Function

Course: Programming for Problem Solving (100204)

Topic: Function in C Lnaguage (Unit-6)

L-T-P: 3-0-4 Class: 2nd Semester

Branch:- Mechanical + Civil

Lectures: Monday, Tuesday, Thursday and Saturday

Labs: Monday, Tuesday, Wednesday, Friday and Saturday

Course co-ordinator: Akhilesh Kumar (8544152761)
akhilesh1987@gmail.com

Darbhanga College of Engineering, Darbhanga.

- **Why would we want to write functions?**
- It often happens that a particular piece of code is repeated many times in your program. It's repeated either literally or only with some minor modifications consisting of the use of other variables in the same algorithm.

Introduction of Function

A function in C language is a block of code that performs a specific task. It has a name and it is reusable i.e. it can be executed from as many different parts in a C Program as required. It also optionally returns a value to the calling program So function in a C program has some properties discussed below.

- Every function has a unique name. This name is used to call function from "main()" function. A function can be called from within another function.

- A function is independent and it can perform its task without intervention from or interfering with other parts of the program.

- A function performs a specific task. A task is a distinct job that your program must perform as a part of its overall operation, such as adding two or more integer, sorting an array into numerical order, or calculating a cube root etc.

Introduction of Function

- A function performs a specific task. A task is a distinct job that your program must perform as a part of its overall operation, such as adding two or more integer, sorting an array into numerical order, or calculating a cube root etc.

- A function returns a value to the calling program. This is optional and depends upon the task your function is going to accomplish. Suppose you want to just show few lines through function then it is not necessary to return a value. But if you are calculating area of rectangle and wanted to use result somewhere in program then you have to send back (return) value to the calling function.

- C language is collection of various inbuilt functions. If you have written a program in C then it is evident that you have used C's inbuilt functions. Printf, scanf, clrscr etc. all are C's inbuilt functions. You cannot imagine a C program without function.

Structure of a Function

- A general form of a C function looks like this:

<return type> Function Name (Argument1, Argument2, Argument3……)
{
Statement1;
Statement2;
Statement3;
}

- An example of function.
  ```
  int sum (int x, int y)
  {
  int result;
   result = x + y;
  return (result);
  }
  ```

Advantages of using functions:

- It makes possible top down modular programming. In this style of programming, the high level logic of the overall problem is solved first while the details of each lower level functions is addressed later.
- The length of the source program can be reduced by using functions at appropriate places.
- It becomes uncomplicated to locate and separate a faulty function for further study.
- A function may be used later by many other programs this means that a c programmer can use function written by others, instead of starting over from scratch.
- A function can be used to keep away from rewriting the same block of codes which we are going use two or more locations in a program. This is especially useful if the code involved is long or complicated.

Types of functions:

- Functions with no arguments and no return values.
- Functions with arguments and no return values.
- Functions with arguments and return values.
- Functions that return multiple values.
- Functions with no arguments and return values.

Example of a simple function to add two integers.

- #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);
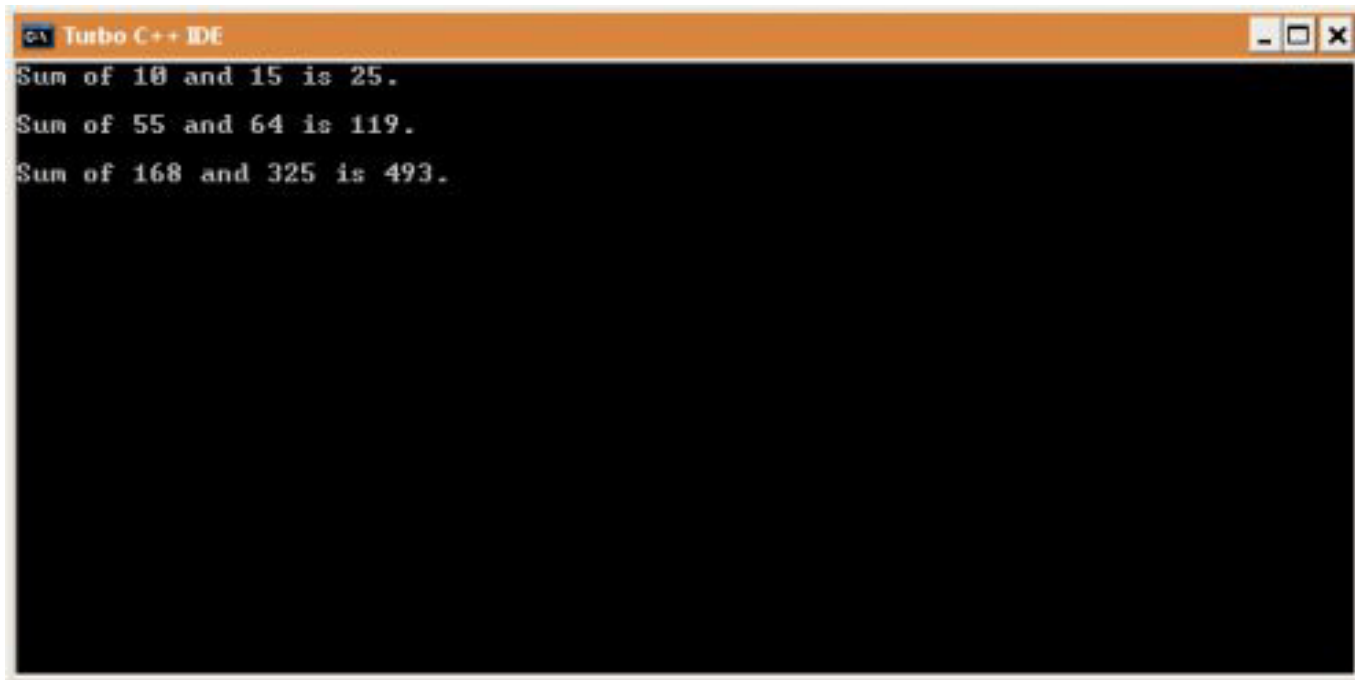- getch();
- }

```c
#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);

}
```

Used Defined Function

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

Function Calling

Output:

Turbo C++ IDE

```
Sum of 10 and 15 is 25.

Sum of 55 and 64 is 119.

Sum of 168 and 325 is 493.
```

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.

```c
#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();
}
```

- Logic of the functions with no arguments and no return value

| Control Passing |
| No Argument Passing |
| No Return Value |
| Control Passing |

void function1 ()
{
....
function2 ();
....
}

Calling Function

void function2()
{
....
....
}

Called Function
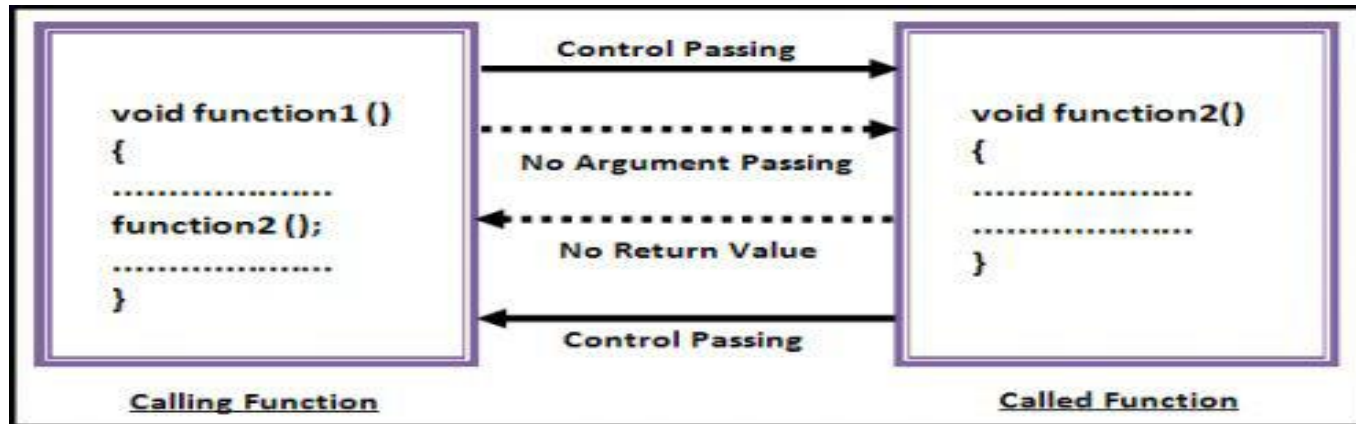

```
Turbo C++ IDE                                    _ □ ×
Welcome to function in C
Function easy to learn.
```
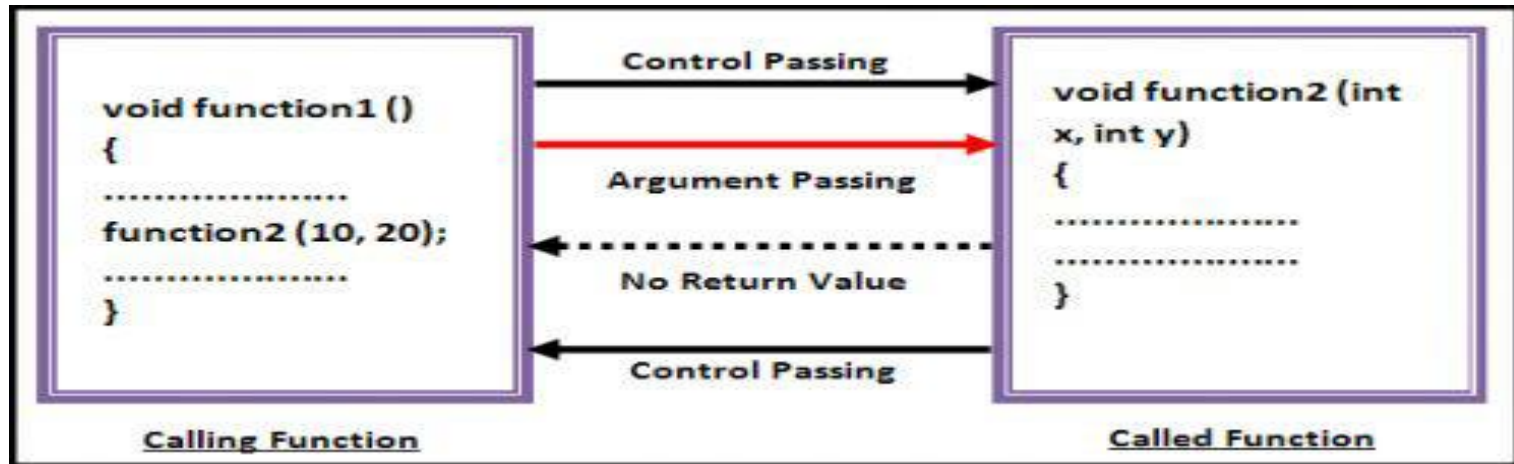
Source Code Explanation:

- 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()".

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.

```c
1.  #include<stdio.h>
2.  #include<conio.h>
3.  void add(int x, int y)
4.  {
5.  int result;
6.  result = x+y;
7.   printf("Sum of %d and %d is %d.\n\n",x,y,result);
8.  }
9.  void main()
10.  {
11. clrscr();
12. add(30,15);
13.  add(63,49);
14.  add(952,321);
15. getch();
16.  }
```

Logic of the function with arguments and no return value, and Output.


```
void function1 ()
{
...................
function2 (10, 20);
...................
}
```
Calling Function

Control Passing →

Argument Passing →

No Return Value ←

Control Passing ←

```
void function2 (int x, int y)
{
...................
...................
}
```
Called Function

```
Turbo C++ IDE                                    _ □ ×
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.

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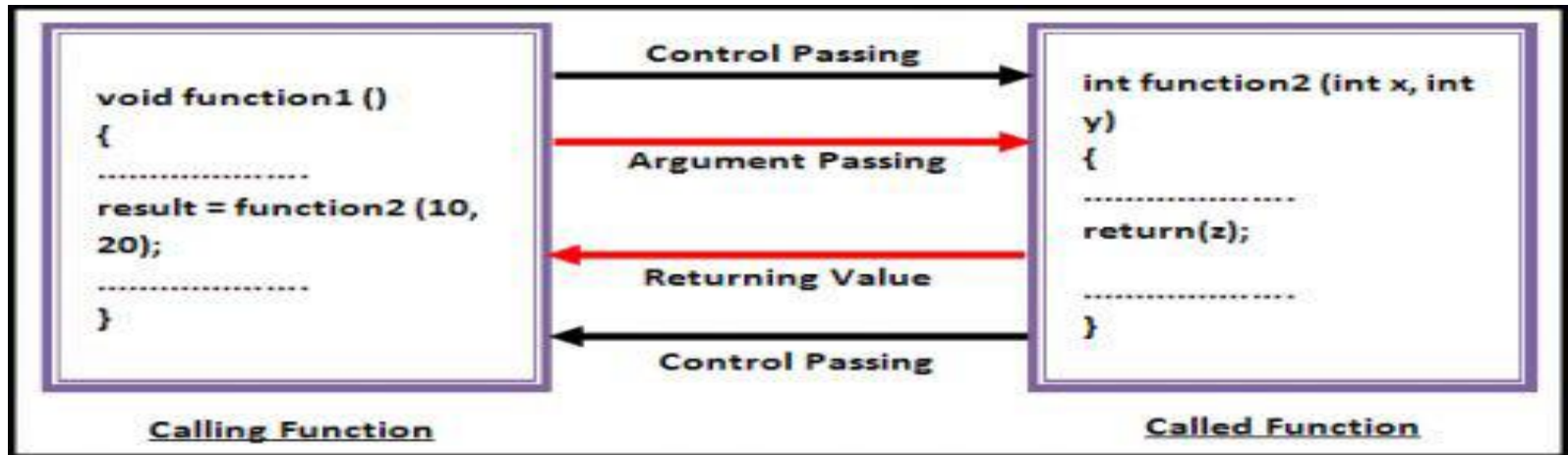
```c
1.#include<stdio.h>
2. #include<conio.h>
3. int add(int x, int y)
4. {
5. int result;
6. result = x+y;
7. return(result);
8. }
9. void main()
10. {
11. int z;
12. clrscr();
13. z = add(952,321);
14. printf("Result %d.\n\n",add(30,55));
15. printf("Result %d.\n\n",z);
16. getch();
17. }
```

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.

## Logic of the function with arguments and return value and Output.

Control Passing

```
void function1 ()
{
.....................
result = function2 (10,
20);
.....................
}
```

Argument Passing

Returning Value

Control Passing

```
int function2 (int x, int
y)
{
.....................
return(z);
.....................
}
```

Calling Function

Called Function

Turbo C++ IDE

```
Result 85.
Result 1273.
```

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.

```c
1. #include<stdio.h>
2. #include<conio.h>
3. int send()
4. {
5. int no1;
6. printf("Enter a no : ");
7. scanf("%d",&no1);
8. return(no1);
9. }
10. void main()
11. {
12. int z;
13. clrscr();
14. z = send();
15. printf("\nYou entered : %d.", z);
16. getch();
17. }
```

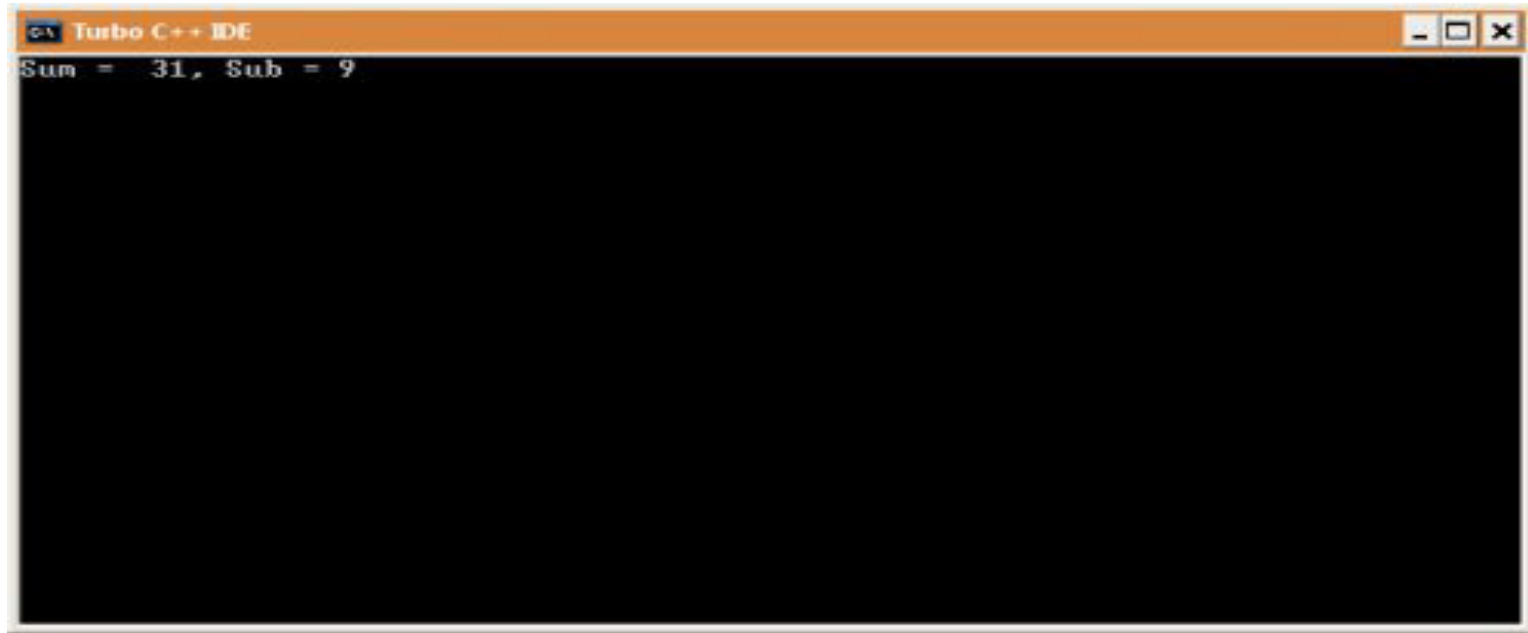# Functions with no arguments and return values and output.
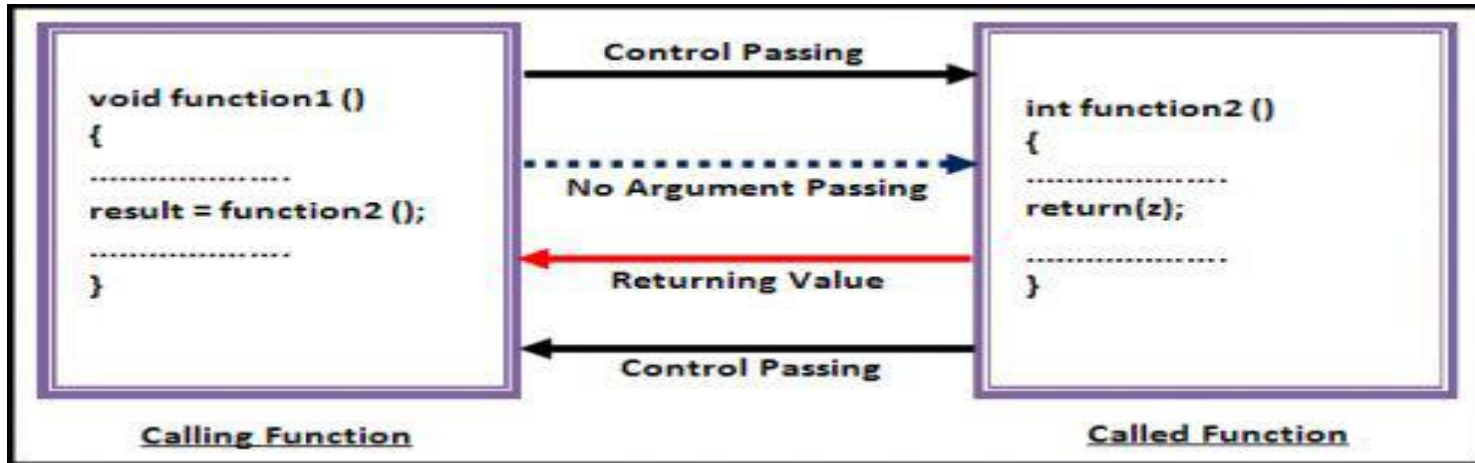
Functions that return multiple values.

- 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.

```c
1. #include<stdio.h>
2. #include<conio.h>
3. void calc(int x, int y, int *add, int *sub)
4. {
5. *add = x+y;
6. *sub = x-y;
7. }
8. void main()
9. {
10. int a=20, b=11, p,q;
11. clrscr();
12. calc(a,b,&p,&q);
13. printf("Sum = %d, Sub = %d",p,q);
14. getch();
15. }
```

Output of the above program

Turbo C++ IDE

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

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.

Thank You