Call by value and Call by reference to a Function

Functions can be invoked in two ways: **Call by Value** or **Call by Reference**. These two ways are generally differentiated by the type of values passed to them as parameters.
The parameters passed to function are called *actual parameters* whereas the parameters received by function are called *formal parameters*.

Call By Value: In this parameter passing method, values of actual parameters are copied to function's formal parameters and the two types of parameters are stored in different memory locations. So any changes made inside functions are not reflected in actual parameters of caller.

Call by reference

In call by reference, the address of the variable is passed into the function call as the actual parameter. The value of the actual parameters can be modified by changing the formal parameters since the address of the actual parameters is passed. In call by reference, the memory allocation is similar for both formal parameters and actual parameters. All the operations in the function are performed on the value stored at the address of the actual parameters, and the modified value gets stored at the same address.

Call by value in C	**Call by reference**
A copy of the value is passed into the function	An address of value is passed into the function
Changes made inside the function is limited to the function only. The values of the actual parameters do not change by changing the formal parameters.	Changes made inside the function validate outside of the function also. The values of the actual parameters do change by changing the formal parameters.
Actual and formal arguments are created at the different memory location	Actual and formal arguments are created at the same memory location

Call by value in C

```c
#include<stdio.h>
void change(int num) {
    printf("Before adding value inside function num=%d \n",num);

    num=num+100;
    printf("After adding value inside function num=%d \n", num);

}
int main() {
    int x=100;
    printf("Before function call x=%d \n", x);
    change(x);//passing value in function
    printf("After function call x=%d \n", x);
return 0;
}
```

Call by reference

```c
#include<stdio.h>
void change(int *num) {
    printf("Before adding value inside function num=%d \n",*num);

    (*num) += 100;
    printf("After adding value inside function num=%d \n", *num);

}
int main() {
    int x=100;
    printf("Before function call x=%d \n", x);
    change(&x);//passing reference in function
    printf("After function call x=%d \n", x);
return 0;
}
```

Swapping using Call by reference

```
#include <stdio.h>
void swap(int *, int *); //prototype of the function
int main()
{
    int a = 10;
    int b = 20;
    printf("Before swapping the values in main a = %d, b = %d\n",a,b); // printing the value of a and b in main
    swap(&a,&b);
    printf("After swapping values in main a = %d, b = %d\n",a,b); // The values of actual parameters do change in call
by reference, a = 10, b = 20
}
void swap (int *a, int *b)
{
    int temp;
    temp = *a;
    *a=*b;
    *b=temp;
    printf("After swapping values in function a = %d, b = %d\n",*a,*b); // Formal parameters, a = 20, b = 10
}
```

NOTE: Thus actual values of a and b get changed after exchanging values of x and y. In call by reference we can alter the values of variables through function calls. Pointer variables are necessary to define to store the address values of variables.

Swapping using call value

```
#include <stdio.h>
void swap(int , int); //prototype of the function
int main()
{
    int a = 10;
    int b = 20;
    printf("Before swapping the values in main a = %d, b = %d\n",a,b); // printing the value of a and b in main
    swap(a,b);
    printf("After swapping values in main a = %d, b = %d\n",a,b); // The value of actual parameters do not change by c
hanging the formal parameters in call by value, a = 10, b = 20
}
void swap (int a, int b)
{
    int temp;
    temp = a;
    a=b;
    b=temp;
    printf("After swapping values in function a = %d, b = %d\n",a,b); // Formal parameters, a = 20, b = 10
}
```

Note: Thus actual values of a and b remain unchanged even after exchanging the values of x and y. In call by values we cannot alter the values of actual variables through function calls. Values of variables are passes by Simple technique.

Reference: https://www.sitesbay.com/cprogramming/c-call-by-value-call-by-reference

https://www.youtube.com/watch?v=HEiPxjVR8CU&t=194s

https://www.youtube.com/watch?v=Xs22zOBfaoM&t=750s