Lecture : Stacks - 1

Agenda

- Introduction
- Implementation using arrays
- " " stacks
- Balanced parentheses
- Remove consecutive duplicates
- Evaluate postfix

Introduction to stacks

→ A linear data structure that works on LIFO principle.
Last in first out.

Examples

Pile of plates.

Stack of chairs

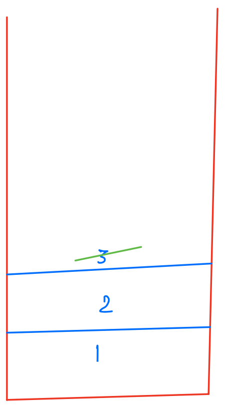
Algorithmic examples

Recursion.

Undo.

Operations on stack

Operations	Understanding
push	insert a new el in stack.
pop	Remove top element of stack Return.
peek	Returns top element of stack
isempty	if stack is empty or not?
size.	size of stack.

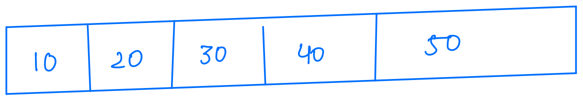


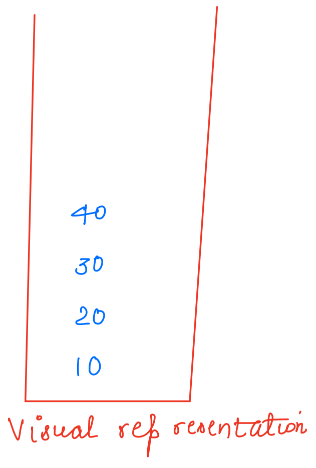
push(1)
push(2)
push(3)
peek() $\longrightarrow$ 3
pop() $\longrightarrow$
peek() $\longrightarrow$ 2
size() $\longrightarrow$ 2
isempty() $\longrightarrow$ false

Implementation of stack using array

fixed size.

arr[5] =





idx = -1

push(10) ⟶ idx++
arr[idx] = 10

push(20) ⟶ idx++
arr[idx] = 20

push(30) ⟶ idx++
arr[idx] = 30

push(40) ⟶ idx++
arr[idx] = 40

push(50) ⟶ idx++
arr[idx] = 50

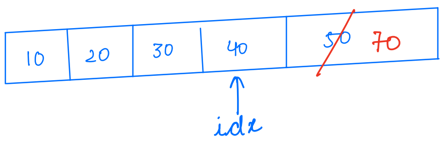
push(60) ⟶ idx++ // 5
arr[idx] = 60 idx out of bound.

pop() ⟶ idx--

peek() ⟶ return arr[idx] // 40

push(70) ⟶ idx++
arr[idx] = 70

arr[5] =



size() ⟶ return idx+1.

isEmpty ⟶ check for size.

Pseudocode

```
int[]  A
int idx = -1;

void push(int val) {
        if (idx == A.length - 1) {
                overflow.
        }
        idx++;
        A[idx] = val;
}

void pop() {
        if (isEmpty()) {
                return;
        }
        idx--;
}

int  peek() {
      if (isEmpty()) {
          underflow.
      }
      return arr[idx];
}

boolean isEmpty() {
      return idx == -1;
}

int size() {
      return idx+1;
}
```

ArrayList $\longrightarrow$ taken care of fixed size

Implementation using linked list

push(10)

push(20)

push(30)

push(40)

push(50)

push(60)

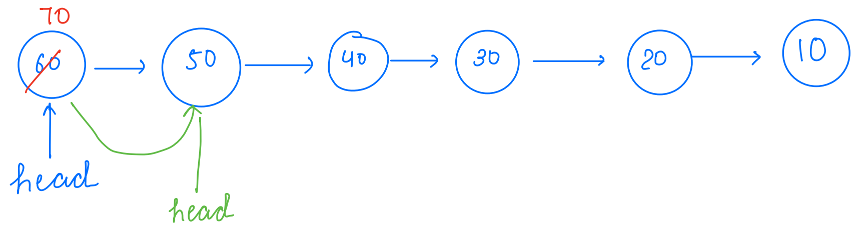
pop()

peek()

push(70)

size()

isEmpty



Pseudocode

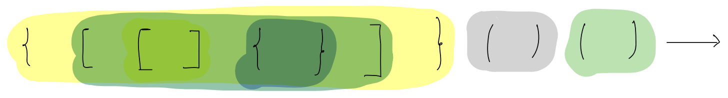
```
void push(int x) {
        xn = new Node(x);
        xn.next = head;
        head = xn;
        t++;
}

int pop() {
        if(isEmpty()){
            return -1;
        }
        int top = head.data;
        head = head.next'
        t--;
        return top;
}

int top() {
    if(isEmpty()){
        return -1;
    }
    return head.data;
}

boolean isEmpty() {
    return t ==0;
            size()==0;
}

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

Qu **Balanced parentheses**

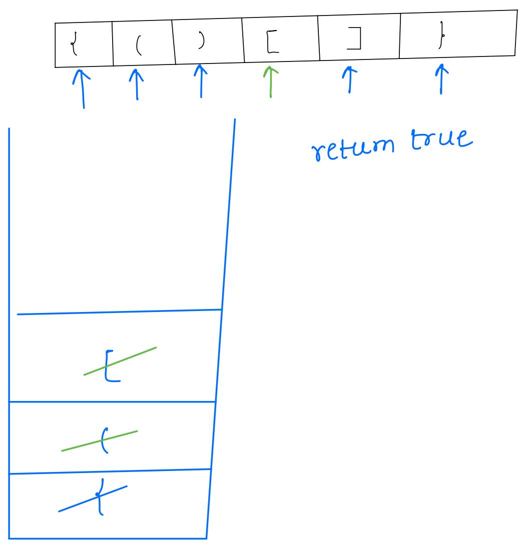
Check whether given sequence of parentheses is valid?

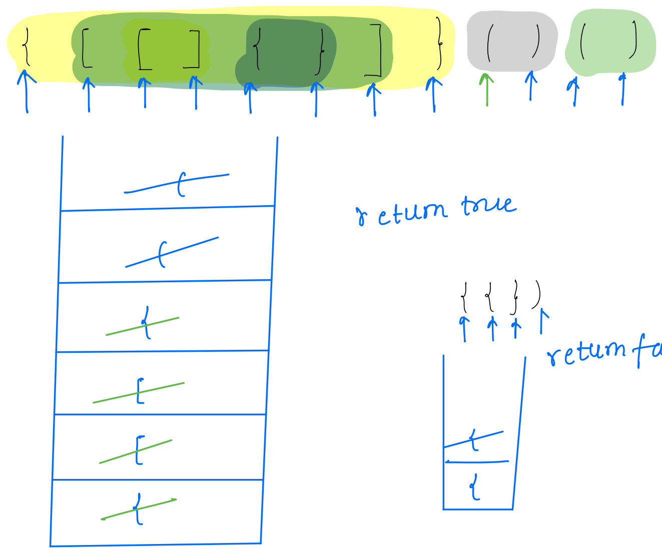
(({ })) ⟶ valid

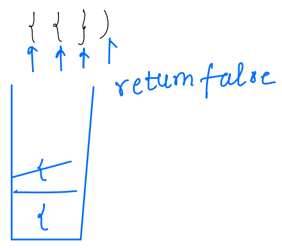
{ { }) ⟶ invalid



Approach







Pseudocode

```
boolean isValid ( String ip ) {
        Stack<Character> st = new Stack<>();

        for(i=0; i<ip.length(); i++) {
                char ch = ip.charAt(i);
                if ( isOpenBracket ( ch)) {
                        st.push( ch);
                } else {
                        if( st.isEmpty()) {
                                return false;
                        }
                        char top = st.pop();
                        if (!isValid ( ch, top) ) {
                                return false;
                        }
                }
        }

        if ( st.isEmpty()) {
                return true;
        }
        return false;
}
```

TC: O(n)
SC: O(n)

Qu Given a string, remove equal no. of consecutive elements if possible.

Example

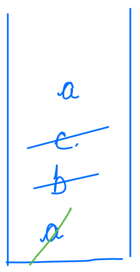
a b c d d c
↓
a b c c
↓
ab

a a b c c b a
↓
b b a
↓
a

$\underbrace{a\ a}\ \underbrace{a\ a}\ b\ \underbrace{c\ c} \longrightarrow b$

$\underbrace{a\ a}\ a\ b\ \underbrace{c\ c} \longrightarrow ab$

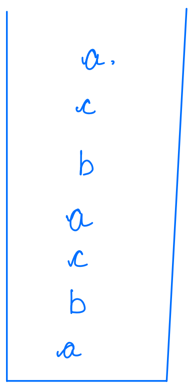
a a b c c b a
↑ ↑ ↑ ↑ ↑ ↑



$\boxed{\text{Ans} = a}$

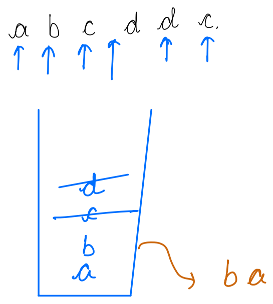
$\underline{a\ a}\ a\ b\ c\ \underbrace{c\ c}$:-
↑
abc.

a b c a b c a :-
↑ ↑ ↑ ↑ ↑ ↑ ↑



Pseudocode

```
String removeCons.DupCharacters(String input) {
    Stack<Character> st = new Stack<>();
    for(i=0; i<input.length(); i++) {
        char ch = input.charAt(i);
        if( !st.isEmpty() . && ch == st.peek()) {
            st.pop();
        } else {
            st.push(ch);
        }
    }
    String ans = "";
    while(!st.isEmpty()) {
        ans = ~~ans + st.pop();~~
              st.pop + ans;
    }
    return ans.~~reverse()~~;
}
```



TC: O(n)
SC: O(n)

Break: 8:40 – 8:50

a $\underbrace{b\ b}$ c $\underbrace{b\ b}$ c a c x

a $\underbrace{c\ c}$ a c x

$\underbrace{a\ a}$ c x

cx }

Qu Evaluate postfix expression

operands → operator

$2 + 3 \xrightarrow{\text{postfix}} 2\ 3\ +$

$4\ \underbrace{3\ \ 3\ *}_{3*3}\ +\ 2\ - \implies$ ans = 11

$\underbrace{4\ \ 9\ +}\ 2\ -$

$\underbrace{13\ \ 2\ -}$

11

3) $\underbrace{100\ \ 200\ +}\ 2\ /\ 5\ *\ 7\ +$

$\underbrace{300\ \ 2\ /}\ 5\ *\ 7\ +$

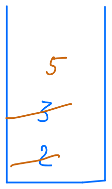
$\underbrace{150\ \ 5\ *}\ 7\ +$

$\underbrace{750\ \ 7\ +}$

757 Ans

Approach

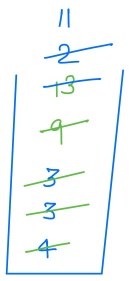
① $\underbrace{2\ \ 3\ \ +}_{5}$



② $4\ \underbrace{3\ \ 3\ \ *}_{9}\ +\ 2\ -$

13 2 −

11



ans = 11

$\underbrace{3\ \ 5\ +}_{8}\ 2\ -\ 2\ 5\ *\ -$

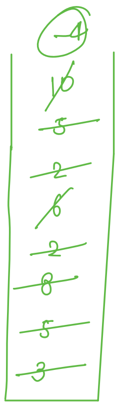
$\underbrace{8\ \ 2\ -}\ 2\ 5\ *\ -$

$6\ \underbrace{2\ \ 5\ *}\ -$

$6\ \ 10\ -$

$\boxed{-4}$

wrong exp.



Pseudocode

```
int evaluatePostfix(String input) {
        Stack<Character> st = new Stack<>();
        for(i=0; i<input.length(); i++) {
                char ch = input.charAt(i);
                if(ch is operand) {
                        st.push(ch);
                } else {
                        el2 = st.pop();
                        el1 = st.pop();
                        perform op on el1 & el2.
                        if(ch=='+') {
                                st.push( el1 + el2);
                        }
                        -
                        *        } h/w
                        /
                }
        }
        top el of stack = ans
        return st.pop();
}
```

TC: O(n)
SC: O(n)

Thankyou ☺