Lecture :- Stacks-1

Agenda

- Introduction
- Implementation
  - Dynamic array
  - Linked List
- Balanced paranthesis
- Double character trouble
- Evaluate postfix expression.

Stacks

- Stack of plates
- TOH (stack of discs)

Lifo : Last in first out

Data can only be added to top & removed only from the top.

Real life examples

1.) Recursion :- Stack [call | func" stack]

2.) Undo functionality.

3.) Browsers

Operations on stack

1.) push(x) : add x in stack [top]

2.) pop() : remove el from stack [top]

3.) peek() : return top most el.

4.) isEmpty() :

5.) size()

Implementation

1. ArrayList [Dynamic array]

push(2)

push(3)

push(8)

pop() → 8

peek() → 3

push(5)

isEmpty() → false → top == -1

pop() → 5

size() → 2

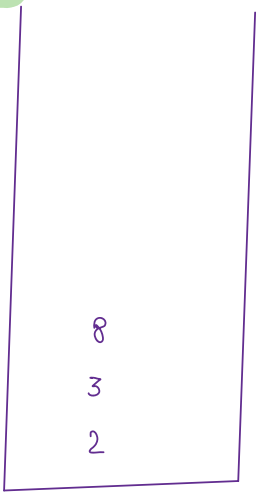
[topIdx + 1]

ArrayList:

[2 , 3 , ~~8~~ 5 ,]

topIdx = ~~-1~~ ~~0~~ ~~1~~ ~~2~~ ~~1~~ ~~2~~ 1

Visual representation:



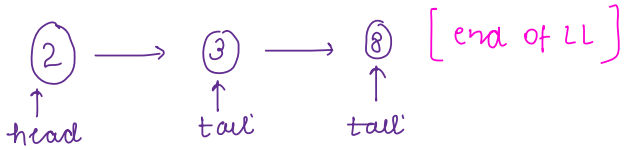
```
class AyushStack {

    List<Integer> stack;
    int top;

    AyushStack() {
        stack = new ArrayList<>();
        top = -1;
    }

    void push(int x) {                    // O(1)
        topIdx += 1;
        stack.set(topIdx, x);
    }                  ↑ func^n of arraylist

    int pop() {                           // O(1)
        if (topIdx == -1) {
            // Throw exception
        }
        deletedEl = stack.get(topIdx);
        topIdx--;
        return deletedEl;
    }

    int peek() {                          // O(1)
        if (topIdx == -1) {
            // Throw exception
        }
        return stack.get(topIdx);
    }

    boolean isEmpty() {                   // O(1)
        return top == -1;
    }

    int size() {                          // O(1)
        return topIdx+1;
    }
}
```

2.> Linked List

push(2)
push(3)
push(8)
pop() →
peek() →
push(5)
isEmpty() —
pop() —
size() —

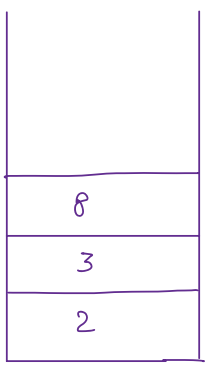
way 1:

head = null; // 2
tail = null; // 2 , 3



topIdx = ~~0~~, ~~1~~ ~~2~~ 3

visual representation:

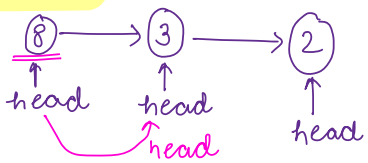


Way 2:

push(2)
push(3)
push(8)
pop() → 8
peek() → 3
push(5)
isEmpty() — false →
pop() — 5
size() — 2

[topIdx + 1]

head = null



topIdx = ~~0~~ ~~1~~ ~~2~~ 3

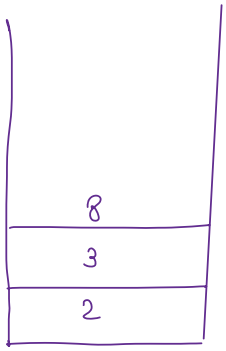
Insert

xn = new Node(x)
x.next = head;
head = xn.

Deletion

head = head.next;

visual representation:



```
class Ayush stack using LL {
    Node head;
    int topIdx;

    Ayush stack using LL () {
        head = null;
        topIdx = 0;
    }

    void push(x) {        — O(1)
        xn = new Node(x);
        if ( head == null) {
            head = xn;
        } else {
            xn.next = head;
            head = xn;
        }
        topIdx += 1;
    }

int pop() {        — O(1)
    if (head == null) {
        throw any exception
    }
    temp = head;
    head = head.next;
    topIdx--;
    return temp.data;
}

int peek() {        — O(1)
    if (head == null) {
        throw any exception
    }
    return head.data;
}

boolean isempty() {        — O(1)
    return head == null;
}

int size() {        — O(1)
    return topIdx;
}
```

Qu1 **Balanced parenthesis**

( ) [{ } ( )] — valid

( ) [{ (})] — invalid

({ } [ ] — invalid

) ([ ] — invalid

({ [ ] }) — valid

Logic: ([{ }) → invalid

] – invalid

) — invalid

Example:

( ) [{ (})]

↑ ↑ ↑ ↑ ↑ ↑

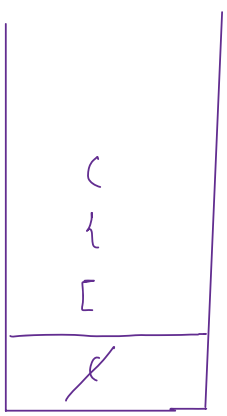
open — push()

close —

Look for top most element

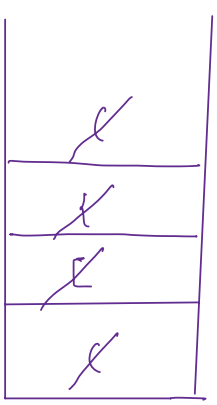
if match — go ahead [pop]

else — invalid



2.) ( ) [{ } ( )]

↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑



true [valid]

```
boolean validParanthesis(string str) {
        Stack<Character> stack = new Stack<>();
        for(i=0; i<n; i++) {
            char ch = input.charAt(i);
            if(ch=='(' || ch='{' || ch='[') {
                stack.push(ch);
            } else {
                if(stack.isEmpty()) {
                    return false;
                }
                top = stack.peek();
                if( ismatch(ch, top)) {
                    stack.pop();
                } else {
                    return false;
                }
            }
        }
        return stack.isEmpty();
}

    boolean ismatch(char o, char c) {
        if(o=='(' && c==')') {
            return true;
        }
        if(o=='{' && c=='}') {
            return true;
        }
        if(o=='[' && c==']') {
            return true;
        }
        return false;
    }
```

Break: 8 : 55 AM

Qu Given a string, remove equal pair of consecutive characters multiple times till possible and return final string

Example: 1.) a b b c : a c — ans

2.) a b c c b d e :

a b b d e

a d e — ans

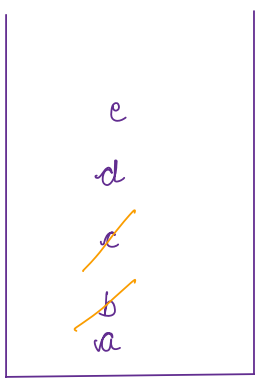
3.) a b b b c c c a :

a b c c c a

a b c a — ans

Ex:

a b c c b d e :

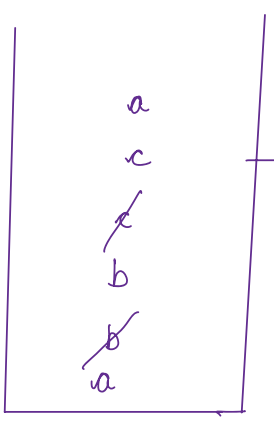


e d a rev a d e

a b b b c c c a :

a a a a a a b b c c :

empty string ""



a c b a

rev

a b c a

Qu3:

Infix

3 + 5

operand1 operator operand2

opr1 operator opr2

Postfix

3 5 +

opr1 opr2 operator.

1.) (a * b) - c —postfix→ ab * c -

ab* - c

ab * c -

2.) a + b - c * (d + e)

a + b - c * de +

a + b - cde + *

ab + - cde + *

ab + cde + * -

Q Why postfix?

Infix :- TC

a + b - c * (d + e) (f = d + e)

Traversal: figure which to solve

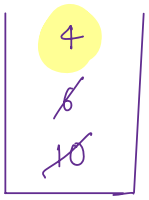
a + b - c * f (x = c * f)

a + b - x (s = a + b)

c - x Ans

Evaluate postfix expression

Q. $10 - 6 \longrightarrow 10\ 6\ -$



$10 - 6 = 4$

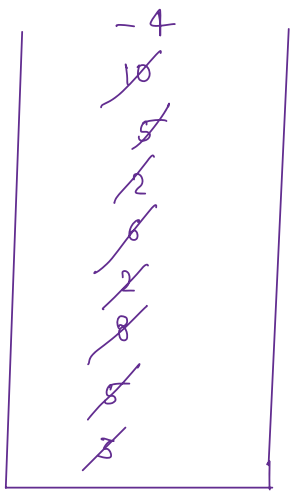
Q $3\ 5\ +\ 2\ -\ 2\ 5\ *\ -$

a b c c b

int a = 3

int b = 5

int c = 2



$3 + 5 = 8$

$8 - 2 = 6$

$2 * 5 = 10$

$6 - 10 = -4$ Ans

TC of postfix is better than infix

$O(n)$ $O(n^2)$

```
int evaluatePostfix(String str) {
    Stack<Character> stack = new Stack<>();

    for(i=0; i<str.length; i++) {
        char ch = str.charAt(i);
        if ( isOperand(ch)) {
            stack.push(ch - '0');
        } else{
            int b = stack.pop();
            int a = stack.pop();
            int c = perform(a, b, ch);
            stack.push(c);
        }
    }

    return stack.peek();
}

int perform(int a, int b, char o) {
    if(o == '+') {
        return a+b;
    }
    if(o =='-'){
        return a-b;
    }
    if(o ==' *'){
        return a*b;
    }
    return a|b;
}
```

Please handle edge cases

Edge case

Edge case

Edge case

Thankyou ☺