Custom Search

COURSES

HIRE WITH US

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.

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.

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

Why Stack Overflow error occurs 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 fact(int n)
{
    // wrong base case (it may cause
    // stack overflow).
    if (n == 100)
        return 1;

    else
        return n*fact(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.

What is the difference between direct and indirect recursion?

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

```
// An example of direct recursion
void directRecFun()
```

```
    {
        // Some code....

        directRecFun();

        // Some code...
    }

    // An example of indirect recursion
    void indirectRecFun1()
    {
        // Some code...

        indirectRecFun2();

        // Some code...
    }
    void indirectRecFun2()
    {
        // Some code...

        indirectRecFun1();

        // Some code...
    }
```

What is difference between tailed and non-tailed recursion?

A recursive function is tail recursive when recursive call is the last thing executed by the function. Please refer tail recursion article for details.

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 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.

Let us take the example how recursion works by taking a simple function.

CPP

```cpp
// A C++ program to demonstrate working of
// recursion
#include<bits/stdc++.h>
using namespace std;

void printFun(int test)
{
    if (test < 1)
        return;
    else
    {
        cout << test << " ";
        printFun(test-1);    // statement 2
        cout << test << " ";
        return;
    }
}

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

Java

```java
// A Java program to demonstrate working of
// recursion
```

```java
class GFG{
static void printFun(int test)
{
    if (test < 1)
        return;
    else
    {
        System.out.printf("%d ",test);
        printFun(test-1); // statement 2
        System.out.printf("%d ",test);
        return;
    }
}

public static void main(String[] args)
{
    int test = 3;
    printFun(test);
}
}

// This code is contributed by
// Smitha Dinesh Semwal
```

Python3

```python
# A Python 3 program to
# demonstrate working of
# recursion

def printFun(test):

    if (test < 1):
        return
    else:

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


test = 3
printFun(test)

# This code is contributed by
# Smitha Dinesh Semwal
```

C#

```csharp
// A C# program to demonstrate
// working of recursion
using System;

class GFG {

    // function to demonstrate
    // working of recursion
    static void printFun(int test)
    {
        if (test < 1)
            return;
        else
        {
            Console.Write(test + " ");

            // statement 2
            printFun(test - 1);

            Console.Write(test + " ");
            return;
        }
    }

    // Driver Code
    public static void Main(String[] args)
```

```
    {
        int test = 3;
        printFun(test);
    }
}

// This code is contributed by Anshul Aggarwal.
```

PHP

```php
<?php
// PHP program to demonstrate
// working of recursion

// function to demonstrate
// working of recursion
function printFun($test)
{
    if ($test < 1)
        return;
    else
    {
        echo("$test ");

        // statement 2
        printFun($test-1);

        echo("$test ");
        return;
    }
}

// Driver Code
$test = 3;
printFun($test);

// This code is contributed by
// Smitha Dinesh Semwal.
?>
```

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.
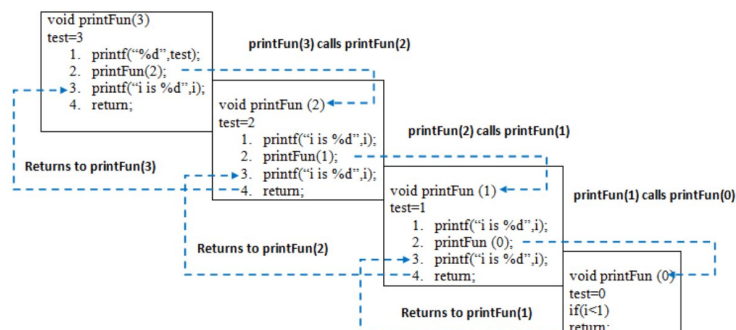
What are the disadvantages of recursive programming over iterative programming?
Note that both recursive and iterative programs have the same problem-solving powers, i.e., every recursive program can be written iteratively

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.

What are the 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.

Output based practice problems for beginners:

Practice Questions for Recursion | Set 1

Practice Questions for Recursion | Set 2

Practice Questions for Recursion | Set 3

Practice Questions for Recursion | Set 4

Practice Questions for Recursion | Set 5

Practice Questions for Recursion | Set 6

Practice Questions for Recursion | Set 7

Quiz on Recursion

Coding Practice on Recursion:

All Articles on Recursion

Recursive Practice Problems with Solutions

This article is contributed by **Sonal Tuteja**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Recommended Posts:

Sum of the series 1^1 + 2^2 + 3^3 + ... + n^n using recursion

Tail Recursion

Find the value of ln(N!) using Recursion

Product of 2 Numbers using Recursion

Sum of digit of a number using recursion

Generating subarrays using recursion

Generating all possible Subsequences using Recursion

Product of 2 numbers using recursion | Set 2

Difference between Recursion and Iteration

Reversing a queue using recursion

Program to Calculate e^x by Recursion

Practice Questions for Recursion | Set 6

Sum of even elements of an Array using Recursion

Practice Questions for Recursion | Set 5

Practice Questions for Recursion | Set 4

Improved By : Anshul_Aggarwal, Kirti_Mangal

Article Tags :   Algorithms   Recursion   tail-recursion

Practice Tags :   Recursion   Algorithms

👍

32

☐ To-do ☐ Done

2.5

Based on **37** vote(s)

(Feedback/ Suggest Improvement)   (Notes) (Improve Article)

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments

A computer science portal for geeks

5th Floor, A-118,
Sector-136, Noida, Uttar Pradesh - 201305
feedback@geeksforgeeks.org

COMPANY

About Us
Careers
Privacy Policy
Contact Us

LEARN

Algorithms
Data Structures
Languages
CS Subjects
Video Tutorials

PRACTICE

Courses
Company-wise
Topic-wise
How to begin?

CONTRIBUTE

Write an Article
Write Interview Experience
Internships
Videos