Recussion Basics

Day18
28/07/2025

1. Functions
2. stack
3. Why and Where recusssion is used
   a. folder structure
   b. family tree
4. Important techniques to write recussive code
5. common pitfalls of Recussion
6. Recussion Infinite Loop or Stack overflow
7. Some Example programs.

Before going for Recussion lets know what is function.

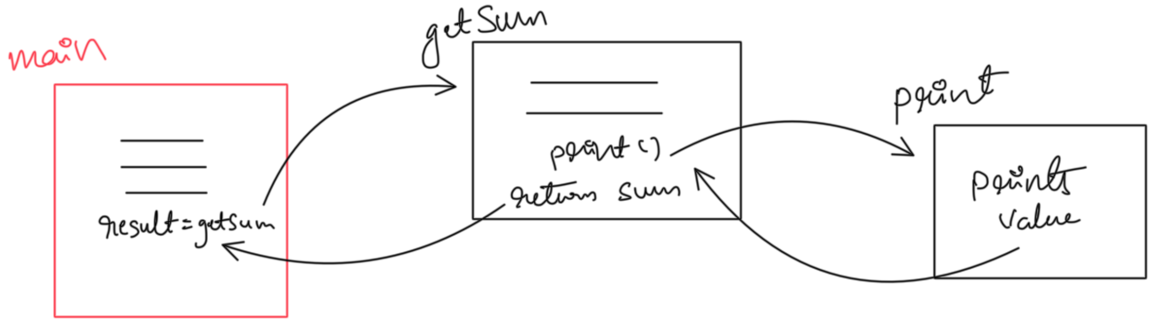
< return type > functionname (Input Arguments)
↓ return datatype
↓ function name
↓ Input's

function

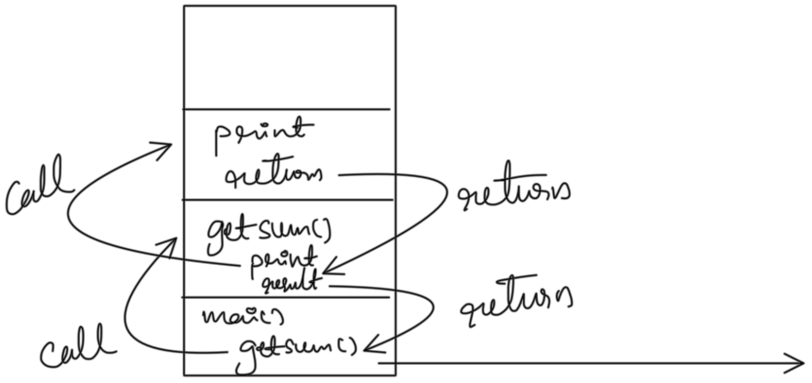
```
int   getSum (int num1, int num2)
{
        // function defination

        Print (result) → Calling another function
        return result
}
```

Before returning the result the "getSum" function is Calling another function call "print";

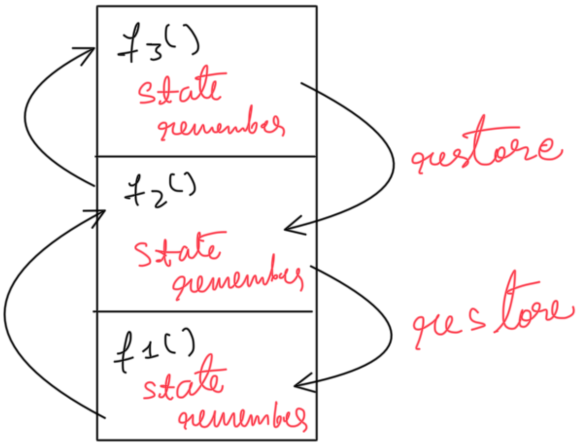


As we know for Any program or the function is Running there is a stack Memory for that Will be allocated.

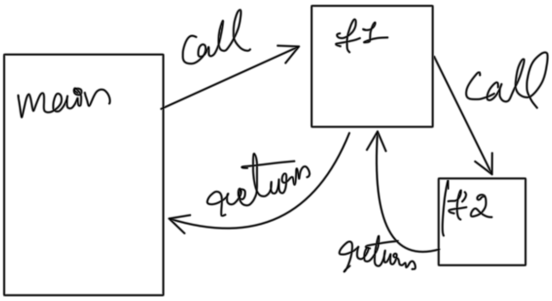


In Each function call the stack memory formed

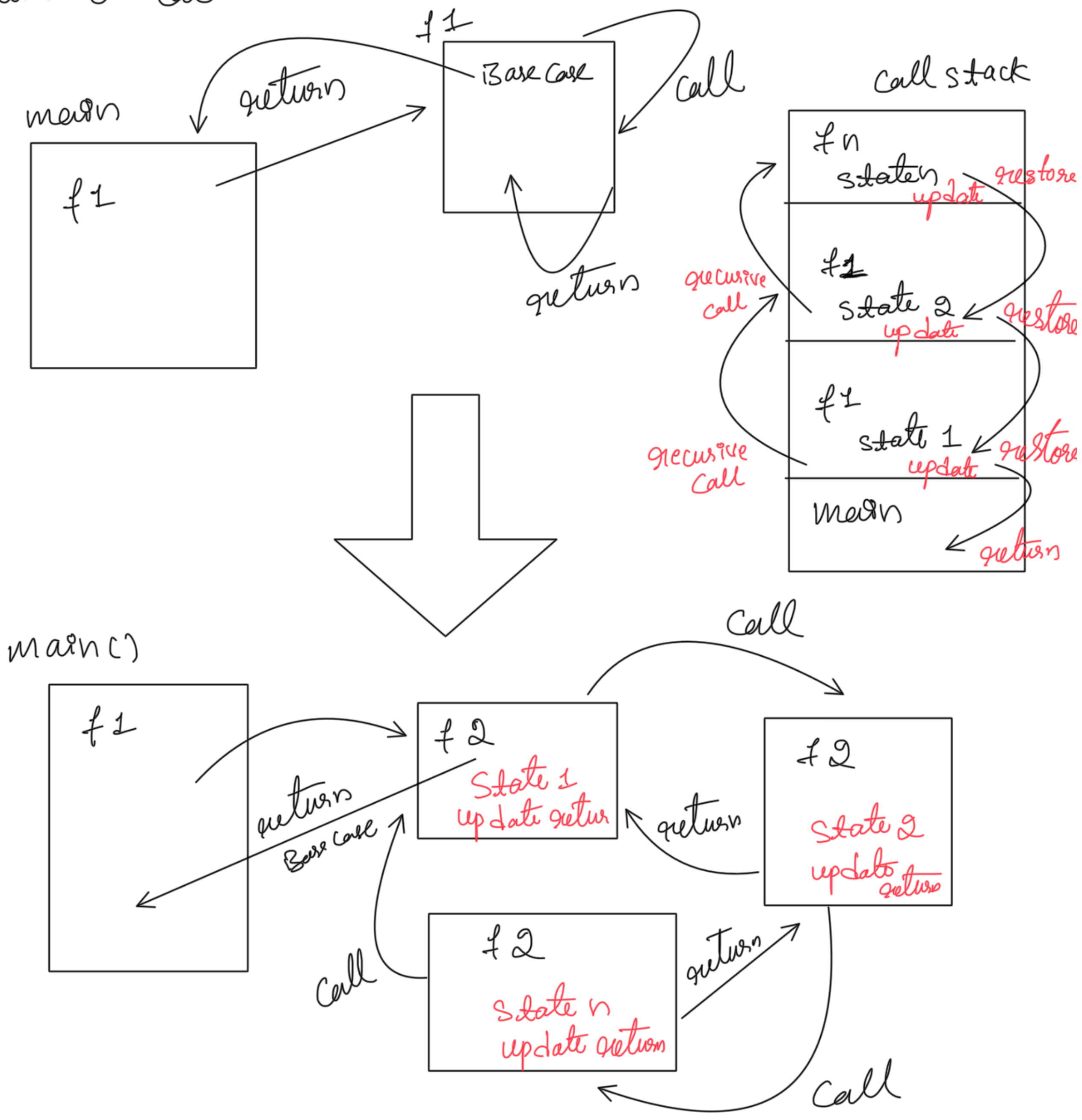
Rembers the state of data. function (Variable value, or any data)



Normal function call

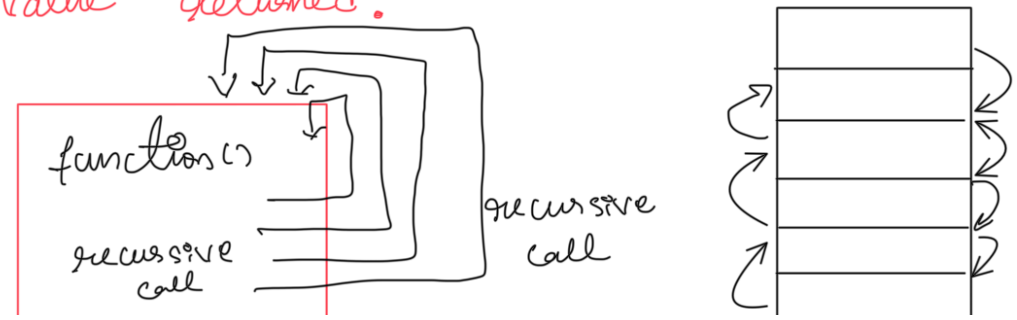


Recursive Call

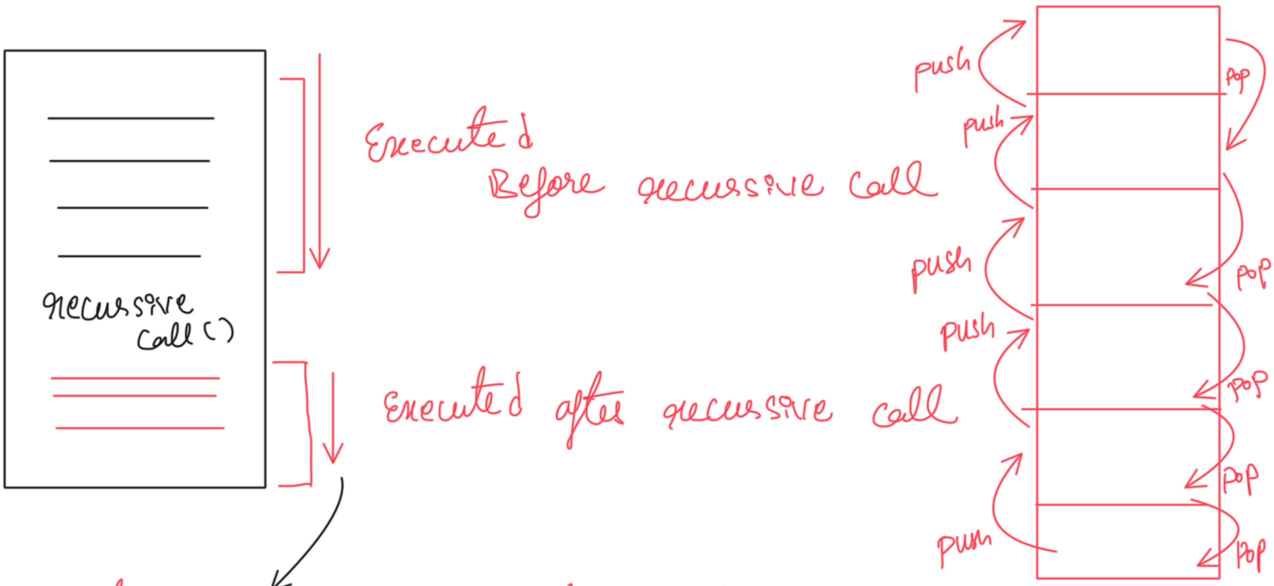


Every state of Each Recursive call is remembered In the call STACK Separately. after returning from function call state is updated & updated state or value returned.

pattern 1

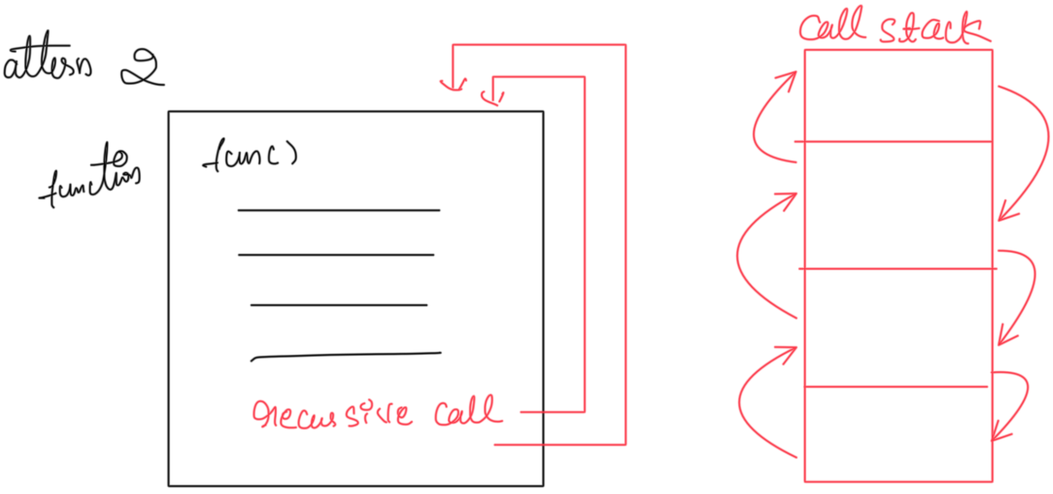


Which Block of code Executed when



Majority code is Executed when pop operation is performed.

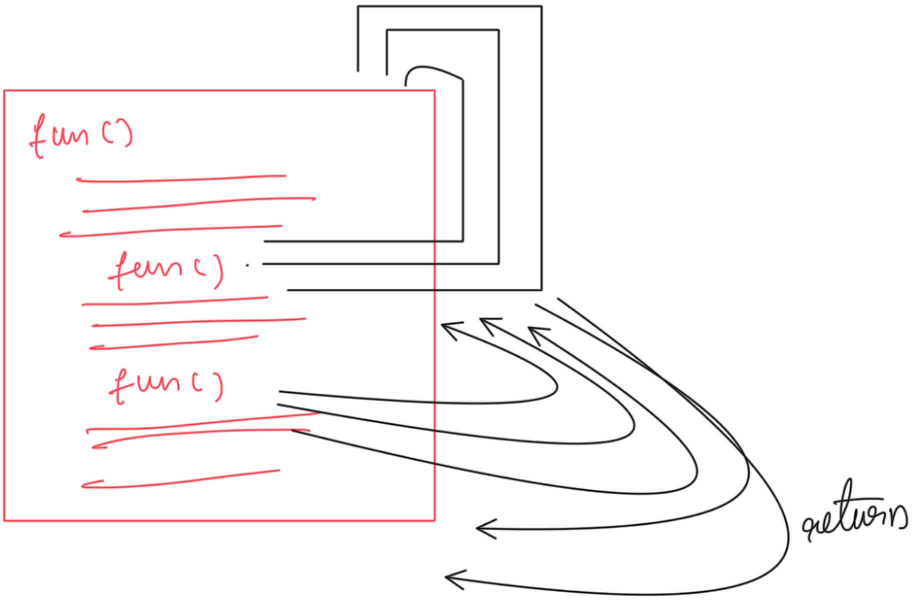
pattern 2



Majority of the code Executed Before push Operation

pattern 3

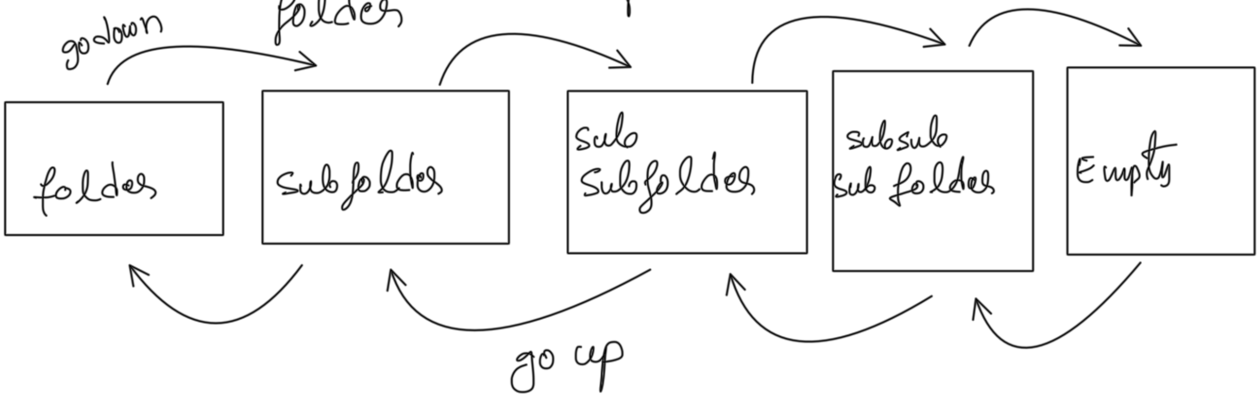
Having multiple recursive calls inside the function which creates a confusion and makes Hard to understand.



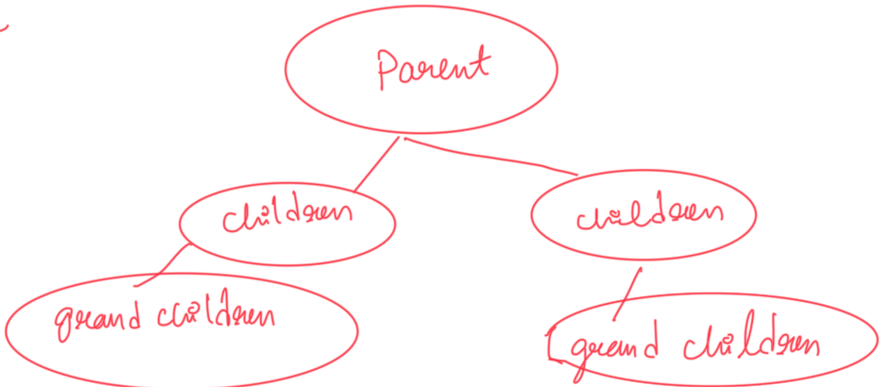
Why & where Recursion is used?

let take folder structure as Example

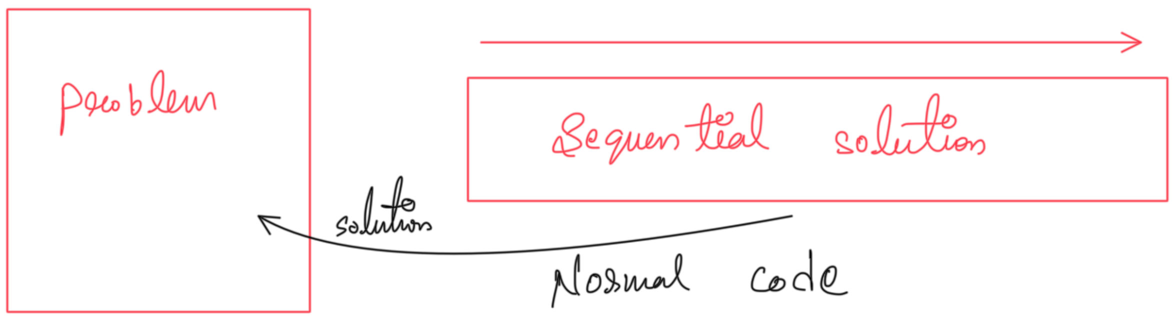
To keep track of subfolders inside a folder

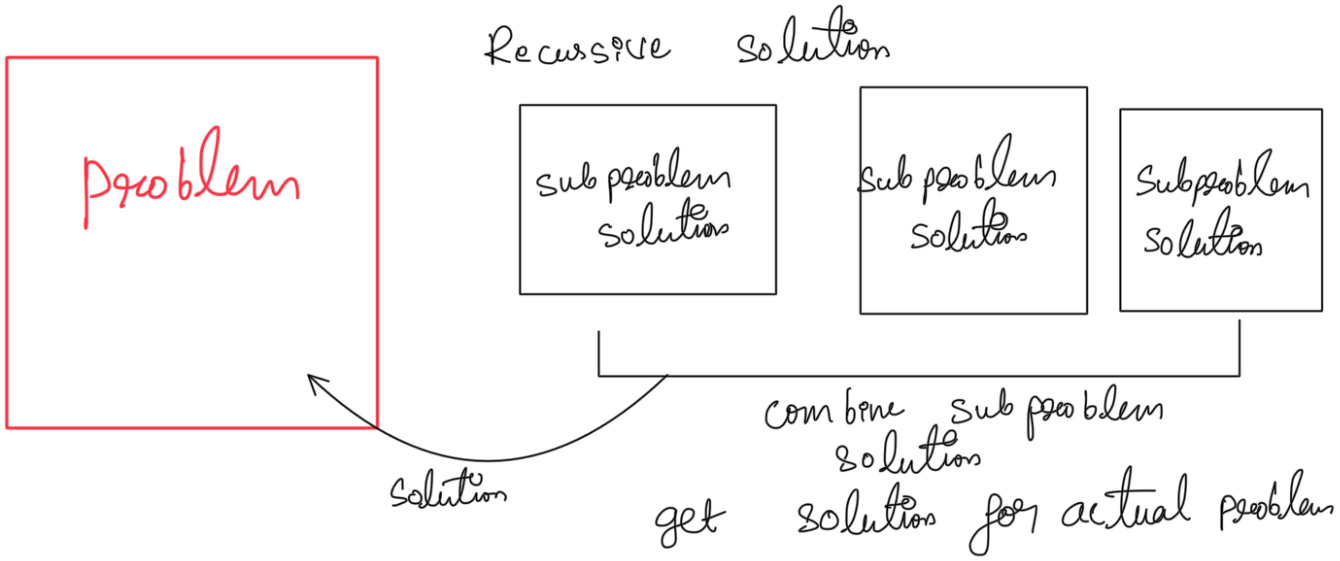


family tree



Techniques to Write Recursion





solving Sub problem with the help of divide & Conquerer helps to get solution for main Problem.

Every recursive call should have a break condition Otherwise we cannot get the result which Causes the stack overflow in the memory.

The Break condition is also called as Base case The Base Case is the Condition to stop recursive call. (To avoid Infinite loop)

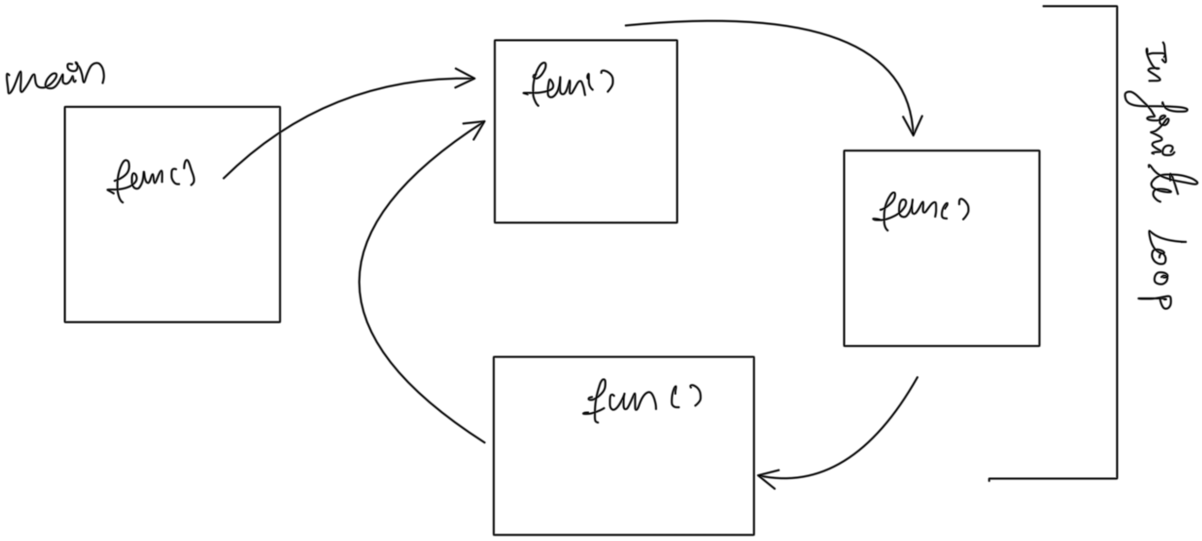


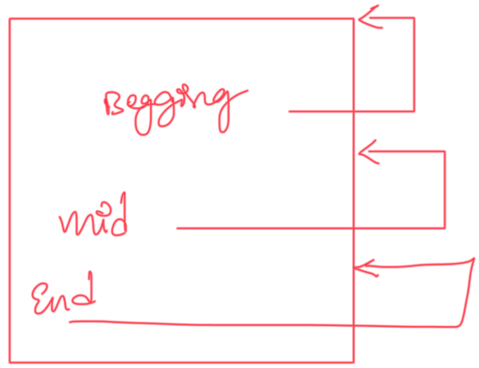
fig ÷ Recussive function call without a Base Case

The Infinite Loop Creates a stack overflow or System crash.

---

points to understand before writting Recursive call

Condition ①



from where you are Calling the function

Eg then you are calling from Beginning or you are calling from mid or you are calling from End.

recursive call decides the result
wrong recursive call It effects the result

condition ② deciding which Block of code should be Executed when

Eg thus Befor recussive Call or

func()

After recursive call

Before the recursive call what you write after the recursive call what you write

condition ③ Another factor is weather you are calling recursive function one time or calling multiple times

func()

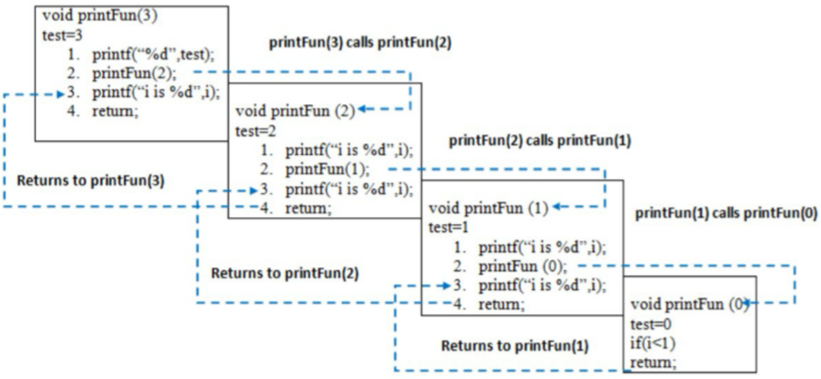
func()
func()

func()
func()
func()

func()
⋮
func()

If Any of the Above conditions are failed in writting recursive call which effects the results

So the above 3 mentioned conditions are the common pitfall.



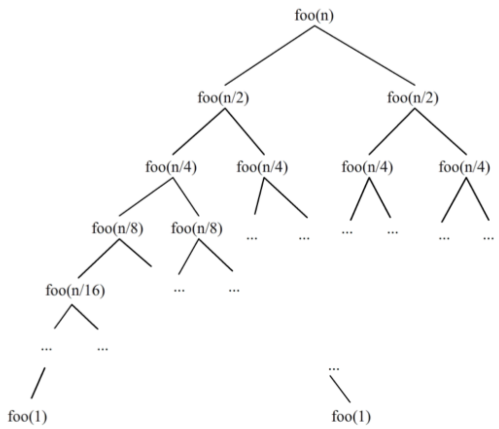
```
void printFun(3)
test=3
  1. printf("%d",test);
  2. printFun(2);
  3. printf("i is %d",i);
  4. return;
```

```
void printFun (2)
test=2
  1. printf("i is %d",i);
  2. printFun(1);
  3. printf("i is %d",i);
  4. return;
```

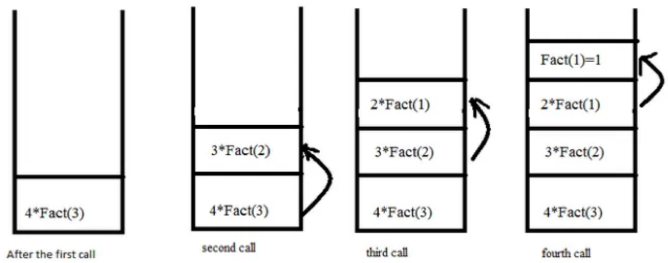
```
void printFun (1)
test=1
  1. printf("i is %d",i);
  2. printFun (0);
  3. printf("i is %d",i);
  4. return;
```

```
void printFun (0)
test=0
if(i<1)
return;
```

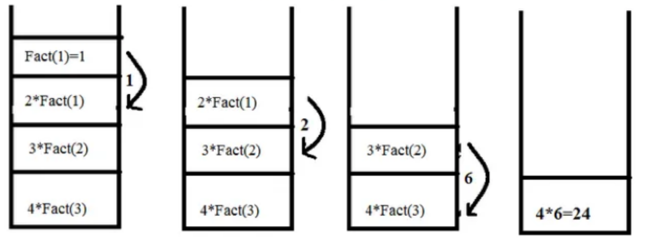
printFun(3) calls printFun(2)
printFun(2) calls printFun(1)
printFun(1) calls printFun(0)
Returns to printFun(3)
Returns to printFun(2)
Returns to printFun(1)

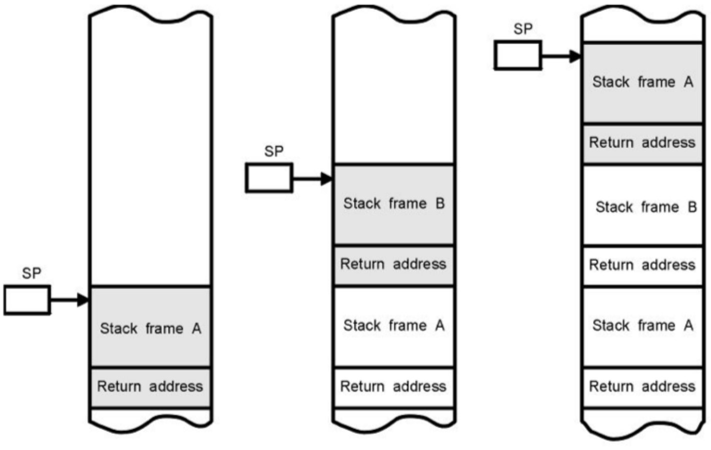


When function call happens previous variables gets stored in stack



Returning values from base case to caller function

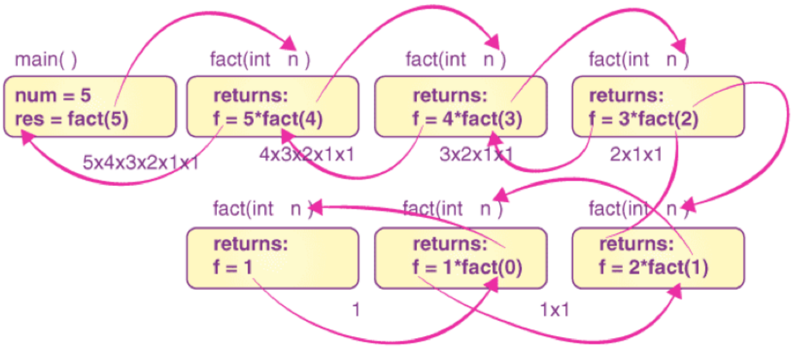


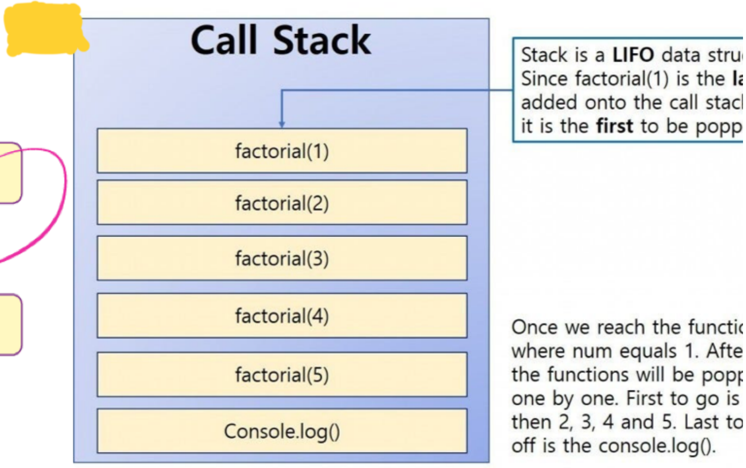


a. The state of the stack during subroutine A

b. The state of the stack during subroutine B

c. The state of the stack during a second call to subroutine A





Stack is a **LIFO** data stru
Since factorial(1) is the **la**
added onto the call stacl
it is the **first** to be popp

Once we reach the functi
where num equals 1. Afte
the functions will be popp
one by one. First to go is
then 2, 3, 4 and 5. Last to
off is the console.log().

Recursion in Java

1 What is Recursion?

Recursion is a programming technique where a function calls itself to solve smaller instances of a problem. It is a powerful tool for problems that have a repetitive or nested structure.

2 Relation to Functions

Recursion is built entirely upon functions. A recursive function:

- Calls itself directly or indirectly.
- Uses the call stack to track function calls.

3 Why and Where Do We Use Recursion?

Common Use Cases

- Tree Traversals (Inorder, Preorder, Postorder)
- Graph Traversals (DFS)
- Divide and Conquer Algorithms (Merge Sort, Quick Sort)
- Dynamic Programming (Top-down with Memoization)
- Backtracking (N-Queens, Sudoku)
- Mathematical Computations (Factorial, Fibonacci)

4 Techniques to Write Recursive Functions

- **Base Case:** The stopping condition.
- **Recursive Case:** Function calls itself with a smaller input.
- **Progress:** Move towards the base case in every call.

General Template

```
// Java
public void recursiveFunction(Parameters) {
    if (baseCaseCondition) {
        return; // stop recursion
    }
    // process
    recursiveFunction(smallerInput);
}
```

5 Common Pitfalls

- Missing base case: leads to infinite recursion.
- No input reduction: recursion never ends.
- Redundant calls: leads to inefficiency.
- Deep stack: leads to StackOverflowError.

6 Examples

1. Factorial

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

2. Fibonacci

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

3. Inorder Tree Traversal

```
void inorder(TreeNode root) {
    if (root == null) return;
    inorder(root.left);
    System.out.print(root.val + " ");
    inorder(root.right);
}
```

4. Reverse String

```
public static String reverse(String str) {
    if (str.isEmpty()) return str;
    return reverse(str.substring(1)) + str.charAt(0);
}
```

5. Tower of Hanoi

```
public static void solve(int n, char from, char to, char aux) {
    if (n == 1) {
        System.out.println("Move disk 1 from " + from + " to " +
           to);
        return;
    }
    solve(n - 1, from, aux, to);
    System.out.println("Move disk " + n + " from " + from + " to
       " + to);
    solve(n - 1, aux, to, from);
}
```

7 Stack Overflow and Infinite Recursion

When a recursive function lacks a base case, it calls itself forever, leading to a stack overflow.

Example

```
public static void infiniteRecursion() {
    System.out.println("Hello");
    infiniteRecursion();
}
```

Output:

```
Hello
Hello
...
Exception in thread ”main” java.lang.StackOverflowError
```

8 Real-Life Applications

- File system traversal
- JSON/XML parsing
- AI state space exploration
- Recursive math functions in engineering models

9 Best Practices

- Always define a clear base case.
- Use memoization or dynamic programming if overlapping subproblems exist.
- Consider converting to iteration if recursion is too deep.