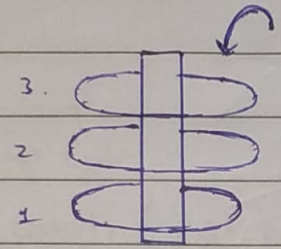
Stack.



→ Can insert at only 1 position.

→ Can access only topmost disk.

→ To access any disk, I need to remove from top.

LIFO :- Last - In - ~~First~~ First - Out.

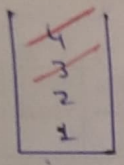
FILO :- First - In - Last - Out.

⇒ Operation on stack.

① Push :- st.push(x). ⟶ (To insert an ele in stacks)

② Pop :- st.pop(). ⟶ (To remove & print top most ele)

③ Peek :- st.peek(). ⟶ (To Only to print topmost ele).



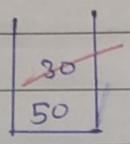
st.push(1)
st.push(2)
st.push(3)
st.push(4).

st.pop() ⟶ 4.
st.peek() ⟶ 3.
print :- 1 2 3
st.pop() ⟶ 3
Print :- 1 2
st.size() ⟶ 2.

⟹ STL for stack [Unlimited size]

Stack <Data_type> st = new Stack<>();

eg:- Stack<Integer> st = new Stack<>();



st.push(50);
st.push(30);
Sout(st.pop()); ⟶ 30

st.isEmpty() ⟶ whether stack is empty /not.
⟵ False

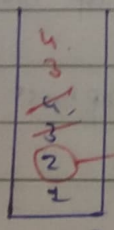
Return type is Boolean [True or False]

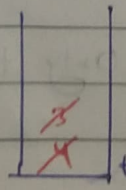
T.C :- O(N) ⟶ To access 1st element.

Comparing Arrays, Linked List & Stacks

Get Method :-

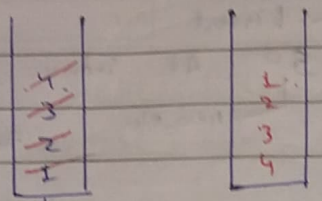
	T.C	S.C
Arrays	O(1)	O(1)
Linked List	O(n)	O(1)
Stack	O(n)	O(n)





temp.push(st.pop);
temp.push(st.pop);
Sout(st.peek());

Reverse A Stack.



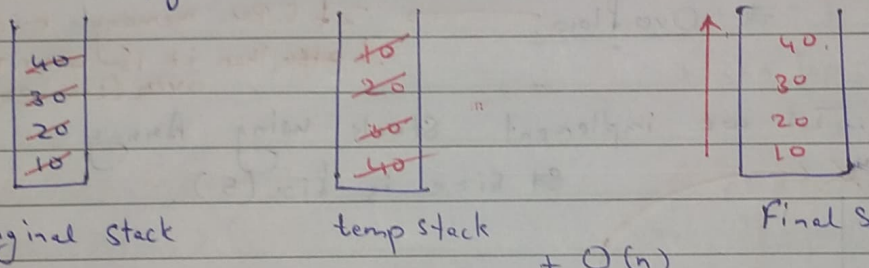
T.C :- O(n)
=> We pop ele from Original Stack & push in reverse Stack

S.C :- O(n)
=> for auxillary stack space.

=> Code :-

```
while(!st.isEmpty()){
    st_reversed.push(st.pop());
}
Sout(st_reversed);
```

Copy stack into another stack in Same Order.

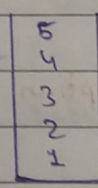


T.C :- O(n) + O(n) + O(n) => O(n)
S.C :- O(n) + O(n) => O(n)

=> Code :-

```
while(!st.isEmpty()){
    st_temp.push(st.pop());
}
```

```
while(!st_temp.isEmpty()){
    st_final.push(st_temp.pop());
}
```



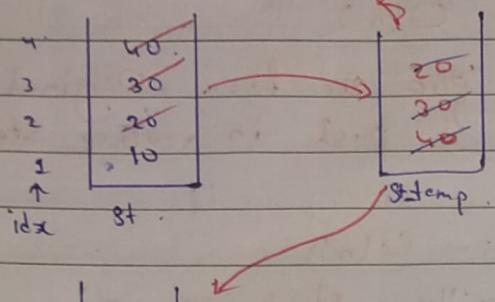
Display Stack. ⟶ Sout(st); {T.C :- O(n);}

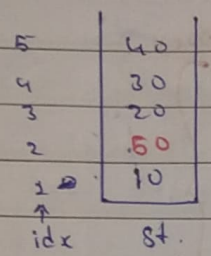
=> Code :-

Reverse Stack Recursively ⟶ T.C :- O(n); S.C :- O(n) (Recursive Stack Space.)

```
psv displayReverse(Stack<Integer> st){
    if(st.isEmpty()) return; // Base Case.
    int top = st.pop();
    Sout(top + " "); // To print in Reverse Order {5,4,3,2,1}.
    displayReverse(st);
    Sout(top + " "); // To print as it is {1,2,3,4,5}.
    st.push(top);
}
```

Push Ele At Bottom / Any Index.





T.C = O(2n)
O(n)

S.C = O(n).

insert 50 At index 2
↳ new_ele
idx.

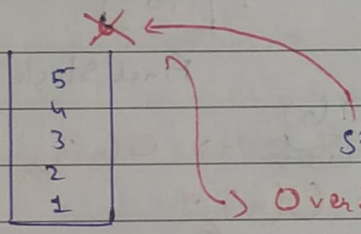
=> Code :-

```
while(st.size() >= idx){
    st_temp.push(st.pop());
}
st.push(new_ele);
while(!st_temp.isEmpty()){
    st.push(st_temp.pop());
}
Sout(st);
```

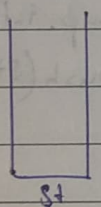
Overflow → If CPU memory exceeds then it is also Stack overflow error.

→ If we implement Stack using Array.

Size is fix (5).



Underflow



→ if Stack is empty & then.

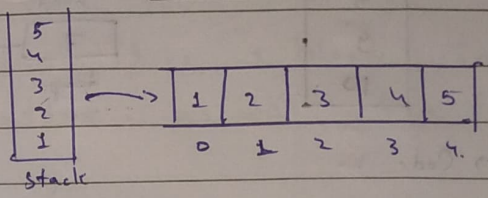
St.pop() / st.peek()

This will throw an Exception.

Array Implementation.

=> Code :-

```
public Static Class Stack {
    private int[] arr = new int[5];
    int n = arr.length;
    private int idx = 0;
```



// Push Method.

```
void push(int x){
    if(isFull()){
        Sout("Stack is Full");
        return;
    }
    ~~int top = arr~~
    arr[idx] = x;
    idx++;
}
```

// Pop Method.

```
int pop(){
    if(isEmpty()){
        Sout("Stack is Empty");
        return -1;
    }
    int top = arr[idx-1];
    arr[idx-1] = 0;
    idx--;
    return top;
}
```

// Peek Method.

```
int peek(){
    if(idx == 0){
        Sout("The Stack is Empty");
        return -1;
    }
    return arr[idx-1];
}
```

// Display Method.

```
Void display(){
    for(int i=0; i<idx; i++){
        Sout(arr[i] + " ");
    }
    Sout();
}
```

// Size Method.

```
int Size(){
    return idx;
}
```

// isEmpty Method.

```
boolean isEmpty(){
    if(idx==0) return true;
    else return false;
}
```

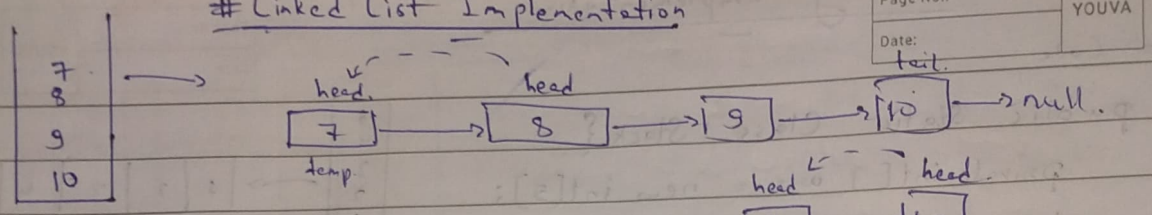
// display Any Idx ele.

```
int displayEle(int idx){
    if(x<0 || x>n){
        Sout("Error in Index..");
        return -1;
    }
    return arr[idx-1];
```

// isfull Method.

```
boolean isFull(){
    if(n == idx) return true;
    return false;
}
```

Linked List Implementation



→ Code :-

```
public static class Node{
    int val;
    Node next;
    Node(int val){
        this.val = val;
    }
}

public static class Stack{
    Node head;
    Node tail;
    int size = 0;
    // Push Method
    void push(int x){
        Node temp = new Node(x);
        if(size == 0){
            head = tail = temp;
            size++;
        }
        else{
            temp.next = head;
            head = temp;
            size++;
        }
    }
    // display Method
    void display(){
        displayRec(head);
    }
    void displayRec(Node h){
        if(h == null) return;
        displayRec(h.next);
        Sout(h.val + " ");
    }
```

① Create temp Node
② Point temp to head.
③ make temp as head.
④ Size++

```
    // Pop Method
    int pop(){
        if(head == null){
            Sout("stack is empty...");
            return -1;
        }
        int x = head.val;
        head = head.next;
        size--;
        return x;
    }
    // Peek Method
    int peek(){
        if(head == null){
            Sout("stack is empty...");
            return -1;
        }
        return head.val;
    }
    // isEmpty() Method
    boolean isEmpty(){
        if(size == 0) return true;
        return false;
    }
    // Size Method
    int size(){
        return size;
    }
```

Linked-List V/s Array Implementation

→ Advantages of Array Repr:

① Size :- for every ele space taken is 1 block. In LL, we have to store address as well which takes more memory.

② Display :- S.C :- O(1)

→ Dis Advantages :- ① Size is fixed (Overflow)

→ Advantages of LL Repr:- ① Size is Unlimited

→ Dis Advantages :-

① memory taken to store data is more

② Display :- S.C = O(n) [Recursive space]

Infix Expression

9 - 5 + 3 * 4 / 6.

9 - 5 + 12 / 6

9 - 5 + 2

4 + 2 ⇒ 6

DoDMAS

D, M > A, S.

Left to Right → 1st 3*4 / 2

12 / 2

⇒ 6 //

⇒ Evaluate Infix Using Stacks:-

→ Rules :- we need 2 stacks.

(i) Number stack. (Single digit numbers).

(ii) Operator stack

① If character is a number, push it in numbers stack.

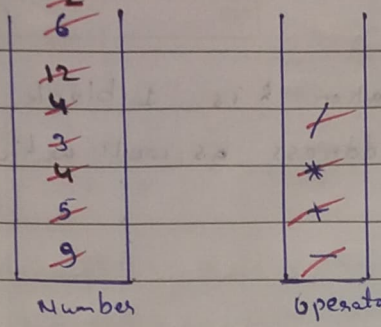
② If character is operator

(a) If operator stack is empty, then push it

(b) If op stack is not empty, we see operator on top of op stack. If its precedence is >= current operator, we first complete operation for operator on top of stack & then push the current operator.

9 - 5 + 3 x 4 / 6

6 → Ans.



Till opstack is not empty {

V_2 = numst.pop();

V_1 = numst.pop();

op = opst.pop();

r = V_1 op V_2

numst.push(r);

}

return numst.peek();

V_2 = numst.pop(). $V_2 = 5$ op = -

V_1 = numst.pop(). $V_1 = 9$ resout = 9 - 5 = 4 //

op = opst.pop(). $V_2 = 4$ op = *

resout = V_1 op V_2. $V_1 = 3$ res = 4 * 3 = 12 //

$V_2 = 6$ op = /

$V_1 = 12$ res = 12 / 6 = 2 //

$V_2 = 2$ op = +

$V_1 = 4$ res = 2 + 4 = 6 //

Infix with Brackets.

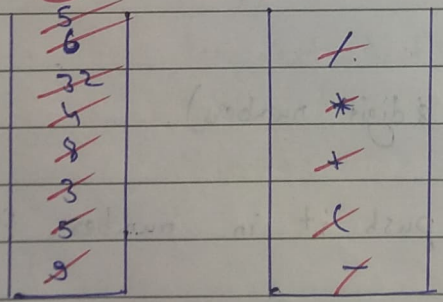
Str = " 9 - (5 + 3) * 4 / 6 ."

=> 9 - 8 * 4 / 6

=> 9 - 32 / 6

=> 9 - 5 → 4 //

4 → Ans.



$V_2 = 5$ op = -

$V_1 = 9$

r = 9 - 5 = 4 //

Extra.

```
if(op.size == 0 || ch == '('
    || op.peek() == '(')
    op.push(ch);
if( ch == ')'){
    while(op.peek() != '('){
        // work.
    }
}
```

$V_2 = 3$ op = +

$V_1 = 5$ r = 3 + 5 = 8 //

$V_2 = 4$ op = *

$V_1 = 8$ r = 4 * 8 = 32 //

$V_2 = 6$ op = /

$V_1 = 32$ r = 32 / 6 = 5 //

=> Code :- psvm( ) {

```
String str = "9-(5+3)*4/6";
Stack<Integer> val = new stack<>();
Stack<Character> op = new stack<>();
for(int i=0; i<n; i++){
    char ch = str.charAt(i);
    int ascii = (int) ch;
    if(ascii >= 48 && ascii <= 57){
        val.push(ascii - 48);

    else if(op.size()==0 || ch=='(' || op.peek()=='('
        op.push(ch);
    else if(ch==')'){
        while(op.peek()!='('){
            int v2 = val.pop();
            int v1 = val.pop();
            if(op.peek()=='-') val.push(v1-v2);
            if(op.peek()=='+') val.push(v1+v2);
            if(op.peek()=='*') val.push(v1*v2);
            if(op.peek()=='/') val.push(v1/v2);
            op.pop();
        }
        op.pop();
    }
    else {
        if(ch=='+' || ch=='-'){
            // Work
            op.push(ch);
        }
        else if(ch=='*' || ch=='/'){
            if(op.peek()=='*' || op.peek()=='/'){
                // Work  No Need to add '+' & '-' Lines
                op.push(ch);
            }
            else op.push(ch);
        }
    }
}
```

Work.

→ Continue.

```
→ while(val.size()>1){
    // Work
  }
  return val.peek();
```

Prefix Expression

Infix : a * b → 2 * 3

Prefix : * a b → * 2 3

Postfix : a b * → 2 3 *

9 - 5 + 3 * 4 / 6

9 - 5 + * 3 4 / 6

$V_2 = 4$ $op = *$ $V_1 = 3$ $* 34$

$V_2 = 6$ $op = /$ $V_1 = * 34$ $/ * 346$

9 - 5 + / * 346

$V_2 = 5$ $op = -$ $V_1 = 9$ -95

$V_2 = / * 346$ $op = +$ $V_1 = -95$ $+ -95 / * 346$

- 95 + / * 346

+ - 95 / * 346 → Ans.

⇒ op + V1 + V2

Postfix Expression

9 - (5 + 3) * 4 / 6

Infix : V_1 op V_2 — 95 + 4 6 / -

Prefix : op V_1 V_2 — 5 3 + 4 * 6 /

Postfix : V_1 V_2 op — 5 3 + 4 * 6 /

String	op
6	/
5 3 + 4 *	/
4	*
5 3 +	+
3	(
5	(
9	-

$V_2 = 3$ $op = +$

$V_1 = 5$ ⇒ 5 3 +

$V_1 V_2$ op

9 - 5 3 + * 4 / 6

9 - 5 3 + 4 * / 6

9 - 5 3 + 4 * 6 /

9 5 3 + 4 * 6 / - → Ans.

$V_2 = 4$ $op = *$

$V_1 = 53+$ ⇒ 5 3 + 4 *

$V_2 = 6$

$V_1 = 53+4*$ $op = /$

⇒ 5 3 + 4 * 6 /

$V_2 = 5\ 3 + 4 * 6 /$

$V_1 = 9$ $op = -$

⇒ 9 5 3 + 4 * 6 / - → Ans.

=> Code :- Psvm() {

```
for(int i = 0 ; i < n ; i++){
    Char ch = str.CharAt(i);
    int ascii = (int) ch;
    if(ascii >= 48 && ascii <= 57){
        String s = "" + ch;
        val.push(s);
    }
    else if(op.size() == 0 || ch == '(' || op.peek() == '('){
        op.push(ch);
    }
    else if(ch == ')'){
        while(op.peek() != '('){
            String v2 = val.pop();
            String v1 = val.pop();
            Char o = op.pop();
            String t = o + v1 + v2;
            val.push(t);
        }
    }
    else {
        if(ch == '+' || ch == '-'){
            Work;
            op.push(ch);
        }
        else if(ch == '*' || ch == '/'){
            if(op.peek() == '*' || op.peek() == '/'){
                work;
                op.push(ch);
            }
            else op.push(ch);
        }
    }
}
while(val.size() > 1){
    work;
}
return val.peek();
```

Work → For Postfix instead of t = v1 + v2 + o

M T W T F S S

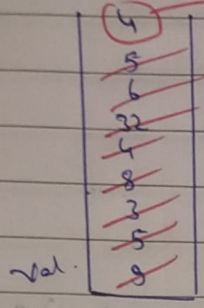
Page No.:

Date:

YOUVA

Evaluation of Postfix Expressions.

Str = " 953 + 4 * 6 / - ".



for (i = 0 to i < n) {
}

O V1 V2

V2 = 3
V1 = 5 => 5 + 3 => 8

V2 = 4
V1 = 8
O = * => 8 * 4 => 32

V2 = 6
V1 = 32 => 32 / 6 = 5

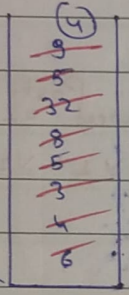
O = /

V2 = 5
V1 = 9 => 9 - 5 = 4

Evaluation of Prefix Expressions

Str = " - 9 / * + 5 3 4 6 "

for (i = n-1 to i >= 0) {
}



O V1 V2

V1 = 5
V2 = 3 } => 5 + 3 = 8

V1 = 8
V2 = 4 } => 8 * 4 = 32

V1 = 32
V2 = 6 } => 32 / 6 => 5

V1 = 9
V2 = 5 } => 9 - 5 = 4

=> Code for Postfix Evaluation :-

```
psvm( ) {
    String str = "953+4*6/-";
    Stack<Integer> val = new Stack<>();
    for (int i = 0 ; i < n ; i++) {
        char ch = str.charAt(i);
        int ascii = (int) ch;
        if (ascii >= 48 && ascii <= 57) val.push(ascii - 48);
        else {
            int v2 = val.pop();
            int v1 = val.pop();
            if (ch == '+') val.push(v1 + v2);
            if (ch == '-') val.push(v1 - v2);
            if (ch == '*') val.push(v1 * v2);
            if (ch == '/') val.push(v1 / v2);
        }
    }
    sout(val.peek());
}
```

for Prefix Evaluation

Changes :-

-> for(int i=n-1 to i>=0)

-> ~~first~~ first V1 then V2

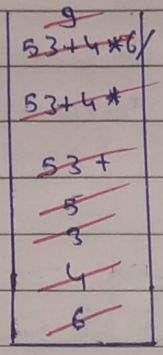
Conversion.

M T W T F S S | Page No.: | Date: | YOUVA

(1) Prefix to Postfix Conversion :-

Str = "−9/*+5346" ← (scan right to left) $V_1\ V_2\ op$

953+4*6/-



String stack.

$V_1 = 5$ $op = +$ $V_1 = 53+$ $op = *$.

$V_2 = 3$. $\Rightarrow 53+$. $V_2 = 4$ $\Rightarrow 53+4*$.

$V_1 = 53+4*$ $op = /$ $V_1 = 9$ $op = -$

$V_2 = 6$ $\Rightarrow 53+4*6/$ $V_2 = 53+4*6/$

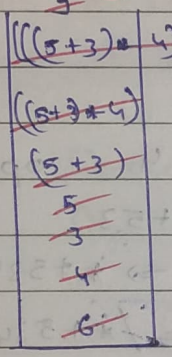
$\Rightarrow 953+4*6/-$ //

↳ Ans.

(2) Prefix to Infix Conversion :-

Str = "−9/*+5346." ← (scan right to left) $V_1\ op\ V_2$

$(9-(((5+3)\times 4)/6))$



$V_1 = 5$ $op = +$ $V_1 = (5+3)$

$V_2 = 3$. $\Rightarrow (5+3)$ $V_2 = 6$. $op = /$

$V_1\ op\ V_2$. $\Rightarrow (((5+3)*4)/6)$

$V_1 = (5+3)$ $op = *$

$V_2 = 4$. $\rightarrow ((5+3)*4)$

$V_1 = 9$ $op = -$

$V_2 = (((5+3)*4)/6)$ $= (9-(((5+3)*4)/6))$

=> Code :- // Prefix to Postfix.

```
for(int i = n-1; i>=0; i--){
    char ch = str.charAt(i);
    int ascii = (int) ch;
    if(ascii >= 48 && ascii <= 57){
        val.push(ch+"");
    }
    else{
        String V1 = val.pop();
        String V2 = val.pop();
        String t = V1 + V2 + ch;
        val.push(t);
    }
}
Sout(val.peek());
```

for Prefix to Infix. Just change

~~t = V1 + V2 + ch;~~

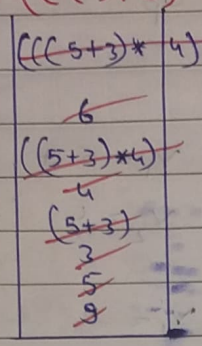
t = "(" + V1 + ch + V2 + ")";

for Prefix to Infix.

① Postfix To Infix Conversion.

Str = "953 + 4 * 6 / -" → *V1 op V2.

(9-(((5+3)*4)/6)



V2 = 3 op = +
V1 = 5 ⇒ (5+3)

V2 = 4 op = *
V1 = (5+3) ⇒ ((5+3)*4)

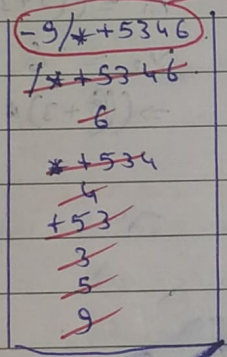
V2 = 6 op = /
V1 = ((5+3)*4) ⇒ (((5+3)*4)/6)

V2 = (((5+3)*4)/6)
V1 = 9 op = - } ⇒ (9-(((5+3)*4)/6))

② Postfix to Prefix

Str = "953 + 4 * 6 / -" → Ans.

Op V1 V2



V2 = 3 op = +
V1 = 5 ⇒ +53
op V1 V2

V2 = 4 op = *
V1 = +53
⇒ *+534

V2 = 6 op = /
V1 = *+534
⇒ /*+5346

V2 = /*+5346
V1 = 9 op = -
⇒ -9/*+5346

⇒ Code :- // Postfix To Infix.

```
for(int i=0 ; i<n ; i++){
    Char ch = str.charAt(i);
    int ascii = (int)ch;
    if(ascii >= 48 && ascii<=57){
        val.push(ch + "");
    }
    else{
        String V2 = val.pop();
        String V1 = val.pop();
        char op = ch;
        String t = "(" + V1 + op + V2 + ")";
        val.push(t);
    }
}
Sout(val.peek());
```

for postfix to Prefix
Change Only this.

```
t = op + V1 + V2;
```

for Postfix to Prefix.