Recursion in C

Recursion is the process which comes into existence when a function calls a copy of itself to work on a smaller problem. Any function which calls itself is called recursive function, and such function calls are called recursive calls. Recursion involves several numbers of recursive calls. However, it is important to impose a termination condition of recursion. Recursion code is shorter than iterative code however it is difficult to understand.

Recursion cannot be applied to all the problem, but it is more useful for the tasks that can be defined in terms of similar subtasks. For Example, recursion may be applied to sorting, searching, and traversal problems.

Generally, iterative solutions are more efficient than recursion since function call is always overhead. Any problem that can be solved recursively, can also be solved iteratively. However, some problems are best suited to be solved by the recursion, for example, tower of Hanoi, Fibonacci series, factorial finding, etc.

In the following example, recursion is used to calculate the factorial of a number.

```
1. #include <stdio.h>
2. int fact (int);
3. int main()
4. {
5.     int n,f;
6.     printf("Enter the number whose factorial you want to calculate?");
7.     scanf("%d",&n);
8.     f = fact(n);
9.     printf("factorial = %d",f);
10. }
11. int fact(int n)
12. {
13.     if (n==0)
14.     {
15.         return 0;
16.     }
17.     else if ( n == 1)
```

```
18.    {
19.        return 1;
20.    }
21.    else
22.    {
23.        return n*fact(n-1);
24.    }
25. }
```

Output

```
Enter the number whose factorial you want to calculate?5
factorial = 120
```

We can understand the above program of the recursive method call by the figure given below:

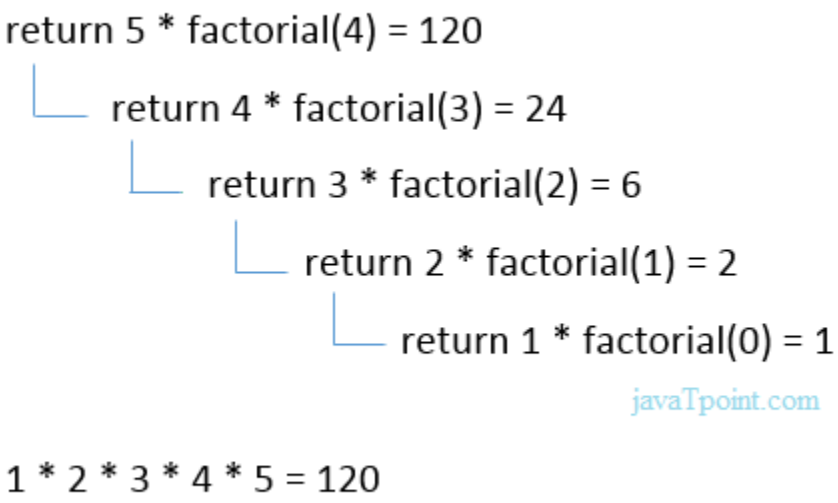

Fig: Recursion

Recursive Function

A recursive function performs the tasks by dividing it into the subtasks. There is a termination condition defined in the function which is satisfied by some specific subtask. After this, the recursion stops and the final result is returned from the function.

The case at which the function doesn't recur is called the base case whereas the instances where the function keeps calling itself to perform a subtask, is called the recursive case. All the recursive functions can be written using this format.

Pseudocode for writing any recursive function is given below.

```
if (test_for_base)
{
    return some_value;
}
else if (test_for_another_base)
{
    return some_another_value;
}
else
{
    // Statements;
    recursive call;
}
```

Example of recursion in C

Let's see an example to find the nth term of the Fibonacci series.

```
#include<stdio.h>
int fibonacci(int);
void main ()
{
    int n,f;
    printf("Enter the value of n?");
    scanf("%d",&n);
    f = fibonacci(n);
    printf("%d",f);
}
int fibonacci (int n)
{
```

```
13.    if (n==0)
14.    {
15.    return 0;
16.    }
17.    else if (n == 1)
18.    {
19.        return 1;
20.    }
21.    else
22.    {
23.        return fibonacci(n-1)+fibonacci(n-2);
24.    }
25. }
```

Memory allocation of Recursive method

Each recursive call creates a new copy of that method in the memory. Once some data is returned by the method, the copy is removed from the memory. Since all the variables and other stuff declared inside function get stored in the stack, therefore a separate stack is maintained at each recursive call. Once the value is returned from the corresponding function, the stack gets destroyed. Recursion involves so much complexity in resolving and tracking the values at each recursive call. Therefore we need to maintain the stack and track the values of the variables defined in the stack.

Let us consider the following example to understand the memory allocation of the recursive functions.

```
1.  int display (int n)
2.  {
3.      if(n == 0)
4.          return 0; // terminating condition
5.      else
6.      {
7.          printf("%d",n);
8.          return display(n-1); // recursive call
9.      }
```