CHAPTER 1

If You've Never Programmed Before

Objectives

After reading this chapter, you should be able to

- Understand the "idea" of a program.
- Understand the concept of "assignment."
- Write simple decisions and loops.

Although some readers will have some previous programming experience, this may be your first exposure to programming. Therefore, I've written this chapter to provide some fundamental prerequisite concepts for those who have never programmed before. In particular, it is critical that neophytes understand the following three topics:

- Programs
- Assignment
- Decisions and loops

If you are a veteran programmer, most of this chapter will be rudimentary. However, those programmers who are unfamiliar with Basic should read the last two sections. These are designed to acquaint you with how VBA represents decisions (that is, If statements) and count-controlled loops.

1.1 THE "IDEA" OF A PROGRAM

Computer programs are like road directions. Suppose I want to tell someone how to drive from point *A* to point *B*. Here's an example:

1. Leave parking lot and turn right onto Winter Street.
2. Drive 1 mile.
3. Turn right onto Route 30 East.
4. Drive 0.5 mile.
5. Turn left onto Wellesley Street.
6. Drive 0.8 mile.
7. Turn left onto Chestnut Street.

A person could follow these directions in a step-by-step fashion and arrive at Chestnut Street with no problem.

However, suppose that we switched two of the instructions:

1. Leave parking lot and turn right onto Winter Street.
2. Drive 1 mile.

3. Turn left onto Wellesley Street.
4. Drive 0.5 mile.
5. Turn right onto Route 30 East.
6. Drive 0.8 mile.
7. Turn left onto Chestnut Street.

As a consequence, the individual would become hopelessly lost. To be effective, road directions must be unambiguous and totally logical. If not, they won't work the way you want them to.

The same holds true for computer programs. A *program* is a sequence of unambiguous instructions that the computer implements one after another in a mindless manner. These instructions are sometimes called *statements*. For example, a very simple VBA Sub program to add two numbers can be written as

```
Sub Adder()
a = 10
b = 33
c = a + b
End Sub
```

It's not too difficult to see that this program assigns two numeric constants to the variables **a** and **b**, and then adds them and assigns the result to the variable **c**.

Now, suppose you redid the program and switched two of the statements:

```
Sub Adder()
a = 10
c = a + b
b = 33
End Sub
```

This wouldn't give the desired result because the program performs the addition before assigning a value to **b**. Therefore, **c** would have a value of 10, rather than the desired result: 43.

Although this example is trivial, it serves to make an important point that novice programmers must understand: The computer is like an *idiot savant* who can implement instructions at an incredibly fast rate. However, if your instructions to it (i.e., the program) are ambiguous and illogical, it just won't work properly.

1.2 THE CONCEPT OF ASSIGNMENT

In the previous section, we used terms like "constants," "variables," and "assignment." Let's now make certain that you understand what these terms and their ramifications for computer equations mean.

There are two fundamental ways in which information is represented in a computer language like VBA: directly as constants and symbolically as variables. As the name implies, a *constant* is a value that does not change. In contrast, a *variable* is a symbolic name that can take on different values. They are related by the fact that constants can be assigned to variables, as in Figure 1.1.

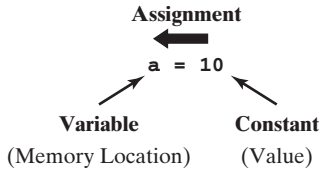


Figure 1.1
A VBA assignment statement in which constants is assigned to a variable.

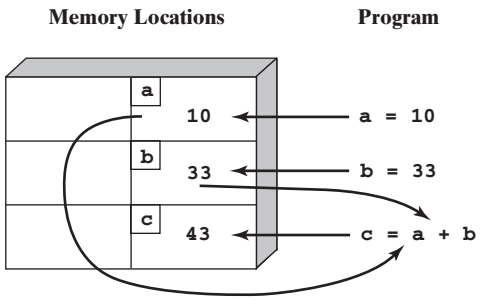


Figure 1.2
Assignment of constants and variables to locations in the computer's memory.

It is useful to understand what the computer actually does when an assignment statement is executed. The variable is the label that the computer associates with the memory location where it stores a constant value. As depicted in Figure 1.2, the first two statements,

```
a = 10
b = 33
```

cause the computer to set up two storage locations in its memory, which it labels **a** and **b**, and into which it stores the constants 10 and 33. The third statement,

```
c = a + b
```

instructs the computer to add the stored values of the variables **a** and **b**, and to assign the result to the variable **c**.

The equals sign in the assignment statement can be thought of as meaning "is replaced by" or "is assigned the result." It is critical to recognize that this use is different from the conventional one wherein the equals sign signifies equality. The distinction is so significant that some computer scientists have actually suggested that a better representation would be an arrow:

```
angle ← 45.
```

That's why languages such as Pascal and MathCad adopted the symbol combination ":=", as in

```
angle := 45.;
```

to explicitly distinguish assignment from equality.

Why is this distinction important? Well, for one thing, assignment statements must always have a single variable on the left side of the equals sign. Because they both violate this rule, the following legitimate algebraic equations would be invalid as VBA statements:

```
a + b = c
3 = x - y + 99
```

On the other hand, although the following statement is not a proper algebraic equation, it is a perfectly valid assignment statement:

```
x = x + 1
```

After this statement was executed, the memory location for the variable **x** would have a value that was one higher than its previous value.

Of course, computers also have to process information other than numbers. For example, they must be able to handle names, labels, and identification numbers, which consist of letters, numbers, and symbols. Such *alphanumerical* information is formally referred to as a *string*. In VBA, *string constants* are enclosed in quotes. The variables in which they are stored are called *string variables*. Here's an example of an assignment of a string constant, "Louis Armstrong", to a string variable, **n**:

```
n = "Louis Armstrong"
```

After this line is executed, a memory location called n will be set up, containing the string constant **"Louis Armstrong"**.

1.3 DECISIONS AND LOOPS

To this point, we understand that a program is a *sequence* of unambiguous instructions that the computer implements one after another. In reality, this statement should be amended to read "A program is a sequence of unambiguous instructions that the computer implements one after another, *unless you instruct it to do otherwise*."

There are two ways in which you can make a program deviate from its sequential mode of operation: decisions and loops. These constructs allow you to vastly increase a program's power to solve numerical problems. In fact, computer scientists have proven that any numerical calculation can be implemented by some combination of sequences, decisions, and loops.

1.3.1 Decisions

As the name implies, a *decision* involves implementing a statement, depending on the outcome of a decision. The *If/Then* statement is the simplest means of implementing decisions in VBA.

Suppose that you want to determine the absolute value of a number. If the number is greater than or equal to zero, you do not have to do anything. However, for a negative number, you would have to make the sign positive. An If/Then statement can be used for this purpose, as in the code

```
If a < 0 Then
  a = -a
End if
```

Because VBA is designed to be easy to understand, the meaning of this If/Then statement is pretty straightforward. In fact, you can read it as if it were an English sentence: "If **a** is less than zero, then set **a** equal to its negative." Because of the nature of assignment, this results in the absolute value being generated and stored in the memory location for **a**.

In addition to generating a single action, a decision can be used to implement one action if a statement is true and another if it is false. Suppose that you want to determine the sign of a number. The following *If/Then/Else* statement can be used for this purpose:

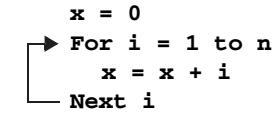
```
If a < 0 Then
  s = -1
Else
  s = 1
End if
```

If **a** is less than zero, **s** will be assigned a value of −1, which is mathematically equivalent to a negative sign. If **a** is not greater than zero (that is it is less than or equal to zero), **s** will be assigned a value of 1.

1.3.2 Loops

A *loop* repeats a VBA statement (or statements) several times. The *For/Next loop* is the simplest way to do this.

Suppose that you want to calculate the sum of the first **n** positive whole numbers. A For/Next loop can be used for this purpose:



```
x = 0
For i = 1 to n
  x = x + i
Next i
```

Because VBA is so easy to understand, you should be able to deduce what's going on here. Suppose that **n** is assigned a value of 5. After **x** is assigned a value of 0, the actual loop begins with the For statement, wherein the variable **i** is initially set to 1. The program then moves to the following line, in which **x + i** is assigned to **x**. Since **i** is equal to 1, **x** would then be equal to 1. At the Next statement, the computer increments **i** by 1 so that **i** becomes 2. The computer then transfers control back to the For statement, where VBA determines whether **i** is greater than **n** (i.e., 5). If so, VBA exits the loop by transferring to the line immediately following the Next statement. If not, it repeats the body of the loop, so that **x** becomes 3. The process is repeated until **i** is greater than **n**. In our example, **x** would be equal to 15 and the loop would be terminated.

1.4 A SIMPLE EXAMPLE

Let's tie together the topics of this chapter by writing a simple program to determine either (*a*) the sum of *n* numbers or (*b*) *n!* (that is, *n* factorial), depending on a decision. First, let's figure out how to use a loop to determine a factorial. Recall that

$$n! = 1 \times 2 \times 3 \times \ldots \times n-1 \times n.$$

A *For/Next loop* can be used for this purpose:

```
x = 1
For i = 1 to n
  x = x * i
Next i
```

Notice how this kind of loop differs from the summation loop we described in the last section. Rather than setting the initial value for **x** to 0, we set it to 1. Otherwise, the loop would always result in **x** = 0. In addition, the interior of the loop performs multiplication, as is appropriate in computing a factorial.

Along with the summation loop, this loop can be integrated into an If/Then/Else decision:

```
calctype = "summation"
n = 5
If calctype = "summation" Then
  x = 0
```

```
    For i = 1 To n
      x = x + i
    Next i
  Else
    x = 1
    For i = 1 To n
      x = x * i
    Next i
  End If
```

Because we have set **calctype** equal to "**summation**", the If/Then statement will be true, and the summation loop will be implemented. Therefore, **x** will be equal to $1 + 2 + 3 + 4 + 5 = 15$. Now, suppose that we set **calctype = "factorial"** and **n** = 4. Then the If/Then statement would be false, and the Else option would be implemented. That is, **x** would be computed as $1 \times 2 \times 3 \times 4 = 24$.

Although this is a pretty simple example, it serves to illustrate how sequences, decisions, and loops can be combined to tell the computer to perform some calculations. If you are new to programming, understanding the material in this chapter is your launching point to understanding how to develop programs of great complexity and value.

KEY TERMS

Alphanumerical	If/Then /Else statement	String
Constant	Loops	String constants
Decision	Program	String variables
For/Next loop	Sequence	Variable
If/Then statement	Statements	

PROBLEMS

1.1 Suppose that we have a cylindrical tank (Figure 1.3) with radius r (m) and height h (m). If the tank is being filled at a flow rate F (m^3/min) for a period of time t (min), write a computer program to determine whether it overtops. The program should consist of the following steps:

- Assign values to the system parameters. In this case, use $r = 5$ m, $h = 10$ m, $F = 15$ m^3/min, and $t = 2$ hr.
- Compute the volume V of the cylinder.
- Compute the volume V_t of water flowing into the tank over time t.
- If the tank overflows, compute the volume of water that is lost, and assign it to the variable V_o. Otherwise, set $V_o = 0$.

Note: x^2 is expressed in VBA as **x ^ 2**.

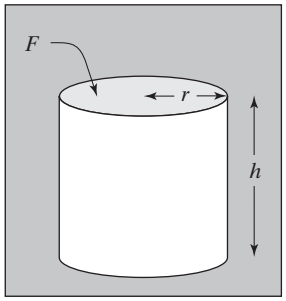


Figure 1.3
A cylindrical tank being filled with a liquid.

1.2 Set up a simple addition program to do the following:

- Assign two values to be added to the variables x and y. (Use 550 and 1679.)
- Assign a value to the variable c_g for a "guess" at the summation. (Use 2133.)
- Compute the summation and assign the result to the variable c.
- If the guess is correct, set the variable $g = 1$. Otherwise, set $g = 0$.

1.3 Write a program to do the following:

- Assign a value to the variable n. For this case set $n = 8$.
- Compute the sum of the squares, ss, of the first n positive whole numbers. After performing the summation, if ss is less than 100, set $ss = 100$.

1.4 Write a program to compute the factorial of an integer, n. Your program should guard against negative values for n.