Call by value VS Call by reference

In this tutorial, you will learn everything about the two different ways of passing arguments to function (call by value & call by reference) and how they are useful and different.

Function arguments

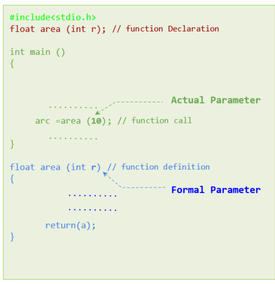
Function arguments in C language are the set of variables used when calling a function. Usually, the arguments take the input values and pass them to their corresponding parameters in the function definition. There are generally two ways through which arguments can be passed into the function:

- Call by value
- Call by reference

Before moving to the two methods let's acquaint with two other important terms used while passing arguments. They are :

- **Actual parameters** - the arguments used in the function call
- **Formal Parameters** - the parameters used in the function definition.

Formal parameters are used when we are defining a function. Whereas actual parameters come into play when the function is called for execution. So formal parameters set the rules and actual ones put the values in it to produce the result. To make it clear see the below visualization:



A function call in C

What is call by value in C

Call by value is an argument passing method used in a programming language to pass the actual parameters to formal parameters. As its name suggests, value of parameters is passed here.Specifically,the value of the actual parameter is copied to the formal parameter while passing. Hence any alteration or adjustments done on formal parameters in function definition has no effect on actual parameters in the function call. This is because two separate memory is created for actual parameters and formal parameters, clearly, we are addressing two different storage locations.

Consider the following program:

```
#include <stdio.h>
int tot( int a, int b)
{
     int c;
      c=a+b;
     printf("Before modification in function definition: a = %d and b = %d  \n",a,b);
     printf("Before modification in function definition : Total of a and b  - is :%d\n\n",c);

     a = a + 5;
     b = b + 5;
     c = a+b;
     printf("After modification in function definition: a = %d and b = %d  \n",a,b);
     printf("After modification in function definition : Total of a and b  - is :%d\n\n",c);

     return 0;
}

int main()
{
    int p =10;
    int q= 20;
    printf("Before function call in main():p = %d and q = %d  \n\n",p,q);
    int z = tot(p, q);
    printf("After function call in main():p = %d and q = %d  \n",p,q);
    return 0;
}
```

In this program,

1. **a** and **b** are formal parameters whose duty is to define the formula i. e. **a+b=c**.
2. On the other hand, **p** and **q** will be considered as actual parameters. Their job is to pass the value 10 and 20 to the formal parameters

The output will be like this:

```
Before function call in main():p = 10 and q = 20

Before modification in function definition: a = 10 and b = 20
Before modification in function definition : Total of a and b  - is :30

After modification in function definition: a = 15 and b = 25
After modification in function definition : Total of a and b  - is :40

After function call in main():p = 10 and q = 20
```

When you observe the output it is pretty clear that formal parameter values are mutable while actual parameters are not when call by value is used for passing arguments.

What is call by reference in C

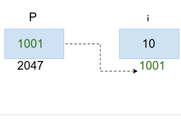
Call by reference is yet another parameter passing method used in C language to pass arguments to parameters in the function definition. In this procedure of function call, instead of value, the address of actual parameters is passed into formal parameters. This means both actual parameter and formal parameters use the same memory location and hence any alteration of the formal parameter will immediately reflect in actual parameters.

To understand this we must get acquainted with pointer variables which can store the address of another variable. Suppose

int i=10;

int *p;

p=&i;



We can see those pointer variables are associated with a '*'. Here p is a pointer variable which has stored the address where 'i' resides. Here are the swapping programs demonstrating the use of call by reference or address.

```
#include <stdio.h>

void swap_add(int * i, int * j);
void main(void) {
    int p = 5, q = 10;
    printf("\n In main() before swapping: p=%d, q=%d", p, q);
    swap_add( & p, & q);
    printf("\n In main () after swapping: p=%d q=%d", p, q);
}
void swap_add(int * i, int * j) {
    int temp;
    temp = * i;
    * i = * j;
    * j = temp;
    printf("\n Inside function definition after swapping:i=%d, j=%d", * i, * j);
}
```

In the above mentioned program *i and *j work with the values stored in the address of p and q respectively. So interchange of address here caused the swapping of the actual values. So output of the program will look like:

```
In main() before swapping: p=5, q=10
 Inside function definition after swapping:i=10, j=5
In main () after swapping: p=10 q=5
```

Difference between Call by value and call by reference.

SL NO	CALL BY VALUE	CALL BY REFERENCE
1	Importance is given to value	importance is given to address
2	Copy of value is passed to the function	Address is passed to the function
3	both formal and actual parameters use separate memory	both formal and actual parameters points to same memory
4	Changes made to value in function does not reflect in main function	Changes made to value in function does reflect in main function