Recursion

Types of recursion

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.

Types of Recursions:
Recursion are mainly of **two types** depending on whether **a function calls itself from within itself** or **more than one function call one another mutually.** The first one is called **direct recursion** and another one is called **indirect recursion**. Thus, the two types of recursion are:

1. **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 Recursion.**

```
void fun(int n)

{    if (n > 0) {

         printf("%d ", n);

      fun(n - 1);

 }   }
```

- **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 Recursion.**

```
void fun(int n)

{    if (n > 0) {

         fun(n - 1);

         printf("%d ", n);

  }  }
```

- **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** (also called **multiple recursion)**.

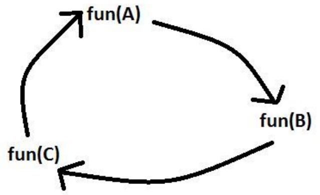
```
void fun(int n)

{    if (n > 0) {

        printf("%d ", n);

        fun(n - 1);

        fun(n - 1);

  }   }
```

- **Nested Recursion**: In this recursion, a recursive function will pass the parameter as a recursive call.That means **“recursion inside recursion”.**

```
int fun(int n)

{    if (n > 100)

        return n - 10;

        return fun(fun(n + 11));

}
```

2. **<u>Indirect Recursion:</u>**

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.

```
void funB(int n);
 void funA(int n)
{    if (n > 0) {
        printf("%d ", n);
        funB(n - 1);
 } }
 void funB(int n)
{    if (n > 1) {
        printf("%d ", n);
        funA(n / 2);
    }
}
```

Differences between recursion and iteration

The **Recursion** and **Iteration** both repeatedly execute the set of instructions. **Recursion** is when a statement in a function **calls itself repeatedly**. The **iteration** is when a loop **repeatedly executes until the controlling condition becomes false**. The primary difference between recursion and iteration is that **recursion** is a process, always applied to a function and **iteration** is applied to the **set of instructions** which we want to get **repeatedly executed**.

Recursion

- Recursion uses **selection structure**.
- **Infinite recursion** occurs if the recursion step does not reduce the problem in a manner that converges on some condition (**base case**) and Infinite recursion can crash the system.
- Recursion terminates when a **base case** is recognized.
- Recursion is usually **slower than iteration** due to the overhead of maintaining the stack.
- Recursion uses **more memory than iteration**.
- Recursion makes the **code smaller**.

Iteration

- Iteration uses **repetition structure**.
- An infinite loop occurs with iteration if the loop condition test never becomes false and Infinite looping uses CPU cycles repeatedly.
- An iteration **terminates** when the **loop condition fails**.
- An iteration does not use the **stack** so it's **faster than recursion**.
- Iteration consumes **less memory.**
- Iteration makes the **code longer**.