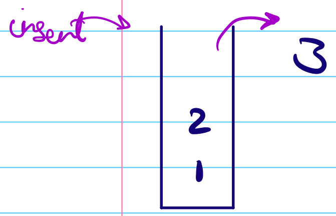
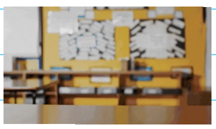
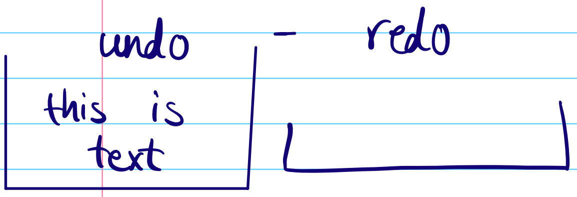
- intro
- Stack Code
- paranthisis
- Double character trouble
- Post fix evaluate

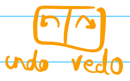
Stacks intro.

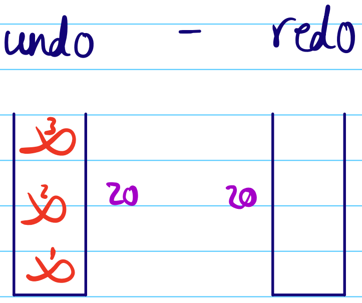
Last in / first out











operations

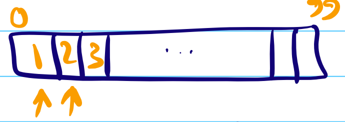
1. push(x) → insert x on top of the stack
2. pop() → removes from top of the stack
3. top / peek() → gets top element of the stack
4. isEmpty() → check if stack empty

   true / false

$\}$ O(1)

Max Size
ex 100

P1 Implement stack using an array



```
class Stack{
private int arr[MaxSize]
        int top = -1      ↘ top index
```

Constructor

```
stack(){
    stack(MaxSize=100)
}
Stack(int MaxSize){
    arr=new ...
    ...
}
```

ArrayList

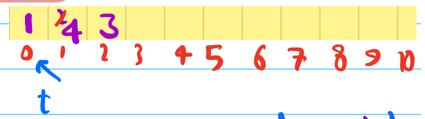
① push(x)

overflow?

```
void push(int x){
    top++          → if(top == MaxSize-1)
    if(top==MaxSize){throw exp, log error, panic ...
                      Stack overflow ...      top--
                     }
    arr[top] = X
}
```

dry run →

push(1) ✓	pop() 4
push(2) ✓	peek() 1
push(3) ✓	pop() 1
pop() 3	pop() under flow
pop() 2	
push(4) ✓	



top = ~~-1~~
-1 ~~0~~ ~~1~~ ~~0~~ ~~1~~ ~~2~~ ~~1~~ ~~0~~
maxSize=10

② pop()

underflow?

```
int pop(){
    if (isEmpty()) throw exp, log error, panic ...
                   Stack underflow...
    x = arr[top]
    top--
    ret x
}
```

③ top / peek()

```
int peek(){
    if (isEmpty()) throw exp, log error, panic ...
                   Stack underflow...
    ret arr[top]
}
```

④ isEmpty()

```
bool isEmpty(){
    ret top==-1
}
```

☑ optional HW ← LL

draw backs
waste memory
resterictive maxSize

true/false
P2 Check whether the given sequence of paranthesis/bracket/curly brace are valid.
( ) [ ] { }

ex ( )[{ }( )] ans=true   ex {{}[] ans=false

ex ( )[{(})] ans=false   ex)([] ans= false
if(...){(2+3) })
100 100

([{}[]][)] ans=false   ex {{[]}} ans=true

if({}...)   '(' ')' '[' ']' '{' '}'

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

```
initialize
init a stack<char>
for all chars in input{
    char c = input[i] //cur char
    if open (,{,[
        stack.push(c)
    else // means closing ),},]
        if (stack.isEmpty) ret false;    // )([]
        topChar=stack.Peek();
        if match(topChar,c){
            stack.pop()
        else // does'nt match
            ret false        // ([{}[]][)]
}
if (! stack.isEmpty) ret false;   // ex {{}[]
ret true
```

```
bool match(char c1, char c2){
    if(c1=='(' && c2==')')
        ret true
    else if... [ ]   ret true
    else if ... { }  ret true
    else ret false
}
```

Double character trouble

P3 Given a char array S, remove equal pair of consecutive characters multiple times as much as possible & return the final string.

ex a ~~bb~~ c → ac

ex a ~~bxxb~~ de → abbde → ade

ex a ~~bb~~ b ~~xx~~ ca → abca

Quiz abbcbbcacx → cx

⏱ 2 min

S= a b c d e e d c a b x x d

stack

a b a b d

TC: O(n)

SC: O(n)

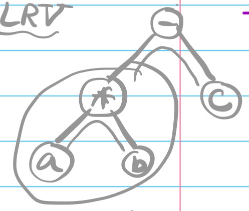
```
initialize stack
init. a stack<char>
for all chars in input { → i→0 ... s.len
    char c = S[i]
    if stack not empty && matches Peek then pop
    else push(c)
} out = ""  → // use StringBuilder for better performance
while (!stack.isEmpty) {
    out = out.append( stack.pop)
}
ret out.reverse();
```

removes double trouble

generates correct order of output

out = d b a b a

LVR
LRV



optional

Infix expession vs postfix expression

Infix expession	postfix expression
a + b	ab +
(a * b) - c	ab * c -
operand1, operator, operand2	operand1, operand2, operator

P4 Evalute a given postfix expression.

ex 10 6 - ans = 4

ex 2 1 * 3 + ans = 5

2 + 3

Quiz 3 5 + 2 - 2 5 * - ans = -4

8 6 10

*what if numbers are more than one char. 10999?

```
input <- string[]            {"3", "15", "+"}
n = input.len                  315 + 2 - 2 5 * -
for (i=0; i<n; i++){
  if input[i] is operator
     pop two nums from stack
     result = n1 operator n2     | if operator == '+' result = n1 + n2
                pop 2nd  pop 1st | if operator == '-' result = n1 - n2
     push(result)                |  :
  else                              18        int.parse( )
     push(num)
}
ret stack.Peak(); // if input is Correct stack will
                     exactly have one element which
                     is the result.
```

-4									

stack of operands

2 min

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