50).

000900 01  THE-NUMBER      PIC 9(2).

001000 01  A-SPACE         PIC X.

001100

001200 PROCEDURE DIVISION.
```

```
001300 PROGRAM-BEGIN.
001400
001500* Initialize the space variable
001600     MOVE " " TO A-SPACE.
001700
001800* Set up and display line 1
001900     MOVE 1 TO THE-NUMBER.
002000     MOVE "Jack be nimble," TO THE-MESSAGE.
002100     DISPLAY
002200       THE-NUMBER
002300       A-SPACE
002400       THE-MESSAGE.
002500
002600* Set up and Display line 2
002700     ADD 1 TO THE-NUMBER.
002800     MOVE "Jack be quick," TO THE-MESSAGE.
002900     DISPLAY
003000       THE-NUMBER
003100       A-SPACE
003200       THE-MESSAGE.
003300
003400* Set up and display line 3
003500     ADD 1 TO THE-NUMBER.
003600     MOVE "Jack jump over the candlestick." TO THE-MESSAGE.
003700     DISPLAY
003800       THE-NUMBER
```

003900 A-SPACE

004000 THE-MESSAGE.

004100

004200 PROGRAM-DONE.

004300 STOP RUN.

004400

Remember that, when you are reading COBOL programs, a sentence continues until a period is encountered, no matter how many lines it takes.

Continuation Characters

When an alphanumeric value is too long to fit on a single line, it can be continued on the next line by using a continuation character. In Listing 2.11, the columns have been included. The message must be continued to the end of Area B (column 72) and ends without a closing quote. The next line begins with a hyphen (-) in column 7 to indicate that the previous quoted string is being continued. The rest of the message starts with a quote and continues as long as is necessary to complete the message. Lines can be continued over more than one line if necessary.

Listing 2.11. The continuation character.

```
      1       2       3       4       5       6       7       8
12345678901234567890123456789012345678901234567890123456789012345678901234
4567890

000500 01  LONG-MESSAGE    PIC X(80) VALUE "This is an incredibly long m
000600-    "essage that will take more than one line to define".
```

Summary

Today, you learned the basics about COBOL's DATA DIVISION, including the following:

- **The WORKING-STORAGE SECTION of the DATA DIVISION is used to create space for the variables of a program.**
- **Variables in WORKING-STORAGE are given names. The names are assigned by the programmer.**

- **Variables can be named using the uppercase characters A through Z, the digits 0 through 9, and the hyphen (-). The hyphen cannot be the first character of a variable name.**
- **Variables are divided into two broad classes: alphanumeric and numeric.**
- **Alphanumeric variables can hold printable characters: A through Z, a through z, 0 through 9, spaces, symbols, and punctuation characters.**
- **Numeric variables can hold numbers.**
- **Alphanumeric values must be enclosed in double quotation marks when being moved to variables.**
- **Numeric values being moved to numeric variables do not require quotation marks.**
- **The MOVE verb moves an alphanumeric value to an alphanumeric variable and pads the variable with spaces on the right if the value is too short to fill the variable.**
- **The MOVE verb moves a numeric value to a numeric variable and pads the value on the left with zeroes if the value is too small to fill the variable.**
- **The DISPLAY verb can display more than one value or variable at a time.**
- **A COBOL sentence can contain commas for punctuation. They do not affect the behavior of the final program, but they can be included to improve readability.**
- **A COBOL sentence ends with a period. It can spread over several lines of the source code file, as long as it stays within Area B, columns 12 through 72.**
- **A continuation character can be used to continue a literal on one or more subsequent lines.**

Q&A

Q Are there other limits on variable names?

A For now, you should ensure that each variable name is different. You have up to 30 characters to use for a variable name, so you should have no trouble coming up with different names.

Q When should you use a variable, and when should you use a constant?

A Most of the work in programming is done with variables. You can use a constant inside the PROCEDURE DIVISION when it will never need to be changed.

Even when a constant will never change, it sometimes is clearer to use a variable, because it explains what is happening. In the following example, the first line indicates that the sales amount is being multiplied by the constant .10 (10 percent), but gives no information on why. The value .10 is a constant because it cannot be changed without first editing the program and recompiling it. The second version indicates that some logic is being executed to calculate a sales commission:

MULTIPLY SALES-AMOUNT BY .10.

MULTIPLY SALES-AMOUNT BY COMMISSION-RATE.

Workshop

Quiz

1. How many bytes of memory are used by the following variable?

01 CUSTOMER-NAME PIC X(30).

2. What type of data can be stored in CUSTOMER-NAME?

3. If you move a value to CUSTOMER-NAME, such as

MOVE "ABC Company" TO CUSTOMER-NAME.

only the first 11 characters of the variable are filled with the value. What is placed in the remaining 19 character positions?

4. What is the largest number that can be moved using MOVE to the following variable?

01 UNITS-SOLD PIC 9(4).

5. What is the smallest value that can be moved using MOVE to UNITS-SOLD?

6. If 12 is moved to UNITS-SOLD, as in

MOVE 12 to UNITS-SOLD.

what values are stored in the four numeric positions of UNITS-SOLD?

Exercises

1. Modify add02.cbl from Listing 2.5 to display a message that tells the user what the program will do.

***Hint*: Add a message at line 001500.**

2. Pick a poem or phrase of your own choosing that has four or more lines (but no more than 10 lines) and display it on the screen with line numbers.

3. Repeat this poem but have the line numbers start at 05 and increment by 5.

***Hint*: You can start by moving 5 to THE-NUMBER, and then increment the value by using ADD 5 TO THE-NUMBER for each line that is being printed.**