Recursion Introduction

What is Recursion?

The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called a recursive function. Using a 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. A recursive function solves a particular problem by calling a copy of itself and solving smaller subproblems of the original problems. Many more recursive calls can be generated as and when required. It is essential to know that we should provide a certain case in order to terminate this recursion process. So we can say that every time the function calls itself with a simpler version of the original problem.

Need of Recursion

Recursion is an amazing technique with the help of which we can reduce the length of our code and make it easier to read and write. It has certain advantages over the iteration technique which will be discussed later. A task that can be defined with its similar subtask, recursion is one of the best solutions for it. For example; The Factorial of a number.

Properties of Recursion:

- Performing the same operations multiple times with different inputs.
- In every step, we try smaller inputs to make the problem smaller.
- Base condition is needed to stop the recursion otherwise infinite loop will occur.

A Mathematical Interpretation

Let us consider a problem that a programmer has to determine the sum of first n natural numbers, there are several ways of doing that but the simplest approach is simply to add the numbers starting from 1 to n. So the function simply looks like this,

> ***approach(1) – Simply adding one by one***
>
> ***f(n) = 1 + 2 + 3 +……..+ n***

but there is another mathematical approach of representing this,

> ***approach(2) – Recursive adding***
>
> ***f(n) = 1 n=1***
>
> ***f(n) = n + f(n-1) n>1***

There is a simple difference between the approach (1) and approach(2) and that is in **approach(2)** the function " **f( )** " itself is being called inside the function, so this phenomenon is named recursion, and the function containing recursion is called recursive function, at the end, this is a great tool in the hand of the programmers to code some problems in a lot

« Prev
Next »

easier and efficient way.

How are recursive functions stored in memory?

Recursion uses more memory, because the recursive function adds to the stack with each recursive call, and keeps the values there until the call is finished. The recursive function uses LIFO (LAST IN FIRST OUT) Structure just like the stack data structure.

What is the base condition in recursion?
In the recursive program, the solution to the base case is provided and the solution to 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, the base case for n < = 1 is defined and the larger value of a number can be solved by converting to a smaller one till the base case is reached.

How a particular problem is solved using recursion?
The idea is to represent a problem in terms of one or more smaller problems, and add one or more base conditions that stop the recursion. For example, we compute factorial n if we know the factorial of (n-1). The base case for factorial would be n = 0. We return 1 when n = 0.

How memory is allocated to different function calls in recursion?
When any function is called from main(), the memory is allocated to it on the stack. A recursive function calls itself, the memory for a called function is allocated on top of memory allocated to the calling function and a different copy of local variables is created for each function call. When the base case is reached, the function returns its value to the function by whom it is called and memory is de-allocated and the process continues.
Let us take the example of how recursion works by taking a simple function.

Python

```
def printFun(test):

    if (test < 1):
        return
    else:

        print(test, end=" ")
        printFun(test-1) # statement 2
        print(test, end=" ")
        return

# Driver Code
test = 3
printFun(test)
```

Output :

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

When **printFun(3)** is called from main(), memory is allocated to **printFun(3)** and a local variable test is initialized to 3 and statement 1 to 4 are pushed on the stack as shown in below diagram. It first prints '3'. In statement 2, **printFun(2)** is called and memory is allocated to **printFun(2)** and a local variable test is initialized to 2 and statement 1 to 4 are pushed into the stack. Similarly, **printFun(2)** calls **printFun(1)** and **printFun(1)** calls **printFun(0)**. **printFun(0)** goes to if statement and it return to **printFun(1)**. The remaining statements of **printFun(1)** are executed and it returns to **printFun(2)** and so on. In the output, values from 3 to 1 are printed and then 1 to 3 are printed.

Recursion VS Iteration

SR No.	Recursion	Iteration
1)	Terminates when the base case becomes true.	Terminates when the condition becomes false.
2)	Used with functions.	Used with loops.
3)	Every recursive call needs extra space in the stack memory.	Every iteration does not require any extra space.
4)	Smaller code size.	Larger code size.

Summary of Recursion:

- There are two types of cases in recursion i.e. recursive case and a base case.
- The base case is used to terminate the recursive function when the case turns out to be true.
- Each recursive call makes a new copy of that method in the stack memory.
- Infinite recursion may lead to running out of stack memory.
- Examples of Recursive algorithms: Merge Sort, Quick Sort, Tower of Hanoi, Fibonacci Series, Factorial Problem, etc.