COIS1020H: Programming for Computing Systems

Chapter 6
Using Arrays

Declaring an Array and Assigning Values to Array Elements

- **Array**
  - –List of data items that all have the same data type and the same name
  - –Each item is distinguished from the others by 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

2

New=operator=request memory
Eg:

```
int[] bob;
bob=new int[20];
```

in this we declared an array “bob” which has 80 bytes memory . 80 because int s 4 bytes and e have declared array of 20 size, so 20*4=80

Also, we can write as

```
Int[] bob=new int[20];
```

Double[] sales=new double[20];
Sales=new double[10];
A New extra memory is there. So basically, the size of array has changed but the original or previous 20 are lost, that memory is basically wasted.

Declaring an Array and Assigning Values to Array Elements (cont'd.)

- **Array element**
  - –Each object in an array
- **Subscript** (or **index**)
  - –Integer contained within square brackets that indicates the position of one of an array's elements
  - –Array's elements are numbered beginning with 0
- **"Off by one"** error
  - –Occurs when you forget that the first element in an array is element 0

3

C/C++ lets us access memory except those in the array which is quite dangerous. but C# will show error when you try to access a memory not included in the array

Declaring an Array and Assigning Values to Array Elements (cont'd.)

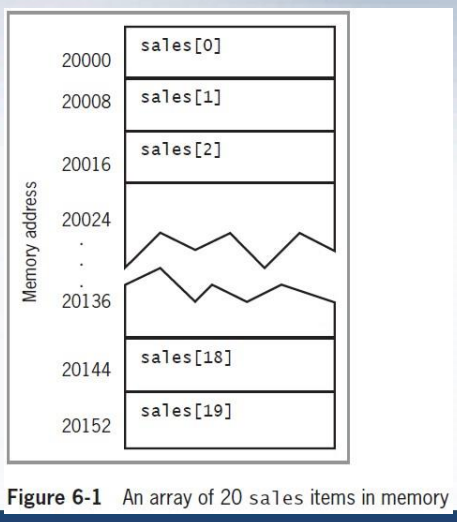


Figure 6-1 An array of 20 `sales` items in memory

4

We start from 0, because it makes finding the position easier.

If I want to find the position of sales[2] we can do that by

Starting_position + sizeOfDatatype*a …… (sales[a])

So here, 20000+2*8=20016

Declaring an Array and Assigning Values to Array Elements (cont'd.)

- Assigning a value to an array element

```
sales[0] = 2100.00;
sales[5] = Convert.ToDouble(Console.ReadLine());
```

- Printing an element value

```
Console.WriteLine(sales[19]);
```

- Using an array element in an equation

```
sales[7] = sales[6] + 1;
sales[i]++;
```

5

Sales=2100.00; is wrong because sales is an array and not only a double

Also, Console.WriteLine(sales); is also wrong because sales id not only a double but an entire array and to print an entire array we need to use loop

Initializing an Array

- In C#, arrays are objects
  - Arrays are instances of a class named `System.Array`
- Initializing objects
  - Numeric fields: 0
  - Character fields: '\u0000' or `null`
  - `bool` fields: `false`
- **Initializer list**
  - List of values provided for an array

Initializing an Array (cont'd.)

- Initializer list examples

```
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};
```

- C# is smart enough to determine the size of the array based on the initializer

7

C# can count the number of elements in an array and hence we don't need to specify the size if we are declaring like this.

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

Accessing Array Elements

- The power of arrays becomes apparent when you use subscripts
  - Can be variables rather than constant values
  - Arrays and `for` loops go hand in hand
- Using a loop to perform arithmetic on each element

```
for (int i = 0; i < 5; ++i)
  myScores[i] += 3;
```

8

If using loop, we are asking the program to print out array values, and we ask more than the size, it will show an error while execution of the program and not as a syntax error

Using the `Length` Property

- **`Length` property**
  - –Member of the `System.Array` class
  - –Array automatically has access to its length

- Examples

```
int[] myScores = {100, 76, 88, 100, 90};
Console.WriteLine("Array size is {0}", myScores.Length);
for (int i = 0; i < myScores.Length; ++i)
   Console.WriteLine(myScores[i]);
```

9

Length is the number of values stored in the array.

array.Length

Using `foreach`

- **`foreach` statement**
  - –Cycles through every array element without using a subscript
  - –Uses a temporary **iteration variable**
    - •Automatically holds each array value in turn
    - •Cannot change the value in the array with `foreach`
- •Example

```
double[] payRate = {6.00, 7.35, 8.12, 12.45, 22.22};
foreach(double money in payRate)
    Console.WriteLine("{0:C}", money);
```

- •Illegal

```
double[] payRate = {6.00, 7.35, 8.12, 12.45, 22.22};
foreach(double money in payRate)
    money = 0;
```

10

Using `foreach` (cont'd.)

- •To Summarize (used for the following):
  - –When you want to access every array element
  - –Since the iteration variable is **read-only**
    - •You cannot assign a value to it

11

Using `foreach` with Enumerations

```
using System;
public class DemoForEachWithEnum
{
   enum Day
   {
      SUNDAY, MONDAY, TUESDAY, WEDNESDAY,
         THURSDAY, FRIDAY, SATURDAY
   }
   public static void Main()
   {
      foreach(string day in Enum.GetNames(typeof(Day)))
         Console.WriteLine(day);
   }
}
```

Figure 6-2 DemoForEachWithEnum program

12

Searching an Array Using a Loop

- Searching options
  - –Using a `for` loop
  - –Using a `while` loop

13

Using a `for` Loop to Search an Array

- Use a `for` statement to loop through the array
  - Set a Boolean variable to `true` when match is found

```
int[] array = {23, 45, 98, 12, 45, 77};
boolfound = false;
intsearchItem= 45;
for (inti= 0; i< array.Length; ++i)
    if (searchItem== array[i])
      {
      found = true;
      location = i;
      }
```

14

Using a `for` Loop: Parallel Arrays

- **Parallel array:** an array with the same number of elements and corresponding data where the same subscript could access additional information

```
string [] name = {"Rich", "Mary", "Phil", Bob"};
int [] id = {1234, 4321, 3465, 2346}; double []
wage = {12.34, 10.76, 20.98, 13.87};
```

- `name[1]`, `id[1]` and `wage[1]` hold information about Mary

```
for(int i = 0; i < name.Length; ++i)
  Console.WriteLine("{0}'s id is {1} and earns {2:C}",
                    name[i], id[i], wage[i]);
```

Rich's id is 1234 and earns $12.34
Mary's id is 4321 and earns $10.76
Phil's id is 3465 and earns $20.98
Bob's id is 2346 and earns $13.87

15

Using a `for` Loop to Search an Array

```
using System;
public class FindPriceWithForLoop
{
   public static void Main()
   {
      int[] validValues = {101, 108, 201, 213, 266,
         304, 311, 409, 411, 412};
      double[] prices = {0.89, 1.23, 3.50, 0.69, 5.79,
         3.19, 0.99, 0.89, 1.26, 8.00};
      int itemOrdered;
      double itemPrice = 0;
      bool isValidItem = false;
      Console.Write("Please enter an item ");
      itemOrdered = Convert.ToInt32(Console.ReadLine());
      for(int x = 0; x < validValues.Length; ++x)
      {
         if(itemOrdered == validValues[x])
         {
            isValidItem = true;
            itemPrice = prices[x];
         }
      }
      if(isValidItem)
         Console.WriteLine("Price is {0}", itemPrice);
      else
         Console.WriteLine("Sorry - item not found");
   }
}
```

Figure 6-4 The FindPriceWithForLoop program

Please enter an item 266
Price is 5.79

Please enter an item 101
Price is 0.89

Please enter an item 267
Sorry – item not found

16

Using a `while` Loop to Search

```
using System;
public class FindPriceWithWhileLoop
{
   public static void Main()
   {
      int x;
      string inputString;
      int itemOrdered;
      double itemPrice = 0;
      bool isValidItem = false;
      int[] validValues = {101, 108, 201, 213, 266,
        304, 311, 409, 411, 412};
      double[] prices = {0.89, 1.23, 3.50, 0.69, 5.79,
        3.19, 0.99, 0.89, 1.26, 8.00};
      Console.Write("Enter item number ");
      inputString = Console.ReadLine();
      itemOrdered = Convert.ToInt32(inputString);
      x = 0;
      while(x < validValues.Length &&
        itemOrdered != validValues[x])
          ++x;
      if(x != validValues.Length)
      {
        isValidItem = true;
        itemPrice = prices[x];
      }
      if(isValidItem)
        Console.WriteLine("Item {0} sells for {1}",
          itemOrdered, itemPrice.ToString("C"));
      else
        Console.WriteLine("No such item as {0}",
          itemOrdered);
   }
}
```

Figure 6-6 The FindPriceWithWhileLoop program that searches with a while loop

Enter item number 409
Item 409 sells for $0.89

Enter item number 201
Item 409 sells for $3.50

Enter item number 410
No such item as 410

17

Another Version

```
using System;
public class FindPriceWithWhileLoop
{
   public static void Main()
   {
      int x = 0;
      string inputString;
      int itemOrdered;
      double itemPrice = 0; ;
      bool isValidItem = false;
      int[] validValues = { 101, 108, 201, 213, 266, 304, 311, 409, 411, 412 };
      double[] prices = { 0.89, 1.23, 3.50, 0.69, 5.79, 3.19, 0.99, 1.26, 8.00 };
      Console.Write("Enter item number");
      inputString = Console.ReadLine();
      itemOrdered = Convert.ToInt32(inputString);
```

18

Another Version (cont'd)

```
while (x < validValues.Length && !isValidItem)
      {
         if(itemOrdered == validValues[x])
         {
            isValidItem = true;
            itemPrice = prices[x];
         }
         ++x;
      }
      if (isValidItem)
         Console.WriteLine("Item {0} sells for {1:C}", itemOrdered, itemPrice);
      else
         Console.WriteLine("No such item as {0}", itemOrdered);
   }
}
```

Enter item number 409
Item 409 sells for $0.89

Enter item number 201
Item 409 sells for $3.50

Enter item number 410
No such item as 410

19

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

- `System.Array` class contains a variety of useful, built-in methods that can search, sort, and manipulate array elements

Using the `BinarySearch()` Method

- **`BinarySearch()` method**
  - –Finds a requested value in a sorted array
  - –Member of the `System.Array` class
- Do not use `BinarySearch()` under these circumstances
  - –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 (cont'd.)

```
using System;
public class BinarySearchDemo
{
   public static void Main()
   {
      int[] idNumbers = {122, 167, 204, 219, 345};
      int x;
      string entryString;
      int entryId;
      Console.Write("Enter an Employee ID ");
      entryString = Console.ReadLine();
      entryId = Convert.ToInt32(entryString);
      x = Array.BinarySearch(idNumbers, entryId);
      if(x < 0)
         Console.WriteLine("ID {0} not found", entryId);
      else
         Console.WriteLine("ID {0} found at position {1} ",
            entryId, x);
   }
}
```

These values must be sorted in ascending order for the `BinarySearch()` method to work correctly.

Figure 6-10 `BinarySearchDemo` program

Enter an Employee ID 219
ID 219 found at position 3

Enter an Employee ID 220
ID 220 not found

22

Using the `Sort()` Method

- **`Sort()` method**
  - –Arranges array items in ascending order
  - –Use it by passing the array name to `Array.Sort()`

```
using System;
public class SortArray
{
   public static void Main()
   {
      string[] names = {"Olive", "Patty",
         "Richard", "Ned", "Mindy"};
      int x;
      Array.Sort(names);
      for(x = 0; x < names.Length; ++x)
         Console.WriteLine(names[x]);
   }
}
```

```
Mindy
Ned
Olive
Patty
Richard
```

Figure 6-12 SortArray program

23

Using the `Reverse()` Method

- **`Reverse()` method**
  - –Reverses the order of items in an array
  - –Element that starts in position 0 is relocated to position `Length -1`
  - –Use it by passing the array name to the method

```
using System;
public class ReverseArray
{
   public static void Main()
   {
      string[] names = {"Zach", "Rose", "Wendy", "Marcia"};
      int x;
      Array.Reverse(names);
      for(x = 0; x < names.Length; ++x)
         Console.WriteLine(names[x]);
   }
}
```

```
Marcia
Wendy
Rose
Zach
```

Figure 6-14 ReverseArray program

24

Passing an Array to a Method

- You can pass a **single array** element to a method
  - – In same manner as you would pass a variable
  - – Variables are **passed by value**
- Local variables store a local copy of the value so the original array element can not be changed
- You can pass an **entire array** as a parameter
  - – Arrays, like all objects but unlike built-in types, are **passed by reference**
  - – Method receives actual memory address of the array
- Thus, the method has access to the actual values in the array elements and can change them

25

```
using System;
public class PassArrayElement
{
   public static void Main()
   {
      int[] someNums = {10, 12, 22, 35};
      int x;
      Console.Write("\nAt beginning of Main() method...");
      for(x = 0; x < someNums.Length; ++x)
         Console.Write("{0,6}", someNums[x]);
      Console.WriteLine();
      for(x = 0; x < someNums.Length; ++x)
         MethodGetsOneInt(someNums[x]);
      Console.Write("At end of Main() method.........");
      for(x = 0; x < someNums.Length; ++x)
         Console.Write("{0,6}", someNums[x]);
   }
   public static void MethodGetsOneInt(int oneVal)
   {
      Console.Write("In MethodGetsOneInt() {0}", oneVal);
      oneVal = 999;
      Console.WriteLine("     After change {0}", oneVal);
   }
}
```

Figure 7-13 PassArrayElement program

```
At beginning of Main() method…   10   12   22   35
In MethodGetsOneInt()  10        After change 999
In MethodGetsOneInt()  12        After change 999
In MethodGetsOneInt()  22        After change 999
In MethodGetsOneInt()  35        After change 999
At end of Main() method…         10   12   22   35
```

- **Pass a single element from an array to a Method**

26

```
using System;
public class PassEntireArray
{
   public static void Main()
   {
      int[] someNums = {10, 12, 22, 35};
      int x;
      Console.Write("\nAt beginning of Main() method...");
      for(x = 0; x < someNums.Length; ++x)
         Console.Write("{0, 6}", someNums[x]);
      Console.WriteLine();
      MethodGetsArray(someNums);
      Console.Write("At end of Main() method.........");
      for(x = 0; x < someNums.Length; ++x)
         Console.Write("{0, 6}", someNums[x]);
   }
   public static void MethodGetsArray(int[] vals)
   {
      int x;
      Console.Write("In MethodGetsArray() ");
      for(x = 0; x < vals.Length; ++x)
         Console.Write(" {0}", vals[x]);
      Console.WriteLine();
      for(x = 0; x < vals.Length; ++x)
         vals[x] = 888;
      Console.Write("After change");
      for(x = 0; x < vals.Length; ++x)
         Console.Write(" {0}", vals[x]);
      Console.WriteLine();
   }
}
```

Figure 7-15 PassEntireArray program

```
At beginning of Main() method…   10   12   22   35
In MethodGetsArray()  10  12  22  35
After change  888  888  888  888
At end of Main() method…          888   888   888   888
```

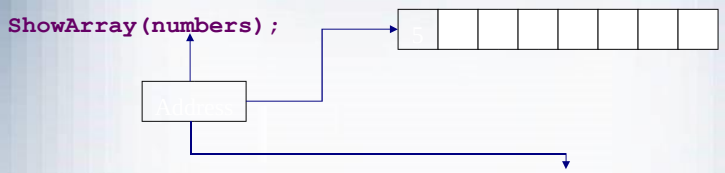
ntire array to a Method

27

•

Passing Arrays as Arguments (more)

- Arrays are objects.
- Their references can be passed to methods like any other object reference variable.




```
public static void ShowArray(int[] array)
{
    for (int i = 0; i < array.Length; i++)
        Console.Write("{0} ",array[i]);
}
```

PassArray.cs

```
// This program demonstrates a more complete example where arrays are
// passed between methods

using System;

 public class PassArray
 {
   public static void Main()
   {
      int[] numbers = new int[4];
      GetValues(numbers);
      Console.WriteLine("Here are the numbers that you entered:");
      ShowArray(numbers);
      Console.ReadLine();
   }
```

PassArray.cs (cont'd)

```
// The GetValues method accepts a reference to an array as its argument. The user is
// asked to enter a value for each element.
public static void GetValues(int[] array)
{
   Console.WriteLine("Enter a series of {0} numbers", array.Length);
   for (int i = 0; i < array.Length; i++)
   {
      Console.Write("Enter number {0}: ",(i + 1));
      array[i] = Convert.ToInt32(Console.ReadLine());
   }
}

// The ShowArray method accepts an array as an argument displays its contents.
public static void ShowArray(int[] array)
{
   for (int i = 0; i < array.Length; i++)
      Console.Write("{0} ",array[i]);
}
}
```

Output:

```
Enter a series of 4 numbers
Enter number 1: 5
Enter number 2: 9
Enter number 3: 23
Enter number 4: 12
Here are the numbers that you entered:
5 9 23 12
```

Returning an Array Reference

- A method can return a reference to an array.
- The return type of the method must be declared as an array of the right type.

```
public static double[] GetArray()
{
  double[] array = { 1.2, 2.3, 4.5, 6.7, 8.9 };
  return array;
}
```

- The GetArray method is a public static method that returns an array of doubles.
- Example: ReturnArray.cs

ReturnArray.cs

```
// This program demonstrates how an array can be returned from a method.
using System;

public class ReturnArray
{
   public static void Main()
   {
      double[] values;
      values = GetArray();
      for (int i = 0; i < values.Length; i++)
        Console.WriteLine("{0} ",values[i]);
      Console.ReadLine();
   }

 // GetArray method
public static double[] GetArray()
   {
      double[] array = {1.2, 2.3, 4.5, 6.7, 8.9};
      return array;
   }
}
```

Output:
1.2
2.3
4.5
6.7
8.9

Searching and Sorting Arrays

- In a selection sort:
  - The smallest value in the array is located and moved to element 0.
  - Then the next smallest value is located and moved to element 1.
  - This process continues until all of the elements have been placed in their proper order.

SelectionSortDemo.cs

```
// This program demonstrates the SelectionSort method in the ArrayTools class.
using System;
public class SelectionSortDemo
{
   public static void Main()
   {
      int[] values = {5, 7, 2, 8, 9, 1};
      // Display the unsorted array.
      Console.WriteLine("The unsorted values are:");
      for (int i = 0; i < values.Length; i++)
         Console.Write("{0} ", values[i]);
      Console.WriteLine();
      // Sort the array.
      SelectionSort(values);
      // Display the sorted array.
      Console.WriteLine("The sorted values are:");
      for (int i = 0; i < values.Length; i++)
         Console.Write("{0} ", values[i]);
      Console.WriteLine();
      Console.ReadLine();
   }
```

SelectionSortDemo.cs (cont'd)

```
// The SelectionSort method performs a selection sort on an
// int array. The array is sorted in ascending order.
//   **param array The array to sort.
public static void SelectionSort(int[] array)
{
   int startScan, index, minIndex, minValue;
   for (startScan = 0; startScan < (array.Length -1); startScan++)
   {
      minIndex = startScan;
      minValue = array[startScan];
      for(index = startScan + 1; index < array.Length; index++)
      {
         if (array[index] < minValue)
         {
            minValue = array[index];
            minIndex = index;
         }
      }
      array[minIndex] = array[startScan];
      array[startScan] = minValue;
   }
}
}
```

Output:
The unsorted values are:
5 7 2 8 9 1
The sorted values are:
1 2 5 7 8 9

Searching and Sorting Arrays

- A search algorithm is a method of locating a specific item in a larger collection of data.
- The *sequential search algorithm* uses a loop to:
  - sequentially step through an array,
  - compare each element with the search value, and
  - stop when
    - the value is found or
    - the end of the array is reached

SearchArray.cs

```
// This program sequentially searches an int array for a specified value.
using System;

public class SearchArray
{
   public static void Main()
   {
      int[] tests = { 87, 75, 98, 100, 82 };
      int results;
       // Search the array for the value 100.
      results = SequentialSearch(tests, 100);
       // Determine whether 100 was found and
      // display an appropriate message.
      if (results == -1)
         Console.WriteLine("You did not earn 100 on any test.");
      else
         Console.WriteLine("You earned 100 on test {0}",(results + 1));

      Console.ReadLine();
   }
```

SearchArray.cs (cont'd)

```
// The SequentialSearch method searches array for value.  If value is found in array, the element's
// subscript is returned. Otherwise, -1 is returned.
public static int SequentialSearch(int[] array, int value)
{
   int i,           // Loop control variable
       index;       // Element the value is found at
   bool found;      // Flag indicating search results
   // Element 0 is the starting point of the search.
   i = 0;
   // Store the default values element and found.
   index = -1;
   found = false;
   // Search the array.
   while (!found && i < array.Length)
   {
      if (array[i] == value)
      {
         found = true;
         index = i;
      }
      i++;
   }
   return index;
  }
}
```

Searching and Sorting Arrays

- A binary search:
  - Requires an array sorted in ascending order.
  - Starts with the element in the middle of the array.
  - If that element is the desired value, the search is over.
  - Otherwise, the value in the middle element is either greater or less than the desired value
  - If it is greater than the desired value, search in the first half of the array.
  - Otherwise, search the second half of the array.
  - Repeat as needed while adjusting start and end points of the search.

BinarySearchDemo.cs

```
// This program demonstrates the binary search method in the ArrayTools class.
using System;

public class BinarySearchDemo
{
   public static void Main()
   {
      // The values in the following array are sorted
      // in ascending order.
      int[] numbers = {101, 142, 147, 189, 199, 207, 222, 234, 289,
         296, 310, 319, 388, 394, 417, 429, 447, 521, 536, 600};
      int result, searchValue;
      char input;
```

BinarySearchDemo.cs (cont'd)

```
   do
   {
      // Get a value to search for.
      Console.Write("Enter a value to search for: ");
      searchValue = Convert.ToInt32(Console.ReadLine());
      // Search for the value
      result = BinarySearch(numbers, searchValue);
      // Display the results.
      if (result == -1)
         Console.WriteLine("{0} was not found.",searchValue);
      else
         Console.WriteLine("{0} was found at " + "index {1}",
            searchValue, result);
      // Does the user want to search again?
      Console.WriteLine("Do you want to search again? (Y or N): ");
      input = Convert.ToChar(Console.ReadLine());
   } while (input == 'y' || input == 'Y');
   Console.ReadLine();
}
```

BinarySearchDemo.cs (cont'd)

```
// The BinarySearch method performs a binary search on an
// integer array. The array is searched for the number passed
// to value. If the number is found, its array subscript is
// returned. Otherwise, -1 is returned indicating the value was
// not found in the array.
public static int BinarySearch(int[] array, int value)
{
   int first,      // First array element
       last,       // Last array element
       middle,     // Mid point of search
       position;   // Position of search value
   bool found;     // Flag
   // Set the inital values.
   first = 0;
   last = array.Length -1;
   position = -1;
   found = false;
```

BinarySearchDemo.cs (cont'd)

```
   // Search for the value.
    while (!found && first <= last)
    {
       // Calculate mid point
       middle = (first + last) / 2;
       // If value is found at midpoint...
       if (array[middle] == value)
       {
          found = true;
          position = middle;
       }
       // else if value is in lower half...
       else if (array[middle] > value)
          last = middle -1;
       // else if value is in upper half....
       else
          first = middle + 1;
    }
    // Return the position of the item, or -1 if it was not found.
    return position;
  }
}
```

Output:

```
Enter a value to search for: 101
101 was found at index 0
Do you want to search again? (Y or N):
y
Enter a value to search for: 123
123 was not found.
Do you want to search again? (Y or N):
y
Enter a value to search for: 388
388 was found at index 12
Do you want to search again? (Y or N):
n
```

Using Multidimensional Arrays

- **One-dimensional** or **single-dimensional array**
  - –Picture as a column of values
  - –Elements can be accessed using a single subscript
- **Multidimensional arrays**
  - –Require multiple subscripts to access the array elements
  - – **Two-dimensional arrays**
    - •Have two or more columns of values for each row
    - •Also called **rectangulararray** , **matrix**, or a **table**

44

Using Multidimensional Arrays (cont'd.)

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 6-16 View of a rectangular, two-dimensional array in memory

45

Using Multidimensional Arrays (cont'd.)

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

Figure 6-17 Rents charged (in dollars)

Using Multidimensional Arrays (cont'd.)

```
using System;
public class RentFinder
{
   public static void Main()
   {
      int[ , ] rents = { {400, 450, 510},
                         {500, 560, 630},
                         {625, 676, 740},
                         {1000, 1250, 1600} };
      int floor;
      int bedrooms;
      string inputString;
      Console.Write("Enter the floor on which you want to live ");
      inputString = Console.ReadLine();
      floor = Convert.ToInt32(inputString);
      Console.Write("Enter the number of bedrooms you need ");
      inputString = Console.ReadLine();
      bedrooms = Convert.ToInt32(inputString);
      Console.WriteLine("The rent is {0}",
         rents[floor, bedrooms]);
   }
}
```

Figure 6-18 The RentFinder program

Enter the floor on which you wan
live 2
Enter the number of bedrooms yo
need 1 The rent is 676

Using Multidimensional Arrays (cont'd.)

- **Jagged array**
  - When the rows of a two-dimensional array are of different lengths, the array is known as a ragged array.
  - You can create a ragged array by creating a two-dimensional array with a specific number of rows and a varying number of columns. `int[][] ragged = new int[4][];` – Then create the individual rows.

```
ragged[0] = new int[3];
ragged[1] = new int[4];
ragged[2] = new int[5];
ragged[3] = new int[6];
```

48

Passing an Array to a Method (more)

- You can pass a multidimensional array to a method
  - By indicating the appropriate number of dimensions after the data type in the method header
  - Example

```
public static void displaySales(int[,]sales)
```

- Jagged arrays
  - Insert the appropriate number of square brackets after the data type in the method header
  - Example

```
public static void displayInfo(int[][] ragged)
```

49

Pass2DArray.cs

```
// This program demonstrates methods that accept two-dimensional array as
arguments.
using System;

public class Pass2Darray
{
   public static void Main()
   {
      int[,] numbers = { { 1,  2,  3,  4  }, { 5,  6,  7,  8  }, { 9, 10, 11, 12  } };

      // Display the contents of the array.
      Console.WriteLine("Here are the values in the array.");
      ShowArray(numbers);

      // Display the sum of the array's values.
      Console.WriteLine("The sum of the values is {0} ", ArraySum(numbers));

      Console.ReadLine();
   }
```

Pass2DArray.cs (cont'd)

```
 // The ShowArray method displays the contents
 // of a two-dimensional int array.
public static void ShowArray(int[,] array)
{
   for (int row = 0; row < array.GetLength(0); row++)
   {
      for (int col = 0; col < array.GetLength(1); col++)
         Console.Write("{0} ",array[row,col]);
         Console.WriteLine();
   }
}
```

Pass2DArray.cs (cont'd)

```
// The ArraySum method returns the sum of the
// values in a two-dimensional int array.
public static int ArraySum(int[,] array)
{
   int total = 0;   // Accumulator

   for (int row = 0; row < array.GetLength(0); row++)
      for (int col = 0; col < array.GetLength(1); col++)
         total += array[row, col];

   return total;
}
}
```

Output:

```
Here are the values in the array.
1 2 3 4
5 6 7 8
9 10 11 12
The sum of the values is 78
```

Summary

- An array is a list of data items
  - –All of which have the same type and the same name
  - –Items are distinguished using a subscript or index
- In C#, arrays are objects of a class named `System.Array`
- The power of arrays becomes apparent when you begin to use subscripts
- Subscript you use remains in the range of 0 through `length -1`

53

Summary (cont'd.)

- `foreach` statement cycles through every array element without using subscripts
- You can compare a variable to a list of values in an array
- You can create parallel arrays to more easily perform a range match
- The `BinarySearch()` method finds a requested value in a sorted array
- The `Sort()` method arranges array items in ascending order

54

Summary (cont'd.)

- The `Reverse()` method reverses the order of items in an array
- You can pass an array as a parameter to a method
- Multidimensional arrays require multiple subscripts to access the array elements
- Types of multidimensional arrays
  - Two-dimensional arrays (rectangular arrays)
  - Jagged arrays

55