Advanced Programming in C#

Arrays in C#

An array is a *contiguous* group of memory locations that all have the same type.

- To refer to a particular location or element in the array, we specify the name of the array and the **position number** of the particular element in the array.
- Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables.

Figure 1 below shows an integer array called **c** of size 12 **elements**.

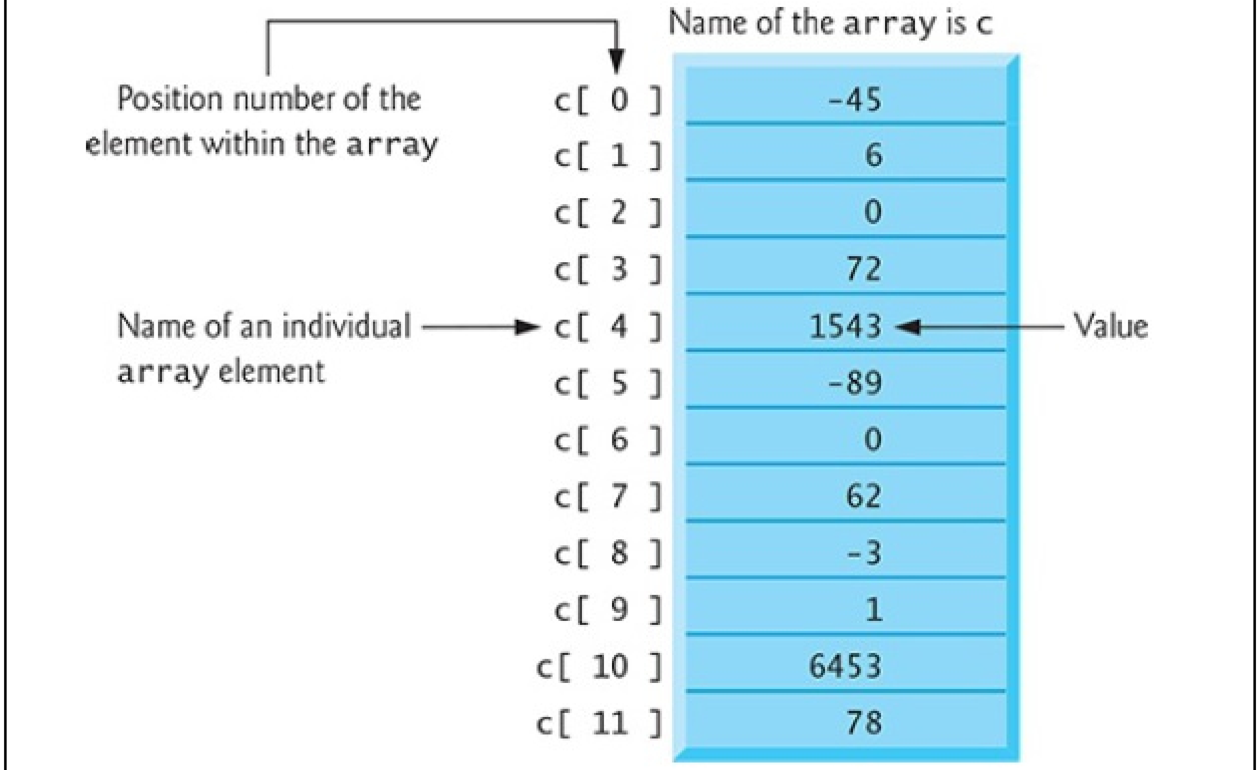

Figure 1 Array of 12 elements

- A specific element in an array is accessed by an index.
- You refer to anyone of these elements by giving the array name followed by the particular element’s *position number* in square brackets ([]).
- The position number is more formally called a **subscript** or **index** (this number specifies the number of elements from the beginning of the array).
- The first element has **subscript 0**(**zero**) and is sometimes called the **zeroth element**.
- Thus, the elements of array c are c[0] (pronounced “ c of zero”), c[1] , c[2]and so on.
- The highest subscript in array c is 11, which is 1 less than the number of elements in the array (12).
- Array names follow the same conventions as other variable names.

Declaring arrays

- To declare an array in C#, you can use the following syntax:

```
datatype[] arrayName;
```

where:

- `datatype` is used to specify the type of elements in the array.
- [ ] specifies the rank of the array. The rank specifies the size of the array.
- `arrayName` specifies the name of the array.

Example:

```
double[] balance; //declaring an array of type
double named balance
```

Initializing an Array

- Declaring an array does not initialize the array in the memory.
- When the array variable is initialized, you can assign values to the array.
- You need to use the **`new`** keyword to allocate memory for the array.

Example:

```
double[] balance = new double[10]; // balance is
an array of 10 double values
```

Assigning Values to an Array

- You can assign values to individual array elements, by using the index number, like:

```
double[] balance = new double[10];

balance[0] = 4500.0;
```

- You can assign values to the array at the time of declaration, as shown:

```
int[] marks = new int[5]  {99,98, 92, 97, 95};
```

- You can copy an array variable into another target array variable. In such case, both the target and source point to the same memory location:

```
int[] marks = new int[5] {99, 98, 92, 97, 95};

int[] score = marks;
```

Accessing Array Elements

- An element is accessed by indexing the array name. This is done by placing the index of the element within square brackets after the name of the array.

```
double salary = balance[9];
```

Declaring an array and Using a Loop to Initialize the array’s elements

- The following example, demonstrates the above-mentioned concepts declaration, assignment, and accessing arrays:
- The program below declares 10-element integer array n (line 1).
- Lines 4–7 use a for statement to initialize the array elements to 100 plus the index.
- The first output statement (line 11) displays the column headings for the columns printed in the subsequent for statement (lines 12–15), which prints the array in tabular format.
- Remember that “\t” prints 8 spaces and we used it here for formatting purposes.

```
1 int[] n = new int[10]; /* n is an array of 10
integers */
2             int i, j;
3    /* initialize elements of array n */
4    for (i = 0; i < 10; i++)
5      {
6       n[i] = i + 100;
7      }
8
9   /* output each array element's value */
10 Console.WriteLine("Element\tValue");
```

```
for (j = 0; j < 10; j++)
{
   Console.WriteLine(j + "\t" + n[j]);
 }
```

- When the above code is compiled and executed, it produces the following result:

```
Element    Value
0           100
1           101
2           102
3           103
4           104
5           105
6           106
7           107
8           108
9           109
```

Using the *foreach* Loop

- In the previous example, we used a for loop **`for`** accessing each array element.
- You can also use a **`foreach`** statement to iterate through an array.

```
int[] n = new int[10]; /* n is an array of 10
integers */
```

```
/* initialize elements of array n */
  for (int i = 0; i < 10; i++)
      {
        n[i] = i + 100;
      }

    /* output each array element's value */
    Console.WriteLine("Element\tValue");
    foreach (int j in n)
      {
        int i = j - 100;
        Console.WriteLine(i + "\t" + j);
      }
```

- When the above code is compiled and executed, it produces the same previous result

Accessing array elements:

- To access array elements, index the array name by placing the element's index in square brackets following the array name.

Example:

```
int[] b = new int[5]  {11, 45, 62, 70, 88};

Console.WriteLine (b[0]);  //Outputs 11
Console.WriteLine (b[3]);  //Outputs 70
```

- Index numbers may also be used to assign a new value to an element. For example:

```
b[2] = 49;
```

- This assigns a value of 49 to the array's third element. Always remember that the list of elements always begins with the index of 0.

Reading array elements from keyboard

- Array elements can be entered from keyboard using **`ReadLine()`** statement like any other primary data type variables but inside a loop in this case.

Example:

```
int[] marks = new int[10];

for(int i=0;i<10;i++)
{
 marks[i] = Convert.ToInt32(Console.ReadLine());
}
```

- In the above example, the user will be asked to enter the array elements one by one from the keyboard.

Arrays in Calculations

- The following code creates a program that uses a for loop to calculate the sum of all elements of an array.

```
int[]arr = new int[5] {11, 35, 62, 476, 989};
int sum = 0;
```

```
for (int x = 0; x < 5; x++) {
sum += arr[x];
}
 Console.WriteLine(sum);
//Outputs 1573
```

- In the example above, we declared an array and a variable sum that will hold the sum of the elements.
- Next, we utilized a for loop to iterate through each element of the array, and added the corresponding element's value to our sum variable.

Different Examples on arrays:

```
string[] cars = new string[4] {"Volvo", "BMW",
"Ford", "Nissan"};
double[] temperature = new double[30];
int[] marks = new int[25];
```

H.W:

1. Write a program in C# to find and print the sum of all elements of an **integer** array
2. Write a program in C# to count the frequency of each element of an **integer** array.
3. Write a program in C# to read an array of type **double** and create another array that contains the first one without duplicates.

4. Write a program in C# to read an array of type **int** and reverse the array and print the result. (**Note:** the original array should be reversed and you should use one array only)
5. Write a program in C# to read an integer array called **marks** of size $\boldsymbol{n}$. The program should create another array named **below,** which contains all marks that are below the average.
6. Write a program in C# to read an integer array called **marks** of size ***N***. The program should calculate and print the Standard Deviation of all the marks according to this formula:

$$\boldsymbol{SD = \sqrt{\frac{\sum(x_i - \mu)^2}{N}}}$$

where $\boldsymbol{x_i}$ represents the marks and μ represents the average of the marks.

7. Write a program in C# to read two double arrays of size **n** and check whether the two arrays are equal or not.
8. Write a program in C# to read an integer array of size n and check how many prime number the array contains.
9. Write a program in C# to read an integer array of size n and check if the array is reciprocal. (**Example:** 1 2 9 2 1 is reciprocal).
10. Write a program in C# to read two integer arrays of size n and create a third array that contains only the common elements of the two arrays.