Module 6:
Arrays

Declaring an Array and Assigning Values

- **Array**
  - A list of data items that all have the same data type and the same name
- Each object in an array is an **array element**
  - You can distinguish each element from the others in an array with a **subscript**
    - A subscript is also called an index
- Declaring and creating an array

```
double[] sales;
    sales = new double[20];
```

- **new operator**
  - Used to create objects
- You can change the size of an array

Declaring an Array and Assigning Values

- **Array element**
  - Each object in an array
- **Subscript** (or **index**)
  - An integer contained within square brackets that indicates the position of one of an array's elements
  - An array's elements are numbered beginning with 0
- "Off by one"
  - Remember to start counting at zero!
    - Item 1 in an array is Element [0]
    - Item 2 in an array is Element [1]

Declaring an Array and Assigning Values

- Assigning a value to an array element

  sales[0] = 2100.00;

- Accessing an element value

  double aSale = sales[19];

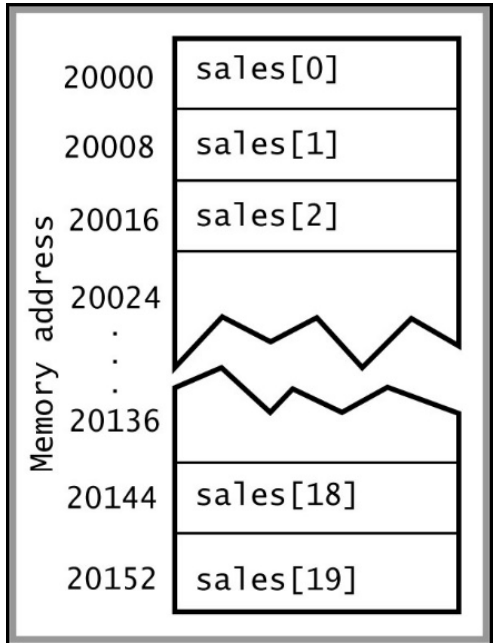


Figure 6-1 An array of 20 `sales` items in memory

Initializing an Array

- In C#, arrays are objects
  - Arrays are instances of a class named **System.Array**
- Initializing objects
  - Numeric fields are set to 0
  - Character fields are set to **null**
  - **bool** fields are set to **false**
- **Initializer list**
  - A list of values provided for an array

Initializing an Array

- An **Initializer list** is the list of values provided for an array
  - Examples (each does the same thing)

```
int[ ] myScores = new int[5] {100, 76, 88, 100, 90};

int[ ] myScores = new int[ ]  {100, 76, 88, 100, 90};

int[ ] myScores = {100, 76, 88, 100, 90};
```

- If you declare a size, then you must list a value for each element (example 1)
  - If you initialize an array with values, you are not required to give the array a size (example 2)
  - If you initialize an array, you do not need to use the keyword **new** (example 3)

Using the Length Property

- **Length property**
  - A member of the **System.Array** class
  - Automatically holds an array's length
  - If the subscript, or index, is greater than the last valid index (e.g., the index value is 20 for a 20-element array), you will get a System.IndexOutOfRangeException…the highest valid index value is 19

– Examples

```
int[ ] myScores = {100, 76, 88, 100, 90};
lblSize.Text = $"Array size is {myScores.Length}";
```

Using foreach

- **foreach statement**
  - Cycles through every array element without using a subscript
  - Uses a temporary **iteration variable** that automatically holds each array value in turn

```
double[] payRates = { 6.00, 7.35, 8.12, 12.45, 22.22 };
string output = string.Empty;
foreach (double aRate in payRates)
{
        output += aRate.ToString("c") + "\n";
}

lblRates.Text = output;
```

Using foreach

- The foreach statement is used only under certain circumstances:
  - Use when you want to access every array element
  - Since the iteration variable is **read-only**, you can access it but you cannot assign a value to it

Searching an Array Using a Loop

- Instead of creating a long series of if statements to search an array
  - There are a couple other options
- Searching options
  - Using a **for** loop
  - Using a **while** loop

Using a for Loop to Search an Array

- Example of using a for loop to search an array:

```
int[] validValues = { 2, 4, 6, 10, 12, 13, 16, 24 };
bool isValidItem = false;
int itemOrdered = 12;

for (int x = 0; x < validValues.Length; ++x)
{
    if (itemOrdered == validValues[x])
    {
        isValidItem = true;
    }
}
```

- Called a **Sequential Search** because each array element is examined in sequence
- **Parallel arrays** (a second array to hold corresponding data) allow you to use the same subscript to access additional information

Using a for Loop to Search an Array

```
int [] productIDs = { 101, 108, 210, 213, 266, 304, 311, 409, 411, 412 };
double[] productPrices = { 0.89, 1.23, 3.50, 0.69, 5.79, 3.19, 0.99, 0.89, 1.26, 8.00 };

  int orderedProductID = Convert.ToInt32(nudProductID.Value);
  double orderProductPrice = 0;
  bool isValidProductID = false;

  for (int x = 0; x < productIDs.Length; ++x)
  {
     if (orderedProductID == productIDs[x])
     {
        isValidProductID = true;
        orderProductPrice = productPrices[x];
     }
  }

  if (isValidProductID)
  {
     lblMessage.Text = $"Price of Ordered Product is {orderProductPrice.ToString("c")}";
  }
  else
  {
     lblMessage.Text = $"Sorry - Product ID {orderedProductID} is not valid";
  }
```

Improving a Loop's Efficiency

- In the code segment shown in the previous slide, you could change the comparison in the middle section of the for statement to a compound statement:

  for (int x = 0; x < productIDs.Length && !isValidProductID; ++x)

- If you decide to leave a loop as soon as a match is found
  - The most efficient strategy is to place the most common items first so they are matched sooner

- Some programmers disapprove of exiting a **for** loop early
  - Programs are easier to debug and maintain if each program segment has only one entry and one exit point
  - Select an approach that uses a **while** statement

Using a while Loop to Search an Array

```
int [] productIDs = { 101, 108, 210, 213, 266, 304, 311, 409, 411, 412 };
double[] productPrices = { 0.89, 1.23, 3.50, 0.69, 5.79, 3.19, 0.99, 0.89, 1.26, 8.00 };

int orderedProductID = Convert.ToInt32(nudProductID.Value);
double orderProductPrice = 0;
bool isValidProductID = false;

int x = 0;
while (x < productIDs.Length && !isValidProductID)
{
    if (orderedProductID == productIDs[x])
    {
        isValidProductID = true;
        orderProductPrice = productPrices[x];
    }
    ++x;
}

if (isValidProductID)
{
    lblMessage.Text = $"Price of Product with ID: {orderedProductID} is {orderProductPrice.ToString("c")}";
}
else
{
    lblMessage.Text = $"Sorry - Product ID: {orderedProductID} is not valid";
}
```

Using the BinarySearch(), Sort(), and Reverse() Methods

- The **System.Array** class contains a variety of useful, built-in methods that can search, sort, and manipulate array elements
- This section shows you how to use the methods:
  - **Array.BinarySearch()** – to find an element in an array
  - **Array.Sort()** – to sort an array's elements
  - **Array.Reverse()** – to reverse the order of elements
- If you include the statement using static System.Array();
  - You can use each of these methods without using the Array class name and the dot

Using the BinarySearch() Method

- **BinarySearch() method**
  - a sorted list of objects is split in half repeatedly as the search gets closer and closer to a match
  - Similar to the “guess a Number between 1 and 100” game
  - Start with Is it less than 50? Less than 25? Less than 12? Etc.
- When not to use **BinarySearch()**
  - If your array items are not arranged in ascending order
  - If your array holds duplicate values and you want to find all of them
  - If you want to find a range match rather than an exact match

Using the BinarySearch() Method

```
int [] productIDs = { 101, 108, 210, 213, 266, 304, 311, 409, 411, 412 };
double[] productPrices = { 0.89, 1.23, 3.50, 0.69, 5.79, 3.19, 0.99, 0.89, 1.26, 8.00 };

int orderedProductID = Convert.ToInt32(nudProductID.Value);
int matchedSub = -1;

matchedSub = Array.BinarySearch(productIDs, orderedProductID);

if (matchedSub >= 0)
{
    lblMessage.Text = $"Price of Product with ID: {orderedProductID} is
                                                    {productPrices[matchedSub].ToString("c")}";
}
else
{
    lblMessage.Text = $"Sorry - Product ID: {orderedProductID} is not valid";
}
```

Using the Sort() Method

- Sort() method
  - Arranges array items in ascending order
  - Works numerically for number types and alphabetically for characters and strings
  - Use it by passing the array name to **Array.Sort()**
  - Often used before using the **BinarySearch()** method which requires a sorted array

Using the Sort() Method

```
string[] names = { "Olive", "Patti", "Richard", "Ned", "Mindy" };
string output = string.Empty;

Array.Sort(names);

foreach (string aName in names)
{
    output += aName + "\n";
}

lblNames.Text = output;
```

Using the Reverse() Method

- **Reverse() method**
  - The Reverse() method does not sort array elements; it only rearranges their values to the opposite order
  - An element that starts in position 0 is relocated to position Length – 1 and an element that starts in position 1 is relocated to position Length – 2
  - Use it by passing the array name to the method

Using the Reverse() Method

```
string[] names = { "Olive", "Patti", "Richard", "Ned", "Mindy" };
string output = string.Empty;

Array.Sort(names);
Array.Reverse(names);

foreach (string aName in names)
{
    output += aName + "\n";
}

lblNames.Text = output;
```

Using Multidimensional Arrays

- **One-dimensional** or **single-dimensional array**
  - Picture it as a column of values
  - Elements can be accessed using a single subscript
- **Multidimensional array**
  - Requires multiple subscripts to access the array elements
- **Two-dimensional array**
  - Has two or more columns of values for each row
- R**ectangular array** (also called a **matrix**, or a **table**)
  - Each row has the same number of columns
  - Similar to a spreadsheet

Using Multidimensional Arrays

sales[0, 0]	sales[0, 1]	sales[0, 2]	sales[0, 3]
sales[1, 0]	sales[1, 1]	sales[1, 2]	sales[1, 3]
sales[2, 0]	sales[2, 1]	sales[2, 2]	sales[2, 3]

Figure 6-17 View of a rectangular, two-dimensional array in memory

CENGAGE

© 2018 Cengage. May not be copied, scanned, or duplicated, in whole or in part, except for use as permitted in a license distributed with a certain product or service or otherwise on a password-protected website for classroom use.

Using Multidimensional Arrays

- Initializing the array:

```
double[ , ] sales = {{14.00, 15.00, 16.00, 17.00},
                     {21.99, 34.55, 67.88, 31.99},
                     {12.03, 55.55, 32.89, 1.17}};
```

- The sales array contains three rows and four columns
- When you refer to an element in a two-dimensional array
  - The first value within the brackets following the array name always refers to the row
  - The second value, after the comma, refers to the column

Using Multidimensional Arrays

- Assume rent depends on the floor and number of bedrooms
  - **int[,] rents = { {400, 450, 510},**
    {500, 560, 630},
    {625, 676, 740},
    {1000, 1250, 1600} };

Floor	Zero Bedrooms	One Bedroom	Two Bedrooms
0	400	450	510
1	500	560	630
2	625	676	740
3	1000	1250	1600

Figure 6-18 Rents charged (in dollars)

Using Multidimensional Arrays

```
int[,] rents = { { 400, 450, 510 },
                 { 500, 560, 630 },
                 { 625, 676, 740 },
                 { 1000, 1250, 1600 } };

int floor = int.Parse(lstFloor.Text);
int bedrooms = int.Parse(lstNumberOfBedrooms.Text);

string rent = $"The rent for a {bedrooms} bedroom apartment on  the
{floor} floor is {rents[floor, bedrooms].ToString("c")}";

lblRent.Text = rent;
```

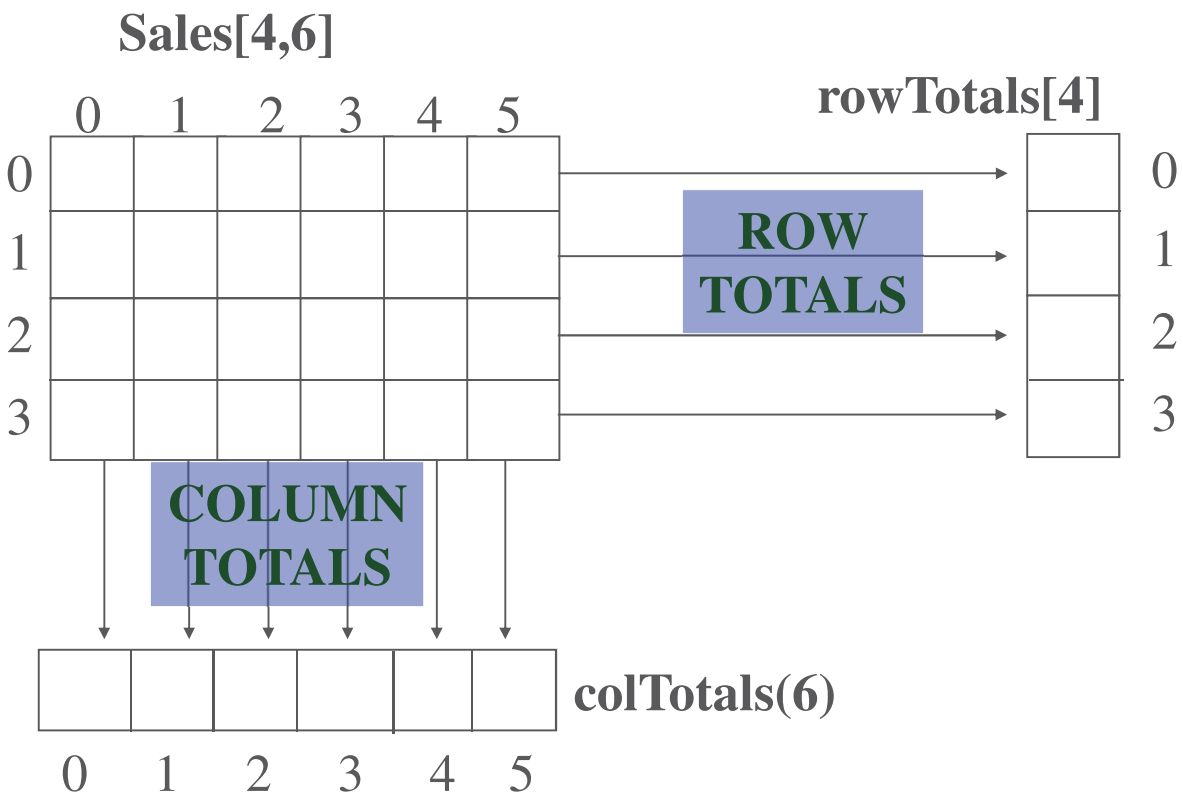
Initializing a Two-Dimensional Array

```
string[,] cheers = new string[2, 4];

for (int row = 0; row <= cheers.GetUpperBound(0); ++row)
{
    for (int col = 0; col <= cheers.GetUpperBound(1); ++col)
    {
        cheers[row, col] = "Go Rams";
    }
}
```

Passing an Array to a Method

```
private void DisplayTeams(string[] teams)
{
    string output = string.Empty;

    foreach (string aTeam in teams)
    {
    output += aTeam + "\n";
    }

        lblTeams.Text = output;
}

string[] afcWestTeams = { "Broncos", "Chargers", "Chiefs", "Raiders" };

DisplayTeams(afcWestTeams);
```

Summing Rows and Columns of a Two-Dimensional Array



Summing Rows and Columns of a Two-Dimensional Array

```
int[,] boxScores = { { 6, 7, 7, 3 }, { 3, 0, 7, 10 } };

int[] teamPoints = new int[boxScores.GetLength(0)];

int[] byQuarterPoints = new int[boxScores.GetLength(1)];

for (int row = 0; row <= boxScores.GetUpperBound(0); ++row)
{
    for (int col = 0; col <= boxScores.GetUpperBound(1); ++col)
    {
        teamPoints[row] += boxScores[row, col];
        byQuarterPoints[col] += boxScores[row, col];
    }
}
```