FUNCTIONS in C

"A function is a set of statements that take inputs, do some specific computation and produces output."

"A function is a group of statements that together perform a specific task."

Function is a sub-program written for a specific task, either by programmer or by user.

Every c program has at least one function, which is main( ). There are other functions such as printf( ), scanf( ).

There are two types of functions.

1. Library function (or pre-defined function)
   Ex. printf( ), scanf( ) [stdio.h]
   getch( ), clrscr( ) [conio.h]

2. User-defined function (or sub-program)
   Ex. add( ), sum( ),factorial( ) etc.

The idea is to put some commonly or repeatedly done task together and make a function, so that instead of writing the same code again and again for different inputs, we can call the function.

Ex: Below is a simple C program to demonstrate functions in C.

```
/*Example of user-defined function*/
#include<stdio.h>
#include<conio.h>
void main( )
{
    int a,b,c;
    int add(int, int );              // function declaration
    printf("Enter any two values");
    scanf("%d %d",&a,&b);
    c=add(a,b );                     // calling function with actual arguments a, b
    printf("Sum=%d",c);
    getch();
}

int add( int x, int y)               // called function with formal arguments x,y
{
    int z;
    z=x+y;
    return z;
}
```

Function declaration and prototype

Any C function by default returns integer value.
But if the function returns value other than integer, there is need to write the return type of the function prefix to function name.

"If function doesn't return any value, the return type of the function is void."

ex. *float add(float x, float y)*
void int(int a,int b);
double square( );

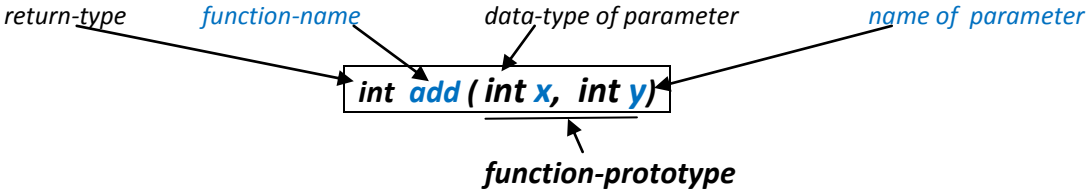
Like the simple variable declaration there is need to declare the function before its use.
The function declaration contains and tell tells compiler about :-

1. return type of function
2. name of the function
3. name and number of parameters in the function
4. data-types of parameters

Syntax of function declaration/prototype as below:

Return-type function-name (argument data-type list)

Ex.



"The ***(argument data-type list)*** in function declaration with data type and return value is called as ***function-prototype"***

Where, ***Return-type*** = *data-type of return value from function defination*
function-name = *name of user defined function*
(argument data-type list) = *data type and name of the formal parameters*

Putting parameter names in function declaration is optional [***int add(int , int)***,]
but it is necessary to put them in function definition [***int add(int x, int y)***.]

```
/*Example of function declaration and prototype*/
#include<stdio.h>
#include<conio.h>
void main( )
{
    float a,b,c;
    float add(float, float);                    // function declaration and prototype
    printf("Enter any two values");
    scanf("%f %f",&a,&b);
    c=add(a,b );                    // calling function with actual arguments a, b
    printf("Sum=%f",c);
    getch();
}

    float add(float  x, float y)      // called function with formal arguments x,y
    {
        float z;
        z=x+y;
        return z;
    }
```

In the above example ***float add(float, float)*** is the prototype of the function.
In function definition it returns value ***z*** ,which is of float type.
Hence the return type of the function is **float.**

Function definition

After function definition there is need to define it.
The function definition contains:-

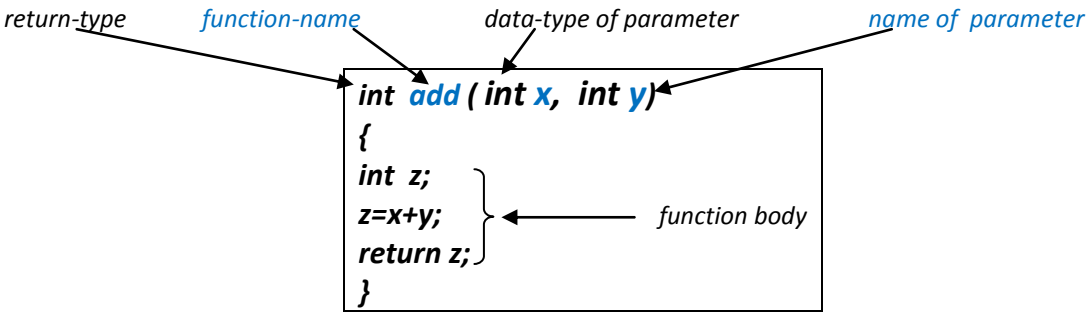
1. return type of function
2. name of the function
3. name and number of formal parameters in the function
4. data-types of parameters
5. task to be performed in function

Syntax of function definition as below

```
Return-type    function-name   (argument data-type list)
{
   Declaration of local variable;
   Executable part;                    } or function-body
   return (e);
}
```

Where, ***Return-type*** = *data-type of return value from function defination*
function-name = *name of user defined function*
(argument data-type list) = *data type and name of the formal parameters*
Declaration of local variable = declaration of formal parameters
Executable part = c statements for performing the actual task
return (e) = it returns the result to calling function

Ex.



In above example it defines the fuction add, with 2-formal parameters(x,y) and returns the addition which is stored in z using return statement.
As data ype of return- value (z) is int, return-type of the function is also int.

Function calling

After function declaration there is need to call it.
The function definition contains:-

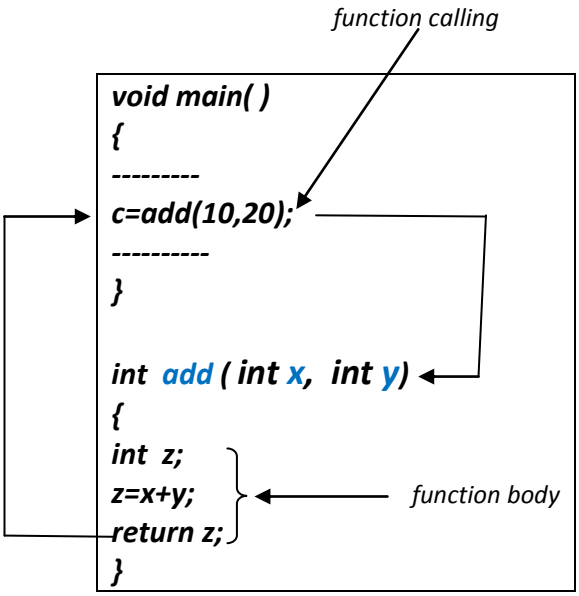
1. name of the function
2. value of actual parameters

Syntax of function calling as below

Var-name = function-name (value of actual-parameters);

Where, ***var-name =*** *name of the variable to which result (return value) is asssign*
function-name *= name of user defined function*
(value of actual-parameters) *=actual values on which we want to perform operation*

Ex. *c=add(10, 20);*
square(1.5);



In above example the line ***add(10,20)*** calls the function definition and
Function definition returns the addition which is stored in z using return statement.
As data ype of return- value (z) is int, return-type of the function is also int.

Function returning (return statement)

After calling the function the control is transfer to function definition.
In function definition it perform the specific task and calculate the result.
This result from function definition (called function) is transferred to calling function.

The function definition contains:-

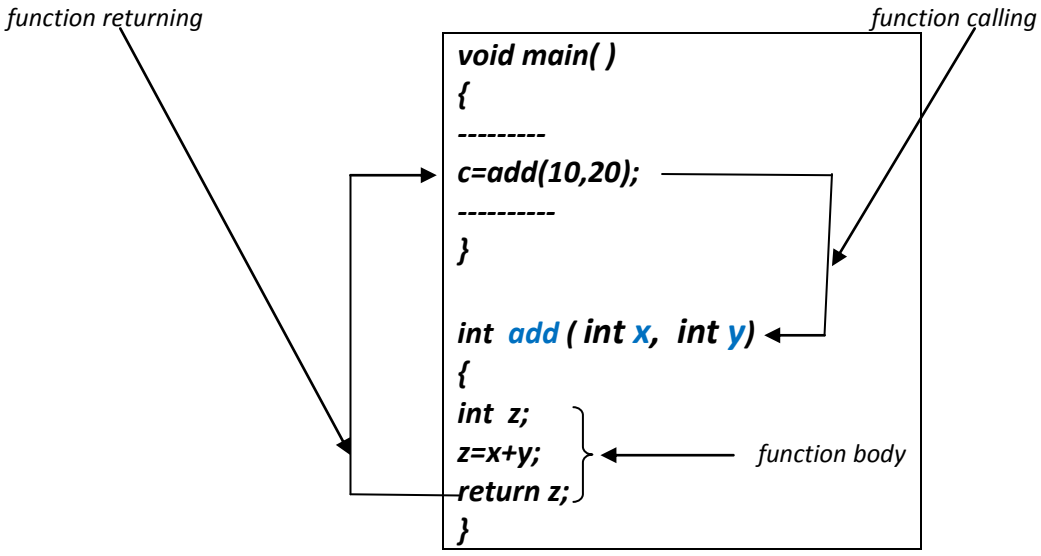
1. name of the function
2. value of actual parameters

Syntax of function returnig as below:-

return(expression);

Where, ***return=*** *returns the result.*
(expression)*= it may be variable or expression*

Ex. *return z;*
return (x+y);



In above example the line ***add(10,20)*** calls the function definition and
Function definition returns the addition which is stored in z using return statement.
As data ype of return- value (z) is int, return-type of the function is also int.

Call by value and Call by pointer(reference) in C

- There are two ways to pass arguments/parameters to function calls -- *call by value* and *call by reference*.
- The major difference between call by value and call by reference is that in ***call by value a copy of actual arguments is passed to respective formal arguments.***
- While, in ***call by reference the location (address) of actual arguments is passed to formal arguments,*** hence any change made to formal arguments will also reflect in actual arguments.

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.

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

```
/*Program for swapping of two numbers using call by value*/
#include <stdio.h>
#include<conio.h>
    void main( )
    {
            int  a = 10,  b = 20;
            void swap(int , int);                    //prototype of the function
             printf("Before swapping the values in main a = %d, b = %d ",a,b);
                                                     // printing the value of a and b in main
            swap(a,b);
             printf("After swapping values in main a = %d, b = %d ",a,b);
                                /* The value of actual parameters do not change by changing
                                 the formal parameters in call by value, a = 10, b = 20 */
            getch( );
    }
```

```
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
}
```

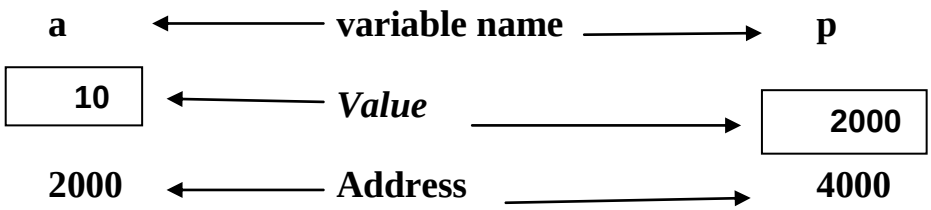
Output

```
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
```

Pointer

Simple variable:- int a = 10;

Pointer variable:- int *p = &a;



**Address of operator *
Value of operator &**

Hence, value of p is 2000 i.e address of variable a

And *p gives result 10

Call by pointer (reference) in C

There are some cases where we need *call by reference*:

1. The ***called function communicates to the calling function only through return statement***
2. ***Return statement can only send only one value back to the calling function.*** *If there are more than one value we want to alter, call by reference is required*
3. If the size of data is large , copying actual arguments to formal arguments could be a time consuming operation and occupies more memory.
4. **To achieve call by reference functionality in C language the calling function provides the address of the variable**
5. **The called function declares the parameter to be a pointer**
6. **Since the address of the argument is passed to the function, code within the called function can change the value of the actual arguments.**

- 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 reference Example: Swapping the values of the two variables

```
/*Program for swapping of two numbers using call by reference*/
#include <stdio.h>
#include<conio.h>
void main()
{
    int a = 10,  b = 20;
    void swap(int *, int *);        //prototype of the function

    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 i
                                    n 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
}
```

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 = 20, b = 10

Difference between call by value and call by reference in c

No.	Call by value	Call by reference
1	A copy of the value is passed into the function	An address of value is passed into the function
2	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.
3	Actual and formal arguments are created at the different memory location	Actual and formal arguments are created at the same memory location
4	In call by value, actual arguments will remain safe, they cannot be modified accidentally	In *call by reference*, alteration to actual arguments is possible within from called function; therefore the code must handle arguments carefully else you get unexpected results.
5	In *call by value*, a copy of actual arguments is passed to formal arguments of the called function and any change made to the formal arguments in the called function have no effect on the values of actual arguments in the calling function	In *call by reference*, the location (address) of actual arguments is passed to formal arguments of the called function. This means by accessing the addresses of actual arguments we can alter them within from the called function.

Difference between Library function and user defined function

No.	Library function	User defined function
1.	These are predefined functions.	These are the functions created by user, as per his own requirement.
2.	Library functions are part of header file.(math.h)	User defined functions are part of program.
3.	Its name is decided by developer	Its name is decided by user
4.	Its name can't be change.	Its name can be change.
5.	Prototype is not require.	Prototype is require.
Ex.	sin( ),cos( )	Add( ), sum ( )

Difference between Local variable and Global variable.

No.	Local variable	Global variable
1.	It is declared inside the function body.	It is declared outside the function body.
2.	It is declared after the main( ) funcrtion.	It is declared before the main( ) funcrtion.
3.	The life and scope of local variable is upto the end of the function (in which it is declared).	The life and scope of global variable is upto the end of program (throughout the program.)
4.	It may be declared within the main( ) or any other user defined fuction.	It must be decared before the main( ).
5.	The value of local variable can't be used in another function.	

Difference between Actual argument and Formal argument.

No.	Actual argument	Formal argument
1.	Actual arguments are present in calling function.	Formal arguments are present in called function.
2.	It is present in main( ) function.	It is present in function definition (user defined function).

Pointer to function

- Every type of variable has an address. C functions also have an address.
- If we know the functions address, we can point to it, which provides the way to call the function.
- The ***address*** of function is ***obtained by*** using the ***functions name without any paranthesis or argument.***

Ex. int add(int a,int b);

Here above functions address is obtained by using,

add ← *(That is function name without argument and paranthesis.)*

- It is possible to declare a pointer to a function, which can be used as an argument in another function.

Syntax for declaration of pointer to function as below:

```
return_type (*Pointer_name) (function argument list);
```

For example: ***int (*fptr) (int,int);***

fptr = add; ***or*** ***fptr = &add;***

Here ***int*** is a return type of function,
fptr is name of the function pointer and
(int,int) is an argument list of this function.

Lets understand this with the help of an example: Here we have a function ***add*** that calculates the sum of two numbers and returns the sum. We have created a pointer ***fptr*** that points to this function, we are invoking the function using this function pointer ***fptr***.

```
/* Program for pointer to function */
#include<stdio.h>
#include<conio.h>
void main()
{
        int sum1,sum2;
        int add(int,int);
        int (*fptr)(int,int);
        fptr = add;                    or  fptr = &add;
        sum1 = add(10,20);
        sum2 = fptr(10,20);
        printf(" sum using function = %d ",sum1);
        printf(" sum using function pointer = %d ",sum2);
        getch( );
}
```

```
int add(int x,int y)
{
        int z;
        z=x+y;
        return  z;
}
```

or

```
return x+y;
```

Output:-

sum using function = 30

sum using function pointer = 30

Some points regarding pointer to function:

1. It is possible to declare a function pointer and assign a function address to it in a single statement like this:

```
void (*fun_ptr)(int) = &fun;
```

2. Also it is possible to remove the ampersand from this statement because a function name alone represents the function address. This means the above statement can also be written like this:

```
void (*fun_ptr)(int) = fun;
```

```
/* Program for address of function*/
#include<stdio.h>
#include<conio.h>
void main( )
{
    int display( );
    printf("Address of function= %u",display);
    display( );
    getch( );
}

    display( )
    {
        printf("Hello");
    }
```

Output:

Address of function = 2000

Hello