Chapter 8
Arrays

Chapter Objectives

- Establish an array and refer to individual elements in the array with subscripts
- Use a foreach loop to traverse the elements of an array
- Accumulate totals using arrays
- Write a table lookup for matching an array element
- Combine the advantages of list box controls with arrays
- Store and look up data in multidimensional arrays

Single-Dimension Arrays

- A list or series of values similar to a list of values for list boxes and combo boxes—without the box
- Use an array to store a series of variables for later processing
- May be referred to as tables or subscripted (or indexed) variables
- Individual variables (elements) are treated the same as any other variable and may be used in any statement
### Subscripts

- Advantage of an array is using variables for subscripts in place of constants
- Subscripts are constants, variables, or numeric expressions
- Must be integers
- Array name and number of elements are declared
  - No limit to the number of elements (except for available memory)

### The Declaration Statement for Arrays

- Declare arrays as `public` or `private`
  - Defaults to private
- Location of declaration determines scope and lifetime of array variables
- Declare an array by placing opening and closing square brackets after the data type
- Declare the number of elements or the initial values of array elements (which determine the number of elements)

### The Declaration Statement for Arrays - General Form

```csharp
Datatype[] arrayName = new Datatype[NumberOfElements];
Datatype[] arrayName = new Datatype{ InitialValueList};
Datatype[] arrayName = { InitialValueList};
[public|private] Datatype[] arrayName = new Datatype[NumberOfElements];
```
The Declaration Statement for Arrays - Examples

- Array subscripts are zero based
  - First element is always element zero
- All array elements must be the same data type

```csharp
string[] nameString = new string[25];
decimal[] balanceDecimal = new decimal[10];
int[] numbersInteger = new int[] {1, 5, 12, 18, 20};
string[] departmentsString = new string[] {"Accounting", "Marketing", "Human Relations"};
private string[] categoryString = new string[10];
public string[] IDNumbersString = new string[5];
string[] nameString = {"Sean", "Sam", "Sally", "Sara"};
```

Valid Subscripts

- A subscript must reference a valid element of an array
  - Zero through one less than the number of elements
- C# rounds fractional subscripts
- C# throws an exception for a subscript that is out of range
- Arrays are based on the System.Array collection

foreach Loops

- To reference each element of an array use for loops or foreach loops
- Advantages of a foreach loop
  - The subscripts of the array do not have to be manipulated
  - How many elements there are in an array does not have to be known
- Array elements are read only in the body of a foreach loop
- C# automatically references each element of the array and assigns its value to ElementName
  - Makes one pass through the loop per element
  - The variable used for ElementName must be the same data type as array elements or an Object data type
**foreach Loops – cont’d**

- Best to declare the variable for `ElementName` as part of the `foreach` statement to create a block-level variable
- An `foreach` loop will execute if the array has at least one element
- Loop continues to execute until all elements are processed
- Execution of code continues with the line following the closing braces when the loop finishes
- You can use a `break` statement to exit the loop early

**foreach Loops**

- **General Form**
  ```csharp
  foreach (DataType ElementName in ArrayName)
  {
    // Statement(s) in the loop.
  }
  ```

- **Examples**
  ```csharp
  foreach (string oneNameString in nameString)
  {
    Console.WriteLine(oneNameString); // Write one element of the array.
  }
  ```
  ```csharp
  foreach (decimal oneItemDecimal in allItemsDecimal) // Display all elements in the array.
  {
    totalsRichTextBox.Text += indexInteger++ + "" + totalDecimal.ToString() + "n";
  }
  ```

**Using Array Elements for Accumulators**

- Array elements are regular variables
  - Perform the same way as all variables used so far
  - Can be used for counters or total accumulators
- Eight totals will be accumulated to demonstrate the use of array elements as total accumulators
Adding to the Correct Total

- Use the totalDecimal array to hold the sales totals for eight scout troops
  - User enters the group number and sales
  - Subtract one from the group number and use it as a subscript for the correct total
  - Add the sales amount to the corresponding total
  - This technique is called direct reference of the array

```csharp
// Convert input group number to subscript.
groupNumberInteger = int.Parse(groupTextBox.Text) - 1;
// Add sale to the correct total.
saleDecimal = decimal.Parse(saleTextBox.Text);
totalDecimal[groupNumberInteger] += saleDecimal;
```

Table Lookup

- Values used to identify a series of elements may not be sequential
  - Establish a structure and declare an array of the structure
  - In the Form_Load event handler, initialize the array elements
  - Use a table lookup to find the correct array element, the subscript
  - Compare the input group number to each element in the table, one by one
  - When a match is found, add to the corresponding total

```csharp
public struct GroupInfo
{
    public string groupNumberString;
    public decimal totalDecimal;
}
public GroupInfo[] arrayGroup = new GroupInfo[8];
```
CIS 3260 – Intro to Programming with C#

Coding a Table Lookup

- **do loop works better than a foreach loop**
  - Logic of a lookup operation

  ```csharp
  // Convert input group number to a subscript.
  do
  { if (groupTextBox.Text == arrayGroup[groupNumberInteger].groupNumberString) 
    { // A match is found.
      decimal saleDecimal = decimal.Parse(salesTextBox.Text);
      arrayGroup[groupNumberInteger].totalDecimal += saleDecimal;
      foundBoolean = true;
      // Clear the controls.
      groupTextBox.Clear();
      salesTextBox.Clear();
      groupTextBox.Focus();
    }
  groupNumberInteger++;
  } while (groupNumberInteger < 8 && !foundBoolean);
  ```

Coding a Table Lookup – cont’d

- Validate group number entry
  - If wrong, display a message box
- Check foundBoolean after loop completes to determine why loop terminated
  - Did it terminate because of a match or without a match
- Table lookup works for any table, numeric or string
- Searched fields do not have to be in any sequence

Using List Boxes with Arrays

- Use a list box or combo box rather than a text box for user input
- Use the list’s SelectedIndex property to determine the array subscript
  - SelectedIndex property holds the position of the selected list item

  ```csharp
  groupNumberInteger = groupComboBox.SelectedIndex;
  ```
### Multidimensional Arrays

To define a two-dimensional array or table:
- The declaration statement specifies the number of rows and columns.
- The row is horizontal and the column is vertical.
- Specify the number of elements within parentheses and/or specify initial values.
- Specify the row with the first subscript and the column with the second subscript.
- Use a comma between the subscripts.
- Always use two subscripts when referring to individual elements of the table.

### Multidimensional Arrays – cont’d

Elements of an array are used the same way as other variables:
- Accumulators, counts, reference fields for lookup.
- In statements.
- As conditions.

Invalid references would include a value greater than the subscript for the row or column.

### General form

```
DataType[,] ArrayName = new DataType[NumberOfElements, NumberOfColumns];
```

### Examples

```
string[,] nameString = new string[3, 4];
string[,] nameString = new string[,] = {
    {"James", "Mary", "Sammie", "Sean"},
    {"Tom", "Lee", "Leon", "Larry" },
    {"Maria", "Margaret", "Jill", "John" }
};
```

- Both of these statements create arrays with 3 rows and 4 columns.
- Note the comma between square brackets.
- Specifies that the array has two dimensions.
Initializing Two-Dimensional Arrays

- Numeric array elements are initially set to zero
- String elements are initially set to an empty string
- You may need to initialize (or reinitialize) arrays to some other value
  - Nested for loops can set each array element to an initial value
  - Cannot use a foreach loop to initialize an array

Nested for Loop Example

```csharp
for (int rowInteger = 0; rowInteger < 3; rowInteger++)
{
    for (int columnInteger = 0; columnInteger < 4; columnInteger++)
    {
        nameString[rowInteger, columnInteger] = ""; // Initialize each element.
    }
}
```

Printing a Two-Dimensional Table

- Use a foreach loop to print one array element per line

```csharp
foreach (string elementString in nameString)
{
    // Set up a line.
    e.Graphics.DrawString(elementString, printFont, Brushes.Black,
                          printXFloat, printYFloat);
    // Increment the Y position for the next line.
    printYFloat += lineHeightFloat;
}
```
Use same techniques as for single dimension arrays
- Direct reference – If meaningful row and column subscripts are available
- Table lookup

Many 2D tables used for lookup will require additional one-dimensional arrays or lists to aid in the lookup process.

Example – User selects the weight from one list and the zone from the second list
- The SelectedIndex from the Weight list is used to find the correct row
- The SelectedIndex from the Zone list is used to find the correct column