# ARRAYS

SESSION 7

# Session Objectives

Identify single dimensional arrays

Identify two dimensional arrays

# Arrays

A group of elements, which are of the same size, same data type and have the same name

# Arrays (Contd.)

Program to add 5 numbers using 5 variables:

Start
Declare num1, num2, num3, num4, num5 and Sum as integers
Accept num1
Accept num2
Accept num3
Accept num4
Accept num5
Sum = num1+num2+num3+num4+num5
Display Sum
End

# Arrays (Contd.)

Disadvantages of the above program:

- Number of variables to be declared
- Repetition of the code

# Arrays (Contd.)

Program to add five numbers using a single variable and a loop

```
Declare num, sum and count as integers
sum is 0
count is 0
while (count <5)
do
   Accept value into num
   Add to the value in sum
   Increment the value of count by 1
enddo
Display sum
End
```

# Arrays (Contd.)

Declaration

**Array** num[5] as an integer

**Array** is just a word used in an algorithm to declare an array

**num[5]** translates to num[0],num[1]…num[4]

**5** is called the array bounds

The array bound sets the maximum number of elements that the array can handle

# Arrays (Example)

```
Start
Array num[5] is an integer
Declare sum, count as integers
sum=0
count=0
while (count<5)
do
   Accept value into num[count]
   Add num[count] to the value in sum
   Increment the value of count by 1
enddo
Display sum
End
```

# Array Index

Indicates the array element to be accessed

Also referred to as a subscript or the dimension

Written using the convention:

**Array_Name [Index]**

Typically starts at 0

# Arrays in C#

## Arrays are objects

- Arrays are allocated by new, manipulated by reference

*int [ ] a = new int [100];*

## Arrays can be initialized[Dense Array]

*int [ ] smallPrimes = {2,3,5,7,11,13};*

# The array data structure

An array is an indexed sequence of components

- Typically, the array occupies sequential storage locations
- The length of the array is determined when the array is created, and cannot be changed
- Each component of the array has a fixed, unique index
  - Indices range from a lower bound to an upper bound
- Any component of the array can be inspected or updated by using its index

# Arrays in C#

Array indices are integers

- The bracket notation `a[i]` is used (and not overloaded)
- Bracket operator performs bounds checking

An array of length `n` has bounds `0` and `n-1`

# Arrays in C#

## Arrays are reflective

- a.length is the length of array a

```
for (int i= 0; i < a .length; i++)
    Console.WriteLine(a[i]);
```

# Declaring an Array

An array must be declared before assigning a value to it. In C#, an integer array is declared as:

```
int[] num=new int[3]
```

Array names are chosen according to the same rules used for naming variables

Each element of an array can be used anywhere that a variable can be used

# Assigning Values to Array Elements

Assigning value to the third element of an integer array

num[2] = 10

Assigning values to a character array

**Array** var[3] is a character

var[0] = ‘L’

var[1] = ‘o’

var[2] = ‘g’

# Arrays (Contd.)

Retrieving values from an array

```
i = num[5]
```

Array elements in memory

num[5] is an integer

- Requires 5 integers * 4 bytes per integer = 20 bytes
- Array elements are stored in consecutive memory locations

# Subscripting

- Suppose

```
int [] A = new int[10];   // array of 10 integers A[0],… A[9]
```

- To access individual element must apply a subscript to list name **A**
  - A subscript is a bracketed expression also known as the index
  - First element of list has index 0

    **A[0]**
  - Second element of list has index 1, and so on

    **A[1]**
  - Last element has an index one less than the size of the list

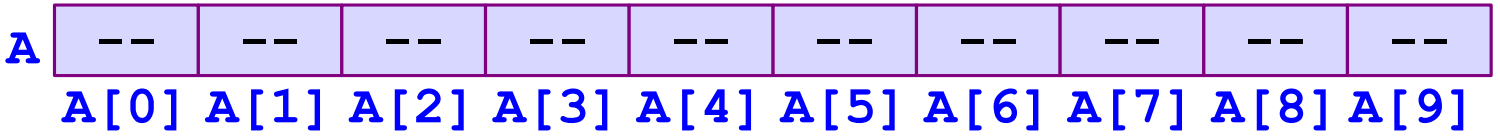
    **A[9]**
  - Incorrect indexing is a common error

    **A[10]  // does not exist**

# Array Elements

Suppose

```
int [] A = new int[10]; // array of 10 uninitialized ints
```

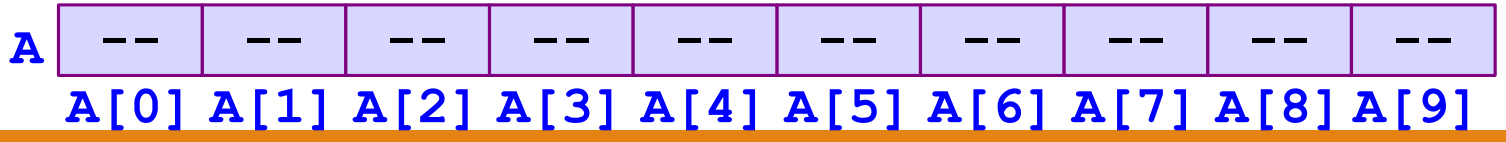



To access an individual element we must apply a subscript to list name `A`

# Array Element Manipulation

Consider

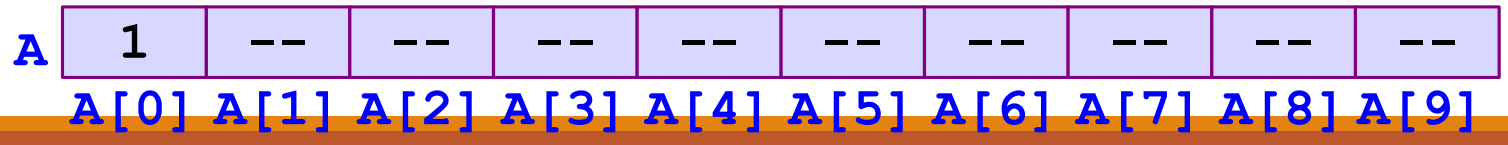
```
int i = 7, j = 2, k = 4;
A[0] = 1;
A[i] = 5;
A[j] = A[i] + 3;
A[j+1] = A[i] + A[0];
A[A[j]] = 12;
A[k]=Convert.ToInt32(Console.ReadLine()); // where next input value is 3
```



# Array Element Manipulation

Consider

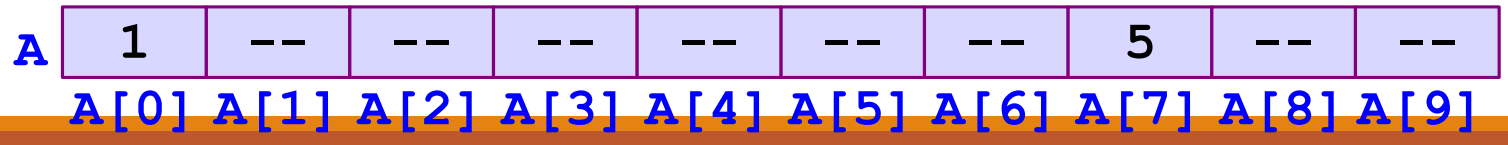
```
int i = 7, j = 2, k = 4;
A[0] = 1;
A[i] = 5;
A[j] = A[i] + 3;
A[j+1] = A[i] + A[0];
A[A[j]] = 12;
A[k]=Convert.ToInt32(Console.ReadLine()); // where next input value is 3
```



# Array Element Manipulation

Consider

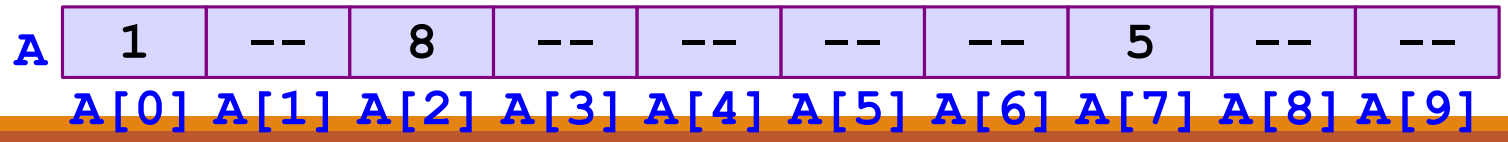
```
int i = 7, j = 2, k = 4;
A[0] = 1;
A[i] = 5;
A[j] = A[i] + 3;
A[j+1] = A[i] + A[0];
A[A[j]] = 12;
A[k]=Convert.ToInt32(Console.ReadLine()); // where next input value is 3
```



# Array Element Manipulation

Consider

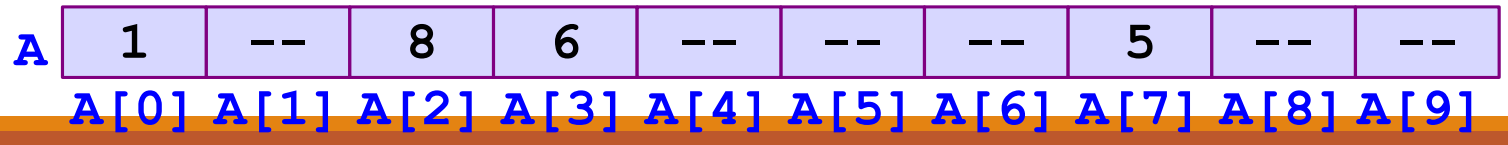
```
int i = 7, j = 2, k = 4;
A[0] = 1;
A[i] = 5;
A[j] = A[i] + 3;
A[j+1] = A[i] + A[0];
A[A[j]] = 12;
A[k]=Convert.ToInt32(Console.ReadLine()); // where next input value is 3
```



# Array Element Manipulation

Consider

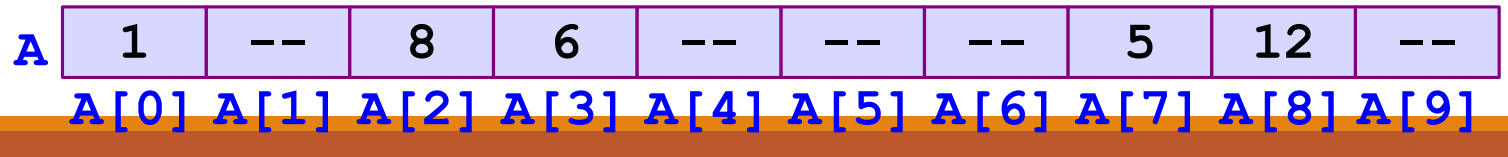
```
int i = 7, j = 2, k = 4;
A[0] = 1;
A[i] = 5;
A[j] = A[i] + 3;
A[j+1] = A[i] + A[0];
A[A[j]] = 12;
A[k]=Convert.ToInt32(Console.ReadLine()); // where next input value is 3
```



# Array Element Manipulation

Consider

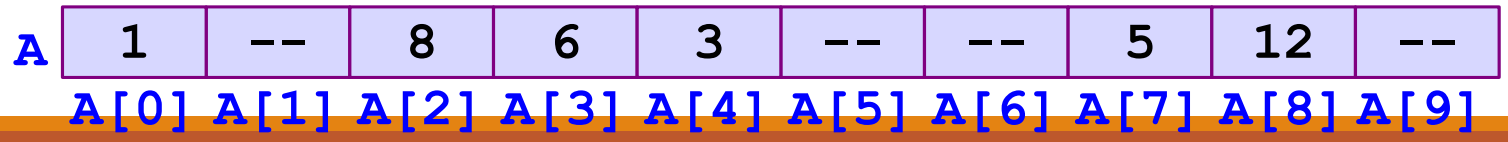
```
int i = 7, j = 2, k = 4;
A[0] = 1;
A[i] = 5;
A[j] = A[i] + 3;
A[j+1] = A[i] + A[0];
A[A[j]] = 12;
A[k]=Convert.ToInt32(Console.ReadLine()); // where next input value is 3
```



# Array Element Manipulation

Consider

```
int i = 7, j = 2, k = 4;
A[0] = 1;
A[i] = 5;
A[j] = A[i] + 3;
A[j+1] = A[i] + A[0];
A[A[j]] = 12;
A[k]=Convert.ToInt32(Console.ReadLine()); // where next
                                          input value is 3
```



# Two Dimensional Arrays

| | Physics | Chemistry | Mathematics |
|---|---|---|---|
| **Julia** | 45 | 60 | 90 |
| **Ben** | 20 | 67 | 92 |
| **Nicholas** | 90 | 35 | 56 |
| **Demi** | 78 | 50 | 80 |

This table consists of 4 rows and 3 columns

# Two-dimensional arrays II

In a 2D array, we generally consider the first index to be the row, and the second to be the column: a[*row, col*]

| | | *columns* | | | | |
|---|---|---|---|---|---|---|
| | | *0* | *1* | *2* | *3* | *4* |
| *rows* | *0* | *0,0* | *0,1* | *0,2* | *0,3* | *0,4* |
| | *1* | *1,0* | *1,1* | *1,2* | *1,3* | *1,4* |
| | *2* | *2,0* | *2,1* | *2,2* | *2,3* | *2,4* |
| | *3* | *3,0* | *3,1* | *3,2* | *3,3* | *3,4* |

# 2D arrays in C#

C# does support real 2D arrays

- `int [,] x;` denotes a 2D array `x` of *integer* components

We can *define* the above array like this:

```
int[,] x = new int[5,8];
```

and treat it as a regular 2D array

```
X[2,3] = 15;
```

# `foreach` Repetition Structure

The foreach repetition structure is used to iterate through values in arrays

No counter

A variable is used to represent the value of each element

```
foreach ( int Mark in Marks)
{
    if ( Mark < 50 )
      Console.WriteLine("Fail");
}
```

# Dynamic array

C# supports both static and dynamic arrays.

A static array has a fixed size and defined when an array is declared. The following code defines an array that can hold 5 int type data only.

**int**[] odds = **new int**[5];

A dynamic array does not have a predefined size. The size of a dynamic array increases as you add new items to the array. You can declare an array of fixed length or dynamic. You can even change a dynamic array to static after it is defined. The following code snippet declares a dynamic array where the size of the array is not provided.

**int**[] numArray = **new int**[] {};

# Cont….

Dynamic arrays can be initialized as static arrays. The following code snippet declares a dynamic array and initializes.

**int**[] numArray = **new int**[] { 1, 3, 5, 7, 9, 11, 13 };

# Dynamic array of a collection/Generic data type

```
List<string> courses = new List<string>();
courses.Add(“Computer Programming");
courses.Add(“Applied Calculus");
courses.Add(“Applied Physics");
courses.Add(“Computing Fundamental");

 foreach (string c in courses)
                    Console.WriteLine(c);
               Console.WriteLine(courses[1]);
```

# Cont….

A foreach loop can be used to iterate through the elements of an array.

```
// Dynamic array of string type
string[] strArray = new string[] { "Muhammad Zeeshan",
“Asif Raza", "Aleem Ahmed", “Azam Khan" };
foreach (string str in strArray)
Console.WriteLine(str);
```

# When to use dynamic arrays

Unless you limit an array size, the default arrays are dynamic arrays. Dynamic arrays are used when you're not sure about the number of elements an array can store. Static arrays should be used when you do not want an array to hold more items than its predefined size.

# ArrayList collection

C# ArrayList is a non-generic collection. The ArrayList class represents an array list and it can contain elements of any data types. The ArrayList class is defined in the System.Collections namespace. An ArrayList is dynamic array and grows automatically when new items are added to the collection

The ArrayList class is often used to work with a collection of objects. It is a dynamic collection and provides built-in methods to work with list items such as add items, remove items, copy, clone, search, and sort array.

# Cont…

```
static void Main(string[] args)
        {
           ArrayList personList= new ArrayList();
            personList.Add(“Asif");
            personList.Add(“Asim");
            personList.Add(24);
           foreach (var item in personList)
            {
               Console.WriteLine(item);
            }
        }
```

# ArrayList Common Methods

| Methods | Description |
|---|---|
| Add()/AddRange() | Add() method adds single elements at the end of ArrayList.<br>AddRange() method adds all the elements from the specified collection into ArrayList. |
| Insert()/InsertRange() | Insert() method insert a single elements at the specified index in ArrayList.<br>InsertRange() method insert all the elements of the specified collection starting from specified index in ArrayList. |
| Remove()/RemoveRange() | Remove() method removes the specified element from the ArrayList.<br>RemoveRange() method removes a range of elements from the ArrayList. |
| RemoveAt() | Removes the element at the specified index from the ArrayList. |
| Sort() | Sorts entire elements of the ArrayList. |
| Reverse() | Reverses the order of the elements in the entire ArrayList. |
| Contains | Checks whether specified element exists in the ArrayList or not. Returns true if exists otherwise false. |
| Clear | Removes all the elements in ArrayList. |
| CopyTo | Copies all the elements or range of elements to compitible Array. |
| IndexOf | Search specified element and returns zero based index if found. Returns -1 if element not found. |
| ToArray | Returns compatible array from an ArrayList. |

# Python Array

```
cars = ["Ford", "Volvo", "BMW"]

x = cars[0]

cars[0] = "Toyota“

x = len(cars)


for x in cars:
  print(x)

cars.append("Honda")

cars.remove("Volvo")

#Delete the second element of the cars array:

cars.pop(1)
```

# Conclusions

Arrays have the following advantages:

- Accessing an element by its index is very fast (constant time)

Arrays have the following disadvantages:

- All elements must be of the same type
- The array size is fixed and can never be changed
- Insertion into arrays and deletion from arrays is very slow.