CGT 215 Lecture 7

Arrays

Data Structures

- ***Data structures*** are collections of related data items.

Arrays

- ***Arrays*** are data structures consisting of related data items of the same type.

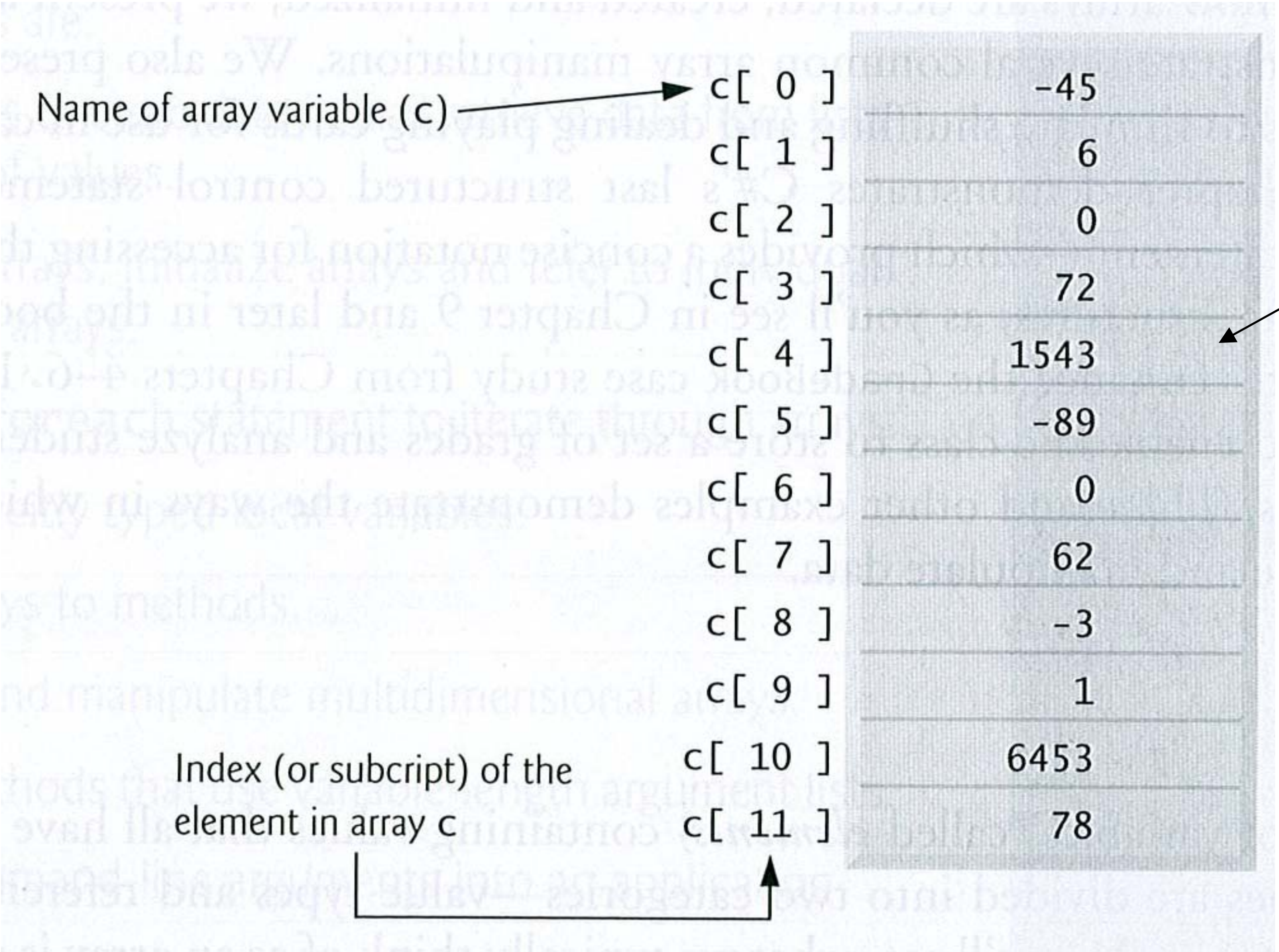
- Arrays are fixed-length entities – they remain the same length once they are created, although an array variable may be reassigned such that it refers to a new array of a different length.

Arrays

- An array is a group of variables (called elements) containing values that all have the same type.

- The position number of the element within the array is called the element’s ***index***

Arrays


Name of array variable (c)
Index (or subscript) of the element in array c

c[0]	-45
c[1]	6
c[2]	0
c[3]	72
c[4]	1543
c[5]	-89
c[6]	0
c[7]	62
c[8]	-3
c[9]	1
c[10]	6453
c[11]	78

Value stored in index 4 of array c

Or

Value stored in c[4]

Array indices

- The first element in every array has index zero and is sometimes called the ***zeroth element***.

Declaring – Single Dimension

- private int[] x;

- private int[] numbers; //declare numbers as an int array of any size

- private string[] words; //declare words as a string array of any size

- private Dog[] myDogs; //declare myDogs as a Dog array of any size

Creating a new instance

- After you declare the array, you can specify the size:

- numbers = new int[7]; //numbers is a 7-element array
- numbers = new int[15]; //now it's a 15-element array

- words = new string[5]; //words is a 5-element array
- words = new string[20]; //now it’s a 20-element array

- myDogs = new Dog[3]; //myDogs is an array of 3 Dogs
- myDogs = new Dog[30]; //now it’s a 30-element array

Initializing

- int[] numbers = new int[5] {1, 2, 3, 4, 5};

- string[] words = new string[3] {"Bottle", "Cup", "Art"};

- // Dog is a little more involved
  - private Dog doggie1, doggie2;
    …
  - doggie1 = new Dog();
  - doggie2 = new Dog();

  - Dog[] myDogs = new Dog[2] {doggie1, doggie2};

Setting/Retrieving values from array

- numbers[2]   //accesses the 3rd element of the array
- words[0]   //accesses the 1st element of the array
- myDogs[5]   //accesses the 6th element of the array

- numbers[3] = 5;
  - //sets the 4th element equal to the number 5
- words[1] = “aardvark”;
  - //sets the 2nd element equal to “aardvark”
- myDogs[2] = doggie1;
  - //sets the 3rd element equal to the dog object: doggie1

Setting/Retrieving values from array

- private int x;
- private string text;
- private Dog puppy;
- puppy = new Dog();

- x = numbers[4];
  - //retrieves the 5th element from numbers and stores the value into x
- text = words[0];
  - //retrieves the 1st element from words and stores the value into text
- puppy = myDogs[1];
  - //retrieves the 2nd element from myDogs and stores the value into puppy

Length of an array

- int lengthOfNums, lengthOfWords, lengthOfDogs;

- lengthOfNums = numbers.Length;
- lengthOfWords = words.Length;
- lengthOfDogs = myDogs.Length;

Length of an array

- //you can use a variable for the index number

-
  ```
  for( int i=0;  i < words.Length;  i++ )
  {
      tb.Text  +=  ( words[i].ToString() + “\r\n” );
  }
  ```
- //this for loop would produce:
  - Bottle
  - Aardvark
  - Art

Alternately – using foreach

- //Again, using a variable as the array index
- foreach(string **word** in words)

  {

  tb.Text += (**word**.ToString() + “\r\n”);

  }
- //this foreach loop would produce:
  - Bottle
  - Aardvark
  - Art

Common Programming Error 8.4

- The *foreach* statement can be used only to access array elements – it cannot be used to modify elements. Any attempt to change the value of the iteration variable in the body of a *foreach* statement will cause a compilation error.

Passing array into methods

- double[] hourlyTemp = new double[24];

- ModifyArray(hourlyTemp);

- public void ModifyArray(double[] ht)

  ```
  {
      //set the 1st element to the temperature 76.8 degrees
      ht[0] = 76.8;
  }
  ```

- //hourlyTemp[0] now equals 76.8

Multidimensional Arrays

- ***Multidimensional arrays*** with two dimensions are often used to represent *tables of values* consisting of information in *rows* and *columns*.

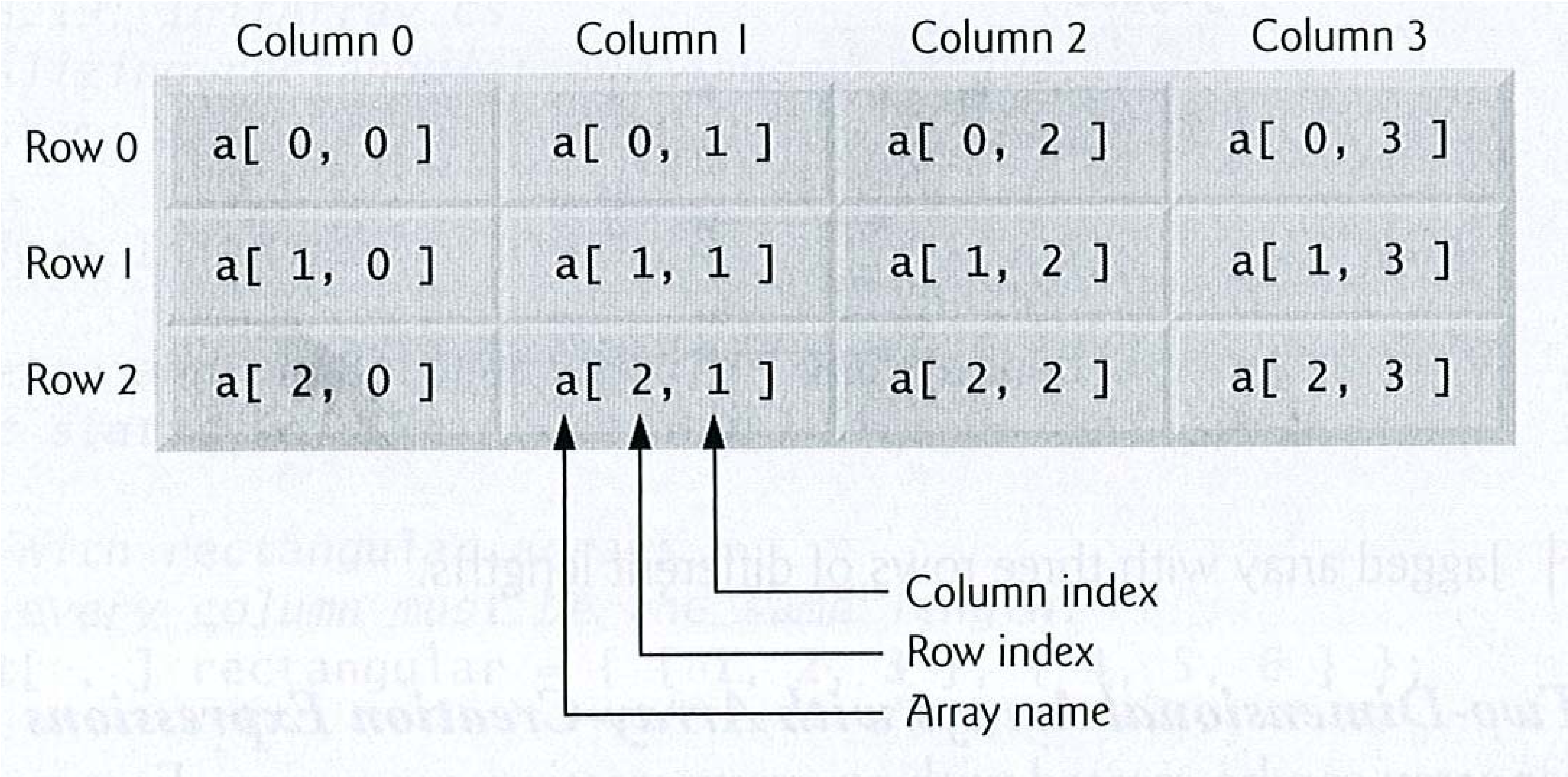
- Think of a two-dimensional array like a spreadsheet, rows and columns.

Rectangular Arrays

- ***Rectangular arrays*** are used to represent tables of information in the form of rows and columns, where each row has the same number of columns.

- An array with **m** rows and **n** columns is called an ***m-by-n array***

Rectangular Arrays



	Column 0	Column 1	Column 2	Column 3
Row 0	a[0, 0]	a[0, 1]	a[0, 2]	a[0, 3]
Row 1	a[1, 0]	a[1, 1]	a[1, 2]	a[1, 3]
Row 2	a[2, 0]	a[2, 1]	a[2, 2]	a[2, 3]

Declaring – Two Dimensional

- private int[,] x;

  - private int[,] counters;
    - //declare counters as a 2-dimensional int array of any size
  - private string[,] names;
    - //declare names as a 2-dimensional string array of any size
  - private cat[,] kittens;
    - //declare kittens as a 2-dimensional cat array of any size

Creating a new instance

- After you declare the array, you can specify the size:
  - counters = new int[7,7]; //counters has 7 rows and 7 cols
  - counters = new int[3,7]; //now it has 3 rows and 7 cols

  - names = new string[5,4]; //names has 5 rows and 4 cols
  - names = new string[2,2]; //now it has 2 rows and 2 cols

  - kittens = new cat[3,3]; //kittens has 3 rows and 3 cols
  - kittens = new cat[9,9]; //now it has 9 rows and 9 cols

Initializing (3 ways to do the same thing)

-
  ```
  int[,] counters = new int[2,3] {{1, 2, 3},
                                  {4, 5, 6}
                                 };
  ```
- **OR**

-
  ```
  int[,] counters  =  new int[,] {{1, 2, 3},
                                  {4, 5, 6}
                                 };
  ```
- **OR**

-
  ```
  int[,] counters  =  {{1, 2, 3},
                       {4, 5, 6}
                      };
  ```

Initializing (3 ways to do the same thing)

-
  ```
  string[,] names = new string[3,2]{{“Sam”, “Tom”},
                                    {“Pat”, “Jim”},
                                    {“Scott”, “Craig”}
                                   };
  ```

- **OR**

-
  ```
  string[,] names =  new string[,] {{“Sam”, “Tom”},
                                    {“Pat”, “Jim”},
                                    {“Scott”, “Craig”}
                                   };
  ```

- **OR**

-
  ```
  string[,] names =  {{“Sam”, “Tom”},
                      {“Pat”, “Jim”},
                      {“Scott”, “Craig”}
                     };
  ```

Initializing (3 ways to do the same thing)

- //cat is a little more involved
  - private cat kitten1, kitten2, kitten3, kitten4;
    …
  - kitten1 = new cat();
  - kitten2 = new cat();
  - kitten3 = new cat();
  - kitten4 = new cat();

- //continued on next slide…

Initializing (3 ways to do the same thing)

- //continued from previous slide…

-
  ```
  cat[,] litter = new cat[2,2] {{kitten1, kitten2},
                                {kitten3, kitten4}
                               };
  ```

- **OR**

-
  ```
  cat[,] litter =  new cat[,]  {{kitten1, kitten2},
                                {kitten3, kitten4}
                               };
  ```

- **OR**

-
  ```
  cat[,] litter =  {{kitten1, kitten2},
                    {kitten3, kitten4}
                   };
  ```

Declare & Initialize a 9x9 int array

```
private int[,] solution1 = { {7,9,2,3,5,1,8,4,6},
                             {4,6,8,9,2,7,5,1,3},
                             {1,3,5,6,8,4,7,9,2},
                             {6,2,1,5,7,9,4,3,8},
                             {5,8,3,2,4,6,1,7,9},
                             {9,7,4,8,1,3,2,6,5},
                             {8,1,6,4,9,2,3,5,7},
                             {3,5,7,1,6,8,9,2,4},
                             {2,4,9,7,3,5,6,8,1}
                           };
```

Declare a 3x8 array of integers

-
  ```
  //Declare array and variables
  private int[,] colors;
  private int r=0, g=1, b=2;   //rows

  //create new instance – 3 rows (r,g,b), 8 columns
  colors = new int[3,8];

  //set a value
  colors[r,3] = 1;  //set row 0, column 4 equal to 1
  colors[r,4] = 0;  //set row 0, column 5 equal to 0
  colors[r,5] = 1;  //set row 0, column 6 equal to 1

  colors[g,0] = 0;  //set row 1, column 1 equal to 0
  colors[g,1] = 0;  //set row 1, column 2 equal to 0
  colors[g,2] = 1;  //set row 1, column 3 equal to 1
  ```

Declare a 3x8 array of integers

- *//Alternately:*

```
private int[,] colors = {{0,0,0,0,0,0,0,0},
                          {0,0,0,0,0,0,0,0},
                          {0,0,0,0,0,0,0,0}
                         };

//then
colors[b,5] = 1;  //set row 2, column 6 equal to 1
colors[b,6] = 0;  //set row 2, column 7 equal to 0
colors[b,7] = 1;  //set row 2, column 8 equal to 1
```

Retrieving values from array

- counters[0,2]
  - //accesses the integer in the 1st row, 3rd column of the array
- names[1,0]
  - //accesses the string in the 2nd row, 1st column of the array
- cat[5,4]
  - //accesses the cat object in the 6th row, 5th column of the array

- counters[3,1] = 5;
  - //sets the integer in the 4th row, 2nd column of the array equal to the number 5
- names[1,3] = “Harry”;
  - //sets the string in the 2nd row, 4th column of the array equal to “Harry”
- cat[0,1] = kitten1;
  - //sets the cat object in the 1st row, 2nd column equal to the cat object: kitten1

Length of a 2-dimensional array

-
  ```
  int[,] solution  =  { {1,2,3,4},
                        {5,6,7,8},
                        {9,10,11,12}
                      };
  ```

- tb.Text += solution.Length.ToString();

  - //writes out: 12
  - //there are 12 values in the array

for loop for a 2-dimensional array

```
//rows
for (int i = 0; i < 3; i++)
{
    //cols
    for (int k = 0; k < 4; k++)
    {
        //check for last array item-don't put comma after last one
        if( ((i+1) * (k+1)) == solution.Length)
               tb.Text += ( solution[i,k].ToString() + “\r\n” );
        else
               tb.Text += ( solution[i,k].ToString() + “\r\n” );

    } //end inner for loop
} //end outer for loop

//writes out:   1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12
```

More Advanced…

- 3-dimensional array:
  - `int[,,] items = new int[3,4,5];`
- Jagged array:

  -
    ```
    int[][] numbers = {new int[]{1,2,3},
                       new int[]{4,5,6,7,8,9},
                       new int[]{3} };
    ```

- There are others…