C LIBRARY FUNCTIONS

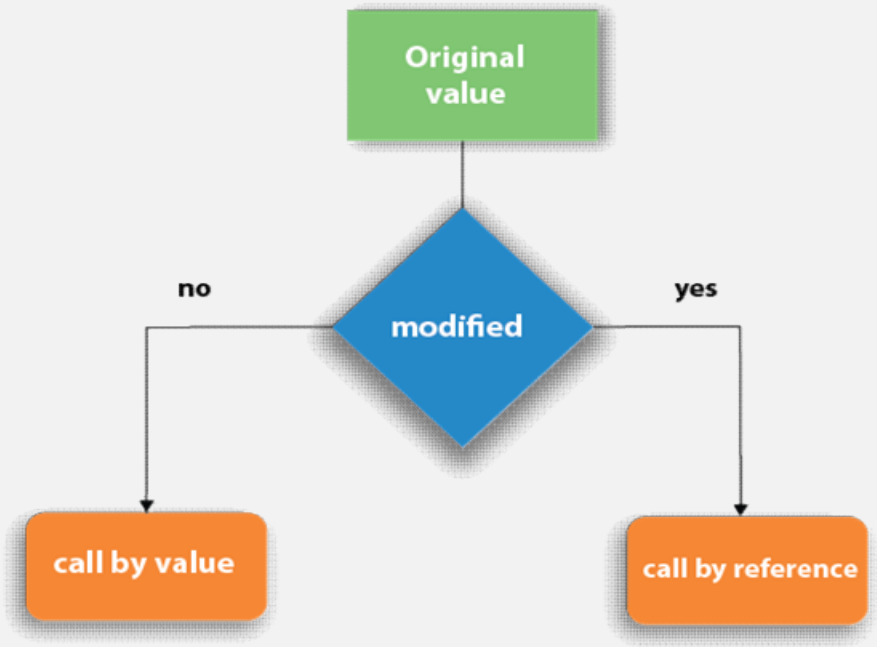
- Library functions are the inbuilt function in C that are grouped and placed at a common place called the library.
- Such functions are used to perform some specific operations.
- For example, printf is a library function used to print on the console.
- The library functions are created by the designers of compilers.
- All C standard library functions are defined inside the different header files saved with the extension **.h**.
- We need to include these header files in our program to make use of the library functions defined in such header files.
- For example, To use the library functions such as printf/scanf we need to include stdio.h in our program which is a header file that contains all the library functions regarding standard input/output.

SN	Header file	Description
1	stdio.h	This is a standard input/output header file. It contains all the library functions regarding standard input/output.
2	conio.h	This is a console input/output header file.
3	string.h	It contains all string related library functions like gets(), puts(),etc.
4	stdlib.h	This header file contains all the general library functions like malloc(), calloc(), exit(), etc.
5	math.h	This header file contains all the math operations related functions like sqrt(), pow(), etc.
6	time.h	This header file contains all the time-related functions.
7	ctype.h	This header file contains all character handling functions.

SN	Header file	Description
8	stdarg.h	Variable argument functions are defined in this header file.
9	signal.h	All the signal handling functions are defined in this header file.
10	setjmp.h	This file contains all the jump functions.
11	locale.h	This file contains locale functions.
12	errno.h	This file contains error handling functions.
13	assert.h	This file contains diagnostics functions.

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 call by value method, the value of the actual parameters is copied into the formal parameters. In other words, we can say that the value of the variable is used in the function call in the call by value method.
- In 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 formal parameter is the argument which is used in the function definition.

```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;
}
```

```
Before function call x=100
Before adding value inside function num=100
After adding value inside function num=200
After function call x=100
```

```c
#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 changing the
formal parameters in call by value, a = 10, b = 20
}
```

```c
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
}
```

```
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
```

PASSING ARRAY TO FUNCTION IN C

- Consider the following syntax to pass an array to the function.

```
functionname(arrayname);//passing array
```

METHODS TO DECLARE A FUNCTION THAT RECEIVES AN ARRAY AS AN ARGUMENT

- There are 3 ways to declare the function which is intended to receive an array as an argument.

First way:

- return_type function(type arrayname[])
- Declaring blank subscript notation [] is the widely used technique.

Second way:

- return_type function(type arrayname[SIZE])
- Optionally, we can define size in subscript notation [].

Third way:

- return_type function(type *arrayname)
- You can also use the concept of a pointer. In pointer chapter, we will learn about it.

```
#include<stdio.h>
int minarray(int arr[],int size){
int min=arr[0];
int i=0;
for(i=1;i<size;i++){
if(min>arr[i]){
min=arr[i];
}
}//end of for
return min;
}//end of function
```

```
int main(){
int i=0,min=0;
int numbers[]={4,5,7,3,8,9};//declaration of array
min=minarray(numbers,6);//passing array with size
printf("minimum number is %d \n",min);
return 0;
}
```

```
minimum number is 3
```

C FUNCTION TO SORT THE ARRAY

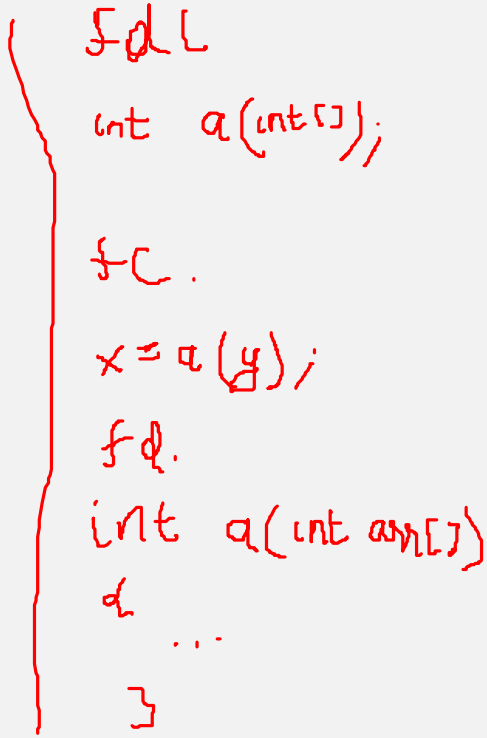
```
#include<stdio.h>
void Bubble_Sort(int[]);
void main ()
{
    int arr[10] = { 10, 9, 7, 101, 23, 44, 12, 78, 34, 23};

    Bubble_Sort(arr);
}
```

```
void Bubble_Sort(int a[]) //array a[] points to arr.
{
int i, j,temp;
    for(i = 0; i<10; i++)
    {
        for(j = i+1; j<10; j++)
        {
            if(a[j] < a[i])
            {
                temp = a[i];
                a[i] = a[j];
                a[j] = temp;
            }
        }
    }
    printf("Printing Sorted Element List ...\n");
        for(i = 0; i<10; i++)
        {
            printf("%d\n",a[i]);
        }
}
```

Printing Sorted Element List ...
7
9
10
12
23
23
34
44
78
101

```c
#include <stdio.h>
void getarray(int arr[])
{
    printf("Elements of array are : ");
    for(int i=0;i<5;i++)
    {
        printf("%d ", arr[i]);
    }
}
```

```c
int main()
{
   int arr[5]={45,67,34,78,90};
   getarray(arr);
   return 0;
}
```




```
Elements of array are : 45 67 34 78 90
```