Function Calls :-

By now we are well familiar with how to use functions in C. But, if we observe carefully, whenever we called a function and passed something to it, we have always passed the ‘values’ of variables to the called function. Such function calls are called ‘call by value’. Similarly, we have also learned that variables are stored somewhere in memory. So instead of passing the value of a variable, can we pass the location an address of the variable to a function. Such function calls are called ‘**call by reference**’.

This call by reference functions needs the knowledge of a concept called ‘**pointers**’. It is the use of pointers that makes the C programming an excellent language.

What are the Pointers?

A pointer in C is a variable that allocates memory dynamically. It holds the value that is the address of another variable, i.e., direct address of the memory location. The syntax of a pointer variable declaration is

type *variable_name;

```
int *ptr;   /* This is the pointer to an integer */
```

Here, * is used to denote the pointer variable, and to return the address of the variable, we use operator **'&'**.

Let us now get back to function calls that are the main focus of this tutorial. The function calls are of two types. In function calls, one of the important concepts is the formal and actual parameters.

- Formal Parameter: Formal parameters are the local variable which are assigned values from the arguments when the function is called.
- Actual Parameter: When a function is called, the values(expression) that are passed in the call are called arguments or actual parameters.

There are two ways we generally pass the arguments to functions:

(a) Call by values

In this method, the ‘call by value’ of each of the actual arguments in the calling function is copied into corresponding formal arguments of the called function. In this function call, the changes made to the formal arguments in the called function have no effect on the values of actual arguments in the calling function. Actual and formal arguments will be created in a different memory location. The following program is the example of ‘Call by Value’.

```
void swap(int x, int y)
{
```

```
int temp;
temp=x;
x=y;
y=temp;
}
void main()
{
int r=10, v=20;
swap(r, v);  // passing value to function
printf("\nValue of r: %d",r);
printf("\nValue of v: %d",v);
}
```

(b) Call by reference

In this method, the addresses of actual arguments in the calling function are copied into formal arguments of the called function. This means that using these addresses we could access the actual arguments and hence we would be able to manipulate them. The changes that are made to the parameter affect the argument. This is because the address is used to access the actual argument. Formal and actual arguments will be created in the same memory location. The following program is the example of 'Call by Reference'.

```
void swap(int *x, int *y)
{
int temp;
 temp=*x;
*x=*y;
*y=temp;
}
void main()
{
int r=10, v=20;
swap(&r, &v);  // passing value to function
printf("\nValue of r: %d",r);
printf("\nValue of v: %d",v);
}
```

Usually, we use call by value function call. This means that in general, we cannot alter the actual arguments. But if desired, we can use call by reference for that purpose. We can make a function return more than one value at a time by using call by reference which is not possible ordinarily.

Conclusions

From the above discussion, we can draw the following conclusions:

- If we want the value of an actual argument to not get changed in the function being called, pass the actual argument by value.
- If we want the value of an actual argument should get changed in the function being called, then pass the actual argument by reference.
- If a function is to be made to return more than one value at a time then use call by reference method for that purpose.

```c
#include <stdio.h>

void changeValue(int* address)
{
    *address = 365;
}

int main()
{
    int a = 34, b =71;
    printf("The value of a now is %d\n", a);
    changeValue(&a);
    printf("The value of a now is %d\n", a);
    return 0;
}
```