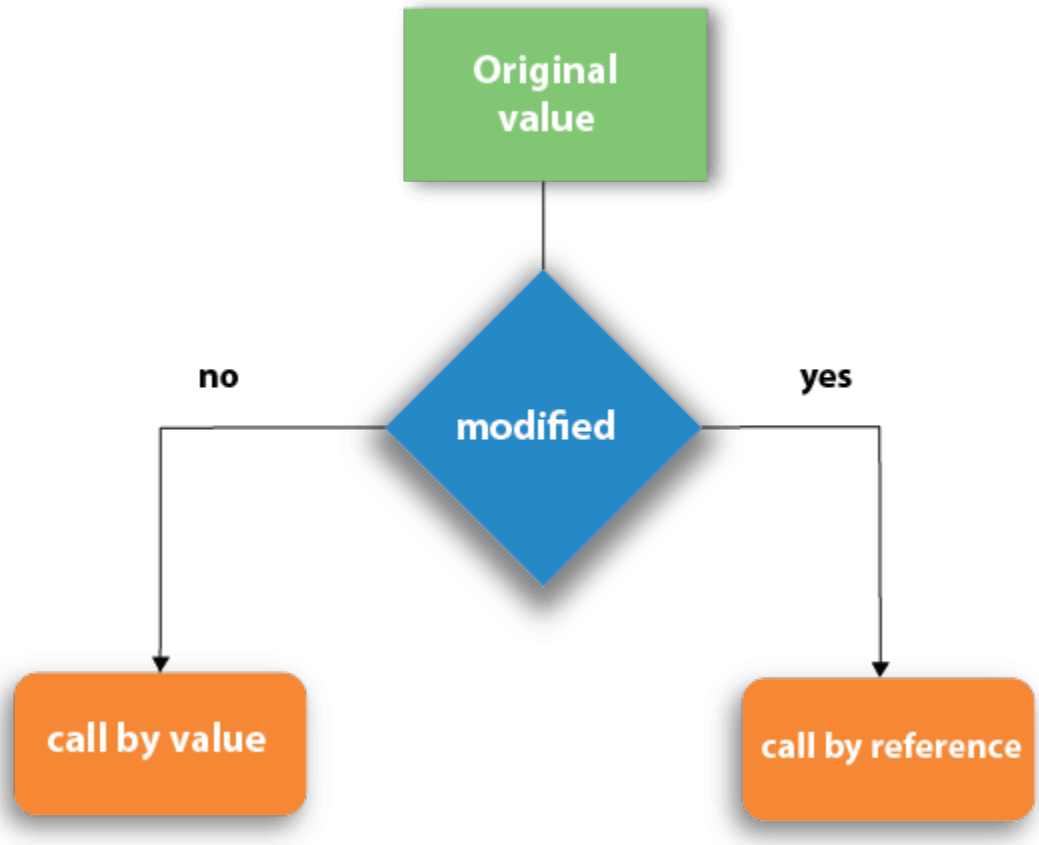
Call by value and Call by reference in C

There are two methods to pass the data into the function in C language, i.e., *call by value* and *call by reference*.



Call by value in C

- In the call by value method, the value of the actual parameters is copied into the formal parameters.
- In the call by value method, we can not modify the value of the actual parameter by the formal parameter.

- In call by value, different memory is allocated for actual and formal parameters since the value of the actual parameter is copied into the formal parameter.
- The actual parameter is the argument which is used in the function call whereas the formal parameter is the argument which is used in the function definition.
- Call by value method copies the value of an argument into the formal parameter of that function. Therefore, changes made to the parameter of the main function do not affect the argument.

Call by Value Example: Swapping the values of the two variables

```
#include<stdio.h>
void swap(int a, int b); //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);
   swap(a,b); //function call
   printf("After swapping values in main a = %d, b = %d\n",a,b);
}
void swap(int a, int b)   //function definition  //a=10    //b=20
{
   int temp;
   temp = a;         //temp=10
   a=b;              //a=20
```

```
    b=temp;           //b=10
    printf("After swapping values in function a = %d, b = %d\n",a,b);
}
```

Output:

Before swapping the values in main a = 10, b = 20

After swapping values in function a = 20, b = 10

After swapping values in main a = 10, b = 20

Call by reference in C

- 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.

While calling a function, instead of passing the values of variables, we pass address of variables(location of variables) to the function known as “Call By References.

```
// C program to swap 2 numbers by  Call by Reference

#include <stdio.h>

// Function Prototype
```

```
void swapx(int* x, int* y);
// Main function
int main()
{
    int x = 10, y = 20;              //x = 6005 =10     //y=6010  =20
    printf("Before swapping in main:x=%d y=%d\n", x, y);
    // Pass reference
    swapx(&x, &y); //function call
    printf("After swapping in main:x=%d y=%d\n", x, y);
    return 0;
}
void swapx(int* x, int* y)     //*x=6005        *y=6010
{
    int t;
    t = *x;              //t=10
    *x = *y;              //*x=20
    *y = t;              //*y=10
    printf("After swapping in this swapx function:x=%d y=%d\n", *x, *y);
}
```

Output:

Before swapping in main:x=10 y=20

After swapping in this swapx function:x=20 y=10

After swapping in main:x=20 y=10

Differences:-

Parameters	Call by value	Call by reference
Definition	While calling a function, when you pass values by copying variables, it is known as “Call By Values.”	While calling a function, in programming language instead of copying the values of variables, the address of the variables is used; it is known as “Call By References.
Arguments	In this method, a copy of the variable is passed.	In this method, a variable itself is passed.
Effect	Changes made in a copy of a variable never modify the value of the variable outside the function.	Change in the variable also affects the value of the variable outside the function.
Alteration of value	Does not allow you to make any changes in the actual variables.	Allows you to make changes in the values of variables by using function calls.
Passing of variable	Values of variables are passed using a straightforward method.	Pointer variables are required to store the address of variables.

Value modification	Original value not modified.	The original value is modified.
Memory Location	Actual and formal arguments will be created in different memory location	Actual and formal arguments will be created in the same memory location
Safety	Actual arguments remain safe as they cannot be modified accidentally.	Actual arguments are not Safe. They can be accidentally modified, so you need to handle arguments operations carefully.