

FUNCTIONS IN C



Functions

- ❑ A 'C' program is made up of one or more functions.
- ❑ All C programs contain at least one function, called `main()` where execution starts.
- ❑ Execution always begins with `main()`, no matter where it is placed in the program. By convention, `main()` is located before all other functions.
- ❑ When program control encounters a function name, the function is **called (invoked)**.
 - ▮ Program control passes to the function.
 - ▮ The function is executed.
 - ▮ Control is passed back to the calling function.

Advantages



- ❑ Modular Programming
- ❑ Length of source program can be reduced
- ❑ Easy to locate and isolate faulty function
- ❑ **Function Can be called as many times as required**

Types of Functions



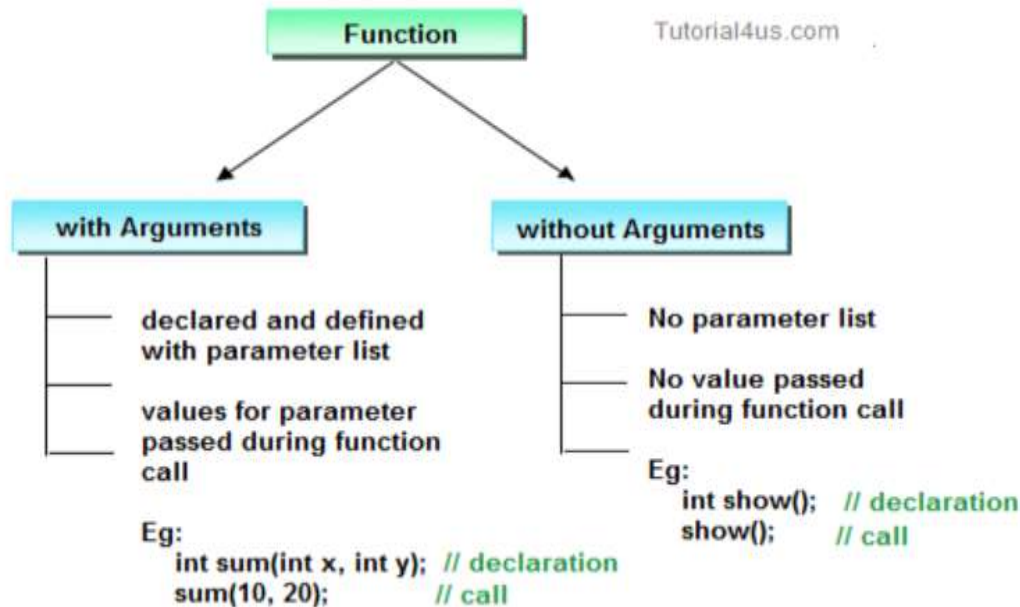
Library (Built In) Functions:

- They are written in the header files.
- To use them appropriate header files should be included

Header Files	Functions Defined
stdio.h	Printf(), scanf(), getchar(), putchar(), gets(), puts(), fopen(), fclose()
conio.h	Clrscr(), getch()
Ctype.h	Toupper(), tolower(), isalpha()
Math.h	Pow(), sqrt(), cos(), log()
Stdlib.h	Rand(), exit()
String.h	Strlen(), strcpy(), strupr()

Call by Value and Call by Reference in C++

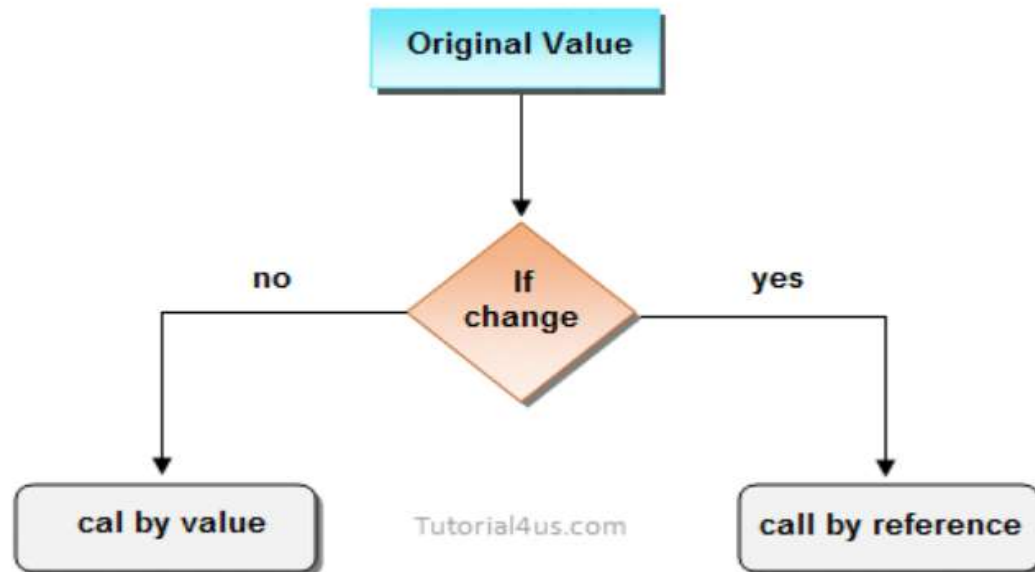
On the basis of arguments there are two types of function are available in C++ language, they are;



- With argument
- Without argument

If a function take any arguments, it must declare variables that accept the values as a arguments. These variables are called the formal parameters of the function. There are two ways to pass value or data to function in C++ language which is given below;

- call by value
- call by reference





Call by value

In call by value, **original value can not be changed** or modified. In call by value, when you passed value to the function it is locally stored by the function parameter in stack memory location. If you change the value of function parameter, it is changed for the current function only but it not change the value of variable inside the caller function such as main().



Call by reference

In call by reference, **original value is changed** or modified because we pass reference (address). Here, address of the value is passed in the function, so actual and formal arguments shares the same address space. Hence, any value changed inside the function, is reflected inside as well as outside the function.

Difference between call by value and call by reference.

	call by value	call by reference
1	This method copy original value into function as a arguments.	This method copy address of arguments into function as a arguments.
2	Changes made to the parameter inside the function have no effect on the argument.	Changes made to the parameter affect the argument. Because address is used to access the actual argument.
3	Actual and formal arguments will be created in different memory location	Actual and formal arguments will be created in same memory location

Function

Calling function()

{

//Body of calling function

Function call;

//after function call

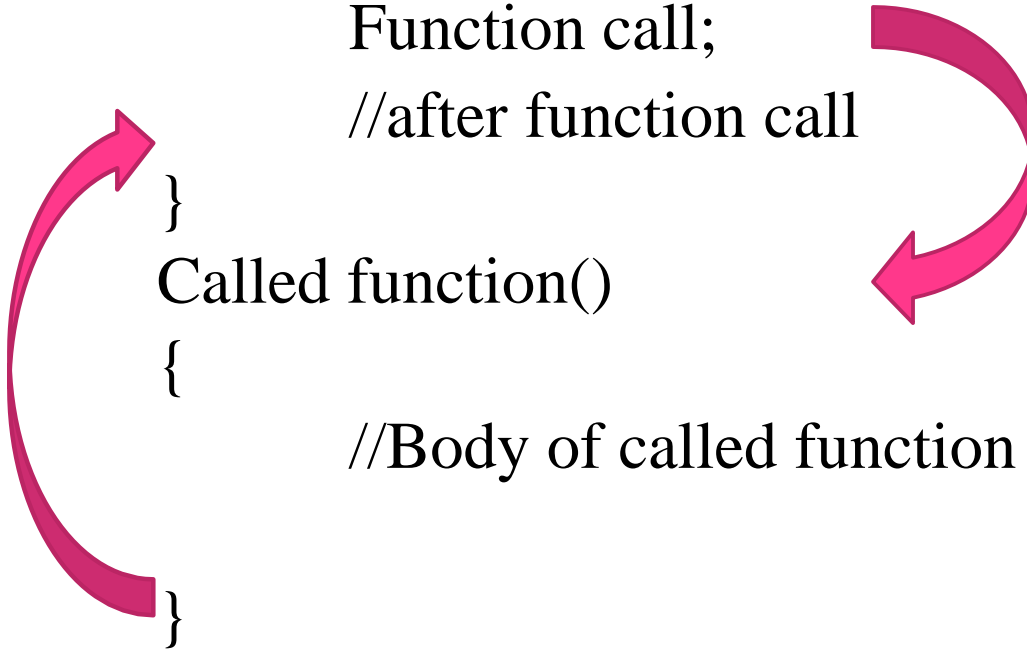
}

Called function()

{

//Body of called function

}



Parts of the function

- Function prototype/ declaration.

```
void add();
```

- Function call.

```
add();
```

- Function definition/body.

```
void add()  
{  
    int a=10,b=20,c;  
    printf("Sum=%d",a+b)  
}
```

Function prototype



- ❑ It specifies the type of value that is to be returned from the function and that is to be passed to the function.
- ❑ It is defined in the beginning before the function call is made.
- ❑ Syntax:
 - return-type name-of-function(list of arguments);
 - Example
 - ❑ Void sum(int, int);

Void add();

Int abc(float,int,float,char);

Functions can be declared before main() or inside main function but not after main().

Function Definition



- ❑ It is the independent program module.
- ❑ It is written to specify the particular task that is performed by the function.
- ❑ The first line of the function is called function declarator and rest line inside { } is called function body

Syntax:

```
Return_type Function_name(formal_parameters list)  
{  
    function body  
}
```

Sample Function Call


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

```
void main ( )
```

printf is the name of a **predefined function** in the stdio library

```
{
```

 `printf ("Hello World!\n") ;`

 this statement is
known as a
function call

 this is a string we are **passing**
as an **argument (parameter)** to
the printf function

Benefits of functions

- It break up a program into easily manageable chunks and makes programs significantly easier to understand.
- Well written functions may be reused in multiple programs. eg. The C standard library functions.
- Different programmers working on one large project can divide the workload by writing different functions.

Function definition

Function definition is a block of code which specifies the objective of the function.

General form of Function definition is:-

```
return_data_type function_name (data_type variable1,...)
{
    //body of function
}
```

Eg:-

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

```
void sum(int x,int y)
{
    int z;
    z=x+y;
    printf("sum of %d & %d = %d ",x,y,z);
}
```


Function call

The general form of function call are:

Return_data_type var=function_name(var1,var2,.....);

Or

function_name(var1,var2,.....);

Var1, var2 etc are called as actual arguments.

```
void main()
{
    int s=sum(10,15);
    printf("sum=%d",s);
}
```

```
void main()
{
    sum(10,15);
}
```

Categories of function



- ❑ Function with no arguments and no return
- ❑ Function with arguments but no return
- ❑ Function with no arguments and return
- ❑ Function with arguments and return

no argument, returns nothing.....

```
#include<stdio.h>
#include<conio.h>
void sum();
void main()
{
    sum();
}
```

```
void sum()
{
    int x, y, z;
    printf("enter two no's");
    scanf("%d%d",&x,&y);
    z=x+y;
    printf("sum of %d & %d = %d ",x,y,z);
}
```

no argument, returns nothing.....

```
#include<stdio.h>
#include<conio.h>
void a();
void b();
void c();
void main()
{
printf("starting main
function");
a();
Printf("Ending main
function");
}
```

```
void a ()
{
printf("Inside function a);
b());
}

Void b()
{
printf("Inside function b");
}

Void c()
{
printf("Inside function c" );
```

with arguments, returns nothing....

```
void sum(int a,int b);  
void main()  
{  
  int a,b,c;  
  printf("enter two no's");  
  scanf("%d%d",&a,&b);  
    sum(a,b);  
}
```

```
void sum(int x,int y)  
{  
    int z;  
    z=x+y;  
  printf("sum of %d & %d  
    = %d ",x,y,z);  
}
```

without argument and returns value..

```
int sum();  
void main()  
{  
    int c;  
    c=sum();  
    printf("sum=%d",c);  
}
```

```
int sum()  
{  
    int x, y, z;  
    printf("enter two no's");  
    scanf("%d%d",&x,&y);  
    z=x+y;  
    return z;  
}
```

with argument & returns value....

```
int sum(int a,int b);  
void main()  
{ int a,b,c;  
  printf("enter two no's");  
  scanf("%d%d",&a,&b);  
  c=sum(a,b);  
  printf("sum=%d",c);  
}
```

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

WAP to calculate factorial of a number using function

```
int factorial(int);  
void main()  
{  
    int n,f;  
    printf("Enter the no");  
    scanf("%d",&n);  
    f=factorial(n);  
    printf("\n Factorial of %d  
        = %d", n,f);  
}
```

```
int factorial(int x)  
{  
    int i,fact=1;  
    for(i=x;i>=1;i--)  
    {  
        fact=fact*i;  
    }  
    return fact;  
}
```


Menu driven program for arithmetic operations

```
#include<stdio.h>

int add(int a,int b)
{
    return(a+b);
}

int sub(int a,int b)
{
    return(a-b);
}

int mul(int a,int b)
{
    return(a*b);
}

int div(int a,int b)
{
    return(a/b);
}
```

```
void main ()
{
    int num1,num2,choice;
    do{
        printf("1 -> Addition 2->Subtraction 3->Multiplication\n");
        printf("4->Division 5->STOP\n");
        printf ("Enter your Choice:\n");
        scanf ("%d", &choice);
        printf ("Enter the Two Numbers:\n");
        scanf ("%d%d",&num1,&num2);
        switch (choice)
        {
            case 1: printf ("\n Sum is %d ", add(num1,num2));
                    break;
            case 2: printf ("\n Diif. is %d ", sub(num1,num2));
                    break;
            case 3: printf ("\n Product is %d ", mul(num1,num2));
                    break;
            case 4: printf ("\n Division is %d ", div(num1,num2));
                    break;
        }
        while(choice!=5);
    }
}
```

program for prime number using function

```
#include<stdio.h>
void prime(int n);
void main()
{
    char c;
    int n;
    printf("Enter a positive integer: ");
    scanf("%d",&n);
    prime(n);
}
```

```
void prime(int n)
{
    int i,count=0;
    for(i=1; i<=n; i++)
    {
        if(n%i==0)
            count++;
    }
    if(count==2)
        printf("\n%d is a prime
number.", n);
    else
        printf("\n%d is not a prime
number.", n);
}
```

program for Armstrong number using function

```
#include<stdio.h>
void armstrong(int n);
void main()
{
    char c;
    int n;
    printf("Enter a positive integer: ");
    scanf("%d",&n);
        armstrong(n);
}
```

```
void armstrong(int n)
{
    int sum=0,rem, temp;
    temp=n;
    while(n!=0)
    {
        rem=n%10;
        num+=rem*rem*rem;
        n/=10;
    }
    if (num==temp)
        printf("\n%d is an Armstrong
number.", n);
    else
        printf("\n%d is not an Armstrong
number.",n);
}
```

program to print Armstrong number between range

```
#include<stdio.h>
void armstrong(int n);
void main()
{
    char c;
    int n;
    For(n=100;n<=999;n++
armstrong(n);

}
```

```
void armstrong(int n)
{
    int sum=0,rem, temp;
    temp=n;
    while(n!=0)
    {
        rem=n%10;
        num+=rem*rem*rem;
        n/=10;
    }
    if (num==temp)
        printf("\n%d.", n);
}
```

Recursion

- A function that calls itself is known as recursive function and this technique is known as recursion in C programming.
- Simple eg.

```
void main()  
{  
    printf("This is recursion");  
    main();  
}
```

WAP to calculate factorial of number using recursion

```
int factorial(int);  
void main()  
{  
    int n,f;  
    printf("Enter the no");  
    scanf("%d",&n);  
    f=factorial(n);  
    printf("\n Factorial of %d  
        = %d", n,f);  
}
```

```
int factorial(int x)  
{  
    if(x==1 || x==0)  
        return 1;  
    else  
        return x*factorial(x-1);  
}
```

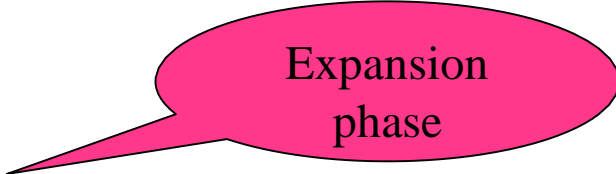
Tracing Recursive Functions

- Executing recursive algorithms goes through two phases:

Expansion in which the recursive step is applied until hitting the base step

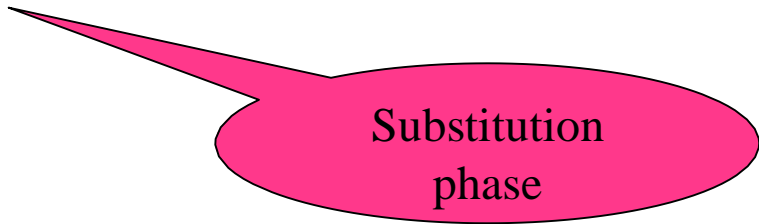
- ▮ “Substitution” in which the solution is constructed backwards starting with the base step

$$\begin{aligned}\text{factorial}(4) &= 4 * \text{factorial}(3) \\ &= 4 * (3 * \text{factorial}(2)) \\ &= 4 * (3 * (2 * \text{factorial}(1))) \\ &= 4 * (3 * (2 * (1)))\end{aligned}$$



Expansion
phase

$$\begin{aligned}&= 4 * (3 * (2 * (1))) \\ &= 4 * (3 * (2 * 1)) \\ &= 4 * (3 * 2) \\ &= 4 * 6 \\ &= 24\end{aligned}$$



Substitution
phase

Example 1: Recursive Factorial

- The following shows the recursive and iterative versions of the factorial function:

Recursive version

```
int factorial (int n)
{
    if (n == 0)
        return 1;
    else
        return n * factorial (n-1);
}
```

Iterative version

```
int factorial (int n)
{
    int i, product=1;
    for (i=n; i>1; --i)
        product=product * i;

    return product;
}
```



Recursive Call

Example 2: Multiplication ...


```
#include <stdio.h>
#include <conio.h>
int multiply(int m, int n);

int main(void) {
    int num1, num2;
    clrscr();
    printf("Enter two integer numbers to multiply: ");
    scanf("%d%d", &num1, &num2);

    printf("%d x %d = %d\n", num1, num2, multiply(num1, num2));
    getch();
}

int multiply(int m, int n) {
    if (n == 1)
        return m;    /* simple case */
    else
        return m + multiply(m, n - 1); /* recursive step */
}
```

Example 2: Multiplication ...


$$\begin{aligned}\text{multiply}(5,4) &= 5 + \text{multiply}(5, 3) \\ &= 5 + (5 + \text{multiply}(5, 2)) \\ &= 5 + (5 + (5 + \text{multiply}(5, 1)))\end{aligned}$$

Expansion
phase

$$\begin{aligned}&= 5 + (5 + (5 + 5)) \\ &= 5 + (5 + 10) \\ &= 5 + 15 \\ &= 20\end{aligned}$$

Substitution phase

Example 3: Power function ...

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

double pow(double x, int n);

void main(void) {
    double x;
    int n;
    clrscr();
    printf("Enter double x and integer n to find pow(x,n): ");
    scanf("%lf%d", &x, &n);

    printf("pow(%f, %d) = %f\n", x, n, pow(x, n));
    getch();
}

double pow(double x, int n) {
    if (n == 0)
        return 1;    /* simple case */
    else
        return x * pow(x, n - 1); /* recursive step */
}
```

C program to find sum of first n natural numbers using recursion

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

```
int sum(int n);
```

```
int main()
```

```
{
```

```
int num,add;
```

```
printf("Enter a positive integer:\n");
```

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

```
add=sum(num);
```

```
printf("sum=%d",add);
```

```
}
```

```
int sum(int n)
```

```
{
```

```
if(n==0)
```

```
return n;
```

```
else
```

```
return n+sum(n-1);
```

```
}
```

Fibonacci series using recursion

```
#include<stdio.h>
#include<conio.h>
void fibonacci(int);
void main(){
    int k,n;
    int i=0,j=1,f;
    printf("Enter the range of the
    Fibonacci series: ");
    scanf("%d",&n);
    printf("Fibonacci Series: ");
    printf("%d %d ",0,1);
    fibonacci(n);
    getch();
}
```

```
void fibonacci(int n)
{
    static int first=0,second=1,sum;
    if(n>0){
        sum = first + second;
        first = second;
        second = sum;
        printf("%ld ",sum);
        fibonacci(n-1);
    }
}
```

Sum of digits using recursion(1)

```
include <stdio.h>
int sum (int a);
void main()
{
    int num, result;
    printf("Enter the
    number: ");
    scanf("%d", &num);
    result = sum(num);
    printf("Sum of digits in
    %d is %d\n", num,
    result);
    getch();
}

int sum (int num)
{
    if (num != 0)
        return (num % 10 + sum (num / 10));
    else
        return 0;
}
```

```

void prime(int n);
void armstrong(int n);
void main()
{
    char c;
    int n;
    printf("Enter a positive integer: ");
    scanf("%d",&n);
    printf("Enter P to check prime and A to
check Armstrong number: ");
    c=getchar();
    if (c=='p' || c=='P')
    {
        prime(n);
    }
    if (c=='a' || c=='A')
    {
        armstrong(n);
    }
}

```

```

void prime(int n)
{
    int i,count=0;
    for(i=1; i<=n; i++)
    {
        if(n%i==0)
            count++;
    }
    if(count==2)
        printf("\n%d is a prime
number.", n);
    else
        printf("\n%d is not a prime
number.", n);
}

void armstrong(int n)
{
    int sum=0,rem, temp;
    temp=n;
    while(n!=0)
    {
        rem=n%10;
        num+=rem*rem*rem;
        n/=10;
    }
    if (num==temp)
        printf("\n%d is an Armstrong
number.", n);
    else

```



What is Recursion in C?

Recursion, in general, can be defined as the repetition of a process in a similar way until the specific condition reaches. In C Programming, if a function calls itself from inside, the same function is called recursion. The function which calls itself is called a recursive function, and the function call is termed a recursive call. The recursion is similar to iteration but more complex to understand. If the problem can be solved by recursion, that means it can be solved by iteration. Problems like sorting, traversal, and searching can be solved using recursion. While using recursion, make sure that it has a base (exit) condition; otherwise, the program will go into the infinite loop.

The recursion contains two cases in its program body.

Base case: When you write a recursive method or function, it keeps calling itself, so the base case is a specific condition in the function. When it is met, it terminates the recursion. It is used to make sure that the program will terminate. Otherwise, it goes into an infinite loop.

Recursive case: The part of code inside the recursive function executed repeatedly while calling the recursive function is known as the recursive case.

Basic Syntax of Recursion

The syntax for recursion is :

```
void recursive_fun() //recursive function
{
    Base_case; // Stopping Condition

    recursive_fun(); //recursive call
}

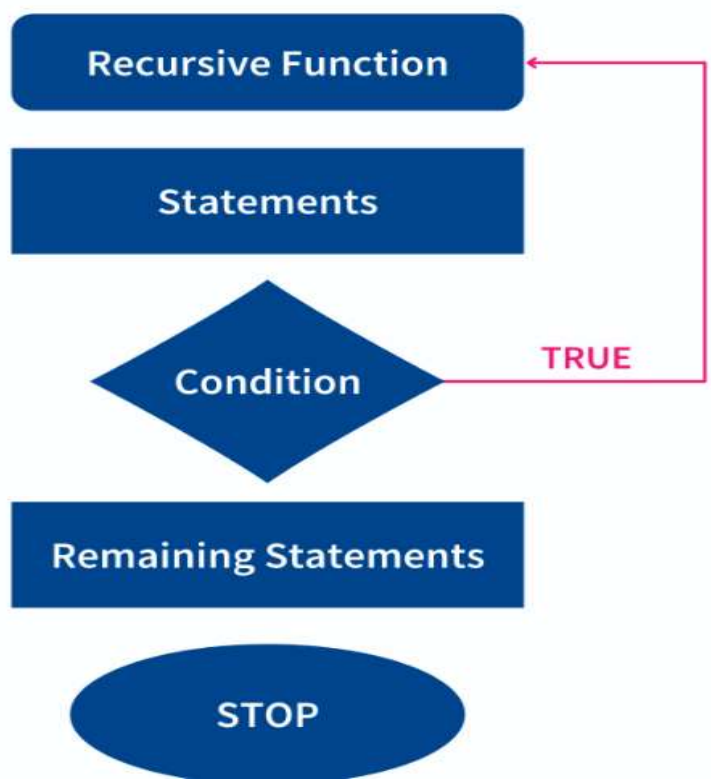
int main()
{

    recursive_fun(); //function call
}
```



Flowchart of Recursion

In the following image, there is a recursive function inside which there is a recursive call that calls the recursive function until the condition of the problem is true. If the condition gets satisfied, then the condition is false, and the program control goes for the remaining statements and stops the program.



Difference between Recursion and Iteration

Property	Recursion	Iteration
Definition	Function calls itself.	A set of instructions repeatedly executed.
Application	For functions.	For loops.
Termination	Through base case, where there will be no function call.	When the termination condition for the iterator ceases to be satisfied.
Usage	Used when code size needs to be small, and time complexity is not an issue.	Used when time complexity needs to be balanced against an expanded code size.
Code Size	Smaller code size	Larger Code Size.
Time Complexity	Very high (generally exponential) time complexity.	Relatively lower time complexity (generally polynomial-logarithmic).



Thank you