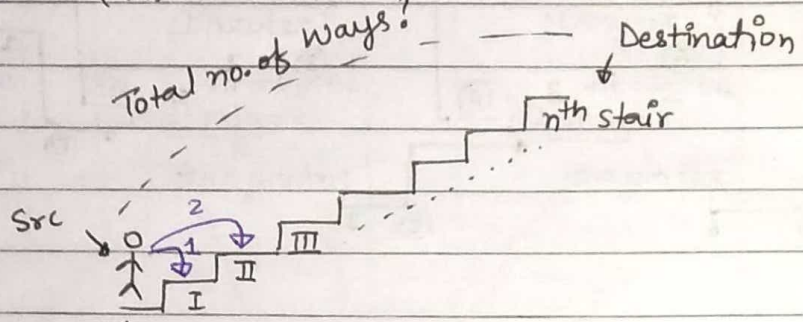
# RECURSION - CLASS 2

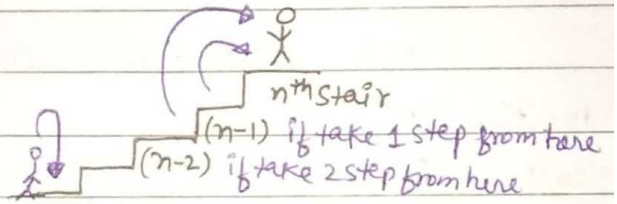
Date ..../..../.....

Q climbing Stairs (70) leetcode.




Can take either 1 step or 2 at once.

f(n) → no. of ways to reach $n^{th}$ stair

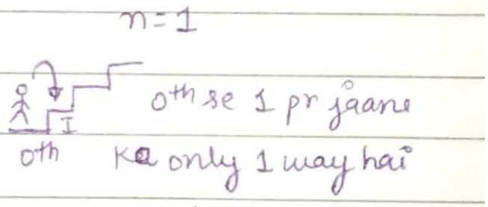



$$f(n) = f(n-1) + f(n-2)$$

R.R

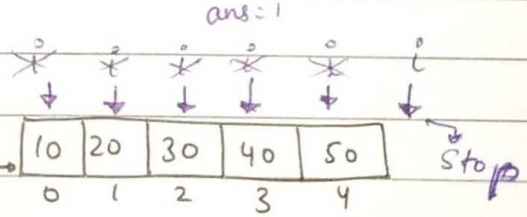
Code:-

```
int climbStairs (int n){
if(n==0) return 1;
if(n==1) return 1;
int ans = climbStairs(n-1) + climbStairs(n-2);
return ans;
```




# Arrays & Recursion:-

Q Print array elements using recursion.




array size index → Pass them in a funcn.

f(arr, 5, 0)
↓ print[10]
index++;

f(arr, 5, 1)
↓ Print 20
index++;

f(arr, 5, 2)
Print 30
index++;

f(arr, 5, 3)
print 40
index++

f(arr, 5, 4)
print 50
index++

Base case
index ≥ size
Rukjao

f(arr, 5, 5)

Date ..../..../.....

Q Search in Array.

i/p → arr →

| 10 | 20 | 30 | 40 | 50 |
|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 |

target = 50 return T/F.

↳ milne pr → return True

↳ entire array traverse hogya → return false

Code:-

```
bool searchArray(int arr[], int size, int target,
                 int index) {
    //Base Case
    if (index >= size) return false;
    //Processing
    if (arr[index] == target) return true;
    //Recursive Relation
    bool ans = searchAns(arr, size, target, index+1);
    return ans;
```

call from main()

searchArr(arr, 3, 30, 0)

if (index ≥ size) ✗ return false;
if (arr[Index] == target) 10 == 30 ✗
int ans = searchArr(arr, size, target, index+1); T
return ans;

searchArr(arr, 3, 30, 1)

if (index ≥ size) 1 ≥ 3 ✗ return false;
if (arr[Index] == target) 20 == 30 ✗
return true;
int ans = searchAns(arr, size, target, index+1); T
return ans;

searchArr(arr, 3, 30, 2)

if (index ≥ size) 2 ≥ 3 ✗ return false;
if (arr[Index] == target) 30 == 30
return true;
int ans = searchArr(arr, size, target, index+1);
return ans;

if (index ≥ size) return false;
if (arr[Index] == target)
return true;
int ans = searchArr(arr, size, target, index+1);
return ans;

T ← return ans; ← return ans; ← True

main → searchArr(arr, 3, 30, 0)

arr →

| 10 | 20 | 30 |
|---|---|---|
| 0 | 1 | 2 |

target → 30

Q find minimum no. in an array.

```
void minfinder(int arr[], int size, int index, int &mini){
    //Base Case
    if (index >= size) return;
    //Processing
    mini = min(mini, arr[index]);
    //Recursive Call
    minfinder(arr, size, index+1, mini);
}
```

KEEP IN MIND:-

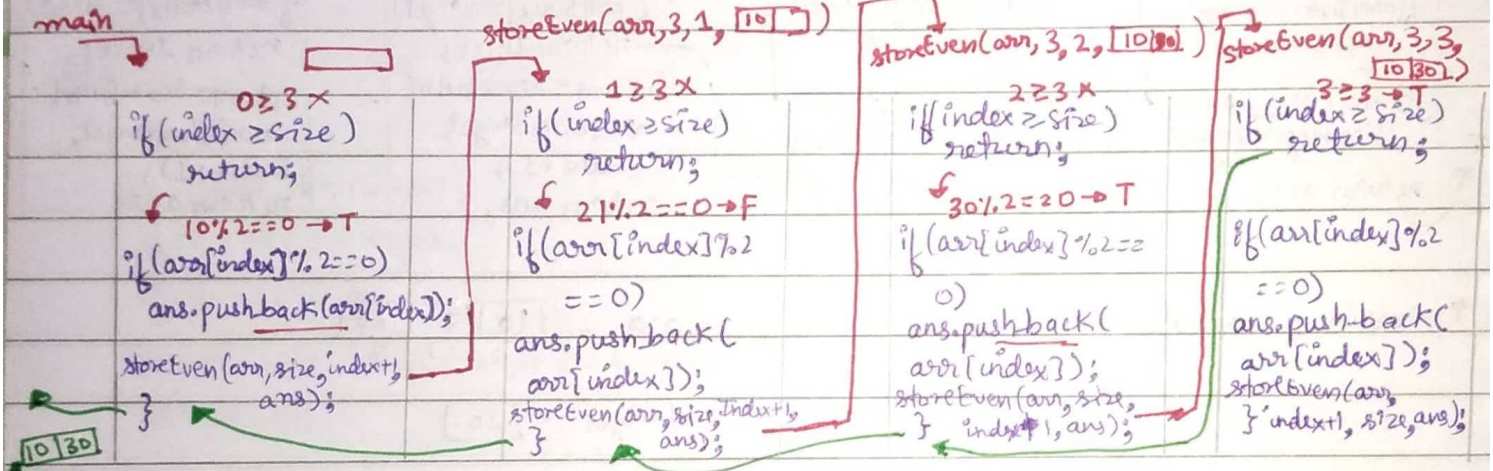
Tumhe agar kisi variable/DataStructure/Location ke andr answer store krna hai or tum usko kisi function me pass kar rahe ho or us function ke andr unme ans ka data store krna hai. So, make sure ki tumne us variable/DataStructure/Location ko BY REFERENCE pass kiya hai. Otherwise unki copy ban jygi or answer nhi store hoga usme.

Spiral

Date ..../..../.....

Q Store even numbers of an array in vector.

```
void storeEven ( int arr[], int size, int index, vector<int> &ans ) {
    // Base Case
    if ( index ≥ size ) return ;
    // Processing
    if ( arr[index] % 2 == 0 ) {
        ans.push_back ( arr[index] );
    }
    // Recursive Relation
    storeEven ( arr , size , index+1 , ans );
}
```




main → storeEven ( arr, 3, 0, ▭ )

array size Index vector<int> ans

| 10 | 21 | 30 |
|---|---|---|
| 0 | 1 | 2 |

Q You have an array in input. Double the values using recursion.

i/p → | 10 | 20 | 30 | 40 | 50 | → o/p | 20 | 40 | 60 | 80 | 100 |

```
void solve ( int arr[], int size, int index )
    // Base Case
    if ( index ≥ size ) return;
    // Processing
    arr[index] = 2 * arr[index] ;
    // Recursive Call
    solve ( arr, size, index + 1);
}
```

Spiral

Date ..../..../.....

Q Traverse the array using recursion and find maximum No.

```
void findMax (int arr, int size, int index, int &maxi ) {
    //Base Case
    if( index>= size) return;
    //Processing
    maxi = max ( maxi , arr[index ]);
    //Recursive Relation
    findMax ( arr, size , index+1, maxi);
}

int main(){
   arr[]= {10,20,30,40}; int size = 4; int index = 0;
   int maxi = INT_MIN;
   findMax ( arr , size, index , maxi);
}
```

Q find target in given array and return its index if its present and return -1 if it's not present.

```
int findTarget ( int arr[], int size, int index , int target) {
   //Base Case
   if( index >= size) return -1;
   // Processing
if(arr[index] == target)  return index;
   // Recursive Relation
   findTarget (arr , size, index+1, target );
}
```

Q Print index of all occurrence of target in Array.

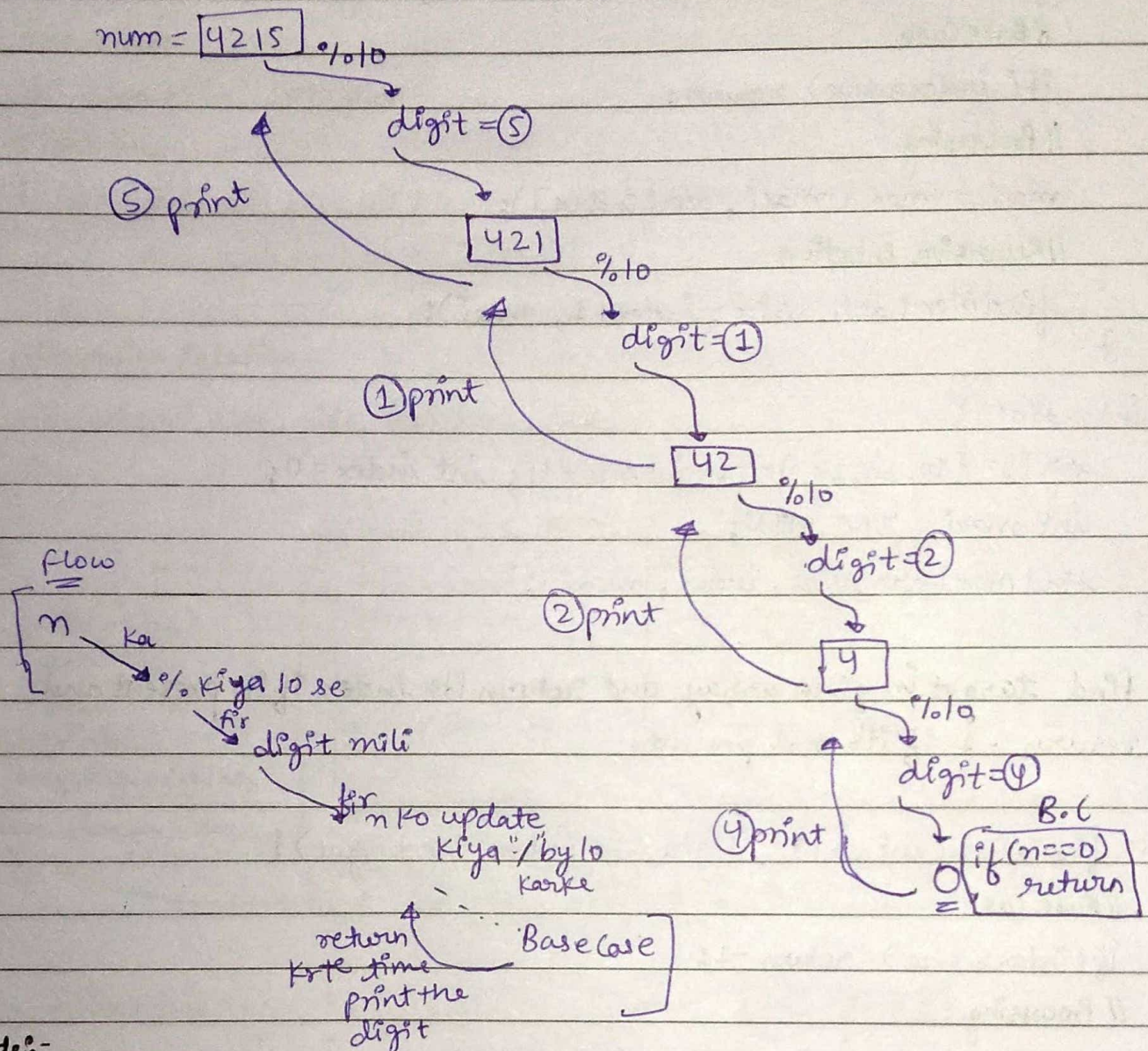
```
void printAllOccurrence( int arr[], int size, int index, int target){
   if ( index >= size) return ;   // Base Case
   if ( arr[index] == target) cout<< index<<" "; //Processing
   printAllOccurrence ( arr , size, index+1, target ); //R.R
}
```

Spiral

Date ..../..../.....

Q You have an integer in input and you have to print its digits.

Ex:-




Code:-

```
void printDigit(int num){
    if(num==0) return;      // Base Case
    int digit = num%10;     // Processing
    num = num/10;           // Updating num
    printDigit(num);        // Recursive Relation
    cout<<digit<<" "<<endl; // Processing
}
```

Date ..../..../.....

Q You have integer input. Store its digits into a vector.

```
void storeDigits ( int num, vector<int> & ans ){
    // Base Case
    if(num == 0) return;
    // Processing
int digit = num % 10;
    // Update num
    num = num / 10;
    // Recursive Relation
    storeDigits ( num, ans);
    // Processing
    ans.push_back(digit);
}
```

Q You have a vector in input. You have to return an integer using values of vector as digits for that integer.

i/p → vector → | 4 | 2 | 1 | 7 |  o/p → | 4217 | → Integer

```
void createInteger ( vector<int> v, int index, int &ans){
    //Base Case
    if ( index ≥ size ) return ;
    // Processing
    ans = ans *10 + v[index];
    //Recursive Call
    createInteger ( v, index+1, ans);
}
```

Date ..../..../...

Q You have a string in input and target character. Print all occurrences of the target character into the string.

Ex:- string s = "Ditij" target = 'i' o/p → 1 3
0 1 2 3 4

```
void printStrTargetIndex (string str, int index, char target){
    // Base Case
    if (index == str.length()) return;
    // Processing
    if (str[index] == target) cout<< index <<" ";
    // Recursive Relation
    printStrTargetIndex (str, index+1, target);
}
```