```
  pupil.courses(1) = "PHIL 200"
  pupil.courses(2) = "ENGL 120"
  club(0) = pupil
  'Enter data for second student
  pupil.name = "Mary Carlson"
  ReDim pupil.courses(3)
  pupil.courses(0) = "BIOL 110"
  pupil.courses(1) = "PHIL 200"
  pupil.courses(2) = "CMSC 100"
  pupil.courses(3) = "MATH 220"
  club(1) = pupil
  pupil.name = "George Hu"
  ReDim pupil.courses(2)
  pupil.courses(0) = "MATH 220"
  pupil.courses(1) = "PSYC 100"
  pupil.courses(2) = "ENGL 200"
  club(2) = pupil
  'Enter names and courses for remaining 7 people in the club
End Sub
```

36. Write the code for a btnDisplay_Click event procedure that displays the names of all the students in the club in a list box.
37. Write the code for a btnDisplay_Click event procedure that displays the names of all the students in the club who are registered for three courses.
38. Write the code for a btnDisplay_Click event procedure that displays the names of all the students in the club who are enrolled in CMSC 100.
39. Write the code for a btnDisplay_Click event procedure that displays the names of all the students in the club who are *not* enrolled in CMSC 100.

Solutions to Practice Problems 7.3

1. The event procedure contains two errors. First, the definition of a structure cannot be inside a procedure; it must be typed into the Declarations section of the Code Editor. Second, the statements `Team.school = "Rice"` and `Team.mascot = "Owls"` are not valid. "Team" should be replaced by a variable of type Team that has previously been declared.
2.
```
Structure Team
  Dim school As String
  Dim mascot As String
End Structure

Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim squad As Team
  squad.school = "Rice"
  squad.mascot = "Owls"
  txtOutput.Text = squad.school & " " & squad.mascot
End Sub
```

7.4 Two-Dimensional Arrays

Each array discussed so far held a single list of items. Such array variables are called **one-dimensional** or **single-subscripted variables**. An array can also hold the contents of a table with several rows and columns. Such array variables are called **two-dimensional** or **double-subscripted variables**. Two tables follow. Table 7.4 on the next page gives the road mileage between certain cities. It has four rows and four columns. Table 7.5 shows the leading universities in three graduate-school programs. It has three rows and five columns.

TABLE 7.4 Road mileage between selected U.S. cities.

	Chicago	Los Angeles	New York	Philadelphia
Chicago	0	2054	802	738
Los Angeles	2054	0	2786	2706
New York	802	2786	0	100
Philadelphia	738	2706	100	0

TABLE 7.5 Rankings of U.S. university graduate-school programs.

	1	2	3	4	5
Education	Stanford	Vanderbilt	UCLA	Columbia	U of OR
Engineering	MIT	Stanford	UC Berk	GA Tech	U of IL
Law	Yale	Harvard	Stanford	Columbia	NYU

Source: U.S. News and World Report, 2009.

Two-dimensional array variables store the contents of tables. They have the same types of names as other array variables. The only difference is that they have two subscripts, each with its own upper bound. The first upper bound is determined by the number of rows in the table, and the second upper bound is determined by the number of columns.

VideoNote
Two-dimensional arrays

Declaring a Two-Dimensional Array Variable

The statement

```
Dim arrayName(m, n) As DataType
```

declares an array of type *DataType* corresponding to a table with rows labeled from 0 to *m* and columns labeled from 0 to *n*. The entry in the *j*th row, *k*th column is *arrayName(j, k)*. For instance, the data in Table 7.4 can be stored in an array named *rm*. The statement

```
Dim rm(3, 3) As Double
```

will declare the array. Each element of the array has the form *rm*(row, column). The values of the elements of the array are

rm(0, 0) = 0	rm(0, 1) = 2054	rm(0, 2) = 802	rm(0, 3) = 738
rm(1, 0) = 2054	rm(1, 1) = 0	rm(1, 2) = 2786	rm(1, 3) = 2706
rm(2, 0) = 802	rm(2, 1) = 2786	rm(2, 2) = 0	rm(2, 3) = 100
rm(3, 0) = 738	rm(3, 1) = 2706	rm(3, 2) = 100	rm(3, 3) = 0

The data in Table 7.5 can be stored in a two-dimensional string array named *univ*. The appropriate array is declared with the statement

```
Dim univ(2, 4) As String
```

Some of the entries of the array are

univ(0, 0) = "Stanford"
univ(1, 2) = "UC Berk"
univ(2, 3) = "Columbia"

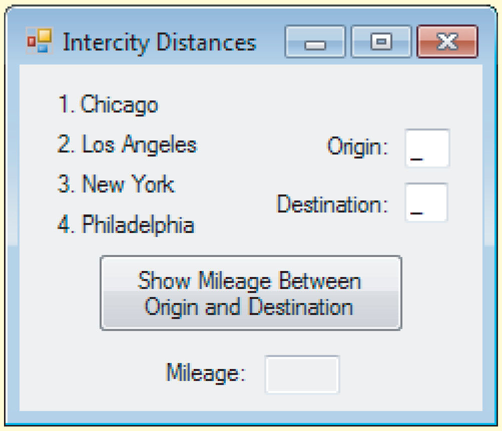
Implicit Array Sizing and Initialization

A two-dimensional array can be declared and initialized at the same time with a statement of the form

```
Dim arrayName(,) As DataType = {{ROW0}, {ROW1}, ..., {ROWm}}
```

where ROW0 consists of the entries in the top row of the corresponding table delimited by commas, ROW1 consists of the entries in the next row of the corresponding table delimited by commas, and so on.

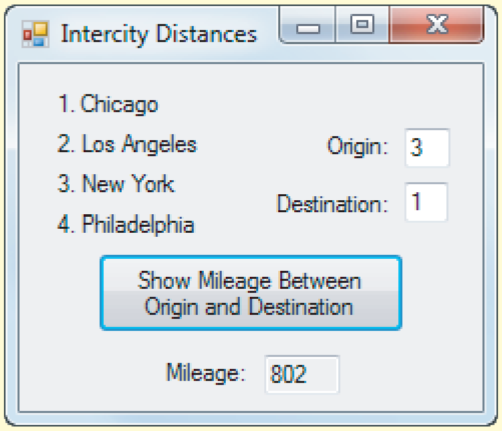
Example 1 The following program stores and accesses the data from Table 7.4.



OBJECT	PROPERTY	SETTING
frmDistances	Text	Intercity Distances
lblCh	Text	1. Chicago
lblLA	Text	2. Los Angeles
lblNY	Text	3. New York
lblPh	Text	4. Philadelphia
lblOrig	Text	Origin:
mtbOrig	Mask	0
lblDest	Text	Destination:
mtbDest	Mask	0
btnShow	Text	Show Mileage Between Origin and Destination
lblMiles	Text	Mileage:
txtMiles	ReadOnly	True

```
Dim rm(,) As Double = {{0, 2054, 802, 738},
                       {2054, 0, 2786, 2706},
                       {802, 2786, 0, 100},
                       {738, 2706, 100, 0}}

Private Sub btnShow_Click(...) Handles btnShow.Click
  'Determine road mileage between cities
  Dim row, col As Integer
  row = CInt(mtbOrig.Text)
  col = CInt(mtbDest.Text)
  If (row >= 1 And row <= 4) And (col >= 1 And col <= 4) Then
    txtMiles.Text = CStr(rm(row - 1, col - 1))
  Else
    MessageBox.Show("Origin and Destination must be numbers from 1 to 4",
                    "Error")
  End If
End Sub
```

[Run, type 3 into the Origin box, type 1 into the Destination box, and click on the button.]



The ReDim Statement

A previously declared array can be resized with

```
ReDim arrayName(r, c)
```

which loses the current contents, or with

```
ReDim Preserve arrayName(r, c)
```

which keeps the current values. However, when the keyword Preserve is used, only the second dimension can be resized. The upper bound of the first dimension of the array is given by `arrayName.GetUpperBound(0)`, and the upper bound of the second dimension is given by `arrayName.GetUpperBound(1)`.

So far, two-dimensional arrays have been used only to store data for convenient lookup. In the next example, an array is used to make a valuable computation.

Example 2 The Center for Science in the Public Interest publishes *The Nutrition Scorebook,* a highly respected rating of foods. The top two foods in each of five categories are shown in Table 7.6 along with some information on their composition. The following program computes the nutritional content of a meal. The table is read into three arrays—a one-dimensional array *foods* for the names of the ten foods, a one-dimensional array *nutrients* for the names of the five nutrients, and a two-dimensional array *nutTable* to hold the numbers from the table. (The value of *nutTable*(*k*, *i*) is the amount of the *i*th nutrient in the *k*th food.) The arrays *foods* and *nutrients* are filled from the files Foods.txt and Nutrients.txt, whose first three lines are as follows:

Foods.txt	Nutrients.txt
`cups of spinach`	`calories`
`medium sweet potatoes`	`protein (grams)`
`8 oz servings of yogurt`	`fat (grams)`

TABLE 7.6 Composition of 10 top-rated foods.

	Calories	Protein (grams)	Fat (grams)	Vit A (IU)	Calcium (mg)
Spinach (1 cup)	23	3	0.3	8100	93
Sweet potato (1 med.)	160	2	1	9230	46
Yogurt (8 oz.)	230	10	3	120	343
Skim milk (1 cup)	85	8	0	500	302
Whole wheat bread (1 slice)	65	3	1	0	24
Brown rice (1 cup)	178	3.8	0.9	0	18
Watermelon (1 wedge)	110	2	1	2510	30
Papaya (1 lg.)	156	2.4	0.4	7000	80
Tuna in water (1 lb)	575	126.8	3.6	0	73
Lobster (1 med.)	405	28.8	26.6	984	190

The array *nutTable* is filled with elements hard-coded into the program.

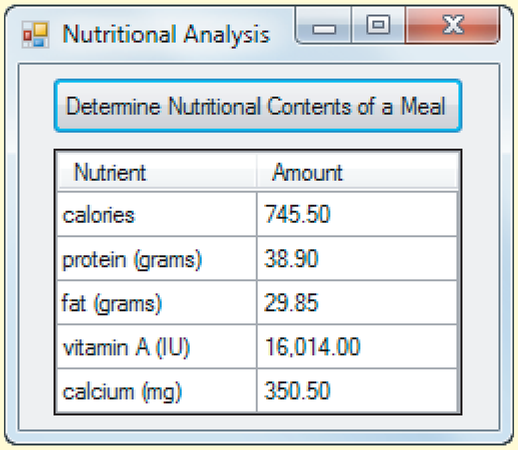
The program uses an array of structures named *nutFacts* of type NutFact and having five elements (one for each nutrient). The structure NutFact has two members; the first member holding the name of a nutrient and the second holding the total amount of that nutrient in the meal.

The program is written in the input-processing-output format. The input Sub procedure GetAmounts loops through 10 input dialog boxes that request the quantities of each food and places them into a one-dimensional array named *servings*. The processing Function procedure ProcessData uses the array *servings*, along with the arrays *nutrients* and *nutTable*, to fill the array of structures *nutFacts*. Finally, the output Sub procedure ShowData uses the array *nutFacts* to display the nutritional content of the meal into a DataGridView control.

```
Structure NutFact
  Dim nutrient As String   'name of one of the five nutrients
  Dim amount As Double     'amount of the nutrient in the meal
End Structure

Private Sub btnDetermine_Click(...) Handles btnDetermine.Click
  Dim servings(9) As Double
  Dim nutFacts(4) As NutFact             'This array of structures has an
  '                                       element for each nutrient.
  GetAmounts(servings)                   'input
  nutFacts = ProcessData(servings)       'processing
  ShowData(nutFacts)                     'output
End Sub

Sub GetAmounts(ByRef servings() As Double)
  Dim foods() As String = IO.File.ReadAllLines("Foods.txt")
  'Get the number of servings of each food.
  For i As Integer = 0 To 9
    servings(i) = CDbl(InputBox("How many servings of " & foods(i)))
  Next
End Sub

Function ProcessData(ByVal servings() As Double) As NutFact()
  Dim nutrients() As String = IO.File.ReadAllLines("Nutrients.txt")
  Dim nutTable(,) As Double = {{23, 3, 0.3, 8100, 93},
                               {160, 2, 1, 9230, 46},
                               {230, 10, 3, 120, 343},
                               {85, 8, 0, 500, 302},
                               {65, 3, 1, 0, 24},
                               {178, 3.8, 0.9, 0, 18},
                               {110, 2, 1, 2510, 30},
                               {156, 2.4, 0.4, 7000, 80},
                               {575, 126.8, 3.6, 0, 73},
                               {405, 28.8, 26.6, 984, 190}}
  Dim nutritionFacts(4) As NutFact  'This array of structures has an
  '                                  element for each nutrient.
  For i As Integer = 0 To 4
    nutritionFacts(i).nutrient = nutrients(i)  'Place the name of a nutrient
    '                                           into an array element.
    'The next five lines calculate the total amount of the nutrient
    'in the meal and place it into the array element.
    Dim sum As Double = 0
    For k As Integer = 0 To 9
      sum += servings(k) * nutTable(k, i)
    Next
    nutritionFacts(i).amount = sum  'Place the amount of the nutrient into
    '                                the array element.
  Next
```

```
    Return nutritionFacts
  End Function

  Sub ShowData(ByVal nutFacts() As NutFact)
    'Create a query and use it to place the data from the array
    'of structures into a DataGridView control.
    Dim query = From element In nutFacts
                Let Nutrient = element.nutrient
                Let Amount = FormatNumber(element.amount)
                Select Nutrient, Amount
    dgvOutput.DataSource = query.ToList
    dgvOutput.CurrentCell = Nothing
  End Sub
```

[Run, click on the button, and enter the following menu: .5 cups of spinach, 1 medium sweet potato, 2 slices of whole wheat bread, .25 of a large papaya, and 1 medium lobster.]



Nutrient	Amount
calories	745.50
protein (grams)	38.90
fat (grams)	29.85
vitamin A (IU)	16,014.00
calcium (mg)	350.50

Filling a Two-Dimensional Array with a Text File

Text files used to fill two-dimensional arrays are similar to those used to fill arrays of structures. Each line of the text file corresponds to a row of the table, with the entries for each row separated by commas. For instance, the array *mileage* discussed earlier can be filled with the text file Distances.txt consisting of the following four lines:

```
0,2054,802,738
2054,0,2786,2706
802,2786,0,100
738,2706,100,0
```

The following code creates and fills the array *mileage*.

```
Dim mileage(3, 3) As Double
Dim rowOfNums() As String = IO.File.ReadAllLines("Distances.txt")
Dim line As String
Dim data() As String
For i As Integer = 0 To mileage.GetUpperBound(0)
  line = rowOfNums(i)
  data = line.Split(","c)
  For j As Integer = 0 To mileage.GetUpperBound(1)
    mileage(i, j) = CDbl(data(j))
  Next
Next
```

Note: These eleven lines of code, with slight modifications, are needed in many of the exercises. You can store this block of code (or any frequently used fragment of code) for later use by highlighting it and dragging it from the Code Editor into the Toolbox. To reuse the code, just drag it back from the Toolbox to the Code Editor. A copy of the code will remain in the Toolbox for further use. Alternately, you can click on the location in the Code Editor where you want the code to be inserted, and then double-click on the code in the Toolbox.
We recommend that you place these lines of code in a program, highlight them, and drag them into the Toolbox. Then you can drag them out whenever you need them.

Using LINQ with Two-Dimensional Arrays

Although LINQ is not as useful with two-dimensional as it is with one-dimensional arrays, it is sometimes helpful. However, LINQ needs a Cast method to convert the two-dimensional array to a source data consisting of a one-dimensional array. Suppose *nums* is a two-dimensional array of type Double and having *m* rows and *n* columns. Then the code

```
Dim query = From num In nums.Cast(Of Double)()
            Select num
```

produces a sequence consisting of the $m \cdot n$ numbers in the array. The methods Count, Max, Min, First, Last, Average, and Sum apply to the query. Also, the sequence of numbers can be displayed in a list box with the DataSource property.

Comments

1. We can define three- (or higher-) dimensional arrays much as we do two-dimensional arrays. A three-dimensional array uses three subscripts, and the assignment of values requires a triple-nested loop. As an example, a meteorologist might use a three-dimensional array to record temperatures for various dates, times, and elevations. The array might be declared with the statement

   ```
   Dim temps(30, 23, 14) As Double
   ```

2. A ReDim statement cannot change the number of dimensions of an array. For instance, it cannot change a one-dimensional into a two-dimensional array.

Practice Problems 7.4

1. Consider the road-mileage program in Example 1. How can it be modified so the actual names of the cities can be supplied by the user?
2. In what types of problems are two-dimensional arrays superior to arrays of structures?

EXERCISES 7.4

In Exercises 1 through 16, assume the array *nums* is of type Double and has been filled with the contents of Table 7.7.

TABLE 7.7

7	3	1	0
2	5	9	8
0	6	4	10

In Exercises 1 through 12, determine or describe the output of the code.

1.
```
lstOutput.Items.Add(nums(0, 2))
```

2.
```
lstOutput.Items.Add(nums(2, 1))
```

3.
```
lstOutput.Items.Add(nums.GetUpperBound(1))
```

4.
```
lstOutput.Items.Add(nums.GetUpperBound(0))
```

5.
```
Dim total As Double = 0
For Each num In nums
  total += num
Next
lstOutput.Items.Add(total)
```

6.
```
Dim total As Double = 0
For c As Integer = 0 To nums.GetUpperBound(1)
  total += nums(2, c)
Next
lstOutput.Items.Add(total)
```

7.
```
Dim total As Double = 0
For r As Integer = 0 To nums.GetUpperBound(0)
  total += nums(r, 2)
Next
lstOutput.Items.Add(total)
```

8.
```
Dim total As Double = 0
For r As Integer = 0 To nums.GetUpperBound(0)
  For c As Integer = 0 To nums.GetUpperBound(1)
    total += nums(r, c)
  Next
Next
lstOutput.Items.Add(total)
```

9.
```
Dim query = From num In nums.Cast(Of Double)()
            Where (num > 8)
            Select num
lstOutput.Items.Add(query.Count)
```

10.
```
Dim query = From num In nums.Cast(Of Double)()
            Select num
lstOutput.Items.Add(query.Max)
```

11.
```
Dim query = From num In nums.Cast(Of Double)()
            Select num
lstOutput.Items.Add(query.Sum)
```

12.
```
Dim query = From num In nums.Cast(Of Double)()
            Where (num Mod 2 = 0)
            Order By num
            Select num / 2
            Distinct
For Each n As Double In query
  lstOutput.Items.Add(n)
Next
```

13. Write code that creates a new array whose entries are twice the entries of *nums*.

14. Write code that uses a For Each loop to find the average of the numbers in *nums*.

15. Write code that finds the sum of the even numbers in *nums* two ways: first with a For Each loop and then with a LINQ query.

16. Write code that finds the average of the odd numbers in *nums* two ways: first with a For Each loop and then with LINQ.

In Exercises 17 and 18, determine the output of the code.

17.
```
Dim nums(1, 2) As Double
Dim rowOfNums() As String = IO.File.ReadAllLines("Digits.txt")
Dim line As String
Dim data() As String
For i As Integer = 0 To nums.GetUpperBound(0)
  line = rowOfNums(i)
  data = line.Split(","c)
  For j As Integer = 0 To nums.GetUpperBound(1)
    nums(i, j) = CDbl(data(j))
  Next
Next
lstOutput.Items.Add(nums(0, 1) + nums(1, 0))
```
(Assume the two lines of the file Digits.txt are `9,7,6` and `5,4,3`.)

18.
```
Dim names(2, 1) As String
Dim rowOfNames() As String = IO.File.ReadAllLines("People.txt")
Dim line As String
Dim data() As String
For i As Integer = 0 To names.GetUpperBound(0)
  line = rowOfNames(i)
  data = line.Split(","c)
  For j As Integer = 0 To names.GetUpperBound(1)
    names(i, j) = data(j)
  Next
Next
lstOutput.Items.Add(names(2, 1) & " " & names(1, 1))
```
(Assume the three lines of the file People.txt are `Felix,Ungar;` `Oscar,Madison;` and `Henry,James.`)

In Exercises 19 through 29, write a program to perform the stated task.

19. A company has two stores (1 and 2), and each store sells three items (1, 2, and 3). The following tables give the inventory at the beginning of the day and the amount of each item sold during that day.

		Beginning Inventory					Sales for Day		
		ITEM					ITEM		
		1	2	3			1	2	3
Store	1	25	64	23	Store	1	7	45	11
	2	30	82	19		2	4	24	8

(a) Record the values of each table in an array.
(b) Adjust the values in the first array to hold the inventories at the end of the day and display these new inventories.
(c) Calculate and display the number of items in each store at the end of the day.

20. Table 7.8 gives the results of a survey on the uses of computers in the workplace. Each entry shows the percentage of respondents from the age category that use the computer for the indicated purpose.

(a) Place the data from the table in an array. (Use Workers.txt.)
(b) Determine the average of the percentages in the Spreadsheets column.

TABLE 7.8 Workers using computers on the job.

Age	Word Processing	Spreadsheets	Internet/ e-mail	Calendar/ Schedule	Programming
18–24	57.9	56.0	62.1	48.9	12.4
25–29	67.8	66.2	75.6	58.3	18.8
30–39	69.8	68.0	78.3	61.8	18.0
40–49	69.7	66.9	77.1	59.0	17.7
50–59	68.1	62.5	76.5	54.6	15.1
60 and older	63.7	53.8	71.2	46.1	10.6

Source: U.S. Center of Educational Statistics, Digest of Educational Statistics, 2003.

21. A university offers 10 courses at each of three campuses. The number of students enrolled in each course is presented in Table 7.9.

(a) Display the total number of course enrollments on each campus.
(b) Display the total number of students taking each course.

TABLE 7.9 Number of students enrolled in courses.

		Course									
		1	2	3	4	5	6	7	8	9	10
	1	5	15	22	21	12	25	16	11	17	23
Campus	2	11	23	51	25	32	35	32	52	25	21
	3	2	12	32	32	25	26	29	12	15	11

22. Table 7.10 gives the 2007 and 2008 U.S. sales for the top five restaurant chains.

(a) Place the data into a two-dimensional array. (Use Restaurants.txt.)
(b) Display the number that gives the total change in sales for these five restaurant chains.

TABLE 7.10 Top restaurant chains.

	2007 Sales $Bil	2008 Sales $Bil
1. McDonald's	28.7	30.0
2. Subway	8.2	9.6
3. Burger King	8.7	9.3
4. Starbucks	6.6	8.8
5. Wendy's	8.0	8.0

Source: QSR Magazine, October 2009.

23. The scores for the top four golfers at the 2009 U.S. Women's Open are shown in Table 7.11.

(a) Place the data into an array. (Use Golf.txt.)
(b) Display the total score for each player.
(c) Display the average score for each round.

TABLE 7.11 2009 U.S. Women's Open.

Round	1	2	3	4
Eun Hee Ji	71	72	70	71
Candie Kung	71	77	68	69
In-Kyung Kim	72	72	72	70
Cristie Kerr	69	70	72	75

24. Table 7.12 contains part of the pay schedule for federal employees in Washington, D.C. Table 7.13 gives the number of employees in each classification in a certain division. Place the data from the two tables into arrays and compute the amount of money this division pays for salaries during the year. (Use GS-Pay.txt and GS-Employees.txt.)

TABLE 7.12 2009 pay schedule for federal white-collar workers.

Step	1	2	3	4
GS-1	17,540	18,126	18,709	19,290
GS-2	19,721	20,190	20,842	21,396
GS-3	21,517	22,234	22,951	23,668
GS-4	24,156	24,961	25,766	26,571
GS-5	27,026	27,927	28,828	29,729
GS-6	30,125	31,129	32,133	33,137
GS-7	33,477	34,593	35,709	36,825

TABLE 7.13 Number of employees in each category.

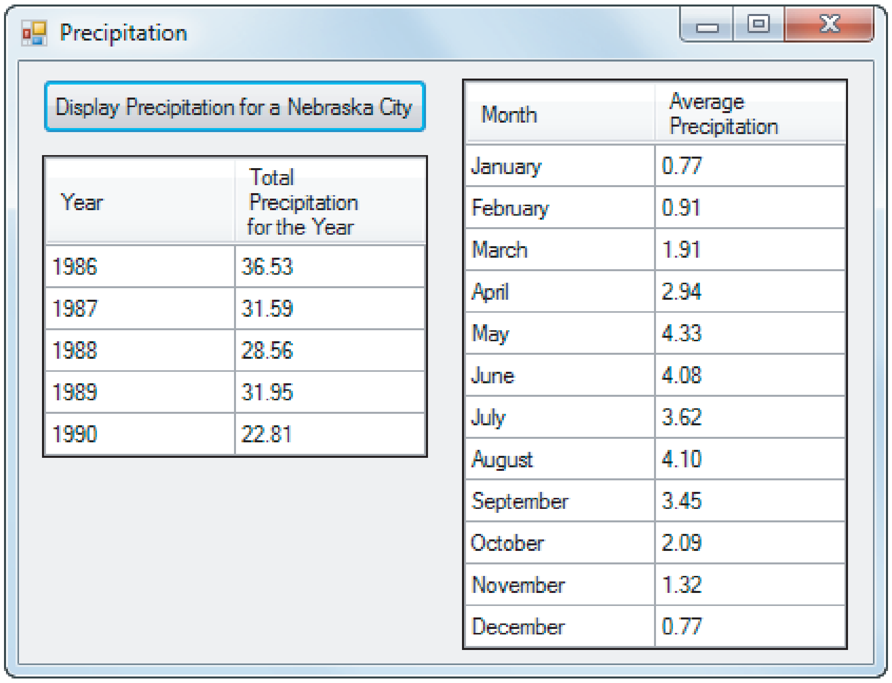
	1	2	3	4
GS-1	0	0	2	1
GS-2	2	3	0	1
GS-3	4	2	5	7
GS-4	12	13	8	3
GS-5	4	5	0	1
GS-6	6	2	4	3
GS-7	8	1	9	2

25. Consider Table 7.5, the rankings of three graduate-school programs. Write a program that places the data into an array, allows the name of a university to be input, and displays the categories in which it appears. Of course, a university might appear more than once or not at all. (Use Ranking.txt.)

26. Table 7.14 gives the monthly precipitation for a typical Nebraska city during a five-year period. Write a program that reads the table from a text file into an array and then displays the output shown in Fig. 7.29. (Use Rain.txt.)

TABLE 7.14 Monthly precipitation (in inches) for a typical Nebraska city.

	Jan.	Feb.	Mar.	Apr.	May	June	July	Aug.	Sept.	Oct.	Nov.	Dec.
1986	0.88	1.11	2.01	3.64	6.44	5.58	4.23	4.34	4.00	2.05	1.48	0.77
1987	0.76	0.94	2.09	3.29	4.68	3.52	3.52	4.82	3.72	2.21	1.24	0.80
1988	0.67	0.80	1.75	2.70	4.01	3.88	3.72	3.78	3.55	1.88	1.21	0.61
1989	0.82	0.80	1.99	3.05	4.19	4.44	3.98	4.57	3.43	2.32	1.61	0.75
1990	0.72	0.90	1.71	2.02	2.33	2.98	2.65	2.99	2.55	1.99	1.05	0.92



Year	Total Precipitation for the Year
1986	36.53
1987	31.59
1988	28.56
1989	31.95
1990	22.81

Month	Average Precipitation
January	0.77
February	0.91
March	1.91
April	2.94
May	4.33
June	4.08
July	3.62
August	4.10
September	3.45
October	2.09
November	1.32
December	0.77

FIGURE 7.29 Outcome of Exercise 26.

27. Suppose that a course has up to 15 students enrolled and that five exams are given during the semester. Write a program that accepts each student's name and grades as input, places the names in a one-dimensional array, and places the grades in a two-dimensional array. The program should then display each student's name and semester average. Also, the program should display the median for each exam. (For an odd number of grades, the median is the middle grade after the grades have been ordered. For an even number of grades, it is the average of the two middle grades.)

28. A square array of numbers is called a *magic square* if the sums of each row, each column, and each diagonal are equal. Figure 7.30 shows an example of a magic square. Write a program to determine if an array input by the user is a magic square. ***Hint:*** If at any time one of the sums is not equal to the sum of the numbers in the first row, then the search is complete.

$$\begin{pmatrix} 4 & 9 & 2 \\ 3 & 5 & 7 \\ 8 & 1 & 6 \end{pmatrix}$$

FIGURE 7.30 A magic square.

29. A company has three stores (1, 2, and 3), and each store sells five items (1, 2, 3, 4, and 5). The following tables give the number of items sold by each store and category on a particular day, and the cost of each item.

(a) Place the data from the left-hand table in a two-dimensional array and the data from the right-hand table in a one-dimensional array.

(b) Compute and display the total dollar amount of sales for each store and for the entire company.

Number of Items Sold During Day

		ITEM				
		1	2	3	4	5
	1	25	64	23	45	14
Store	2	12	82	19	34	63
	3	54	22	17	43	35

ITEM	COST PER ITEM
1	$12.00
2	$17.95
3	$95.00
4	$86.50
5	$78.00

Solutions to Practice Problems 7.4

1. Replace the masked text boxes with ordinary text boxes to hold city names. The function FindCityNum can be used to determine the subscript associated with each city. This function and the modified event procedure btnShow_Click are as follows:

```
Function FindCityNum(ByVal city As String) As Integer
  Select Case city.ToUpper
    Case "CHICAGO"
      Return 1
    Case "LOS ANGELES"
      Return 2
    Case "NEW YORK"
      Return 3
    Case "PHILADELPHIA"
      Return 4
    Case Else
      Return 0
  End Select
End Function

Private Sub btnShow_Click(...) Handles btnShow.Click
  Dim orig, dest As String
  Dim row, col As Integer 'Determine road mileage between cities
  orig = txtOrig.Text
  dest = txtDest.Text
  row = FindCityNum(orig)
  col = FindCityNum(dest)
  If (row <> 0) And (col <> 0) Then
    txtMiles.Text = CStr(rm(row – 1, col – 1))
  Else
    MessageBox.Show("Incorrect Origin and/or Destination", "Error")
  End If
End Sub
```

2. Both arrays of structures and two-dimensional arrays are used to hold related data. If some of the data are numeric and some are string, then structures must be used, because all entries of a two-dimensional array must be of the same type. Arrays of structures should also be used if the data will be sorted. Two-dimensional arrays are best suited to tabular data.

7.5 A Case Study: Analyze a Loan

This case study develops a program to analyze a loan. Assume the loan is to be repaid in equal monthly payments and interest is compounded monthly. The program should request the amount (principal) of the loan, the annual rate of interest, and the number of years over which the loan is to be repaid. The five options to be provided by buttons are as follows:

1. Calculate the monthly payment. The formula is

$$[\text{monthly payment}] = \frac{p \cdot r}{1 - (1 + r)^{-n}}$$

where p is the principal of the loan, r is the monthly interest rate (annual rate divided by 12) given as a number between 0 (for 0 percent) and 1 (for 100 percent), and n is the number of months over which the loan is to be repaid. When a payment computed in this manner results in fractions of a cent, the value should be rounded up to the next nearest cent. This corrected payment can be achieved using the formula

[corrected payment] = **`Math.Round(`*`originalPayment`*` + 0.005, 2)`**

2. Display an amortization schedule—that is, a table showing for each month the amount of interest paid, the amount of principal repaid, and the balance on the loan at the end of the month. At any time, the balance of the loan is the amount of money that must be paid in order to retire the loan. The monthly payment consists of two parts—interest on the balance and repayment of part of the principal. Each month

$$[\text{interest payment}] = r * [\text{balance at beginning of month}]$$

$$[\text{amount of principal repaid}] = [\text{monthly payment}] - [\text{interest payment}]$$

$$[\text{new balance}] = [\text{balance at beginning of month}] - [\text{amount of principal repaid}]$$

3. Calculate the interest paid during a calendar year. (This amount is deductible when itemizing deductions on a Federal income tax return.) The user should specify the number of the payment made in January of that year. For instance, if the first payment was made in September 2009, then the payment made in January 2010 would be payment number 5.
4. Show the effect of changes in the interest rate. Display a table giving the monthly payment for each interest rate from 1% below to 1% above the specified annual rate in steps of one-eighth of 1%.
5. Quit.

The User Interface

Figure 7.31 shows a possible form design and Table 7.15 gives the initial settings for the form and its controls. Figures 7.32, 7.33, 7.34, and 7.35 show possible outputs of the program for each task available through the buttons.

Designing the Analyze-a-Loan Program

Every routine uses data from the three text boxes. Therefore, we create a Sub procedure to read the contents of the text boxes and convert the contents into a usable form. Two of the routines