1 February

RECURSION

Meaning → breaking a larger problem into same problem of smaller size

Imagine I want to find $n!$

$$n! = n \times (n-1) \times (n-2) \cdots \times 1$$

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

if we use loops for this that is called the Iterative approach

Code using loops →

Iterative Algorithm.

```
int factorial(int n){
    int i, ans;
    for(int i=1; i<=n; i++)
    {
        ans *= i;
    }
    return ans;
```

if (n==0) return 1;

0! = 1

Mandatory → base Case
↳ Reursive Case

DATE

Recursive Way →

We can write $n! \rightarrow n \times (n-1)!$

Code →

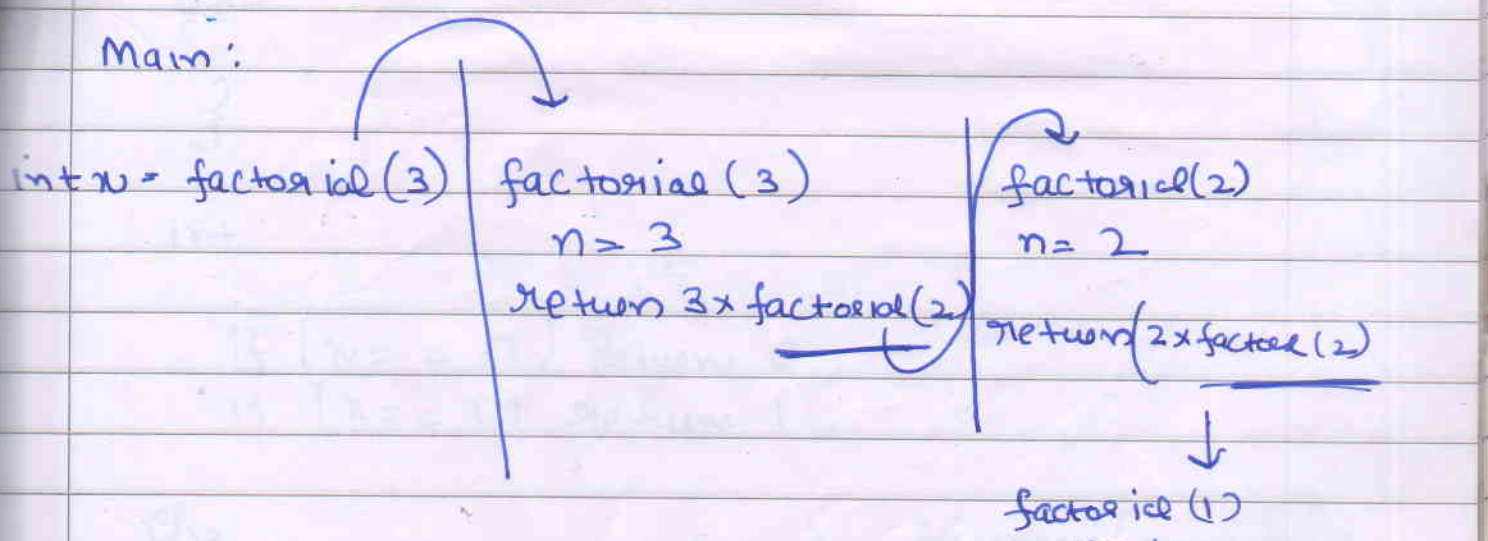
```
int factoraie( int n )
if (n == 0) {
    return 1 ;  // base Case
else
    return (n x factorial (n-1)) ;
```

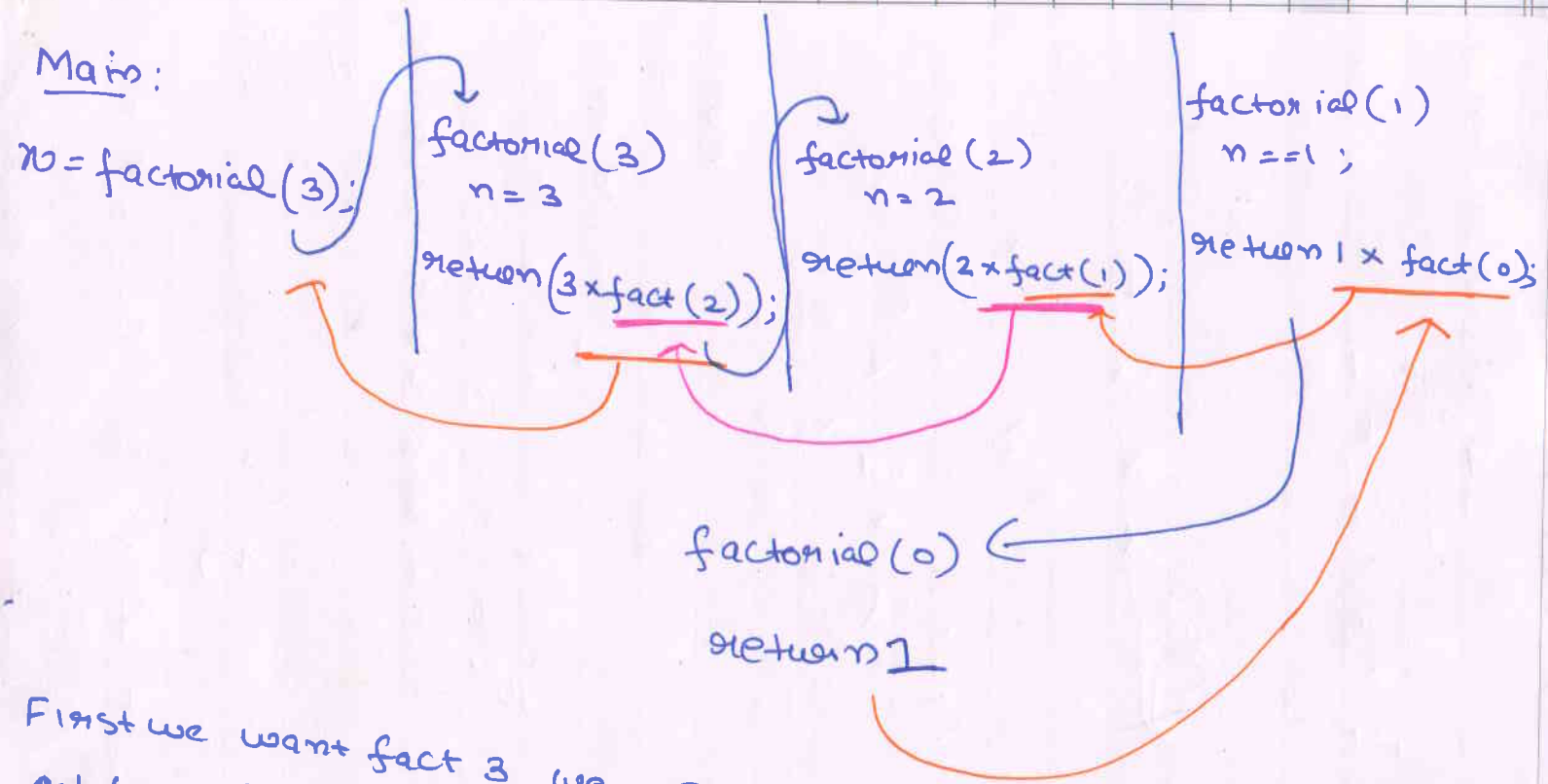
Explanation

5!

n x factorial (n-1)

5 x factorial (4)





First we want fact 3 we stored 3 x fact(2), fact(2) will get fact(1) and when f(0) comes we returned 1 the 1 goes to last one and value from last goes to second last Hence recursion works like winding and unwinding of spring

DATE

Questions on Recursion

Q1) print nth Fabbonacci number → Applied

Q2) print power of 2 → 2^n → babbar

Q3) print factorial of a number → Applied gain

Q4) print Counting 1 — n and n — 1

Q5) Sum of N natural number using Recursion → babbar Striver sheet

Solutions

Answer 1) Nth fabbonacci number

```
int main() {

  int x;
  x = fabbonacci(5);
}

int fabbonacci(int n) {

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

Else
  return (fabbonacci(n-1) + fabbonacci(n-2));
```

Main function

PAGE

DATE

Q Print Power of 2

$2^5 \rightarrow 2^4 \times 2^3 \times 2^2 \times 2^1 \times 2^0$

↓

base case / termination case

$2^n \rightarrow 2^{(n-1)} \times 2$

$2^5 = 2 \times 2^4$

$$2^n \rightarrow 2 \times 2^{(n-1)}$$ → Recursive logic

Code →

```
int main(){

  x = Power(5) ;  → 2^5
Cout << x << endl;
}

int power(int x){

if(x==0) return 1 → {2^0 → 1}

return(2 x Power(x-1));
```

DATE

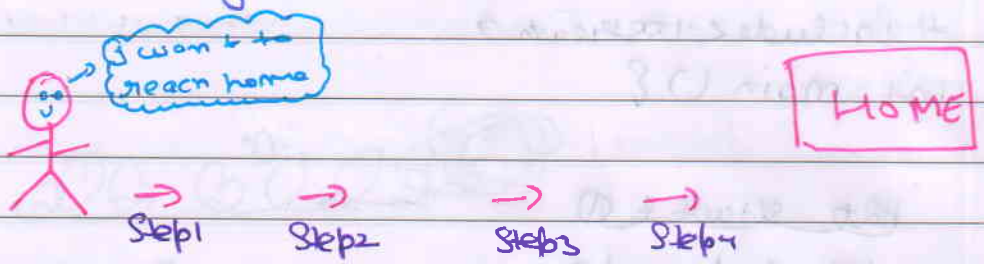
2 Print counting from 1 — 10 & 10 — 1 with the help of recursion.

```
#include<iostream>
int main(){

  to start & 10
  int end = 10

  Count(end);

}
int Count(int end){

  if(end == 0) return;   -> 10 - 1

  Cout << end << endl;
  end--;
  Count(end);

}

int Count(int end){

  Count(-end);

  Cout << end;

}
```

DATE

Lecture - 2

Recursion kya bolti hai



Ato to Recursion

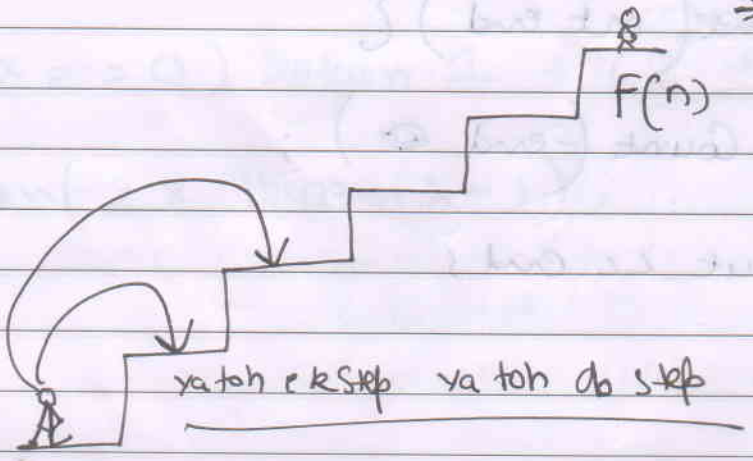
↳ Bhai ek step tera sikh le bakki recursion kardegi.

→ 1 step ee lo

→ Recursion use karlo

→ Rukna kab hai → base Case → Jab ghar pahunch jaye

Question ⇒ Climb Stairs ⇒ Count ways to reach n stairs → Code Studio



DATE

We need to think for last stair

↳ Hum last stair par kese aye hain

ya toh 1 chalang Mark $(n-1)$

ya to 2 chalang Mark $(n-2)$

Recursic Relation → $F(n) = F(n-1) + F(n-2)$

Base Case → if $(n < 0)$ return 0; } → Zero k niche to koi stair hi nai hai

if $(n == 1)$ return 1; } → Vo already 0 stair par hai So He ~~has~~ has one way ie vahi par jump laga le

0th seedi se 0th seedi par aana.

↓

Vapis dubara kud k aaya

Code → int CountStairs (int Stairs)

if (Stairs < 0) return 0;
if (Stairs == 0) return 1; } base Case

Recursive relation →

Return (Count Stair $(n-1)$ + Count stair $(n-2)$);

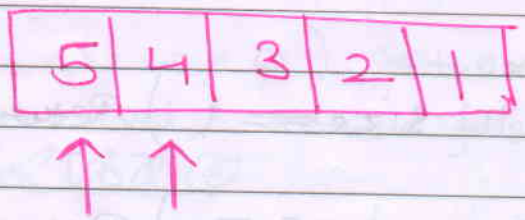
8 Feb, 2022

DATE

Recursion - 3

Ques → Check whether ARRAY is Sorted or not -?

Code given



We will Check for first 2 if true then Check for next two

if Not sorted → Return false!

Code → (Rough)

```
// base case →
if ((size == 0) || (size == 1))
    Return true;

if (arr[0] > arr[1]) return false

else

Call function ( arr+1 , size-1 )
                Next arr   ↓ with less size
```

Key point

classmate

PAGE

DATE

Code →

```
#include <iostream>
using namespace std;
bool isSorted(int arr[], int size){
    // base case
    if(size == 0 || size == 1) return true;
    if(arr[0] > arr[1]) Return false;
    Else return isSorted(arr+1, size-1);
```

Lenier Search Using Recursion

Code →

```
bool lenier_search(int arr[], int size, int target)
    if(size == 0) return false;
    if(arr[0] == target) return true;
    else
        return lenier_search(arr+1, size-1, target
```

Imp things to note

DATE

Binary Search using Recursion

Note → ARRAY Must be SORTed while Binary search

```
int binarysearch( int start, int end, int arr[], int key)
{
  if(arr.size() == 0) return 0;
  if( start > end) return false;
  int mid = s+e/2;

if(arr[mid] == key)
{
   Return mid + 1;
}
else if (arr[mid] > key) {
  binary Search ( start, mid-1, arr )
}
else if ( arr[mid] < key ) {
  binarySearch( mid+1, end, arr )
```