Recursion → When a function calls itself.
↳ When a bigger problem solution depends on small & same type problem.

```
Void solve(){
    solve()
}
```

3

Big Prob

$2^5 \longrightarrow 32$

↓

B.P → 2^5

↓

2×2^4

↳ Small problem

↳ $2^n = 2 \times 2^{n-1}$

↳ Small prob.

Bigger Problem

↳ $f(n) = 2 \times f(n-1)$

→ Recursive Code

- Base condition → mandatory
- Recursive relation / Recursive call → mandatory
- Processing (adding, multiplying etc) → optional

Ex →

```
int Factorial (int n) {
    cout << "Fact. is " << n << endl;

    //base case
    if (n==1)
        return 1;
    int ans = n * fact(n-1); // Recursive call
    return ans;
}
```

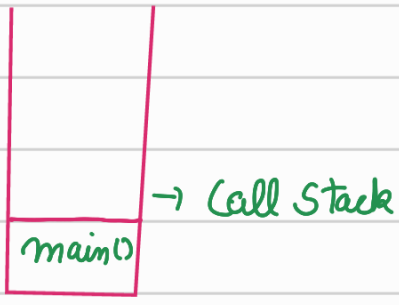
⇒ Call stack For recursion :-

```
int main(){
    int n;
    cin >> n;
Line-1  printCounting(n);
}
```

```
void printCounting(int n){
    if(n==0)                    Line-2
        return
    cout << n << " ";           Line-3

    printCounting(n-1);         Line-4
}
```

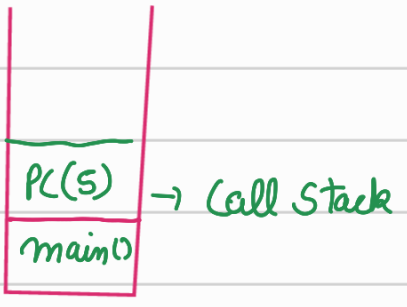
Initially we will have main() in C.S.

↳ Suppose n=5

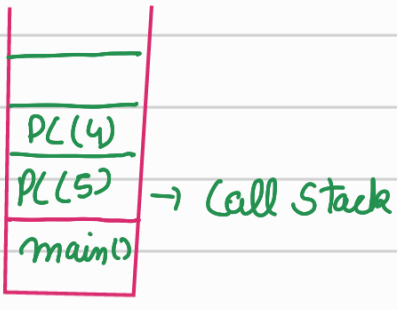


↳ At line-1, pC(5) is called so it's entry will be added to C.S →

↳ At line-3, it will print '5'

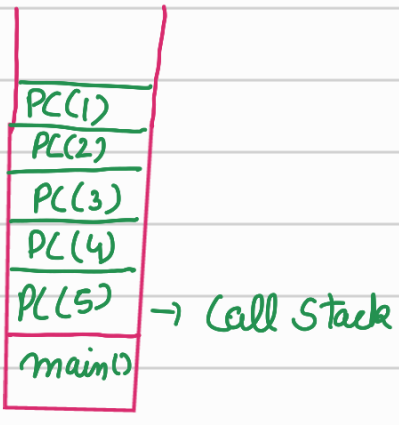


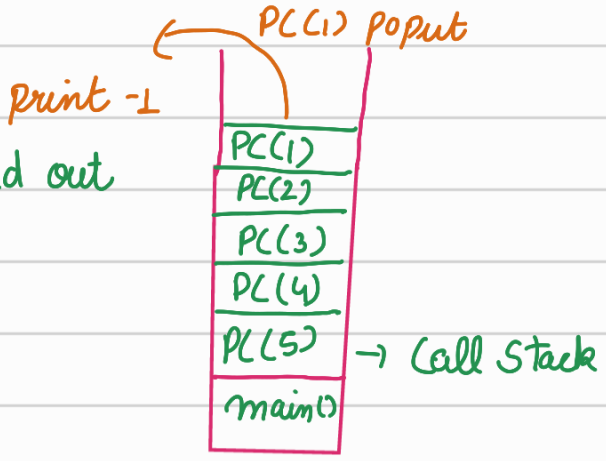
↳ At line-4, it will add p.C(4) to callstack



Similarly all the func will be added to the C.S

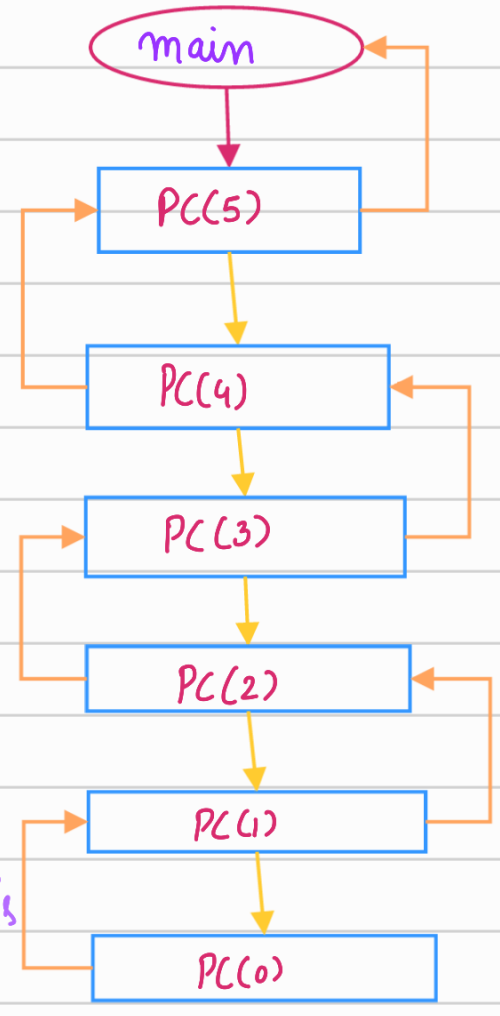
When we reach to last func., it will print (1) & pop out from stack





Similarly other functions will be popped out

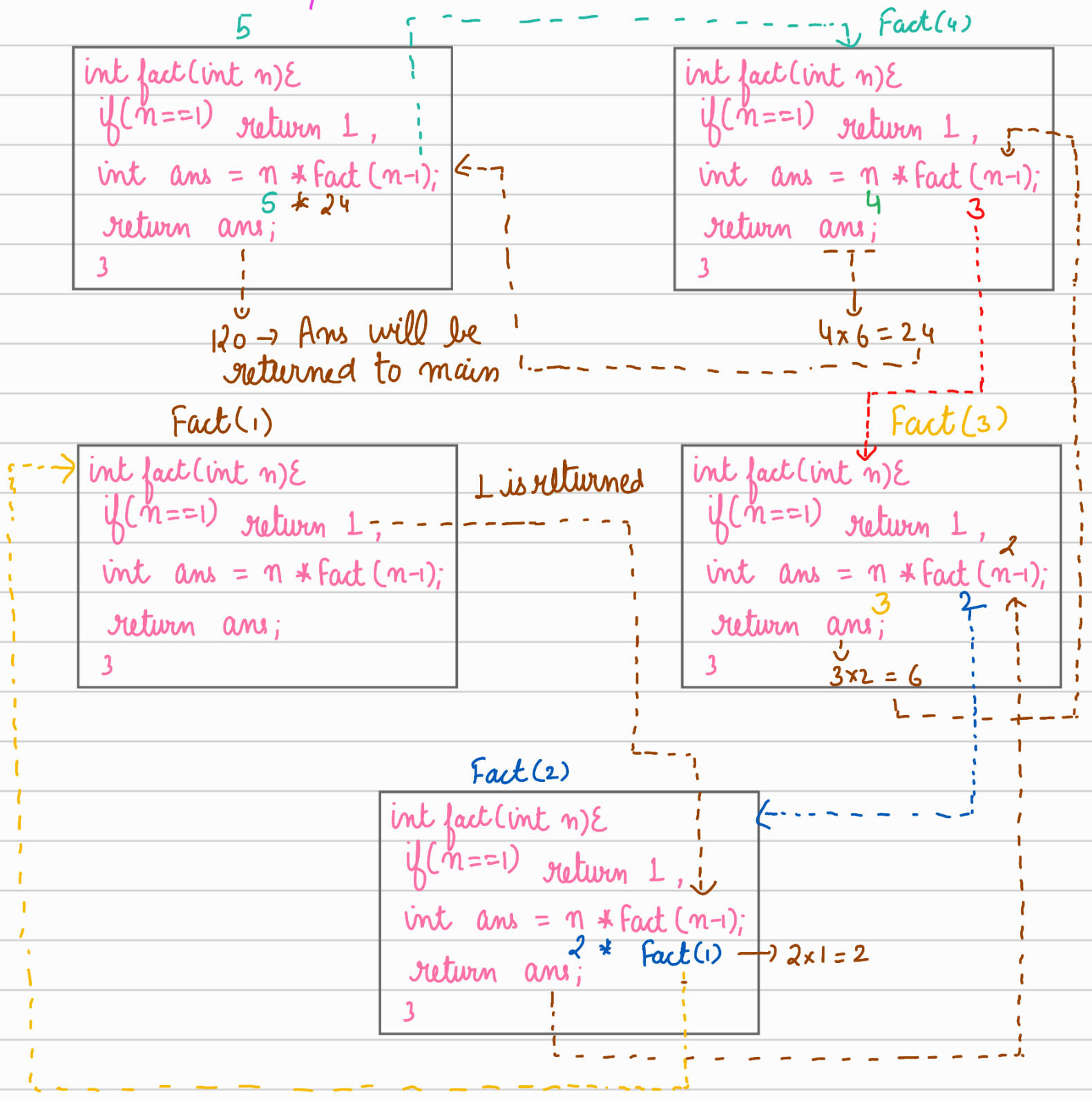
Recursive tree :-



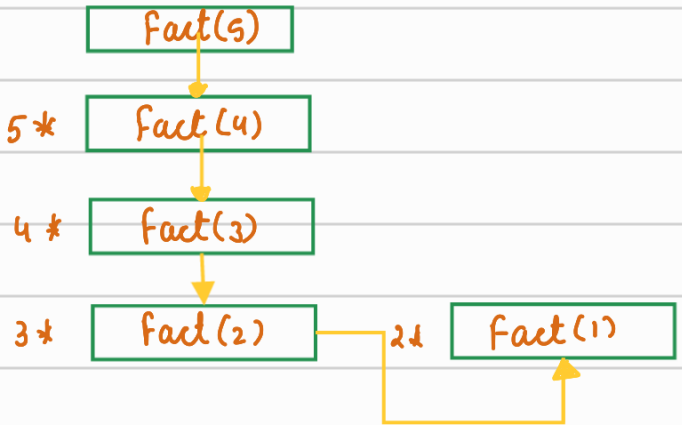
After completion
PC(0) will return it's
execution of PC(1)

⇒ Factorial of a number :-

Call stack explanation



Recursive tree :-



Head / Tail Recursion → If recursion is at last, it's called tail recursion & if recursion is before processing it's head recursion

```
int main ()
{
    1) b.c
    2) processing
    3) Recursion
}
```

↓ tail Rec.

```
int main ()
{
    1) b.c
    2) Recusion
    3) processing
}
```

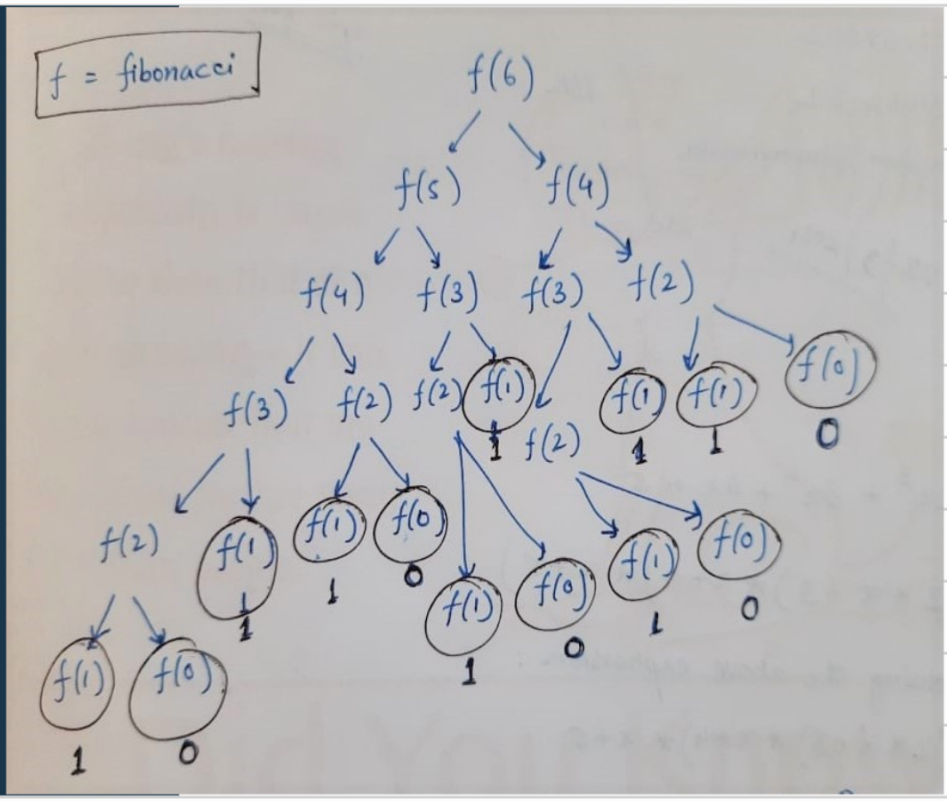
↓ Head recursion

fibonacci series :-

Base case → n=0, return 0

n=1, return 1

```cpp
#include <iostream>
using namespace std;
int fib(int n){
    //base case
    if(n==1){
        //fist term
        return 0;
    }

    if(n==2){
        //second term
        return 1;
    }

    //RR
    return fib(n-1) + fib(n-2);

}

int main()
{
    int n;
    cout << "Enter the term you want to see "<<endl;
    cin >> n;

    int ans = fib(n);
    cout << "The fibonacci term is "<< ans << endl;

    return 0;
}
```



Magic line → 1 case solve krdo baaki recursion sambhal lega