

Solutions to Practice Problems 6.1

1. As a rule of thumb, the condition is checked at the bottom if the loop should be executed at least once.
2. Either precede the loop with the statement `continue = "Yes"`, or change the first line to `Do` and replace the `Loop` statement with `Loop Until continue <> "Yes"`.

6.2 For . . . Next Loops

When we know exactly how many times a loop should be executed, a special type of loop, called a `For . . . Next` loop, can be used. `For . . . Next` loops are easy to read and write and they have features that make them ideal for certain common tasks. The following code uses a `For . . . Next` loop to display a table:

```
Private Sub btnDisplayTable_Click(...) Handles btnDisplayTable.Click
    'Display a table of the first 5 numbers and their squares
    'Assume the font for lstTable is Courier New
    For i As Integer = 1 To 5
        lstTable.Items.Add(i & " " & i ^ 2)
    Next
End Sub
```

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

```
1  1
2  4
3  9
4 16
5 25
```

A similar program written with a `Do` loop is as follows.

```
Private Sub btnDisplayTable_Click(...) Handles btnDisplayTable.Click
    'Display a table of the first 5 numbers and their squares
    Dim i As Integer
    i = 1
    Do While i <= 5
        lstTable.Items.Add(i & " " & i ^ 2)
        i += 1      'Add 1 to i
    Loop
End Sub
```

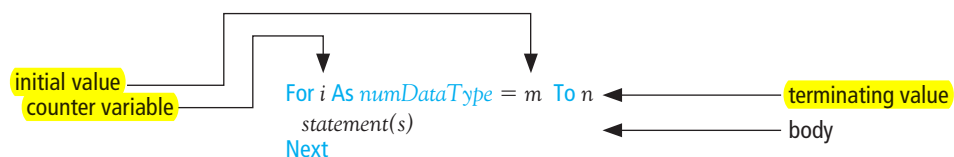


VideoNote

For . . . Next loops

General Form of a For . . . Next Loop

In general, a portion of a program of the form



constitutes a `For . . . Next` loop. The pair of statements `For` and `Next` cause the statements between them to be repeated a specified number of times. The `For` statement declares a numeric variable, called the **counter variable**, that is initialized and then automatically changes after

each pass through the loop. Also, the For statement gives the range of values this variable will assume. The Next statement increments the counter variable. If $m \leq n$, then i is assigned the values $m, m + 1, \dots, n$ in order, and the body is executed once for each of these values. If $m > n$, then the body is skipped and execution continues with the statement after the For...Next loop.

When program execution reaches a For...Next loop, such as the one shown previously, the For statement assigns to the counter variable i the initial value m and checks to see whether i is greater than the terminating value n . If so, then execution jumps to the line following the Next statement. If $i \leq n$, the statements inside the loop are executed. Then, the Next statement increases the value of i by 1 and checks this new value to see if it exceeds n . If not, the entire process is repeated until the value of i exceeds n . When this happens, the program moves to the line following the loop. Figure 6.5 contains the pseudocode and flowchart of a For...Next loop.

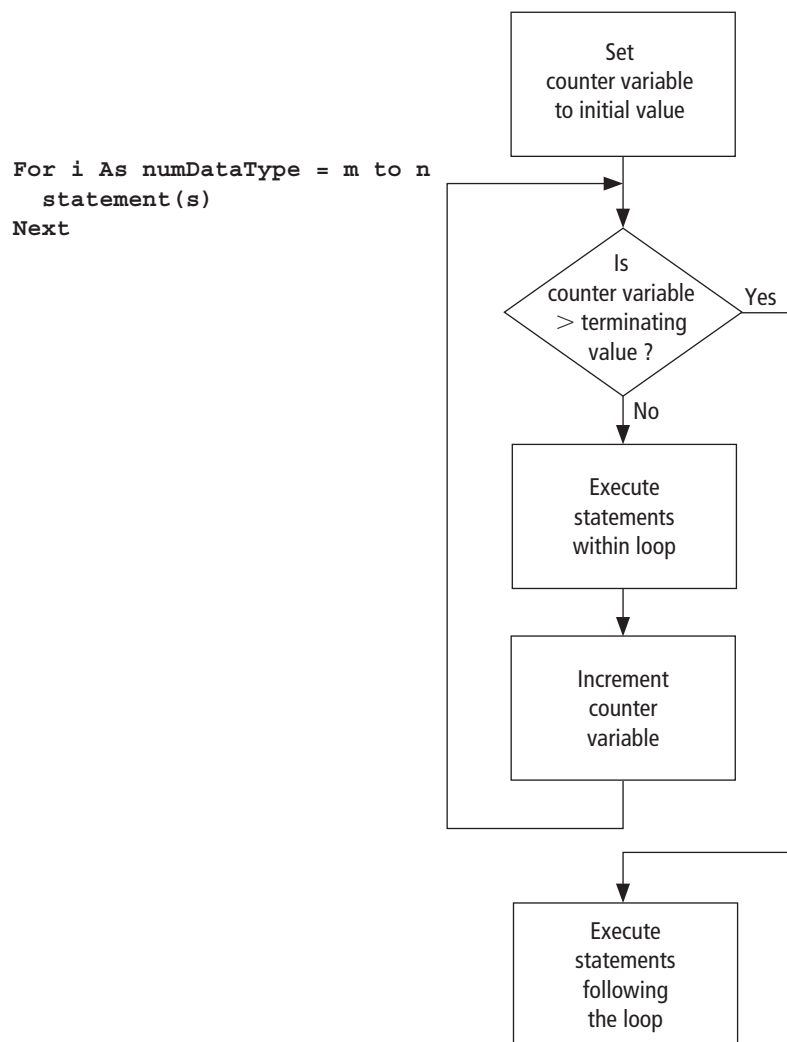


FIGURE 6.5 Pseudocode and flowchart of a For...Next loop.

The counter variable can be any numeric variable. The most common single-letter names are i, j , and k ; however, if appropriate, the name should suggest the purpose of the counter variable.

The counter variable, and any variable declared inside a For...Next loop, has **block-level scope**; that is, the variable cannot be referred to by code outside of the loop.

A counter variable also can be declared with a Dim statement outside of the For...Next loop. For instance, the following program produces the same output as the program shown at the beginning of this section.

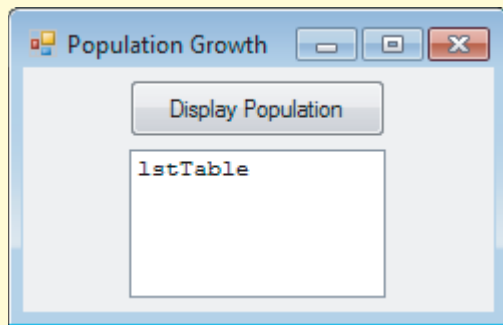
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
    'Display a table of the first 5 numbers and their squares
    'Assume the font for lstTable is Courier New
    Dim i As Integer
    For i = 1 To 5
        lstTable.Items.Add(i & " " & i ^ 2)
    Next
End Sub
```

In this case, the variable *i* does not have block-level scope, and therefore it violates the principle that the scope of a variable should be as small as possible. In this book, we never declare a counter variable outside a For ... Next loop.



Example 1

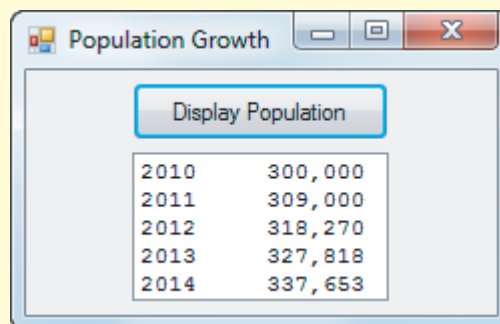
Suppose the population of a city is 300,000 in the year 2010 and is growing at the rate of 3% per year. The following program displays a table showing the population each year until 2014.



OBJECT	PROPERTY	SETTING
frmPopulation	Text	Population Growth
btnDisplay	Text	Display Population
lstTable	Font	Courier New

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
    'Display population from 2010 to 2014
    Dim pop As Double = 300000
    For yr As Integer = 2010 To 2014
        lstTable.Items.Add(yr & " " & FormatNumber(pop, 0))
        pop += 0.03 * pop
    Next
End Sub
```

[Run, and click on the button.]



The initial and terminating values can be literals, variables, or expressions. For instance, the For statement in the preceding program can be replaced by

```
Dim firstYr As Integer = 2010
Dim lastYr As Integer = 2014
For yr As Integer = firstYr To lastYr
```

In Example 1, the counter variable was increased by 1 after each pass through the loop. A variation of the For statement allows any number to be used as the increment. The statement

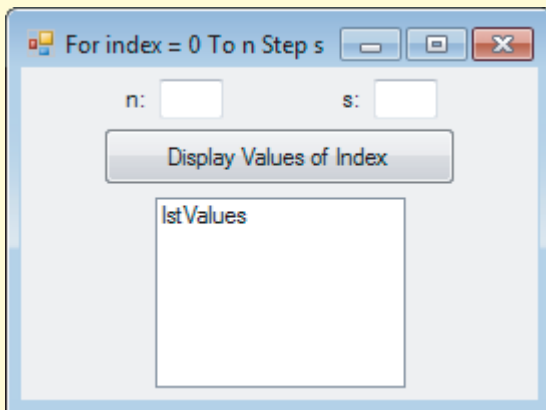
```
For i As numDataType = m To n Step s
```

instructs the Next statement to add *s* to the counter variable instead of 1. The numbers *m*, *n*, and *s* do not have to be whole numbers. The number *s* is called the **step value of the loop**. **Note 1:** If the counter variable will assume values that are not whole numbers, then the variable must be of type Double. **Note 2:** The counter variable is also called the **index**.



Example 2

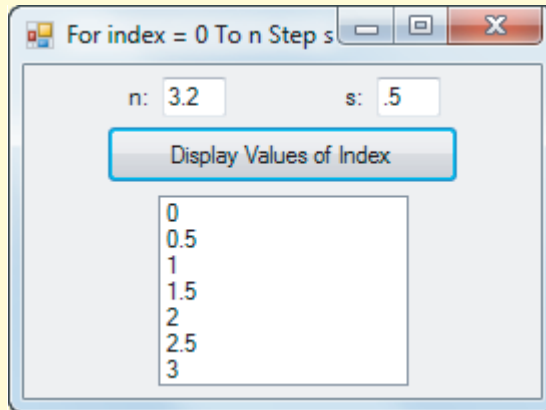
The following program displays the values of the index of a For ... Next loop before terminating and the step values input by the user:



OBJECT	PROPERTY	SETTING
frmIndex	Text	For index = 0 To n Step s
lblN	Text	n:
txtEnd	Text	
lblS	Text	s:
txtStep	Text	
btnDisplay	Text	Display Values of Index
lstValues		

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
    'Display values of index ranging from 0 to n Step s
    Dim n, s As Double
    n = Cdbl(txtEnd.Text)
    s = Cdbl(txtStep.Text)
    lstValues.Items.Clear()
    For index As Double = 0 To n Step s
        lstValues.Items.Add(index)
    Next
End Sub
```

[Run, type 3.2 and .5 into the text boxes, and click on the button.]

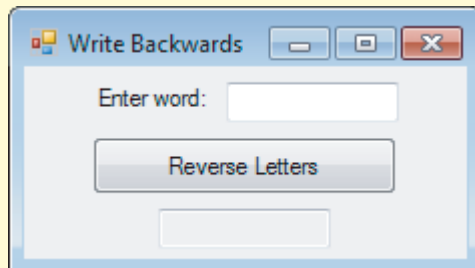


In the examples considered so far, the counter variable was successively increased until it reached the terminating value. However, if a negative step value is used and the initial value is greater than the terminating value, then the counter value is decreased until reaching the terminating value. In other words, the loop counts backward.



Example 3

The following program accepts a word as input and displays it backwards:

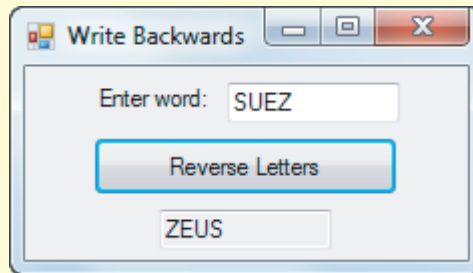


OBJECT	PROPERTY	SETTING
frmBackwards	Text	Write Backwards
lblWord	Text	Enter word:
txtWord		
btnReverse	Text	Reverse Letters
txtBackwards	ReadOnly	True

```
Private Sub btnReverse_Click(...) Handles btnReverse.Click
    txtBackwards.Text = Reverse(txtWord.Text)
End Sub
```

```
Function Reverse(ByVal info As String) As String
    Dim m As Integer, temp As String = ""
    m = info.Length
    For j As Integer = m - 1 To 0 Step -1
        temp &= info.Substring(j, 1)
    Next
    Return temp
End Function
```

[Run, type “SUEZ” into the text box, and click on the button.]



Note: The initial and terminating values of a For . . . Next loop can be expressions. For instance, the third and fourth lines of the function in Example 3 can be consolidated to

```
For j As Integer = info.Length - 1 To 0 Step -1
```

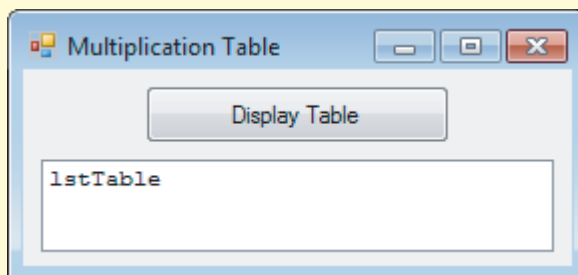
■ Nested For . . . Next Loops

The body of a For . . . Next loop can contain any sequence of Visual Basic statements. In particular, it can contain another For . . . Next loop. However, the second loop must be completely contained inside the first loop and must have a different counter variable. Such a configuration is called **nested** For . . . Next loops.



Example 4

The following program displays a multiplication table for the integers from 1 to 3. Here j denotes the left factors of the products, and k denotes the right factors. Each factor takes on a value from 1 to 3. The values are assigned to j in the outer loop (lines 4–11) and to k in the inner loop (lines 6–9). Initially, j is assigned the value 1, and then the inner loop is traversed three times to produce the first row of products. At the end of these three passes, the value of j will still be 1, and the first execution of the inner loop will be complete. Following this, the statement Next increments the value of j to 2. The statement beginning “For k ” is then executed. It resets the value of k to 1. The second row of products is displayed during the next three executions of the inner loop, and so on.

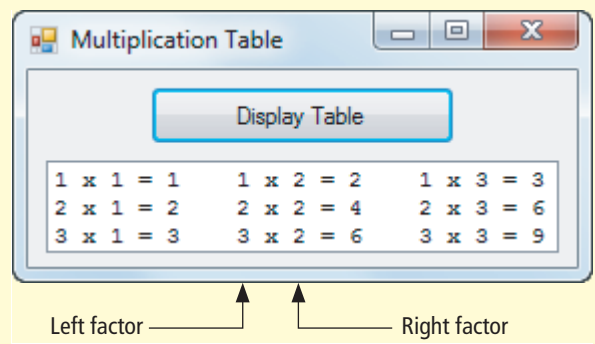


OBJECT	PROPERTY	SETTING
frmTable	Text	Multiplication Table
btnDisplay	Text	Display Table
lstTable	Font	Courier New

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
    Dim row, entry As String
    lstTable.Items.Clear()
    For j As Integer = 1 To 3
        row = ""
        For k As Integer = 1 To 3
```

```
entry = j & " x " & k & " = " & (j * k)
row &= entry & "    "
Next
lstTable.Items.Add(row)
Next
End Sub
```

[Run, and click on the button.]



■ Local Type Inference

Local type inference (also referred to as *implicit typing*) allows you to declare and initialize a local variable without explicitly stating its type with an `As` clause. Local type inference is enabled if `Option Infer` is set to `On` in the VB Defaults dialog box shown in Figure 3.1 of Section 3.2. (It is enabled by default.)

Some examples of the use of local type inference are as follows:

Standard Declaration	Local Type Inference Equivalent
For i As Integer = 1 To 3	For i = 1 To 3
For i As Double = 1 To 3 Step 0.5	For i = 1 To 3 Step 0.5
Dim count As Integer = 5	Dim count = 5
Dim rate as Double = 0.05	Dim rate = 0.05
Dim name As String = "Fred"	Dim name = "Fred"
Dim d as Date = #6/4/2010#	Dim d = #6/4/2010#

With local type inference, the type of a local variable is determined by the values following the equal sign. If the values are all whole numbers (written without a decimal point) and in the range of values for Integers, then the variable is declared to be of type Integer. If any of the values are numbers containing a decimal point or are outside the range of values for Integers, then the variable is declared to be of type Double. If the value is surrounded in quotes, then it is declared to be of type String. If the value is a date literal, then the variable is declared to be of type Date. **Note:** This feature *does not apply to class-level variables*.

The following walkthrough (which assumes that `Option Infer` is `On`) demonstrates how local type inference works in a `For ... Next` loop.

1. Create a new program consisting of a form having a button (`btnConfirm`) and a list box (`lstBox`).
2. Enter the following code.

```
Private Sub btnConfirm_Click(...) Handles btnConfirm.Click
    For i = 1 To 5 Step 2
```

```
lstBox.Items.Add(i)
Next
End Sub
```

3. In the Code Editor, hover the mouse pointer over the variable *i*. (The tooltip `Dim i As Integer` appears to confirm that the variable *i* has indeed been declared as type Integer.)
4. In the header of the For ... Next loop, change the `5 to 5.0` and again hover the pointer over the letter *i*. (This time the tooltip reads `Dim i As Double`.)

Local type inference was added to Visual Basic because it is needed for LINQ (Language INtegrated Query), an innovative language feature that unifies the manipulation of diverse collections of data. LINQ is introduced in Chapter 7 of this book and is used extensively from then on. By necessity, we rely on local type inference when using LINQ. Although we do not use local type inference in the declaration of ordinary variables, ~~you may feel free to do so if you prefer and your instructor permits.~~

Local type inference is also known as **duck typing**. This name comes from the well-known quote, “If it walks like a duck, and quacks like a duck, then it is a duck.”

■ Comments

1. For and Next statements must be paired. If one is missing, the syntax checker will complain with a wavy underline and a message such as “A ‘For’ must be paired with a ‘Next’.”
2. Consider a loop beginning with For *i* = *m* To *n* Step *s*. The loop will be executed exactly once if *m* equals *n* no matter what value *s* has. The loop will not be executed at all if *m* is greater than *n* and *s* is positive, or if *m* is less than *n* and *s* is negative.
3. The value of the counter variable should not be altered within the body of the loop; doing so might cause the loop to repeat indefinitely or have an unpredictable number of repetitions.
4. Noninteger Step values in For ... Next loops can result in unexpected outcomes. For instance, if you run Example 2 with *n* = 2 and *s* = .1, the last number displayed will not be 2 as intended. In general, the use of counter variables of type Double is poor programming practice and should be avoided. From now on, all counter variables appearing in this book will have type Integer.
5. Visual Basic provides a way to abort an iteration in a For ... Next loop. When the statement **Continue For** is encountered in the body of the loop, execution immediately jumps to the Next statement. An analogous statement **Continue Do** is available for Do loops. Typically, Continue For and Continue Do statements appear inside conditional structures such as If blocks.
6. Visual Basic provides a way to back out of a For ... Next loop. When the statement **Exit For** is encountered in the body of the loop, execution jumps immediately to the statement following the Next statement.
7. Counter variables and variables declared inside For ... Next loops have block-level scope; that is, they cannot be referred to by code outside the loops.
8. Any type of loop can be nested inside another loop. For example, For ... Next loops can be nested inside Do loops and vice versa. Also, Do loops can be nested inside other Do loops.

Practice Problem 6.2

1. Why won't the following lines of code work as intended?

```
For i As Integer = 15 To 1
    lstBox.Items.AddItem(i)
Next
```

2. When is a For ... Next loop more appropriate than a Do loop?

EXERCISES 6.2

In Exercises 1 through 10, determine the output displayed in the list box when the button is clicked.

1.

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
    For i As Integer = 1 To 4
        lstBox.Items.Add("Pass #" & i)
    Next
End Sub
```
2.

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
    For i As Integer = 3 To 6
        lstBox.Items.Add(2 * i)
    Next
End Sub
```
3.

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
    For j As Integer = 2 To 8 Step 2
        lstBox.Items.Add(j)
    Next
    lstBox.Items.Add("Who do we appreciate?")
End Sub
```
4.

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
    For countdown As Integer = 10 To 1 Step -1
        lstBox.Items.Add(countdown)
    Next
    lstBox.Items.Add("blastoff")
End Sub
```
5.

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
    Dim num As Integer = 5
    For i As Integer = num To (2 * num - 3)
        lstBox.Items.Add(i)
    Next
End Sub
```
6.

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
    For i As Integer = -9 To -1 Step 3
        lstBox.Items.Add(i)
    Next
End Sub
```
7.

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
    'Chr(149) is a large dot
    Dim stringOfDots As String = ""
    For i As Integer = 1 To 10
        stringOfDots &= Chr(149)
    Next
    txtBox.Text = stringOfDots
End Sub
```

8. Private Sub btnDisplay_Click(...) Handles btnDisplay.Click

```

    Dim n As Integer = 3
    Dim total As Integer = 0
    For i As Integer = 1 To n
        total += i
    Next
    txtBox.Text = CStr(total)
End Sub

```

9. Private Sub btnDisplay_Click(...) Handles btnDisplay.Click

```

'Note: Chr(65) is A and Chr(90) is Z
Dim sentence, letter As String
Dim numCaps As Integer = 0
sentence = "The United States of America"
For i As Integer = 0 To sentence.Length - 1
    letter = sentence.Substring(i, 1)
    If (Asc(letter) >= 65) And (Asc(letter) <= 90) Then
        numCaps += 1
    End If
Next
txtBox.Text = CStr(numCaps)
End Sub

```

10. Private Sub btnDisplay_Click(...) Handles btnDisplay.Click

```

Dim word As String = "courage"
Dim letter As String = ""
Dim numVowels As Integer = 0
For i As Integer = 0 To word.Length - 1
    letter = word.Substring(i, 1)
    If IsVowel(letter) Then
        numVowels += 1
    End If
Next
txtBox.Text = CStr(numVowels)
End Sub

```

```

Function IsVowel(ByVal letter As String) As Boolean
    letter = letter.ToUpper
    If (letter = "A") Or (letter = "E") Or (letter = "I") Or
        (letter = "O") Or (letter = "U") Then
        Return True
    Else
        Return False
    End If
End Function

```

In Exercises 11 through 14, identify the errors.

11. Private Sub btnDisplay_Click(...) Handles btnDisplay.Click

```

    For j As Integer = 1 To 25 Step -1
        lstBox.Items.Add(j)
    Next
End Sub

```

```
12. Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
    For i As Integer = 1 To 3
        lstBox.Items.Add(i & " " & 2 ^ i)
    End Sub
```

```
13. Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
    'Display all numbers from 0 through 20 except for 13
    For i As Integer = 20 To 0
        If i = 13 Then
            i = 12
        End If
        lstBox.Items.Add(i)
    Next
End Sub
```

```
14. Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
    For j As Integer = 1 To 4 Step 0.5
        lstBox.Items.Add(j)
    Next
End Sub
```

In Exercises 15 and 16, rewrite the program using a For . . . Next loop.

```
15. Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
    Dim num As Integer = 1
    Do While num <= 9
        lstBox.Items.Add(num)
        num += 2    'Add 2 to value of num
    Loop
End Sub
```

```
16. Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
    lstBox.Items.Add("hello")
    lstBox.Items.Add("hello")
    lstBox.Items.Add("hello")
    lstBox.Items.Add("hello")
End Sub
```

In Exercises 17 through 37, write a program containing a For . . . Next loop to carry out the stated task.

17. Display the even numbers from 1 through 100 in a list box.

18. Find the sum of the first one hundred positive integers.

19. Find the average of five numbers obtained from the user with input dialog boxes.

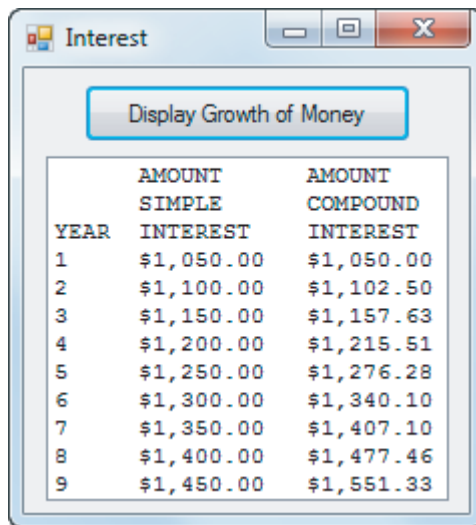
20. Find the largest of five numbers obtained from the user with input dialog boxes.

21. Find the value of $1 + 1/2 + 1/3 + 1/4 + \cdots + 1/100$.

22. Ask the user to input a positive integer (call it n) and then display a string of n large dots. **Note:** Chr(149) is a large dot. Strings of dots can be used to create histograms, as in Fig. 6.11 on page 242.

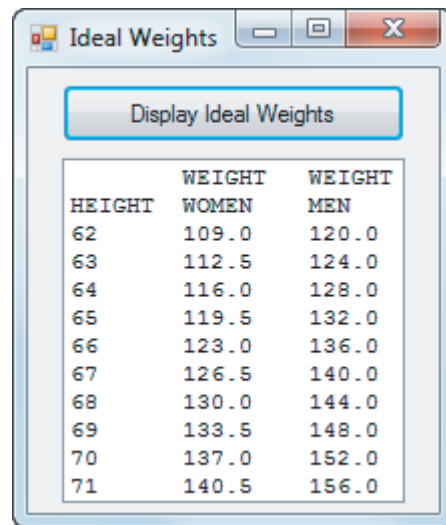
23. *Automobile Depreciation.* A rule of thumb states that cars in personal use depreciate by 15% each year. Suppose a new car is purchased for \$20,000. Produce a table showing the value of the car at the end of each of the next five years.

24. Accept a word as input and determine if its letters are in alphabetical order. (Test the program with the words “almost”, “imply”, and “biopsy”.)
25. Estimate how much a young worker will make before retiring at age 65. Request the worker’s name, age, and starting salary as input. Assume the worker receives a 5% raise each year. For example, if the user enters Helen, 25, and 20000, then the text box should display the following:
- Helen will earn about \$2,415,995.**
26. When \$1000 is invested at 5% simple interest, the amount grows by \$50 each year. When money is invested at 5% interest compounded annually, the amount at the end of each year is 1.05 times the amount at the beginning of that year. Display the amounts for 9 years for a \$1000 investment at 5% simple and compound interest. See Fig. 6.6.



	AMOUNT SIMPLE INTEREST	AMOUNT COMPOUND INTEREST
YEAR		
1	\$1,050.00	\$1,050.00
2	\$1,100.00	\$1,102.50
3	\$1,150.00	\$1,157.63
4	\$1,200.00	\$1,215.51
5	\$1,250.00	\$1,276.28
6	\$1,300.00	\$1,340.10
7	\$1,350.00	\$1,407.10
8	\$1,400.00	\$1,477.46
9	\$1,450.00	\$1,551.33

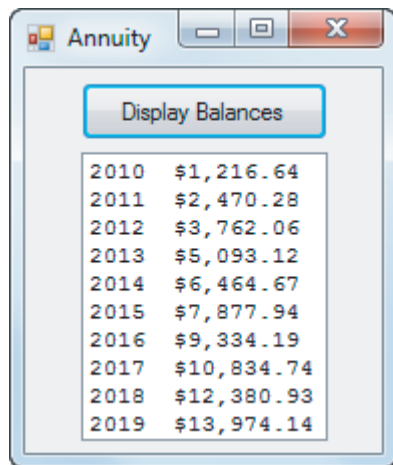
FIGURE 6.6 Output of Exercise 26.



HEIGHT	WEIGHT WOMEN	WEIGHT MEN
62	109.0	120.0
63	112.5	124.0
64	116.0	128.0
65	119.5	132.0
66	123.0	136.0
67	126.5	140.0
68	130.0	144.0
69	133.5	148.0
70	137.0	152.0
71	140.5	156.0

FIGURE 6.7 Possible output for Exercise 27.

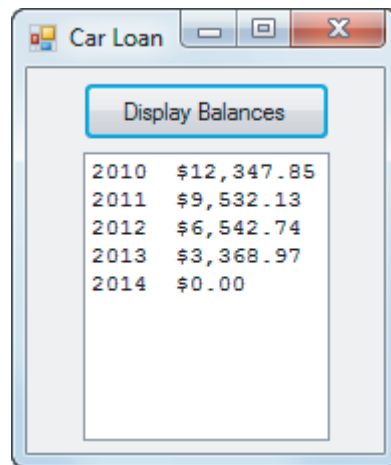
27. According to researchers at Stanford Medical School, the ideal weight for a woman is found by multiplying her height in inches by 3.5 and subtracting 108. The ideal weight for a man is found by multiplying his height in inches by 4 and subtracting 128. Request a lower and upper bound for heights and then produce a table giving the ideal weights for women and men in that height range. For example, when a lower bound of 62 and an upper bound of 71 are specified, Fig. 6.7 shows the output displayed in the list box.
28. Request a sentence, and then determine the number of sibilants (that is, letters S or Z) in the sentence. Carry out the counting with a Function procedure.
29. Refer to the annuity discussed in Exercise 36 of Section 6.1. Assume that the first deposit is made at the end of January 2010, and display the balances in the account at the end of each year from 2010 to 2019. See Fig. 6.8 on the next page.
30. Consider the car loan discussed in Exercise 35 of Section 6.1. The loan will be paid off after five years. Assume that the car was purchased at the beginning of January 2010, and display the balance at the end of each year for five years. See Fig. 6.9. **Note:** The last payment will be slightly less than the other payments, since otherwise the final balance would be a negative amount.
31. *Radioactive Decay.* Cobalt 60, a radioactive form of cobalt used in cancer therapy, decays over a period of time. Each year, 12% of the amount present at the beginning of the year will have decayed. If a container of cobalt 60 initially contains 10 grams, determine the amount remaining after five years.



Display Balances

2010	\$1,216.64
2011	\$2,470.28
2012	\$3,762.06
2013	\$5,093.12
2014	\$6,464.67
2015	\$7,877.94
2016	\$9,334.19
2017	\$10,834.74
2018	\$12,380.93
2019	\$13,974.14

FIGURE 6.8 Output of Exercise 29.



Display Balances

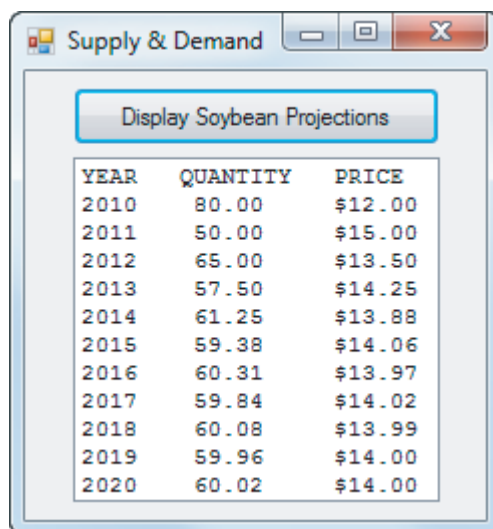
2010	\$12,347.85
2011	\$9,532.13
2012	\$6,542.74
2013	\$3,368.97
2014	\$0.00

FIGURE 6.9 Output of Exercise 30.

- 32.** The keyboard in use on nearly all computers is known as the Qwerty keyboard, since the letters in the top letter line read QWERTYUIOP. A word is called a Qwerty word if all its letters appear on the top letter line of the keyboard. Some examples are *typewriter*, *repertoire*, and *treetop*. Accept a word as input and determine whether or not it is a Qwerty word. Use a Boolean-valued function named `IsQwerty` that accepts a word as input.
- 33.** *Supply and Demand.* Each year's level of production and price (per bushel) for most agricultural products affects the level of production and price for the following year. Suppose the soybean crop in a country was 80 million bushels in 2010 and

$$\begin{aligned} \text{[price each year]} &= 20 - .1 * \text{[quantity that year]} \\ \text{[quantity each year]} &= 5 * (\text{[price from the preceding year]}) - 10, \end{aligned}$$

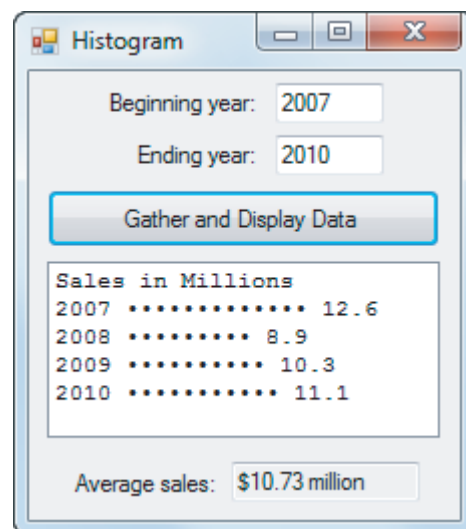
where quantity is measured in units of millions of bushels. Generate a table to show the quantity and price from now until 2020. See Fig. 6.10.



Display Soybean Projections

YEAR	QUANTITY	PRICE
2010	80.00	\$12.00
2011	50.00	\$15.00
2012	65.00	\$13.50
2013	57.50	\$14.25
2014	61.25	\$13.88
2015	59.38	\$14.06
2016	60.31	\$13.97
2017	59.84	\$14.02
2018	60.08	\$13.99
2019	59.96	\$14.00
2020	60.02	\$14.00

FIGURE 6.10 Output of Exercise 33.



Beginning year: 2007
Ending year: 2010

Gather and Display Data

Sales in Millions

2007	12.6
2008	8.9
2009	10.3
2010	11.1

Average sales: \$10.73 million

FIGURE 6.11 Possible output of Exercise 34.

- 34.** Display a company's sales figures for several years in a histogram and calculate the average yearly sales. See Fig. 6.11. When the user clicks on the button, the amount of sales for each year should be obtained from the user with input dialog boxes. **Note:** Chr(149) is a large dot.

- 35.** You are offered two salary options for ten days of work. Option 1: \$100 per day. Option 2: \$1 the first day, \$2 the second day, \$4 the third day, and so on, with the amount doubling each day. Determine which option pays better.

Exercises 36 and 37 should use the following Function procedure.

```
Function DayOfWeek(ByVal d As Date) As String
    Dim str As String = FormatDateTime(d, DateFormat.LongDate)
    Dim n As Integer = str.IndexOf(",")
    Return str.Substring(0, n)
End Function
```

- 36.** Request a year as input and then display the date of the first Tuesday of that year.
- 37.** Request a year as input and then display the dates of the first Tuesdays of each month of that year.

Solutions to Practice Problem 6.2

- The loop will never be entered because 15 is greater than 1. The intended first line might have been

```
For i As Integer = 15 To 1 Step -1
```

or

```
For i As Integer = 1 To 15
```
- If the exact number of times the loop will be executed is known before entering the loop, then a For...Next loop should be used. Otherwise, a Do loop is more appropriate.

6.3 List Boxes and Loops

In previous sections we used list boxes to display output and to facilitate selection. In this section we explore some additional features of list boxes and use loops to analyze data in list boxes.

Some Properties, Methods, and Events of List Boxes

During run time, the value of

`lstBox.Items.Count`

is the number of items currently in the list box. Each item in `lstBox` is identified by an **index number** ranging from 0 through `lstBox.Items.Count - 1`. For instance, if the list box contains 10 items, then the first item has index 0, the second item has index 1, ..., and the last item has index 9. In general, the n th item in a list box has index $n - 1$.

During run time, the user can highlight an item in a list box by clicking on the item with the mouse or by moving to it with the up- and down-arrow keys when the list box has the focus. The `SelectedIndexChanged` event occurs each time an item of a list box is clicked on or each time an arrow key is used to change the highlighted item. It is the default event for list box controls.

The value of

`lstBox.SelectedIndex`

is the index number of the item currently highlighted in `lstBox`. If no item is highlighted, the value of `SelectedIndex` is `-1`. The statement

```
lstBox.SelectedIndex = -1
```

will unhighlight any highlighted item in the list. **Note:** This statement also raises the `SelectedIndexChanged` event.



VideoNote

List boxes
and loops