

**Introducing the  
For...Next Repetition  
Statement  
NumericUpDown Control and  
Multiline TextBox**

## App Requirements

*You're considering investing \$1,000.00 in a savings account that yields 2% interest compounded annually, and you want to forecast how your investment will grow. Assuming that you leave all interest on deposit, calculate and display the amount of money in the account at the end of each year over a period of  $n$  years. To compute these amounts, use the following formula:*

$$a = p(1 + r)^n$$

*where*

$p$  is the original amount of money invested (the principal)

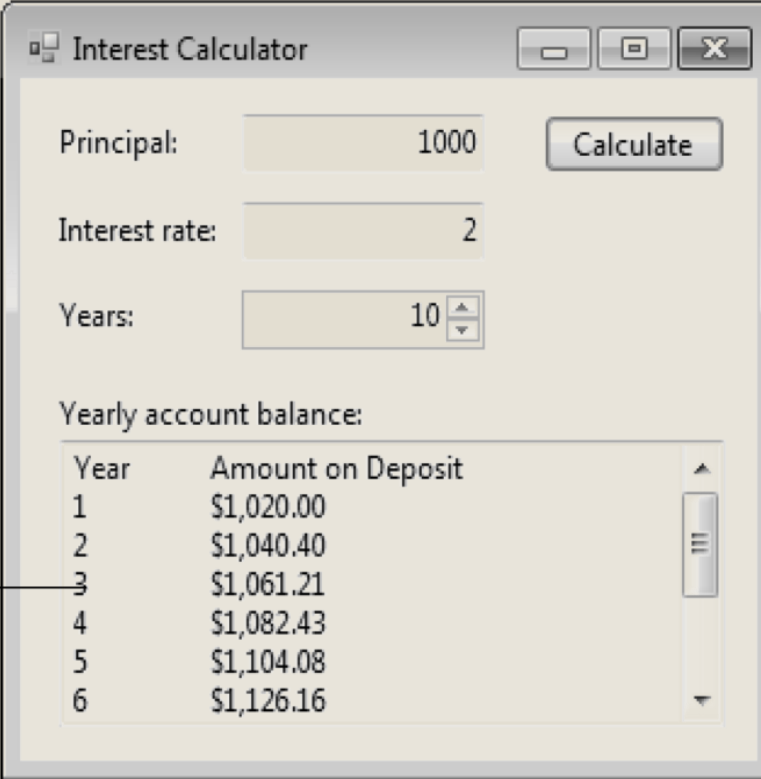
$r$  is the annual interest rate (for example, .02 is equivalent to 2%)

$n$  is the number of years

$a$  is the amount on deposit at the end of the  $n$ th year.

# Test-Driving the Interest Calculator App

- Input some values into the completed app and then click calculate (Fig. 11.2).



The screenshot shows a window titled "Interest Calculator" with standard Windows window controls. It contains three input fields: "Principal" with the value "1000", "Interest rate" with the value "2", and "Years" with the value "10" and a spinner control. A "Calculate" button is positioned to the right of the Principal field. Below the input fields, a section titled "Yearly account balance:" contains a table with two columns: "Year" and "Amount on Deposit". The table lists values for years 1 through 6.

Year	Amount on Deposit
1	\$1,020.00
2	\$1,040.40
3	\$1,061.21
4	\$1,082.43
5	\$1,104.08
6	\$1,126.16

Multiline TextBox displays  
app results

**Figure 11.2** Output of completed Interest Calculator app.

# 11.2 Essentials of the Counter-Controlled Repetition

- The four essential elements of counter-controlled repetition are:
  - the *name* of a *control variable* (or loop counter)
  - the *initial value* of the control variable
  - the *increment* (or *decrement*) by which the control variable is modified during each iteration
  - the *condition* that tests for the *final value* of the control variable
- Figure 11.3 is an example of counter-controlled repetition.

---

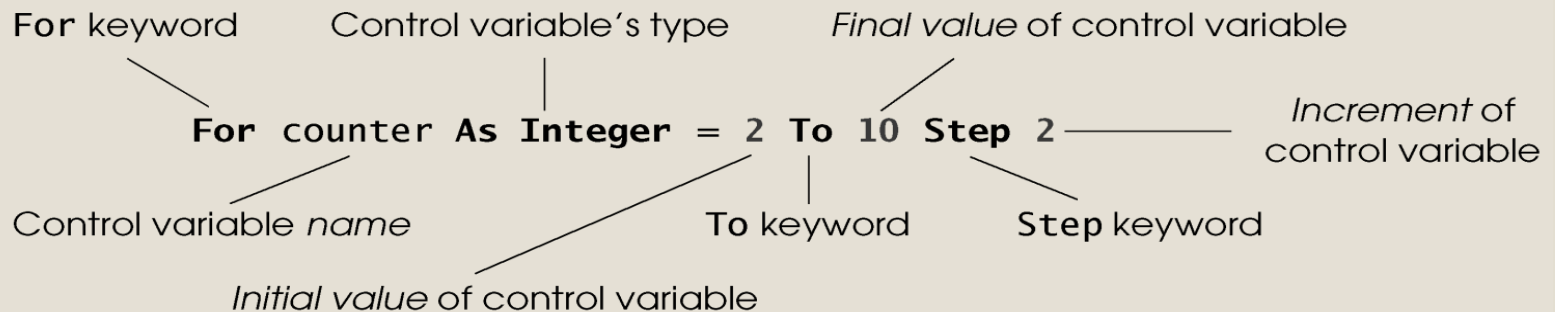
```
1  Dim years As Integer = 2 ' control variable
2
3  Do While years <= 5
4      months = 12 * years ' calculate payment period
5
6      ' calculate payment value
7      monthlyPayment =
8          Pmt(monthlyInterest, months, -loanAmount)
9
10     ' display payment value
11     paymentsListBox.Items.Add(months & ControlChars.Tab &
12         ControlChars.Tab & String.Format("{0:C}", monthlyPayment))
13
14     years += 1 ' increment counter
15 Loop
```

---

**Figure 11.3** Counter-controlled repetition example.

# 11.3 Introducing the For...Next Repetition Statement

- The **For...Next** repetition statement makes it easier for you to write code to perform counter-controlled repetition.
- The **For...Next** statement takes less time to code and is easier to read.
- The **For...Next header** (Fig. 11.4) specifies all four essential elements of counter controlled loops.



**Figure 11.4** For...Next header components.

# 11.3 Introducing the For...Next Repetition Statement (Cont.)

```
For counter As Integer = 2 To 10 Step 2  
    body statement(s)  
Next
```

- A For...Next statement begins with the keyword **For**.
- The statement declares and initializes a control variable.
  - Note that you do not use the **Dim** keyword.
- Following the initial value of the control variable is the keyword **To**, followed by the final value of the control variable.

## 11.3 Introducing the For...Next Repetition Statement (Cont.)

- The **Step** keyword specifies the amount by which to change the control variable each time the loop executes.
  - If you omit the **Step** keyword, the control variable increments by 1 (one) after each repetition.
- The body of a For...Next statement is placed after the For...Next header.
- The keyword **Next** marks the end of the For...Next repetition statement.





## Common Programming Error

---

Attempting to access the control variable (when it's declared in a **For...Next** header) in code after the loop results in a compilation error, because the variable no longer exists.

---

```
1 For years As Integer = 2 To 5
2     months = 12 * years ' calculate payment period
3
4     ' calculate payment value
5     value = Pmt(monthlyRate, months, -loanAmount)
6
7     ' display payment value
8     paymentsListBox.Items.Add(months & ControlChars.Tab &
9         ControlChars.Tab & String.Format("{0:C}", value))
10 Next
```

---

**Figure 11.6** Code segment for the **Car Payment Calculator** app that demonstrates the **For...Next** statement.



## **Common Programming Error**

---

Counter-controlled loops should not be controlled with floating-point variables. These are represented only approximately in the computer's memory, possibly resulting in imprecise counter values and inaccurate tests for termination that could lead to logic errors.



## **Error-Prevention Tip**

---

If you use a `For...Next` loop for counter-controlled repetition, off-by-one errors (which occur when a loop executes for one more or one less iteration than is necessary) are normally avoided, because the terminating value is clear.

# 11.3 Introducing the For...Next Repetition Statement (Cont.)

- Visual Basic provides a feature called **local type inference** that enables it to infer a local variable's type based on the context in which the variable is initialized.

```
Dim x = 7
```

- The compiler infers that the variable x should be of type Integer, because whole-number values, like 7, are of type Integer.

```
Dim y = -123.45
```

- The compiler infers that the variable y should be of type Double, because floating-point number values, like -123.45 are of type Double.

## 11.3 Introducing the For...Next Repetition Statement (Cont.)

- You can also use local type inference with control variables in the header of a For...Next statement.

```
For years = 2 To 5
```

- The local type inference feature is one of several new Visual Basic features that support Language Integrated Query (LINQ).

# 11.4 Examples Using the For...Next Statement

- The following examples demonstrate different ways of varying the control variable in a For...Next statement.

- Vary the control variable from 1 to 100 in increments of 1.

**For i As Integer = 1 To 100**

or

**For i As Integer = 1 To 100 Step 1**

- Vary the control variable from 100 to 1 in increments of -1 (decrements of 1).

**For i As Integer = 100 To 1 Step -1**

# 11.4 Examples Using the For...Next Statement (Cont.)

- Vary the control variable from 7 to 77 in increments of 7.

**For i As Integer = 7 To 77 Step 7**

- Vary the control variable from 20 to -20 in increments of -2 (decrements of 2).

**For i As Integer = 20 To -20 Step -2**

- Vary the control variable over the sequence of the following values: 2, 5, 8, 11, 14, 17, 20.

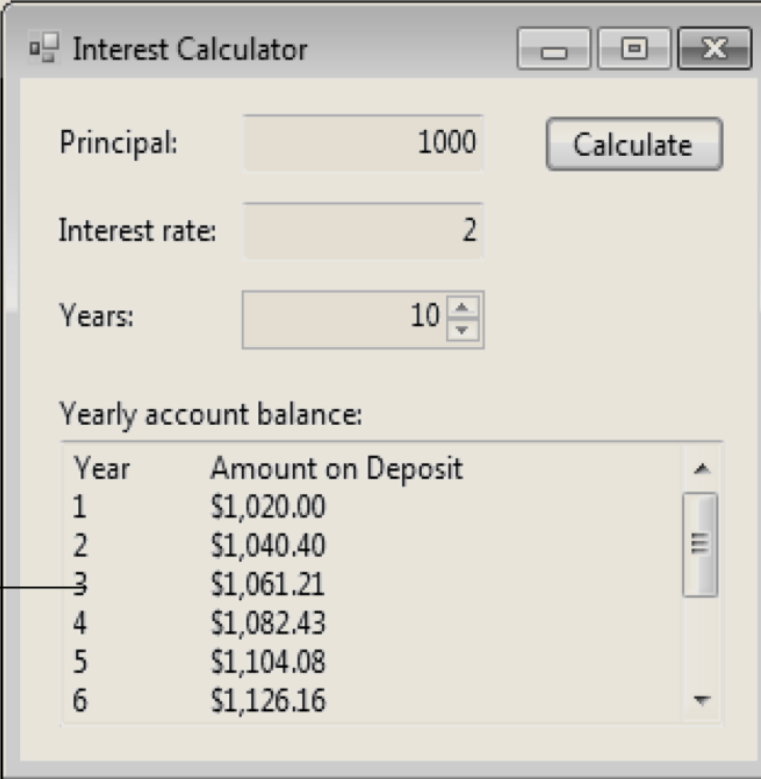
**For i As Integer = 2 To 20 Step 3**

- Vary the control variable over the sequence of the following values: 99, 88, 77, 66, 55, 44, 33, 22, 11, 0.

**For i As Integer = 99 To 0 Step -11**

# 11.5 Constructing the Interest Calculator Application

- Input some values into the completed app and then click calculate (Fig. 11.2).



The screenshot shows a window titled "Interest Calculator" with three input fields: "Principal:" with the value "1000", "Interest rate:" with the value "2", and "Years:" with the value "10". A "Calculate" button is located to the right of the Principal field. Below the input fields, a section titled "Yearly account balance:" contains a table with two columns: "Year" and "Amount on Deposit". The table lists values for years 1 through 6.

Year	Amount on Deposit
1	\$1,020.00
2	\$1,040.40
3	\$1,061.21
4	\$1,082.43
5	\$1,104.08
6	\$1,126.16

Multiline TextBox displays  
app results

**Figure 11.2** Output of completed Interest Calculator app.

# 11.5 Constructing the Interest Calculator Application

When the user clicks the Calculate Button

Get the values for the principal, interest rate and years entered by the user

Store a header String to be added to the output TextBox

For each year (starting at 1 and ending with the number of years entered)

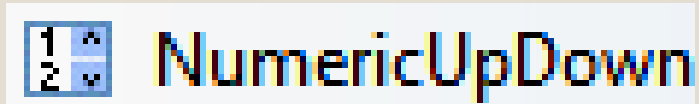
Calculate the current value of the investment

Append the year and the current value of the investment to the String that will be displayed in the output TextBox

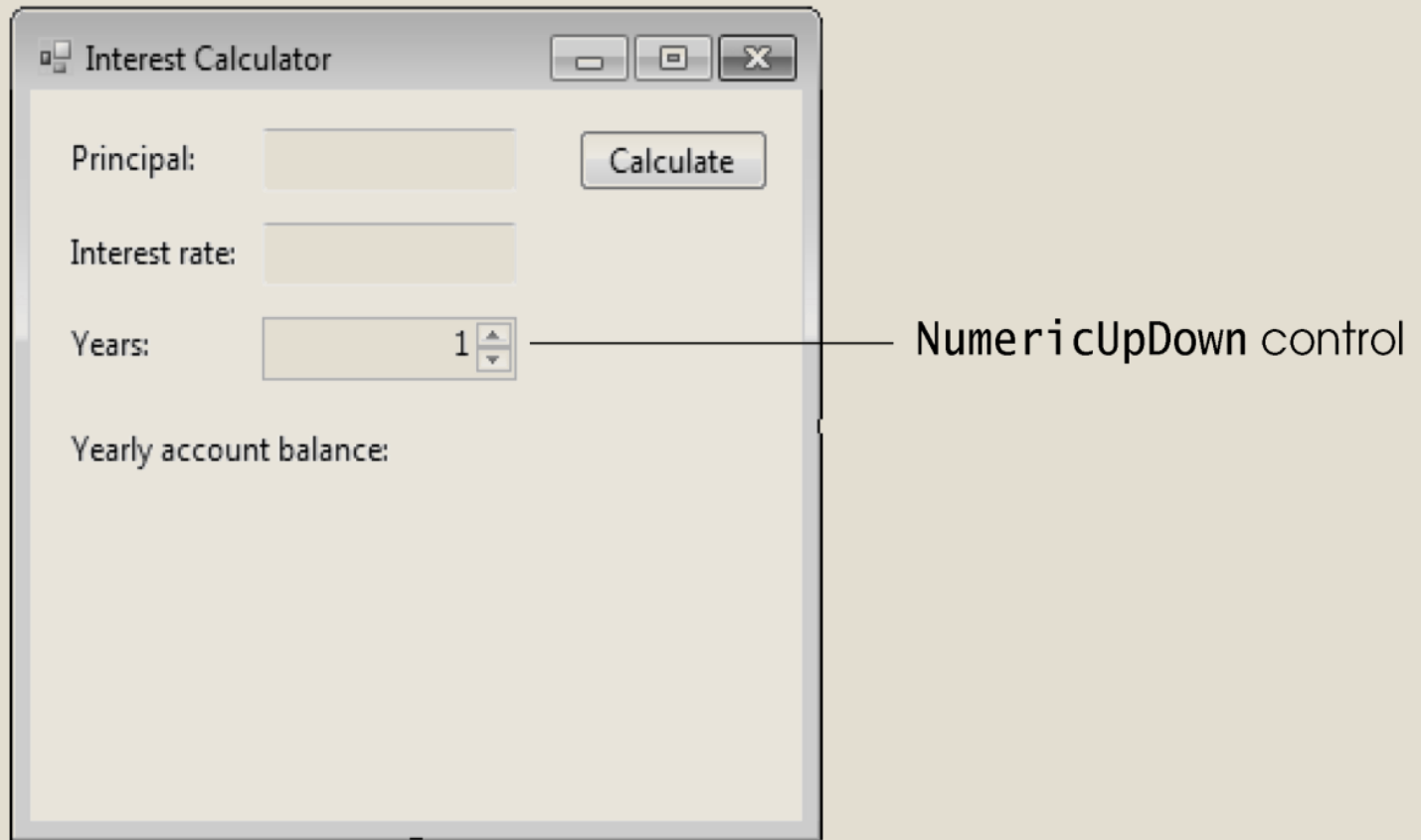


# Adding and Customizing a NumericUpDown Control

- Add a `NumericUpDown` control to the Form (Fig. 11.9), and change the Name property to `yearUpDown`.



- Set `yearUpDown`'s `Location` property to 91, 82 and its `width` property to 100. Set property `TextAlign` to `Right`.



**Figure 11.9** NumericUpDown control added to Interest Calculator app.

# Adding and Customizing a NumericUpDown Control (Cont.)

- By default, this control sets 0 as the minimum and 100 as the maximum.
  - To change these values, set the **Minimum** property of the **Years: NumericUpDown** control to 1 and its **Maximum** property to 10.
  - If the user inputs a value less than **Minimum** or greater than **Maximum**, the value is automatically set to the minimum or maximum value.
- The **Increment** property specifies by how much the current number in the **NumericUpDown** control changes when the user clicks the control's up or down arrow.
  - This app uses the default **Increment** property value, 1.



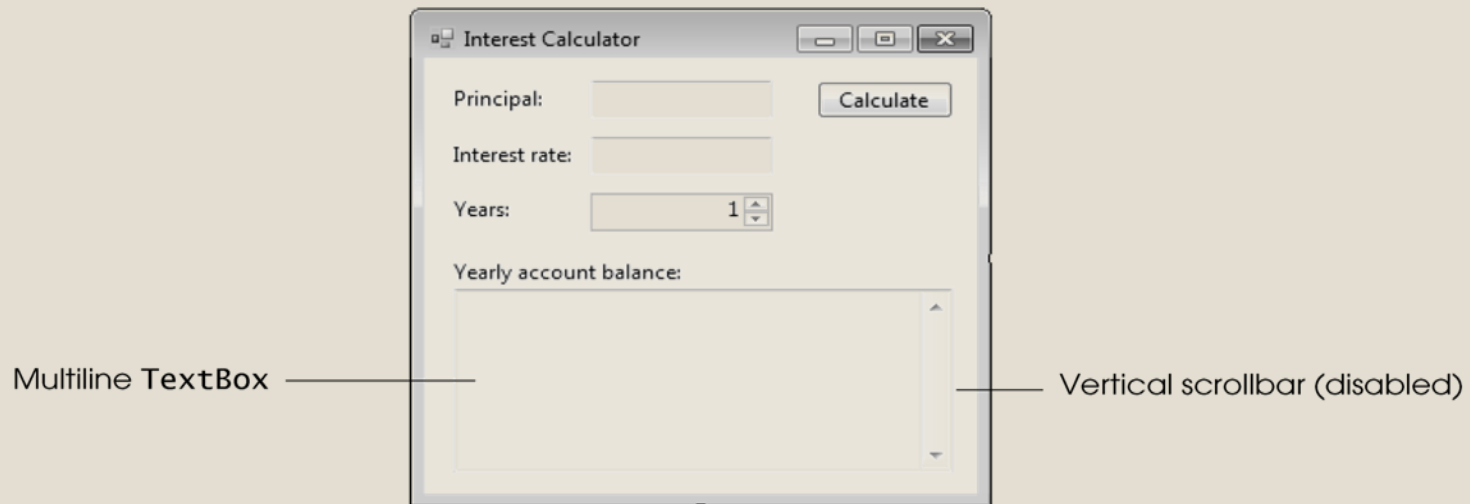
## GUI Design Tip

---

Use a **NumericUpDown** control to limit the range of user input.

# Adding and Customizing a Multiline TextBox with a Scrollbar

- Add a TextBox named resultTextBox (Fig. 11.10).
  - Change the **TextBox's** Multiline property value to True.
  - Set the **ReadOnly** property to True.



**Figure 11.10** Multiline TextBox with vertical scrollbar added to the Form.



## GUI Design Tip

---

If a `TextBox` will display multiple lines of output, set the `Multiline` property to `True` and left align the output by setting the `TextAlign` property to `Left` (its default value).



## GUI Design Tip

---

If a `TextBox` is used to display output, set the `ReadOnly` property to `True` to ensure that the user cannot change the output.



## GUI Design Tip

---

If a multiline `TextBox` will display many lines of output, limit the `TextBox` height and use a vertical scrollbar to allow users to view additional lines of output.

# Adding and Customizing a Multiline TextBox with a Scrollbar

- Using scrollbars allows you to keep the size of a TextBox small while still allowing the user to view all the information in that TextBox.
  - Set resultTextBox's **ScrollBars** property to **Vertical**.
  - You can also set property **ScrollBars** to **Horizontal** or **Both**.
  - A scrollbar is enabled only when it is needed.
  - Without scrollbars, the user can scroll through the text by using the arrow keys.

# String variables

- **String** variables (Fig. 11.11) store a series of characters.
  - The most commonly used characters are letters and numbers.
  - There are also many special characters, such as \$, \*, ^, tabs and newlines.
  - Labels and TextBoxes both store values in the Text property as values of type String.

```

2      ' handles Calculate Button's Click event
3      Private Sub calculateButton_Click(sender As System.Object,
4          e As System.EventArgs) Handles calculateButton.Click
5
6          ' declare variables to store user input
7          Dim principal As Decimal ' store principal
8          Dim rate As Double ' store interest rate
9          Dim amount As Decimal ' store each calculation
10         Dim output As String ' store output
11
12         ' retrieve user input
13         principal = Val(principalTextBox.Text)
14         rate = Val(rateTextBox.Text)

```

Input variable declarations

Retrieve user input

**Figure 11.11** App code for retrieving and storing user input.



# String variables (cont)

- This TextBox will display the results in two columns, labeled Year and Amount on Deposit (Fig. 11.12).

	16	' set output header
Appending header text	17	[ output = "Year" & ControlChars.Tab &
to the output String	18	

**Figure 11.12** App code for appending the header text to the String variable.

# String variables (cont)

- When assigning new text to a `String` variable, you must begin and end the text with a double quotation mark (`"`).
  - You can append a `String` or a character to the end of another `String` by using the concatenation operator (`&`).
- The `ControlChars.Tab` constant inserts a tab character.
- The `ControlChars.CrLf` constant inserts a new line.

# Calculating Cumulative Interest with a For...Next Statement

- The For...Next statement (Fig. 11.13) executes its body once for each year up to the value of yearUpDown's **Value** property.
- Lines 24-25 append text to the end of output, using the **&= operator**.

Using the For...Next statement to format and append text to the output String

```
20 ' calculate amount after each year and append to string
21 For yearCounter As Integer = 1 To yearUpDown.Value
22     amount =
23         principal * ((1 + rate / 100) ^ yearCounter)
24     output &= (yearCounter & ControlChars.Tab &
25         String.Format("{0:C}", amount) & ControlChars.CrLf)
26 Next
```

**Figure 11.13** App code for the For...Next statement.

# Calculating Cumulative Interest with a For . . . Next Statement (Cont.)

- The For . . . Next statement executes until the control variable exceeds the number of years specified by the user.
- After exiting the For . . . Next statement, output is ready to be displayed to the user in resultTextBox (Fig. 11.14).
- Your app can now calculate and display the amount on deposit for each year.

Displaying in the	28	resultTextBox.Text = output ' display result
multiline TextBox the	29	End Sub ' calculateButton_Click
result of the calculations	30	End Class ' InterestCalculatorForm
performed in the		
For...Next statement		

**Figure 11.14** App code for displaying calculation results.

---

```

1  Public Class InterestCalculatorForm
2      ' handles Calculate Button's Click event
3      Private Sub calculateButton_Click(sender As System.Object,
4          e As System.EventArgs) Handles calculateButton.Click
5
6          ' declare variables to store user input
7          Dim principal As Decimal ' store principal
8          Dim rate As Double ' store interest rate
9          Dim amount As Decimal ' store each calculation
10         Dim output As String ' store output
11
12         ' retrieve user input
13         principal = Val(principalTextBox.Text)
14         rate = Val(rateTextBox.Text)
15
16         ' set output header
17         output = "Year" & ControlChars.Tab &
18             "Amount on Deposit" & ControlChars.CrLf
19

```

Declare a variable  
of type String — 10

Construct a header  
for the TextBox  
as a String — 17

---

**Figure 11.15** Interest Calculator app. (Part 1 of 2.)

---

Loop from 1 to the	<b>20</b>	' calculate amount after each year and append to string
value specified by the	<b>21</b>	<b>For</b> yearCounter = 1 To yearUpDown.Value
user in the yearUpDown	<b>22</b>	amount =
control	<b>23</b>	principal * ((1 + rate / 100) ^ yearCounter)
Append result of	<b>24</b>	output &= (yearCounter & ControlChars.Tab &
calculation to the	<b>25</b>	String.Format("{0:C}", amount) & ControlChars.CrLf)
string named output	<b>26</b>	<b>Next</b>
	<b>27</b>	
Display results in	<b>28</b>	resultTextBox.Text = output ' display result
resultTextBox	<b>29</b>	<b>End Sub</b> ' calculateButton_Click
	<b>30</b>	<b>End Class</b> ' InterestCalculatorForm

---

**Figure 11.15** Interest Calculator app. (Part 2 of 2.)

The formula to convert temperature recorded on Fahrenheit scale into Celsius scale is as follows:

$$\frac{C}{5} = \frac{F - 32}{9}$$

### Fahrenheit-Celsius conversion table

<b>Fahrenheit</b>	<b>Celsius</b>
<b>105</b>	<b>40.5</b>
<b>104</b>	<b>40.0</b>
<b>103</b>	<b>39.4</b>
<b>102</b>	<b>38.9</b>
<b>101</b>	<b>38.3</b>
<b>100</b>	<b>37.7</b>
<b>99</b>	<b>37.2</b>
<b>98</b>	<b>36.6</b>
<b>97</b>	<b>36.1</b>
<b>96</b>	<b>35.5</b>