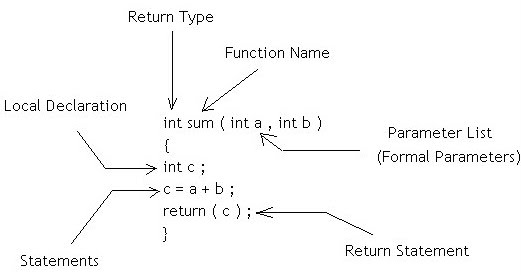
**Function Explanation**



### Example

The figure above is a simple example of a function with all its major components pointed out.

**Components of a function**

**Return-type**: This specifies the data type of the value being returned by the function. A function may or may not return a value. If the function does not return a value then the return type is void. In this case, the return value is an integer value c, which means the return type would be *int*.

**Parameter list**: The parameter list is the list of formal parameters being passed onto the function. In this case, there are two parameters of type *int* passed to the function.

**Local variables or local declarations**: The variables that are declared inside the function are called local variables. The scope of these variables lies within the function and they are not accessible outside the function.

**Function body**: A function body comprises of everything inside the curly brackets { and } following the return type, function name and the parameter list.

**Function Name**: A function name can be anything that you want. The standard is to make it something related to what it is supposed to do. The naming convention follows the same rule as that of variable naming convention in C.

**Function declaration** : A function declaration simply tells the compiler all about the function. These include the function's name, the return type and the number and types of parameters. The body of the function having the function definition can be defined somewhere else. A function declaration has the following parts:

return\_type function\_name( parameter list );

The example above can be declared as follows:

1. int sum(int num1, int num2);

or

1. int sum(int, int);

A prototype declares the function name, its parameters, and its return type to the rest of the program prior to the function's actual declaration. To understand why function prototypes are useful, enter the following code and run it:

#include <stdio.h>

void main()

{

printf("%d\n",add(3));

}

int add(int i, int j)

{

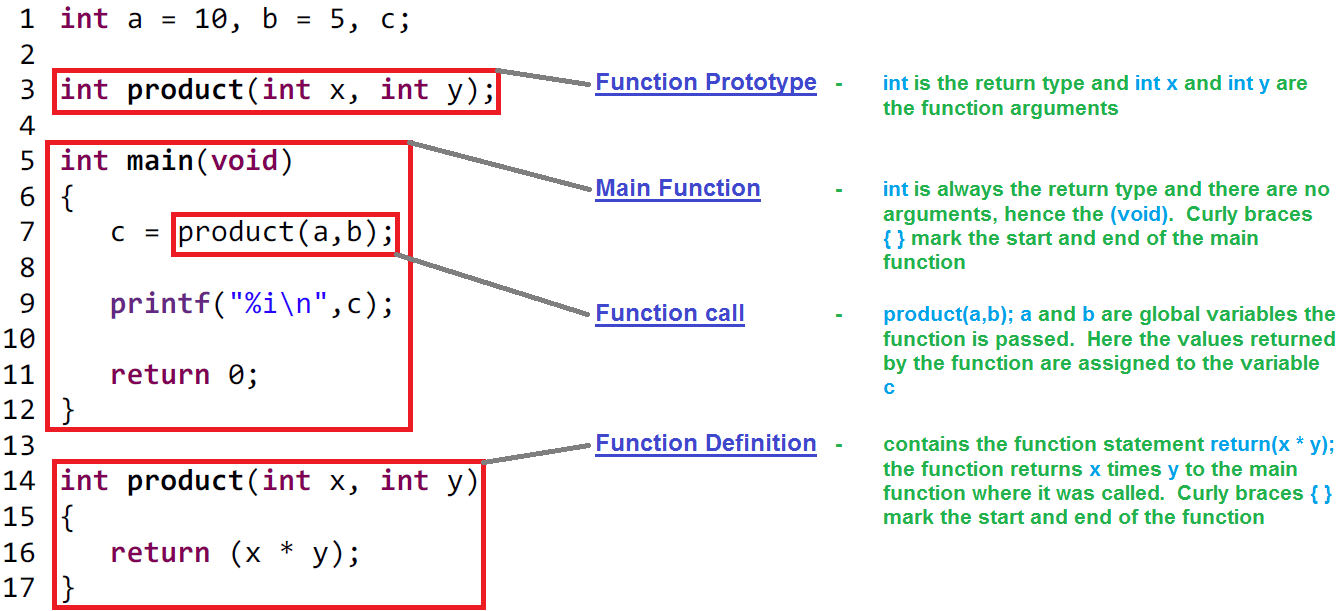
return i+j;

}

This code compiles on many compilers without giving you a warning, even though **add** expects two parameters but receives only one. It works because many C compilers do not check for parameter matching either in type or count. You can waste an enormous amount of time debugging code in which you are simply passing one too many or too few parameters by mistake. The above code compiles properly, but it produces the wrong answer.

Function prototype definition:

A function prototype is a function declaration that specifies the data types of its arguments in the parameter list. The compiler uses the information in a function prototype to ensure that the corresponding function definition and all corresponding function declarations and calls within the scope of the prototype contain the correct number of arguments or parameters, and that each argument or parameter is of the correct data type.



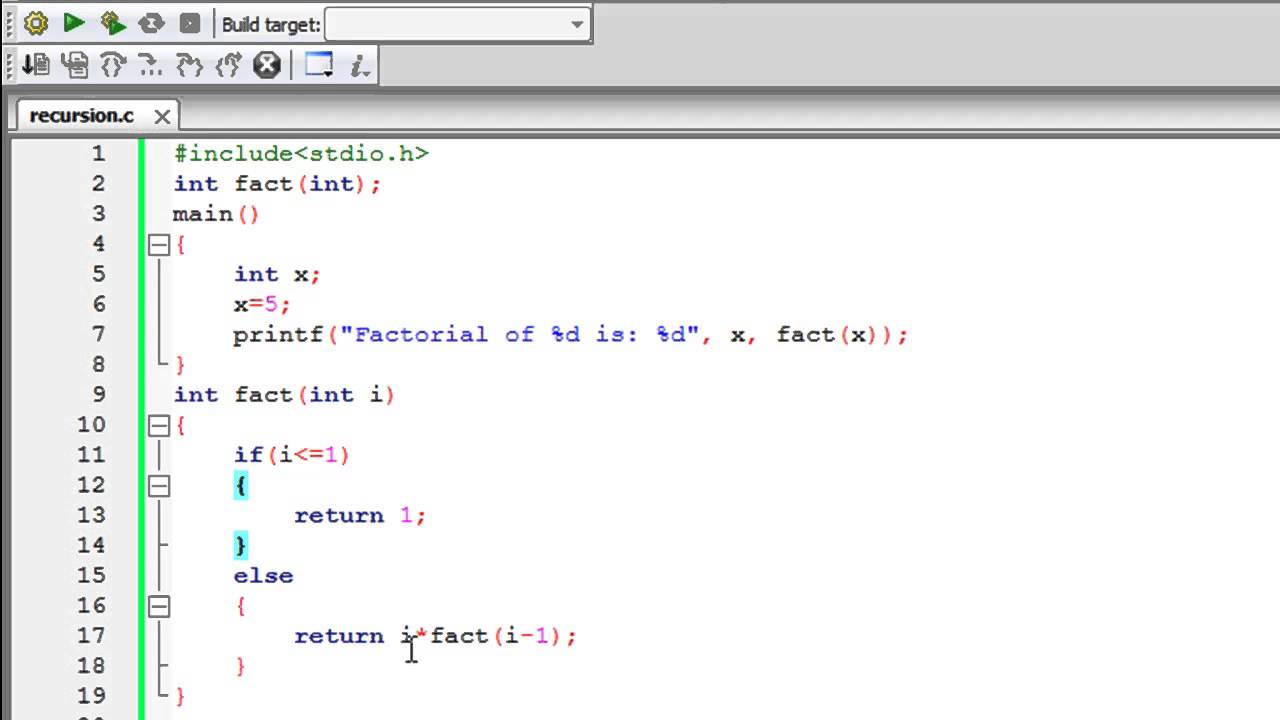
RECURSION IN C

The process of calling a function by itself is called recursion and the function which calls itself is called recursive function.

* Recursion is used to solve various mathematical problems by dividing it into smaller problems.
* This method of solving a problem is called Divide and Conquer.
* In programming, it is used to divide complex problem into simpler ones and solving them individually.

SYNTAX OF RECURSIVE FUNCTION

returntype recursive\_func ([argument list])   
{   
 statements;   
 ... ... ...   
 recursive\_func ([actual argument]);   
 ... ... ...   
}



## **How to declare arrays?**

data\_type array\_name[array\_size];

**For example,**

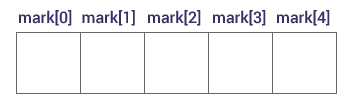
float mark[5];

Here, we declared an array, mark, of floating-point type and size 5. Meaning, it can hold 5 floating-point values.

## Elements of an Array and How to access them?

You can access elements of an array by indices.

Suppose you declared an array mark as above. The first element is mark[0], second element is mark[1] and so on.



#### Few key notes:

* Arrays have 0 as the first index not 1. In this example, mark[0]
* If the size of an array is n, to access the last element, (n-1) index is used. In this example, mark[4]
* Suppose the starting address of mark[0] is 2120d. Then, the next address, a[1], will be 2124d, address of a[2] will be 2128d and so on. It's because the size of a float is 4 bytes.

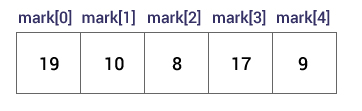
### How to initialize an array?

It's possible to initialize an array during declaration. For example,

int mark[5] = {19, 10, 8, 17, 9};

Another method to initialize array during declaration:

int mark[] = {19, 10, 8, 17, 9};



Here,

mark[0] is equal to 19

mark[1] is equal to 10

mark[2] is equal to 8

mark[3] is equal to 17

mark[4] is equal to 9

### How to insert and print array elements?

int mark[5] = {19, 10, 8, 17, 9}

// insert different value to third element

mark[3] = 9;

// take input from the user and insert in third element

​scanf("%d", &mark[2]);

// take input from the user and insert in (i+1)th element

scanf("%d", &mark[i]);

// print first element of an array

printf("%d", mark[0]);

// print ith element of an array

printf("%d", mark[i-1]);

## Example: C Arrays

// Program to find the average of n (n < 10) numbers using arrays

#include <stdio.h>

int main()

{

int marks[10], i, n, sum = 0, average;

printf("Enter n: ");

scanf("%d", &n);

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

{

printf("Enter number%d: ",i+1);

scanf("%d", &marks[i]);

sum += marks[i];

}

average = sum/n;

printf("Average = %d", average);

return 0;

}

**Output**

Enter n: 5

Enter number1: 45

Enter number2: 35

Enter number3: 38

Enter number4: 31

Enter number5: 49

Average = 39

## **Two-dimensional Arrays**

The simplest form of multidimensional array is the two-dimensional array. A two-dimensional array is, in essence, a list of one-dimensional arrays. To declare a two-dimensional integer array of size [x][y], you would write something as follows −

type arrayName [ x ][ y ];

Where **type** can be any valid C data type and **arrayName** will be a valid C identifier. A two-dimensional array can be considered as a table which will have x number of rows and y number of columns. A two-dimensional array **a**, which contains three rows and four columns can be shown as follows −



Thus, every element in the array **a** is identified by an element name of the form **a[ i ][ j ]**, where 'a' is the name of the array, and 'i' and 'j' are the subscripts that uniquely identify each element in 'a'.

## **Initializing Two-Dimensional Arrays**

Multidimensional arrays may be initialized by specifying bracketed values for each row. Following is an array with 3 rows and each row has 4 columns.

int a[3][4] = {

{0, 1, 2, 3} , /\* initializers for row indexed by 0 \*/

{4, 5, 6, 7} , /\* initializers for row indexed by 1 \*/

{8, 9, 10, 11} /\* initializers for row indexed by 2 \*/

};

The nested braces, which indicate the intended row, are optional. The following initialization is equivalent to the previous example −

int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};

## **Accessing Two-Dimensional 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. For example −

int val = a[2][3];

The above statement will take the 4th element from the 3rd row of the array. You can verify it in the above figure. Let us check the following program where we have used a nested loop to handle a two-dimensional array −

#include <stdio.h>

int main () {

/\* an array with 5 rows and 2 columns\*/

int a[5][2] = { {0,0}, {1,2}, {2,4}, {3,6},{4,8}};

int i, j;

/\* output each array element's value \*/

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

for ( j = 0; j < 2; j++ ) {

printf("a[%d][%d] = %d\n", i,j, a[i][j] );

}

}

return 0;

}

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

a[0][0]: 0

a[0][1]: 0

a[1][0]: 1

a[1][1]: 2

a[2][0]: 2

a[2][1]: 4

a[3][0]: 3

a[3][1]: 6

a[4][0]: 4

a[4][1]: 8

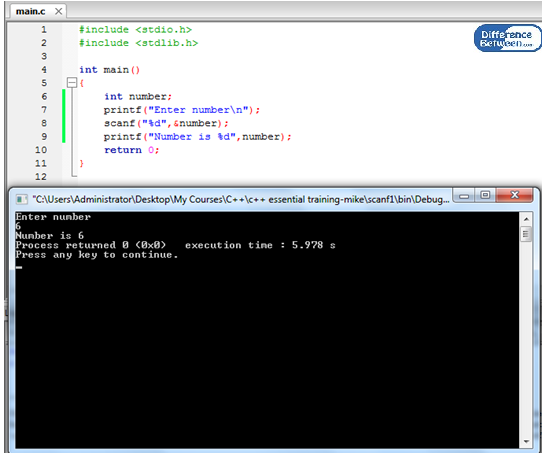
As explained above, you can have arrays with any number of dimensions, although it is likely that most of the arrays you create will be of one or two dimensions.

## **What is scanf?**

The scanf function can read input from keyboard and stores them according to the given format specifier. It reads the input till encountering a whitespace, newline or EOF. The syntax is as follows.

            scanf(“format string”, list of address of variables);

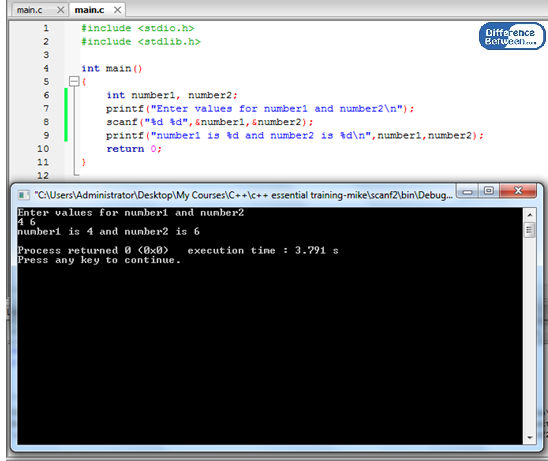
Refer the example given below to understand scanf.



**Figure 01:  scanf with one input**

According to the above program, the input getting from the keyboard is an integer, so the format specifier is %d. If it is getting a character value, the format specifier is %c. If getting a floating-point value, the format specifier is %f. The received input value should be stored in the number variable. Therefore, the address of variable number is passed into the scanf function. Now the number variable contains the value given by the user from the keyboard. Finally, we can print the number variable to check the value.

It is also possible to receive more than one value at a time.



**Figure 02: scanf with multiple inputs**

The received inputs are stored in variable number1 and number2. Values can be checked by using [printf](https://www.differencebetween.com/difference-between-printf-and-vs-fprintf/#printf).

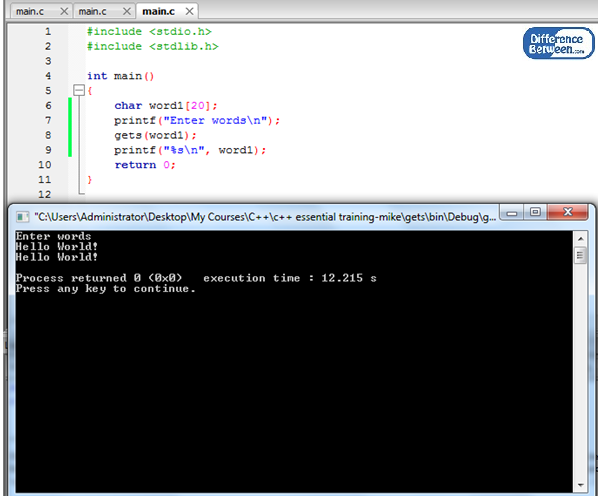
## What is gets?

gets function is using to receive input from the keyboard till encountering a newline or EOF. The whitespace is considered as a part of the input. The syntax for gets function is as follows.

            gets(“where to store the string”);

If there is an error when receiving the string, the gets function will return a null value.

Refer the below example,



**Figure 03: gets**

The input is received by the gets function and stored in variable word1. If the programmer used scanf instead of gets and input a string such as “hello world”, scanf will read the string as two strings because of the whitespace. But gets will read it as one string “hello world”.

## What are the Similarities Between scanf and gets?

* Both are functions provided by C programming language.
* Both should include header file stdio.h to use these functions.
* Both can be used to get input from the standard input.

## What is the Difference Between scanf and gets?

|  |  |
| --- | --- |
| scanf vs gets | |
| scanf is a C function to read input from the standard input until encountering a whitespace, newline or EOF. | gets is a C function to read input from standard input until encountering a newline or EOF. It considers whitespace as a part of the input. |
| **Syntax** | |
| scanf function takes the format string and list of addresses of variables. e.g. scanf(“%d”, &number); | gets function takes the name of the variable to store the received value. E.g. gets(name); |
| **Flexibility** | |
| scanf can read multiple values of different data types. | gets() will only get character string data. |

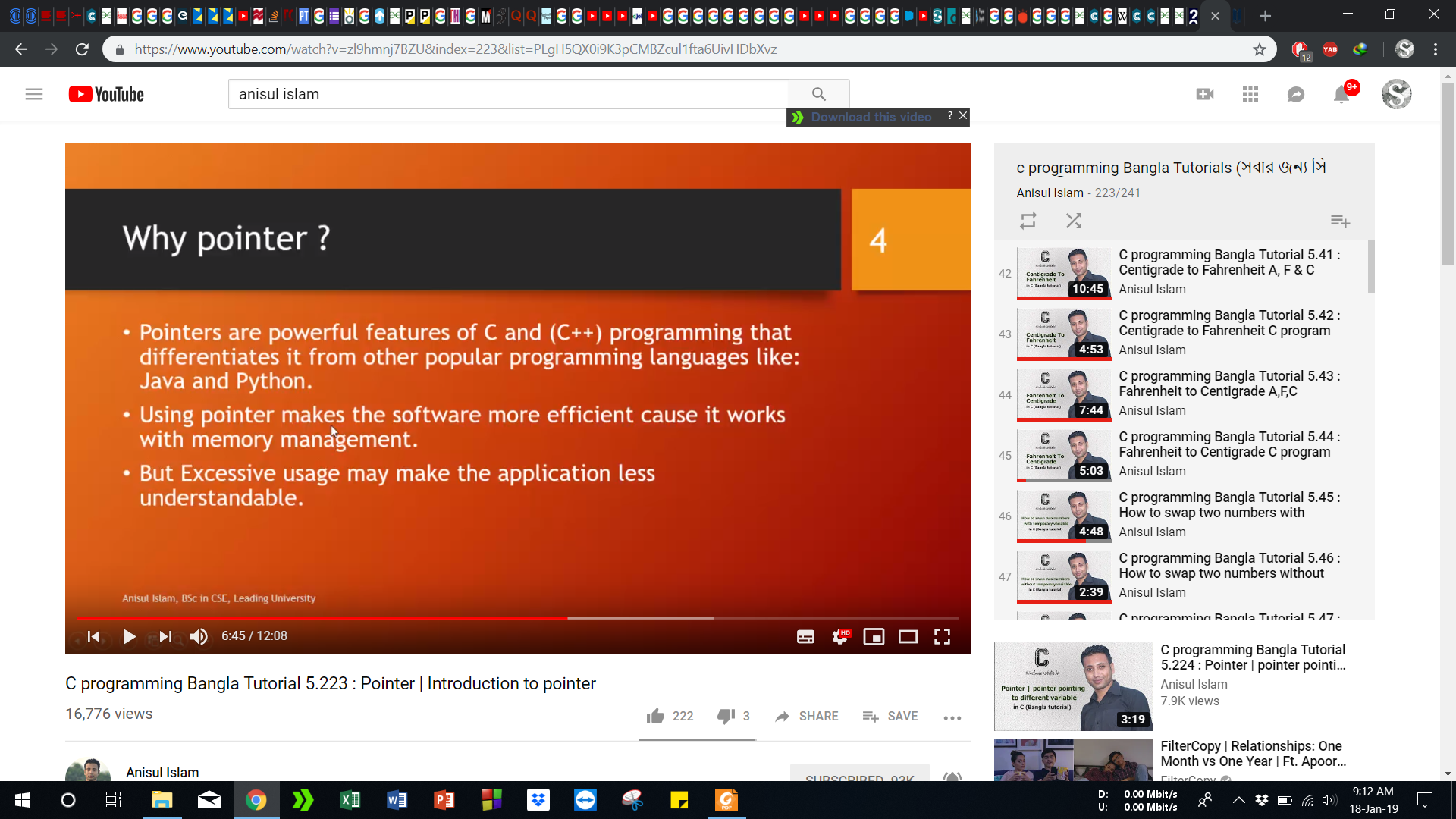
## Summary – **scanf vs gets**

scanf and gets are functions provided by the programming language C. User does not need to implement these functions from the beginning. They can directly use them in their programs. The difference between scanf and gets is that, scanf ends taking input upon encountering a whitespace, newline or End Of File (EOF) and gets considers a whitespace as the part of the input string and ends the input upon encountering newline or EOF. Using scanf or gets depends on the way to receive user input from the standard input which is the keyboard most of the time. scanf is more flexible than gets.

### Difference between gets() and scanf()

**Examples with scanf() and gets() functions  
  
#include<stdio.h>  
#include<stdio.h>  
int main()  
{  
    char str[20];  
    printf("enter the string:");  
    gets(str);  
    printf("%s",str);  
  
}  
  
Input:  
enter the string:hello world  
Output:hello world  
  
  
#include<stdio.h>  
#include<stdio.h>  
int main()  
{  
    char str[20];  
    printf("enter the string:");  
    scanf("%s",str);  
    printf("%s",str);  
  
}  
  
Input:  
enter the string:hello world  
Output:hello  
  
Reason is the compiler will add the null character of every end of the string. then that compiler will treat like a end of string so whenever we give the space between the input string the scanf() function will accept the before space characters only. so it dont allow the spaces  
The gets() function will allow the spaces**

**Pointers**



Sum

