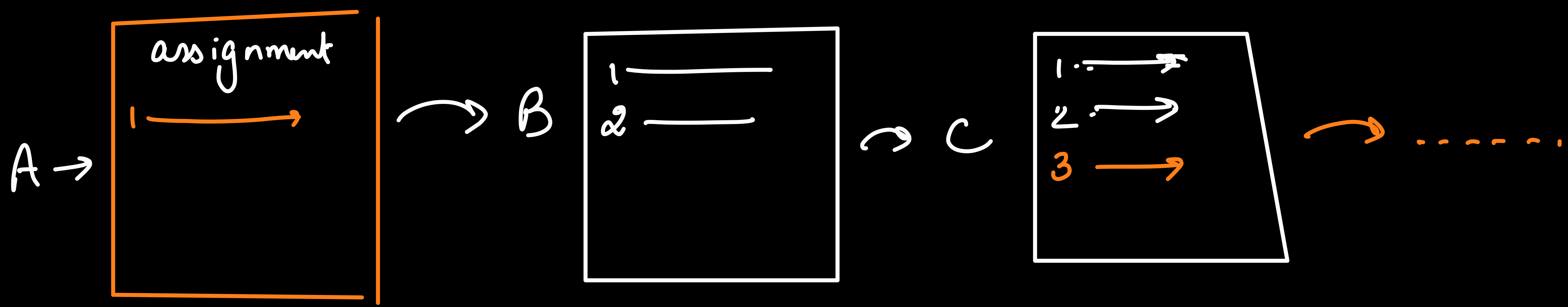
Recursion

A child couldn't sleep, so her mother told her a story about a little frog,
who couldn't sleep, so the frog's mother told her a story about a little bear,
who couldn't sleep, so the bear's mother told her a story about a little weasel...
who fell asleep.
...and the little bear fell asleep;
...and the little frog fell asleep;
...and the child fell asleep.



Factorial

$n \geq 0$

$n! = n \times (n-1) \times (n-2) \times (n-3) \ldots\ldots\ldots \times 3 \times 2 \times 1$

$5! = 5 \times 4 \times 3 \times 2 \times 1 \rightarrow 120$

$3! = 3 \times 2!$

$\boxed{0! = 1}$

$\boxed{7! = 7 \times 6!}$

7×720

$4! = 4 \times 3 \times 2 \times 1 = 24$

$6! = 6 \times 5 \times 4 \times 3 \times 2 \times 1$

$\boxed{5! = 5 \times 4!}$

$\boxed{6! = 6 \times 5!}$

$2! = 2 \times 1!$

5040 ← 7! ← 720 ← 6! ← 120 ← 5! ← 24 ← 4! ← 6 ← 3! ← 2 ← 2! ← 1 ← 1!

You $\xrightarrow{6!}$ A $\xrightarrow{5!}$ B $\xrightarrow{4!}$ C $\xrightarrow{3!}$ D $\xrightarrow{2!}$ E $\xrightarrow{1!}$ F

$\boxed{1! = 1}$

trivial

Recursion is not just a programming topic, but the discussion about recursion, you can find in maths also

↳ Discrete Maths

if we talk about recursion in programming terms then, it is a technique using which we write a func^n, that denotes a bigger problem, then we call that func^n inside itself that represent smaller problems. (we call ~~the~~ same func^n with mostly diff params)

a function calling itself

bigger problem of size n.

$$f(n) = f(n')$$

represents smaller subproblem

$n' < n$

PMI (Principal Of Mathematical Induction)

↳ It is prooving technique.

↳ To proove a formula correct –

① PMI checks the answer for the most trivial value.
(Base Case)

② PMI assumes that the formula is correct for some value k
(Assumption)

③ Then PMI prooves the formula for one more term apart from k, may be $k+1$ or $k-1$ etc.
(Self Work)

Q => Proove that sum of first N natural no:s is

$$\frac{N \times (N+1)}{2}$$

N=1

→ Natural NO → 1, 2, 3, 4 .... N

base case

① The most $\boxed{\text{trivial value}}$ is generally something for which we already know the ans.

→ N=1 for N=1, we know the ans will be 1.

Now let's verify that whether the formula works for N=1 or not:

$$f(N) = \frac{N \times (N+1)}{2} \rightarrow f(1) = \frac{1 \times (1+1)}{2} = \frac{1 \times \cancel{2}}{\cancel{2}} \Rightarrow 1$$

(2) This is the step of assumption. Here we assume without any calc that formula is correct for some input value k.

$$f(N) = \frac{N \times (N+1)}{2}$$

assume formula works correctly for $N = K$. (some term k)

$$f(K) = \frac{K \times (K+1)}{2}$$

→ we are assuming this is the correct value for $N = K$

(3) In this step using the assumption of prev step we try to calculate ans for one more term.

let's proove formula works for $K+1$ also.

$$f(K+1) = \underbrace{1 + 2 + 3 + 4 + \dots\dots\dots (K-1) + (K)}_{\text{Sum of first } K \text{ natural no.}} + (K+1)$$

$$f(K+1) = \frac{K \times (K+1)}{2} + (K+1) \quad (\text{taking } (K+1) \text{ common})$$

$$= (K+1)\left(\frac{K}{2} + 1\right) \rightarrow (K+1)\left(\frac{K+2}{2}\right)$$

$$= \frac{(K+1)(K+2)}{2}$$

$$f(N) = \frac{N \times (N+1)}{2}$$

$$f(K+1) = \frac{(K+1)(K+1+1)}{2}$$

$$= \frac{(K+1)(K+2)}{2}$$

H.P

$\Rightarrow$ Base $\quad N = 1 \rightarrow$ definitely correct

$K \longrightarrow 4 \rightarrow f(4) = \dfrac{4 \times (4+1)}{2} \Rightarrow 2 \times 5 \Rightarrow 10$

$(K+1) \longrightarrow 5 \rightarrow f(5) = 1+2+3+4+5$

$= 10 + 5 \Rightarrow 15$

$\dfrac{5 \times (5+1)}{2} \rightarrow \dfrac{5 \times 6}{2} \rightarrow 5 \times 3 = 15$

$\boxed{K \rightarrow 3} \rightarrow f(3) = \dfrac{3 \times (3+1)}{2} \rightarrow 3 \times 2 = 6$

$(K+1) \rightarrow f(4) = 1+2+3+4 \rightarrow 6+4 = 10$

$\hookrightarrow \dfrac{4 \times (4+1)}{2} \rightarrow \dfrac{\cancel{2}4 \times 5}{\cancel{2}} \rightarrow 10$

$K \to 2 \to f(2) \to \frac{\cancel{2} \times (2+1)}{\cancel{2}} \to \underline{\underline{3}}$

$(K+1) \to f(3) = \underline{\underline{1+2+3}} \to 3+3 \to \underline{\underline{6}}$

$\to \frac{3 \times (3+1)}{2} \to 3 \times 2 = 6$

$K \to 1 \to f(1) \to \underline{\underline{1}}$

$K+1 \to f(2) = \underline{\underline{1}} + 2 \to 1+2 = 3$

$f(2) \to \frac{\cancel{2} \times (2+1)}{\cancel{2}} \to \underline{\underline{3}}$

Qn Proove that $\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \ldots\ldots + \frac{1}{2^n} = 1 - \frac{1}{2^n}$

$\forall\, n \in$ Natural no:'s

$f(n) = 1 - \frac{1}{2^n}$

$\hookrightarrow \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} \ldots \frac{1}{2^n}$

$n = k+1$

$1 - \frac{1}{2^{k+1}}$

① Base Case

for $N=1$, we know that ans well be $1/2$

Let's verify $\rightarrow$ $f(1) = 1 - \frac{1}{2^1} \Rightarrow 1 - \frac{1}{2} \rightarrow \frac{2-1}{2} \Rightarrow \frac{1}{2}$

② Assumption

let's assume formula works well for $N = K$

$$f(K) = 1 - \frac{1}{2^K}$$

③ Self work

→ let's try to manually prove using the assumption that formula works correctly for $K+1$ also

$$f(K+1) = \underbrace{\frac{1}{2} + \frac{1}{4} + \frac{1}{8} \cdots\cdots \frac{1}{2^{K-1}} + \frac{1}{2^K}}_{f(K)} + \frac{1}{2^{K+1}}$$

$$f(k+1) = \left(1 - \frac{1}{2^k}\right) + \frac{1}{2^{k+1}}$$

$$f(k+1) = 1 - \frac{1}{2^k} + \frac{1}{2^k \times 2}$$

$$f(k+1) = 1 - \frac{1}{2^k}\left(1 - \frac{1}{2}\right) \Rightarrow 1 - \frac{1}{2^k}\left(\frac{2-1}{2}\right)$$

$$f(k+1) \Rightarrow 1 - \frac{1}{2^k}\left(\frac{1}{2}\right)$$

$$f(k+1) = 1 - \frac{1}{2^{k+1}}$$

H.P

Q: Given a value n, calc n! recursively

$$f(n) = \overbrace{n \times f(n-1)}^{\text{self work is multiplication}}$$

this func^n calculates n!

assume f(n-1) correctly gives (n-1)!

Base Case → $f(1) = 1$ (n=1) →

```
if (n == 1) {
    return 1;
}
```

Assumption →

$n!$

assume for $n = (k-1)$ the formula work correct

$f(k-1) \longrightarrow (k-1)!$

lets proove for $n = k$

$$f(k) = \underbrace{1 \times 2 \times 3 \times 4 \ldots\ldots (k-2) \times (k-1)}_{f(k-1)} \times k$$

$$\boxed{f(k) = k \times f(k-1)}$$

$k! = k \times (k-1)!$

```cpp
// factorial recursive // f(n) = n * f(n-1);
int f(int n) {
    // base case
    if(n == 1) return 1;

    return n * f(n-1);
}
```

bigger problem

smaller problem

main()

f(4)

f(5) → 120

f(4) 24

f(3) 6

f(2) 2

f(1) 1

f(4) f(4-1) f(3-1) f(2-1)

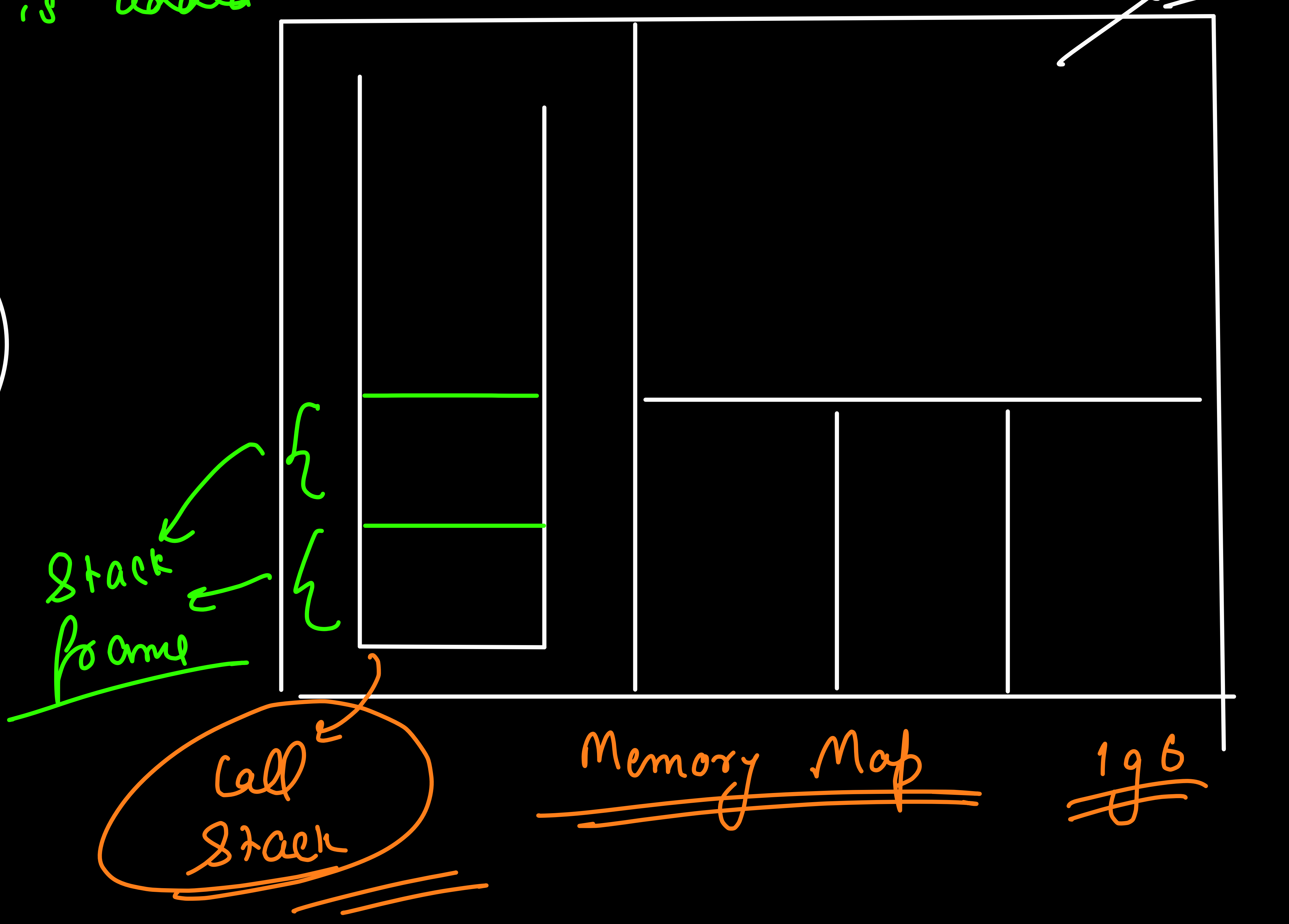
You → A → B → C - - -

whenever you call a func^n from anywhere, a new entry called as stack frame is added to the call stack.

local variables

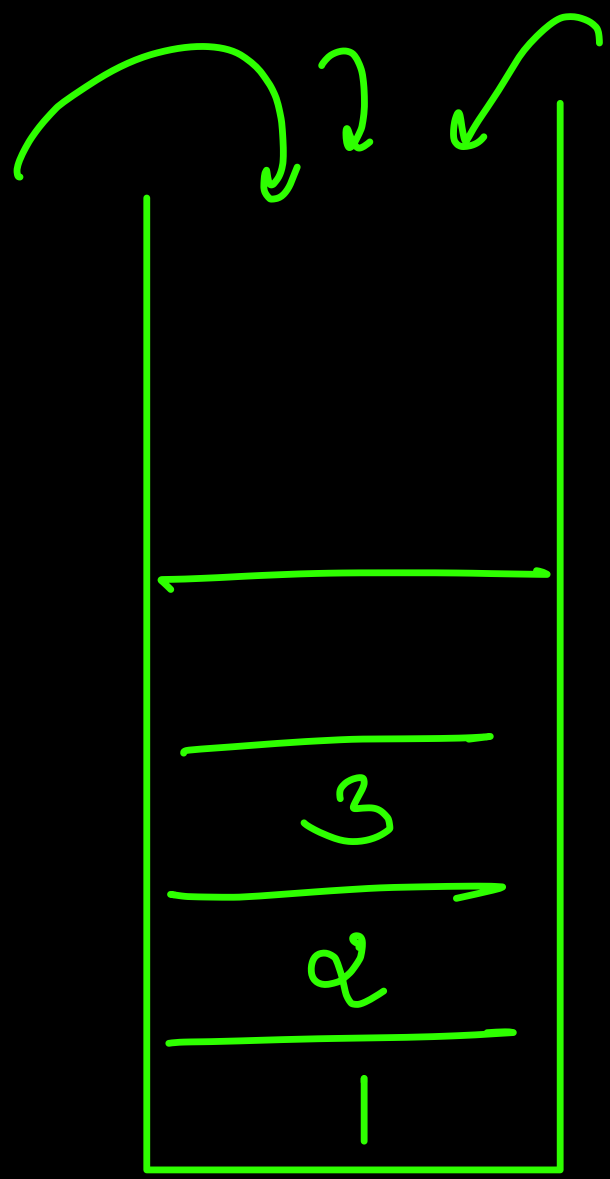
what lines are executed

⋮



once the func^n hits return or all lines are executed, the stack frame is removed showing func^n completion.

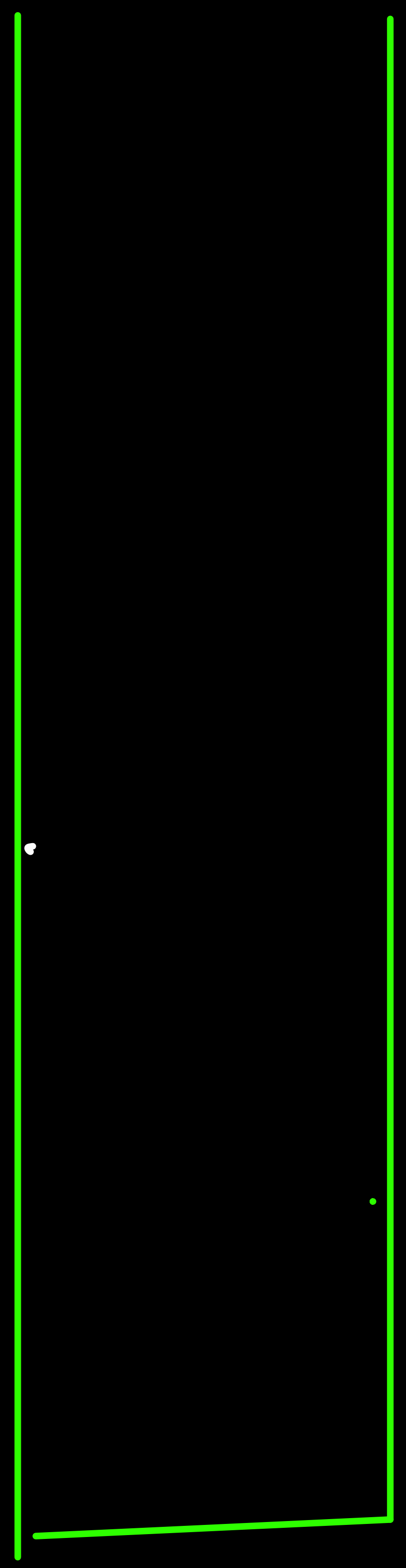
1, 2, 3, 4 X



```
void f() {
    int i=0
    cout<<i<<"\n";
}

void g() {
    int j=0;
    cout<<j;
}

int main() {
    f()
    g()
    return 0;
}
```



0
0