



Department of Computer Science and Engineering

Name:-Harleen Kaur

UID:- 22BCS12125

Section:- 22KPIT-901-B

### DAY-3

Q.1. Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

Implement the MinStack class:

- MinStack() initializes the stack object.
- void push(int val) pushes the element val onto the stack.
- void pop() removes the element on the top of the stack.
- int top() gets the top element of the stack.
- int getMin() retrieves the minimum element in the stack.

You must implement a solution with  $O(1)$  time complexity for each function.

**Example 1:**

**Input**

```
["MinStack","push","push","push","getMin","pop","top","getMin"]  
[[],[-2],[0],[-3],[[],[],[],[]]]
```

**Output**

```
[null,null,null,null,-3,null,0,-2]
```

**Explanation**

```
MinStack minStack = new MinStack();  
minStack.push(-2);  
minStack.push(0);  
minStack.push(-3);  
minStack.getMin(); // return -3  
minStack.pop();  
minStack.top();    // return 0  
minStack.getMin(); // return -2
```

## Program Code:-

```
#include <stack>
#include <iostream>
using namespace std;

class MinStack {
private:
    stack<int> mainStack;
    stack<int> minStack;

public:
    MinStack() {}

    void push(int val) {
        mainStack.push(val);
        if (minStack.empty() || val <= minStack.top()) {
            minStack.push(val);
        }
    }

    void pop() {
        if (mainStack.top() == minStack.top()) {
            minStack.pop();
        }
        mainStack.pop();
    }

    int top() {
        return mainStack.top();
    }

    int getMin() {
        return minStack.top();
    }
};

int main() {
    MinStack minStack;
    minStack.push(-2);
    minStack.push(0);
    minStack.push(-3);
    cout << minStack.getMin() << endl;
    minStack.pop();
    cout << minStack.top() << endl;
    cout << minStack.getMin() << endl;
    return 0;
}
```

```
}
```

### Output:-

```
-3  
0  
-2  
PS C:\Users\DELL\Desktop> 
```

**Q2** Given a string *s*, find the first non-repeating character in it and return its index. If it does not exist, return -1.

#### Example 1:

**Input:** *s* = "leetcode"

**Output:** 0

#### Explanation:

The character 'l' at index 0 is the first character that does not occur at any other index.

#### Example 2:

**Input:** *s* = "loveleetcode"

**Output:** 2

#### Example 3:

**Input:** *s* = "aabb"

**Output:** -1

#### Constraints:

- $1 \leq s.length \leq 105$
- *s* consists of only lowercase English letters.

## Program Code:-

```
#include <iostream>
#include <string>
using namespace std;

class Solution {
public:
    int firstUniqChar(string s) {
        int lower[26] = {0};
        for (int i = 0; i < s.length(); i++) {
            lower[s[i] - 'a']++;
        }
        for (int i = 0; i < s.length(); i++) {
            if (lower[s[i] - 'a'] == 1) return i;
        }
        return -1;
    }
};

int main() {
    string s;
    cout << "Enter a string: ";
    cin >> s;
    Solution solution;
    int index = solution.firstUniqChar(s);
    if (index != -1) {
        cout << "The first unique character is at index: " << index << endl;
    } else {
        cout << "No unique character found in the string." << endl;
    }
    return 0;
}
```

## Output:-

```
Enter a string: yfxywsg
The first unique character is at index: 1
PS D:\Coding\CAMP>
```

### Q3

Implement a simple text editor. The editor initially contains an empty string, S. Perform Q operations of the following 4 types:

- append(W) - Append string W to the end of S.
- delete (k)- Delete the last k characters of S.
- print (k)- Print the k<sup>th</sup> character of S.
- undo() - Undo the last (not previously undone) operation of type 1 or 2, reverting S to the state it was in prior to that operation.

Example 1

S = 'abcde'

Ops=['1 fg', '3 6', '2 5', '4', '3 7', '4', '3 4']

operation

index S ops[index] explanation

0	abcde	1 fg	append fg
1	abcdefg	3 6	print the 6th letter - f
2	abcdefg	2 5	delete the last 5 letters
3	ab	4	undo the last operation, index 2
4	abcdefg	3 7	print the 7th character - g
5	abcdefg	4	undo the last operation, index 0
6	abcde	3 4	print the 4th character – d

The results should be printed as:

f

g

d

Input Format

The first line contains an integer, Q, denoting the number of operations.

Each line i of the Q subsequent lines (where  $0 \leq i < Q$ ) defines an operation to be performed. Each operation starts with a single integer, t (where  $t \in \{1, 2, 3, 4\}$ ), denoting a type of operation as defined in the Problem Statement above. If the operation requires an argument, t is followed by its space-separated argument. For example, if  $t=1$  and  $W="abcd"$ , line i will be 1 abcd.

## Program Code:-

```
#include <iostream>
#include <vector>
#include <stack>
#include <string>
using namespace std;

class TextEditor {
private:
    string s;
    stack<string> undoStack;

public:
    TextEditor() {}

    void append(string w) {
        undoStack.push("1 " + w);
        s += w;
    }

    void deleteChars(int k) {
        string deleted = s.substr(s.size() - k);
        undoStack.push("2 " + deleted);
        s = s.substr(0, s.size() - k);
    }

    void print(int k) {
        cout << s[k - 1] << endl;
    }

    void undo() {
        if (undoStack.empty()) return;

        string operation = undoStack.top();
        undoStack.pop();

        if (operation[0] == '1') {
            s = s.substr(0, s.size() - operation.substr(2).size());
        } else if (operation[0] == '2') {
            s += operation.substr(2);
        }
    }
};

int main() {
    TextEditor editor;
```

```

    editor.append("abc");
    editor.print(3);    // c
    editor.deleteChars(3);
    editor.append("xy");
    editor.print(2);    // y
    editor.undo();
    editor.print(1);    // a
    return 0;
}

```

## Output:-

```

pp -o A3 } ; if ($?) { .\A3 }
c
y

```

Q4.

**Implement a first in first out (FIFO) queue using only two stacks. The implemented queue should support all the functions of a normal queue (push, peek, pop, and empty).**

**Implement the MyQueue class:**

**void push(int x)** Pushes element x to the back of the queue.

**int pop()** Removes the element from the front of the queue and returns it.

**int peek()** Returns the element at the front of the queue.

**boolean empty()** Returns true if the queue is empty, false otherwise.

**Notes:**

You must use only standard operations of a stack, which means only push to top, peek/pop from top, size, and is empty operations are valid.

Depending on your language, the stack may not be supported natively. You may simulate a stack using a list or deque (double-ended queue) as long as you use only a stack's standard operations.

**Example 1:**

**Input**

["MyQueue", "push", "push", "peek", "pop", "empty"]

[[], [1], [2], [], [], []]

**Output**

[null, null, null, 1, 1, false]

**Explanation**

- MyQueue myQueue = new MyQueue();
- myQueue.push(1); // queue is: [1]
- myQueue.push(2); // queue is: [1, 2] (leftmost is front of the queue)
- myQueue.peek(); // return 1
- myQueue.pop(); // return 1, queue is [2]

- myQueue.empty(); // return false

## Program Code:-

```
#include <stack>
#include <iostream>
using namespace std;

class MyQueue {
private:
    stack<int> stack1, stack2;

public:
    MyQueue() {}

    void push(int x) {
        stack1.push(x);
    }
    int pop() {
        if (stack2.empty()) {
            while (!stack1.empty()) {
                stack2.push(stack1.top());
                stack1.pop();
            }
        }
        int val = stack2.top();
        stack2.pop();
        return val;
    }
    int peek() {
        if (stack2.empty()) {
            while (!stack1.empty()) {
                stack2.push(stack1.top());
                stack1.pop();
            }
        }
        return stack2.top();
    }
    bool empty() {
        return stack1.empty() && stack2.empty();
    }
};

int main() {
    MyQueue queue;
    queue.push(1);
    queue.push(2);
```



```

    cout << queue.peek() << endl;
    cout << queue.pop() << endl;
    cout << queue.empty() << endl;
    return 0;
}

```

### Output:-

```

1
1
0
PS C:\Users\DELL\Desktop>

```

Q5. You are given an array of strings tokens that represents an arithmetic expression in a Reverse Polish Notation.

Evaluate the expression. Return an integer that represents the value of the expression.

Note that:

- The valid operators are '+', '-', '\*', and '/'.
- Each operand may be an integer or another expression.
- The division between two integers always truncates toward zero.
- There will not be any division by zero.
- The input represents a valid arithmetic expression in a reverse polish notation.
- The answer and all the intermediate calculations can be represented in a 32-bit integer.

Example 1:

Input: tokens = ["2","1","+","3","\*"]

Output: 9

Explanation:  $((2 + 1) * 3) = 9$

### Program Code:-

```

#include <iostream>
#include <stack>
#include <vector>
#include <string>
#include <cstdlib>
using namespace std;

```

```

int evalRPN(vector<string>& tokens) {
    stack<int> stk;
    for (string& token : tokens) {
        if (token == "+" || token == "-" || token == "*" || token == "/") {
            int b = stk.top();
            stk.pop();
            int a = stk.top();
            stk.pop();

            if (token == "+") stk.push(a + b);
            else if (token == "-") stk.push(a - b);
            else if (token == "*") stk.push(a * b);
            else if (token == "/") stk.push(a / b);
        } else {
            stk.push(atoi(token.c_str()));
        }
    }
    return stk.top();
}

int main() {
    vector<string> tokens = {"2", "1", "+", "3", "*"};
    cout << evalRPN(tokens) << endl;

    tokens = {"4", "13", "5", "/", "+"};
    cout << evalRPN(tokens) << endl;

    tokens = {"10", "6", "9", "3", "+", "-11", "*", "/", "*", "17", "+", "5", "+"};
    cout << evalRPN(tokens) << endl;

    return 0;
}

```

**Output:-**

```

9
6
22
PS C:\Users\DELL\Desktop>

```