Department of Computing,
Rajarata University of Sri Lanka

COM 1407
Computer Programming

LESSON 7 (PART I) – ARRAYS

BY :PIYUMI HERATH

Objectives

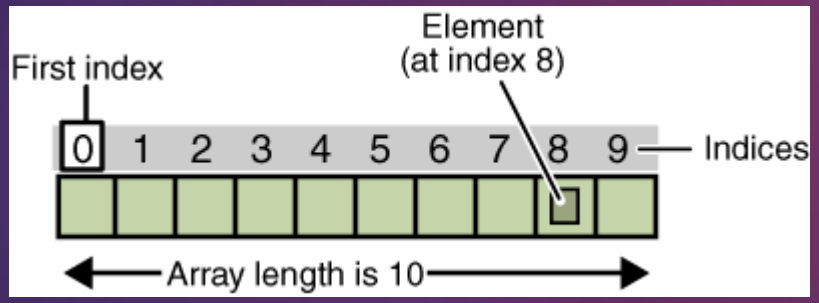
At the end of the lesson ,the students should be able to

- Be able to recognize when it would be appropriate to use one-dimensional arrays in a program.
- Be able to declare and use one-dimensional arrays.
- Know how to declare and use multidimensional(2D) arrays.
- Know how to use loops in I/O statements for input and output of multidimensional arrays.
- Be able to use and interpret multidimensional array expressions.

Arrays

- An *array* is a data structure, which can store a fixed-size collection of elements of the same data type
- Arrays are a convenient way of grouping a lot of variables under a single variable name.
- All arrays consist of contiguous memory locations. The lowest address corresponds to the first element/member and the highest address to the last element/member.
- Each element in an array has a numerical index, which is used to identify the element.
- Arrays have zero based indexing

One Dimensional/Linear Arrays



The number of subscript or index determines the dimensions of the array.

An array of one dimension is known as a one-dimensional array or 1-D array

Declaration

- To declare an array in C, you have to specify the type of the elements and the number of elements required by the array as follows −

  dataType arrayName [arraySize];

  Example:

  int prices[5];

  float marks [50];

  char grades [7];

Initializing Arrays

You can assign initial values to the elements of an array when they are declared. This is done by simply listing the initial values of the array, starting from the first element. Values in the list are separated by commas and the entire list is enclosed in a pair of braces.

- int counters[5] = { 0, 0, 0, 0, 0 };// declares an array called counters to contain five integer values and initializes each of these elements to zero
- int integers[5] = { 0, 1, 2, 3, 4 };
- char letters[5] = { 'a', 'b', 'c', 'd', 'e' };

- It is not necessary to completely initialize an entire array. If fewer initial values are specified, only an equal number of elements are initialized. The remaining values in the array are set to zero.

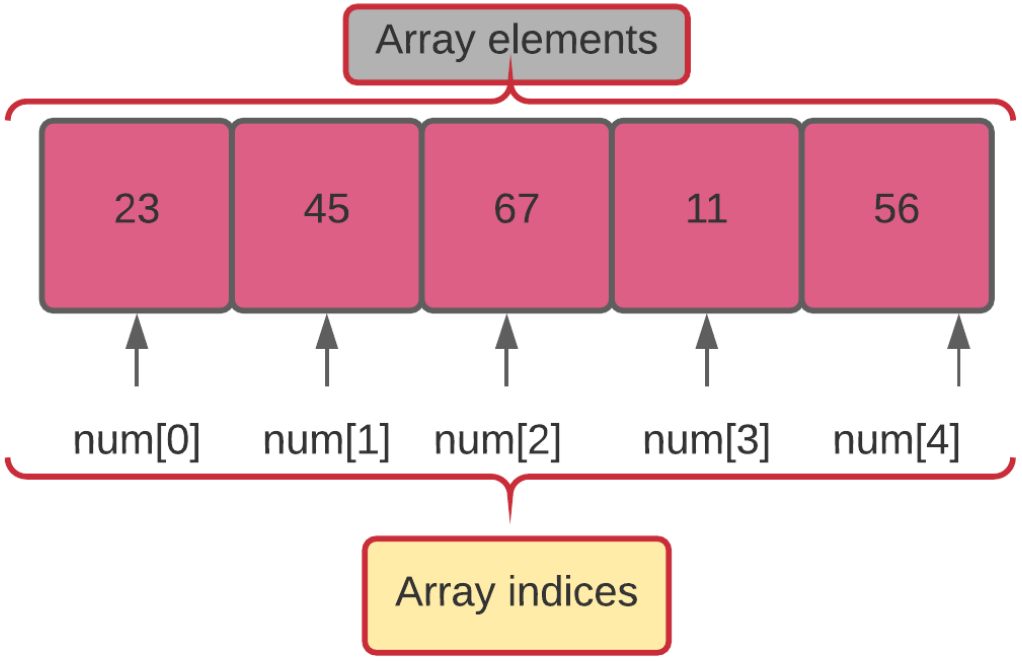
So the declaration :

float sampleData[500] = { 100.0, 300.0, 500.5 };

initializes the first three values of sampleData to 100.0, 300.0, and 500.5, and sets the remaining 497 elements to zero.

Here's an integer array named **num**

int num[ ] = {23,45,67,11,56}



- The C language allows you to define 1D arrays without specifying the number of elements.
- If this is done, the size of the array is determined automatically based on the number of initialization elements.

```
float temp[] = {12.3, 4.1, 3.8, 9.5, 4.5}; // an array of 5 floats
int arr[] = {11, 22, 33, 44, 55, 66, 77, 88, 99}; // an array of 9 ints
```

Accessing array elements

- Elements of an array are accessed by specifying the index (offset) of the desired element within square [ ] brackets after the array name.

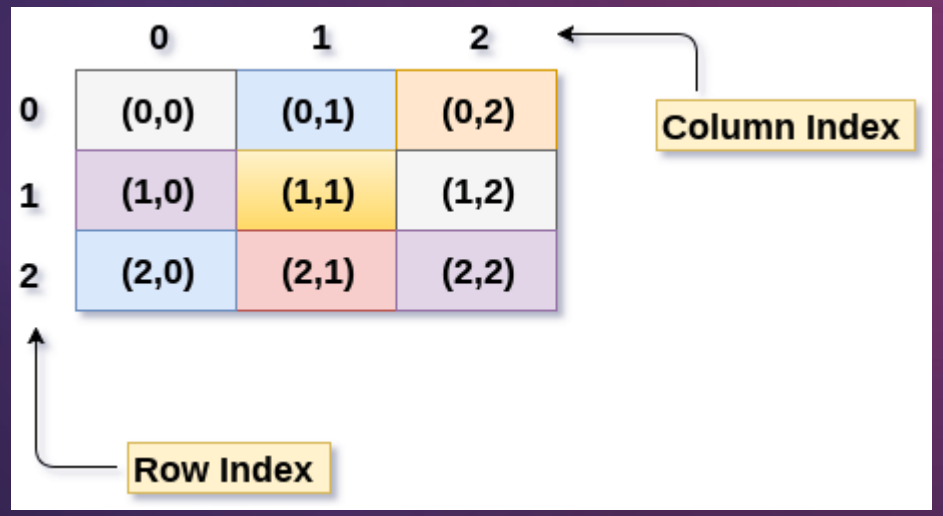
```
double salary = balance[9];
```

The above statement will take the 10th element from the array and assign the value to salary variable

Example:

```
#include <stdio.h>
int main ( )
{
        char word[] = { 'H', 'e', 'l', 'l', 'o', '!' };
        int i;
        for ( i = 0; i < 6; ++i ){
                printf ("%c\n", word[i]);
        }
return 0;
}
```

Multi Dimensional Arrays



The C language allows arrays of any dimension to
be defined.

A **multi-dimensional array** is an array that has more than one dimension. It is an array that has multiple levels.

The simplest multi-dimensional array is the **2D array**, or two-dimensional array. It's technically an array of arrays

Declaring 2D arrays

The elements of a 2D array are arranged in rows and columns. The size of the array dimensions has to be given within two pairs of square brackets.

Syntax:

dataType arrayName[number_of_rows][number_of_columns];

Example**:**

int age[3][4];

Initializing 2D arrays

When we initialize a 1D array during declaration, we need not to specify the size of it.

But with 2D arrays, you **must always specify the second dimension** even if you are specifying elements during the declaration.

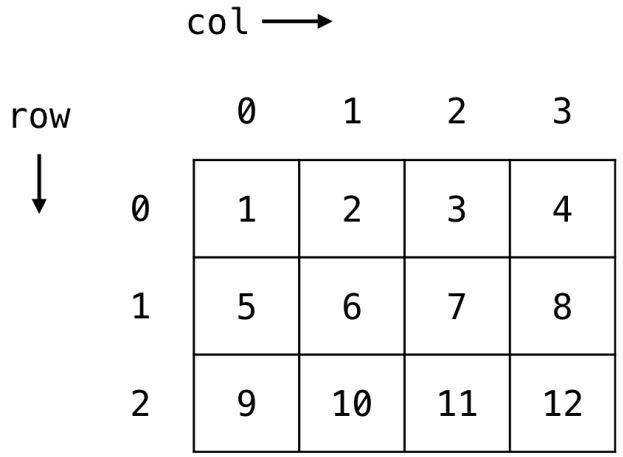
- int abc[2][2] = {1, 2, 3 ,4 } /* Valid declaration*/
- int abc[][2] = {1, 2, 3 ,4 } /* Valid declaration*/
- int abc[][] = {1, 2, 3 ,4 } /* Invalid declaration – you must specify second dimension*/
- int abc[2][] = {1, 2, 3 ,4 } /* Invalid because of the same reason mentioned above*/

- Different ways to initialize two-dimensional array

```
int c[2][3] = {{1, 3, 0}, {-1, 5, 9}};
int c[][3] = {{1, 3, 0}, {-1, 5, 9}};
int c[2][3] = {1, 3, 0, -1, 5, 9};
```

- The 2D array is organized as matrices which can be represented as the collection of rows and columns. Here, *num* is a two-dimensional (2d) array. The array can hold 12 elements. You can think the array as a table with 3 rows and each row has 4 columns.

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



Access 2D array Elements

An element in a two-dimensional array is accessed by using the subscripts, i.e., row index and column index of the array.

Example :

```
int val = a[2][3];
```

The above statement will take the 4th element from the 3rd row of the array.

Ex : Check the following program where a nested loop is used to access all the elements of a two-dimensional array

```
#include<stdio.h>
int main()
{
        int row,column;
        int arr[4][3]={{1,2,3},{2,3,4},{3,4,5},{4,5,6}};

        //traversing 2D array
        for(row=0;row<4;row++)
        {
                for(column=0; column<3; column++)
                {
                        printf("arr[%d] [%d] = %d \n",row,column,arr[row][column]);
                 }//end of column
        }//end of row
return 0;
}
```

Exercise:

- What are the advantages and disadvantages of using arrays in programming?
- List down five applications of arrays.

References

- Chapter 07 (Pages 95- 117) -Programming in C, 3rdEdition, Stephen G. Kochan

Thank You!

Next Lesson : C Structures