Function Arguments:

If a function is to use arguments, it must declare variables that accept the values of the arguments. These variables are called the **formal parameters** of the function.

The formal parameters behave like other local variables inside the function and are created upon entry into the function and destroyed upon exit.

While calling a function, there are two ways that arguments can be passed to a function:

Call Type	Description
Call by value	This method copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument.
Call by reference	This method copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. This means that changes made to the parameter affect the argument.

By default, C uses **call by value** to pass arguments. In general, this means that code within a function cannot alter the arguments used to call the function and above mentioned example while calling max() function used the same method.

Structures in C

http://www.tutorialspoint.com/cprogramming/c_structures.htm

C arrays allow you to define type of variables that can hold several data items of the same kind but **structure** is another user defined data type available in C programming, which allows you to combine data items of different kinds.

Checkout the following links for indepth tutorials:

http://www.tutorialspoint.com/cprogramming/c_passing_arrays_to_functions.htm

http://beginnersbook.com/2014/01/c-passing-array-to-function-example/

http://beginnersbook.com/2014/01/c-arrays-example/

http://ftp.tuwien.ac.at/languages/c/programming-bbrown/c_047.htm

http://www.programiz.com/c-programming/c-arrays-functions

http://www.studytonight.com/c/array-in-function-in-c.php

http://crasseux.com/books/ctutorial/Arrays-as-Parameters.html

The **call by reference** method of passing arguments to a function copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. This means that changes made to the parameter affect the passed argument.

To pass the value by reference, argument pointers are passed to the functions just like any other value. So accordingly you need to declare the function parameters as pointer types as in the following function **swap()**, which exchanges the values of the two integer variables pointed to by its arguments.

```
/* function definition to swap the values */
void swap(int *x, int *y)
{
   int temp;
   temp = *x;    /* save the value at address x */
   *x = *y;      /* put y into x */
   *y = temp;    /* put temp into y */

   return;
}
```

For now, let us call the function **swap()** by passing values by reference as below:

#include <stdio.h>

```
void swap(int *x, int *y);            /* function declaration */

int main ()
{
   int a = 100;
   int b = 200;               /* local variable definition */

   printf("Before swap, value of a : %d\n", a );
   printf("Before swap, value of b : %d\n", b );

   /* calling a function to swap the values.
    * &a indicates pointer to a ie. address of variable a and
    * &b indicates pointer to b ie. address of variable b.
   */
   swap(&a, &b);

   printf("After swap, value of a : %d\n", a );
   printf("After swap, value of b : %d\n", b );

   return 0;
}
```

Let us put above code in a single C file, compile and execute it, it will produce the following result:

```
Before swap, value of a :100     Before swap, value of b :200

After swap, value of a :200      After swap, value of b :100
```