JAVA & DSA Series

Stacks

Part - 1

Raghav Garg

Recap

Linked List , OOPs

Today’s checklist

- Introduction to stacks
- Types of operations on stacks
- Stacks in C++ STL
- Overflow
- Underflow
- Array implementation of stacks
- Linked list implementation of stacks
- Advantages of array implementation of stacks
- Advantages of linked list implementation of stacks

Stack :

st.push( )

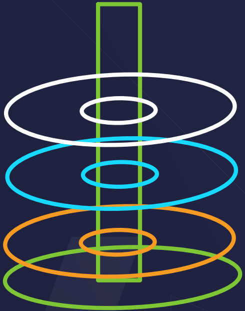
st.pop( )

st.peek( )

st.size( )

st.isEmpty( );

st.isFull( );



Stack Overflow

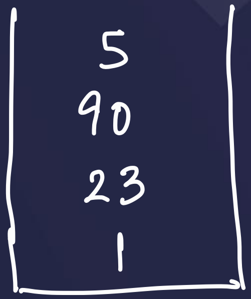
Last In First Out , First In Last Out

Stacks in C++ STL

LIFO / FILO

Syntax and In-built functions

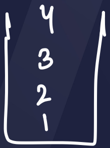
```
Stack < Data-Type>  st  =  new  Stack<>();

st.push(1);
st.push(23);
st.push(90);
st.push(5);
```



Fayda → Unlimited Size

Get

	T.C.	S.C.
Arrays	O(1)	O(1)
LL	O(n)	O(1)
Stack	O(n)	O(n)



Stack → Discipline

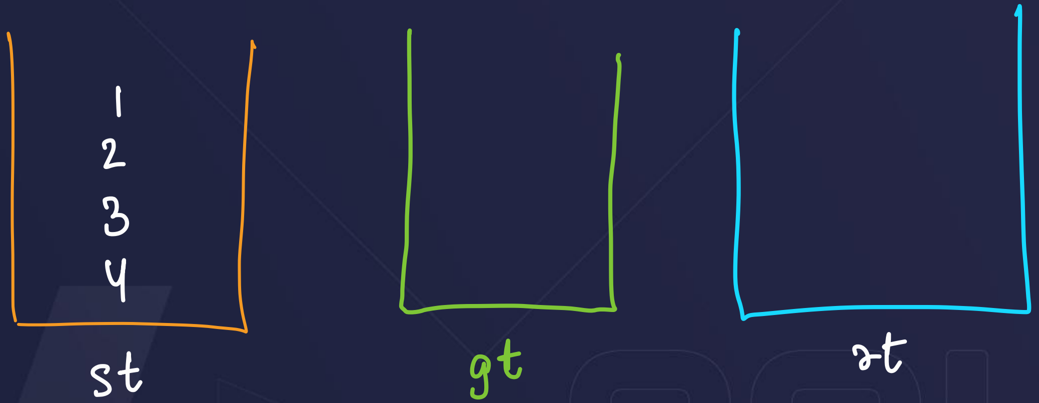
↓

top

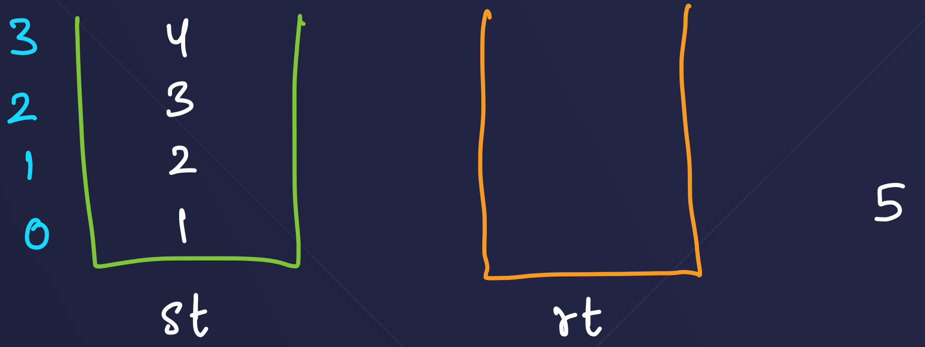
Q. Copy Stack

Copy contents of one stack to another in same order

→ Move

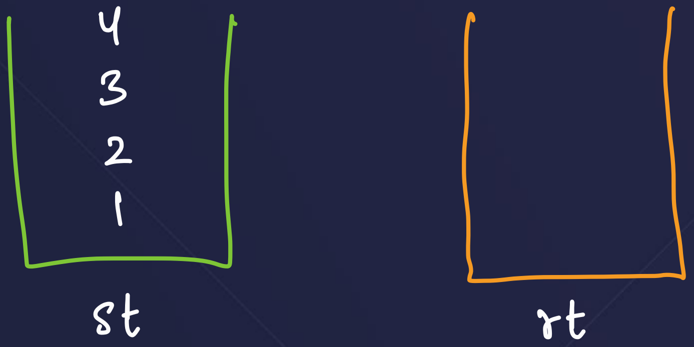


Q. Insert at bottom / any index



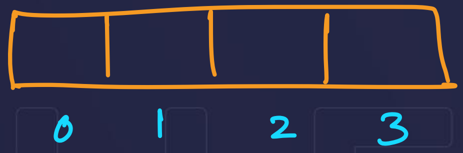
```
while (st.size( )>0)
1    rt. push(st.pop());
3
```

Q. Insert at bottom / any index



1 2 3 4

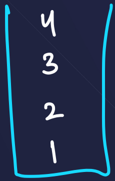
```
int n = st.size();
int[] arr = new int[n];
```



T.C. → O(n)

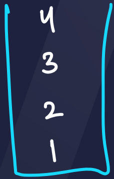
S.C. → O(n)

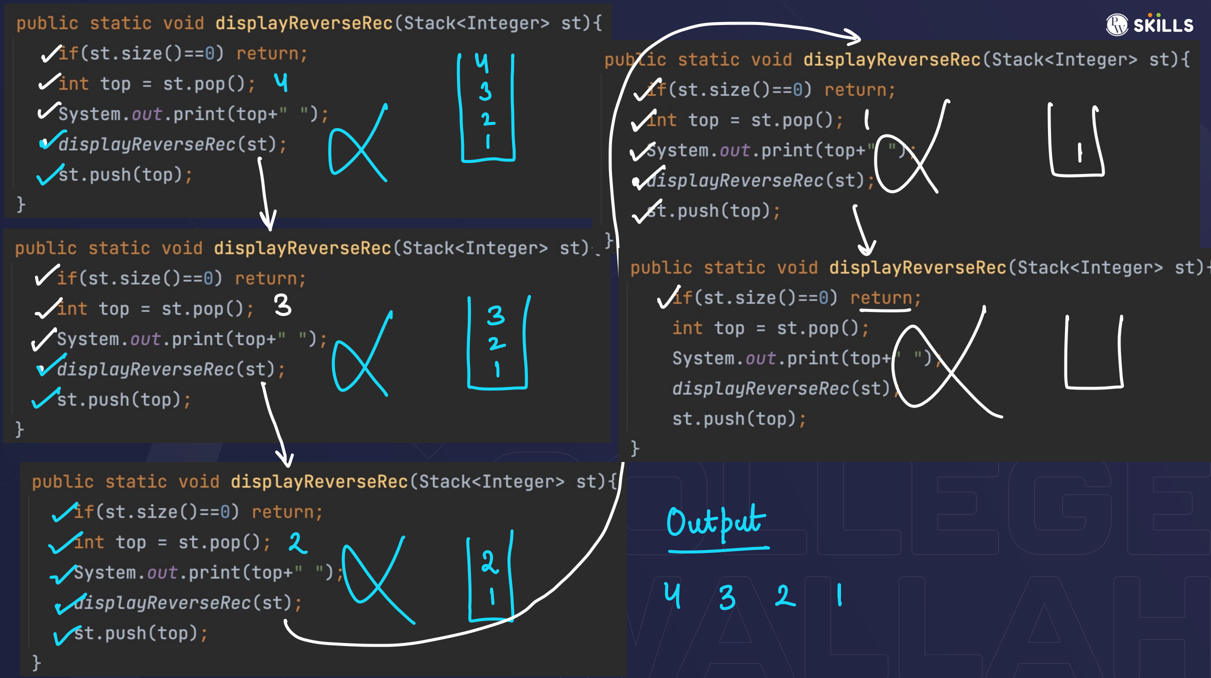
Display reverse stack Recursively :



```
display ( st ) {
    int top = st. pop( );
    Sout (top) ;
    display (st) ;
    st. push ( top);
}
```

4 3 2 1





```java
public static void displayReverseRec(Stack<Integer> st){
    if(st.size()==0) return;
    int top = st.pop();
    System.out.print(top+" ");
    displayReverseRec(st);
    st.push(top);
}
```

```java
public static void displayReverseRec(Stack<Integer> st){
    if(st.size()==0) return;
    int top = st.pop();
    System.out.print(top+" ");
    displayReverseRec(st);
    st.push(top);
}
```

```java
public static void displayReverseRec(Stack<Integer> st){
    if(st.size()==0) return;
    int top = st.pop();
    System.out.print(top+" ");
    displayReverseRec(st);
    st.push(top);
}
```

```java
public static void displayReverseRec(Stack<Integer> st){
    if(st.size()==0) return;
    int top = st.pop();
    System.out.print(top+" ");
    displayReverseRec(st);
    st.push(top);
}
```

```java
public static void displayReverseRec(Stack<Integer> st){
    if(st.size()==0) return;
    int top = st.pop();
    System.out.print(top+" ");
    displayReverseRec(st);
    st.push(top);
}
```

Output

4 3 2 1

Q// Push at Bottom → Recursion   Q// Reverse stack recursively

Classwork

Recursively display :

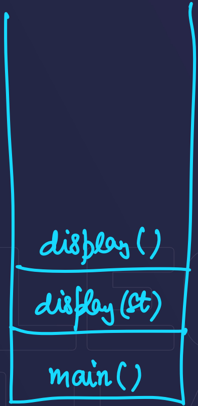
$T.C. = O(n)$

$S.C. = O(1)$ ✗

$\downarrow$

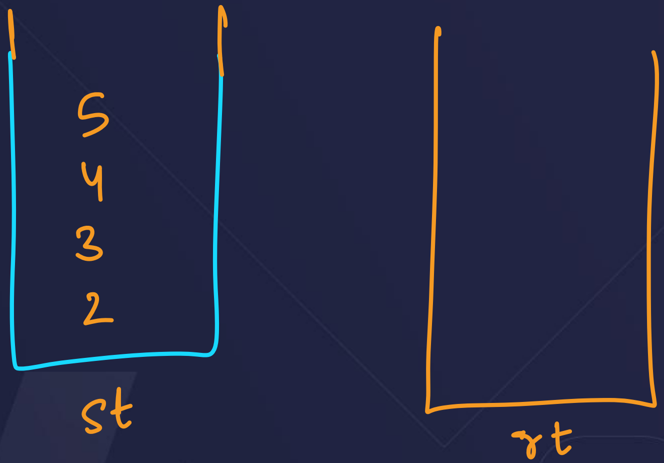
$\approx O(n)$

Call stack



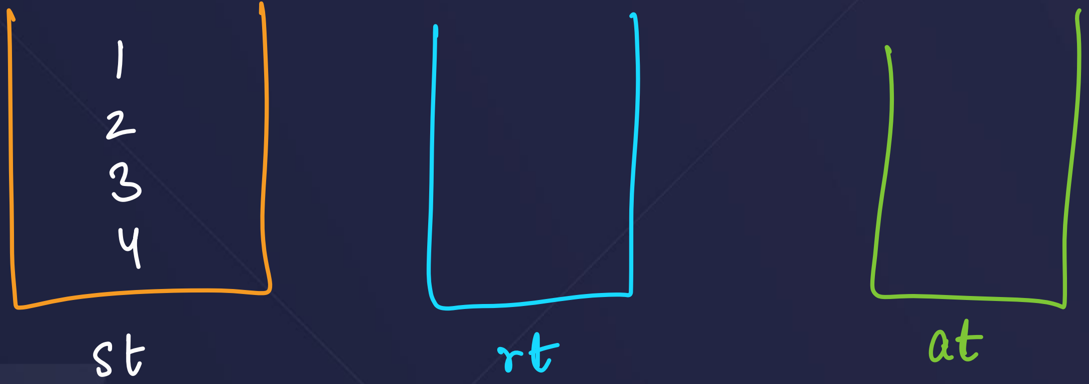
Q. Remove from bottom / any index

Classwork



```
while (st.size() > 1){
        rt.push(st.pop());
}
st.pop();

while (rt.size() > 0)
|    st.push(rt.pop());
}
```

Q. Reverse stack (iterative)

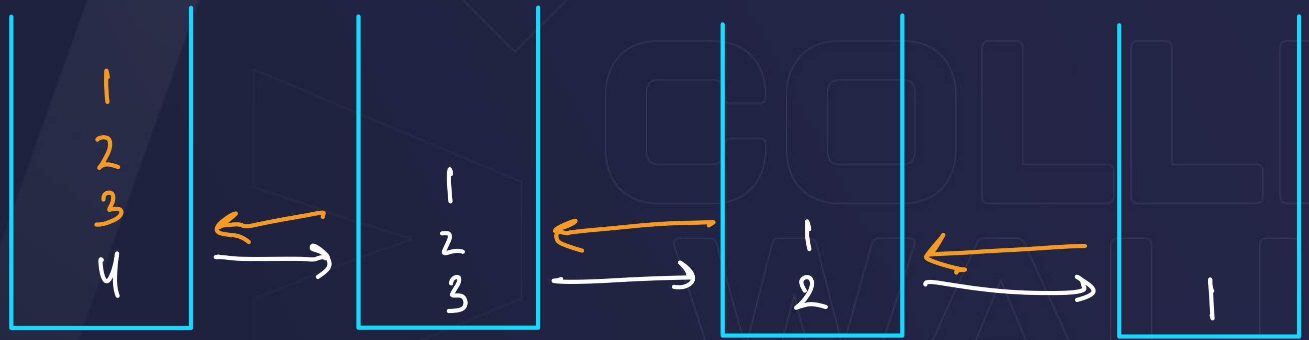


S.C. → 2 extra stacks → O(n)

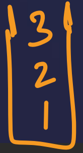
Q. Reverse stack

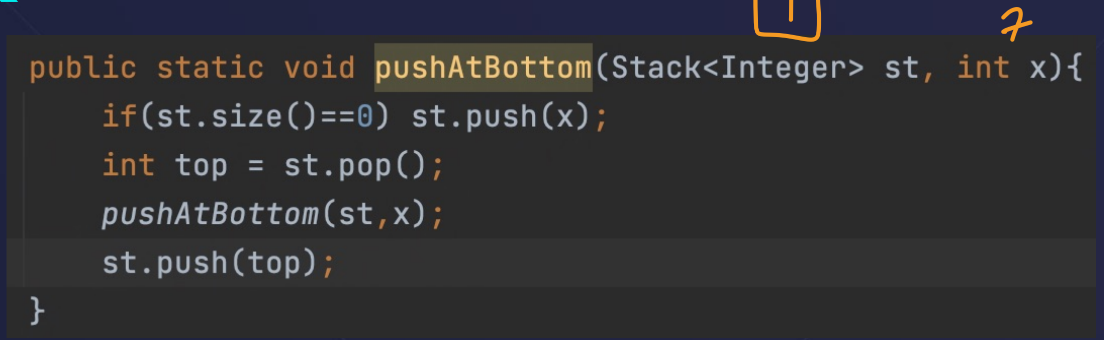
(recursive)

```
reverse(st){
    if(st.size==1) return;
    int top = st.pop();

    reverse(st);

    pushAtBottom(top);

}
```

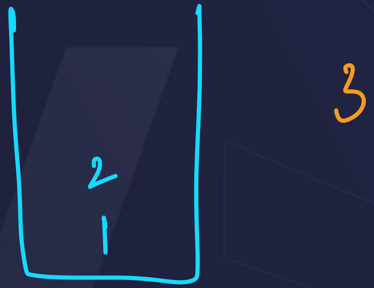


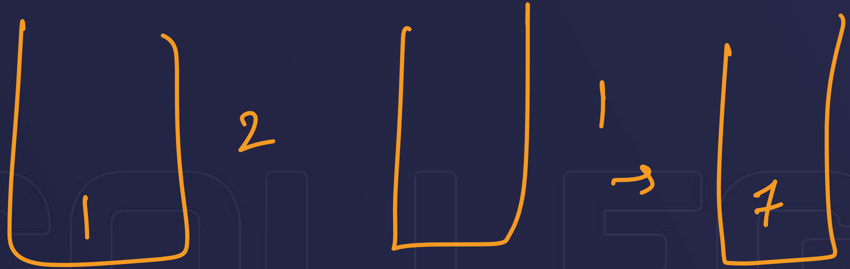
Q. Reverse stack





```
public static void pushAtBottom(Stack<Integer> st, int x){
    if(st.size()==0) st.push(x);
    int top = st.pop();
    pushAtBottom(st,x);
    st.push(top);
}
```



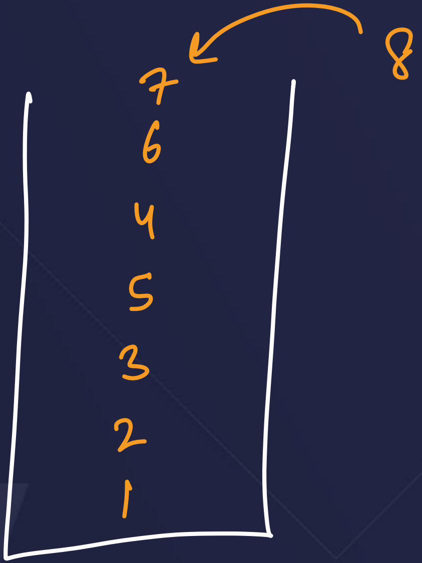


Overflow

$\longrightarrow$ Implement using Arrays

$\downarrow$

size $\rightarrow$ fixed



if stack is full &

you are trying to

st. push(-) , $\rightarrow$ Error

$\downarrow$

Actual memory is full

Underflow

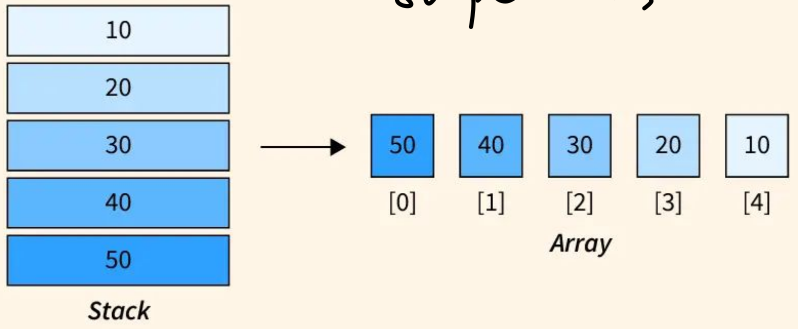
↪ if stack is empty → st.peek() or st.pop()

↓

Empty Stack Exception (Error)

Array implementation of Stacks

```
st.push(-);
st.pop( );
st.peek( );
```

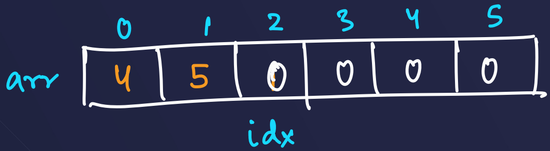


OOPS → mandatory

```
class Stack{
    int[] arr = new int[10];

    void push(int x){


    }
    int peek( ){


    }
```

```
int pop(){


}
int size(){


}
boolean isEmpty(){


}
```

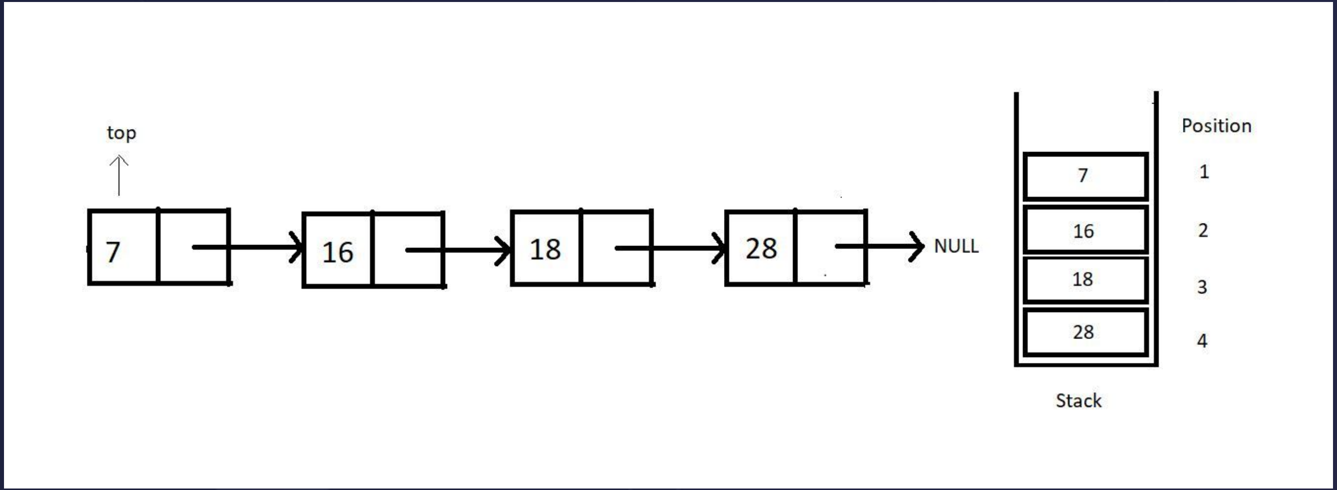
```
boolean isFull()


}
int capacity()
  return arr.length;
}
```



```
void push(int x){
    arr[idx] = x;
    idx++;
}
int pop(){
    if(idx==0) → →
    int top = arr[idx-1];
    arr[idx-1] = 0
    idx--;
}   return top;
```

```
int peek(){
    if(idx==0){ return -1 ; sout <Stack
                                 is
                                 empty
    return arr[idx-1];
}
```

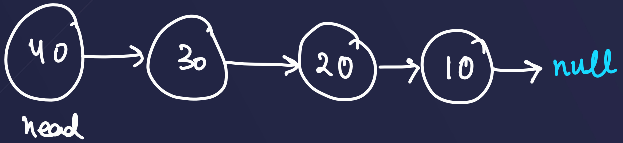
Linked list implementation of Stacks



```
class Node{
    int val;
    Node next;
}

class stack{
    Node head = null;
    int size = 0;



}
```

```
st.push(10);
st.push(20);
st.push(30);
st.push(40);
```

```
st.peek();
st.pop();
st.pop();
st.peek();
st.push(100);
```



Advantage of array implementation of stacks

1) Size → for every element → space taken is one block
2) display → $O(1)$

Disadvantages :

1) Size → fixed → overflow

int[] arr = new int[100]; //10 size → $\boxed{90}$

Advantage of linked list implementation of stacks

1) Unlimited Size

Disadvantages :

1) Size → two data members

2) display → reverse → space complexity ~ $O(n)$

Summary

- In this lecture we studied about the concept of stacks.

Questions → basic → (STL)

* Implementation → Array, LL

Upcoming lecture

- Stacks - 2

$\downarrow$

Interview Questions

LeetCode Bhar Bhar Ke

$\downarrow$

GFG

Thank You

$\downarrow$

Maza aa gaya