In this class, Mistakes are......

Expected

Respected

Inspected

CORRECTED!

Keerthi.

SDE-3 Adobe.

Wipro, 2stats, flipkart.

2015, EEE, PESIT, Bangalore

Chikmagalore.

5 years into teaching

4.5 years into math.

Today's content

(i) Introduction to stacks

(ii) Implementation using linked lists

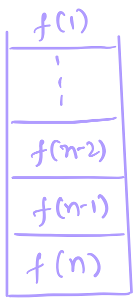
(iii) Double character trouble

(iv) Expression evaluation.

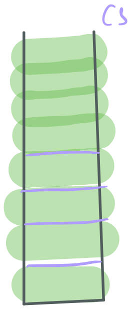
Stacks.

LIFO -> last in first out.

One on top of other



recursive calls in fu^n.

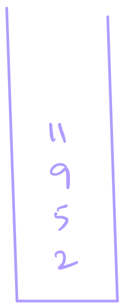


kitchen.
(stack of plates).

Operations possible.

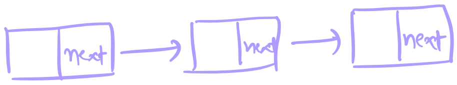
(i) push(x).
(ii) pop()
(iii) top(), peek()
(iv) size().

O(1) time



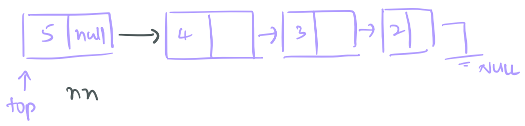
2	5	7	pop()	top()	9	11	8	pop()	top().
✓	✓	✓	7	5	✓	✓	✓	8	11.

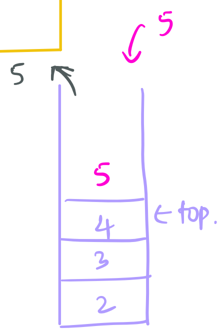
Q1 : Implement stack using linked list.



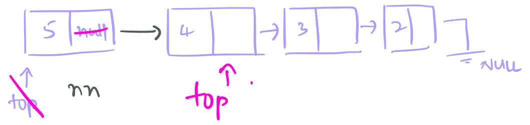
```
class Stack.
    Node top.// length.
    def push(data)
        new_node = Node(data)
        if (top == null)
            top = new_node.
        else
            new_node.next = top.
            top = new_node.
```

```
class Node
    data
    next
```

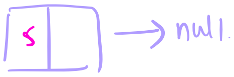




```
Node pop()
    if (top==null)
        return null
    else
        popped_node = top.
        top = top.next
        popped_node.next= null.
        return popped-node
```



top.



```
Node top(), len().

    TODO,
```

Java:

Stack // Deqeue.

↳ push
pop.
size
top.

Python. , deque() → stack, → [ ].

append(x)
pop()
stack[-1]
len(stack)

2Q. Given a string you need to remove all adjacent characters that are same.
Until there are no more adjacent characters that are same.

Ex1:
a b b d → ad.
a e b e a → aebea
a b c c b d e → abbde → ade.
a b b b e → abe
a d e b b e c a a c d e d → aed

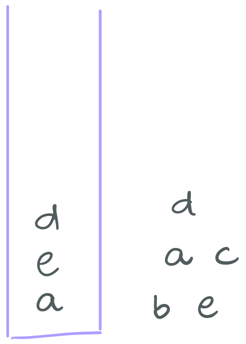
Ideas

Try to insert each character into stack.

(i) If top of stack and new character are same.
=> pop the top ele.

(ii) Else, push new element to the stack.

(iii) Reverse the stack, append every character & return.



TODO : i) Try pushing ele from last.
ii) Complete the code.

Code.

```
String removeAdjacents(String A)
    Stack<character> st;
    for(i=0;i<n;i++)
        if (!st.empty() && st.peek()==A.charAt(i))
            st.pop()
        else
            st.push()
    StringBuilder sb;
    while(!st.empty())
        sb.append(st.pop())
    return sb.reverse().toString();
```

38. Expression evaluation.

1. 8 * 5 + 4 = 44

2. 10 + 3 * 4 - 6 / 3

   10 + 12 − 2

   ⇒ 20.

3. 7 * 1 + 2 − 8 * 3 + 10 / 5

   7 + 2 − 24 + 2

   ⇒ -13.

BODMAS.

/, * : same precedence

+, − : same precedence

/, * → which to evaluate first?

evaluate from left to right.

[a+b] → a & b operands.

⊕ → operator.

Infix → Operator in between operands.

Postfix → operator comes after operands.

Infix Expressions		Postfix Expressions
a + b	→	ab+
a − b	→	ab−
b − a	→	ba−
a * b	→	ab*
a / b	→	ab/

Conversion. (Infix to postfix).

1) 4 + 8 * 7

   4 + 87*

   487*+

2) 10 + 3 * 4 - 7

   10 + 34* -7

   10 34* + -7

   10 34* + 7 −

3) 10 / (4 − 2) * 6 + 9

   10/ 42 − * 6 + 9

   10 42−/ * 6 + 9

   10 42−/ 6* + 9

   10 42−/ 6* 9 +

[postfix to infix → how?].

How does the evaluation actually happens? (by computer).

ex)

10 / (4-2) * 6 + 9

10 / 2 * 6 + 9

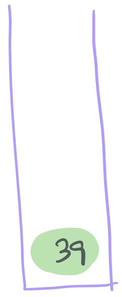
5 * 6 + 9

30 + 9

39.

10 4 2 - / 6 * 9 +.

(i)	a	b	a-b.
	4	2	4 -2 = 2.
(ii)	a	b	a/b.
	10	2	10/2 = 5
(iii)	a	b	a*b.
	5	6	5*6 = 30
(iv)	a	b	a+b
	30	9	30+9 = 39



Postfix expression evaluation:

* Iterate on expression.

operand → push into stack.

operator → pop two elements. ⇒ $\underset{\text{second pop}}{\underset{\downarrow}{a}}$, $\underset{\text{first pop}}{\underset{\downarrow}{b}}$.

Perform $a \underset{\text{any operator.}}{\underset{\downarrow}{\oplus}} b \Rightarrow$ result.

push result to stack.

Your top of stack is the answer.

```
                                                 ↗ array of strings.
int    postfixExpEvaluation ( String  expression)

    Stack <character>   stack =  new   stack<>();              TC : O(N)

                                                                SC : O(N).
    for ( int  i=0 ;  i < expression.length ;  i++)

    {
            if (!isCharacterAnOperator(expression. charAt (i)))
            {
                        Stack.push(expression.CharAt (i));
            }
            else
            {
                        int   b =  stack.pop()
                        int   a  =  stack.pop().

                        evaluate   ( a ⊕ b)      using  a switch.
                                      ↳ operator

                        result  =  a ⊕ b.
                        stack. push(result)

            }

    }
    return   stack.top()

}
```

Refer next page for working code.

Complete code:

```
int  evalExpression( List<String>  A)
        Stack<Integer>  st;
        int  a, b;
        for (String  str : A)
            if ( equalStrings (str, "+"))
                b = st.pop()
                a = st.pop()
                st.push( first + second);
            else if ( equalStrings (str, "-"))
                b = st.pop()
                a = st.pop()
                st.push( first - second);
            // add similar code for "/" & "*". in else if cond^n's.
            // last else => it's an integer.
            else
                st.push( Integer.parseInt (str))
        return  st.peek()
```

```
boolean equalStrings (String s1,
                      String s2)
{
    return  s1.equalsIgnoreCase(s2)
}
```