

# Warmup

Q.) You will be given a number  $x$  and you have to compute the value of  $f(x)$

$F[0]=1, F[1]=1$

$F(x)=F(x-1)+F(x-2) \quad x \geq 2$

Fibonacci Series-> 1 1 2 3 5 8 13 21....

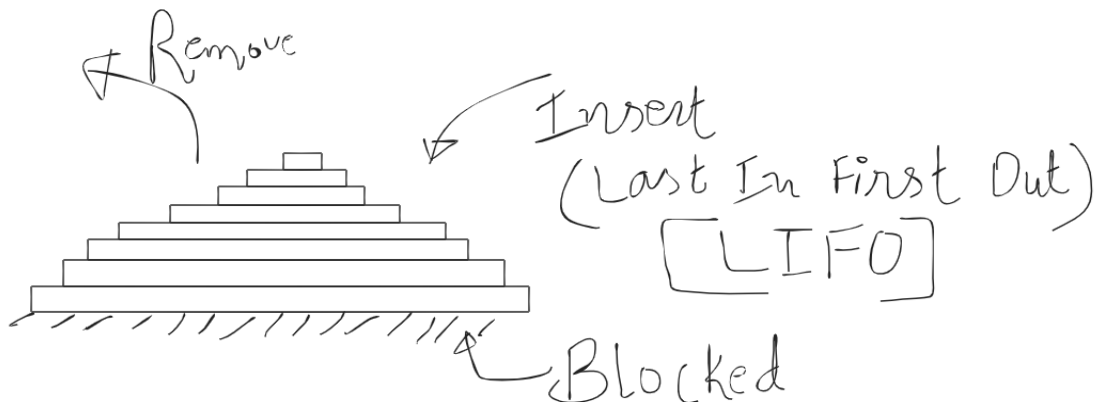
**Recursive Solution->**

```
int f(int x){  
    if(x==0 || x==1) return 1;  
    return f(x-1)+f(x-2);  
}
```

//f(5)-> f(4), f(3)-> f(3), f(2);

# Stack

Stack is a linear data structure. The insertion and deletion happens only at one end. It follows the property of “**Last In First Out**”(LIFO)



## Syntax for creating a stack

```
stack<data_type> stack_Name;
```

### Example:

```
stack<char> st;  
stack<double> st;
```

### Functions related to stack: Time Complexity= $O(1)$

1. push()-> Insert the last element at the back of your stack.

```
st.push(4);
```

2. pop() -> Remove the last element from the back.

```
st.pop();
```

3. empty()-> return true if the stack is empty()

```
st.empty(); //True/False
```

4. size() -> return the number of elements present in the stack.

5. top() -> show the top element

```
cout<<st.top();
```

## Queue

Queue is also a linear data structure in which insertion happens from one end and deletion happens from another end. It follows the property of “**First In First Out**”(FIFO)



### Syntax for creating a queue

```
queue<data_type> queue_Name;
```

### Example:

```
queue<char> q;  
queue<double> q;
```

### Functions related to queue: Time Complexity= $O(1)$

1. push() -> insert the element at the back.

```
q.push(4);
```

2. pop() -> remove the element at the front.

```
q.pop();
```

3. front() -> show the element present at the front

```
cout<<q.front();
```

4. empty() -> return true if the queue is empty()

```
q.empty(); // returns True or False
```

5. size()->return the number of elements present in the queue.

## Deque (Double ended queue)

It is also a linear data structure but the main difference with queue is that insertion and deletion happens from both ends.



I-> Insertion

R-> Removal

**Syntax of creating a deque**

```
deque<data_type> deque_Name;
```

**Example**

```
deque<char> dq;  
deque<double> dq;
```

### Functions related to deque: Time Complexity= $O(1)$

1. `push_back()` -> Insert an element at the back
2. `pop_back()` -> Remove the last element present at the back
3. `push_front()` -> Insert an element at the front
4. `pop_front()` -> Remove the first element present at the front
5. `empty()` -> return true if the deque is empty()

```
dq.empty(); // returns True or False
```

6. `size()` -> return the number of elements present in the deque.

**Note:** All these functions for stack, queue and deque have  $O(1)$  constant time complexity

## Some Problems

### Q.) Balanced Parentheses.

A bracket is considered to be any one of the following characters:

(, ), {, }, [, or ].

Two brackets are considered to be a matched pair if the opening bracket (i.e., (, [, or {) occurs to the left of a closing bracket (i.e., ), ], or }) of the exact same type. There are three types of matched pairs of brackets: [], {}, and ()

Return whether the given string is a balanced parentheses or not.

#### Link of the problem:

<https://www.hackerrank.com/challenges/balanced-brackets/problem>

#### Examples:

() -> Balanced

[()]-> Balanced

[(()]-> not balanced

[()]-> not balanced

{()}-> not balanced

## Definition of Balanced Parentheses

Empty string is a balanced parentheses.

{}, [], () -> balanced parentheses

Lets say a string s -> s is balanced.

{s}, (s), [s] -> balanced parentheses.

S and t -> s and t are balanced.

Then s+t -> balanced.

([])

## Solution Approach:

i<sup>th</sup> element -> if it is (, [, { -> push in the stack

-> if it is closing bracket ] -> [ , )->(, }-> {

( it should match with the top element in stack)

[()[]]

i=0 -> s[i]= [ -> push in stack

Sol:

```
string s; // [[
cin>>s;
stack<char> st;
bool balanced=true;
for(int i=0;i<s.length();i++){
    if(s[i]=='('||s[i]=='{'||s[i]=='['){
        st.push(s[i]);
        continue;
    }
    // a closing bracket encountered....pop one element
from the stack
    if(!st.empty()){
        char c = st.top();
        st.pop();
        if((s[i]==')'&&c=='(')||(s[i]=='}'&&c=='{'||(s[i]==']'&&c==
 '[')){
            // match of s[i] is found
        }else{
            // c and s[i] are not matching
            balanced=false;
            break;
        }
    }else{
        balanced=false;
        break;
    }
}
```

```
    }  
}  
if(!st.empty()){  
    balanced=false;  
}  
if(balanced){  
    cout<<"Balanced";  
}else{  
    cout<<"Not balanced";  
}
```

### Q: Next Greater Element

<https://www.codechef.com/problems/DC206>

4,6,5

4,5,6

4,5,7,6

Maintain stack of possible answers

```
int a[n];  
stack<int> s;
```



```
for(int i=n-1;i>=0;i--){
    while(!s.empty()&& a[i]>=s.top()){
        s.pop();
    }
    if(!s.empty()){
        ans[i]=s.top();
    }
    else
        ans[i]=-1;
    s.push(a[i]);
}
```

### HW:

Try to write code for finding "previous smaller element" for each element.

## Practice Problems on Stack, Queue, Deque

1. <https://www.hackerrank.com/challenges/deque-stl/problem>
2. <https://www.spoj.com/problems/STPAR/>
3. <https://www.hackerrank.com/challenges/largest-rectangle/problem>
4. <https://www.hackerrank.com/challenges/queries-with-fixed-length/problem>
5. <https://codeforces.com/contest/1373/problem/B>
6. <https://www.spoj.com/problems/JNEXT/>
7. <https://codeforces.com/contest/1374/problem/C>

8. <https://codeforces.com/contest/1313/problem/C2>