

Laboratory work report

Submitted by:

Madiyar Zhumabekov

IS-1812K

Submitted to:

Yeskendir Sultanov

**Laboratory work #4.**

Solve these problems using stack, queue, deque data structures.

Deadline: week 5

<https://leetcode.com/problems/valid-parentheses/>

**Task:** Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

An input string is valid if:

1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.

**Answer:**  Push brackets one by one. Once we have a open and closed brackets we pop the stack and if the stack is not empty we return false. In for loop, subtracting one or two from ch and comparing to top of stack because that is the difference between open and closed brackets in the ascii table.

**My code:**

class Solution {

public boolean isValid(String s) {

Stack<Character> stack = new Stack();

for(char ch: s.toCharArray()){

if(!stack.empty() && (stack.peek() == ch - 1 || stack.peek() == ch - 2)){

stack.pop();

}else{

stack.push(ch);

}

}

return stack.empty();

}

}

<https://leetcode.com/problems/min-stack/>

**Task:** Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

* push(x) -- Push element x onto stack.
* pop() -- Removes the element on top of the stack.
* top() -- Get the top element.
* getMin() -- Retrieve the minimum element in the stack.

**Answer:** In the push method, we add elements immediately by finding the minimum element and adding it. In the pop method, it deletes the last element, and gives the value in currMin the value of the penultimate element. In the top method, shows the last element without deleting it. In the getMin method, it returns the minimum element.

**My code:**

class MinStack {

private Stack<Integer> stack;

private int currMin;

/\*\* initialize your data structure here. \*/

public MinStack() {

stack = new Stack<Integer>();

}

public void push(int x) {

if(stack.isEmpty()){

currMin = x;

}

stack.push(currMin);

if(x < currMin){

currMin = x;

}

stack.push(x);

}

public void pop() {

stack.pop();

currMin = stack.pop();

}

public int top() {

return stack.peek();

}

public int getMin() {

return currMin;

}

}

/\*\*

\* Your MinStack object will be instantiated and called as such:

\* MinStack obj = new MinStack();

\* obj.push(x);

\* obj.pop();

\* int param\_3 = obj.top();

\* int param\_4 = obj.getMin();

\*/

<https://leetcode.com/problems/backspace-string-compare/>

**Task:** Given two strings S and T, return if they are equal when both are typed into empty text editors. # means a backspace character.

Note that after backspacing an empty text, the text will continue empty.

**Answer:** Used 2 of the stack. If character is not equal to#, we add it, and if it is equal, we delete the previous element. If they are not equal returns false.

**My code:**

class Solution {

public boolean backspaceCompare(String S, String T) {

Stack<Character> sStack = new Stack<>();

Stack<Character> tStack = new Stack<>();

for(char c: S.toCharArray()) {

if(c != '#') {

sStack.push(c);

}

else if(!sStack.isEmpty()) {

sStack.pop();

}

}

for(char c: T.toCharArray()) {

if(c != '#') {

tStack.push(c);

}

else if(!tStack.isEmpty()) {

tStack.pop();

}

}

while(!sStack.isEmpty()) {

char current = sStack.pop();

if(tStack.isEmpty() || tStack.pop() != current) {

return false;

}

}

return sStack.isEmpty() && tStack.isEmpty();

}

}

<https://leetcode.com/problems/evaluate-reverse-polish-notation/>

**Task:** Evaluate the value of an arithmetic expression in [Reverse Polish Notation](http://en.wikipedia.org/wiki/Reverse_Polish_notation).

Valid operators are +, -, \*, /. Each operand may be an integer or another expression.

**Note:**

* Division between two integers should truncate toward zero.
* The given RPN expression is always valid. That means the expression would always evaluate to a result and there won't be any divide by zero operation.

**Answer:** Using the stack. Adds elements if this arithmetic expression takes the last 2 elements and calculates and adds them to the stack.

**My code:**

class Solution {

public int evalRPN(String[] tokens) {

if(tokens == null || tokens.length==0) return 0;

Stack<Integer> stack = new Stack<>();

for(String token : tokens){

if (token.length()==1 && (token.charAt(0) == '+' || token.charAt(0) == '-' || token.charAt(0) == '/' || token.charAt(0) == '\*')) {

if (token.charAt(0) == '+') {

int val1 = stack.pop();

int val2 = stack.pop();

int result = val2 + val1;

stack.push(result);

} else if (token.charAt(0) == '-') {

int val1 = stack.pop();

int val2 = stack.pop();

int result = val2 - val1;

stack.push(result);

} else if (token.charAt(0) == '/') {

int val1 = stack.pop();

int val2 = stack.pop();

int result = val2 / val1;

stack.push(result);

} else if (token.charAt(0) == '\*') {

int val1 = stack.pop();

int val2 = stack.pop();

int result = val2 \* val1;

stack.push(result);

}

} else {

stack.push(Integer.parseInt(token));

}

}

return stack.pop();

}

}