Recursion

Recursion

- Very powerful technique to write a complicated algorithm in easy way.
- Defined as defining something in terms of a simpler case of itself.
- A function which calls itself is called a recursive function.

How Recursion Works

- The compiler uses a stack to store the values before the next function call is given.
- After the last function call is executed, the compiler pops values from the stack to resume execution of the previous function call.

Eg. Factorial of a given number

- 5! = 5 * 4!
  = 5 * 4 * 3!
  = 5 * 4 * 3 * 2!
  = 5 * 4 * 3 * 2 * 1!
  = 5 * 4 * 3 * 2 * 1 * 0!
  = 5 * 4 * 3 * 2 * 1 * 1
  = 120
- n! = n * (n-1)! For all n > 0
  = 1 For n = 0

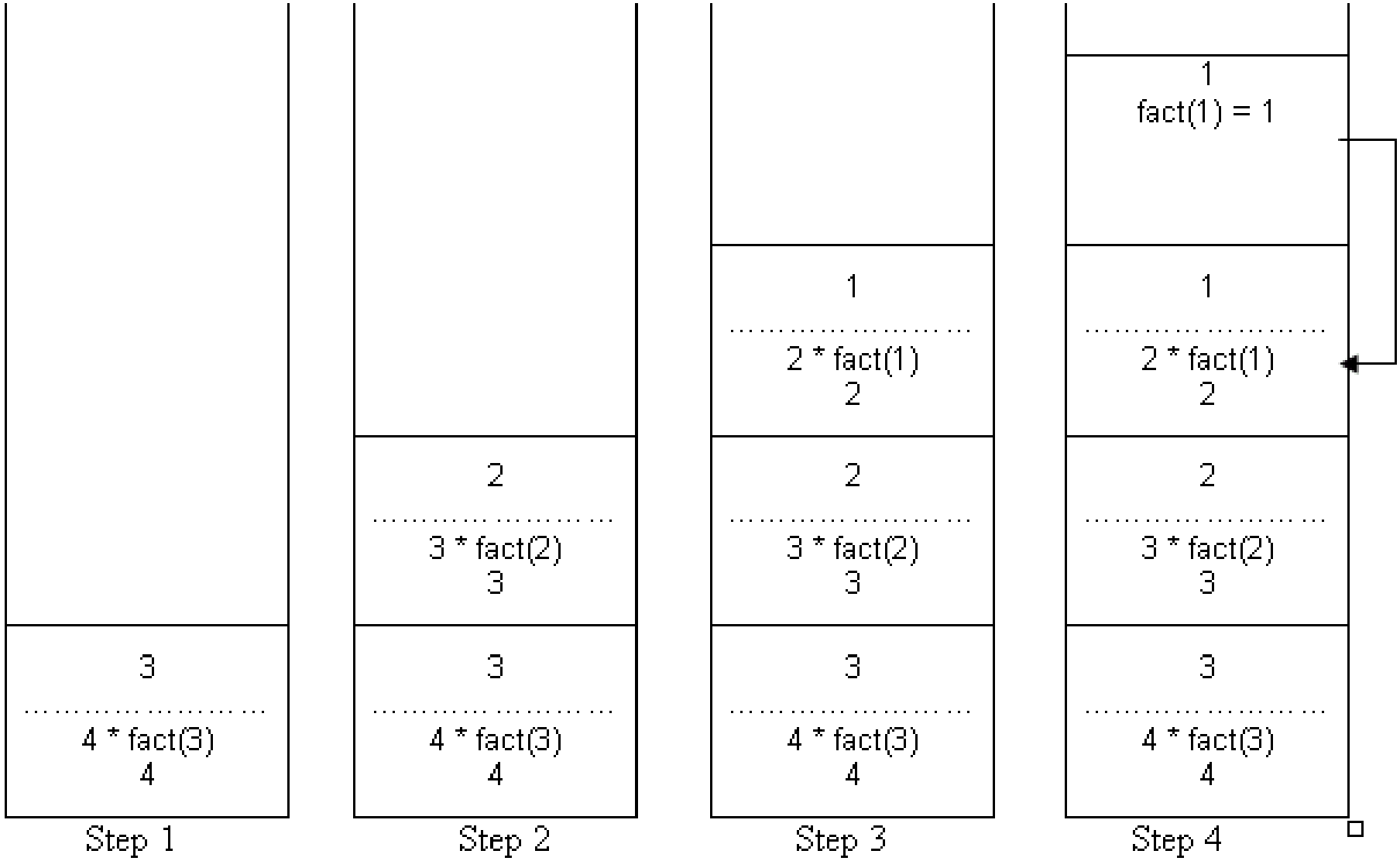
Recursive function to find factorial of a given number

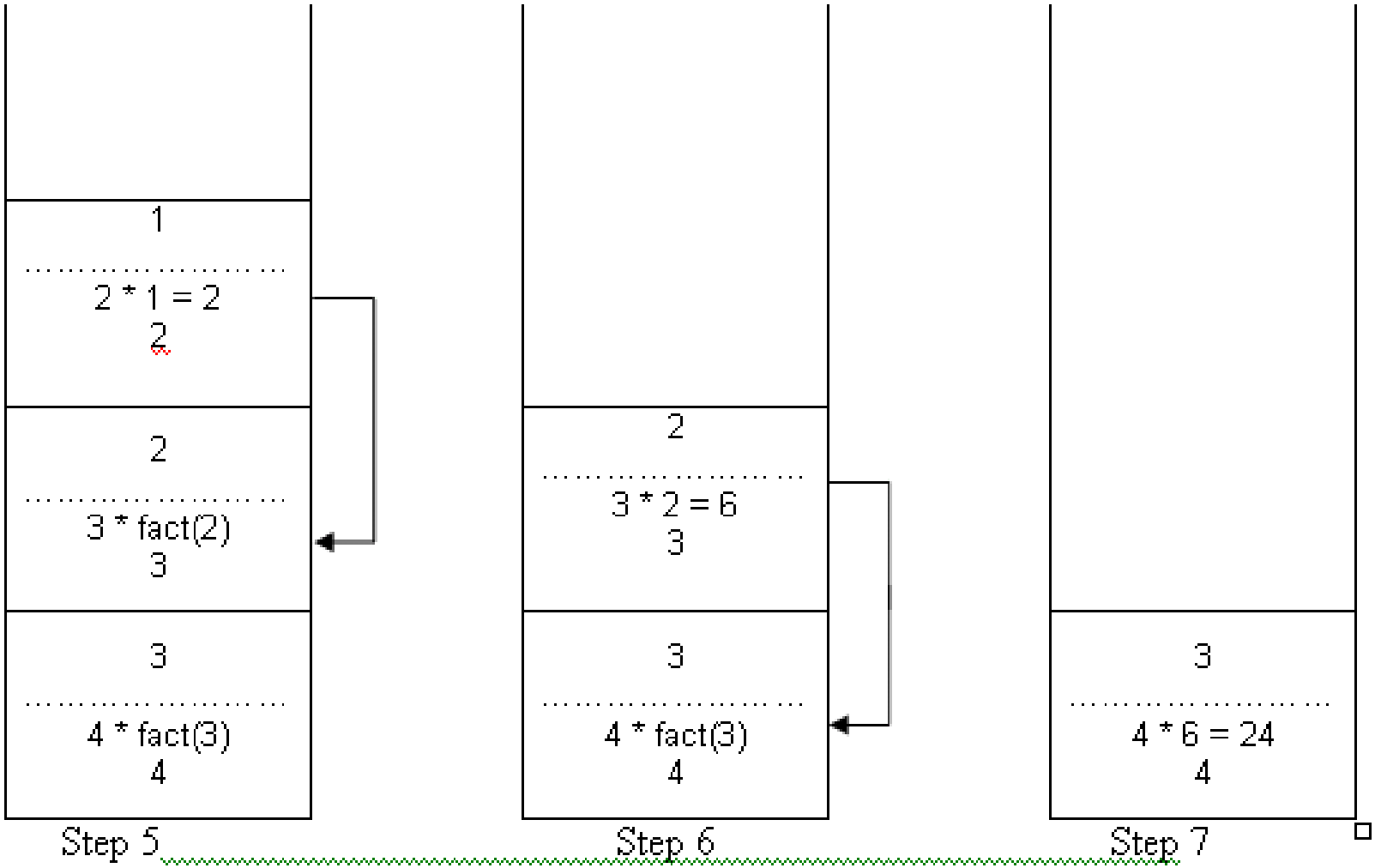
```
int factorial(int n)
{
    if(n==1 || n == 0)
        return(1);
    else
        return(n*factorial(n-1));
}
```

Recursive Program

```
#include<stdio.h>
void main()
{
    int n,fact;
    printf("\nEnter a no : ");
    scanf("%s",&n);
    fact = factorial(n);
    printf("%d",fact);
}
```

```
int factorial(int n)
{
    int f;
    if(n==1 || n == 0)
        return(1);
    else
     {
        f = n * factorial(n-1);
        return(f);
     }
}
```


3
4 * fact(3)
4
Step 1
2
3 * fact(2)
3
3
4 * fact(3)
4
Step 2
1
2 * fact(1)
2
2
3 * fact(2)
3
3
4 * fact(3)
4
Step 3
1
fact(1) = 1
1
2 * fact(1)
2
2
3 * fact(2)
3
3
4 * fact(3)
4
Step 4


1
2 * 1 = 2
2
2
3 * fact(2)
3
3
4 * fact(3)
4
Step 5
2
3 * 2 = 6
3
3
4 * fact(3)
4
Step 6
3
4 * 6 = 24
4
Step 7

Tail Recursion

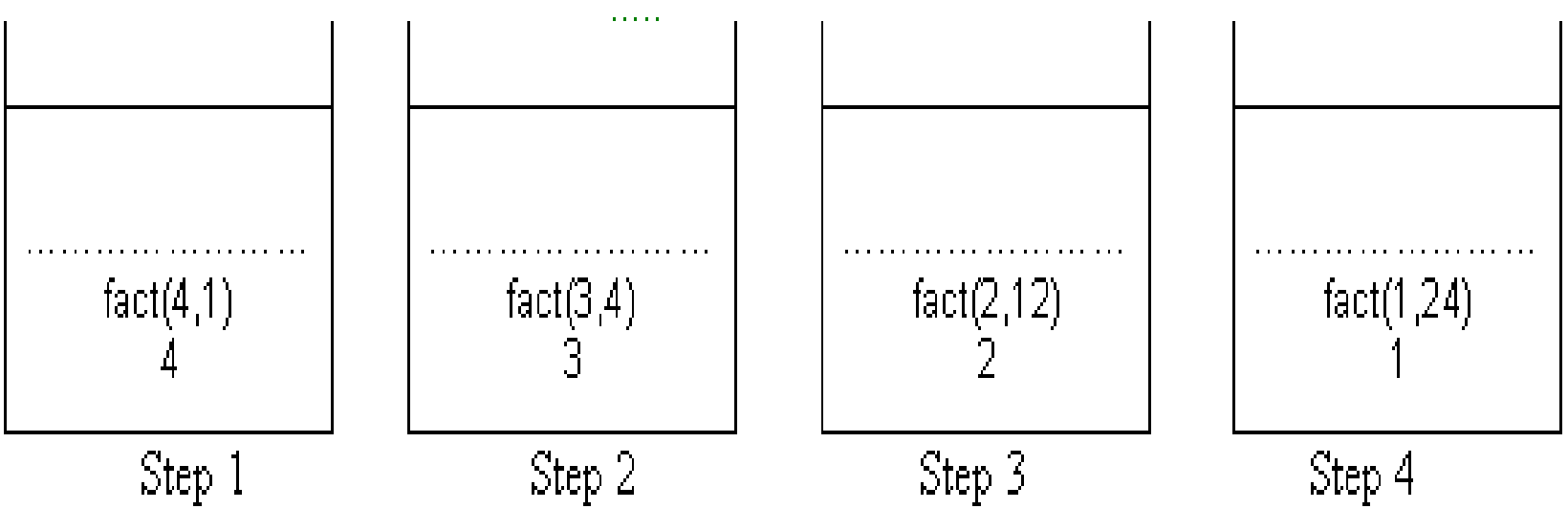
- As we know the process in which function calls itself in function body is called as recursion. But when this called function is the last executed statement in the function body then it is called tail recursion.
- Here we take the return value as one parameter of function itself. We use stack to maintain all the functions but here it will not append new function in stack but it will overwrite the value of previous function with the current one.
- So we can say the function call time and stack implementation time will be reduced and it will give better performance.

Recursive Program

```
#include<stdio.h>
void main()
{
    int n,fact;
    printf("\nEnter a no : ");
    scanf("%s",&n);
    fact = factorial(n,1);
    printf("%d",fact);
}
```

```
int factorial(int n,int f)
{
    if(n==1 || n == 0)
        return(f);
    else
        factorial(n-1,n*f);
}
```

Stack Implementation



Here we can see tail recursion takes only 4 steps for getting factorial of number 4.
It reduces space and time both and improves the performance.

Types of Recursion

- Direct
- Indirect
- Tail Recursion

Efficiency of Recursion

- In general a non recursive solution will be more efficient than a recursive solution in terms of time and space complexity.
- In the non recursive solution, there is no extra overhead involved in function calls and returns and hence faster execution.
- A function may have many variables and constants which need not be saved and restored using a stack. However, in recursion, all these will be stacked thereby utilizing stack space.
- Thus recursive solution solutions have a poor space complexity due to stack space requirements.
- However, in some cases, a recursive solution is the most logical and natural solution. Complex problems can be easily solved using Recursion.

Examples

- Finding nth term of Fibonacci series

Fibo(n) = Fibo(n-1) + Fibo(n-2) if n > 2

= 1 if n = 1 or 2

- Finding GCD of two numbers x & y

GCD(x,y) = x if y = 0

= GCD(y , x %y) otherwise