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.

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 factorial(int n)
{
    if (n < = 1) // base case
        return 1;
    else
        return n*factorial(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.

Solution for a particular problem 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 factorial of (n-1). The base case for factorial would be n = 0. We return 1 when n = 0.

Stack Overflow error in recursion

If the base case is not reached or not defined, then the stack overflow problem may arise. Let us take an example to understand this.

```
int factorial(int n)
{
    // wrong base case (it may cause
    // stack overflow).
    if (n == 100)
        return 1;

    else
```

```
        return n*factorial(n-1);
}
```

If fact(10) is called, it will call fact(9), fact(8), fact(7) and so on but the number will never reach 100. So, the base case is not reached. If the memory is exhausted by these functions on the stack, it will cause a stack overflow error.

Difference between direct and indirect recursion

A function fun is called direct recursive if it calls the same function fun. A function A is called indirect recursive if it calls another function say B and B calls A directly or indirectly. Difference between direct and indirect recursion has been illustrated in Table 1.

```
// An example of direct recursion
void A()
{
    // Some code....

    A();

    // Some code...
}

// An example of indirect recursion
void A()
{
    // Some code...

    B();

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

    A();

    // Some code...
}
```

Memory allocation 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 calling function and different copy of local variables is created for each function call. When the base case is reached, the function returns its value

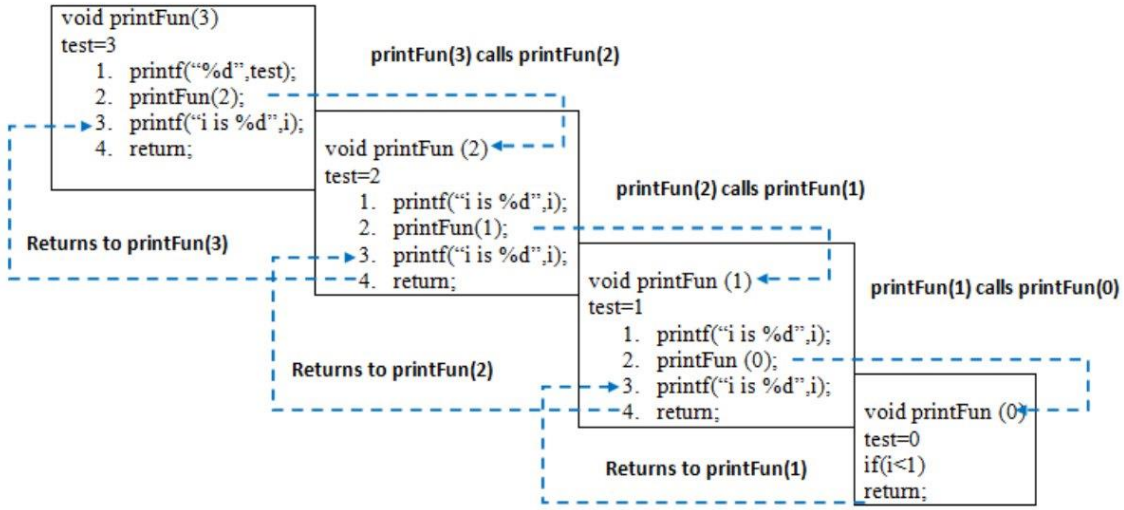
to the function by whom it is called and memory is de-allocated and the process continues.

```
/ A C program to demonstrate working of recursion

#include<stdio.h>

void printFun(int test)
{
    if (test < 1)
        return;
    else
    {
        printf("%d ", test);
        printFun(test-1);    // statement 2
        printf("%d ", test);
        return;
    }
}

int main()
{
    int test = 3;
    printFun(test);
    return(0);
}
```

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 in 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)**. Remaining statements of **printFun(1)** are executed and it returns to **printFun(2)** and so on. In the output, value from 3 to 1 are printed and then 1 to 3 are printed. The memory stack has been shown in below diagram.



Advantages of recursive programming over iterative programming

Recursion provides a clean and simple way to write code. Some problems are inherently recursive like tree traversals, Tower of Hanoi, etc. For such problems, it is preferred to write recursive code. We can write such codes also iteratively with the help of a stack data structure. For example refer Inorder Tree Traversal without Recursion, Iterative Tower of Hanoi.

Disadvantages of recursive programming over iterative programming

Note that both recursive and iterative programs have the same problem-solving powers, i.e., every recursive program can be written iteratively and vice versa is also true. The recursive program has greater space requirements than iterative program as all functions will remain in the stack until the base case is reached. It also has greater time requirements because of function calls and returns overhead.

Courtesy : https://www.geeksforgeeks.org/recursion/