3

Variables, Input, and Output

3.1 Numbers

Much of the data processed by computers consists of numbers. In computerese, numbers are called **numeric literals**. This section discusses the operations that are performed with numbers and the ways numbers are displayed.

Arithmetic Operations

The five standard arithmetic operations in Visual Basic are addition, subtraction, multiplication, division, and exponentiation. Addition, subtraction, and division are denoted in Visual Basic by the standard symbols +, −, and /, respectively. However, the notations for multiplication and exponentiation differ from the customary mathematical notations as follows:

Mathematical Notation	Visual Basic Notation
$a \cdot b$ or $a \times b$	`a*b`
a^r	`a^r`

(The asterisk [*] is the upper character of the 8 key. The caret [^] is the upper character of the 6 key.)

One way to show a number on the screen is to display it in a list box. If *n* is a number, then the instruction

```
lstBox.Items.Add(n)
```

displays the number *n* as the last item in the list box. *Add* is called a **method**. (Generally, a method is a process that performs a task for a particular object.) If the parentheses contain a combination of numbers and arithmetic operations, the *Add* method carries out the operations and displays the result. Another important method is *Clear*. The statement

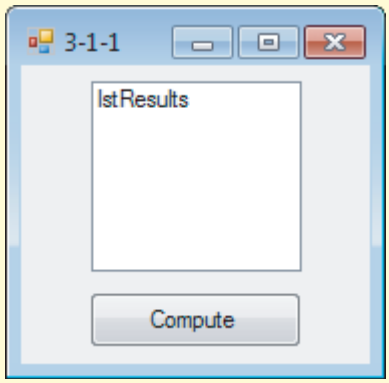
```
lstBox.Items.Clear()
```

removes all the items displayed in the list box lstBox.

Example 1 The following program applies each of the five arithmetic operations. Preceding the program are the form design and a table showing the names of the objects on the form and the settings, if any, for properties of these objects. This form design is also used in the discussion and examples in the remainder of this section.

The word "Run" in the phrasing [Run . . .] indicates that the *Start Debugging* button or F5 should be pressed to execute the program. Notice that in the output 3 / 2 is displayed in decimal form. Visual Basic never displays numbers as fractions. In the evaluation of `2*(3 + 4)`, the operation inside the parentheses is calculated first.

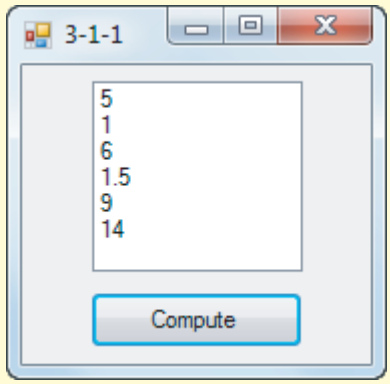
Note: All programs appearing in examples and case studies are provided on the companion website for this book. See the discussion on page xv for details.



OBJECT	PROPERTY	SETTING
frmArithmetic	Text	3-1-1
lstResults		
btnCompute	Text	Compute

```
Private Sub btnCompute_Click(...) Handles btnCompute.Click
  lstResults.Items.Clear()
  lstResults.Items.Add(3 + 2)
  lstResults.Items.Add(3 – 2)
  lstResults.Items.Add(3 * 2)
  lstResults.Items.Add(3 / 2)
  lstResults.Items.Add(3 ^ 2)
  lstResults.Items.Add(2 * (3 + 4))
End Sub
```

[Run, and then click on the button.]



Variables

In applied mathematics problems, quantities are referred to by names. For instance, consider the following high school algebra problem: "If a car travels at 50 miles per hour, how far will it travel in 14 hours? Also, how many hours are required to travel 410 miles?" The solution to this problem uses the well-known formula

$$\text{distance} = \text{speed} \times \text{time elapsed}$$

Here's how this problem would be solved with a computer program:

```
Private Sub btnCompute_Click(...) Handles btnCompute.Click
  Dim speed As Double
```

```
    Dim timeElapsed As Double
    Dim distance As Double
    lstResults.Items.Clear()
    speed = 50
    timeElapsed = 14
    distance = speed * timeElapsed
    lstResults.Items.Add(distance)
    distance = 410
    timeElapsed = distance / speed
    lstResults.Items.Add(timeElapsed)
End Sub
```

[Run, and then click on the button. The following is displayed in the list box.]

```
700
8.2
```

Skip the second, third, and fourth lines of the event procedure for now. We will return to them soon. The sixth line sets the speed to 50, and the seventh line sets the time elapsed to 14. The eighth line multiplies the value for the speed by the value for the time elapsed and sets the distance to this product. The next line displays the answer to the distance-traveled question. The three lines before the End Sub statement answer the time-required question in a similar manner.

The names *speed*, *timeElapsed*, and *distance*, which hold values, are referred to as **variables**. Consider the variable *timeElapsed*. In the seventh line, its value was set to 14. In the eleventh line, its value was changed as the result of a computation. On the other hand, the variable *speed* had the same value, 50, throughout the program.

VideoNote
Numbers

In general, a variable is a name that is used to refer to an item of data. The value assigned to the variable may change during the execution of the program. In Visual Basic, variable names must begin with a letter or an underscore, and can consist only of letters, digits, and underscores. (The shortest variable names consist of a single letter.) Visual Basic does not distinguish between uppercase and lowercase letters used in variable names. Some examples of variable names are *total*, *numberOfCars*, *taxRate_2010*, and *n*. As a convention, we write variable names in lowercase letters except for the first letters of each additional word (as in *gradeOnFirstExam*). This convention is called **camel casing**.

If *var* is a variable and *n* is a numeric literal, then the statement

```
var = n
```

assigns the number *n* to the variable *var*. Such a statement is another example of an **assignment statement**.

A variable is declared to be of a certain type depending on the sort of data that can be assigned to it. The most versatile type for holding numbers is called **Double**. A variable of type Double can hold whole, fractional, or mixed numbers between about $-1.8 \cdot 10^{308}$ and $1.8 \cdot 10^{308}$. Dim statements (also called **declaration statements**) declare the names and types of the variables to be used in the program. The second, third, and fourth lines of this event procedure declare three variables of type Double and give them the names *speed*, *timeElapsed*, and *distance*. Variables must be declared before values can be assigned to them.

In general, a statement of the form

```
Dim varName As Double
```

declares a variable named *varName* to be of type Double. Actually, the Dim statement causes the computer to set aside a location in memory referenced by *varName*. Since *varName* is a

numeric variable, the Dim statement initially places the number zero in that memory location. (We say that zero is the **initial value** or **default value** of the variable.) Each subsequent assignment statement having *varName* to the left of the equal sign will change the value of the number.

The initial value can be set to a value other than zero. To specify a nonzero initial value, follow the declaration statement with an equal sign followed by the initial value. The statement

```
Dim varName As Double = 50
```

declares the specified variable as a variable of type Double and gives it the initial value 50.

The statement

```
lstBox.Items.Add(varName)
```

looks into this memory location for the current value of the variable and displays that value in the list box.

IntelliSense provides assistance with both declaration and assignment statements. Consider the pair of statements

```
Dim interestRate As Double
interestRate = 0.05
```

In the first statement, IntelliSense will suggest the word "As" after you type "Dim interestRate", and will suggest the word "Double" after you type "Dou". In the second statement, IntelliSense will suggest the word "interestRate" after you type "inte".

A combination of literals, variables, and arithmetic operations that can be evaluated to yield a number is called a **numeric expression**. Expressions are evaluated by replacing each variable by its value and carrying out the arithmetic. Some examples of expressions are 2 * distance + 7, $n + 1$, and $(a + b)/3$.

Example 2 The following program displays the default value of a variable and the value of an expression:

```
Private Sub btnCompute_Click(...) Handles btnCompute.Click
  Dim a As Double
  Dim b As Double = 3
  lstResults.Items.Clear()
  lstResults.Items.Add(a)
  lstResults.Items.Add(b)
  a = 5
  lstResults.Items.Add(a * (2 + b))
End Sub
```

[Run, and then click on the button. The following is displayed in the list box.]

```
0
3
25
```

If *var* is a variable, then the assignment statement

```
var = expression
```

first evaluates the expression on the right and *then* assigns its value to the variable on the left. For instance, the event procedure in Example 2 can be written as

```
Private Sub btnCompute_Click(...) Handles btnCompute.Click
  Dim a As Double
  Dim b As Double = 3
  Dim c As Double
  lstResults.Items.Clear()
  lstResults.Items.Add(a)
  lstResults.Items.Add(b)
  a = 5
  c = a * (2 + b)
  lstResults.Items.Add(c)
End Sub
```

The expression a*(2 + b) is evaluated to 25, and then this value is assigned to the variable *c*.

Incrementing the Value of a Variable

Because the expression on the right side of an assignment statement is evaluated *before* an assignment is made, a statement such as

```
var = var + 1
```

is meaningful. It first evaluates the expression on the right (that is, it adds 1 to the value of the variable *var*) and then assigns this sum to the variable *var*. The effect is to increase the value of the variable *var* by 1. In terms of memory locations, the statement retrieves the value of *var* from *var*'s memory location, uses it to compute *var* + 1, and then places the sum back into *var*'s memory location. This type of calculation is so common that Visual Basic provides a special operator to carry it out. The statement `var = var + 1` can be replaced with the statement

```
var += 1
```

In general, if *n* has a numeric value, then the statement

```
var += n
```

adds *n* to the value of *var*.

Built-In Functions: Math.Sqrt, Int, Math.Round

There are several common operations that we often perform on numbers other than the standard arithmetic operations. For instance, we may take the square root of a number or round a number. These operations are performed by built-in functions. Functions associate with one or more values, called the *input*, a single value called the *output*. The function is said to **return** the output value. The three functions considered here have numeric input and output.

The function Math.Sqrt calculates the square root of a number. The function Int finds the greatest integer less than or equal to a number. Therefore, Int discards the decimal part of positive numbers. The value of Math.Round(*n*, *r*) is the number *n* rounded to *r* decimal places. The parameter *r* can be omitted. If so, *n* is rounded to a whole number. Some examples follow:

Math.Sqrt(9) is 3.	Int(2.7) is 2.	Math.Round(2.7) is 3.
Math.Sqrt(0) is 0.	Int(3) is 3.	Math.Round(2.317, 2) is 2.32.
Math.Sqrt(6.25) is 2.5.	Int(−2.7) is −3.	Math.Round(2.317, 1) is 2.3.

The terms inside the parentheses can be numbers (as shown), numeric variables, or numeric expressions. Expressions are first evaluated to produce the input.

Example 3 The following program evaluates each of the functions for a specific input given by the value of the variable *n*:

```
Private Sub btnCompute_Click(...) Handles btnCompute.Click
  Dim n As Double
  Dim root As Double
  n = 6.76
  root = Math.Sqrt(n)
  lstResults.Items.Clear()
  lstResults.Items.Add(root)
  lstResults.Items.Add(Int(n))
  lstResults.Items.Add(Math.Round(n, 1))
End Sub
```

[Run, and then click on the *Compute* button. The following is displayed in the list box.]

```
2.6
6
6.8
```

Example 4 The following program evaluates each of the preceding functions within an expression:

```
Private Sub btnCompute_Click(...) Handles btnCompute.Click
  Dim a As Double
  Dim b As Double
  a = 2
  b = 3
  lstResults.Items.Clear()
  lstResults.Items.Add(Math.Sqrt((5 * b) + 1))
  lstResults.Items.Add(Int((a ^ b) + 0.8))
  lstResults.Items.Add(Math.Round(a / b, 3))
End Sub
```

[Run, and then click on the button. The following is displayed in the list box.]

```
4
8
0.667
```

The Integer Data Type

In this text, we sometimes need to use variables of type Integer. An integer variable is declared with a statement of the form

```
Dim varName As Integer
```

and can be assigned only whole numbers from about −2 billion to 2 billion. Integer variables are commonly used for counting.

Multiple Declarations

Several variables of the same type can be declared with a single Dim statement. For instance, the two Dim statements in Example 2 can be replaced by the single statement

```
Dim a, b As Double
```

Two other types of multiple-declaration statement are

```
Dim a As Double, b As Integer
Dim c As Double = 2, b As Integer = 5
```

Two Integer-Valued Operators

In addition to the five arithmetic operators discussed at the beginning of this section, the **Mod operator** and the **integer division operator (\)** are also useful operators. Let *m* and *n* be positive whole numbers. When you use long division to divide *m* by *n*, you obtain an integer quotient and an integer remainder. In Visual Basic, *m* \ *n* is the integer quotient and *m* Mod *n* is the integer remainder. For instance,

14 \ 3 is 4 and 14 Mod 3 is 2
19 \ 5 is 3 and 19 Mod 5 is 4
10 \ 2 is 5 and 10 Mod 2 is 0.

Example 5 The following program converts 41 inches to 3 feet and 5 inches.

```
Private Sub btnCompute_Click(...) Handles btnCompute.Click
  Dim totalInches, feet, inches As Integer
  totalInches = 41
  feet = totalInches \ 12
  inches = totalInches Mod 12
  lstResults.Items.Add(feet)
  lstResults.Items.Add(inches)
End Sub
```

[Run, and then click on the button. The following is displayed in the list box.]

```
3
5
```

Parentheses

Parentheses should be used when needed to clarify the meaning of an expression. When there are no parentheses, the arithmetic operations are performed in the order exponentiation, multiplication and ordinary division, integer division, Mod, and addition and subtraction. In the event of a tie, the leftmost operation is carried out first. See Table 3.1. ***Note:*** If you use parentheses liberally,

TABLE 3.1 Level of precedence for arithmetic operations.

()	Inner to outer, left to right
^	Left to right in expression
* /	Left to right in expression
\	Left to right in expression
Mod	Left to right in expression
+ −	Left to right in expression

you will not have to rely on the precedence table for arithmetic operations. For instance, write (2*3) + 4 instead of 2*3 + 4. Write (2^3) + 4 instead of 2^3 + 4.

Parentheses cannot be used to indicate multiplication, as is commonly done in algebra. For instance, the expression $x(y + z)$ is not valid. It must be written as $x*(y + z)$.

Three Types of Errors

Grammatical errors, such as misspellings, omissions, or incorrect punctuations, are called **syntax errors**. Most syntax errors are spotted by the Code Editor when they are entered. The editor underlines the syntax error with a blue squiggly line and displays a description of the error when the mouse cursor is hovered over the squiggly line. Some incorrect statements and their errors are as follows:

Statement	*Reason for Error*
lstBox.Itms.Add(3)	The word Items is misspelled.
lstBox.Items.Add(2+)	The number following the plus sign is missing.
Dim m; n As Integer	The semicolon should be a comma.

Errors that occur while a program is running are called **runtime errors** or **exceptions**. They usually occur because something outside the program, such as a user, database, or hard disk, does not behave as expected. For instance, if the file Data.txt is not in the root folder of the C drive, then a statement that refers to the file by the filespec "C:\Data.txt" will cause the program to stop executing and produce a message box with the title

```
FileNotFoundException was unhandled.
```

Also, a yellow arrow will appear at the left side of the line of code that caused the error. At that point, you should end the program.

A third type of error is called a **logic error**. Such an error occurs when a program does not perform the way it was intended. For instance, the line

```
average = firstNum + secondNum / 2
```

is syntactically correct. However, an incorrect value will be generated, since the correct way to calculate the average is

```
average = (firstNum + second Num) / 2
```

Logic errors are the most difficult type to find. Appendix D discusses debugging tools that can be used to detect and correct logic errors.

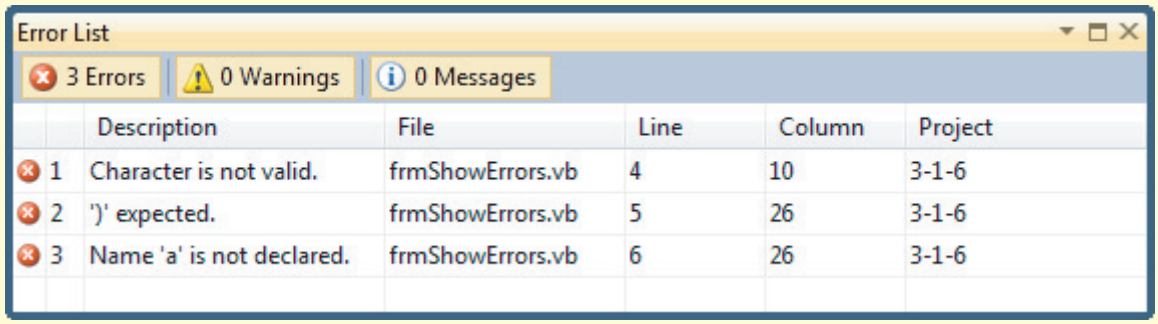
The Error List Window

Syntax errors are not only indicated in the Code Editor, but also are listed in the Error List window. ***Note:*** If the window is not visible, click on *Error List* in the *View/Other Windows* menu.

Example 6 The following program contains three errors. ***Note:*** Line 1 contains the Public Class statement and line 2 is a blank line. Therefore, the Private Sub statement is in line 3 and the Dim statement is in line 4.

```
Private Sub btnCompute_Click(...) Handles btnCompute.Click
  Dim m; n As Double
  lstResults.Items.Add(5
  lstResults.Items.Add(a)
End Sub
```

[Run, click on the button, and click on the *No* button in the error dialog box that appears.]



Comments

1. Declaring variables at the beginning of each event procedure is regarded as good programming practice, because it makes programs easier to read and helps prevent certain types of errors.
2. Keywords (reserved words) cannot be used as names of variables. For instance, the statements `Dim private as Double` and `Dim sub As Double` are not valid.
3. Names given to variables are sometimes referred to as *identifiers*.
4. In math courses, *literals* are referred to as *constants*. However, the word "constant" has a special meaning in programming languages.
5. Numeric literals used in expressions or assigned to variables must not contain commas, dollar signs, or percent signs. Also, mixed numbers, such as 8 1/2, are not allowed.
6. Although requesting the square root of a negative number does not terminate the execution of the program, it can produce unexpected results. For instance, the statement

   ```
   lstBox.Items.Add(Math.Sqrt(-1))
   ```

   displays `NaN`. ***Note:*** NaN is an abbreviation for "Not a Number."
7. If the value of *numVar* is 0 and *numVar* has type Double, then the statements

   ```
   numVarInv = 1 / numVar
   lstBox.Items.Add(numVarInv)
   lstBox.Items.Add(1 / numVarInv)
   ```

   cause the following items to be displayed in the list box:

   ```
   Infinity
   0
   ```
8. When *n* is halfway between two successive whole numbers (such as 1.5, 2.5, 3.5, and 4.5), then it rounds to the nearest even number. For instance, Math.Round (2.5) is 2 and Math.Round (3.5) is 4.
9. In scientific notation, numbers are written in the form $b \cdot 10^r$, where b is a number of magnitude from 1 up to (but not including) 10, and r is an integer. Visual Basic displays very large numbers in **scientific notation**, where $b \cdot 10^r$ is written as bEr. (The letter E is an abbreviation for *exponent*.) For instance, when the statement `lstBox.Items.Add(123 * 10 ^ 15)` is executed, 1.23E+17 is displayed in the list box.
10. If the total number of items added to a list box exceeds the number of items that can be displayed, a vertical scroll bar is automatically added to the list box.
11. When you first enter a statement such as `Dim n As Double`, a green squiggle will appear under the variable name and the Error List window will record a warning. The squiggle

merely indicates that the variable has not yet been assigned a value. If the squiggle is still present after the entire event procedure has been entered, this will tell you that the variable was never used and that the declaration statement should be removed.

Practice Problems 3.1

1. Evaluate 2 + 3 * 4.
2. Explain the difference between the assignment statement

```
var1 = var2
```

and the assignment statement

```
var2 = var1
```

3. Complete the table by filling in the value of each variable after each line is executed.

	a	b	c
`Private Sub btnEvaluate_Click(...) Handles btnEvaluate.Click`			
`  Dim a, b, c As Double`	0	0	0
`  a = 3`	3	0	0
`  b = 4`	3	4	0
`  c = a + b`			
`  a = c * a`			
`  lstResults.Items.Add(a – b)`			
`  b = b * b`			
`End Sub`			

4. Write a statement that increases the value of the numeric variable *var* by 5%.

EXERCISES 3.1

In Exercises 1 through 6, evaluate the numeric expression without the computer, and then use Visual Basic to check your answer.

1. 3*4
2. 7^2
3. 1/(2^3)
4. 3 + (4*5)
5. (5 − 3)*4
6. 3*((−2)^5)

In Exercises 7 through 10, evaluate the expression.

7. 7\3
8. 14 Mod 4
9. 7 Mod 3
10. 14\4

In Exercises 11 through 16, determine whether the name is a valid variable name.

11. sales.2008
12. room&Board
13. fOrM_1040
14. 1040B
15. expenses?
16. INCOME 2008

In Exercises 17 through 22, evaluate the numeric expression where a = 2, *b* = 3, and *c* = 4.

17. (a*b) + c
18. a*(b + c)
19. (1 + b)*c
20. a^c
21. b^(c − a)
22. (c − a)^b

In Exercises 23 through 28, write an event procedure to calculate and display the value of the expression.

23. $7 \cdot 8 + 5$

24. $(1 + 2 \cdot 9)^3$

25. 5.5% of 20

26. $15 - 3(2 + 3^4)$

27. $17(3 + 162)$

28. $4\ 1/2 - 3\ 5/8$

In Exercises 29 and 30, complete the table by filling in the value of each variable after each line is executed.

29.

	x	y
`Private Sub btnEvaluate Click(...) Handles btnEvaluate.Click`		
`  Dim x, y As Double`		
`  x = 2`		
`  y = 3 * x`		
`  x = y + 5`		
`  lstResults.Items.Clear()`		
`  lstResults.Items.Add(x + 4)`		
`  y = y + 1`		
`End Sub`		

30.

	bal	inter	withDr
`Private Sub btnEvaluate_Click(...) Handles btnEvaluate.Click`			
`  Dim bal, inter, withDr As Double`			
`  bal = 100`			
`  inter = 0.05`			
`  withDr = 25`			
`  bal += inter * bal`			
`  bal = bal – withDr`			
`End Sub`			

In Exercises 31 through 38, determine the output displayed in the list box by the lines of code.

31.
```
Dim amount As Double
amount = 10
lstOutput.Items.Add(amount – 4)
```

32.
```
Dim a, b As Integer
a = 4
b = 5 * a
lstOutput.Items.Add(a + b)
```

33.
```
Dim n As Integer = 7
n += 1
lstOutput.Items.Add(1)
lstOutput.Items.Add(n)
lstOutput.Items.Add(n + 1)
```

34.
```
Dim num As Integer = 5
num = 2 * num
lstOutput.Items.Add(num)
```

35.
```
Dim a, b As Integer
lstOutput.Items.Add(a + 1)
a = 4
b = a * a
lstOutput.Items.Add(a * b)
```

36.
```
Dim tax As Double
tax = 200
tax = 25 + tax
lstOutput.Items.Add(tax)
```

37.
```
Dim totalMinutes, hours, minutes As Integer
totalMinutes = 135
hours = totalMinutes \ 60
minutes = totalMinutes Mod 60
lstResults.Items.Add(hours)
lstResults.Items.Add(minutes)
```

38.
```
Dim totalOunces, pounds, ounces As Integer
totalOunces = 90
pounds = totalOunces \ 16
ounces = totalOunces Mod 16
lstResults.Items.Add(pounds)
lstResults.Items.Add(ounces)
```

In Exercises 39 through 44, identify the errors.

39.
```
Dim a, b, c As Double
a = 2
b = 3
a + b = c
lstOutput.Items.Add(c)
```

40.
```
Dim a, b, c, d As Double
a = 2
b = 3
c = d = 4
lstOutput.Items.Add(5((a + b) / (c + d)
```

41.
```
Dim balance, deposit As Double
balance = 1,234
deposit = $100
lstOutput.Items.Add(balance + deposit)
```

42.
```
Dim interest, balance As Double
0.05 = interest
balance = 800
lstOutput.Items.Add(interest * balance)
```

43.
```
Dim 9W As Double
9W = 2 * 9W
lstOutput.Items.Add(9W)
```

44.
```
Dim n As Double = 1.2345
lstOutput.Items.Add(Round(n, 2))
```

In Exercises 45 and 46, rewrite the code using one line.

45.
```
Dim quantity As Integer
quantity = 12
```

46.
```
Dim m As Integer
Dim n As Double
m = 2
n = 3
```

In Exercises 47 through 52, find the value of the given function.

47. Int(10.75) **48.** Int(9 − 2) **49.** Math.Sqrt(3 * 12)
50. Math.Sqrt(64) **51.** Math.Round(3.1279, 3) **52.** Math.Round(−2.6)

In Exercises 53 through 58, find the value of the given function where *a* and *b* are numeric variables of type Double, *a* = 5 and *b* = 3.

53. Int(−a / 2) **54.** Math.Round(a / b) **55.** Math.Sqrt(a − 5)
56. Math.Sqrt(4 + a) **57.** Math.Round(a + .5) **58.** Int(b * 0.5)

In Exercises 59 through 66, write an event procedure with the header `Private Sub btnCompute_Click(...) Handles btnCompute.Click`, and having one line for each step. Lines that display data should use the given variable names.

59. The following steps calculate a company's profit:

(a) Declare all variables as type Double.
(b) Assign the value 98456 to the variable *revenue*.
(c) Assign the value 45000 to the variable *costs*.
(d) Assign the difference between the variables *revenue* and *costs* to the variable *profit*.
(e) Display the value of the variable *profit* in a list box.

60. The following steps calculate the amount of a stock purchase:

(a) Declare all variables as type Double.
(b) Assign the value 25.625 to the variable *costPerShare*.
(c) Assign the value 400 to the variable *numberOfShares*.
(d) Assign the product of *costPerShare* and *numberOfShares* to the variable *amount*.
(e) Display the value of the variable *amount* in a list box.

61. The following steps calculate the price of an item after a 30% reduction:

(a) Declare all variables as type Double.
(b) Assign the value 19.95 to the variable *price*.
(c) Assign the value 30 to the variable *discountPercent*.
(d) Assign the value of (*discountPercent* divided by 100) times *price* to the variable *markdown*.
(e) Decrease *price* by *markdown*.
(f) Display the value of *price* (rounded to two decimal places) in a list box.

62. The following steps calculate a company's break-even point, the number of units of goods the company must manufacture and sell in order to break even:

(a) Declare all variables as type Double.
(b) Assign the value 5000 to the variable *fixedCosts*.
(c) Assign the value 8 to the variable *pricePerUnit*.
(d) Assign the value 6 to the variable *costPerUnit*.

(e) Assign the value *fixedCosts* divided by (the difference of *pricePerUnit* and *costPerUnit)* to the variable *breakEvenPoint*.
(f) Display the value of the variable *breakEvenPoint* in a list box.

63. The following steps calculate the balance after three years when $100 is deposited in a savings account at 5% interest compounded annually:

(a) Declare all variables as type Double.
(b) Assign the value 100 to the variable *balance*.
(c) Increase the variable *balance* by 5% of its value.
(d) Increase the variable *balance* by 5% of its value.
(e) Increase the variable *balance* by 5% of its value.
(f) Display the value of *balance* (rounded to two decimal places) in a list box.

64. The following steps calculate the balance at the end of three years when $100 is deposited at the beginning of each year in a savings account at 5% interest compounded annually:

(a) Declare all variables as type Double.
(b) Assign the value 100 to the variable *balance*.
(c) Increase the variable *balance by* 5% of its value, and add 100.
(d) Increase the variable *balance* by 5% of its value, and add 100.
(e) Increase the variable *balance* by 5% of its value.
(f) Display the value of *balance* (rounded to two decimal places) in a list box.

65. The following steps calculate the balance after 10 years when $100 is deposited in a savings account at 5% interest compounded annually:

(a) Declare all variables as type Double.
(b) Assign the value 100 to the variable *balance*.
(c) Multiply the variable *balance* by 1.05 raised to the 10th power.
(d) Display the value of *balance* (rounded to two decimal places) in a list box.

66. The following steps calculate the percentage profit from the sale of a stock:

(a) Declare all variables as type Double.
(b) Assign the value 10 to the variable *purchasePrice*.
(c) Assign the value 15 to the variable *sellingPrice*.
(d) Assign, to the variable *percentProfit*, 100 times the value of the difference between *sellingPrice* and *purchasePrice* divided by *purchasePrice*.
(e) Display the value of the variable *percentProfit* in a list box.

In Exercises 67 through 72, write a program to solve the problem and display the answer in a list box. The program should use variables for each of the quantities.

67. Suppose each acre of farmland produces 18 tons of corn. How many tons of corn can be produced on a 30-acre farm?

68. Suppose a ball is thrown straight up in the air with an initial velocity of 50 feet per second and an initial height of 5 feet. How high will the ball be after 3 seconds?

Note: The height after t seconds is given by the expression $-16t^2 + v_0t + h_0$, where v_0 is the initial velocity and h_0 is the initial height.

69. If a car left Washington, D.C., at 2 o'clock and arrived in New York at 7 o'clock, what was its average speed? ***Note:*** New York is 233 miles from Washington.

70. A motorist wants to determine her gas mileage. At 23,352 miles (on the odometer) the tank is filled. At 23,695 miles the tank is filled again with 14 gallons. How many miles per gallon did the car average between the two fillings?

71. A U.S. geological survey showed that Americans use an average of 1600 gallons of water per person per day, including industrial use. How many gallons of water are used each year in the United States? ***Note:*** The current population of the United States is about 315 million people.

72. According to FHA specifications, each room in a house should have a window area equal to at least 10% of the floor area of the room. What is the minimum window area for a 14-ft by 16-ft room?

Solutions to Practice Problem 3.1

1. 14. Multiplications are performed before additions. If the intent is for the addition to be performed first, the expression should be written (2 + 3)*4.
2. The first assignment statement assigns the value of the variable *var2* to the variable *var1*, whereas the second assignment statement assigns *var1*'s value to *var2*.
3.

	a	b	c
`Private Sub btnEvaluate_Click(...) Handles btnEvaluate.Click`			
`Dim a, b, c As Double`	0	0	0
`a = 3`	3	0	0
`b = 4`	3	4	0
`c = a + b`	3	4	7
`a = c * a`	21	4	7
`lstResults.Items.Add(a – b)`	21	4	7
`b = b * b`	21	16	7
`End Sub`			

Each time an assignment statement is executed, only one variable (the variable to the left of the equal sign) has its value changed.

4. Each of the three following statements increases the value of *var* by 5%.

```
var = var + (0.05 * var)
var = 1.05 * var
var += 0.05 * var
```

3.2 Strings

The most common types of data processed by Visual Basic are numbers and strings. Sentences, phrases, words, letters of the alphabet, names, telephone numbers, addresses, and social security numbers are all examples of strings. Formally, a **string literal** is a sequence of characters that is treated as a single item. String literals can be assigned to variables, displayed in text boxes and list boxes, and combined by an operation called concatenation (denoted by &).

VideoNote
Strings

Variables and Strings

A **string variable** is a name used to refer to a string. The allowable names of string variables are the same as those of numeric variables. The value of a string variable is assigned or altered with assignment statements and displayed in a list box like the value of a numeric variable. String variables are declared with statements of the form

```
Dim varName As String
```