

Introduction

Some computer programming languages allow a module or function to call itself. This technique is known as recursion. In a recursive function, function calls itself. In this, to solve the problems we break the programs into smaller and smaller sub-programs.

What is Recursion?

The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called as recursive function. Using recursive algorithm, certain problems can be solved quite easily. Examples of such problems are Towers of Hanoi (TOH), Inorder/Preorder/Postorder Tree Traversals, DFS of Graph, etc.

For Example:

$$4! = 4 \times 3 \times 2 \times 1 = 24$$

This problem of finding factorial of 4 is been broken down into a sub-problem of multiplying the factorial of 3 with 4

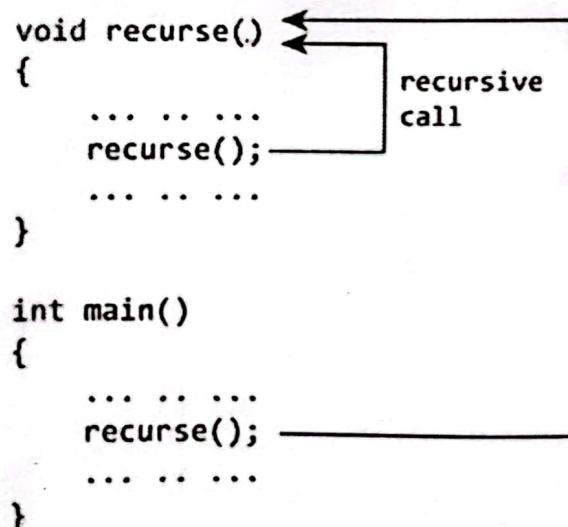
$$4! = 4 \times 3$$

Or more generally,

$$n! = n \times (n-1)!$$

Without this, the recursion will not end and continue.

How does recursion work?



The recursion continues until some condition is met to prevent it.

To prevent infinite recursion, if...else statement (or similar approach) can be used where one branch makes the recursive call and other doesn't.

Recursive functions:

Many mathematical functions can be defined recursively:

1. factorial
2. Fibonacci
3. Euclid's GCD (greatest common denominator)

What is the difference between direct and indirect recursion?

A function fun is called direct recursive if it calls the same function fun. A function fun is called indirect recursive if it calls another function say fun_new and fun_new calls fun directly or indirectly. Difference between direct and indirect recursion has been illustrated below:

// An example of direct recursion

```
void directRecFun()
{
    // Some code....

    directRecFun();

    // Some code...
}
```

// An example of indirect recursion

```
void indirectRecFun1()
{
    // Some code...

    indirectRecFun2();
```

```

    // Some code...
}
void indirectRecFun2()
{
    // Some code...

    indirectRecFun1();

    // Some code...
}

```

Properties:

A recursive function can go infinite like a loop. To avoid infinite running of recursive function, there are two properties that a recursive function must have –

Base criteria – There must be at least one base criteria or condition, such that, when this condition is met the function stops calling itself recursively.

Progressive approach – The recursive calls should progress in such a way that each time a recursive call is made it comes closer to the base criteria.

What is base condition in recursion?

In recursive program, the solution to base case is provided and solution of bigger problem is expressed in terms of smaller problems.

```

int fact (int n)
{
    if (n <= 1) // base case
        return 1;
    else

```

```
return n*fact (n-1);
```

```
}
```

In the above example, base case for $n \leq 1$ is defined and larger value of number can be solved by converting to smaller one till base case is reached.

Differences Between Recursion and Iteration

Both recursion and iteration are used for executing some instructions repeatedly until some condition is true. Recursion is when a method in a program repeatedly calls itself whereas, iteration is when a set of instructions in a program are repeatedly executed. A same problem can be solved with recursion as well as iteration but still there are several differences in their working and performance that I have mentioned below.

Recursion	Iteration
- Recursion refers to a situation where a function calls itself again and again until some base condition is not reached. - A recursive method contains set of instructions, statement calling itself, and a termination condition. - It is comparatively slower because before each function call the current state of function is stored in stack. After the return statement the previous function state is again restored from stack. - Memory usage is more as stack is used to store the current function state. - Size of code is comparatively smaller in recursion.	- Iteration refers to a situation where some statements are executed again and again using loops until some condition is true. - iteration statements contain initialization, increment, condition, set of instruction within a loop and a control variable. - Its execution is faster because it doesn't use stack. - Memory usage is less as it doesn't use stack. - Iteration makes the code size bigger.
Recursion Example <pre>#include <stdio.h> int factorial (int n) { if (n == 0) return 1; else return n * factorial(n-1); } int main () { printf ("Factorial for 5 is %d", factorial (5)); return 0; }</pre>	Iteration Example <pre>#include <stdio.h> int main () { int i, n = 5, fac = 1; for (i = 1; i <= n; ++i) fac = fac * i; printf ("Factorial for 5 is %d", fac); return 0; }</pre>

Tail Recursion:

Tail recursion refers to recursive call at last line. The tail recursive functions considered better as the recursive call is at the last statement so there is nothing left to do in the current function. It saves the current function's stack frame is of no use.

Tail recursion can be eliminated by changing the recursive call to a goto preceded by a set of assignments per function call. This simulates the recursive call because nothing needs to be saved after the recursive call finishes. We can just goto the top of the function with the values that would have been used in a recursive call.

Recursive Function TOH with Tail Recursion

```
void TOH(int n,char x,char y,char z)
{
    if(n>0)
    {
        TOH(n-1,x,z,y);
        printf("\n%c-> %c",x,y);
        TOH(n-1,z,y,x);    //tail recursion
    }
}
```

Without Tail Recursion

```
void TOH(int n,char x,char y,char z)
{
    char temp;
    label:
    if(n>0)
    {
        TOH(n-1,x,z,y);
        printf("\n%c-> %c",x,y);
        temp=x;
        x=z;
```

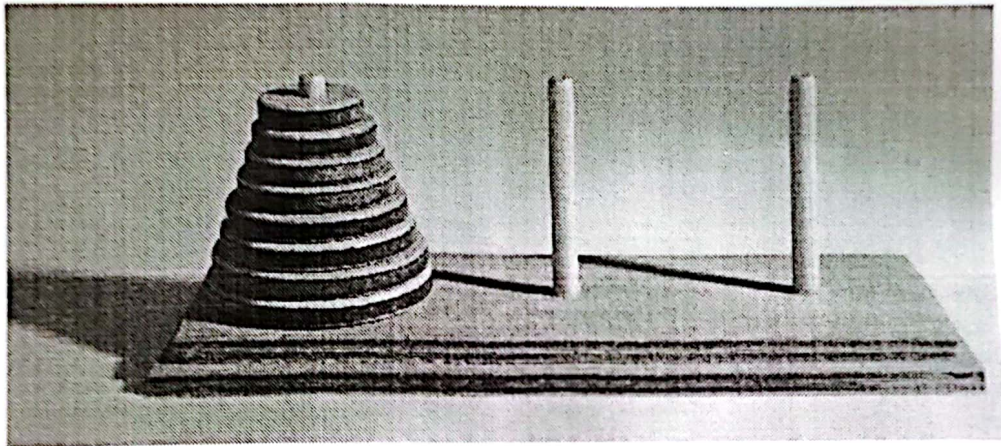
```

    z=temp;
    n=n-1;
    goto label;
}
}

```

Tower of Hanoi (TOH):

The Tower of Hanoi (also called the Tower of Brahma or Lucas' Tower and sometimes pluralized) is a mathematical game or puzzle. It consists of three rods, and a number of disks of different sizes which can slide onto any rod. The puzzle starts with the disks in a neat stack in ascending order of size on one rod, the smallest at the top, thus making a conical shape as shown in below image.



The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:

1. Only one disk can be moved at a time.
2. Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
3. No disk may be placed on top of a smaller disk.

With three disks, the puzzle can be solved in seven moves. The minimum number of moves required to solve a Tower of Hanoi puzzle is $2^n - 1$, where n is the number of disks.

Algorithm(TOH):

A recursive algorithm for Tower of Hanoi can be driven as follows –

START

Procedure Hanoi(disk, source, dest, aux)

IF disk == 1, THEN

 move disk from source to dest

ELSE

 Hanoi(disk - 1, source, aux, dest) // Step 1

 move disk from source to dest // Step 2

 Hanoi (disk - 1, aux, dest, source) // Step 3

END IF

END Procedure

STOP

Example_1: To calculate the 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 num)
{
    if (num!=0)
```

```

        return num + sum(num-1); // sum () function calls itself
    else
        return num;
}

```

Output

Enter a positive integer:3

sum = 6

Example_2: To calculate the factorial value using recursive function.

```

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

void main()
{
    int fac,n;
    int factorial(int);
    clrscr();

    printf("Enter any number:");
    scanf("%d",&n);

    fac=factorial(n);
    printf("Factorial=%d",fac);
    getch();
}

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

    return f;
}

```

Output

Enter any number 0

Factorial value = 1

Example_3: To calculate the GCD of given number using recursive function.

```
#include <stdio.h>

/* Function declaration */

int gcd(int a, int b);

int main()
{
    int num1, num2, hcf;

    /* Input two numbers from user */
    printf("Enter any two numbers to find GCD: ");
    scanf("%d%d", &num1, &num2);
    hcf = gcd(num1, num2);
    printf("GCD of %d and %d = %d", num1, num2, hcf);
    return 0;
}

/* Recursive approach of euclidean algorithm to find GCD of two numbers */
int gcd(int a, int b)
{
    if(b == 0)
        return a;
    else
        return gcd(b, a%b);
}
```

Output:

Enter any two numbers to find GCD: 12

30

GCD of 12 and 30 = 6

Example_4: To calculate the Fibonacci Sequence using recursive function.

```
#include<stdio.h>
int Fibonacci(int);
int main()
{
    int n, i = 0, c;
    scanf("%d",&n);
    printf("Fibonacci series\n");
    for ( c = 1 ; c <= n ; c++ )
    {
        printf("%d\n", Fibonacci(i));
        i++;
    }
    return 0;
}

int Fibonacci(int n)
{
    if ( n == 0 )
        return 0;
    else if ( n == 1 )
        return 1;
    else
        return ( Fibonacci(n-1) + Fibonacci(n-2) );
}
```

Output:

fibonacci series

0
1
1
2
3
5
8
13
21
34

Example_5: To calculate the Tower of Hanoi using recursive function.

```
#include <stdio.h>
#include <stdlib.h>
void hanoirecursion(int num,char ndl1, char ndl2, char ndl3)
{
    if(num == 1)
    {
        printf("Move top disk from needle %c to needle %c\n",ndl1,ndl2);
        return;
    }

    hanoirecursion(num-1,ndl1,ndl3,ndl2);
    printf("Move top disk from needle %c to needle %c\n",ndl1,ndl2);
    hanoirecursion(num-1,ndl3,ndl2,ndl1);
}

int main()
{
    int no;
    printf("Enter the Number of Disks to be Transferred:\n\n");
```

```

scanf("%d",&no);
if(no<1)
{
    printf("\n There is nothing to move \n");
}
else
{
    printf("\n Recursive:\n");
    hanoi recursion(no,'A','B','C');
}
getch();
return 0;
}

```

Output:

```

Enter the Number of Disks to be Transferred:
3
Recursive:
Move top disk from needle A to needle B
Move top disk from needle A to needle C
Move top disk from needle B to needle C
Move top disk from needle A to needle B
Move top disk from needle C to needle A
Move top disk from needle C to needle B
Move top disk from needle A to needle B

```

Applications and Efficiency of Recursion:

The application of recursion is vital in computer science,

- This is the backbone of AI.
- The NP problem can't be solved in general, but that can only be solved using recursion upto a certain extent (*not completely*) by limiting depth of recursion.
- The most important data structure '*Tree*' doesn't exist without recursion we can solve that in iterative way also but that will be a very tough task.
- Many of the well-known sorting algo (Quick sort, Merge sort, etc) uses recursion.
- All the puzzle games (Chess, Candy crush, etc) broadly uses recursion.
- The uses of recursion are uncountable, now a day because it is the backbone of searching, which is most important thing.

***** THE END *****