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.

Recursion must have

- Base or Anchor Step (for stopping the repetition)
- Recursive Step (for repeating the process)

What is base condition in recursion?

In the recursive program, the solution to the base case is provided and the solution of the 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 < = 1 is defined and larger value of number can be solved by converting to smaller one till base case is reached.

Types of Recursions : Recursion are mainly of **two types**

- **Direct Recursion : a function calls itself from within itself**
- **Indirect Recursion : more than one function call one another mutually**

Direct Recursion: These can be further categorized into **four types**

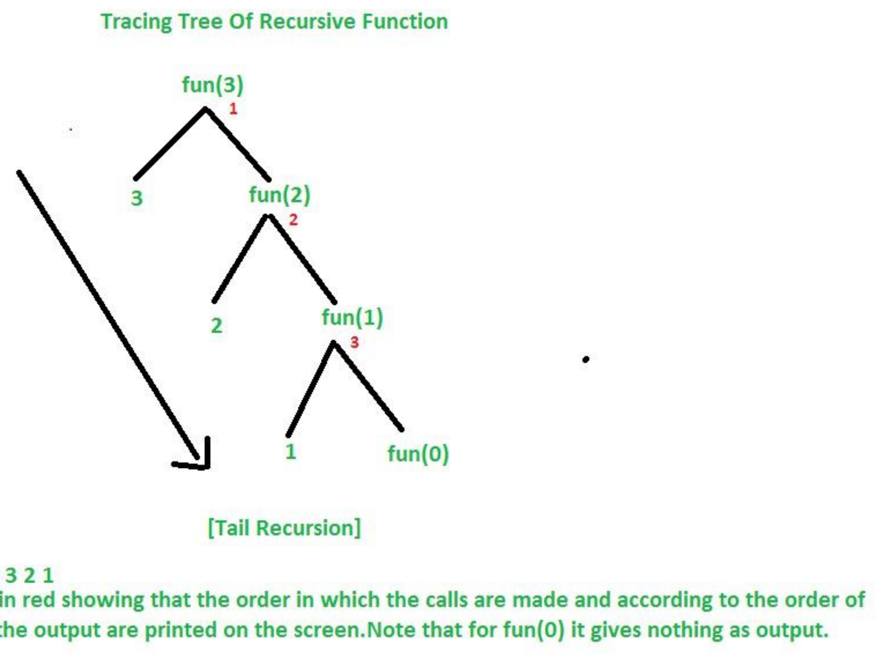
- **Tail Recursion**: If a recursive function calling itself and that recursive call is the last statement in the function then it's known as **Tail**
- **Example:**

```
#include <stdio.h>

// Recursion function
void fun(int n)
```

```
{
    if (n > 0) {
        printf("%d ", n);

        // Last statement in the function
        fun(n - 1);
    }
}

// Driver Code
int main()
{
    int x = 3;
    fun(x);
    return 0;
}
```

Output:

```
3 2 1
```

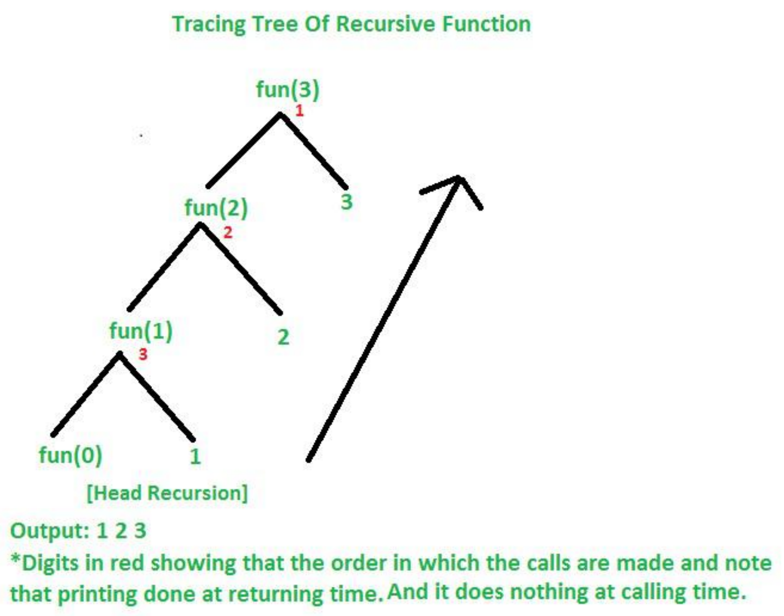

Tracing Tree Of Recursive Function
fun(3)
1
3
fun(2)
2
2
fun(1)
3
1
fun(0)
[Tail Recursion]
Output: 3 2 1
*Digits in red showing that the order in which the calls are made and according to the order of calling the output are printed on the screen.Note that for fun(0) it gives nothing as output.

- ***Head Recursion**: If a recursive function calling itself and that recursive call is the first statement in the function then it's known as **Head***
- ***Example:***

```
#include <stdio.h>

// Recursive function
void fun(int n)
{
    if (n > 0) {
        // First statement in the function
        fun(n - 1);

        printf("%d ", n);
    }
}

// Driver code
int main()
{
    int x = 3;
    fun(x);
    return 0;
}
```

Output:

```
1 2 3
```



Tracing Tree Of Recursive Function
fun(3)
1
fun(2)
3
2
fun(1)
2
3
fun(0)
1
[Head Recursion]
Output: 1 2 3
*Digits in red showing that the order in which the calls are made and note that printing done at returning time. And it does nothing at calling time.

- ***Tree Recursion***: *To understand **Tree Recursion** let's first understand **Linear Recursion**. If a recursive function calling itself for one time then it's known as **Linear Recursion**. Otherwise if a recursive function calling itself for more than one time then it's known as **Tree Recursion**.*

- ***Example:***

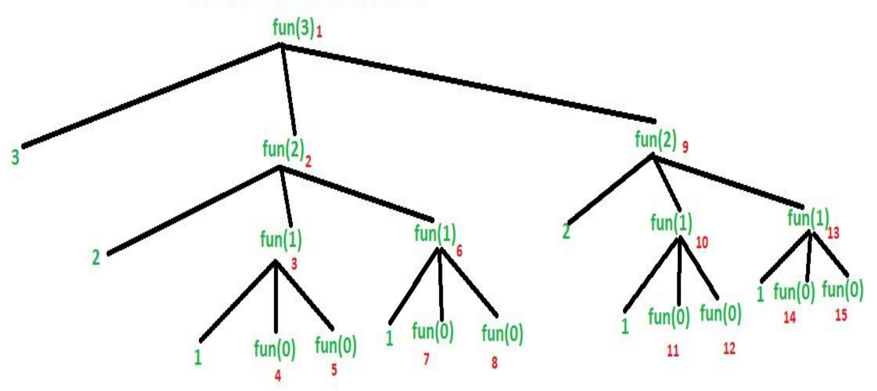
```
// C program to show Tree Recursion

#include <stdio.h>

// Recursive function
void fun(int n)
```

```
{
    if (n > 0) {
        printf("%d ", n);

        // Calling once
        fun(n - 1);

        // Calling twice
        fun(n - 1);
    }
}

// Driver code
int main()
{
    fun(3);
    return 0;
}
```

Output:

```
3 2 1 1 2 1 1
```

Tracing Tree Of Recursive Function

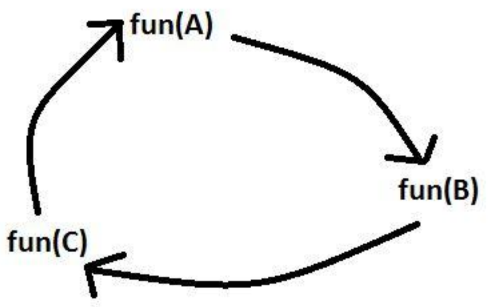


[Tree Recursion]

Output: 3 2 1 1 2 1 1
*Digits in red showing that the order in which the calls are made and according to the order of calling the output are printed on the screen.Note that for fun(0) it gives nothing as output.

- ***Nested Recursion***: *In this recursion, a recursive function will pass the parameter as a recursive call.That means "**recursion inside recursion**". Let see the example to understand this recursion.*

- ***Indirect Recursion***: *In this recursion, there may be more than one functions and they are calling one another in a circular manner. From the above diagram* fun(A) is calling for fun(B), fun(B) is calling for fun(C) and fun(C) is calling for fun(A) and thus it makes a cycle.

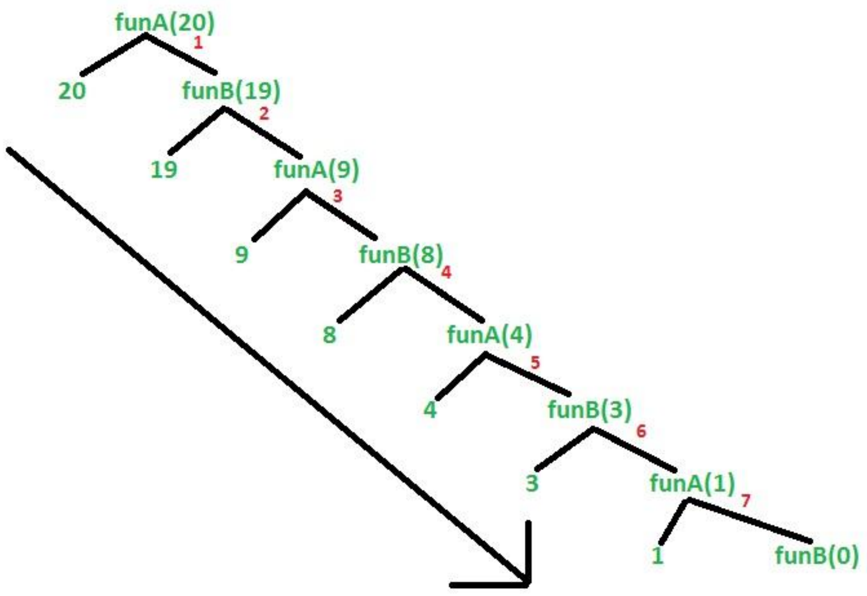


```c
#include <stdio.h>

void funB(int n);

void funA(int n)
{
    if (n > 0) {
        printf("%d ", n);

        // Fun(A) is calling fun(B)
        funB(n - 1);
    }
}

void funB(int n)
{
    if (n > 1) {
        printf("%d ", n);

        // Fun(B) is calling fun(A)
        funA(n / 2);
    }
}

// Driver code
int main()
{
    funA(20);
    return 0;
}
```

Output:

```
20 19 9 8 4 3 1
```

Tracing Tree Of Recursive Function



[Indirect recursion]

Output: 20 19 9 8 4 3 1

*Digits in red showing that the order in which the calls are made and according to the order of calling the output are printed on the screen.Note that for fun(0) it gives nothing as output.

Recurrence

- Performance of recursive algorithms typically specified with **recurrence equations**
- Recurrence Equations also known as Recurrence and Recurrence Relations
- Recurrence equations require special techniques for solving
- The solution of the recurrence can be described using four different methods
  - Substitution Method
  - Iteration Method
  - Recursion Tree Method
  - Master Method