# Types of functions

## Types of functions:

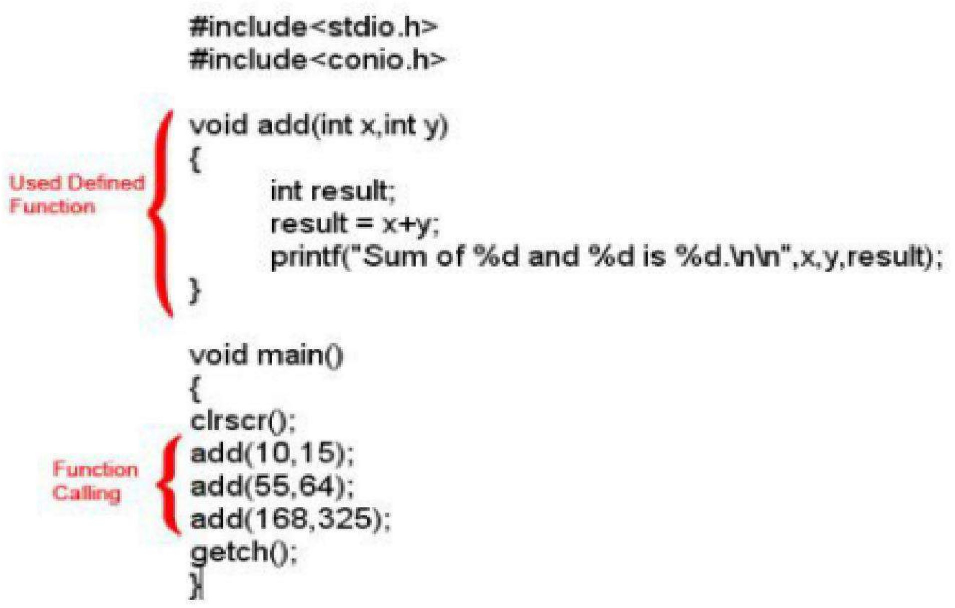
A function may belong to any one of the following categories:

1. Functions with no arguments and no return values.
2. Functions with arguments and no return values.
3. Functions with arguments and return values.
4. Functions that return multiple values.
5. Functions with no arguments and return values.

### Example of a simple function to add two integers.

view plain

```
#include<stdio.h>
#include<conio.h>
void add(int x,int y)
{
int result;
result = x+y;
printf("Sum of %d and %d is %d.\n\n",x,y,result);
}
void main()
{
clrscr();
add(10,15);
add(55,64);
add(168,325);
getch();
}
```

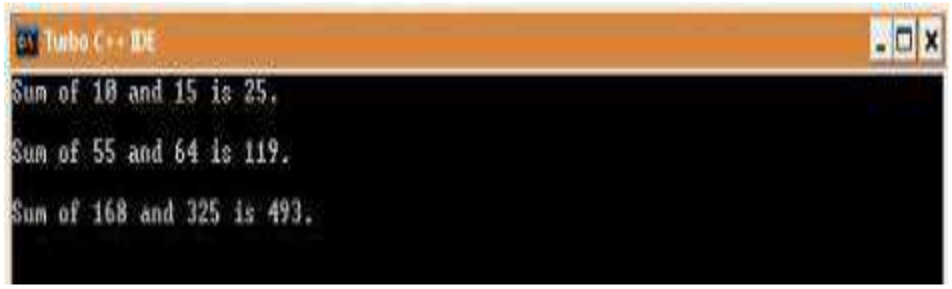


```
#include<stdio.h>
#include<conio.h>

void add(int x,int y)
{
        int result;
        result = x+y;
        printf("Sum of %d and %d is %d.\n\n",x,y,result);
}

void main()
{
clrscr();
add(10,15);
add(55,64);
add(168,325);
getch();
}
```

## Program Output



```
Sum of 10 and 15 is 25.

Sum of 55 and 64 is 119.

Sum of 168 and 325 is 493.
```

## Types of Function in C Programming Languages:

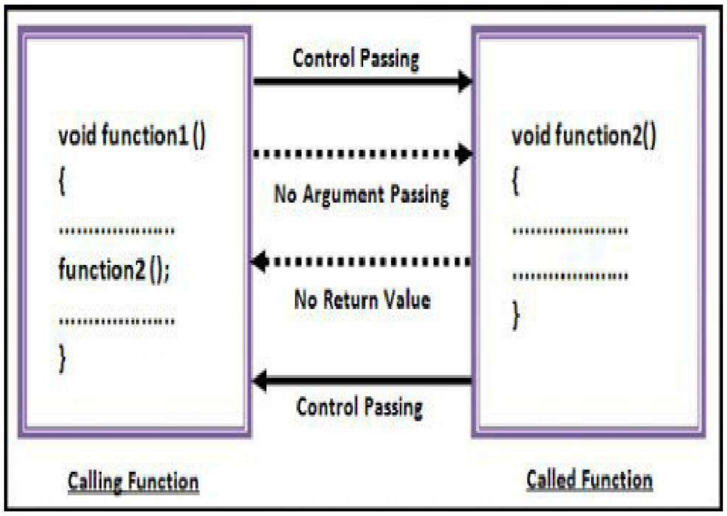
five types of functions and they are:

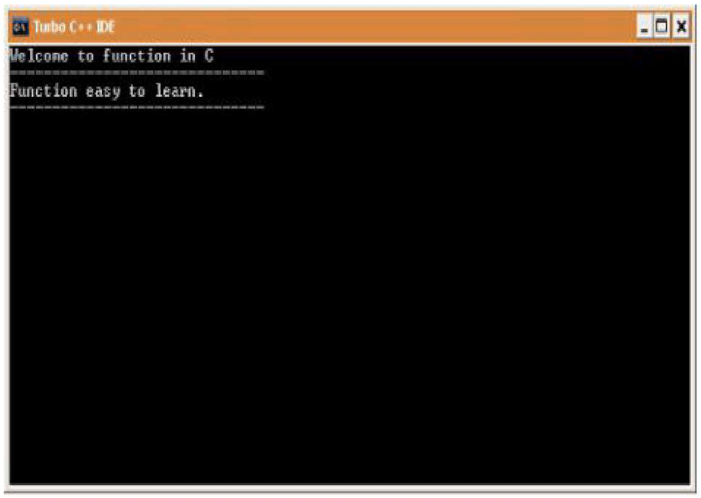
1. Functions with no arguments and no return values.
2. Functions with arguments and no return values.
3. Functions with arguments and return values.
4. Functions that return multiple values.
5. Functions with no arguments and return values.

### 1. Functions with no arguments and no return value.

A C function without any arguments means you cannot pass data (values like int, char etc) to the called function. Similarly, function with no return type does not pass back data to the calling function. It is one of the simplest types of function in C. This type of function which does not return any value cannot be used in an expression it can be used only as independent statement. Let's have an example to illustrate this.

```
#include<stdio.h>
#include<conio.h>
void printline()
{
int i;
printf("\n");
for(i=0;i<30;i++)
{
printf("-");
}
printf("\n");
}
void main()
{
clrscr();
printf("Welcome to function in C");
printline();
printf("Function easy to learn.");
printline();
getch();
}
```





```
Welcome to function in C
------------------------------
Function easy to learn.
------------------------------
```

Output of above program.

## Source Code Explanation:

The above C program example illustrates that how to declare a function with no argument and no return type. I am going to explain only important lines only because this C program example is for those who are above the beginner level.

**Line 3-12:** This C code block is a user defined function (UDF) whose task is to print a horizontal line. This is a simple function and a basic programmer can understand this. As you can see in line no. 7 I have declared a "for loop" which loops 30 time and prints "-" symbol continuously.

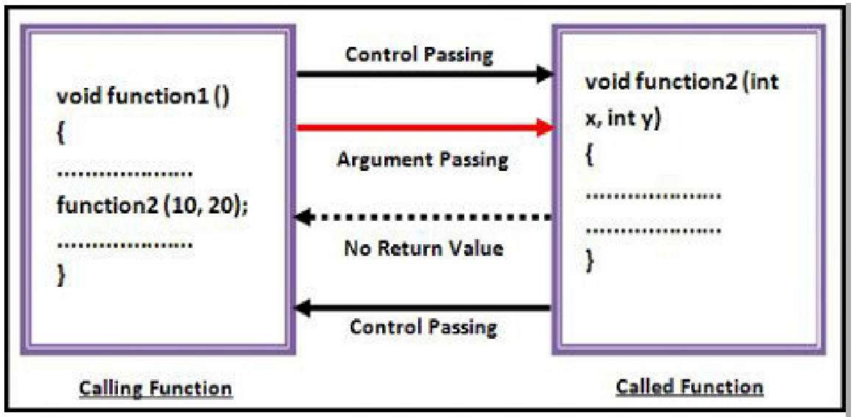
**Line 13-21:** These line are "main()" function code block. Line no. 16 and 18 simply prints two different messages. And line no. 17 and 18 calls our user defined function "printline()". You can see output this program below

## 2. Functions with arguments and no return value.

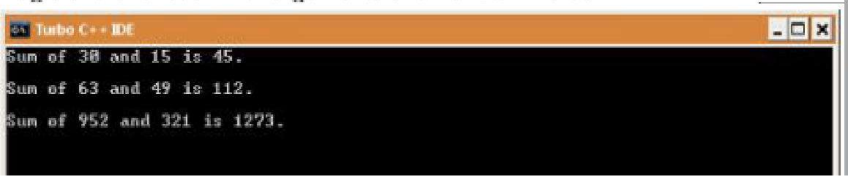
In our previous example what we have noticed that "main()" function has no control over the UDF "printfline()", it cannot control its output. Whenever "main()" calls "printline()", it simply prints line every time. So the result remains the same.

A C function with arguments can perform much better than previous function type. This type of function can accept data from calling function. In other words, you send data to the called function from calling function but you cannot send result data back to the calling function. Rather, it displays the result on the terminal. But we can control the output of function by providing various values as arguments. Let's have an example to get it better.

```
#include<stdio.h>
#include<conio.h>
void add(int x, int y)
{
int result;
result = x+y;
printf("Sum of %d and %d is %d.\n\n",x,y,result);
}
void main()
{
clrscr();
add(30,15);
add(63,49);
add(952,321);
getch();
}
```



Logic of the function with arguments and no return value.



```
Sum of 30 and 15 is 45.

Sum of 63 and 49 is 112.

Sum of 952 and 321 is 1273.
```

## Source Code Explanation:

This program simply sends two integer arguments to the UDF "add()" which, further, calculates its sum and stores in another variable and then prints that value. So simple program to understand.

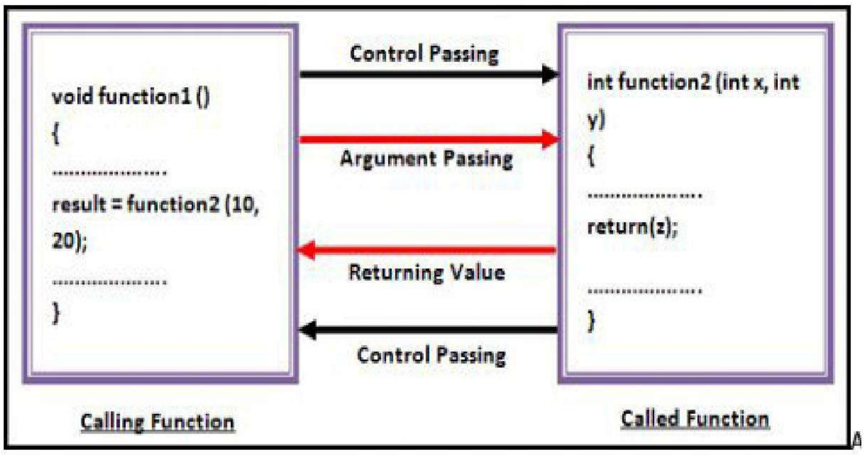
**Line 3-8:** This C code block is "add()" which accepts two integer type arguments. This UDF also has a integer variable "result" which stores the sum of values passed by calling function (in this example "main()"). And line no. 7 simply prints the result along with argument variable values.

**Line 9-16:** This code block is a "main()" function but only line no. 12, 13, 14 is important for us now. In these three lines we have called same function "add()" three times but with different values and each function call gives different output. So, you can see, we can control function's output by providing different integer parameters which was not possible in function type 1. This is the difference

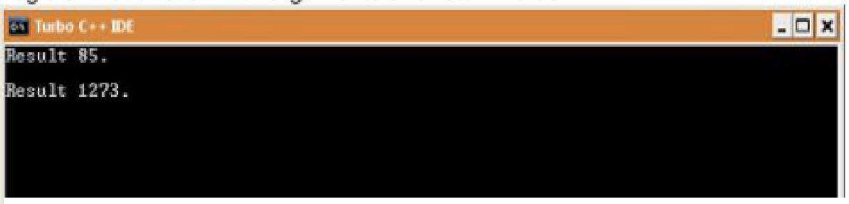
## 3. Functions with arguments and return value.

This type of function can send arguments (data) from the calling function to the called function and wait for the result to be returned back from the called function back to the calling function. And this type of function is mostly used in programming world because it can do two way communications; it can accept data as arguments as well as can send back data as return value. The data returned by the function can be used later in our program for further calculations.

```
1.   #include<stdio.h>
2.   #include<conio.h>
3.   int add(int x, int y)
4.   {
5.   int result;
6.   result = x+y;
7.   return(result);
8.   }
9.   void main()
10.  {
11.  int z;
12.  clrscr();
13.  z = add(952,321);
14.  printf("Result %d.\n\n",add(30,55));
15.  printf("Result %d.\n\n",z);
16.  getch();
17.  }
```



Logic of the function with arguments and return value.



```
Result 85.

Result 1273.
```

Output of the above program.

## Source Code Explanation:

This program sends two integer values (x and y) to the UDF "add()", "add()" function adds these two values and sends back the result to the calling function (in this program to "main()" function). Later result is printed on the terminal.

**Line No. 3-8:** Look line no. 3 carefully, it starts with **int**. This int is the return type of the function, means it can only return integer type data to the calling function. If you want any function to return character values then you must change this to char type. On line no. 7 you can see return statement, return is a keyword and in bracket we can give values which we want to return. You can assign any integer value to experiment with this return which ultimately will change its output. Do experiment with all you program and don't hesitate.

**Line No. 9-17:** In this code block only line no. 13, 14 and 15 is important. We have declared an integer "z" which we used in line no. 13. Why we are using integer variable "z" here? You know that our UDF "add()" returns an integer value on calling. To store that value we have declared an integer value. We have passed 952, 321 to the "add()" function, which finally return 1273 as result. This value will be stored in "z" integer variable. Now we can use "z" to print its value or to other function.

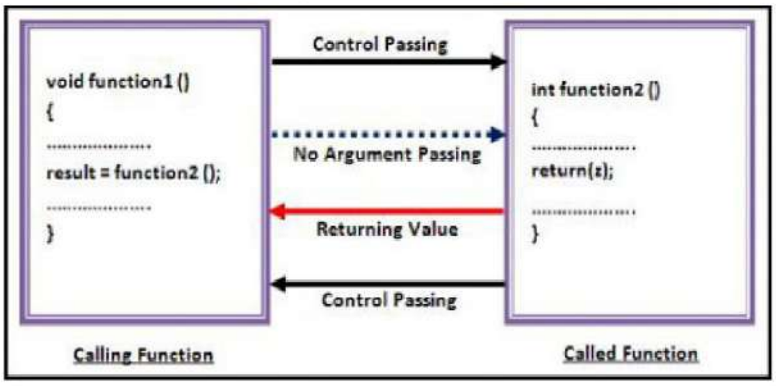
You will also notice some strange statement in line no. 14. Actually line no. 14 and 15 does the same job, in line no. 15 we have used an extra variable whereas on line no. 14 we directly printed the value without using any extra variable. This was simply to show you how we can use function in different ways.

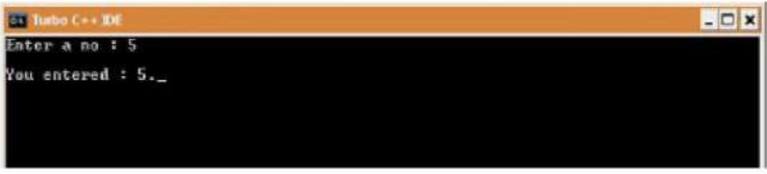
### 4. Functions with no arguments but returns value.

We may need a function which does not take any argument but only returns values to the calling function then this type of function is useful. The best example of this type of function is "getchar()" library function which is declared in the header file "stdio.h". We can declare a similar library function of own. Take a look.

```
#include<stdio.h>
#include<conio.h>
int send()
{
int no1;
printf("Enter a no : ");
scanf("%d",&no1);
return(no1);
}
void main()
{
int z;
clrscr();
z = send();
printf("\nYou entered : %d.", z);
getch();
}
```

**Functions with no arguments and return values.**





```
Enter a no : 5
You entered : 5._
```

### Source Code Explanation:

In this program we have a UDF which takes one integer as input from keyboard and sends back to the calling function. This is a very easy code to understand if you have followed all above code explanation. So I am not going to explain this code. But if you find difficulty please post your problem and I will solve that.

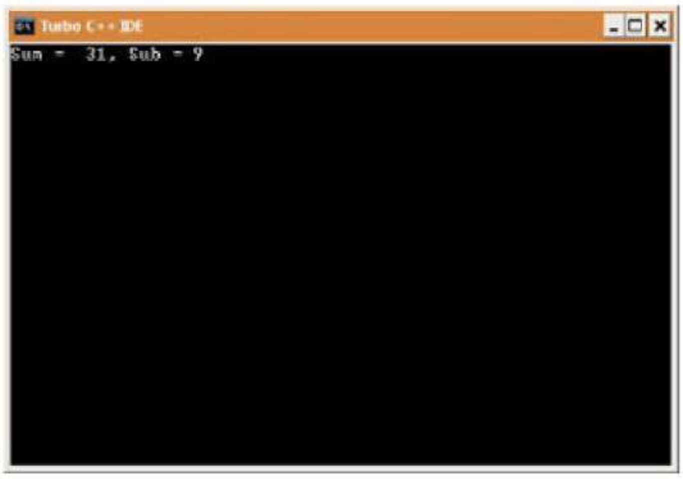
### 5. Functions that return multiple values.

So far, we have learned and seen that in a function, return statement was able to return only single value. That is because; a return statement can return only one value. But if we want to send back more than one value then how we could do this?

We have used arguments to send values to the called function, in the same way we can also use arguments to send back information to the calling function. The arguments that are used to send back data are called **Output Parameters**.

It is a bit difficult for novice because this type of function uses pointer. Let's see an example:

```
#include<stdio.h>
#include<conio.h>
void calc(int x, int y, int *add, int *sub)
{
*add = x+y;
*sub = x-y;
}
void main()
{
int a=20, b=11, p,q;
clrscr();
calc(a,b,&p,&q);
printf("Sum =  %d, Sub = %d",p,q);
getch();
}
```



```
Sum =  31, Sub = 9
```

Output of the above program.

## Source Code Explanation:

Logic of this program is that we call UDF "calc()" and sends argument then it adds and subtract that two values and store that values in their respective pointers. The "*" is known as indirection operator whereas "&" known as address operator. We can get memory address of any variable by simply placing "&" before variable name. In the same way we get value stored at specific memory location by using "*" just before memory address. These things are a bit confusing but when you will understand pointer then these thing will become clearer.

**Line no. 3-7:** This UDF function is different from all above UDF because it implements pointer. I know line no. 3 looks something strange, let's have a clear idea of it. "Calc()" function has four arguments, first two arguments need no explanation. Last two arguments are integer pointer which works as output parameters (arguments). Pointer can only store address of the value rather than value but when we add * to pointer variable then we can store value at that address.

**Line no. 8-15:** When we call "calc()" function in the line no. 12 then following assignments occurs. Value of variable "a" is assigned to "x", value of variable "b" is assigned to "y", address of "p" and "q" to "add" and "sub" respectively. In line no. 5 and 6 we are adding and subtracting values and storing the result at their respective memory location. This is how the program works.