Recursion-1

Introduction

The process in which a function calls itself is called **recursion** and the corresponding function is called a **recursive function**.

Since computer programming is a fundamental application of mathematics, so let us first try to understand the mathematical reasoning behind recursion.

In general, we all are aware of the concept of functions. In a nutshell, functions are mathematical equations that produce an output on providing input. **For example:** Suppose the function **F(x)** is a function defined by:

$$F(x) = x^2 + 4$$

We can write the **Java Code** for this function as:

```
public static int F(int x){
    return (x * x + 4);
}
```

Now, we can pass different values of x to this function and receive our output accordingly.

Before moving onto the recursion, let's try to understand another mathematical concept known as the **Principle of Mathematical Induction (PMI)**.

Principle of Mathematical Induction (PMI) is a technique for proving a statement, a formula, or a theorem that is asserted about a set of natural numbers. It has the following three steps:

1. **Step of the trivial case:** In this step, we will prove the desired statement for a base case like **n = 0** or **n = 1**.
2. **Step of assumption:** In this step, we will assume that the desired statement is valid for **n = k**.
3. **To prove step:** From the results of the assumption step, we will prove that, **n = k + 1** is also true for the desired equation whenever **n = k** is true.

For Example: Let's prove using the **Principle of Mathematical Induction** that:

```
S(N): 1 + 2 + 3 + ... + N = (N * (N + 1))/2
```

(The sum of first N natural numbers)

Proof:

Step 1: For N = 1, S(1) = 1 is true.

Step 2: Assume, the given statement is true for N = k, i.e.,

```
1 + 2 + 3 + .... + k = (k * (k + 1))/2
```

Step 3: Let's prove the statement for N = k + 1 using step 2.

To Prove: `1 + 2 + 3 + ... + (k+1) = ((k+1)*(k+2))/2`

Proof:

Adding (k+1) to both LHS and RHS in the result obtained on step 2:

```
1 + 2 + 3 + ... + (k+1) = (k*(k+1))/2 + (k+1)
```

Now, taking (k+1) common from RHS side:

```
1 + 2 + 3 + ... + (k+1) = (k+1)*((k + 2)/2)
```

According the statement that we are trying to prove:

```
1 + 2 + 3 + ... + (k+1) = ((k+1)*(k+2))/2
```

Hence proved.

One can think, why are we discussing these over here. To answer this question, we need to know that these three steps of PMI are related to the three steps of recursion, which are as follows:

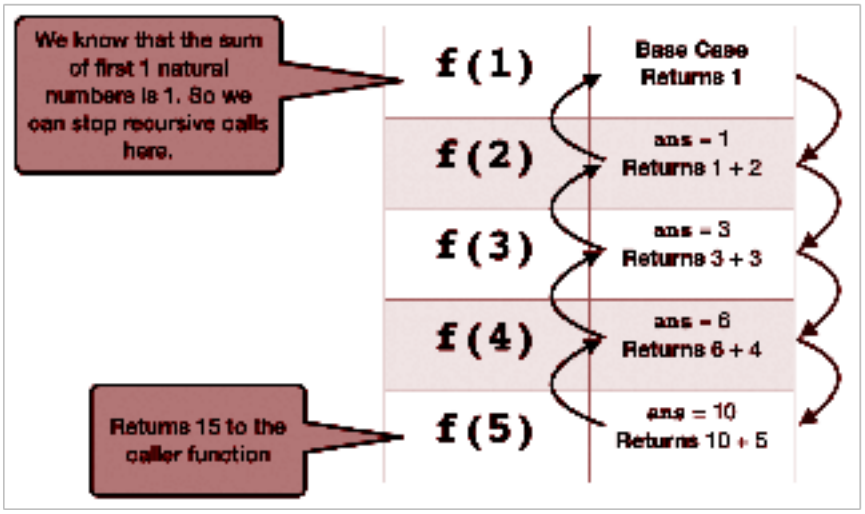
1. **Induction Step and Induction Hypothesis:** Here, the Induction Step is the main problem which we are trying to solve using recursion, whereas the Induction Hypothesis is the sub-problem, using which we'll solve the induction step. Let's define the Induction Step and Induction Hypothesis for our running example:
   Induction Step: Sum of first n natural numbers - F(n)
   Induction Hypothesis: This gives us the sum of the first n-1 natural numbers - F(n-1)
2. Express F(n) in terms of F(n-1) and write code:

```
F(N) = F(N-1)+ N
```

   Thus, we can write the Java code as:

```
public static int f(int N){
    int ans = f(N-1); //Induction Hypothesis step
    return ans + N;  //Solving problem from result in previous step
}
```

3. The code is still not complete. The missing part is the base case. Now we will dry run to find the case where the recursion needs to stop.



4. After the dry run, we can conclude that for N equals 1, the answer is 1, which we already know. So we'll use this as our base case. Hence the final code becomes:

```
public static int f(int N){
    if(N == 1)   // Base Case
        return 1;
    int ans = f(N-1);
    return ans + N;
}
```

This is the main idea to solve recursive problems. To summarize, we will always focus on finding the solution to our starting problem and tell the function to compute the rest for us using the particular hypothesis. This idea will be studied in detail in further sections with more examples.

Now, we'll learn more about recursion by solving problems which contain smaller subproblems of the same kind. Recursion in computer science is a method where the solution to the question depends on solutions to smaller instances of the same problem. By the exact nature, it means that the approach that we use to solve the original problem can be used to solve smaller problems as well. So, in other words, in recursion, a function calls itself to solve smaller problems. **Recursion** is a popular approach for solving problems because recursive solutions are generally easier to think than their iterative counterparts, and the code is also shorter and easier to understand.

Working of recursion

We can define the steps of the recursive approach by summarizing the above three steps:

- **Base case:** A recursive function must have a terminating condition at which the process will stop calling itself. Such a case is known as the base case. In the absence of a base case, it will keep calling itself and get stuck in an infinite loop. Soon, the **recursion depth*** will be exceeded and it will throw an error.
- **Recursive call:** The recursive function will invoke itself on a smaller version of the main problem. We need to be careful while writing this step as it is crucial to correctly figure out what your smaller problem is.
- **Small calculation:** Generally, we perform a calculation step in each recursive call. We can achieve this calculation step before or after the recursive call depending upon the nature of the problem.

Note*: Recursion uses an in-built stack which stores recursive calls. Hence, the number of recursive calls must be as small as possible to avoid memory-overflow. If the number of recursion calls exceeded the maximum permissible amount, the **recursion depth*** will be exceeded.
Now, let us see how to solve a few common problems using Recursion.

Problem Statement - Find Factorial of a Number

We want to find out the factorial of a natural number.

Approach: Figuring out the three steps of PMI and then relating the same using recursion.

1. **Induction Step:** Calculating the factorial of a number n - **F(n)**