CHAPTER 7: ARRAYS

- Set up and use a control array
- Establish an array to variables and refer to individual elements in the array with variable subscripts
- Use the For Each/Next to traverse the array
- Store data in multidimensional array

INTRODUCTION TO ARRAYS

- An **array** variable is simply a variable that can store more than one value
- Each individual item in array that contains a value is called an **element**
- Arrays provide access to data by using a numeric **index**, or **subscript**, to identify each element in the array

INITIALIZE AN ARRAY (1 OF 4)

- An array declaration statement, includes the name of the array, how many items it can store, and what sort of data it can store.
- Must specify the number of array elements by indicating the **upper-bound index** of the array.
- Upper-bound index specifies the index of the last element of the array.
- **Dimensioning array** is setting the size of an array.

General Format: Define an Array

```
Dim intReservations(300) as Integer
```

intReservations assigns the array name.

300 is the index or subscript reserving the amount of memory needed – it is the highest numbered index.

Integer determines the data type of the entire array.

INITIALIZE AN ARRAY (2 OF 4)

```
13        Dim strNames() As String = {"Baker", "Lopez", "Buck", "Chan", "Tirrell"}
14        Dim intReservations() As Integer = {4, 5, 12, 2, 8}
```

- You can declare an array by assigning values to each element.
- The array is **implicitly sized** when a number is not used in the declaration statement to state the size of the array.
  - Do not place an upper-bound index in the parentheses because an error will occur.

INITIALIZE AN ARRAY (3 OF 4)

```
13        Dim strNames() As String = {"Baker", "Lopez", "Buck", "Chan", "Tirrell"}
14        Dim intReservations() As Integer = {4, 5, 12, 2, 8}
```

- **Parallel arrays** store related data in two or more arrays

strNames(0)	strNames(1)	strNames(2)	strNames(3)	strNames(4)
Baker	Lopez	Buck	Chan	Tirrell

intReservations(0)	intReservations(1)	intReservations(2)	intReservations(3)	intReservations(4)
4	5	12	2	8

```
17          Dim strAthlete(5) As String
18
19          strAthlete(0) = "Football"
20          strAthlete(1) = "Soccer"
21          strAthlete(2) = "Lacrosse"
22          strAthlete(3) = "Baseball"
23          strAthlete(4) = "Tennis"
24          strAthlete(5) = "Hockey"
```

INITIALIZE AN ARRAY (4 OF 4)

- An array can be declared by specify its upper-bound index and assign each item of the array one by one.

Data Type	Default Value
All numeric data types	0
String data type	Null
Boolean data type	False

INITIALIZE AN ARRAY WITH DEFAULT VALUES

- Each element is assigned a default value when you initialize an array but do not assign values immediately.

ACCESS ARRAY ELEMENTS USING A LOOP

- Loop is used to reference each element of an array.

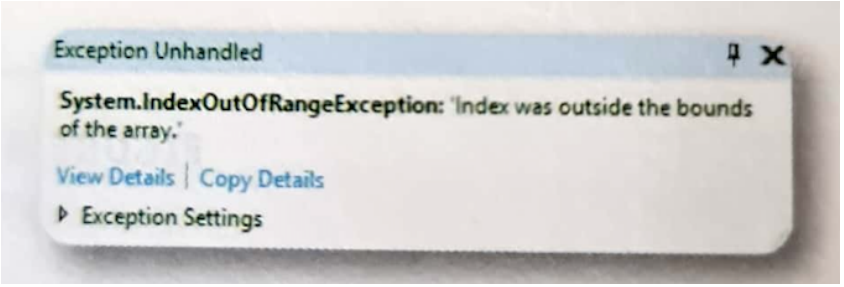
```
26          Dim intDailyTempJanuary(31) As Integer
27          Dim strTemp As String
28          Dim intDays As Integer
29
30          For intDays = 0 To 30
31              strTemp = InputBox("Enter the lowest temperature on January " _
32                  & intDays + 1, "Obtain Temperatures")
33              intDailyTempJanuary(intDays) = Convert.ToInt32(strTemp)
34          Next
```

ARRAY BOUNDARIES

- The Visual Basic compiler determines if each subscript is within the boundaries set when you initialized the array.

```
37        Dim intDailyTempJanuary(30) As Integer
38        Dim strTemp As String
39        Dim intDays As Integer
40
41        For intDays = 0 To 31
42            strTemp = InputBox("Enter the lowest temperature on January " _
43                & intDays + 1, "Obtain Temperatures")
44            intDailyTempJanuary(intDays) = Convert.ToInt32(strTemp)
45        Next
```

- The exception occurs when the loop tries to reference an element with the subscript 31.
- This element does not exist because the array contains 31 elements with an upper-bound index of 30.



UPPER-BOUND INDEX CONSTANT

- An array can use a constant value to represent its upper-bound index.
- By using a constant, the size of several arrays can be specified quickly.

```
47        Const intUpperBound As Integer = 40
48        Dim strFirstNames(intUpperBound) As String
49        Dim strLastNames(intUpperBound) As String
```

REINITIALIZE AN ARRAY (1 OF 2)

- Every array in Visual Basic is considered **dynamic**, which means that you can resize it at run time.
- When you change the number of elements in an existing array, you re-dimension it.
- The **`ReDim`** statement assigns a new array size to the specified array variable.

```
Dim strEmployees(50) As String
' Later in the code
ReDim strEmployees(65)
```

REINITIALIZE AN ARRAY (2 OF 2)

- When you used the `ReDim` statement to re-dimension the array, all data contained in the array is lost.
- If you want to preserve the existing data, you can use the keyword **`Preserve`**.

```
55          Dim strEmployees(50) As String
56          ' Later in the code
57          ReDim Preserve strEmployees(65)
```

Length Property of a One-Dimensional Array

Syntax

arrayName.**Length**

Example

```
Dim strNames(3) As String
Dim intNumElements As Integer
intNumElements = strNames.Length
```

assigns the number 4 to the `intNumElements` variable

Length property

USE THE LENGTH PROPERTY (1 OF 2)

- The Length property of an array contains the number of elements in an array.

USE THE LENGTH PROPERTY (2 OF 2)

- Using the Length property can prevent the program from throwing the `IndexOutOfRange` exception.
- From the example, For loop uses the Length property to determine the number of loop iterations.

```
63          Dim intYear(99) As Integer
64          Dim intCount As Integer
65
66          ' Assigns the years from 2001 to 2100 to the elements in the table
67          For intCount = 0 To (intYear.Length - 1)
68              intYear(intCount) = 2001 + intCount
69          Next
```

General Format: For Each

For Each ***Control Variable Name in Array Name***

```
    ' Lines of Code
```

Next

For Each — This type of loop iterates through an array until the array reaches the last element.

Control Variable Name — This variable will contain each individual element of the array without a subscript as the loop is processed. During the first iteration of the loop, the first element in the array is assigned to the control variable.

Array Name( ) — The name of the array that the loop cycles through. The array must be initialized first.

Next — This statement continues the loop to its next iteration.

THE FOR EACH LOOP (1 OF 2)

- **`For Each`** loop is a special loop designed specially for arrays.
- The `For Each` loop cycles through each array element until the end of the array.

THE FOR EACH LOOP (2 OF 2)

- Each element in the array is assigned to the control variable `strPioneer` as the `For Each` loop is executed.
- The array elements and the control variables are both `String`.
- When the loop begins, the first element of the `strFamousComputerPioneers` array is placed in the `strPioneer` variable and the body of the loop is executed.
- The looping continues until all elements within the array have been processed.

```
93      Private Sub btnHistory_Click(ByVal sender As System.Object, ByVal e As System.
        EventArgs) Handles btnHistory.Click
94
95          Dim strFamousComputerPioneers() As String = {"Pascal", _
96              "Babbage", "Ada", "Aiken", "Jobs"}
97          Dim strHeading As String = "Computer Pioneers:"
98          Dim strPioneer As String
99
100         lstPioneers.Items.Add(strHeading)
101         lstPioneers.Items.Add("")
102
103         For Each strPioneer In strFamousComputerPioneers
104             lstPioneers.Items.Add(strPioneer)
105         Next
106
107     End Sub
```

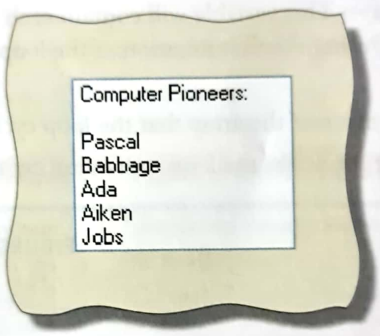


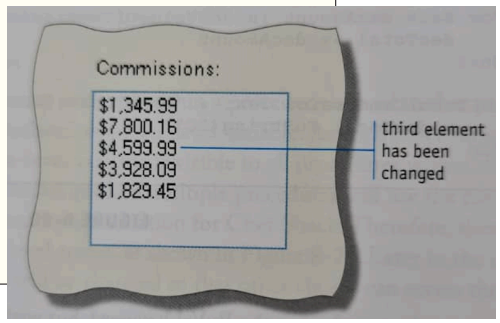
FIGURE 8-28

```
 9      Public Class frmDepreciation
10
11      ' Class Level Private variables
12      Private _intLifeOfItems As Integer = 5
13      Private _intSizeOfArray As Integer = 7
14      Private _strInventoryItem(_intSizeOfArray) As String
15      Private _strItemId(_intSizeOfArray) As String
16      Private _decInitialPrice(_intSizeOfArray) As Decimal
17      Private _intQuantity(_intSizeOfArray) As Integer
```

SCOPE OF ARRAYS

- The scope of an array declared within a procedure is local to that procedure.
- An array can be declared as a class-level variable and the array is visible to all procedures within the class.

```
131     Private Sub btnCommission_Click(ByVal sender As System.Object, ByVal e As System. ↙
        EventArgs) Handles btnCommission.Click
132
133         Dim decCommissionAmounts() As Decimal = {1345.99, 7800.16, _
134             5699.99, 3928.09, 1829.45}
135         Dim decDisplay As Decimal
136
137         ChangeValue(decCommissionAmounts)
138
139         For Each decDisplay In decCommissionAmounts
140             lstDisplay.Items.Add(decDisplay.ToString("C"))
141         Next
142
143     End Sub
144
145     Private Sub ChangeValue(ByVal decValueOfCommission() As Decimal)
146
147         decValueOfCommission(2) = 4599.99
148
149     End Sub
```



PASS AN ARRAY

- An array can be passed as an argument to a Sub procedure or a Function procedure.
- If you change the value of any array element in a procedure, the original array is changed.

Syntax

Array.Sort(*arrayName***)**
Array.Reverse(*arrayName***)**

Example 1

```
Dim intScores() As Integer = {78, 90, 75, 83}
Array.Sort(intScores)
```

sorts the contents of the array in ascending order, as follows: 75, 78, 83, and 90

Example 2

```
Dim intScores() As Integer = {78, 90, 75, 83}
Array.Reverse(intScores)
```

reverses the contents of the array, placing the values in the following order: 83, 75, 90, and 78

Example 3

```
Dim intScores() As Integer = {78, 90, 75, 83}
Array.Sort(intScores)
Array.Reverse(intScores)
```

sorts the contents of the array in ascending order and then reverses the contents, placing the values in descending order as follows: 90, 83, 78, and 75

SORT AN ARRAY

- You can use the `Array.Sort` method to sort the values in an array in ascending order.
- To sort the values in descending order, you first use the `Array.Sort` method to sort the values in ascending order; then use the `Array.Reverse` method to reverse the sorted values.

SEARCH AN ARRAY

- Searching each element in an array is called a **sequential search**.
- The **BinarySearch** method searches a sorted array for a value using a binary search algorithm.
  - The binary search algorithm searches an array by repeatedly dividing the search interval in half.

General Format: BinarySearch Procedure

```
intValue = Array.BinarySearch(arrayname, value)
```

If intValue returns a positive number or zero, a match was found at the subscript number equal to intValue.

If intValue returns a negative number, a match was not found.

CREATE A TWO-DIMENSIONAL ARRAY (1 OF 2)

- A **two-dimensional** array is like an array of arrays.
- A **two-dimensional** array holds data that is arranged in rows and columns.

rows columns

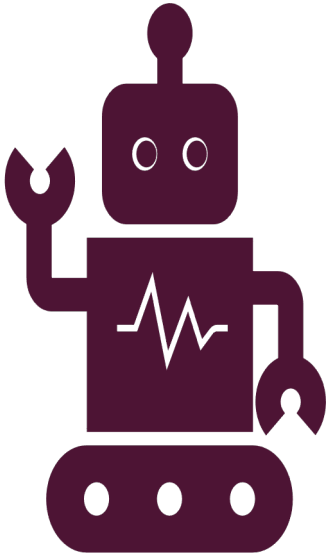
```
182         Dim intVal(2, 3) As Integer
```

	column 0	column 1	column 2	column 3
row 0	intVal(0,0)	intVal(0,1)	intVal(0,2)	intVal(0,3)
row 1	intVal(1,0)	intVal(1,1)	intVal(1,2)	intVal(1,3)
row 2	intVal(2,0)	intVal(2,1)	intVal(2,2)	intVal(2,3)

```
192         Dim intPassengers(,) As Integer = {{35, 34}, {28, 27}, {24, 23}, _
193             {20, 19}}
194         Dim intTotalColumn As Integer = 0
195         Dim intCol As Integer
196         Dim intRow As Integer
197
198         For intCol = 0 To 1
199             'Resets the total to 0
200             intTotalColumn = 0
201             For intRow = 0 To 3
202                 intTotalColumn += intPassengers(intRow, intCol)
203             Next
204             MsgBox("The Sum of Column #" & intCol + 1 & " is " _
205                 & intTotalColumn.ToString & " million.")
206         Next
```

CREATE A TWO-DIMENSIONAL ARRAY (2 OF 2)

- Nested loop is used to process two-dimensional arrays.
- The outer loop controls the column index and the inner loop controls the row index of the array.



END OF TOPIC … ARRAYS