UNIT I

FUNCTIONS IN C

## Functions

Definition: A function is a block of code/group of statements/self contained block of statements/basic building blocks in a program that performs a particular task. It is also knownas procedure or subroutine or module, in other programming languages.

To perform any task, we can create function. A function can be called many times. It provides modularity and code reusability.

## Advantage of functions

### 1) Code Reusability

By creating functions in C, you can call it many times. So we don't need to write the same code again and again.

### 2) Code optimization

It makes the code optimized we don't need to write much code.

### 3) Easily to debug the program.

Example: Suppose, you have to check 3 numbers (781, 883 and 531) whether it is prime number or not. Without using function, you need to write the prime number logic 3 times. So, there is repetition of code. But if you use functions, you need to write the logic only once and you can reuse it several times.

## Types of Functions

There are two types of functions in C programming:

1. **Library Functions**: are the functions which are declared in the C header files such as scanf(), printf(), gets(), puts(), ceil(), floor() etc. You just need to include appropriate header files to use these functions. These are already declared and defined in C libraries.

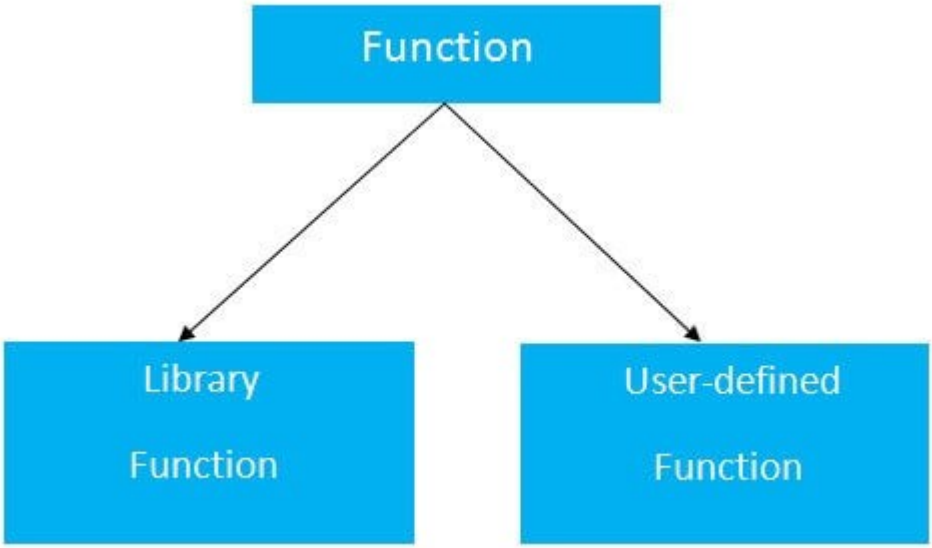
### Points to be Remembered

System defined functions are declared in header files

To use system defined functions the respective header file must be included.

**2. User-defined functions**: are the functions which are created by the C programmer, so that he/she can use it many times. It reduces complexity of a big program and optimizes the code. Depending upon the complexity and

requirement of the program, you can create as many user-defined functions as you want.




**ELEMENTS OF USER-DEFINED FUNCTIONS :**

In order to write an efficient user defined function, the programmer must familiar with the following three elements.

1 : Function Declaration. (Function Prototype).

2 : Function Call.

3 : Function Definition

**Function Declaration. (Function Prototype).**

A function declaration is the process of tells the compiler about a function name.

**Syntax**

return_type function_name(parameter/argument);

return_type function-name();

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

int add();

Note: At the time of function declaration function must be terminated with ;.

**Calling a function/function call**

When we call any function control goes to function body and execute entire code.

**Syntax** :

function-name();

function-name(parameter/argument);

return value/ variable = function-name(parameter/argument)

Ex : add(); // function without parameter/argument

add(a,b); // function with parameter/argument

c=fun(a,b); // function with parameter/argument and return values

**Defining a function.**

Defining of function is nothing but give body of function that means write logic inside function body.

**Syntax**

return_ type function-name(parameter list) // function header.

{

**declaration of variables;**

body of function; // Function body

return statement; (expression or value) //optional

}

Eg:

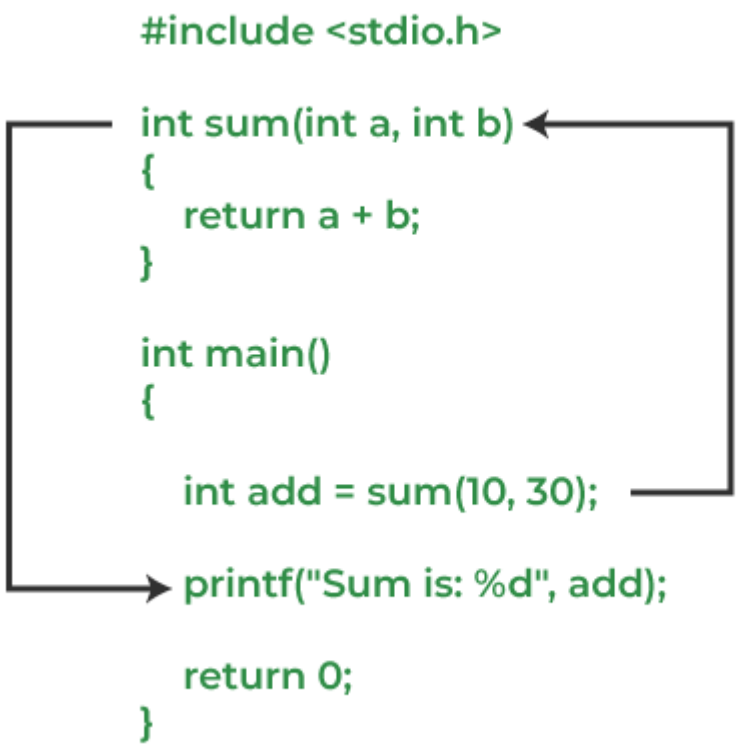
```
int add( int x, int y)
{
int z;
z = x + y;
return z;
}
```

( or )

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

**Function Call**

To Call a function parameters are passed along the function name. In the below example, the first sum function is called and 10,30 are passed to the sum function. After the function call sum of a and b is returned and control is also returned back to the main function of the program.




**1. User Defined Function**

Functions that are created by the programmer are known as User-Defined functions or **"tailor-made functions"**. User-defined functions can be improved and modified according to the need of the programmer. Whenever we write a function that is case-specific and is not defined in any header file, we need to declare and define our own functions according to the syntax.

*Advantages of User-Defined functions*

- Changeable functions can be modified as per need.
- The Code of these functions is reusable in other programs.
- These functions are easy to understand, debug and maintain.

**Example**

```
// C program to show

// user-defined functions

#include <stdio.h>

int sum(int a, int b)
```

```
{
  return a + b;
}
// Driver code
 int main()
{
   int a = 30, b = 40;
  // function call
   int res = sum(a, b);
printf("Sum is: %d", res);
  return0;
}
```

**Output**
Sum is: 70

## 2. Library Function

A library function is also referred to as a **“built-in function”**. There is a compiler package that already exists that contains these functions, each of which has a specific meaning and is included in the package. Built-in functions have the advantage of being directly usable without being defined, whereas user-defined functions must be declared and defined before being used.

**For Example:**
pow(), sqrt(), strcmp(), strcpy() etc.

*Advantages of C library functions*

- C Library functions are easy to use and optimized for better performance.
- C library functions save a lot of time i.e, function development time.
- C library functions are convenient as they always work.

**Example:**

```
// C program to implement
// the above approach
#include <math.h>
#include <stdio.h>
```

```
// Driver code
 int main()
{
   double Number;
  Number = 49;

  // Computing the square root with
  // the help of predefined C
  // library function
  double squareRoot = sqrt(Number);

  printf("The Square root of %.2lf = %.2lf",
         Number, squareRoot);
  return 0;
}
```

**Output**

The Square root of 49.00 = 7.00

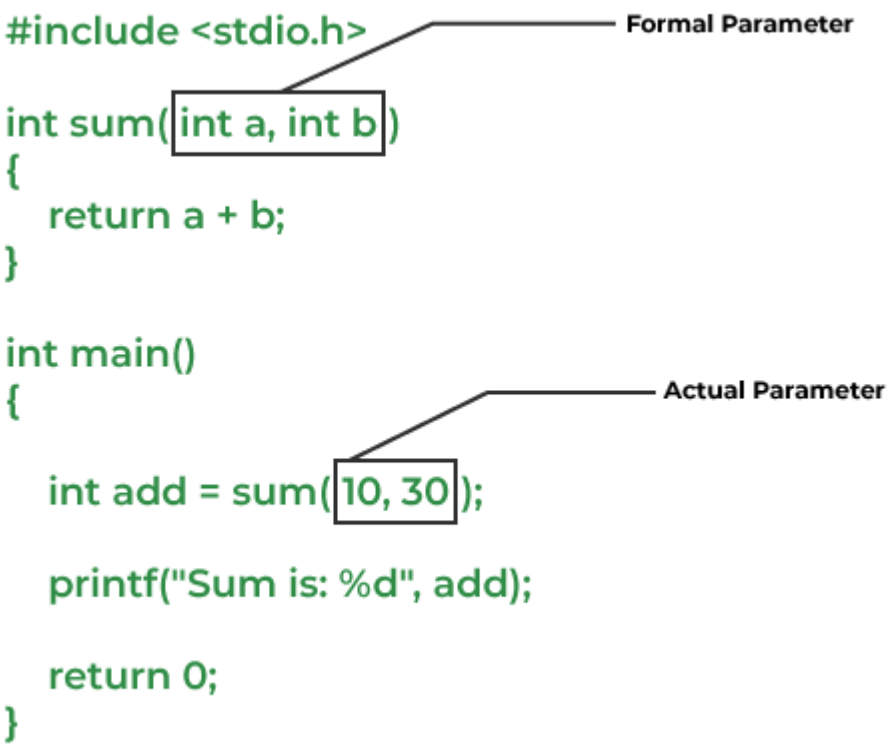
**Passing Parameters to Functions**

The value of the function which is passed when the function is being invoked is known as the **Actual parameters**. In the below program 10 and 30 are known as actual parameters.

**Formal Parameters** are the variable and the data type as mentioned in the function declaration. In the below program, a and b are known as formal parameters.

**1. Pass by Value:** Parameter passing in this method copies values from actual parameters into function formal parameters. As a result, any changes made inside the functions do not reflect in the caller's parameters.

Below is the C program to show pass-by-value:




The execution of a C program begins from the main() function.

When the compiler encounters functionName(); inside the main function, control of the program jumps to void functionName() And, the compiler starts executing the codes inside the user-defined function.

The control of the program jumps to statement next to functionName(); once all the codes inside the function definition are executed.

**Example:**

```
#include<stdio.h>
#include<conio.h>
void sum(); // declaring a function
clrscr();
int a=10,b=20, c;
void sum() // defining function
{
c=a+b;
printf("Sum: %d", c);
```

```
}
void main()
{
sum(); // calling function
}
```

**Output**

Sum:30

**Example:**

```
#include <stdio.h>
 int addNumbers(int a, int b); // function prototype
 int main()
{
int n1,n2,sum;
printf("Enters two numbers: ");
scanf("%d %d",&n1,&n2);
sum = addNumbers(n1, n2); // function call
printf("sum = %d",sum);
return 0;
}
 int addNumbers(int a,int  b) // function definition
{
int result;
result = a+b;
return result; // return statement
}
```

Return Statement

Syntax of return statement

Syntax : return; // does not return any value

or

return(exp); // the specified exp value to calling function.

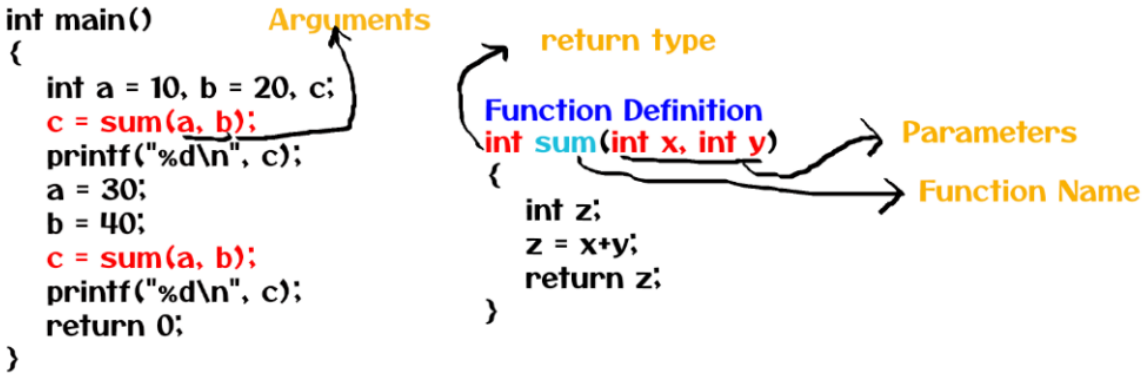
For example,

return a;

return (a+b);

The return statement terminates the execution of a function and returns a value to the calling function. The program control is transferred to the calling function after return statement.

In the above example, the value of variable result is returned to the variable sum in the main() function.




PARAMETERS :

parameters provides the data communication between the calling function and called function.

They are two types of parametes

1 : Actual parameters.

2 : Formal parameters.

1 : Actual Parameters : These are the parameters transferred from the calling function (main program) to the called function (function).

2 : Formal Parameters :These are the parameters transferred into the calling function (main program) from the called function(function).

- The parameters specified in calling function are said to be Actual Parameters.
- The parameters declared in called function are said to be Formal Parameters.
- The value of actual parameters is always copied into formal parameters.

Ex : main()

{

fun1( a , b ); //Calling function

}

fun1( x, y ) //called function

{

Where

. . . . . .

}

a, b are the Actual Parameters

x, y are the Formal Parameters

## Different aspects of function calling

A function may or may not accept any argument. It may or may not return any value. Based on these facts, There are four different aspects of function calls.

1 : Functions with no Parameters and no Return Values.

2 : Functions with no Parameters and Return Values.

3 : Functions with Parameters and no Return Values.

4 : Functions with Parameters and Return Values.

**1 : Functions with no Parameters and no Return Values :**

1 : In this category, there is no data transfer between the calling function and called function.

2 : But there is flow of control from calling function to the called function.

3 : When no parameters are there , the function cannot receive any value from the calling function.

4: When the function does not  return a value, the calling function cannot receive any value from the called function.

**Example 1:**

```
 #include<stdio.h>
#include<conio.h>
void sum();
void main()
{
sum();
getch();
}
void sum()
{
   int a,b,c;
printf("Enter the values of a and b");
scanf("%d%d",&a,&b);
c=a+b;
printf("sum=%d",c);
}
```

**Output:**

Enter the values of a and b

3  4

Sum=7

Example 2:

```
#include<stdio.h>
void printName();
void main ()
{
   printf("Welcome ");
   printName();
}
void printName()
{
   printf("C Programming");
}
```

**Output:**

Welcome C Programming

**2 : Functions with no Parameters and Return Values.**

1 :  In this category, there is no data transfer between the calling function and called function.

2 :  But there is data transfer from called function to the calling function.

3 :  When no parameters are there , the function cannot receive any values from the calling function.

4: When the function returns a value, the calling function receives one value from the called function.

**Ex ample 1:**

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

```
#include<conio.h>
   int sum();
void main()
{
int c;
clrscr();
c=sum();
printf("sum=%d",c);
getch();
}
int sum()
{
int a,b,c;
printf("enter the values of a and b");
scanf("%d%d",&a,&b);
c=a+b;
return c;
}
```

Output:

Enter the values of a and b

5  6

Sum=11

Example 2:

```
#include<stdio.h>
float square ();
void main()
{
```

```
        float area = square();
    printf("The area of the square: %f\n",area);
}
float square()
{
    float side;
    printf("Enter the length of the side  ");
    scanf("%f",&side);
    return side * side;
}
```

Output:

Enter the length of the side

2.5

The area of the square: 6.250000

**3 : Functions with Parameters and no Return Values.**

1 : In this category, there is data transfer from the calling function to the called function using parameters.

2 : But there is no data transfer from called function to the calling function.

3 : When parameters are there , the function can receive any values from the calling function.

4: When the function does not return a value, the calling function cannot receive any value from the called function.

**Ex ample 1:**

```
#include<stdio.h>
#include<conio.h>
void sum(int a,int b);
void main()
{
 int m,n;
clrscr();
```

```
printf("Enter m and n values:");
scanf("%d%d",&m,&n);
sum(m,n);
getch();
}
void sum(int a,int b)
{
int c;
c=a+b;
printf("sum=%d",c);
}
```

Output:

Enter m and n values:

7 8

Sum=15

**4 : Functions with Parameters and Return Values.**

1 : In this category, there is data transfer from the calling function to the called function using parameters.

2 : But there is data transfer from called function to the calling function.

3 : When parameters are there , the function can receive any values from the calling function.

4: When the function returns a value, the calling function receive a value from the called function.

**Example 1:**

```
 #include<stdio.h>
#include<conio.h>
int sum(int a,int b);
void main()
{
```

```
int m,n,c;
clrscr();
printf("Enter m and n values");
scanf("%d%d",&m,&n);
c=sum(m,n);
printf("sum=%d",c);
getch();
}

int sum(int a,int b)
{
int c;
c=a+b;
return c;
}
```

**Output:**

```
Enter m and n values
32  65
Sum=97
```

**Example 2:**

```
#include<stdio.h>
int even_odd(int);
void main()
{
 int n,flag=0;
  printf("\nEnter the number: ");
 scanf("%d",&n);
 flag = even_odd(n);
 if(flag == 0)
```

```
 {
    printf("\nThe number is odd");
 }
 else
 {
    printf("\nThe number is even");
 }
}
int even_odd(int n)
{
    if(n%2 == 0)
    {
        return 1;
    }
    else
    {
        return 0;

    }
   }
```

**Storage Classes**

In C language, each variable has a storage class which is used to define scope and life time of a variable.

Storage: Any variable declared in a program can be stored either in memory or registers.

Registers are small amount of storage in CPU. The data stored in registers has fast access compared to data stored in memory.

Storage class of a variable gives information about the location of the variable

in which it is stored, initial value of the variable, if storage class is not specified; scope of the variable; life of the variable.

There are four storage classes in C programming.

1 : Automatic Storage class.

2 : Register Storage class.

3 : Static Storage class.

4 : External Storage class.

**1: Automatic Storage class :** To define a variable as automatic storage class, the keyword „auto‟ is used. By defining a variable as automatic storage class, it is stored in the memory. The default value of the variable will be garbage value. Scope of the variable is within the block where it is defined and the life of the variable is until the control remains within the block.

Syntax : auto data_typevariable_name;

auto int a,b;

Example:

```
void main()
{
int detail;
or
auto int detail; //Both are same
}
```

The variables a and b are declared as integer type and auto. The keyword auto is not mandatory. Because the default storage class in C is auto.

Note: A variable declared inside a function without any storage class specification, is by default an automatic variable. Automatic variables can also be called local variables because they are local to a function.

**Ex :**

```
void function1();
void function2();
void main()
```

**OUTPUT**

10

```
{ 0
int x=100; 100
function2();
printf(“%d”,x);
}
void function1()
{
int x=10;
printf(“%d”,x);
}
void function2()
{
int x=0;
function1();
printf(“%d”,x);}
```

**2: Register Storage class** : To define a variable as register storage class, the keyword „register“ is used. If CPU cannot store the variables in CPU registers, then the variables are assumed as auto and stored in the memory. When a variable is declared as register, it is stored in the CPU registers. The default value of the variable will be garbage value. Scope of the variable within the block where it is defined and the life of the variables is until the control

remains within the block.

Register variable has faster access than normal variable. Frequently used variables are kept in

register. Only few variables can be placed inside register.

NOTE : We can't get the address of register variable

**Syntax** : register data_typevariable_name;

Ex: register int i;

**Ex :** void demo();

void main()

**OUTPUT**

20

{ 20

demo(); 20

demo();

demo();

}

void demo()

{

register int i=20;

printf(“%d\n”,i);

i++;

}

**3 : Static Storage class** : When a variable is declared as static, it is stored in the memory. The

default value of the variable will be zero. Scope of the variable is within the block where it is

defined and the life of the variable persists between different function calls. To define a variable

as static storage class, the keyword „static‟ is used. A static variable can be initialized only

once, it cannot be reinitialized.

Syntax : static data_typevariable_name;

Ex: static inti;

Ex : void demo();

void main()

OUTPUT

```
20
{ 21
demo(); 22
demo();
demo();
}
void demo()
{
static int i=20;
printf(“%d”,i);
i++;
}
```

**4 : External Storage class**: When a variable is declared as extern, it is stored in the memory.

The default value is initialized to zero. The scope of the variable is global and the life of the variable is until the program execution comes to an end. To define a variable as external storage class, the keyword „extern‟ is used. An extern variable is also called as a global variable.

Global variables remain available throughout the entire program. One important thing to remember about global variable is that their values can be changed by any function in the program.

Systax : extern data_typevariable_name;

```
extern int i;
Ex:
int number;
void main()
{
number=10;
```

```
}
fun1()
{
number=20;
}
fun2()
{
number=30;
}
```

Here the global variable number is available to all three functions.

Ex :

```
void fun1();
void fun2();
int e=20;
void main()
{
fun1();
fun2();
}
void fun1()
{
extern int e;
printf(“e number is :%d”,e);
}void fun2()
{
printf(“e number is :%d”,e);
}
```

extern keyword

The extern keyword is used before a variable to inform the compiler that this variable is declared,somewhere else. The extern declaration does not allocate storage for variables.

Problem when extern is not used

```
main()
{
a = 10; //Error:cannot find variable a
printf("%d",a);
}
```

Example Using extern in same file

```
main()
{
extern int x; //Tells compiler that it is defined somewhere else
x = 10;
printf("%d",x);
}
int x; //Global variable x
```

The extern Storage Class

The **extern** storage class is used to give a reference of a global variable that is visible to ALL the program files. When you use 'extern', the variable cannot be initialized however, it points the variable name at a storage location that has been previously defined.

When you have multiple files and you define a global variable or function, which will also be used in other files, then *extern* will be used in another file to provide the reference of defined variable or function. Just for understanding, *extern* is used to declare a global variable or function in another file.

The extern modifier is most commonly used when there are two or more files sharing the same global variables or functions as explained below.

**First File: main.c**

```
#include<stdio.h>
intcount ;
externvoidwrite_extern();

main(){
   count =5;
write_extern();
}
```

**Second File: support.c**

```
#include<stdio.h>

externint count;

voidwrite_extern(void){
printf("count is %d\n", count);
```

Here, *extern* is being used to declare *count* in the second file, where as it has its definition in the first file, main.c. Now, compile these two files as follows −

$gccmain.csupport.c

It will produce the executable program **a.out**. When this program is executed, it produces the following result −count is 5

| Storage Classes | Storage Place | Default Value | Scope | Lifetime |
|---|---|---|---|---|
| auto | RAM | Garbage Value | Local | Within function |
| extern | RAM | Zero | Global | Till the end of the main program Maybe declared anywhere in the program |
| static | RAM | Zero | Local | Till the end of the main program, Retains value between multiple functions call |
| register | Register | Garbage Value | Local | Within the function |

**Recursion**

When function is called within the same function, it is known as recursion in C. The function

which calls the same function, is known as recursive function.

A function that calls itself, and doesn't perform any task after function call, is know as tail

recursion. In tail recursion, we generally call the same function with return statement.

**Features :**

- There should be at least one if statement used to terminate recursion.
- It does not contain any looping statements.

**Advantages :**

- It is easy touse.
- It represents compact programming structures.

**Disadvantages :**

- It is slower than that of looping statements because each time function is called.

Note: while using recursion, programmers need to be careful to define an exit condition from the

function, otherwise it will go into an infinite loop. Recursive functions are very useful to solve

many mathematical problems, such as calculating the factorial of a number, generating Fibonacci

series, etc.

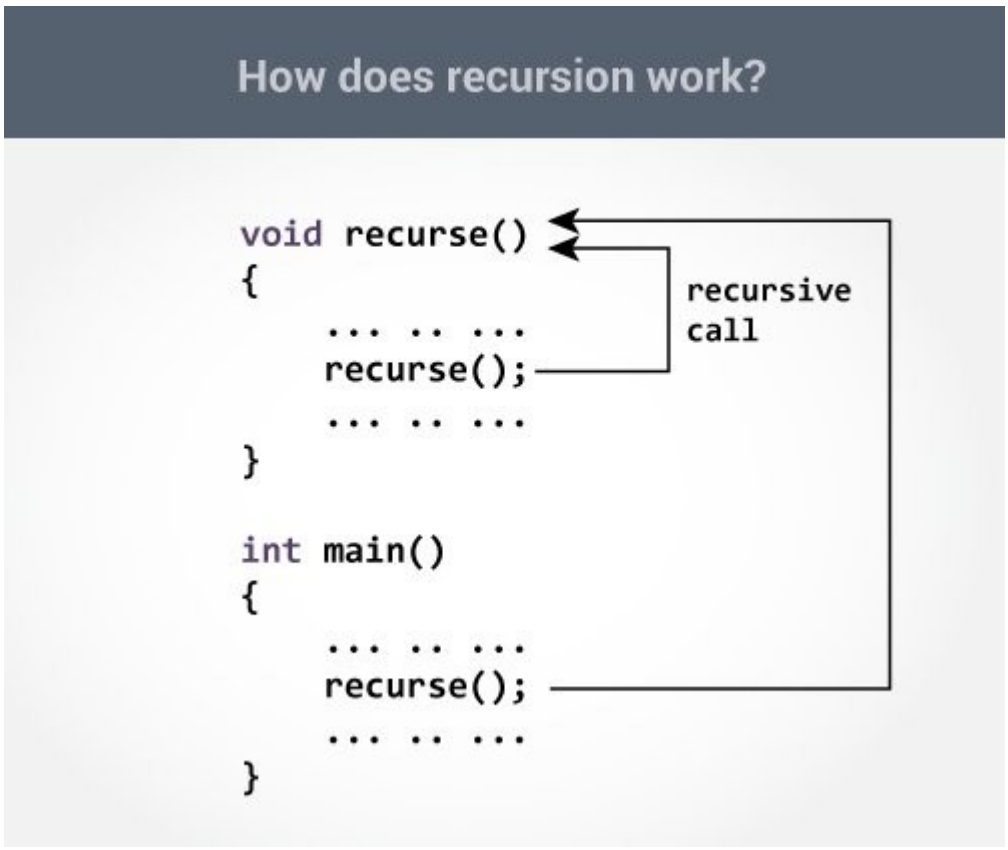
Example of recursion.

```
recursionfunction(){
recursionfunction();//calling self function
}
```

### How recursion works?

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

Int main()
{
    ... .. ...
recurse();
    ... .. ...
}
```




**Example: Sum of Natural Numbers Using Recursion**

```
#include <stdio.h>
int sum(int n);

int main() {
int number, result;

printf("Enter a positive integer: ");
scanf("%d", &number);

    result = sum(number);

printf("sum = %d", result);
return 0;
}

int sum(int n) {
if (n != 0)
// sum() function calls itself
return n + sum(n-1);
else
return n;
}
```

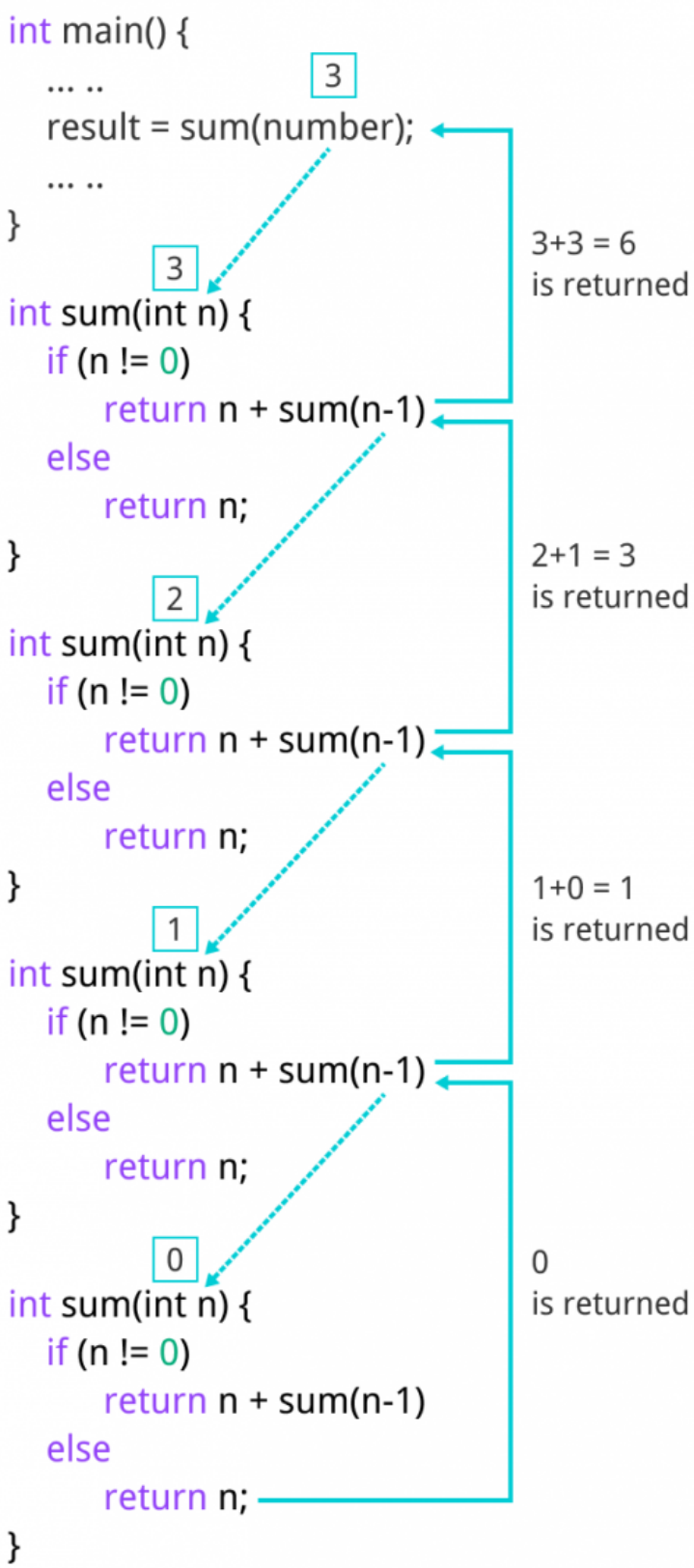
**Output**

```
Enter a positive integer:3
sum = 6
```

Initially, the `sum()` is called from the `main()` function with `number` passed as an argument.

Suppose, the value of `n` inside `sum()` is 3 initially. During the next function call, 2 is passed to the `sum()` function. This process continues until `n` is equal to 0.

When `n` is equal to 0, the `if` condition fails and the `else` part is executed returning the sum of integers ultimately to the `main()` function.


int main() {
... ..
3
result = sum(number);
... ..
}
3
int sum(int n) {
if (n != 0)
return n + sum(n-1)
else
return n;
}
2
int sum(int n) {
if (n != 0)
return n + sum(n-1)
else
return n;
}
1
int sum(int n) {
if (n != 0)
return n + sum(n-1)
else
return n;
}
0
int sum(int n) {
if (n != 0)
return n + sum(n-1)
else
return n;
}
3+3 = 6
is returned
2+1 = 3
is returned
1+0 = 1
is returned
0
is returned

Example of tail recursion in C

```
// print factorial number using tail recursion
#include<stdio.h>
#include<conio.h>
int factorial (int n)
{
if ( n< 0)
return -1; /*Wrong value*/
if (n == 0)
return 1; /*Terminating condition*/
return (n * factorial (n -1));
}
void main(){
int fact=0;
clrscr();
fact=factorial(5);
printf("\n factorial of 5 is %d",fact);
getch(); } Outputfactorial of 5 is 120
```

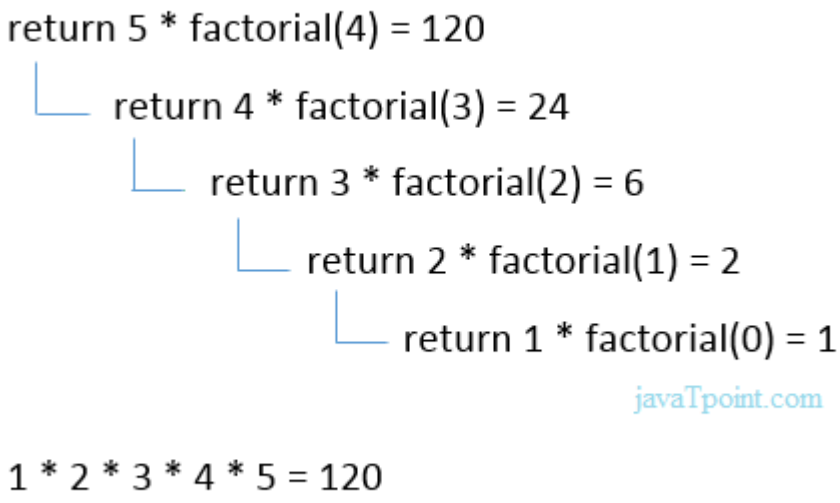



**Fig: Recursion**

## C Strings

**Declaration of String:** C does not support string as a data type. However, it allows us to represent strings as character arrays. In C, a string variable is any valid C variable name and it is always declared as an array of characters.

The general form of declaration of a string variable is :

**Syntax**: char string_name[size];

The size determines the number of characters in the string name.

Note: In declaration of string size must be required to mention otherwise it gives an error.

Ex: char str[]; // Invalid

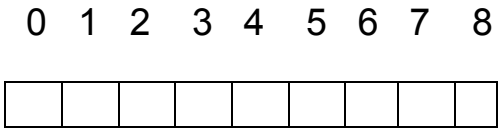
char str[0]; // Invalid

char str[-1]; // Invalid

char str[10]; // Valid

char a[9]; //Valid

Using this declaration the compiler allocates 9 memory locations for the variable a ranging from 0 to 8.




Here, the string variable a can hold maximum of 9 characters including NULL(\0) character.

Initializing Array string

**Syntax** : char string_name[size]={“string” };

Note: In Initialization of the string if the specific number of character is not initialized it then rest of all character will be initialized with NULL.

char str[5]={'5','+','A'};

str[0]; ---> 5

str[1]; ---> +

str[2]; ---> A

str[3]; ---> NULL

str[4]; ---> NULL

Note: In initialization of the string we can not initialized more than size of string elements.

Ex:

char str[2]={'5','+','A','B'}; // Invalid

**Standard String Library Functions**

The various string handling functions that are supported in C language are as show

| Function | Use |
|---|---|
| strlen | To find length of a string |
| strlwr | To convert all characters of a string to lowercase |
| strupr | To convert all characters of a string to uppercase |
| strcpy | To copy one string into another |
| strncpy | To copy first n characters of a string into another |
| strrev | To reverse a string |
| strcat | To append one string at the end of another string |
| strcmp | To compare two strings |

All these functions are defined in string.h header file.
1 : strlen(string) – String Length : This function is used to count and return the number of characters present in a string.

**Syntax** : var=strlen(string);

Ex : Progrm using strlen() function

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
char name[]="JBREC";
int len1,len2;
clrscr();
len1=strlen(name);
len2=strlen("JBRECECE");
printf("The string length of %s is: %d\n",name,len1);
printf("The string length of %s is: %d","JBRECECE",len2);
getch();
}
```

**OUTPUT :**
The string length of JBREC is : 5
The string length of JBRECECE is :8

Write a program to find the length of string

```
#include<stdio.h>
#include<conio.h>
void main()
{
char str[10];
int index;
printf("Enter the string:");
scanf("%s",str);
for(index=0;str[index]!=0;index++);
printf("The length of string is:%d",index);
getch();
}
```

**OUTPUT :**

Enter the string : subbareddy
The length of string is :10

2 : strcpy(string1,string2) – String Copy : This function is used to copy the contents of one string to another string.

**Syntax** : strcpy(string1,string2);
Where
string1 : is the destination string.
string 2: is the source string.
i.e the contents of string2 is assigned to the contents of string1.

Ex : Progrm using strcpy() function

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
char str1[]="REDDY";
char str2[10];
strcpy(str2,str1);
printf("The string1 is :%s\n",str1);
printf("The string2 is :%s\n",str2);
strcpy(str2,str1+1);
printf("The string1 is :%s\n",str1);
printf("The string2 is :%s",str2);
}
```

**OUTPUT :**

```
The string1 is : REDDY
The string2 is : REDDY
The string1 is : REDDY
The string2 is : EDDY
```

//Write a program to copy contents of one string to another string.

```
#include<stdio.h>
#include<conio.h>
void main()
{
char str1[10],str2[20];
int index;
printf("Enter the string\n");
```

```
scanf(“%s”,str1);
for(index=0;str1[index]!='\0';index++)
str2[index]=str1[index];
str2[index]='\0';
printf("String1 is :%s\n",str1);
printf("String2 is :%s\n",str2);
getch();
}
```

**OUTPUT :**

```
Enter the string : cprogramming
String1 is : cprogramming
String2 is : cprogramming
```

3 : strlwr(string) – String LowerCase : This function is used to converts upper case letters of the string in to lower case letters.

**Syntax** : strlwr(string);

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
charstr[]="JBREC";
clrscr();
strlwr(str);
printf("The lowercase is :%s\n",str);
getch();
}
```

**OUTPUT :**

```
The lowercase is : jbrec
```

Write a program to which converts given string in to lowercase.

```
#include<stdio.h>
#include<conio.h>
void main()
{
char str[10];
int index;
printf("Enter the string:");
scanf("%s",str);
```

```
for(index=0;str[index]!='\0';index++)
{
if(str[index]>='A' && str[index]<='Z')
str[index]=str[index]+32;
}
printf("After conversionis :%s",str);
getch();
}
```

**OUTPUT** :

```
 Enter the string : SUBBAREDDY
After conversion string is :subbareddy
```

4 : strupr(string) – String UpperCase : This function is used to converts lower case letters of the string in to upper case letters.

**Syntax** : strupr(string);

Program using strupr() function.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
charstr[]="jbrec";
strupr(str);
printf("UpperCase is :%s\n",str);
getch();
}
```

**OUTPUT :**

```
UpperCase is : JBREC
```

Write a program to which converts given string in to uppercase.

```
#include<stdio.h>
#include<conio.h>
void main()
{
char str[10];
int index;
printf("Enter the string:");
scanf("%s",str);
for(index=0;str[index]!='\0';index++)
```

```
{
if(str[index]>='a' && str[index]<='z')
str[index]=str[index]-32;
}
printf("After conversionis :%s",str);
getch();
}
```

**OUTPUT** :
Enter the string : subbareddy

After conversion string is :SUBBAREDDY

5 : strcmp(string1,string2) – String Comparision : This function is used to compares two strings to find out whether they are same or different. If two strings are compared character by character until the end of one of the string is reached. If the two strings are same strcmp() returns a value zero. If they are not equal, it returns the numeric difference between the first non-matching characters.

**Syntax** : strcmp(string1,string2);
Program using strcmp() function

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
char str1[]="reddy";
char str2[]="reddy";
int i,j,k;
i=strcmp(str1,str2);
j=strcmp(str1,"subba");
k=strcmp(str2,"Subba");
printf("%5d%5d%5d\n",i,j,k);
}
```

**OUTPUT :** 0 -1 32

Write a C program to find the comparision of two strings.

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

```
#include<conio.h>
#include<string.h>
void main()
{
char str1[10],str2[20];
int index,l1,l2,flag=1;
printf("Enter first string:");
scanf("%s",str1);
printf("Enter second string:");
scanf("%s",str2);
l1=strlen(str1);
l2=strlen(str2);
printf("Length of string1:%d\n",l1);
printf("Length of string2:%d\n",l2);
if(l1==l2)
{
for(index=0;str1[index]!='\0';index++)
{
if(str1[index]!=str2[index])
{
flag=0;
break;
}
}
}
else
flag=0;
if(flag==1)
printf("Strings are equal");
else
printf("Strings are not equal");
}
```

**OUTPUT** :

```
Enter the first string :jbrec
Enter the second string:jbrec
Length of string1 :5
Length of string2 :5
Strings are equal
```

6: strcat(string1,string2) – String Concatenation : This function is used to concatenate or combine, two strings together and forms a new concatenated string.

**Syntax** : strcat(sting1,string2);
Where
string1 : is the firdt string1.
string2 : is the second string2
when the above function is executed, string2 is combined with string1 and it removes the null character (\0) of string1 and places string2 from there.

Program using strcat() function.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
charstr1[10]="jbrec";
char str2[]="ece";
strcat(str1,str2);
printf("%s\n",str1);
printf("%s\n",str2);
getch();
}
```

**OUTPUT :** jbrecece
ece

7 : strrev(string) - String Reverse :This function is used to reverse a string. This function takes only one argument and return one argument.

**Syntax** : strrev(string);

Ex : Program using strrev() function

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
char str[20];
printf("Enter the string:");
scanf("%s",str);
```

```
printf("The string reversed is:%s",strrev(str));
getch();
}
```

**OUTPUT** : Enter the string :subbareddy
The string reversed is : ydderabbus