

PF LAB # 07

Recursion & Arrays



October 9, 2019

Nida Munawar

Romasha khursheed

# Learning Objectives

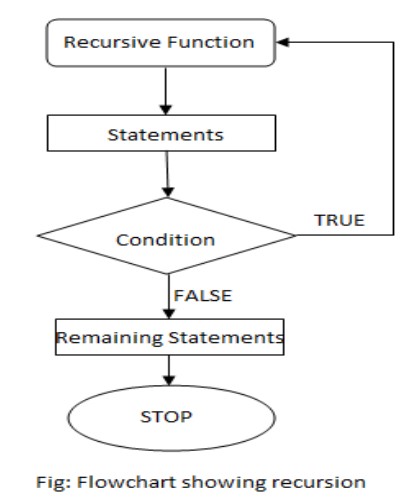
This lab will cover the following topics:

* Recursion
* Arrays
* 1D Array
* 2D Array

# Recursion

When a function invokes itself, the call is known as a recursive call. Recursion (the ability of a function to call itself) is an alternative control structure to repetition (looping). Rather than use a looping statement to execute a program segment, the program uses a selection statement to determine whether to repeat the code by calling the function again or to stop the process.

## Flowchart for recursion:

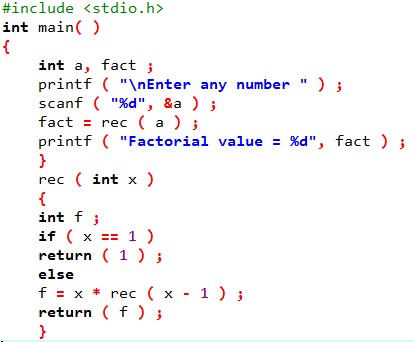


Each recursive solution has at least two cases: the base case and the general case.

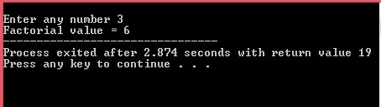
The **base case** is the one to which we have an answer; the **general case** expresses the solution in terms of a call to itself with a smaller version of the problem. Because the general case solves a smaller and smaller version of the original problem, eventually the program reaches the base case, where an answer is known, and the recursion stops.

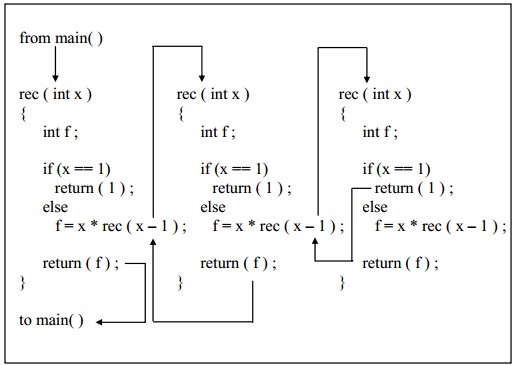
For example, a classic recursive problem is the factorial. The factorial of a number is defined as the number times the product of all the numbers between itself and 0: N! = N \* (N-1)! The factorial of 0 is 1. We have a base case, Factorial (0) is 1, and we have a general case, Factorial (N) is N \* Factorial (N-1). An if statement can evaluate N to see if it is 0 (the base case) or greater than 0 (the general case). Because N is clearly getting smaller with each call, the base case is reached.

Following is the recursive version of the function to calculate the factorial value.



Assume that the number entered through scanf( ) is 3. The figure below explains what exactly happens when the recursive function rec( ) gets called.





## Disadvantages of recursion

Recursive programs are generally slower than non-recursive programs. This is because, recursive function needs to store the previous function call addresses for the correct program jump to take place.

Requires more memory to hold intermediate states. It is because, recursive program requires the allocation of a new stack frame and each state needs to be placed into the stack frame, unlike non-recursive (iterative) programs.

# Arrays:

Array is a very popular data type. C language provides a capability that enables the user to design a set of similar data types, called array. C Array is a collection of variables belongings to the same data type(Homogeneous). You can store group of data of same data type in an array.

## What are Arrays

Array is a linear data structure that hold finite sequential collection of homogeneous data. We can store a collection of values in an array.

Array uses an integer value **index** to access a specific element. Index starts from 0 and goes till N-1 (where N is the size of array).

let us consider the following program:

main( ) {

int x ;

x = 5 ;

x = 10 ;

printf ( "\nx = %d", x ) ;

}

No doubt, this program will print the value of x as 10. Why so? Because when a value 10 is assigned to x, the earlier value of x, i.e. 5, is lost. Thus, ordinary variables (the ones which we have used so far) are capable of holding only one value at a time (as in the above example). However, there are situations in which we would want to store more than one value at a time in a single variable. A simple reason for this is, it would be much easier to handle one variable than handling 100 different variables.

## TYPES OF C ARRAYS:

**There are 2 types of C arrays. They are,**

1. One dimensional array
2. Multi-dimensional array
   * Two-dimensional array
   * Three-dimensional array
   * four-dimensional array etc.…

# What Is One Dimensional Array

An array which has only one subscript is known as **one dimensional array** i.e) int marks[10]. The bracket ( [ ] ) tells the compiler that we are dealing with an array.

## Array Declaration and Array Initialization

:

Syntax

data\_type varname[size];

declaration-1

int student[5] = {89, 76, 98, 91, 84};

declaration-2

int student[] = {89, 76, 98, 91, 84};

Note the following points carefully:

(a) Till the array elements are not given any specific values, they are supposed to contain garbage values.

(b) If the array is initialized where it is declared, mentioning the dimension of the array is optional

## Rules for Declaring One Dimensional Array

* An array variable must be declared before being used in a program.
* The declaration must have a data type(int, float, char, double, etc.), variable name, and subscript.
* The subscript represents the size of the array. If the size is declared as 10, programmers can store 10 elements.
* An array index always starts from 0. For example, if an array variable is declared as s[10], then it ranges from 0 to 9.
* Each array element stored in a separate memory location.

## Accessing Elements of an Array

This is done with subscript, the number in the brackets following the array name. This number specifies the element’s position in the array. All the array elements are numbered, starting with 0. Thus, marks[2] is not the second element of the array, but the third.

* n array is a collection of similar elements.
* The first element in the array is numbered 0, so the last element is 1 less than the size of the array.
* An array is also known as a subscripted variable.
* Before using an array its type and dimension must be declared.
* However big an array its elements are always stored in contiguous memory locations.

## A Simple Program Using Array

Let us try to write a program to find average marks obtained by a class of 5 students in a test.

main( )

{

int avg, sum = 0 ;

int i ;

int marks[5] ; /\* array declaration \*/

for ( i = 0 ; i <= 4 ; i++ ) {

printf ( "\nEnter marks " ) ;

scanf ( "%d", &marks[i] ) ; /\* store data in array \*/

}

for ( i = 0 ; i <= 4 ; i++ )

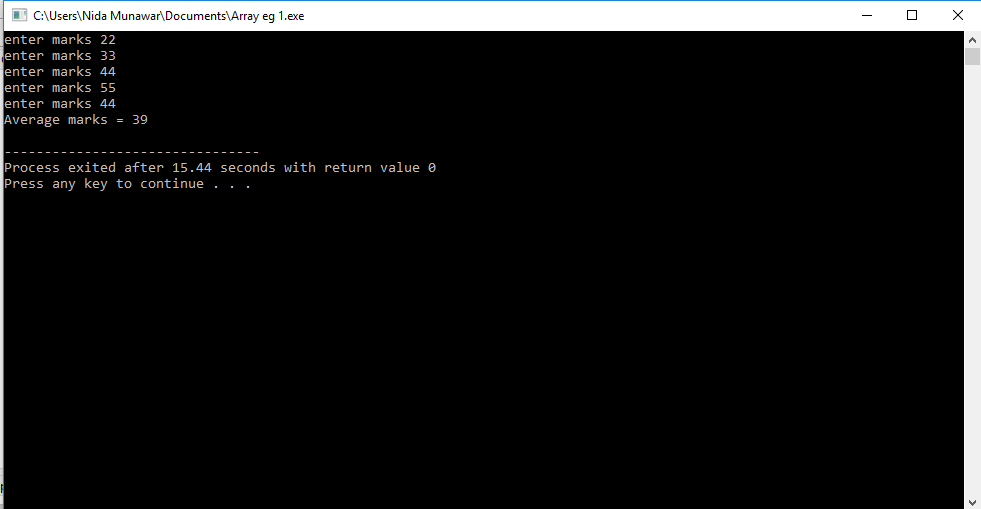
sum = sum + marks[i] ; /\* read data from an array\*/

avg = sum / 5 ;

printf ( "\nAverage marks = %d", avg ) ;

}

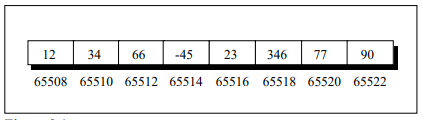
Output



## Array Elements in Memory

int arr[8] ;

What happens in memory when we make this declaration? 16 bytes get immediately reserved in memory, 2 bytes each for the 8 integers. And since the array is not being initialized, all eight values present in it would be garbage values. Whatever be the initial values, all the array elements would always be present in contiguous memory locations.



Here is a program that prints out the memory locations in which the elements of this array are stored.

main( ) {

int num[ ] = { 1,2,3,4,5 } ;

int i ;

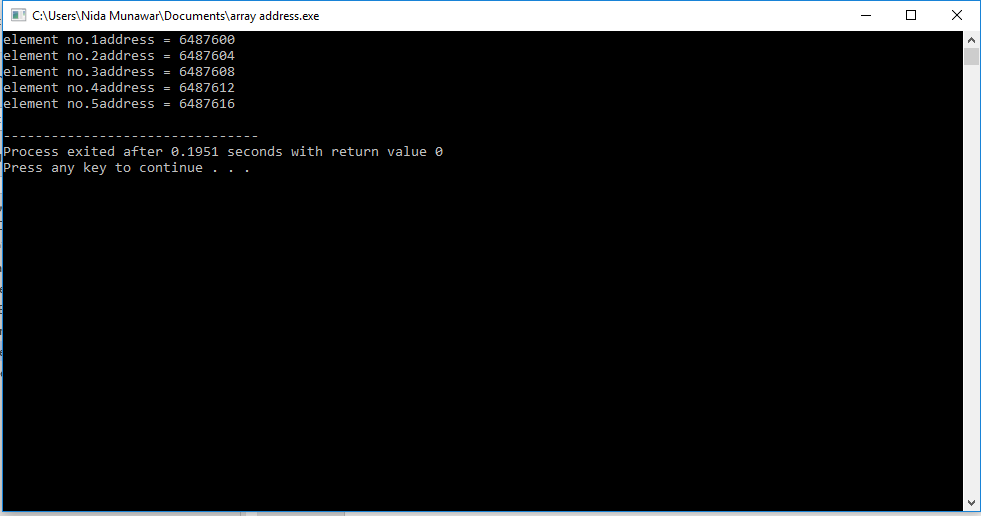
for ( i = 0 ; i <= 5 ; i++ ) {

printf ( "\naddress = %u ", &num[i] ) ;

printf ( "element = %d", num[i] ) ; }

}

**Output**



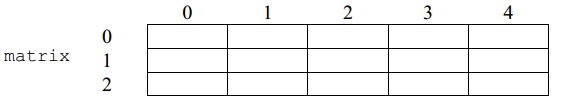
## Bounds Checking

In C there is no check to see if the subscript used for an array exceeds the size of the array. Data entered with a subscript exceeding the array size will simply be placed in memory outside the array; probably on top of other data, or on the program itself. This will lead to unpredictable results, to say the least, and there will be no error message to warn you that you are going beyond the array size.

# Two Dimensional Arrays:

In many programming applications you naturally organize data into rows and columns. In C you can use a two-dimensional array to store data in this form. The two dimensional arrays can be describe as “arrays of arrays”. All of them of a same uniform data type.

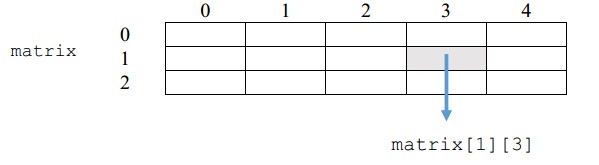
For example:



matrix represents a two-dimensional array of 3 per 5 elements of type int. The C syntax for this is:

int matrix [3][5];

and, for example, the way to reference the second element vertically and fourth horizontally in an expression would be:



## Initializing a 2-Dimensional Array

int stud[4][2] = {

{1234, 56},

{1212, 33},

{1434, 80},

{1312, 78}

} ;

or even this would work...

int stud[4][2] = { 1234, 56, 1212, 33, 1434, 80, 1312, 78 } ;

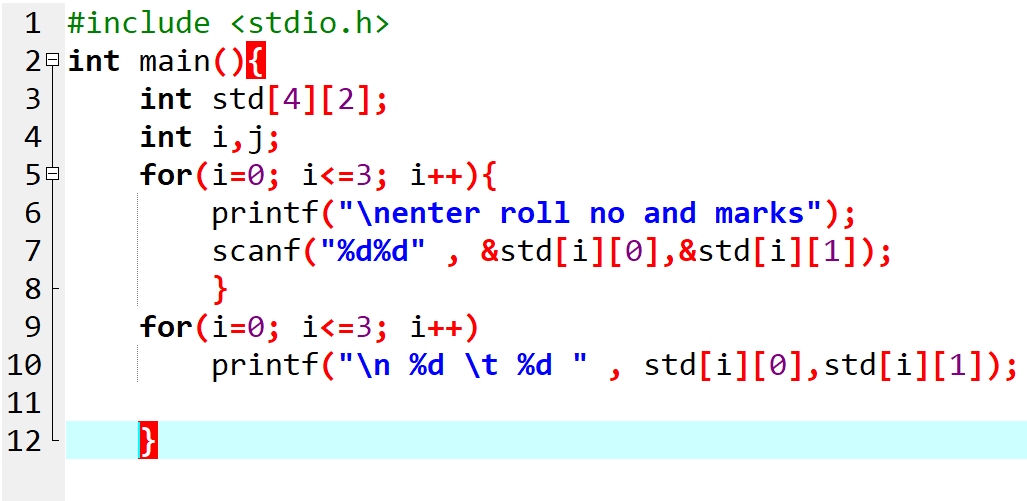
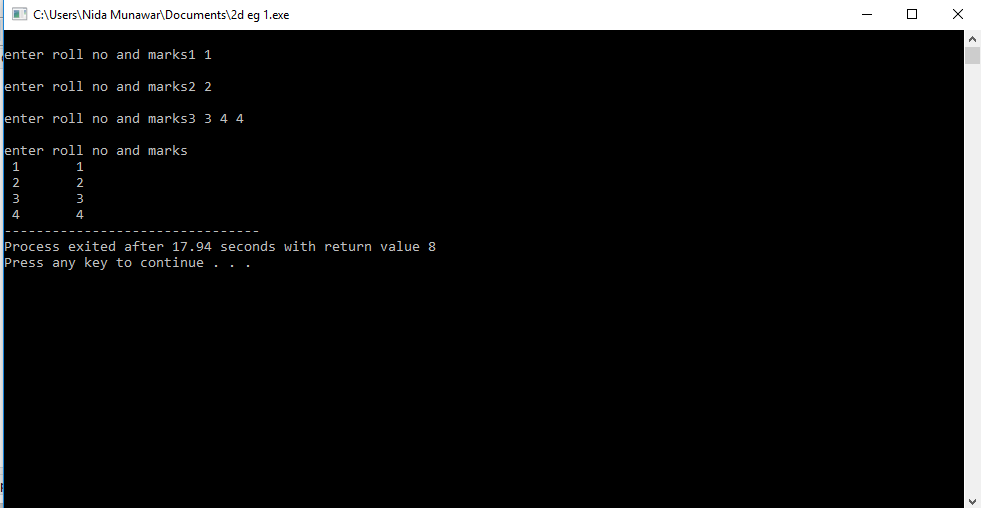
of course, with a corresponding loss in readability.

It is important to remember that while initializing a 2-D array it is necessary to mention the second (column) dimension, whereas the first dimension (row) is optional. Thus, the declarations,

int arr[2][3] = { 12, 34, 23, 45, 56, 45 } ;

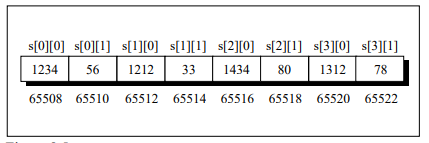
int arr[ ][3] = { 12, 34, 23, 45, 56, 45 } ;

are perfectly acceptable

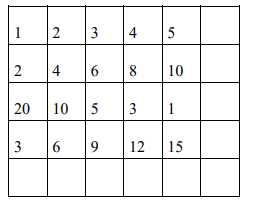


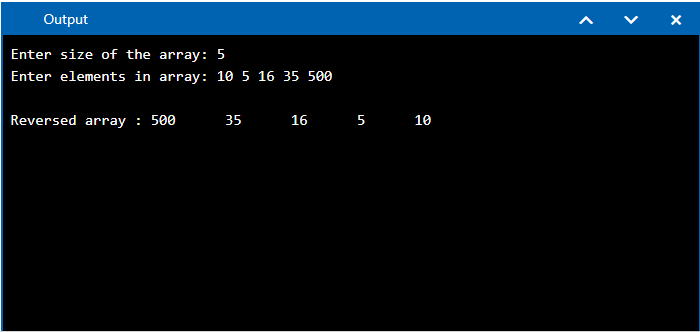
## Memory Map of a 2-Dimensional Array

## 



# LAB Activity

1. Write a C program to find maximum and minimum element in array
2. Write a C program to count even and odd elements in array
3. Write a C program to find second largest number in array
4. Write a C program to find maximum and minimum element in array using Recursion
5. Ali, a student of 2nd year studying in university is suffering from fever since last week. Doctor wants to diagnose his temperature values for whole week. Write a program in C that inputs the temperature values of entire week and displays the day on which he had maximum and minimum temperature.
6. The partially initialized array "table" can be viewed as a primitive spreadsheet, in which the last column and bottom row have been left blank. Write the code to fill in this row and column with the totals of each column, each row, and the grand total.
   1. 
7. Write a C program to find reverse of array



1. Searching has different techniques when it comes to programming. Finding any certain value from a given range can be done by the help of different algorithms. Generate a random list consisting of 10 numbers and search for a value taken from user. Stop only when user finds specified value. Also mention at which position it was found.