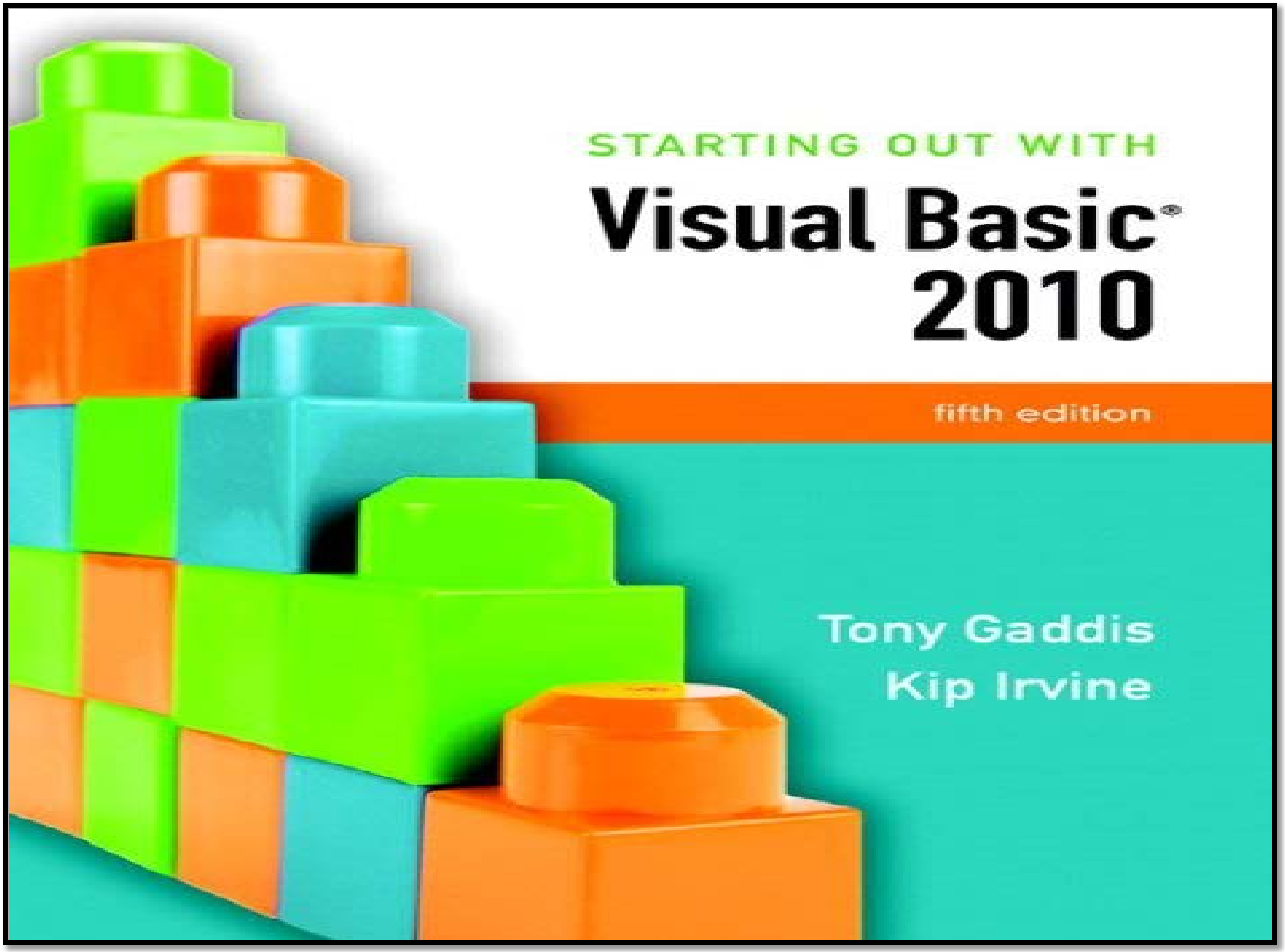
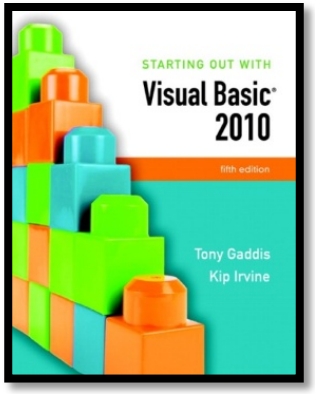

STARTING OUT WITH
Visual Basic®
2010
fifth edition
Tony Gaddis
Kip Irvine




# Chapter 8

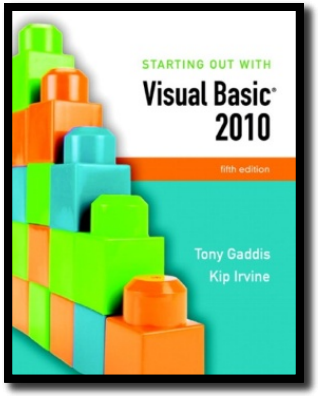
## Arrays and More

**Addison Wesley**
is an imprint of

PEARSON

# Introduction

- Arrays are like groups of variables that allow you to store sets of similar data
  - A single dimension array is useful for storing and working with a single set of data
  - A multidimensional array can be used to store and work with multiple sets of data
- Array programming techniques covered
  - Summing and averaging all the elements in an array
  - Summing all the columns in a two-dimensional array
  - Searching an array for a specific value
  - Using parallel arrays




Section 8.1

# ARRAYS

An array is like a group of variables with one name. You store and work with values in an array by using a subscript.

**Addison Wesley**
is an imprint of

PEARSON

# Array Characteristics

- An **array** stores multiple values of same type
  - Like a group of variables with a single name
- For example, the days of the week might be:
  - a set of 7 string variables
  - with a maximum length of 9 characters
- All variables within an array are called **elements** and must be of the same data type
- You access the elements in an array through a subscript

# Subscript Characteristics

- A **subscript**, also called an **index**, is a number that identifies a specific element within an array
- Subscript numbering works like a list box index:
  - Subscript numbering begins at 0
  - 1st element in an array is always subscript 0
  - Last element is total number of elements – 1
- An array with 7 elements refers to the 1st element as subscript 0 and the last element as subscript 6

# Declaring an Array

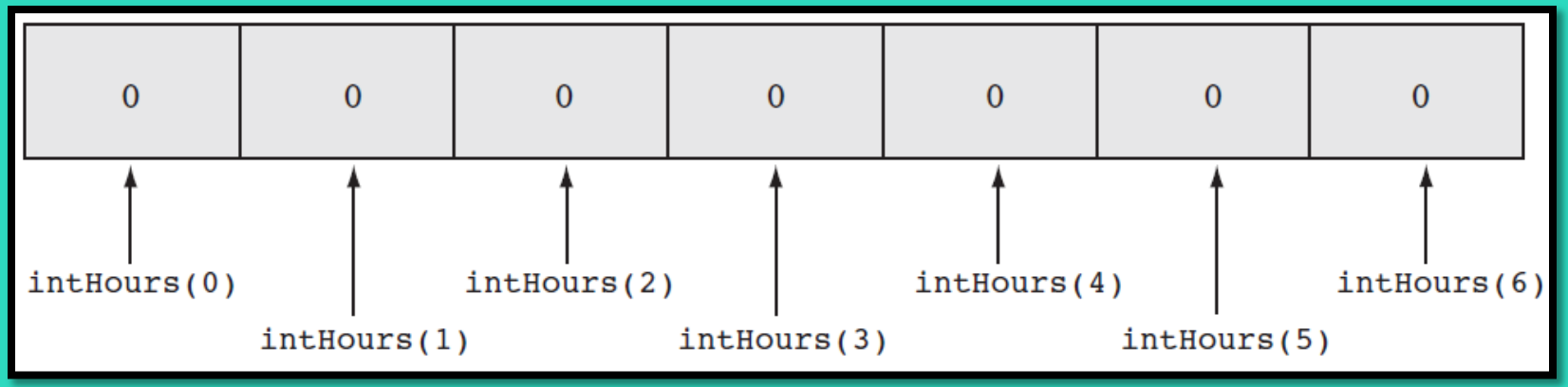
- Declare an array much like a regular variable

  **Dim *ArrayName (UpperSubscript) As DataType***

  – ***ArrayName*** is the name of the array
  – ***UpperSubscript*** is the value of the array's highest subscript
    - Must be a positive Integer
    - Positive Integer named constant
    - Or an Integer variable containing a positive number
  – ***DataType*** is a Visual Basic data type

# Array Declaration Example

**Dim intHours(6) As Integer**




- This statement declares **intHours** as an array of Integers
  - **(6)** indicates that the array's highest subscript is 6
  - Consists of seven elements with subscripts 0 through 6
  - Array elements are initialized to 0

# Default Initialization

- All elements of an Integer array are initialized to zero
  - Same initialization as an integer variable
- Each array element is initialized exactly the same as a simple variable of that data type
  - Decimal elements are initialized to zero (0.0)
  - String elements are initialized to the special value **Nothing**

# Implicit Array Sizing and Initialization

- An array can be initialized when declared
- Example:

  **Dim intNumbers() As Integer = { 2, 4, 6, 8, 10, 12 }**
- This array is implicitly sized
  - Upper subscript value is left blank
  - Number of elements implied from initialization
  - Upper subscript of 5 implied by this example
  - This results in a 6 element array
- Elements are assigned the values shown

# Using Named Constants as Subscripts in Array Declarations

- A named constant may be used as an array's highest subscript instead of a number

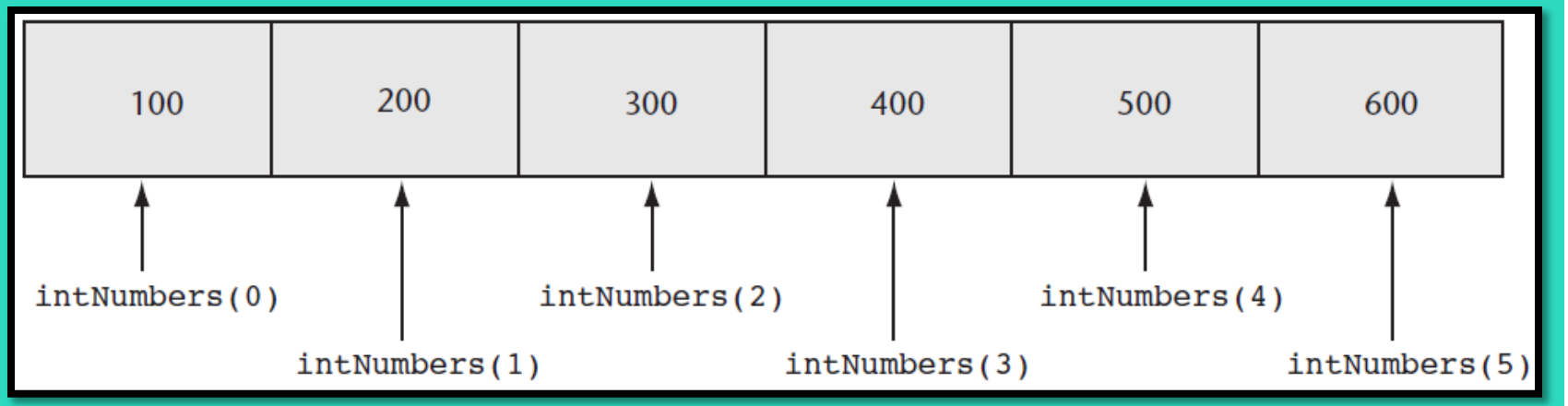
```
Const intMAX_SUBSCRIPT As Integer = 100
Dim intArray(intMAX_SUBSCRIPT) As Integer
```

- This is a common use for named constants
  - Highest subscript is often used multiple times
  - If highest subscript changes, use of a named constant allows it to be changed in one place

# Working with Array Elements

- You can store a value in an array element with an assignment statement

**intNumbers(0) = 100**
**intNumbers(1) = 200**
**intNumbers(2) = 300**
**intNumbers(3) = 400**
**intNumbers(4) = 500**
**intNumbers(5) = 600**



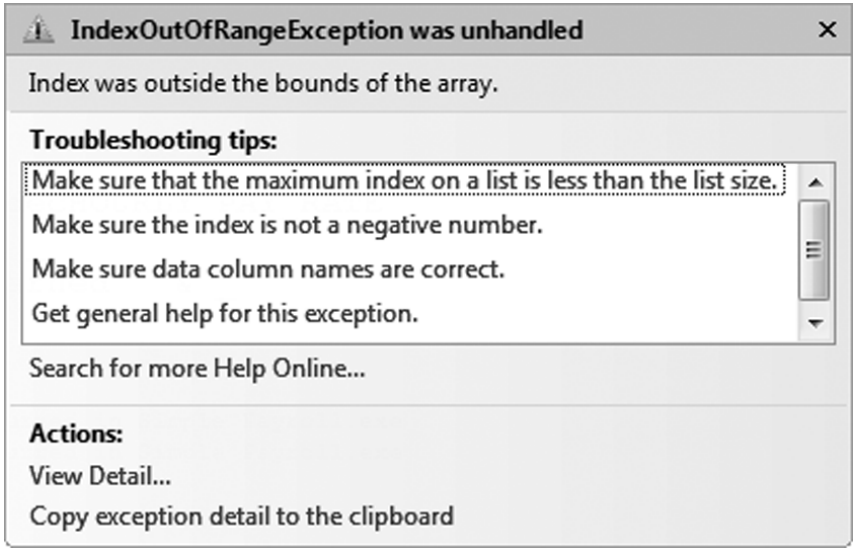
# Accessing Array Elements with a Loop

- Loops are frequently used to process arrays
  - Using an Integer variable as a subscript
  - For example, the following code stores an empty string in each element of **strNames**, a 1000-element array of strings:

```
Const intMAX_SUBSCRIPT As Integer = 999
Dim strNames(intMAX_SUBSCRIPT) As String
Dim intCount As Integer

For intCount = 0 To intMAX_SUBSCRIPT
   strNames(intCount) = String.Empty
Next
```

# Array Bounds Checking

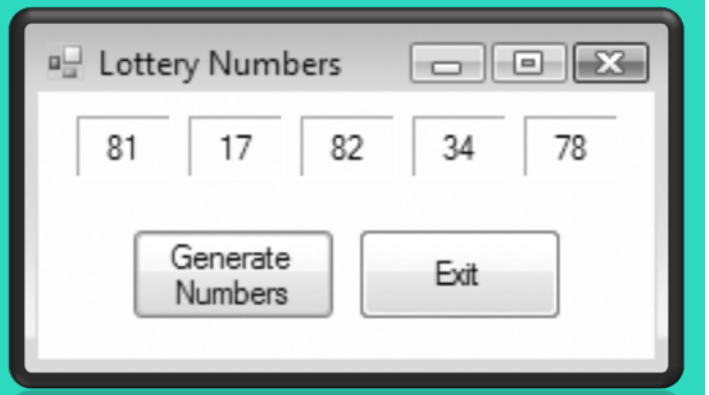
- The Visual Basic runtime system performs **array bounds checking**
  - It does not allow a statement to use a subscript outside the range of valid subscripts for an array
    - An invalid subscript causes VB to throw a run-time exception
    - Bounds checking is *not* done at design time



# Using an Array to Hold a List of Random Numbers

- In Tutorial 8-1 you will create an application that randomly generates lottery numbers

```
Const intMAX_SUBSCRIPT As Integer = 4
Dim intNumbers(intMAX_SUBSCRIPT) As Integer
Dim intCount As Integer

Dim rand As New Random

For intCount = 0 To intMAX_SUBSCRIPT
    intNumbers(intCount) = rand.Next(100)
Next
```



# Using Array Elements to Store Input

- Array elements can hold data entered by the user

```
Const intMAX_SUBSCRIPT As Integer = 9
Dim intSeries(intMAX_SUBSCRIPT) As Integer
Dim intCount As Integer

For intCount = 0 To intMAX_SUBSCRIPT
   intSeries(intCount) = CInt(InputBox("Enter a number."))
Next
```

- In Tutorial 8-2 you will create an application that
  - Uses input boxes to read a sequence of strings as input
  - Stores those strings in an array

# Getting the Length of an Array

- Arrays have a Length property
  - Holds the number of elements in the array
- For example

```
Dim strNames() As String = { "Joe", "Geri", "Rose" }

For intCount = 0 to strNames.Length – 1
    MessageBox.Show(strNames(intCount))
Next
```

  - **strNames.Length – 1** as the loop's upper limit
  - Length property is 1 greater than the upper subscript

# Processing Array Contents

- Array elements can be used just like regular variables in operations
  - For example
    - Multiplication

      **decGrossPay = intHours(3) * decPayRate**

    - Addition

      **intTallies(0) += 1**

    - Format String

      **MessageBox.Show(decPay(5).ToString("c"))**
- In Tutorial 8-3 you will complete an application that performs calculations using array elements

# Accessing Array Elements with a **For Each** Loop

- The **For Each loop** can simplify array processing
  - Retrieves the value of each element
  - Cannot modify values
- Here is the general format:

**For Each** *var* ***As*** *type* ***In*** *array*
***statements***
**Next**

  - ***var*** is the name of a variable just for use with the loop
  - ***type*** is the data type of the array
  - ***array*** is the name of an array

- For example, suppose we have the following array declaration:

```
Dim intArray() As Integer = {10, 20,
                             30, 40,
                             50, 60}
```

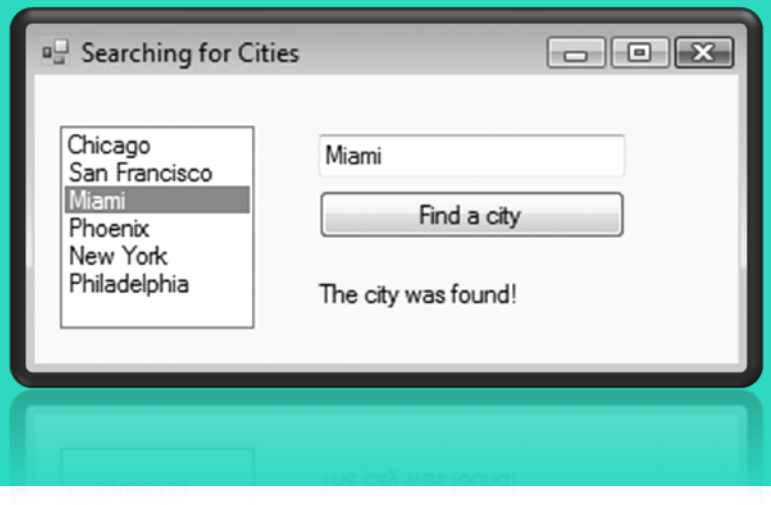
- The following **For Each** loop displays all the values in a list box named **lstShow**:

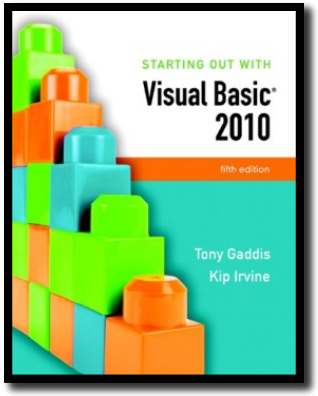
```
For Each intVal As Integer In intArray
    lstShow.Items.Add(intVal)
Next
```

# Optional Topic: Using the **For Each** Loop with a ListBox

- A **For Each** loop can also be used to process items in a collection
  - For example, to search for a city name in the Items collection of a ListBox control named **lstCities**

```
For Each strCity As String In lstCities.Items
    If strCity = txtCity.Text Then
        lblResult.Text = "The city was found!"
    End If
Next
```






Section 8.2

# MORE ABOUT ARRAY PROCESSING

There are many uses for arrays, and many programming techniques can be applied to them. You can total values and search for data. Related information may be stored in multiple parallel arrays. In addition, arrays can be resized at runtime.

**Addison Wesley**
is an imprint of

PEARSON

# How to Total the Values in a Numeric Array

- To total the values in a numeric array
  - Use a **For…Next** loop with an accumulator variable

```
Const intMAX_SUBSCRIPT As Integer = 24
Dim intUnits(intMAX_SUBSCRIPT) As Integer

Dim intTotal As Integer = 0
Dim intCount As Integer

For intCount = 0 To (intUnits.Length – 1)
    intTotal += intUnits(intCount)
Next
```

# How to Total the Values in a Numeric Array

- You can also use a **For Each** loop with an accumulator variable

```
Const intMAX_SUBSCRIPT As Integer = 24
Dim intUnits(intMAX_SUBSCRIPT) As Integer

Dim intTotal As Integer = 0

For Each intVal As Integer In intUnits
    intTotal += intVal
Next
```

# Calculating the Average Value in a Numeric Array

- Sum the values in the array
- Divide the sum by the number of elements

```
Const intMAX_SUBSCRIPT As Integer = 24
Dim intUnits(intMAX_SUBSCRIPT) As Integer

Dim intTotal As Integer = 0
Dim dblAverage As Double
Dim intCount As Integer

For intCount = 0 To (intUnits.Length – 1)
    intTotal += intUnits(intCount)
Next

' Use floating-point division to compute the average.
dblAverage = intTotal / intUnits.Length
```

# Find the Highest and Lowest Values in an Integer Array

- Highest Value

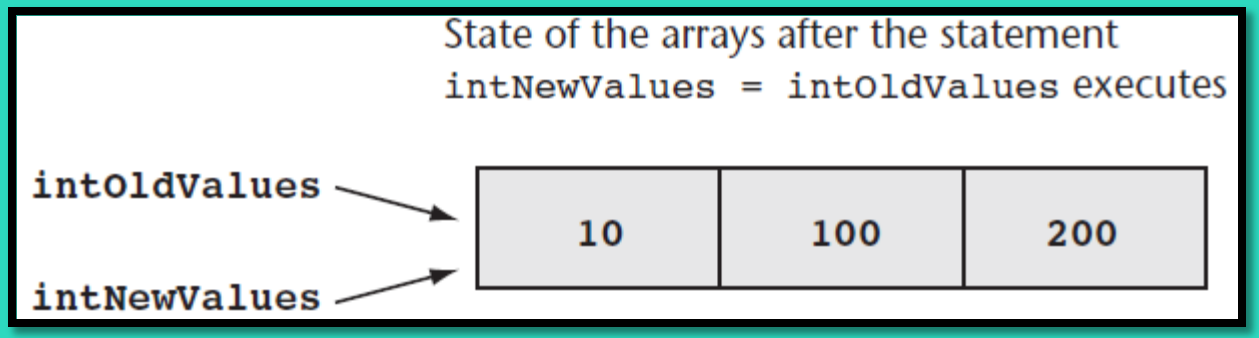
```
Dim intUnits() As Integer = {1, 2, 3, 4, 5}
Dim intCount As Integer
Dim intHighest As Integer

' Get the first element.
intHighest = intUnits(0)

' Search for the highest value.
For intCount = 1 To (intUnits.Length - 1)
    If intUnits(intCount) > intHighest Then
        intHighest = intNumbers(intCount)
    End If
Next
```

- Lowest Value

```
Dim intUnits() As Integer = {1, 2, 3, 4, 5}
Dim intCount As Integer
Dim intLowest As Integer

' Get the first element.
intLowest = intUnits(0)

' Search for the lowest value.
For intCount = 1 To (intUnits.Length - 1)
    If intUnits(intCount) < intLowest Then
        intLowest = intNumbers(intCount)
    End If
Next
```

# Copying One Array's Contents to Another

- A single assignment statement
  - Does not copy array values into another array
  - Causes both array names to reference the same array in memory

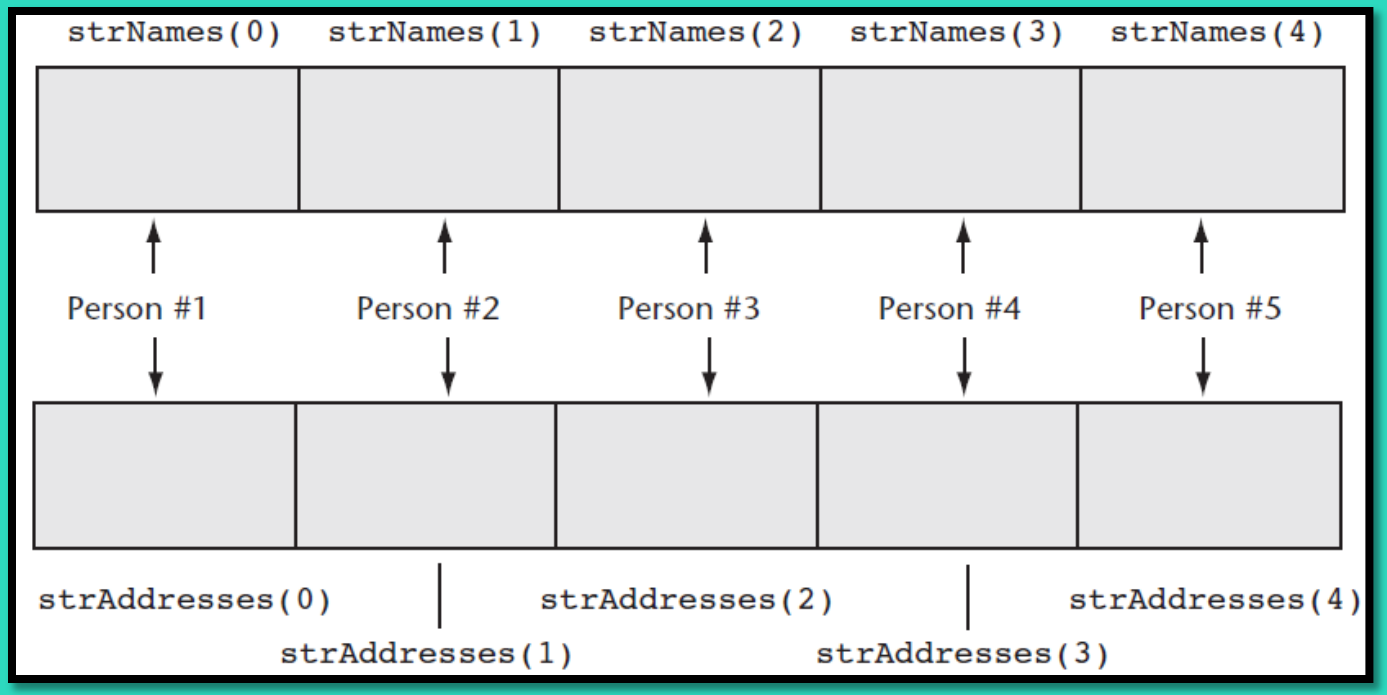



- A loop must be used to copy individual elements from one array to another

```
For intCount = 0 To (intOldValues.Length-1)
   intNewValues(intCount) = intOldValues(intCount)
Next
```

# Parallel Arrays

- Related data in multiple arrays can be accessed using the same subscript




- Tutorial 8-4 examines an application that uses parallel arrays

# Parallel Relationships between Arrays, List Boxes, and Combo Boxes

```
' A list box with names
lstPeople.Items.Add("Jean James")          ' Index 0
lstPeople.Items.Add("Kevin Smith")         ' Index 1
lstPeople.Items.Add("Joe Harrison")        ' Index 2

' An array with corresponding phone numbers
phoneNumbers(0) = "555-2987"               ' Element 0
phoneNumbers(1) = "555-5656"               ' Element 1
phoneNumbers(2) = "555-8897"               ' Element 2

' Display the phone number for the selected person's name.
If lstPeople.SelectedIndex  >  -1  And
   lstPeople.SelectedIndex < phoneNumbers.Length Then
   MessageBox.Show(phoneNumbers(lstPeople.SelectedIndex))
Else
   MessageBox.Show("That is not a valid selection.")
End If
```

# Searching Arrays

- The most basic method of searching an array is the **sequential search**
  - Uses a loop to examine elements in the array
  - Compares each element with the search value
  - Stops when the value is found or the end of the array is reached

- The Pseudocode for a sequential search is as follows:

```
found = False
subscript = 0
Do While found is False and
subscript < array's length
   If array(subscript) = searchValue Then
      found = True
      position = subscript
   End If
   subscript += 1
End While
```

# Sorting an Array

- Programmers often want to sort, or arrange the elements of an array in **ascending order**
  - Values are arranged from lowest to highest
    - Lowest value is stored in the first element
    - Highest value is stored in the last element
- To sort an array in ascending order
  - Use the **Array.Sort** method

- Here is the general format:

  **Array.Sort(*ArrayName*)**

- ***ArrayName*** is the name of the array you want to sort
  - For example:

```
Dim intNumbers() As Integer = {7, 12,
                               1, 6, 3}
Array.Sort(intNumbers)
```

- After the statement executes, the array values are in the following order
  - 1, 3, 6, 7, 12

# Sorting an Array

- When you pass an array of strings to the **Array.Sort** method the array is sorted in ascending order
  - According to the Unicode encoding scheme
  - Sort occurs in alphabetic order
    - Numeric digits first
    - Uppercase letters second
    - Lowercase letters last

- For example:

```
Dim strNames() As String = {"dan",
                            "Kim",
                            "Adam",
                            "Bill"}

Array.Sort(strNames)
```

- After the statement executes, the values in the array appear in this order:
  - **"Adam", "Bill", "Kim", "dan"**

# Dynamically Sizing Arrays

- You can change the number of elements in an array at runtime, using the **ReDim** statement

  **ReDim [Preserve]** ***Arrayname (UpperSubscript)***

  - ***Preserve*** is optional
    - If used, the existing values of the array are preserved
    - If not, the existing values are destroyed
  - ***Arrayname*** is the name of the array being resized
  - ***UpperSubscript*** is the new upper subscript
    - Must be a positive whole number
    - If smaller that it was, elements at the end are lost

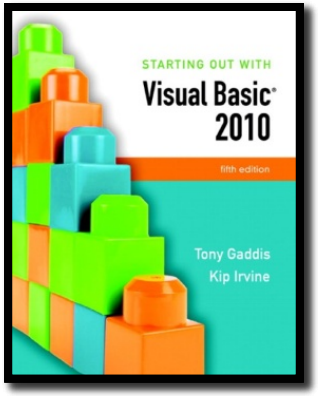
# Dynamically Sizing Arrays Example

- You can initially declare an array with no size, as follows:

  **Dim dblScores() As Double**

  - Then prompt the user for the number of elements
  - And resize the  array based on user input

```
intNumScores = CInt(InputBox("Enter the number of test scores."))
If intNumScores > 0 Then
   ReDim dblScores (intNumScores - 1)
Else
   MessageBox.Show("You must enter 1 or greater.")
End If
```




Section 8.3

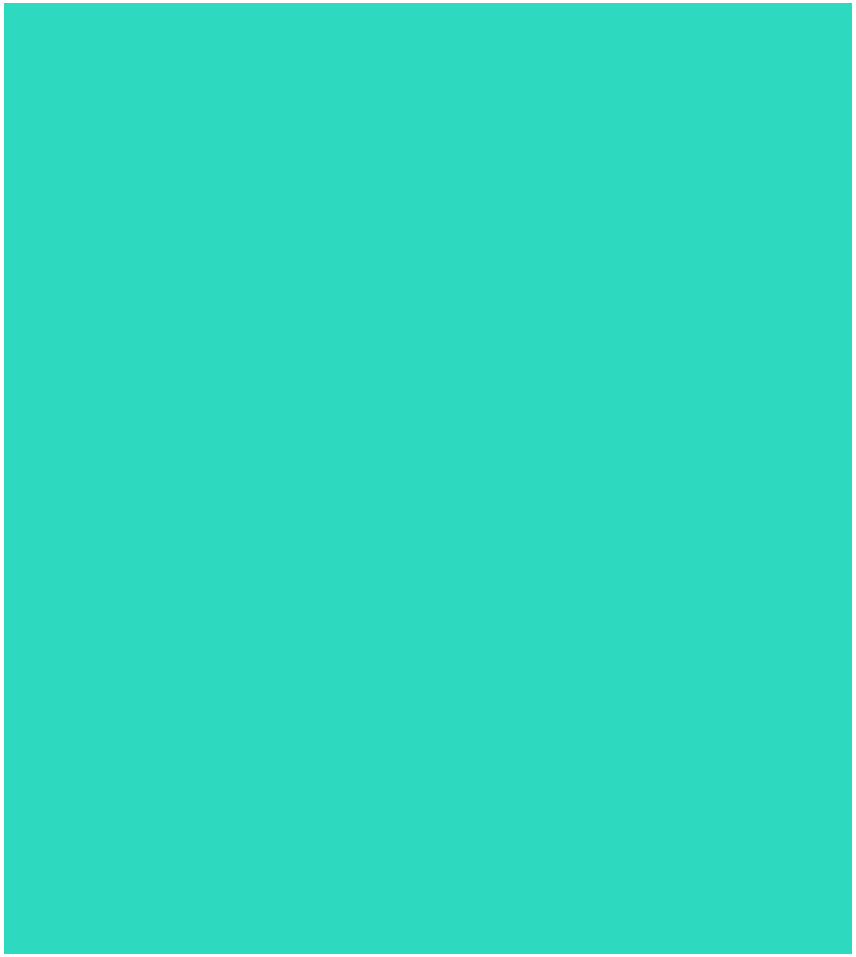
# PROCEDURES AND FUNCTIONS THAT WORK WITH ARRAYS

You can pass arrays as arguments to procedures and functions. You can return an array from a function. These capabilities allow you to write procedures and functions that perform general operations with arrays.

**Addison Wesley**
is an imprint of

PEARSON

# Passing Arrays as Arguments



```
' The DisplaySum procedure displays the
' sum of the elements in the argument array.
Sub DisplaySum(ByVal intArray() As Integer)
Dim intTotal As Integer = 0  ' Accumulator
Dim intCount As Integer      ' Loop counter

For intCount = 0 To (intArray.Length - 1)
   intTotal += intArray(intCount)
Next

MessageBox.Show("The total is " &
                          intTotal.ToString())
End Sub

Dim intNumbers() As Integer = { 2, 4, 7, 9, 8,
                                12, 10 }
DisplaySum(intNumbers)
```

# Passing Arrays by Value and by Reference

- Array arguments can be accessed and modified if passed **ByVal** or **ByRef**
  - **ByVal** prevents an array from being assigned to another array
  - **ByRef** allows an array to be assigned to another array

```
Dim intNumbers() As Integer = { 1, 2, 3, 4, 5 }
ResetValues(intNumbers)

Sub ResetValues(ByVal intArray() As Integer)
    Dim newArray() As Integer = {0, 0, 0, 0, 0}
    intArray = newArray
End Sub
```

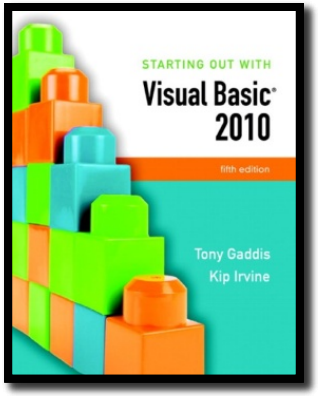
- After the **ResetValues** procedure executes
  - If passed **ByVal**, **intNumbers** is unchanged and keeps the values **{ 1, 2, 3, 4, 5 }**
  - If passed **ByRef**, **intNumbers** will reference the **newArray** values **{ 0, 0, 0, 0, 0 }**

# Returning an Array from a Function

```
' Get three names from the user and return them as an array of strings.
Function GetNames() As String()
    Const intMAX_SUBSCRIPT As Integer = 2
    Dim strNames(intMAX_SUBSCRIPT) As String
    Dim intCount As Integer

    For intCount = 0 To intMAX_SUBSCRIPT
        strNames(intCount) = InputBox("Enter name " &  (intCount + 1).ToString())
    Next

    Return strNames
End Function

Dim strCustomers() As String
strCustomers = GetNames()
```

*In Tutorial 8-5, you examine an application containing several functions that work with arrays*

***An array returned from a function must be assigned to an array of the same type***




Section 8.4

# MULTIDIMENSIONAL ARRAYS

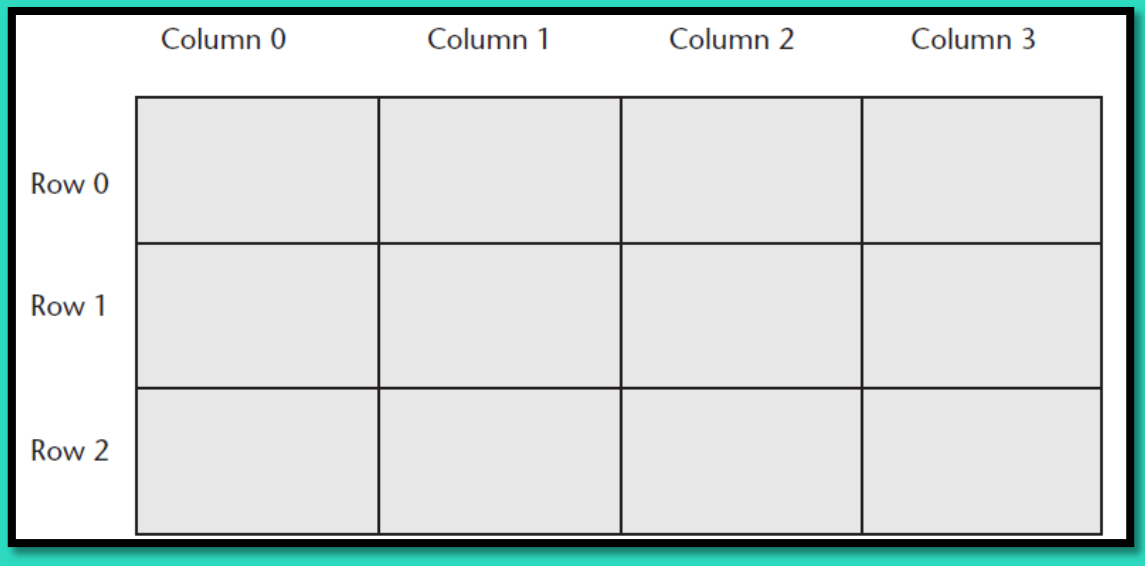
You may create arrays with more than two dimensions to hold complex sets of data

**Addison Wesley**
is an imprint of

PEARSON

# Two-Dimensional Arrays

- An array with one subscript is called a **one-dimensional array**
  - Useful for storing and working with a single set of data
- A **two-dimensional array** is like an array of arrays
  - Used to hold multiple sets of values
  - Think of it as having rows and columns of elements



# Declaring a Two-Dimensional Array

- A two-dimensional array declaration requires two sets of upper subscripts
  - First upper subscript is for the rows
  - Second upper subscript for the columns

    **Dim *ArrayName (UpperRow,UpperColumn) As DataType***
    - ***ArrayName*** is the name of the array
    - ***UpperRow*** is the value of the highest row subscript
      - Must be a positive integer
    - ***UpperColumn*** is the value of the highest column subscript
      - Must be a positive integer
    - ***DataType*** is the Visual Basic data type
- Example declaration with three rows and four columns:

  **Dim dblScores (2, 3) As Double**

# Processing Data in Two-Dimensional Arrays

- Use named constants to specify the upper subscripts

```
Const intMAX_ROW As Integer = 2
Const intMAX_COL As Integer = 3
Dim dblScores(intMAX_ROW, intMAX_COL) As Double
```

| | Column 0 | Column 1 | Column 2 | Column 3 |
|---|---|---|---|---|
| Row 0 | dblScores(0, 0) | dblScores(0, 1) | dblScores(0, 2) | dblScores(0, 3) |
| Row 1 | dblScores(1, 0) | dblScores(1, 1) | dblScores(1, 2) | dblScores(1, 3) |
| Row 2 | dblScores(2, 0) | dblScores(2, 1) | dblScores(2, 2) | dblScores(2, 3) |

# Processing Data in Two-Dimensional Arrays

- The elements in row 0 are referenced as follows:

```
dblScores(0, 0)      ' Element in row 0, column 0
dblScores(0, 1)      ' Element in row 0, column 1
dblScores(0, 2)      ' Element in row 0, column 2
dblScores(0, 3)      ' Element in row 0, column 3
```

- The elements in row 1 are referenced as follows:

```
dblScores(1, 0)      ' Element in row 1, column 0
dblScores(1, 1)      ' Element in row 1, column 1
dblScores(1, 2)      ' Element in row 1, column 2
dblScores(1, 3)      ' Element in row 1, column 3
```

- The elements in row 2 are referenced as follows:

```
dblScores(2, 0)      ' Element in row 2, column 0
dblScores(2, 1)      ' Element in row 2, column 1
dblScores(2, 2)      ' Element in row 2, column 2
dblScores(2, 3)      ' Element in row 2, column 3
```

# Processing Data in Two-Dimensional Arrays

- Example of storing a number in a single element

```
dblScores(2, 1) = 95
```

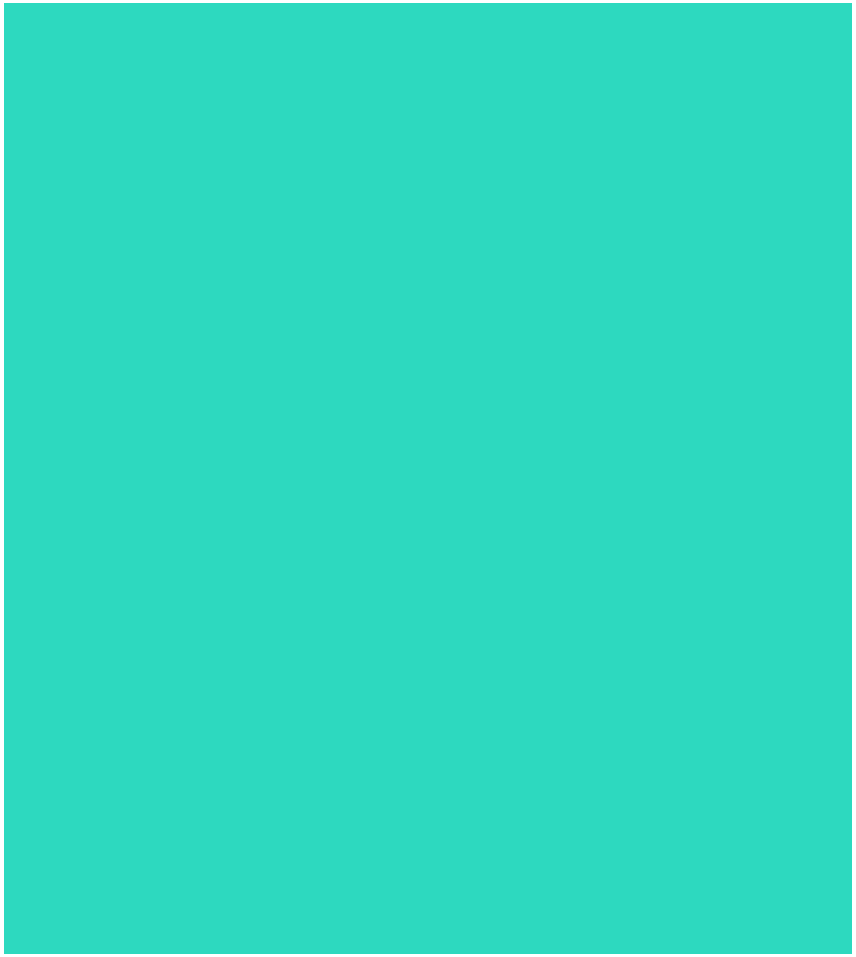
- Example of prompting the user for input, once for each element

```
For intRow = 0 To intMAX_ROW
   For intCol = 0 To intMAX_COL
      dblScores(intRow, intCol) = CDbl(InputBox("Enter a score."))
   Next
Next
```

- Example of displaying all of the elements in the array

```
For intRow = 0 To intMAX_ROW
   For intCol = 0 To intMAX_COL
      lstOutput.Items.Add(dblScores(intRow, intCol).ToString())
   Next
Next
```

# Implicit Sizing and Initialization of Two-Dimensional Arrays



This statement declares an array with three rows and three columns:

```
Dim intNumbers(,) As Integer = { {1, 2, 3} ,
                                 {4, 5, 6} ,
                                 {7, 8, 9} }
```

**intNumbers(0, 0)** is set to **1**
**intNumbers(0, 1)** is set to **2**
**intNumbers(0, 2)** is set to **3**

**intNumbers(1, 0)** is set to **4**
**intNumbers(1, 1)** is set to **5**
**intNumbers(1, 2)** is set to **6**

**intNumbers(2, 0)** is set to **7**
**intNumbers(2, 1)** is set to **8**
**intNumbers(2, 2)** is set to **9**

# Summing the Columns of a Two-Dimensional Array

- The outer loop controls the column subscript
- The inner loop controls the row subscript

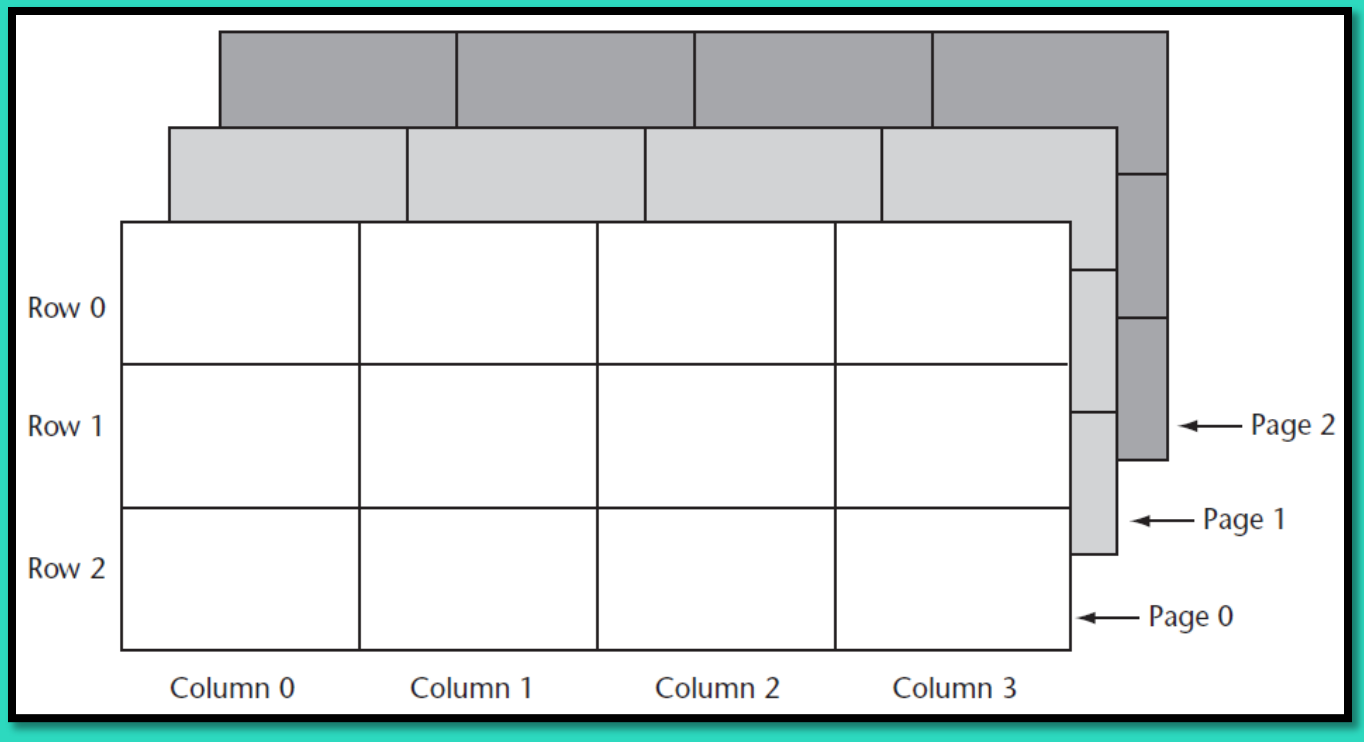
```
' Sum the columns.
For intCol = 0 To intMAX_COL
   ' Initialize the accumulator.
   intTotal = 0
   ' Sum all rows within this column.
   For intRow = 0 To intMAX_ROW
      intTotal += intValues(intRow, intCol)
   Next
   ' Display the sum of the column.
   MessageBox.Show("Sum of column " & intCol.ToString() &
                                  " is " & intTotal.ToString())
Next
```

- Tutorial 8-6 uses a two-dimensional array in the *Seating Chart* application

# Three-Dimensional Arrays and Beyond

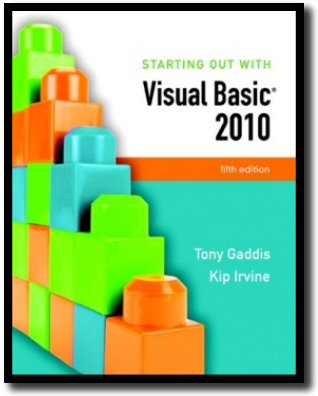
- You can create arrays with up to 32 dimensions
- The following is an example of a three-dimensional array:

**Dim intPages(2, 2, 3) As Decimal**



# Three-Dimensional Arrays and Beyond

- Arrays with more than three dimension are difficult to visualize
  - Useful in some programming applications
  - For example:
    - A factory warehouse where cases of widgets are stacked on pallets, an array of four dimensions can store a part number for each widget
    - The four subscripts of each element can store:
      - Pallet number
      - Case number
      - Row number
      - Column number
    - A five dimensional array could be used for multiple warehouses




Section 8.5

# FOCUS ON GUI DESIGN: THE ENABLED PROPERTY AND THE TIMER CONTROL

You can disable controls by setting their Enabled property to False. The Timer control allows your application to execute a procedure at regular time intervals

**Addison Wesley**
is an imprint of

PEARSON

# The Enabled Property

- Most controls have an **Enabled property**
- If this Boolean property is set to **False** the control is disabled meaning the control:
  - Cannot receive the focus
  - Cannot respond to user generated events
  - Will appear dimmed, or grayed out
- Default value for this property is **True**
- May be set in code when needed as shown:

**radBlue.Enabled = False**

# The Timer Control

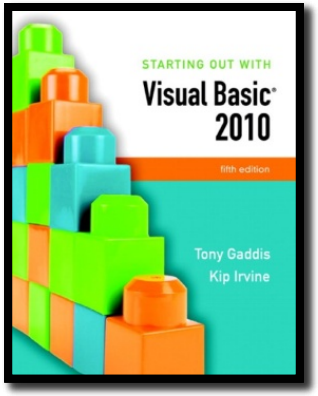
- The **Timer control** allows an application to automatically execute code at regular intervals

- To place a Timer control on a form:
  - Double-click the Timer icon in *Components* section of the *Toolbox*
  - Appears in the component tray at design time
  - Prefix a Timer control's name with **tmr**

- To create a Tick event handler code template:
  - Double-click a Timer control that has been added to the *Component* tray
  - Code will be executed at regular intervals

# Timer Control Properties

- Timer control has two important properties:
  - The Enabled property
    - Must be set to **True** to respond to Tick events
  - The **Interval property**
    - The number of milliseconds that elapse between events
- Tutorial 8-7 demonstrates the Timer control
- In Tutorial 8-8 you will use the Timer control to create a game application




Section 8.6

# FOCUS ON GUI DESIGN: ANCHORING AND DOCKING CONTROLS

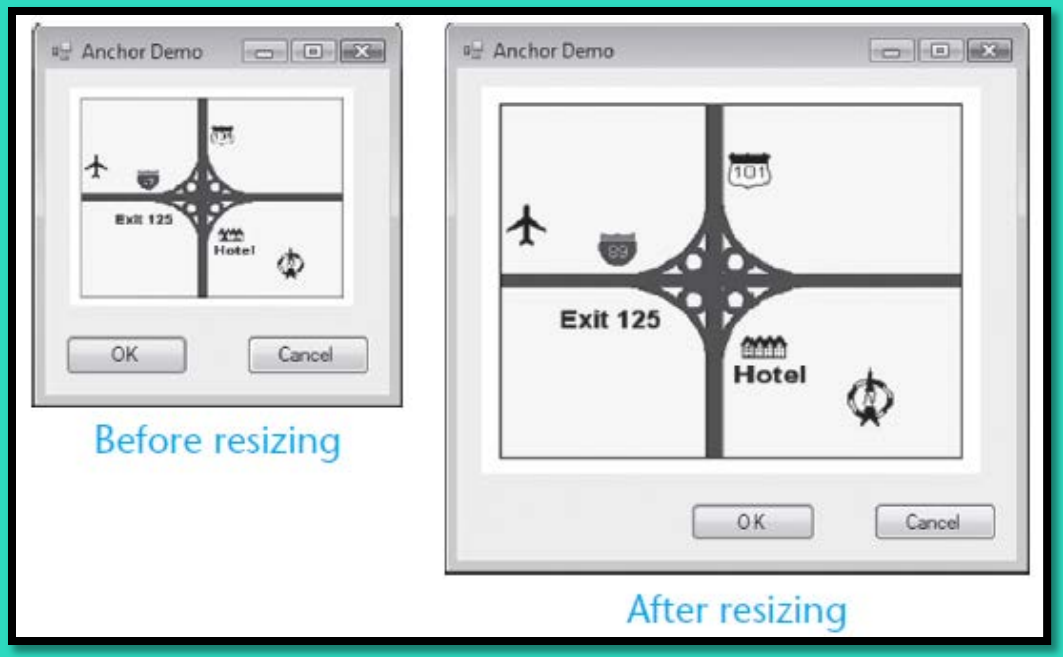
Controls have two properties, Anchor and Dock, which allow you to control the control's position on the form when the form is resized at runtime.

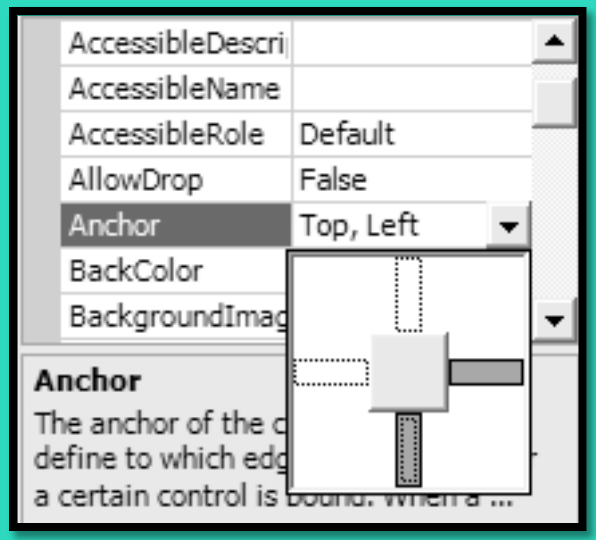
**Addison Wesley**
is an imprint of

PEARSON

# The Anchor Property

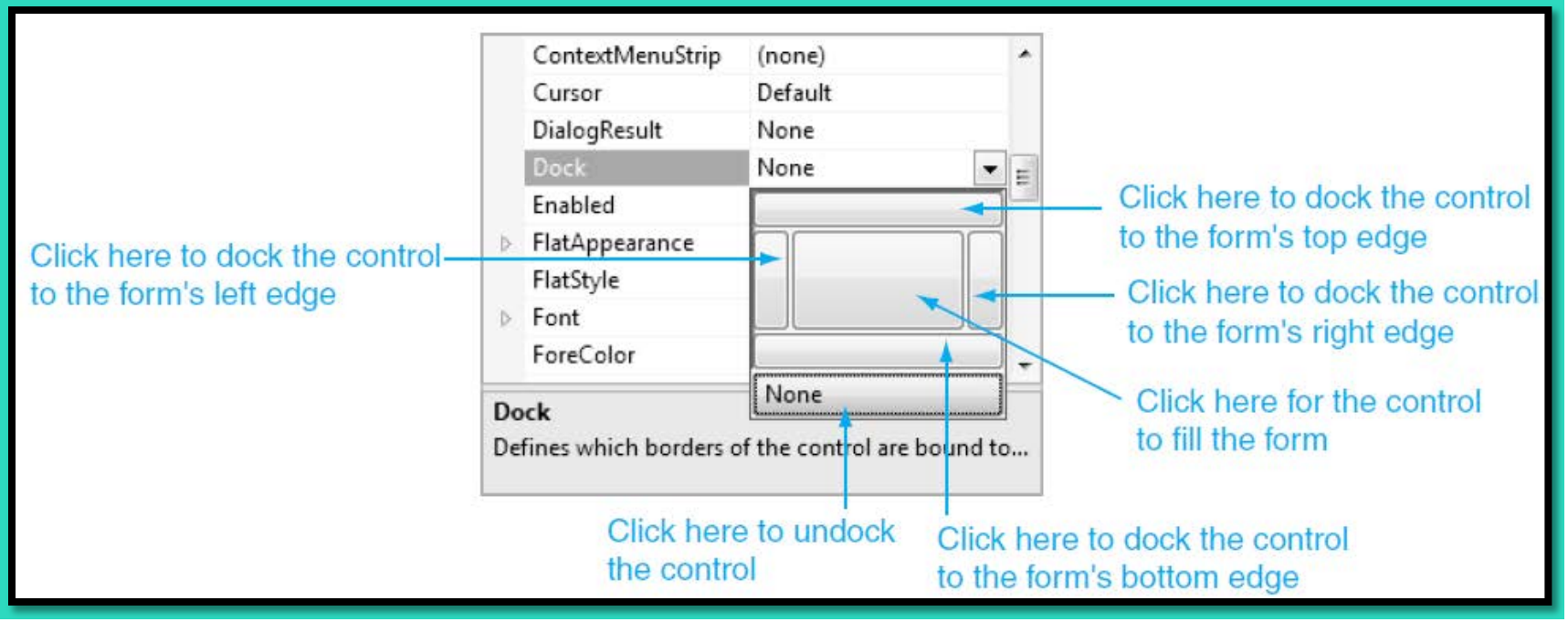
- The **Anchor property** allows you to anchor the control to one or more edges of a form
  - Controls are anchored to the top and left edges of the form by default

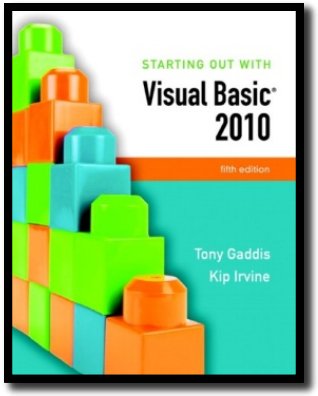


# The Dock Property

- The **Dock property** docks a control against a form's edge
  - Buttons are automatically sized to fill up in edge which they are docked






Section 8.7

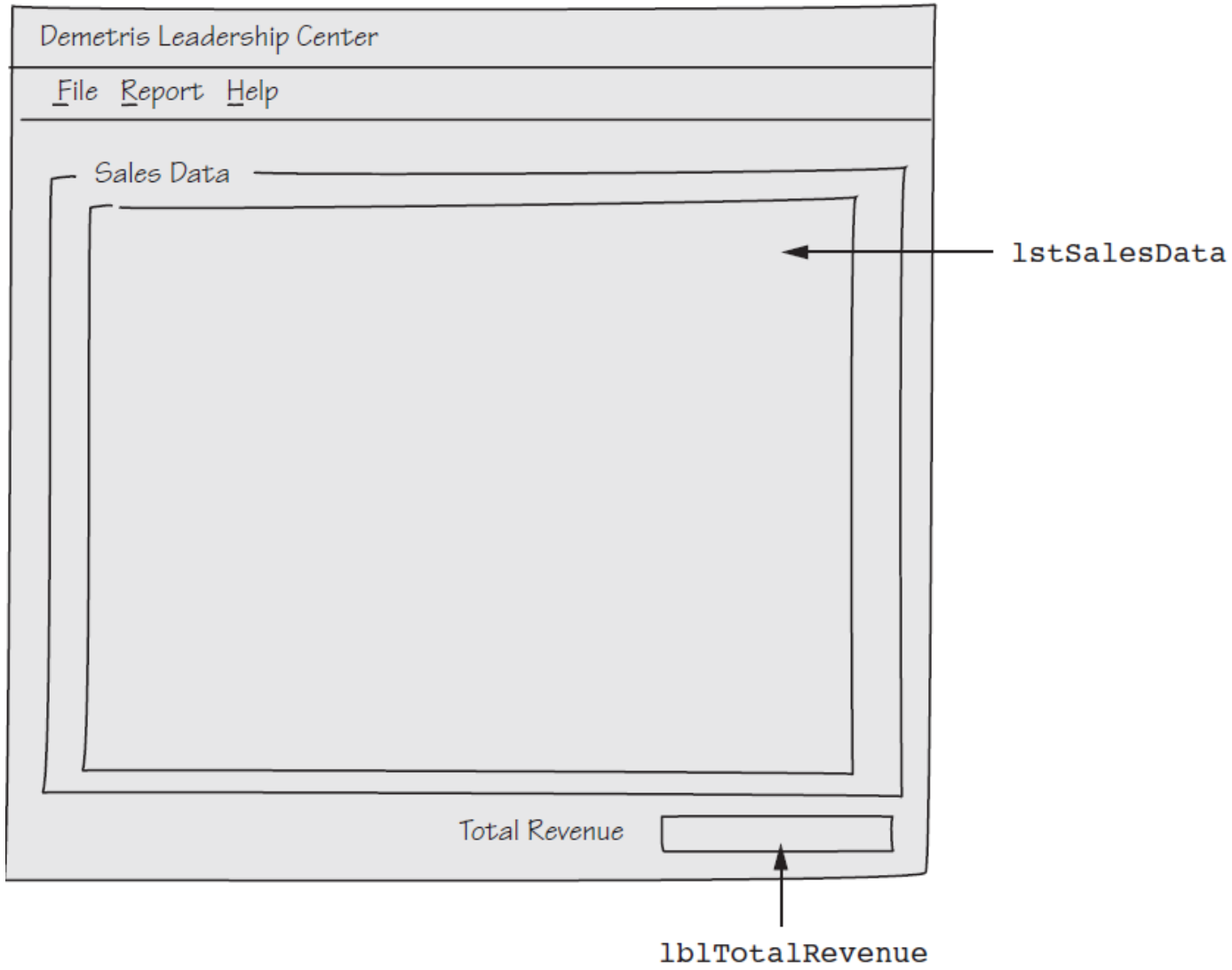
# FOCUS ON PROBLEM SOLVING: BUILDING THE DEMETRIS LEADERSHIP CENTER APPLICATION

In this section you build an application that uses data stored in parallel arrays.

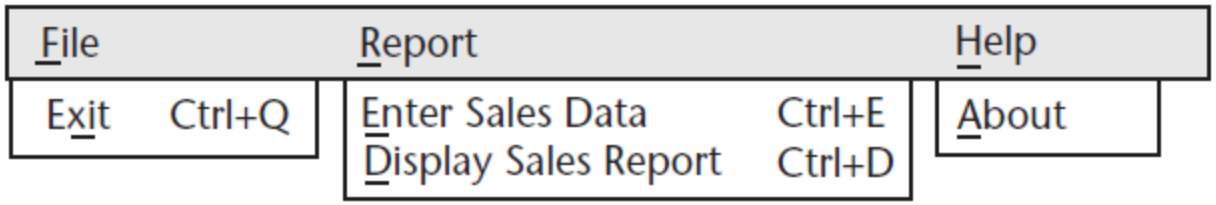
**Addison Wesley**
is an imprint of

PEARSON

# The Application's Form



# The Menu System




| Menu Item Name | Text | Shortcut Key |
|---|---|---|
| `mnuFile` | &File | |
| `mnuFileExit` | E&xit | Ctrl+Q |
| `mnuReport` | &Report | |
| `mnuReportData` | &Enter Sales Data | Ctrl +E |
| `mnuReportDisplay` | &Display Sales Report | Ctrl +D |
| `mnuHelp` | &Help | |
| `mnuHelpAbout` | &About | |

# Class-Level Declarations

| Name | Description |
|---|---|
| `IntMAX_SUBSCRIPT` | A constant, set to 8, holding the upper subscript of the class-level arrays, and the upper limit of counters used in loops that process information in the arrays |
| `strProdNames` | An array of strings; this array holds the names of the DLC products |
| `strDesc` | An array of strings; this array holds the descriptions of the DLC products |
| `intProdNums` | An array of integers; this array holds the product numbers of the DLC products |
| `decPrices` | An array of Decimal variables; this array holds the prices of the DLC products |
| `intUnitsSold` | An array of integers; this array holds the number of units sold for each of the DLC products |

# Methods

| Method | Description |
|---|---|
| `InitArrays` | Procedure; assigns the names, descriptions, product numbers, and unit prices of the DLC products to the class-level arrays |
| `mnuFileExit_Click` | Ends the application |
| `mnuReportData_Click` | Prompts the user for sales data |
| `mnuReportDisplay_Click` | Calculates and displays the revenue for each product and the total revenue |
| `mnuHelpAbout_Click` | Displays an *About* box |
| `Form1_Load` | Calls the `InitArrays` procedure |

# Sales Report Displayed