Recursion

Good Evening

Today's content
- → Function call tracing
- → Recursion
- → How to write a recursive code?
- → Tracing the recursive code
- → TC/SC of recursive code } Next class

Why recursion ?
- → Used in Mergesort / Quicksort
- → Binary Tree / BST / Tries
- → Dynamic programming
- → Back tracking
- → Graphs

Function call tracing

```
int add(n,m){
    return n+m;
}
```

```
int mul(x,y){
    return x*y
}
```

```
int sub(x,y){
    return x-y;
}
```

```
main(){
    x=10 , y=20
    print(sub(mul(add(x,y),30),75));
}
```

825

900

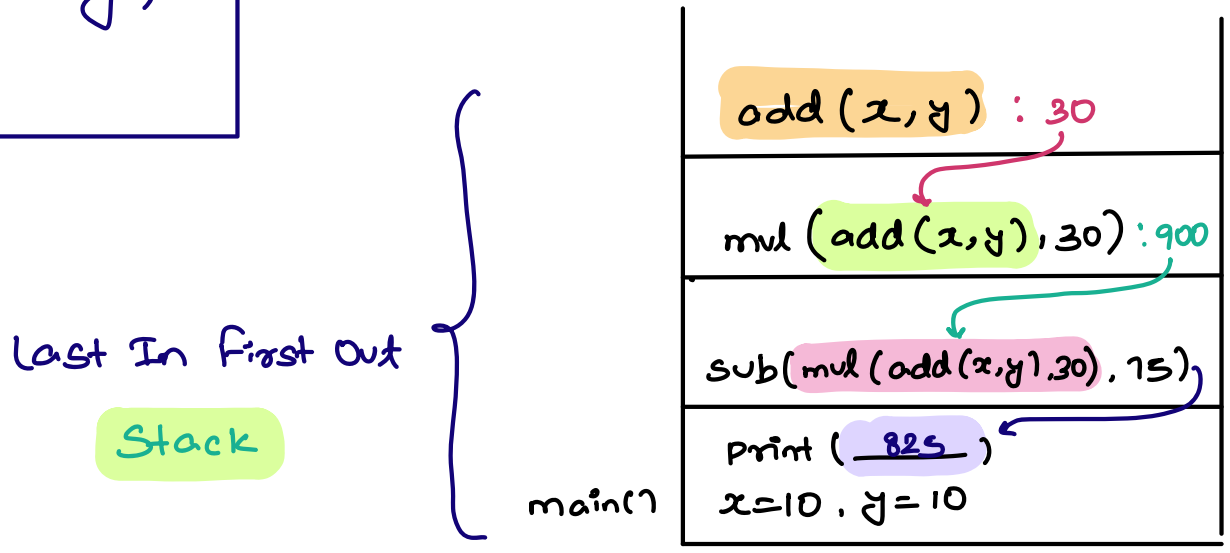
sub(mul(add(x,y),30),75); = 825

30

mul(add(x,y),30) = 900

add(x,y) = 30

10, 20



Last In First Out

Stack

Obs 1 → Whenever we add a function, it gets added at top

Obs 2 → Whenever a function returns something, it gets wiped out of the memory (At top)

Recursion ⟶ A function calling itself

↳ Repetitive code

↳ Solving a problem using a smaller instance of problem

Subproblem

01. Sum of n natural number

sum(4) = 1+2+3+4

sum(5) = 1+2+3+4 + 5

sum(4) + 5

a subproblem

sum(n) = 1+2+3+ ... +(n-1) + n

sum(n) = sum(n-1) + n

steps to write Rearsive code

01. Assumption/ Expectation → Fix what our function supposed to do/ what is expected from the function

02. Mainlogic → Solving assumption using subproblem.

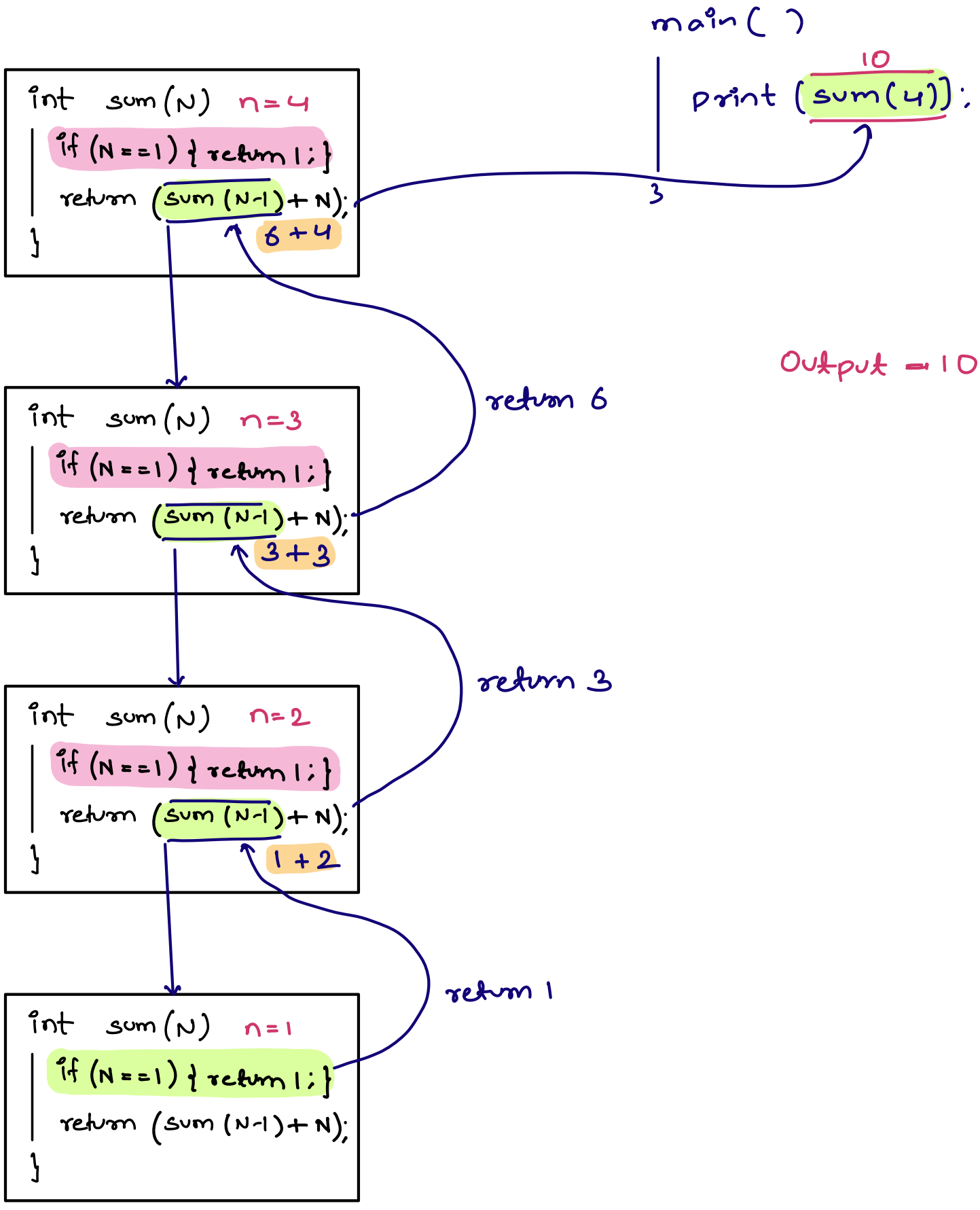
03. Base condition → Last valid input for which the recursion has to stop

01. Sum of n natural number $n>0$

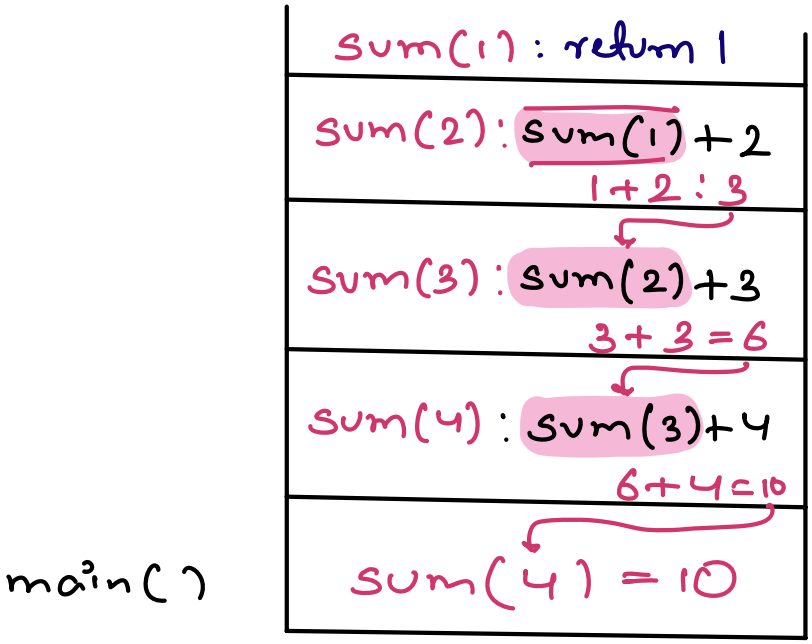
// Assumption → Given n, calculate & return the sum of 1 to n no.

```
int sum(n)
{
    if (n==1) return 1;

    return sum(n-1)+n

}
```

Tracing


main( )
print (sum(4));
10
int sum(N) n=4
if (N==1) { return 1; }
return (sum(N-1)+N);
6+4
3
int sum(N) n=3
if (N==1) { return 1; }
return (sum(N-1)+N);
3+3
int sum(N) n=2
if (N==1) { return 1; }
return (sum(N-1)+N);
1+2
int sum(N) n=1
if (N==1) { return 1; }
return (sum(N-1)+N);
return 6
return 3
return 1
Output = 10

Stack Tracing



Without base condition → Recursion will never stop

(a) TLE ✗

(b) Memory limit Exceeded / Stack overflow

Note :- Limit to function calls ≈ 10^5

Note :- If ever getting the stack overflow, check for base conditio

Base condition → Must be present at the top

02. Factorial of a number (n > 0)

fact(3) = 3 * 2 * 1 = 6

fact(4) = 4 * 3 * 2 * 1 = 24

fact(n) = n * (n-1) * (n-2) * ... * 1

fact(4) = 4 * fact(3)

// Assumption → Given n, calculate & return n!

```
int fact(n)
{
    if(n==1) return 1;

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

n ≥ 0

```
int fact(n)
{
    if(n==0) return 1

    return fact(n-1)*n
}
```

0! = 1

10:23 → 10:33 pm

Fibonacci numbers $(n \geq 0)$

# Input	0	1	2	3	4	5	6	7	8
Fib()	0	1	1	2	3	5	8	13	21

fib(n) = sum of two previous fib no.

$$fib(7) = fib(6) + fib(5)$$

$$fib(8) = fib(7) + fib(6)$$

$$fib(n) = fib(n-1) + fib(n-2)$$

// Assumption → Calculate & return n^{th} fib number

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

if (n==0) return 0;
if (n==1) return 1;

Base Case → Last valid input for which we want our main logic to stop

$fib(0) = fib(0-1) + fib(0-2)$

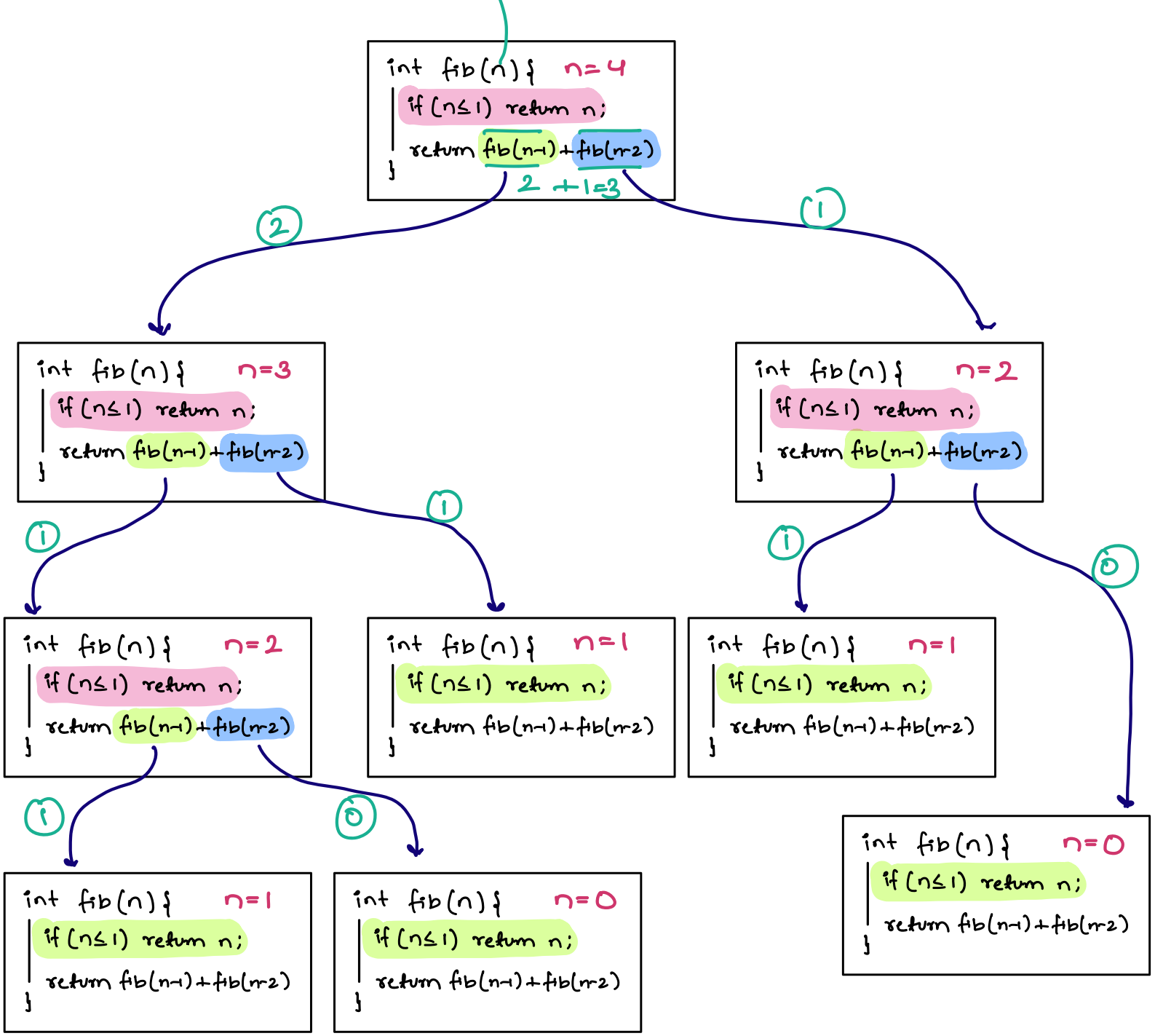
$fib(1) = fib(1-1) + fib(1-2)$

} for 0 & 1 main logic is invalid, make them as base condition

$fib(2) = fib(2-1) + fib(2-2)$

```
main(){
    print(fib(4)) ——→ 3
}
```

Tracing = 4



03. Given n, print all the numbers from 1 to N in increasing order (n>0)

Inc (3) ⟶ 1 2 3

Inc (4) ⟶ 1 2 3 4

Inc (3) 4

Inc (n) ⟶ 1 2 3 4 ... (n-1) n

Inc(n-1)

print(n);

// Assumption → Given n, print all the no. from 1 to N

```
void Inc (n);
|   if (n==1) { print(1);
|                return; }
|   Inc(n-1)
|   print(n);
}
```

Note:- Always return from the base condition

Tracing -

```
void Inc (n)  n=3
|   if (n==1) { print(1);
|                return; }
|   Inc(n-1)
|   print(n);
}
```

```
void Inc (n)  n=2
|   if (n==1) { print(1);
|                return; }
|   Inc(n-1)
|   print(n);
}
```

```
void Inc (n)   n=1
|  if (n==1) { print(1);
|               return; }
|  Inc(n-1)
|  print(n);
}
```

Output

1 2 3

Dec(4) ⟶ 4 3 2 1

Dec(n) ⟶ n n-1 n-2 n-3 . . . 1

TODO

05. Given a ch[], check if the given ch[] is palindrome from indexes s to e (s<e)

ch[] = { a m a z o n } s = 2, e = 4 — False
0 1 2 3 4 5

ch[] = { n a y a n } s = 0, e = 4 — True
0 1 2 3 4

ch[] = { g o o d d a d } s = 4, e = 6 True
0 1 2 3 4 5 6

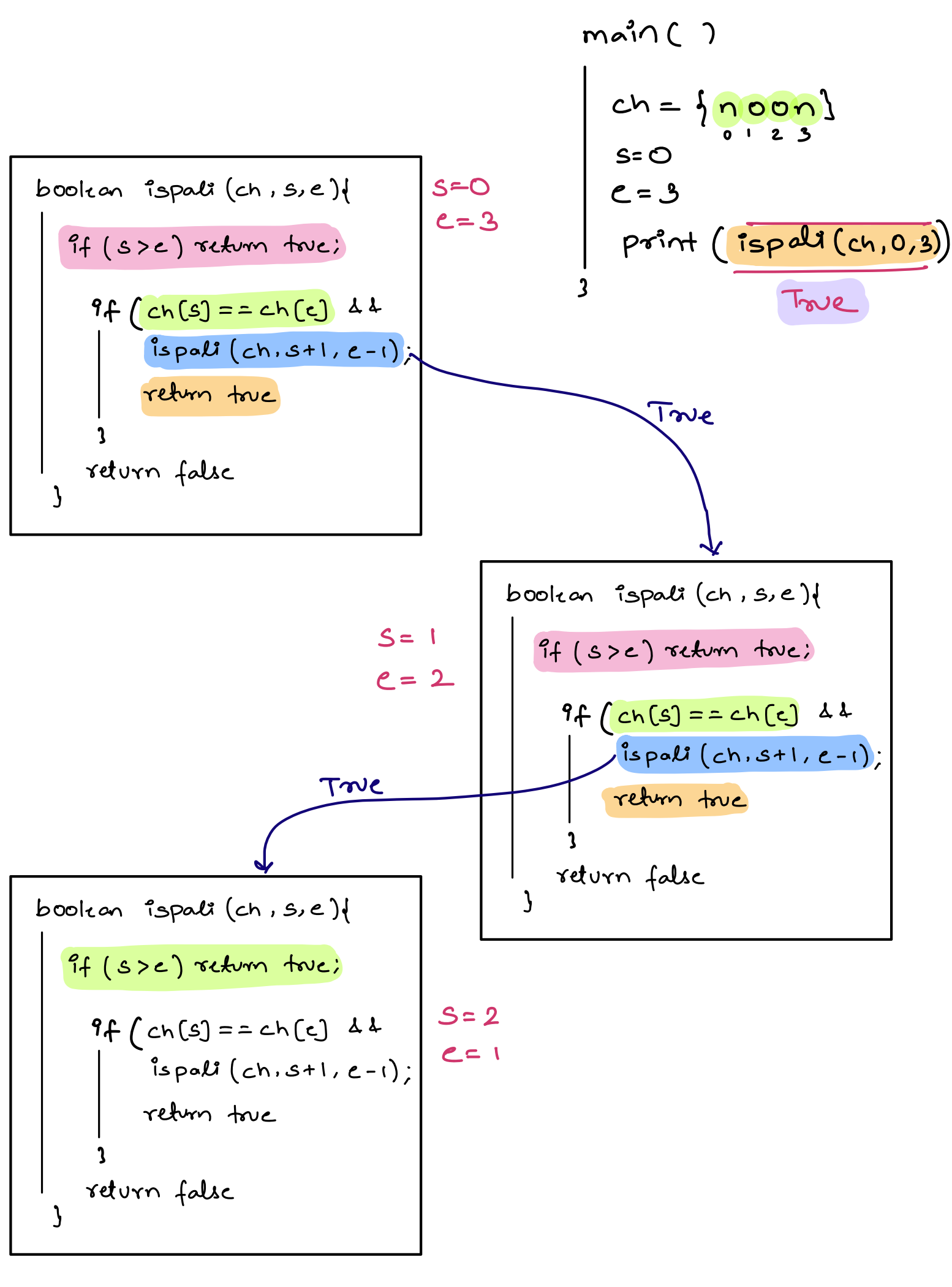
ch[] = { g o o d d a d } s = 2, e = 5 false
0 1 2 3 4 5 6

ch[] = { s s+1 ___ e-1 e }

ch[] = { m a d a m } s = 0, e = 4 True
0 1 2 3 4

ch[] = { m p c m } s = 0, e = 3 False
0 1 2 3

```
boolean ispali (ch[], s, e)
    if (s>e) return True :
    if (ch[s] == ch[e] && ispali(ch, s+1, e-1) == T)
        return True :
    3
    return false :
3
```


main( )
ch = {noon}
0 1 2 3
s=0
e=3
print (ispali(ch,0,3))
True
boolean ispali (ch, s, e){
if (s>e) return true;
if (ch[s] == ch[e] &&
ispali (ch, s+1, e-1);
return true
}
return false
}
s=0
e=3
True
boolean ispali (ch, s, e){
if (s>e) return true;
if (ch[s] == ch[e] &&
ispali (ch, s+1, e-1);
return true
}
return false
}
s=1
e=2
True
boolean ispali (ch, s, e){
if (s>e) return true;
if (ch[s] == ch[e] &&
ispali (ch, s+1, e-1);
return true
}
return false
}
s=2
e=1