

CC-112L

Programming Fundamentals

Lab 09

Arrays – I

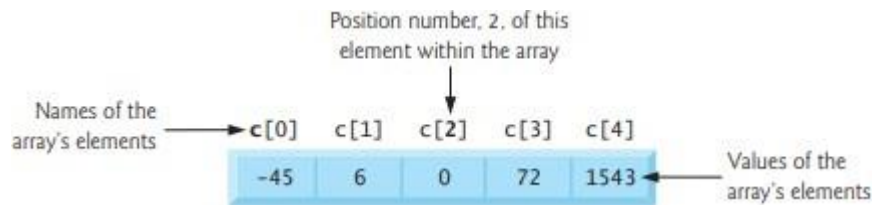
Version: 2.0.0

Release Date: 01-12-2023

**Department of Information Technology University
of the Punjab
Lahore, Pakistan**

Array:

An array is a group of elements of the same type stored contiguously in memory. The following figure shows an integer array containing five elements:



To refer to a particular location or element in the array, we specify the array's name, followed by the element's position number in square brackets (`[]`). The first element is located at position number 0. The position number is called the element's subscript (or index). A subscript must be a non-negative integer or integer expression.

Defining Array:

When you define an array, you specify its element type and number of elements. The following definition reserves five elements for integer array "A", which has subscripts (indexes) in the range 0-9.

```
int A[10];
```

Array Elements:

Like variables, uninitialized array elements contain garbage values. The following figure uses for statements to set array elements to one.

```
1  #include<stdio.h>
2
3  int main()
4  {
5      int Array[5]; //Defining array
6
7      for (int i = 0; i < 5; i++)
8      {
9          Array[i] = 1; //Setting Element at location i to 1
10     }
11 }
```

Initializer List:

We can initialize an array's elements when defining the array by providing a comma separated list of array initializers in braces `{}`.

If there are fewer initializers than array elements, the remaining elements are initialized to 0.

```

1  #include<stdio.h>
2
3  int main()
4  {
5      int Array1[5] = {1,2,3,4,5}; //Initializing an Array with Initializer List
6
7      int Array2[5] = { 1 }; // Initializing First Element to 1 and other elements are initialized to zero
8  };

```

Static Local Array:

A static local array is created and initialized once, not each time the function is called. This reduces program execution time. If static array is not explicitly initialized, then that array's elements are initialized to zero by default.

Following code contains a static local array:

```

1  #include<stdio.h>
2  void function();
3  int main(){
4      //First Call
5      function();
6      //Second Call
7      function();
8  };
9  void function()
10 {
11     static int Array[3] = { 1,2,3 };
12     for (int i = 0; i < 3; i++){
13         printf("Value at index %d is: %d\n", i, Array[i]);
14     }
15     printf("\n");
16     //Adding 5
17     for (int i = 0; i < 3; i++){
18         Array[i] += 5;
19         printf("Value at index %d is: %d\n", i, Array[i]);
20     }
21     printf("\n");
22 }

```

The execution of above program is as follows:

- Execution starts from the main()
- At line 5 execution shifts to the function
- At line 11 array elements are initialized to 1,2,3 respectively
- At line 13 & 19, Array element are displayed 1,2,3 & 6,7,8 respectively
- As line 7 execution shifts again to the function
- Now at line 11 array elements are not reinitialized and they retain the value 6,7,8 respectively
- At line 13 & 19, Array element are displayed 6,7,8 & 11,12,13 respectively



```
Microsoft Visual Studio Debug Console
Value at index 0 is: 1
Value at index 1 is: 2
Value at index 2 is: 3

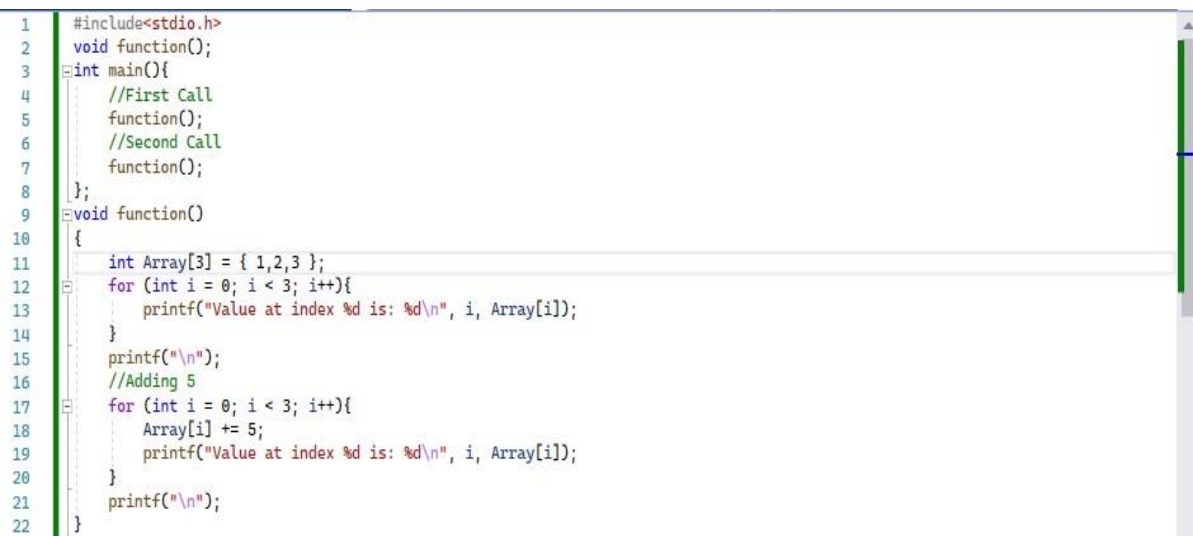
Value at index 0 is: 6
Value at index 1 is: 7
Value at index 2 is: 8

Value at index 0 is: 6
Value at index 1 is: 7
Value at index 2 is: 8

Value at index 0 is: 11
Value at index 1 is: 12
Value at index 2 is: 13
```

Automatic Local Array:

A static automatic array is reinitialized each time the function is called.



```
1  #include<stdio.h>
2  void function();
3  int main(){
4      //First Call
5      function();
6      //Second Call
7      function();
8  };
9  void function()
10 {
11     int Array[3] = { 1,2,3 };
12     for (int i = 0; i < 3; i++){
13         printf("Value at index %d is: %d\n", i, Array[i]);
14     }
15     printf("\n");
16     //Adding 5
17     for (int i = 0; i < 3; i++){
18         Array[i] += 5;
19         printf("Value at index %d is: %d\n", i, Array[i]);
20     }
21     printf("\n");
22 }
```

The execution of above program is as follows:

- Execution starts from the main()
- At line 5 execution shifts to the function
- At line 11 array elements are initialized to 1,2,3 respectively
- At line 13 & 19, Array element are displayed 1,2,3 & 6,7,8 respectively
- As line 7 execution shifts again to the function
- Now at line 11 array elements are reinitialized to the value 1,2,3 respectively
- At line 13 & 19, Array element are displayed 1,2,3 & 6,7,8 respectively

```
Microsoft Visual Studio Debug Console

Value at index 0 is: 1
Value at index 1 is: 2
Value at index 2 is: 3

Value at index 0 is: 6
Value at index 1 is: 7
Value at index 2 is: 8

Value at index 0 is: 1
Value at index 1 is: 2
Value at index 2 is: 3

Value at index 0 is: 6
Value at index 1 is: 7
Value at index 2 is: 8
```

Passing Arrays to Functions:

To pass an array argument to a function, specify the array's name without any brackets. E.g., If array has been defined as:

int Array[5];

The function call will be

function (Array, 5);

And function definition will be

void function (int Array[], int size);

```
1  #include<stdio.h>
2  void function(int Array[], int size);
3  int main(){
4      int Array[3] = { 1,2,3 };
5
6      function(Array,3);
7  };
8  void function(int Array[], int size)
9  {
10     Array[2] = 1;
11 }
```

Passing Individual Element:

To pass an individual array element to a function, the function call use the subscripted array name in argument.

function (Array[2]);

and function definition will be

void function (int element);

```
1  #include <stdio.h>
2  void function(int element);
3
4  int main() {
5      int Array[3] = { 1,2,3 };
6
7      function(Array[2]);
8  }
9
10 void function(int element)
11 {
12     element = 1;
13 }
```

Multidimensional Arrays:

Arrays can have multiple subscripts. A common use of multidimensional arrays is to represent tables of values consisting of information arranged in rows and columns. To identify a particular table element, we specify two subscripts:

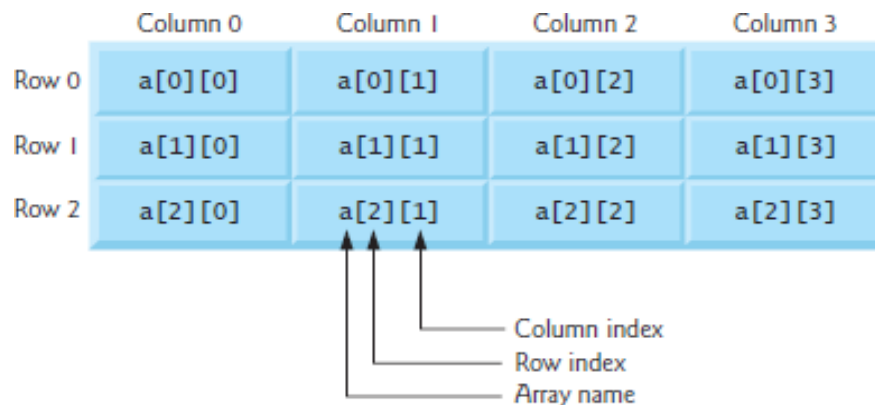
- The first (by convention) identifies the element's row.
- The second (by convention) identifies the element's column.

Arrays that require two subscripts to identify a particular element commonly are called two-dimensional arrays. Multidimensional arrays can have more than two subscripts.

Illustrating a Two-Dimensional Array:

The array contains three rows and four columns, so it's said to be a 3 x 4 array. In general, an array with **m** rows and **n** columns is called an **m x n** array. Every element in array **a** is identified by a name of the form **a[i][j]**, where **a** is the array name, and **i** and **j** are the subscripts that uniquely identify each element. The element names in row 0 all have the first subscript 0.

Pictorial Representation:



Initializing a Double-Subscripted Array:

You can initialize a multidimensional array when you define it. For example, you can define and initialize the two-dimensional array **int b [2][2]** with:

```
int b [2][2] = {{1, 2}, {3, 4}};
```

The values in the initializer list are grouped by row in braces. The values in the first set of braces initialize row 0 and the values in the second set of braces initialize row 1. So, the values 1 and 2 initialize elements **b [0][0]** and **b [0][1]**, respectively, and the values 3 and 4 initialize elements **b [1][0]** and **b [1][1]**, respectively. If there are not enough initializers for a given row, that row's remaining elements are initialized to 0. So, the definition:

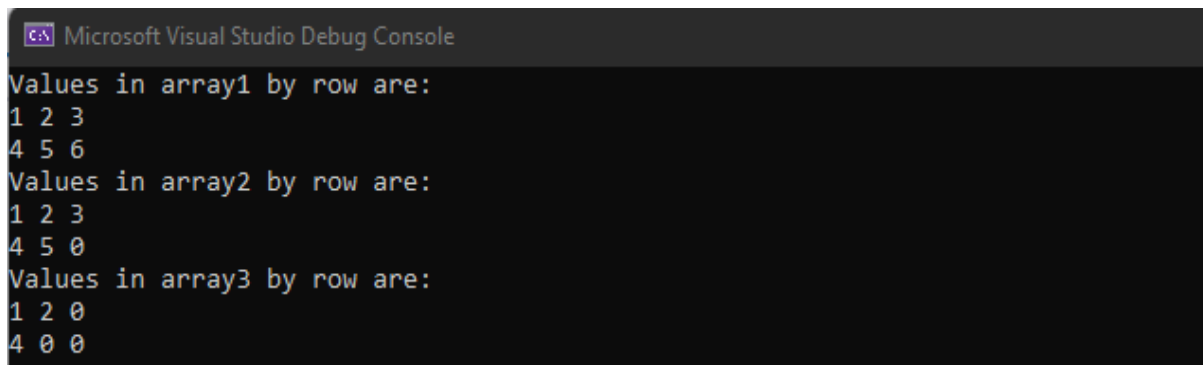
```
int b [2][2] = {{1}, {3, 4}};
```

would initialize **b [0][0]** to 1, **b [0][1]** to 0, **b [1][0]** to 3 and **b [1][1]** to 4.

Example Code:

```
1  #include <stdio.h>
2
3  void printArray(int a[][3]); // function prototype
4
5  int main(void) {
6      int array1[2][3] = { {1, 2, 3}, {4, 5, 6} };
7      puts("Values in array1 by row are:");
8      printArray(array1);
9
10     int array2[2][3] = { {1, 2, 3}, {4, 5} };
11     puts("Values in array2 by row are:");
12     printArray(array2);
13
14     int array3[2][3] = { {1, 2}, {4} };
15     puts("Values in array3 by row are:");
16     printArray(array3);
17 }
18
19 void printArray(int a[][3]) {
20     for (size_t i = 0; i <= 1; ++i) {
21         // output column values
22         for (size_t j = 0; j <= 2; ++j) {
23             printf("%d ", a[i][j]);
24         }
25         printf("\n"); // start new line of output
26     }
27 }
```

Output:

A screenshot of the Microsoft Visual Studio Debug Console window. The window has a title bar with the Visual Studio icon and the text "Microsoft Visual Studio Debug Console". The console output shows the values of three arrays, array1, array2, and array3, printed row by row. The text is as follows:

```
Values in array1 by row are:
1 2 3
4 5 6
Values in array2 by row are:
1 2 3
4 5 0
Values in array3 by row are:
1 2 0
4 0 0
```

Explanation:

array1 Definition: The program defines three arrays of two rows and three columns. The definition of array1 (line 6) provides six initializers in two sublists. The first sublist initializes row 0 to the values 1, 2 and 3, and the second sublist initializes row 1 to the values 4, 5 and 6.

array2 Definition: The definition of array2 (line 10) provides five initializers in two sublists, initializing row 0 to 1, 2 and 3, and row 1 to 4, 5 and 0. Any elements that do not have an explicit initializer are initialized to zero automatically, so **array2[1][2]** is initialized to 0.

array3 Definition: The definition of array3 (line 14) provides three initializers in two sublists. The first row's sublist explicitly initializes the row's first two elements to 1 and 2 and implicitly initializes the third element to 0. The second row's sublist explicitly initializes the first element to 4 and implicitly initializes the last two elements to 0.

Variable Array Length:

To define an array you've to specify its size at compilation time. But what if you cannot determine an array's size until execution time? For cases in which an array's size is not known at compilation time rather it would be defined on the execution time, C programming language provides variable length arrays (**VLAs**). Lengths of these arrays are determined by expressions evaluated at execution time. Please note that the **sizeof** operator when used in variable length array, it operates at run time instead of at compile time.

Example Code:


```

1  #include <stdio.h>
2
3  void oneDArray( int length, int a[ length ] ); //function prototype
4  void twoDArray( int row, int col, int a[ row ][ col ] ); //function prototype
5
6  int main()
7  {
8      int i, j; //counter variable
9      size_t size; //variable to hold size of one dimensional array
10     size_t row1, col1, row2, col2; //number of rows & columns of two D array
11     printf("Enter size of one-dimensional array: ");
12     scanf("%d", &size);
13
14     printf("Enter number of rows & columns of 2-D array:\n");
15     scanf("%d %d", &row1, &col1);
16
17     printf("Enter number of rows & columns of multi-D array:\n");
18     scanf("%d %d", &row2, &col2);
19
20     //declaring arrays
21     int arr[ size ];
22     int arr2D[ row1 ][ col1 ]; //2-D array
23     int arrMD[ row2 ][ col2 ]; //multi-dimensional array
24
25     //one dimensional array
26     for ( i = 0; i < size; ++i)
27     {
28         arr[ i ] = 2 * i;
29     }
30     //two dimensional array
31     for ( i = 0; i < row1; ++i)
32     {
33         for ( j = 0; j < col1; ++j)
34         {
35             arr2D[ i ][ j ] = i + j;
36         }
37     }
38     //multi-dimensional array
39     for ( i = 0; i < row2; ++i)
40     {
41         for ( j = 0; j < col2; ++j)
42         {
43             arrMD[ i ][ j ] = i + j;
44         }
45     }
46     //printing arrays
47     printf("One-dimensional array:\n");
48     oneDArray( size, arr );
49
50     printf("Two-dimensional array:\n");
51     twoDArray( row1, col1, arr2D );
52
53     printf("\nMulti-dimensional array:\n");
54     twoDArray( row2, col2, arrMD );
55     return 0;
56 } //end main

```

```

58 void oneDArray( int length, int a[ length ] )
59 {
60     int i;
61     for ( i = 0; i < length; ++i)
62     {
63         printf("%d ", a[ i ]);
64     } //end of oneDArray
65 void twoDArray( int row, int col, int a[ row ][ col ] )
66 {
67     int i, j;
68     for ( i = 0; i < row; ++i )
69     {
70         printf("\n");
71         for ( j = 0; j < col; ++j )
72         {
73             printf("%5d", a[i][j]);
74         } //end twoDArray

```

Output:

```
Enter size of one-dimensional array: 3
Enter number of rows & columns of 2-D array:
4
4
Enter number of rows & columns of multi-D array:
3
3
One-dimensional array:
0 2 4
Two-dimensional array:
  0  1  2  3
  1  2  3  4
  2  3  4  5
  3  4  5  6
Multi-dimensional array:
  0  1  2
  1  2  3
  2  3  4
```

Explanation:

In the above program, we have declared and printed three variable length arrays. First, we asked the user to enter the size for a one-dimensional array, two-dimensional and, multi-dimensional array.

There are two user-defined functions `oneDArray ()` and `twoDArray ()` for printing arrays. Function `oneDArray ()` takes size as parameter whereas, `twoDArray` takes row and col as its parameter. First, all the elements in the first row of the multi-dimensional array are filled, then proceed for the second row and so on.

Lab Tasks

Task 01: Unique Elements

Write a C program which:

- Defines an array of size “5”
- Set Values of array by taking input from user
- Displays the unique elements on the Console
- If no element is unique then displays the message “No element is unique”

Input	Output
2 3 4 5 6	Unique Elements: 2, 3, 4, 5, 6
89 89 55 34 55	Unique Element: 34
33 33 33 33 33	No element is unique

Note: You may find it helpful to make this function

bool isUnique(int arr[], int size, int element)

Task 02: Reverse an Array

Write a C program which:

- Defines an Array of size “5”
- Initialize the array by taking input from user
- Reverse the elements of array i.e., element at first index should be on the last index and vice versa

Input	Output (After Reversing Array)
2 3 4 5 6	6 5 4 3 2
9 13 12 5 99	99 5 12 13 9

Task 03: Fill the blank area

The following program:

- Defines an array
- Print the elements by calling a function

```
1  #include<stdio.h>
2  int main()
3  {
4      int Array[10] = { 1,2,3,4,5,6,7,8,9,10 };
5      //Code here
6      return 0;
7  }
8  void printArray(/*Parameters*/)
9  {
10     //code here
11 }
```

Task 04: Dry Run Programs.

(a) Dry Run the following programs and fill up the trace table:

```
1  #include<stdio.h>
2  void staticArray();
3  void automaticArray();
4  int main()
5  {
6      automaticArray();
7      staticArray();
8      staticArray();
9      automaticArray();
10     staticArray();
11 }
12 void staticArray()
13 {
14     static int Array1[2] = { 9 };
15     Array1[1] += 6;
16     for (int i = 0; i < 2; i++)
17         printf("Element at index %d: %d", i, Array1[i]);
18 }
19 void automaticArray()
20 {
21     int Array2[2] = {1,2};
22     Array2[1] += 5;
23     for (int i = 0; i < 2; i++)
24         printf("Element at index %d: %d", i, Array2[i]);
25 }
```

Line No	Output on Console
6	1,7
7	9 6
8	9 12
9	1,7
10	9 18

Task 05: Average Marks

Part (a)

Create a C program that: stores marks of 3 students for 5 tests in a two-dimensional array such that each row in the array represents test scores for one student. Display the average of each student's test scores on console.

Sample Output:

```
Enter marks of Student 1:-
    Enter marks of Test : 1: 88
    Enter marks of Test : 2: 97
    Enter marks of Test : 3: 79
    Enter marks of Test : 4: 86
    Enter marks of Test : 5: 94
Enter marks of Student 2:-
    Enter marks of Test : 1: 86
    Enter marks of Test : 2: 91
    Enter marks of Test : 3: 78
    Enter marks of Test : 4: 79
    Enter marks of Test : 5: 84
Enter marks of Student 3:-
    Enter marks of Test : 1: 82
    Enter marks of Test : 2: 73
    Enter marks of Test : 3: 77
    Enter marks of Test : 4: 82
    Enter marks of Test : 5: 89
Score average for student 1 is 88.8
Score average for student 2 is 83.6
Score average for student 3 is 80.6
```

Part (b):

Modify the above program such that it calculates and display average marks of each test.

Sample Output:

```
Enter marks of Student 1:-  
    Enter marks of Test : 1: 88  
    Enter marks of Test : 2: 72  
    Enter marks of Test : 3: 65  
    Enter marks of Test : 4: 54  
    Enter marks of Test : 5: 93
```

```
Enter marks of Student 2:-  
    Enter marks of Test : 1: 52  
    Enter marks of Test : 2: 63  
    Enter marks of Test : 3: 87  
    Enter marks of Test : 4: 96  
    Enter marks of Test : 5: 77
```

```
Enter marks of Student 3:-  
    Enter marks of Test : 1: 71  
    Enter marks of Test : 2: 52  
    Enter marks of Test : 3: 54  
    Enter marks of Test : 4: 89  
    Enter marks of Test : 5: 77
```

```
Class average for test 1 is 70.3333  
Class average for test 2 is 62.3333  
Class average for test 3 is 68.6667  
Class average for test 4 is 79.6667  
Class average for test 5 is 82.3333
```