

# Functions

---

## Function Definition:

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

Every C program has at least one function which is `main()`, and all the most small program can define additional functions. To make programming simple and easy to debug, we break a larger program into smaller subprograms which perform well define task. These subprograms are called **function**. Functions are the building blocks of C program. Every C program can be thought of as a collection of these function.

A function declaration tells the compiler about a function's name and return type. A function definition provides the actual body of the function.

## Advantages:

- The program will be easy to understand.
- The large programs can divided into smaller modules.
- Reusable codes that can be used in other programs.

## They are two types of Function

- Standard library functions
- User-define function

## User-Define Function:

C Programming can allow the user to put the function according to your needs.

## Function Declaration:

The function declaration tells the compiler about a function's name, return type and parameters. It doesn't contain function body.

**Function Declaration is also called as Function Prototype.**

### Syntax:

```
return_type function_name(type1 argument1, type2 argument2, ...);
```

Function declaration is required when you define a function in one source file and you call that function in another file. In such case, you should declare the function at the top of the file calling the function.

## Function Definition:

It contains the block of code to perform a specific task.

The code written for the function in the program is known as "function definition".

### Syntax:

```
returntype functionname(type1 argument1, type2 argument2, ...)  
{
```

```
//body of the function  
}
```

When a function is called, the control of the program is transferred to the function definition.

### Function Call:

Control of the program is transferred to the user-defined function by calling it.

### Syntax:

```
functionname(argument1, argument2, ...);
```

### Function Arguments:

If the function is to use arguments, those parameters that are used to accept value passed by calling function. The arguments declared in the function header is called as **formal arguments**.

The formal arguments are the parameters/arguments in a function declaration. The scope of formal arguments is local to the function definition in which they are used. Formal arguments belong to the called function.

There are two ways in which arguments can be passed to a function:

**Call by Value:** In this method, the value of the each of the actual arguments in the calling function is copied into corresponding formal arguments of the called function.

**Call by Reference:** Instead of passing the values of variables, we pass location of variable to the function.

### **Return Statement:**

The return statement is used to terminate the execution of a function and transfer program control back to the calling function.

### **Example:**

```
#include<stdio.h>
void name();
void main ()
{
    printf("Welcome to the ");
    name();
}
void name()
{
    printf("Gyansabha");
}
```

### **Output:**

Welcome to the Gyansabha

# Function Types

They are 4 types:

1. **Function with no arguments passed and no return value:** A function which does not have any arguments and does not return any value to the calling function.

**Example:**

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

```
void main()
```

```
{
```

```
    void addition (void);
```

```
    clrscr();
```

```
    addition();
```

```
    getch();
```

```
}
```

```
void addition()
```

```
{
```

```
    int a, b, c;
```

```
    printf("Enter Two Numbers: ");
```

```
    scanf("%d%d",&a,&b);
```

```
    c=a+b;
```

```
    printf("Addition of Two Numbers: %d", c);
```

```
}
```

In the above program, function addition() does not have any arguments. Also function has declared as void means it does not return value to the main program.

2. **Function with no arguments passed and with return value:** This type of function has no arguments but return value to the calling function.

**Example:**

```
#include<stdio.h>

void main()
{
    int s;
    int addition (void);
    s=addition();
    printf("%d", s);
    getch();
}

int addition()
{
    int a, b, c;
    printf("Enter Two Numbers: ");
    scanf("%d%d",&a,&b);
    c=a+b;
    return (c);
}
```

In the above program, function addition() does not contain any argument, but it return value to the point.

3. **Function with arguments passed and with no return value:** It is a type of function which accept arguments, but does not return any value.

**Example:**

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

```
void main()
```

```
{
```

```
    int x, y;
```

```
    void addition (int, int);
```

```
    printf("Enter Two Numbers: ");
```

```
    scanf("%d%d", &x, &y);
```

```
    addition(x,y);
```

```
    getch();
```

```
}
```

```
void addition()
```

```
{
```

```
    int c;
```

```
    c=a+b;
```

```
    printf("Addition of Two Numbers: %d", c);
```

```
}
```

In the above program, function addition() has two parameters x, y but it does not return value to the calling program.

#### 4. Function with argument passed and with return value:

**Example:**

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int s, x, y;
    int addition (int, int);
    clrscr();
    printf("Enter Two Number:");
    scanf("%d%d", &x, &y);
    s=addition(x, y);
    printf("sum: %d", s);
    getch();
}
int addition(int a, int b)
{
    int c;
    c=a+b;
    return (c);
}
```

In the program function addition() consists of parameters a and b. Also it return value to the back to the point.




## Recursion

Within a function body, if the function calls itself, the mechanism is known as 'recursion' and the function is known as recursive function. A recursive function calls itself. Recursive function are useful in evaluating certain type of mathematical function.

In this mechanism, a chaining of function calls occurs, so it is necessary for a recursive function to stop somewhere or it will result into infinite calling. So every recursive function must have terminating condition.

### Recursion Work

```
void recurse()  
{  
    ...  
    recurse();  
    ...  
}  
int main()  
{  
    ....  
    recurse();  
    ....  
}
```



### Recursion Process:

- The recursion continues until some condition is met to prevent it. To prevent infinite recursion.
- if-else statement can be used where one branch makes the recursive call and other doesn't.

**Example:**

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

```
int fact(int)
```

```
int main()
```

```
{
```

```
    int n, result;
```

```
    clrscr();
```

```
    printf("Enter n value:");
```

```
    scanf("%d", &n);
```

```
    result=fact(n);
```

```
    printf("Factorial of given number is %d:", result);
```

```
    getch();
```

```
}
```

```
int fact(int b)
```

```
{
```

```
    int res;
```

```
    if(b==0)
```

```
    {
```

```
        res=1;
```

```
    }
```

```
    else{  
        res=b*fact(b-1);  
    }  
    return res;  
}
```

**Output:**

Factorial of 4 is 24

GYANSABHA.IN

## Storage Classes:

Every C variable has a storage class and scope. A variable's storage class tells us:

- Where the variable would be stored.
- What will be the initial value of the variable, if initial value is not specifically assigned.
- What is the scope of variable i.e., in which function, the value of the variable would be available.
- What is the life time of the variable, i.e. how long would the variable exist.

Depending on this there are four storage classes:

### 1. Automatic:

- The variables are declared at the start of a block.
- The scope of automatic variables is local to the block in which they are declared. For these reasons, they are also called local variables.
- Automatic variables may be specified upon declaration to be of storage class auto.
- Automatic variable declared with initializers are initialized each time in the block in which they are declared.

**Example:**

```
#include <stdio.h>

int main()

{

    int a = 20,i;
```

```
printf("%d ",++a);  
  
{  
  
int a = 10;  
  
for (i=0;i<=10;i++)  
  
{  
  
    printf("%d ",a);  
  
}  
  
}  
  
printf("%d ",a);  
  
}
```

**Output:**

21 10 10 10 10 10 10 10 10 10 10 21

## 2. Register:

- Register variables are a special case of automatic variables.
- The storage cells are allocated to CPU registers.
- Register variables provide a certain control over efficiency of program execution.
- Variables which are used repeatedly are declared to storage register.
- It allocates storage upon entry to a block; and the storage is freed when the block is exited.
- The scope of register variables is local to the block in which they are declared.

- register variables are stored in register memory where as auto variables are stored in main CPU memory.
- Only few variables can be stored in register memory. So, we can use variables as register that are used vary often in a C Program.

### 3. Statics:

- Static storage class is declared with the keyword static as the class specifier when the variable is defined.
- Default initial value is zero.
- The scope of static automatic variables is local to the block in which it is defined.
- Static variables may be initialized in their declarations; however initialization is done only once at compile time when memory is allocated for the static variable.
- Value of the variable persists between different function calls.

#### Example:

```
#include<stdio.h>
void fun1(void);
int main()
{
    fun1();
    fun1();
}
void fun1()
{
    static int i=0;
```

```
i++;  
printf("i=%d\n", i);  
}
```

**Output:**

i=1

i=2

**4. External:**

- External variable may be declared outside any function block by specifying its type and name.
- A normal global variable can be made extern by placing the 'extern' keyword before its declaration/definition in any function/block.
- Storage is in memory.
- Default initial value is zero.
- The scope of external variables is global.
- External variables may be initialized in declarations just as automatic variables.
- It is a good programming practice to avoid use of external variables.

**Example:**

```
#include<stdio.h>  
  
int x=20;  
  
void main()
```

```
{  
    extern int y;  
    printf("The value of x is: %d\n", x);  
    printf("The value of y is:%d\n", y);  
    getch();  
}  
int y=30;
```

**Output:**

The value of x is 20

The value of y is 30

The following table shows features of different storage class:

| Storage class    | Keyword  | Default Value | Storage  |
|------------------|----------|---------------|----------|
| <b>Automatic</b> | auto     | Garbage       | RAM      |
| <b>Register</b>  | register | Garbage       | Register |
| <b>Static</b>    | static   | Zero          | RAM      |
| <b>External</b>  | extern   | Zero          | RAM      |