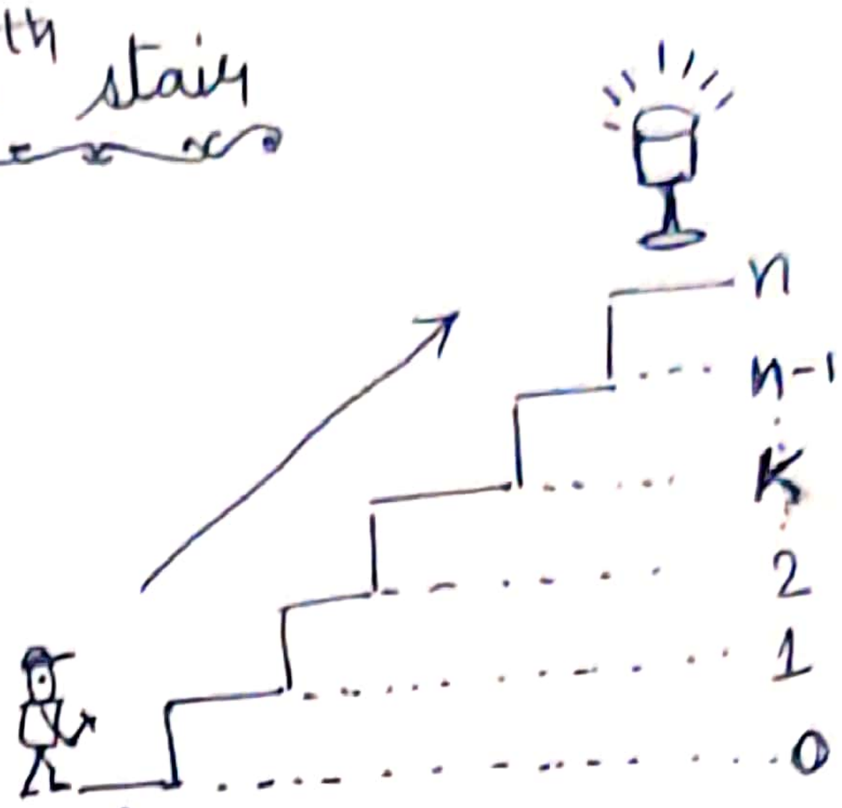
1Q) Climb Stairs Recursion

find number of ways to reach n^{th} stair



steps allowed

1) one step at a time

2) two steps at a time

Solution :-

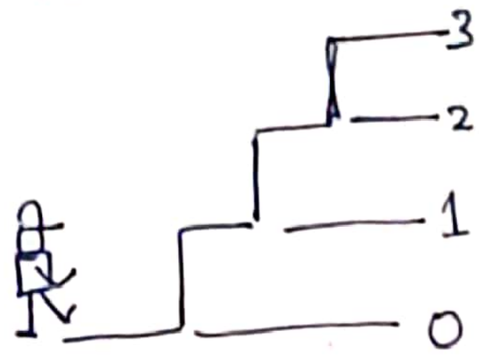
1) You can reach [n] from [n-2] by taking [2] steps!

(or)

2) You can reach [n] from [n-1] by taking [1] step!

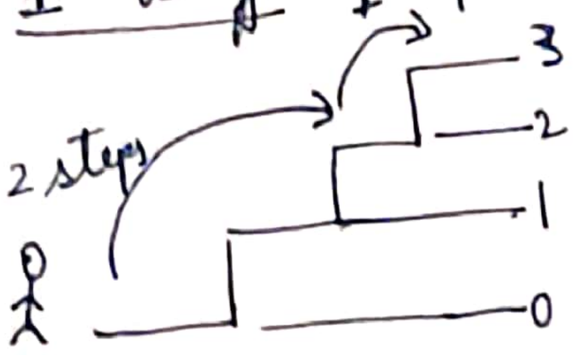
example :-

3 steps are there

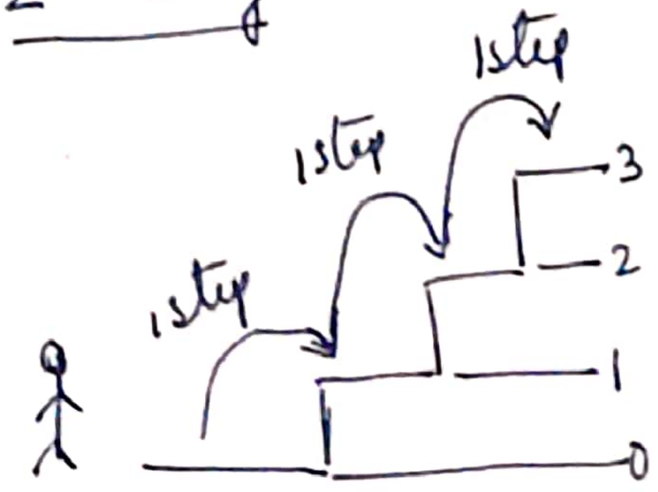


No. of ways :-

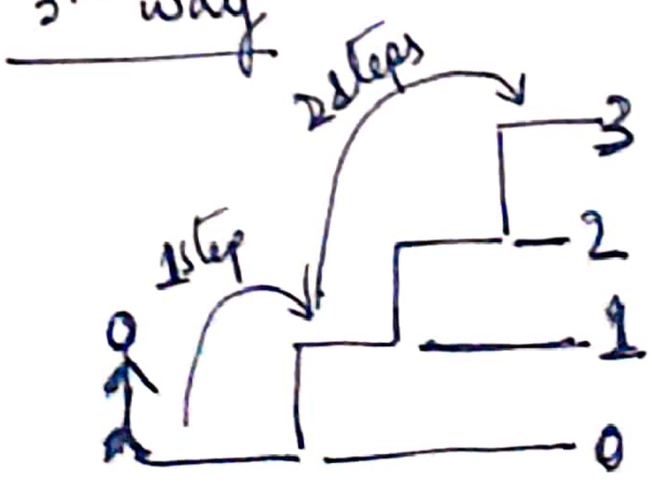
1st ways 1step



2nd way



3rd way



→ This is recursive formula to solve this problem.

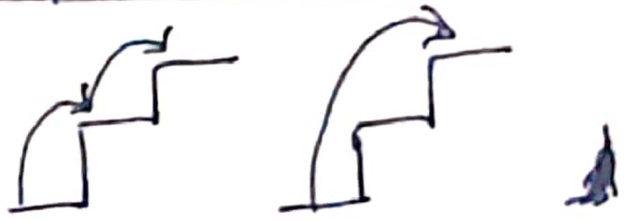
$$f(n) = f(n-1) + f(n-2)$$

→ lets see how this formula we deduced :-

→ if 1 step is there

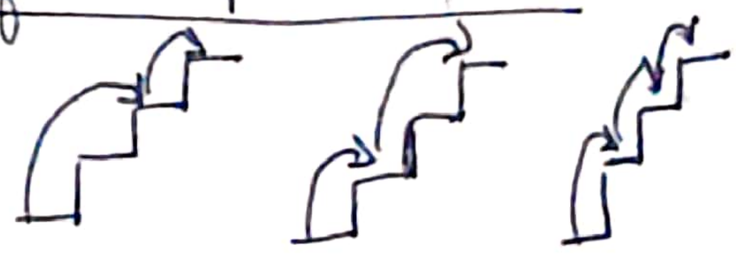
no. of ways = 1

→ if 2 step are there



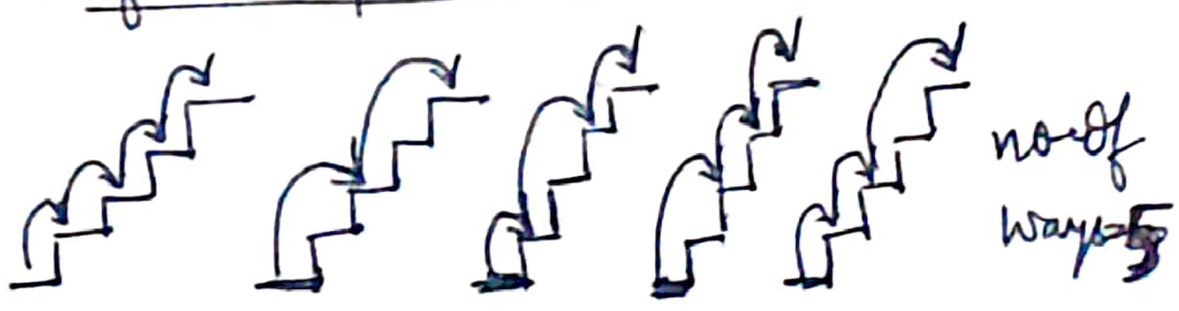
no. of ways = 2

→ if 3 steps are there



No. of ways = 3

→ if 4 steps are there



if you observe this falls in a pattern

no. of stairs	0	1	2	3	4
no. of ways	1	1	2	3	5

no. of ways is falling in fibonacci pattern (1, 1, 2, 3, 5 ...

$$f(n) = f(n-1) + f(n-2)$$ where $f(0)$ or $f(1) = 1$

code :-

```
#include <iostream>
using namespace std;
int climbStairs(int n)
{
    if (n==0 || n==1) // base condition
        return 1;
    int ans = climbStairs(n-1) + climbStairs(n-2);
    return ans;                                    //Recursive Call.
}
int main()
{
    int numberOfStairs = 4;
    int numberOfStep
    int numberOfWaysToReachNthStair = ClimbStairs(numberOfStairs);
    cout << numberOfWaysToReachNthStair << endl;
}
```

Solution decoding :- The pattern of results to reach a n^{th} stair are simillar to fibonacci series so simply we apply $f(n) = f(n-1) + f(n-2)$; this recursive call gives us the answer, but base condition we have to initialize $f(0)$ or $f(1) = 1$.

2Q) Print elements of a 1D array using recursion

Sol:- → we will maintain an index i to print which will increment for every call.

base condition

```
if(i >= (size of array))
{
  stop recursion;
}
```

dry run

arr = {1, 2, 3, 4, 5}
0 1 2 3 4

func(arr, n, index) → arr[0]
  ↳ func(arr, n, index+1) → arr[1]
    ↳ func(arr, n, index+1) → arr[2]
      ↳ func(arr, n, index+1) → arr[3]
        ↳ func(arr, n, index+1) → arr[4]
          ↳ func(arr, n, index+1)
            ↳ n == index+1 (recursion stops).

note:-

- Tail Recursion is used in this case to print (0 → n-1) of array
- Head Recursion prints (n-1 to 0) of array

Code:-

```
#include<iostream>
using namespace std;
void print1DArrayElements(int *A, int index, int size).
{
    if(index >= size)  // base condition
    {
        return;
    }
    cout << A[index] << " "; // 1 case solve karo!
    print1DArrayElements(A, ++i, size); // baki recursion solve kardega!
}
int main()
{
    int arr[]= {1,2,3,4,5};
    print(arr, 0, sizeof(arr)/sizeof(arr[0]));
}
```

30. Print maximum element in an array using recursion.

Solution :-

```
#include <iostream>
#include <limits.h>
using namespace std;
int findMax(int *A, int size, int index);
int main()
{
    int arr[] = {12, -34, 53, 67, -3};
    cout << findMax(arr, sizeof(arr)/sizeof(arr[0]), 0);
    return 0;
}
int findMax(int *A, int size, int index){
{
    static int ans = INT_MIN; // static value remains unchanged
    if (index >= size)    //base_Condition.
    {
        return ans;
    }
    if(ans < A[index])
    {
        ans = A[index];
    }                                  } //Processing
    return findMax(A, size, index+1); // Recursive Call.
}
```

→ Simply we are checking each element in array is greater than the value in ans variable, if greater ans is updated. once we reach the end of array size we return ans value which contains the greatest (Max) value which will be returned through all the recursive calls.

4Q. Print minimum element in 1D array using recursion.

```
#include <iostream>
#include <limits.h>
using namespace std;
int findMin(int *A, int size, int index)
{
    static int ans = INT_MAX; // static Value are
                              maintained in all
    if (index >= size) // base condition  recursions.
    {
        return ans;
    }
    if (ans > A[index])
    {
        ans = A[index];
    }                          //Processing
    return findMin(A, size, index+1); // Recursive
                                      call.
}
int main()
{
    int arr[] = {12, -34, 53, 67, -3};
    cout << findMin(arr, sizeof(arr)/sizeof(arr[0]), 0);
}
```

Solution decoding:- It is absolutely similar to the we we solved for Max element in (Q3) only thing is we compare answer with A[index] at each index for a ~~great~~ smaller element. if smaller element is found we store it inside ans variable.

5Q. find char 'r' is present in a string ?

Solution :-

```
1  #include <iostream>
2  #include <limits.h>
3  using namespace std;
4  bool FindCharInString(string s, int size, char c, int index);
5  int main()
6  {
7      string s = "lovebabbar";
8      cout<< FindCharInString(s, s.length(), 'r', 0);
9  }
10 bool FindCharInString(string s, int size, char c, int index);
11 {
12   if(index >= size)
13   {
14     return false;
15   }
16   if(c == s[index]){
17     return true;
18   }
19   else {
20     return FindCharInString(s, size, c, index+1); // Recursive Call
21   }
22 }
```

→ We are checking for char r at each ~~string~~ index of array(string) and validates if it matches with char 'r', if match is found before reaching the end of string we return true else we return false.

6Q. Print digits of a number using recursion?

Solution :-

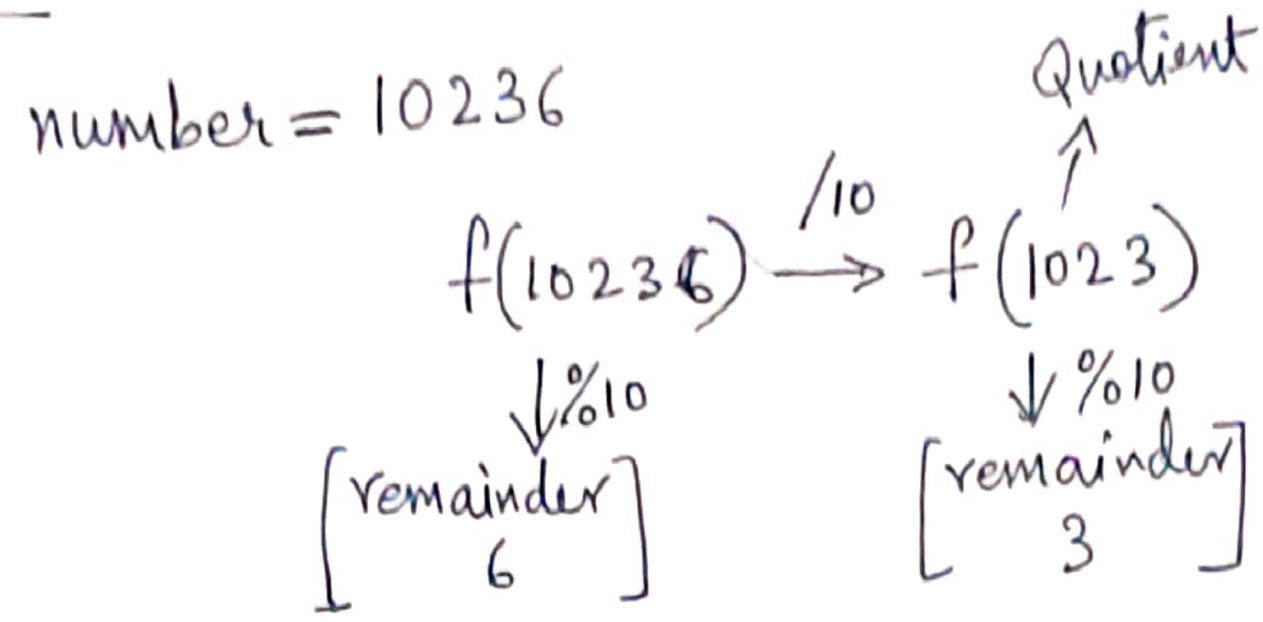
```
#include <iostream>
using namespace std;
void PrintDigits(int number);
int main()
{
    PrintDigits(6270);
}
void PrintDigits(int number)
{
    if (number <= 0)  // base condition
    {
        return;
    }
    int digit = number % 10;
    number = number / 10;          // Processing
    PrintDigits(number); // Recursive call
    cout << digits << " ";
}
```

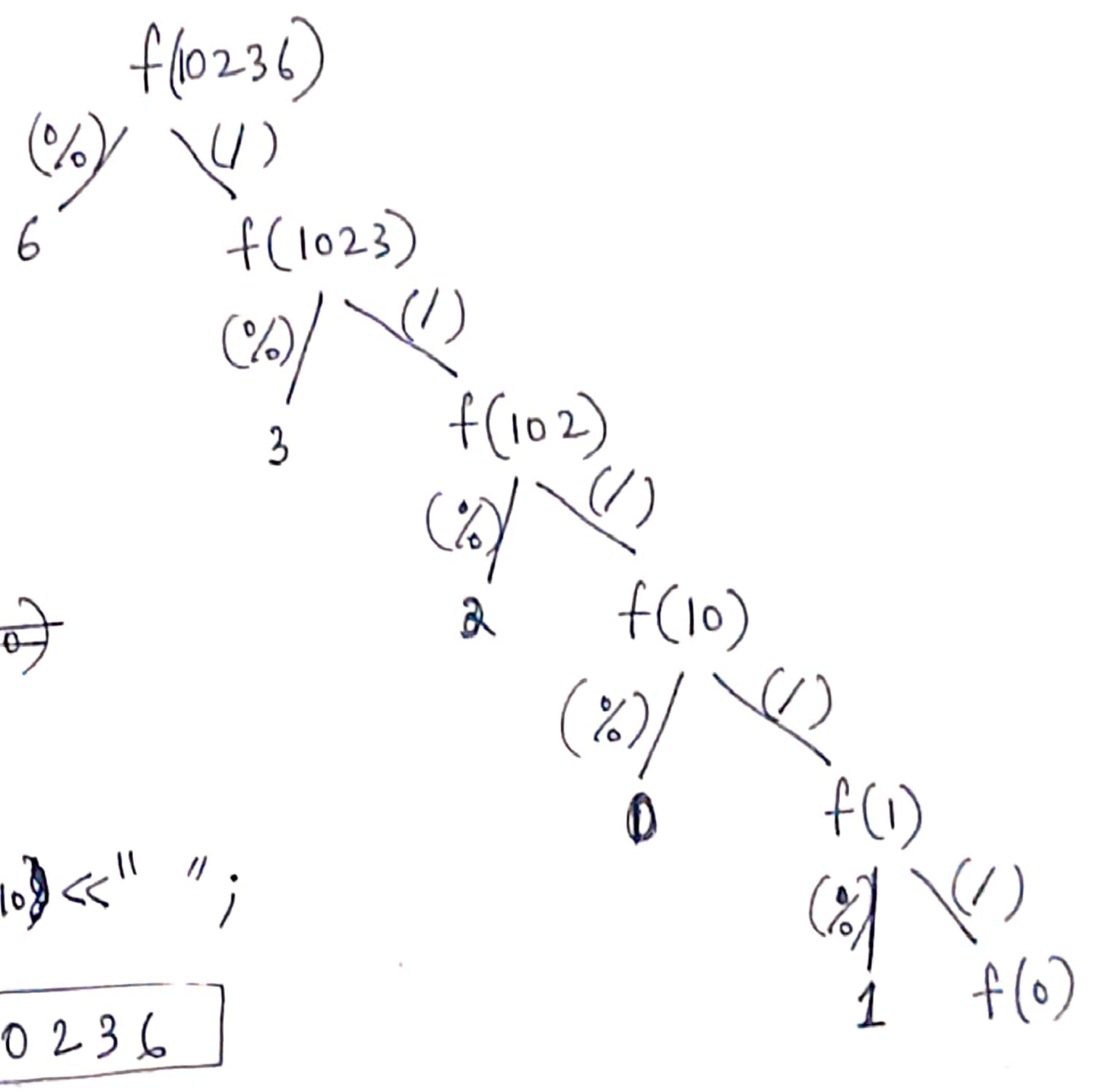
Note.- The Above function doesn't work for number starting zero.
→ generally numbers starting with zero are considered as octal number (i.e., base 8) but here we are considering {0647 Case} decimal number system (i.e., base 10) so it will give wrong answer in case of number starting with zero.

Solution decoding :- Modulo of a number gives remainder & Division of a number gives Quotient, using this we did the number % 10 (because our decimal system is 10) so each time this operation will give us the last digit & we will reduce the number & store it in the same number Variable

by dividing the number with 10. (eg:- 1023/10 ⇒ 102 for integer because int doesn't store decimal values)
for the next recursive call this number 102 is sent as a new number & the process (recursion) repeats.

dry run :





Head recursion

```
f(n) ~~f(n/10)~~
{
   f(n/10);
   cout << n%10 << " ";
}
```

output ⇒ 10236

Tail Recursion

```
f(n)
{
   cout << n%10 << " ";
   f(n/10);
}
```

output ⇒ 6 3 2 0 1