Rule of Three, Recursion, and Backtracking

Rule of Three Example

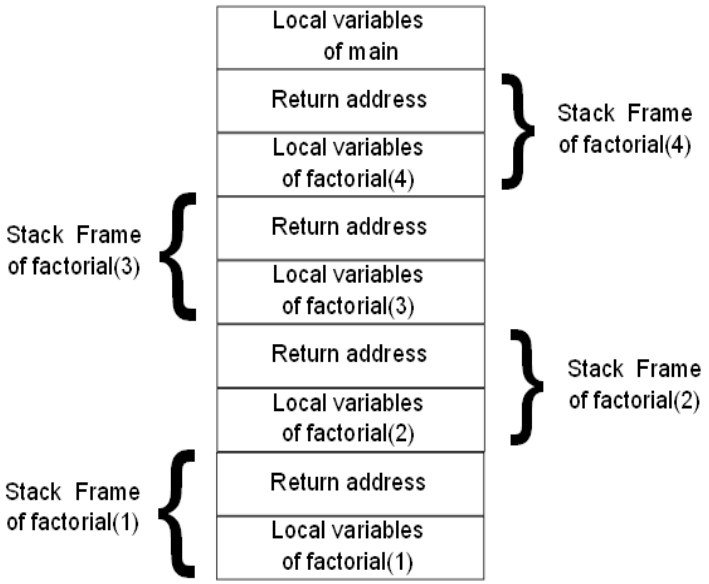
- Implementation of 1D Safe Array

What is Recursion?

- **Recursion** is a method of solving problems.
- It involves breaking a problem down into smaller and smaller sub problems until you get to a small enough problem that it can be solved trivially.
- Usually recursion involves a function calling itself.

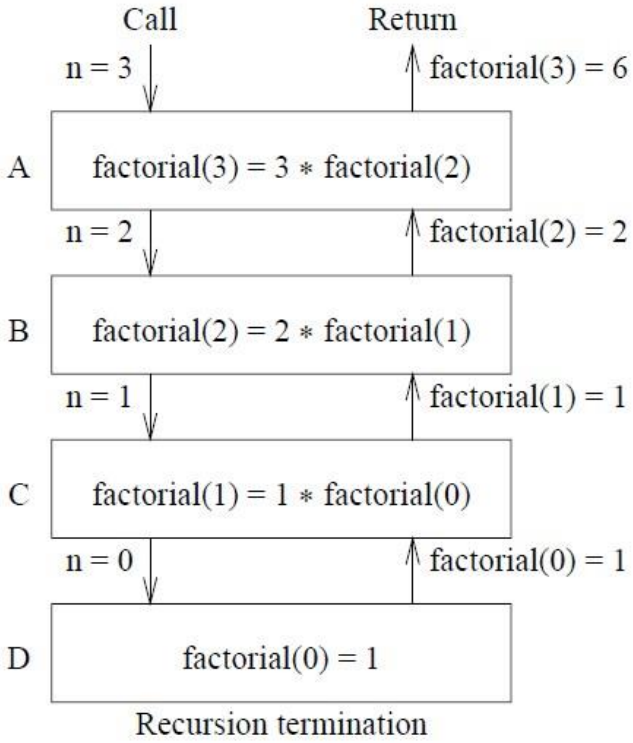
Factorial Example

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

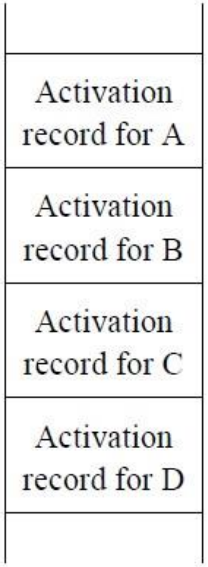


Simplified Stack of factorial

Recursion Activation Records



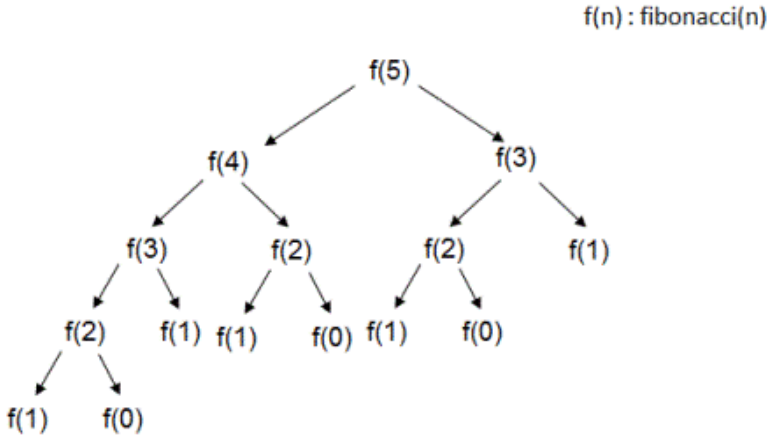
(a)



(b)

Fibonacci Series

```
int fib(int n)
{
    if (n <= 1)
        return n;
    return fib(n-1) +
    fib(n-2);
}
```



Recursion tree generated for computing 5th number of fibonacci sequence

Finding Max

```
Int max(int i, inj j, int *a) {
    if(i > j) exit(1)
    if (i == j) return a[i];
    else
            middle =(i+j)/2;
            int max1 = max(i,middle,a);
            int max2= max(middle+1, j, a );
                    if(max1 > max2)
                            return max1;
                    return max2;
}
```

What is tail recursion?

- Recursive methods are either
  - Tail recursive
  - Non tail recursive
- Tail recursive method has the recursive call as the last statement in the method.

Is factorial a tail recursive?

Is the factorial method a tail recursive method?

```
int fact(int x){
   if (x==0)
      return 1;
   else
      return x*fact(x-1);
}
```

When returning back from a recursive call, there is still one pending operation, multiplication.

Therefore, factorial is a non-tail recursive method.

Another Example

Is this method tail recursive?

```
void tail(int i) {
if (i>0) {
   system.out.print(i+"")
tail(i-1)
}
```

It is tail recursive!

Another Example

Is the following program tail recursive?

```
void non prog(int i) {
   if (i>0) {
         prog(i-1);
         System.out.print(i+"");
         prog(i-1);
    }
}
```

No, because there is an earlier recursive call, other than the last one,

In tail recursion, the recursive call should be the last statement, and there should be no earlier recursive calls whether direct or indirect.

Advantage of tail recursive method

Tail Recursive methods are easy to convert to iterative.

```
void tail(int i){
  if (i>0) {

system.out.println(i+"");
    tail(i-1)
  }
}
```

```
void iterative(int i){
  for (;i>0;i--)

System.out.println(i+"");
}
```

Smart compilers can detect tail recursion and convert it to iterative to optimize code

Used to implement loops in languages that do not support loop structures explicitly (e.g. prolog)

Converting Non-tail to tail recursive

- A non-tail recursive method can be converted to a tail- recursive method by means of an "auxiliary" parameter used to form the result.

```
  int fact_aux(int n, int
  result) {  if (n == 1)
     return result;
   return fact_aux(n - 1, n * result)
}
int fact(n) {
   return fact_aux(n, 1);
}
```