Todays Content:

a) Stack Basics

b) Stack Implementation

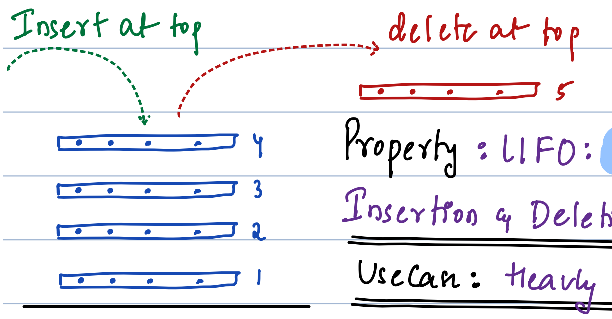
c) Double character trouble

d) Expression Evaluation

    a) Infix → Postfix

    b) Postfix Evaluation

Stack:



Property : LIFO: Last in First Out

Insertion & Deletion at Same Space

UseCase: Heavly Used in Recursion / Memory Management

Functions

push(x): Insert x on top of stack

pop(): delete top most element

peek(): return top ele

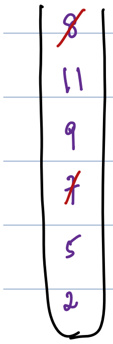
size(): return no:of ele in stack

Note: When ever we use any stack we can only use above 4 functions.

Dry Run:

Ex: 2 5 7 pop() peek() 9 11 8 pop() peek() peek()

7* return 5 8* return 11 return 11



Note: Only element we can acces in stack is top most ele

Obs: Both push & pop happens from same side

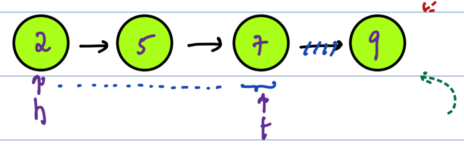
Stack Implementation:

a) Using LinkedList

Stack using linked list:

Ex: 2 5 7 9 top() pop() 11 8 pop() top()

9

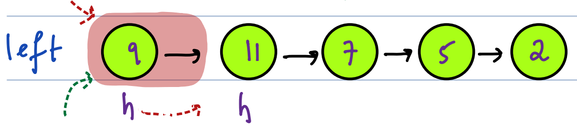


TC: pop() = O(N)

Traverse & go till last.

Ex: 2 5 7 9 top() pop() 11 8 pop() top()

9



TC: push() : O(1)

pop() : O(1)

Note: We insert & delete at head side.

```
Class Node{
    int data;
    Node next;
    Node(int n){
        data=n
        next=NULL
    }
}
```

```
Node h=NULL; int c=0;  Size in O(1)
void Insert(int n){
    Node nn = new Node(n);
    nn.next=h;
    h=nn;
}   c=c+1;
```

```
void pop( ){
    if(h==NULL){return}
    h=h.next;
}   c=c-1;
```

```
int peek(){
    if(h==NULL){return __}
    return h.data;
}
```

```
int size(){
    return c;
}
```

Inbuilt:

```
Stack<datatype> st = new Stack<datatype>();
st.push( );    st.peek();    st.pop();    st.size();
```

↳ If python: Use list for stack purpose. In Java: Can use ArrayList.

10) Double Character Trouble

Given a string s, Remove equal pair of adjacent characters

Return the string without adjacent duplicates

0 1 2 3

Ex1: a ~~b~~ ~~b~~ d → output: ad

0 1 2 3 4 5 6

Ex2: a b ~~c~~ ~~c~~ b d e → output: ade

a ~~b~~ ~~b~~ d e

0 1 2 3 4

Ex3: a ~~b~~ ~~b~~ b e → output: abe

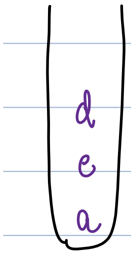
Ex4: a b a b a b → ababab

0	1	2	3	4	5	6	7	8	9	10	11	12
a	d	e	b	b	e	c	a	a	c	d	e	d

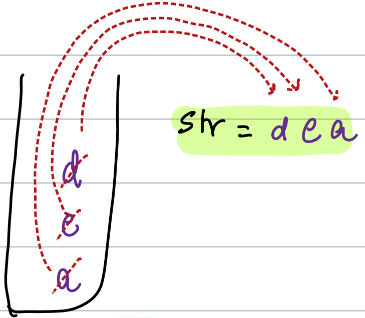
Ex5:

ans = a d e ~~b~~ Note: Insert & deleting a char from same side

Idea: Using stack<character>



1. If new char is same as top of stack, pop top of stack
2. Once all char done:
    a. get top char pop it
       add it to back of String
3. Reverse Entire String.



Note: Iterate from right to left & check with stack, we can get correct String.

```
String character(String str){ TC: O( ) SC: O( )   TC: O(N) SC: O(N)
    Stack<character> st = new Stack<character>();
    int N = str.length();
    for( i = N-1; i>=0; i--){
        if(st.size()==0){
            st.push( str.charAt(i)) // We directly push.
        }
        else if( str.charAt(i) == st.peek()){
            st.pop();
        }
        else{
            st.push(str.charAt(i))
        }
    }

    // Step: Pop characters from stack create ans:
      Note: When ever we want to append, char, string use StringBuilder
      String Builder sb = new String Builder();
      while( st.size()>0){ // Untill empty
          char ch = st.peek();
          st.pop();
          sb.append(ch);
      }
      return sb.toString();
}
```

Expression Evaluation:

Ex1: $8*5+4 =$

$= 40 + 4 = 44$

Ex2: $9 + 3*4 - 6/3$

$= 9 + 12 - 6/3$

$= 9 + 12 - 2 = 19$

Ex3: $7*6/7 =$

$= 42/7 = 6$

Operator Precedence

1. ( )
2. / * : Same Precedence
3. + - : Same Precedence

In above case: If operators have same precedence operator which comes first from left to right.

Ex4: $2*5 + (3-8)*2 + 10/5$

$= 2*5 + -5*2 + 10/5$

$= 10 - 10 + 2$

$= 2$

int x = $\{\underbrace{2*5}_{2nd} + \underbrace{(3-8)}_{1st} \underbrace{*2}_{3rd} + \underbrace{10/5}_{4th}\}$

Expressions:

a. Infix = a+b b. Postfix = ab+

Expressions we write in Infix are converted to Postfix & Evaluated

Infix:		Postfix:
a+b	------→	ab+
a-b	------→	ab-
a/b	------→	ab/
a*b	------→	ab*

Doubts?

1. convert →?
2. Why postfix is undertable to system?

Note: Postfix wont contain any brackets, even if present in infix.

op1 op2

Ex1: $a + b\underbrace{*c}$

$\underbrace{a}_{op1} + \underbrace{bc*}_{op2}$

$\to a\ bc*+$

o1 o2

Ex2: $a/(b-c)+d$

$\underbrace{a}_{o1} / \underbrace{bc-}_{o2} + d$

$\to \underbrace{a\,bc-/}_{o1} + \underbrace{d}_{o2}$

$\to abc-/d+$

Ex3: $8/(\overset{o1}{4}-\overset{o2}{2})*6+9$

$\underbrace{8}_{o1} / \underbrace{42-}_{o2} *6+9$

$\to \underbrace{842-/}_{o1} * \underbrace{6}_{o2} + 9$

$\to \underbrace{842-/6*}_{o1} + \underbrace{9}_{o2}$

$\to 842-/6*9+$

1. Given Infix ⟶ Postfix:

a. Take a stack<character>

b. Iterate on Infix: ✓

1. Operand	1. Add to Postfix ✓
==(2. Open bracket	2. Add to Stack ✓
==) 3. Closed bracket	3. Pop from stack & add them postfix ✓ till you get an open bracket: (delete it, dont add in Post)
4. operator ch ==+, ==-, ==* ==/	4. a. If stack is empty : push ch ✓ b. If top of stack is (: push ch ✓ c. while(stack.size()>0 && ✓ pre(ch) <= pre(stack.peek())} 1. pop & add it postfix } d. Add ch to stack ✓

Note: Higher precedence will be on top lower precedence

e. Pop & add in postfix till stack is empty ✓

Infix: A + B * C - D * (F + G * K) + L * M

Postfix: A B C * + D F G K * + * - L M * +

```
int pre(char ch){
    if(ch=='+' || ch=='-'){return 1}
    if(ch=='*' || ch=='/'){return 2}
}
```

Note: 1. Take Postfix as string Builder because we add char by char

Finally: Convert string Builder to String & return

Evaluating Postfix:

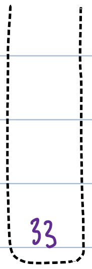
1. We need a stack

2. Iterate on postfix expression.

a. Operand	1. Push in stack
b. Operator ch	2. Get top ele in stack = b & pop it Get top ele in stack = a & pop it Perform a ch b & insert in Stack

3. Top of stack:

Ex1: 8/(4-2)*6+9 ⟶ 8 4 2 - / 6 * 9 +



a	ch	b		
4	-	2	=	2 push
8	/	2	=	4 push
4	*	6	=	24 push
24	+	9	=	33 push

Why Postfix:

Ex1: 8/(4-2)*6+9 ⟶ 8 4 2 - / 6 * 9 +

Operation Execution:

- - : 8/2 * 6 + 9
- / : 4 * 6 + 9
- * : 24 + 9
- + : 33