**STACK AND QUEUE STANDARD QUESTION**

**VERY EASY:**

## 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

## Example 2:

**Input:**

["MinStack", "push", "push", "push", "push", "getMin", "pop", "getMin", "top", "getMin"]

[[], [5], [3], [7], [3], [], [], [], [], []]

## Output

[null, null, null, null, null, 3, null, 3, 7, 3]

## Explanation:

MinStack minStack = new MinStack(); minStack.push(5); # Stack: [5], MinStack: [5] minStack.push(3); # Stack: [5, 3],

MinStack: [5, 3] minStack.push(7); # Stack: [5, 3, 7], MinStack:

[5, 3] minStack.push(3); # Stack: [5, 3, 7, 3], MinStack: [5, 3, 3] minStack.getMin(); # Returns 3 minStack.pop(); # Removes 3; Stack: [5, 3, 7], MinStack: [5, 3] minStack.getMin(); # Returns 3 minStack.top(); # Returns 7 minStack.getMin(); # Returns 3

* + Minimum values are maintained as: [5] → [5, 3] → [5, 3] → [5, 3] ●

After pops, the minimum values update accordingly.

## Example 3:

**Input:**

["MinStack", "push", "push", "push", "getMin", "pop", "getMin", "pop", "getMin"]

[[], [2], [1], [4], [], [], [], [], []]

## Output:

[null, null, null, null, 1, null, 1, null, 2] **Explanation:**

minStack = MinStack() minStack.push(2) minStack.push(1) minStack.push(4) minStack.push(1)

print(minStack.getMin()) # Output: 1 minStack.pop() print(minStack.getMin()) # Output: 1 minStack.pop() print(minStack.getMin()) # Output: 1 minStack.pop()

print(minStack.getMin()) # Output: 2

* + Minimum values are maintained as: [2] → [2, 1] → [2, 1] → [2, 1] ● After pops, the minimum values update accordingly.

## Constraints:

* + -2^31 <= val <= 2^31 - 1
  + Methods pop, top and getMin operations will always be called on non-empty stacks.
  + At most 3 \* 10^4 calls will be made to push, pop, top, and getMin.

Ø Sources[**:**](https://leetcode.com/problems/min-stack/)[**https://leetcode.com/problems/min-stack/**](https://leetcode.com/problems/min-stack/)

# Code

#include <iostream> #include <stack> #include <climits>

class MinStack { private: std::stack<int> stack;

std::stack<int> minStack; public:

MinStack() {}

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

}

}

void pop() { if (stack.top() == minStack.top()) { minStack.pop();

}

stack.pop();

}

int top() { return stack.top();

}

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

}

};

int main() {

MinStack minStack;

minStack.push(-2); minStack.push(0); minStack.push(-3); std::cout << minStack.getMin() << std::endl; minStack.pop(); std::cout << minStack.top()

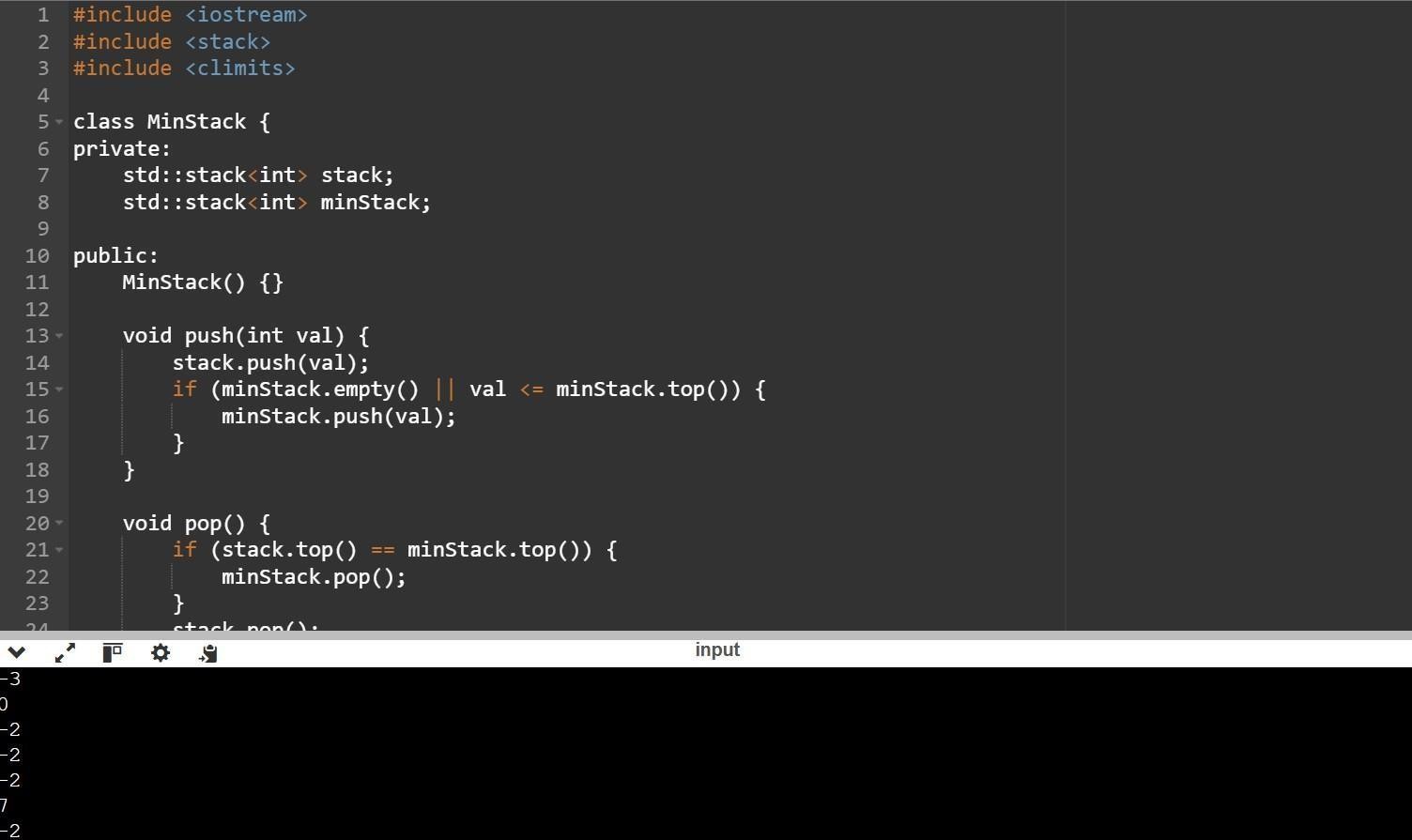
<< std::endl; std::cout << minStack.getMin() << std::endl;

minStack.push(5); minStack.push(3); minStack.push(7); minStack.push(3); std::cout << minStack.getMin() << std::endl; minStack.pop(); std::cout << minStack.getMin() << std::endl; std::cout

<< minStack.top() << std::endl; std::cout << minStack.getMin() << std::endl;

return 0; }

# Output

****

## 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 <= s.length <= 105
* s consists of only lowercase English letters.

## Approach:

Use a hash map or array of size 26 to store the frequency of each character (since the input consists of lowercase English letters).

Traverse the string to count the frequency of each character.

Traverse the string again to find the first character with a frequency of 1. Return its index. If no such character is found, return -1.

**Time Complexity:** ( O(n) ), where ( n ) is the length of the string. The string is traversed twice.

**Space Complexity:** ( O(1) ), as the frequency array has a fixed size of 26.

Reference : <https://leetcode.com/problems/first-unique-character-in-a-> [string/description/](https://leetcode.com/problems/first-unique-character-in-a-)

# Code

#include <iostream> #include <string> #include <vector>

int firstUniqChar(const std::string& s) { std::vector<int> frequency(26, 0);

for (char c : s) { frequency[c - 'a']++;

}

for (int i = 0; i < s.length(); i++) { if (frequency[s[i] - 'a'] == 1) {

return i;

} }

return -1;

}

int main() {

std::string s1 = "leetcode"; std::string s2 = "loveleetcode"; std::string s3 = "aabb";

std::cout << firstUniqChar(s1) << std::endl;

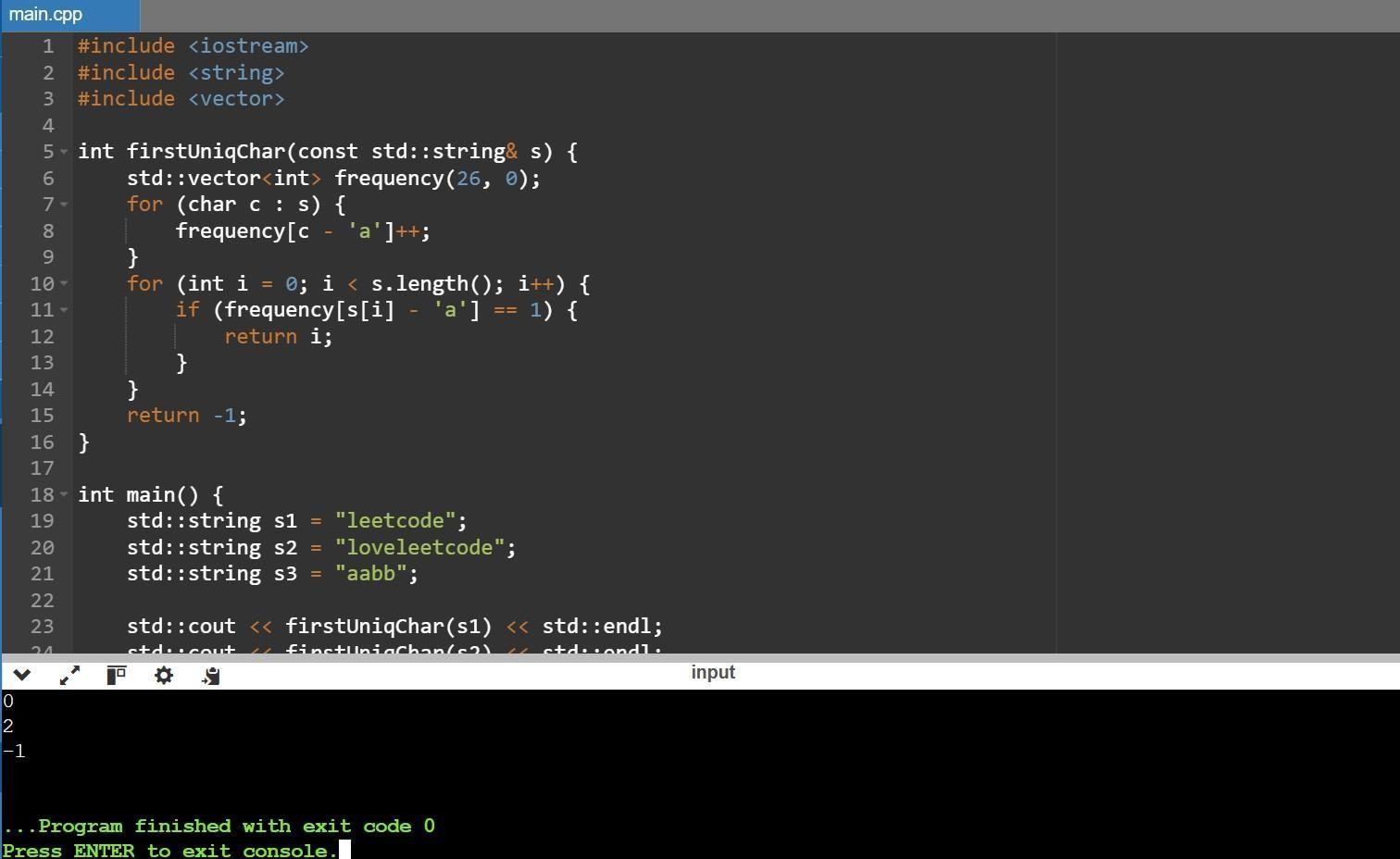
std::cout << firstUniqChar(s2) << std::endl; std::cout

<< firstUniqChar(s3) << std::endl;

return 0;

}

# Output

****

## 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^th 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 characgter - 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≤i<Q ) defines an operation to be performed. Each operation starts with a single integer, t (where t  {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.

Example 2 (Custom Test Case)

## Input 9

1. hello
2. world

3 10

1. 5
2. 5

4

3 10

4

3 1

## Code Execution

operations = [

"1 hello", "1 world", "3 10", "2 5", "3 5",

"4", "3 10", "4", "3 1"

]

result = text\_editor(operations) print("\n".join(result)) # Outputs: d, o, d, h **Output**

d o d h

Example 3 (Custom Test Case)

## Input

10

1. programming

3 1

1. 6
2. 4
3. code

3 4

4

3 7

4

3 8

## Code Execution

operations = [

"1 programming", "3 1", "2 6", "3 4", "1 code",

"3 4", "4", "3 7", "4", "3 8"

]

result = text\_editor(operations) print("\n".join(result)) # Outputs: p, g, o, m **Output**

p g o m

## Constraints

* 1≤ Q ≤ 106
* 1 ≤ k ≤ |S|
* The sum of the lengths of all in the input ≤ 106 .
* The sum of over all delete operations ≤ 2.106 .
* All input characters are lowercase English letters.
* It is guaranteed that the sequence of operations given as input is possible to perform.

## Output Format

Each operation of type 3 must print the k^th character on a new line.

## Sample Input STDIN Function

**----- --------**

8 Q = 8

1 abc ops[0] = '1 abc'

3 3 ops[1] = '3 3'

2 3 ...

1 xy

3 2

4

4

3 1

## Sample Output

c y a

## Explanation

1. Initially,S is empty. The following sequence of 8 operations are described below:
2. S=””. We append abc to S , so S = “abc”.
3. Print the 3rd character on a new line. Currently, the 3rd character is c.
4. Delete the last 3 characters in S(abc), so S= “”.
5. Append xy to S , so S= “xy”.
6. Print the 2nd character on a new line. Currently, the 2nd character is y.
7. Undo the last update to S, making S empty again (i.e.,S=”” ).
8. Undo the next to last update to S (the deletion of the last 3 characters), making S=”abc”.
9. Print the 1st character on a new line. Currently, the 1st character is a.
   * **Frequently Asked By**: Facebook, Atlassian, Adobe(**Year**: 2018–2022)
   * Reference: <https://www.hackerrank.com/challenges/simple-text-editor>

# Code

#include <iostream> #include <stack> #include <string> #include <vector>

int main() { int Q; std::cin >> Q;

std::string S; std::stack<std::pair<int, std::string>> history; std::vector<char> output;

while (Q--) { int t;

std::cin >> t;

if (t == 1) { std::string W; std::cin >> W; history.push({1, W}); S +=

W; } else if (t

== 2) { int k; std::cin >> k; history.push({2, S.substr(S.size() - k)});

S.erase(S.size() - k); } else if (t == 3) { int k; std::cin >> k; output.push\_back(S[k - 1]); } else if (t == 4) { if (!history.empty()) { auto last = history.top();

history.pop(); if (last.first

== 1) {

S.erase(S.size() - last.second.size());

} else if (last.first == 2) { S += last.second;

}

}

}

}

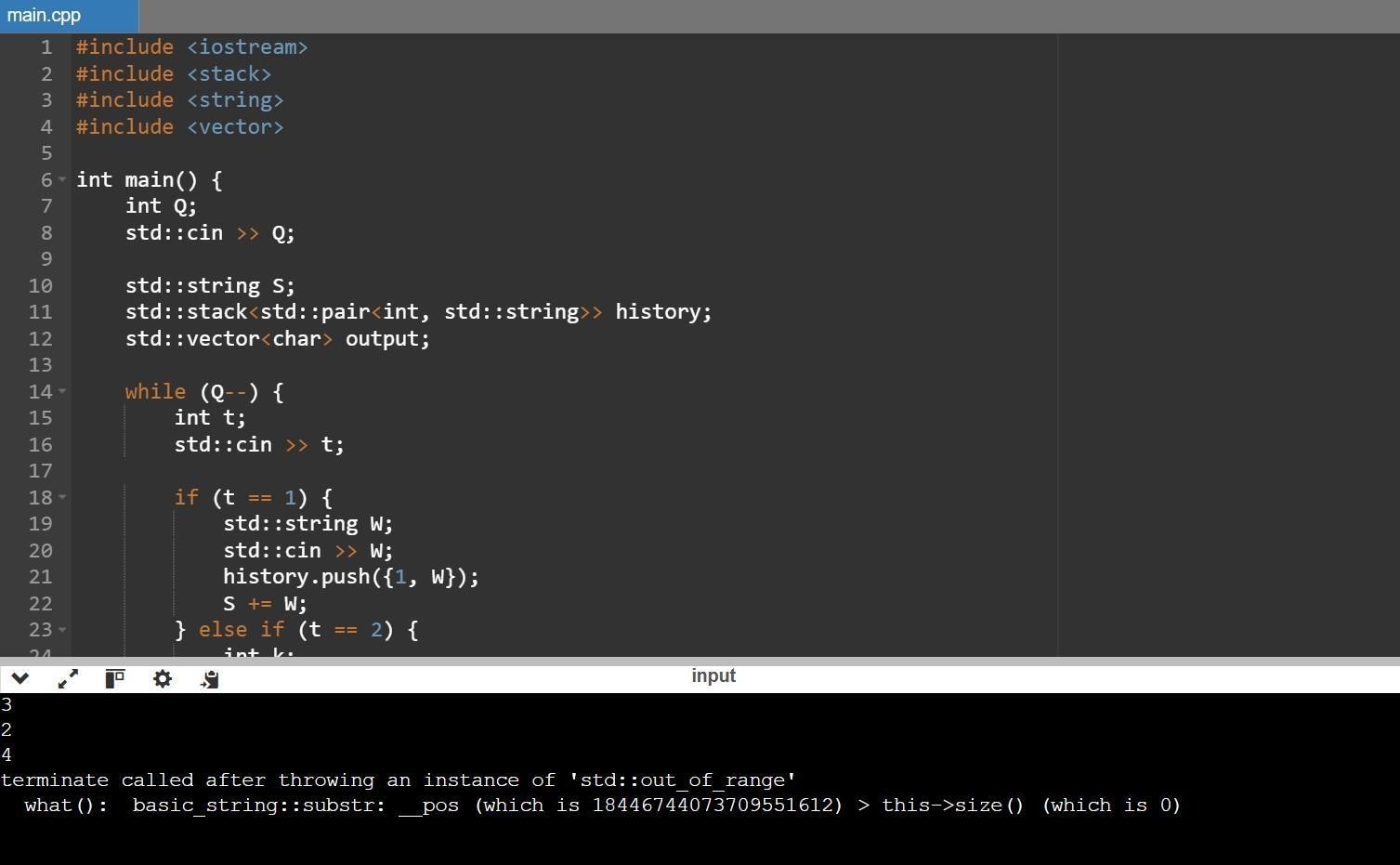
for (char c : output) { std::cout << c << std::endl;

}

return 0;

}

# Output

****

1. **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

## Example 2:

**Input:**

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

[[], [3], [5], [7], [], [], [], []]

## Output:

[null, null, null, null, 3, 3, 5, false] **Explanation:**

MyQueue myQueue = new MyQueue() myQueue.push(3) # queue is: [3] myQueue.push(5) # queue is: [3, 5]

myQueue.push(7) # queue is: [3, 5, 7] myQueue.peek() # return 3 myQueue.pop() # return 3, queue is: [5, 7] myQueue.peek() # return 5 myQueue.empty() # return False

## Example 3:

**Input:**

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

[[], [8], [], [10], [], [], []]

## Output:

[null, null, 8, null, 8, 10, true] **Explanation:**

MyQueue myQueue = new MyQueue() myQueue.push(8) # queue is: [8] myQueue.peek() # return 8 myQueue.push(10) # queue is: [8, 10] myQueue.pop() # return 8, queue is: [10] myQueue.pop() # return 10, queue is: []

myQueue.empty() # return True

## Constraints:

* 1 <= x <= 9
* At most 100 calls will be made to push, pop, peek, and empty.
* All the calls to pop and peek are valid.

**Asked By :** Amazon,Google

**Reference** [:](https://leetcode.com/problems/implement-queue-using-stacks/description/) <https://leetcode.com/problems/implement-queue-using-stacks/description/>

# Code

#include <iostream> #include <stack>

class MyQueue { private:

std::stack<int> stack1, stack2;

void transferStack() { if (stack2.empty()) { while (!stack1.empty()) { stack2.push(stack1.top()); stack1.pop();

}

}

}

public:

MyQueue() {}

void push(int x) { stack1.push(x);

}

int pop() { transferStack(); int front

= stack2.top();

stack2.pop(); return front;

}

int peek() { transferStack(); return stack2.top();

}

bool empty() { return stack1.empty() && stack2.empty();

}

};

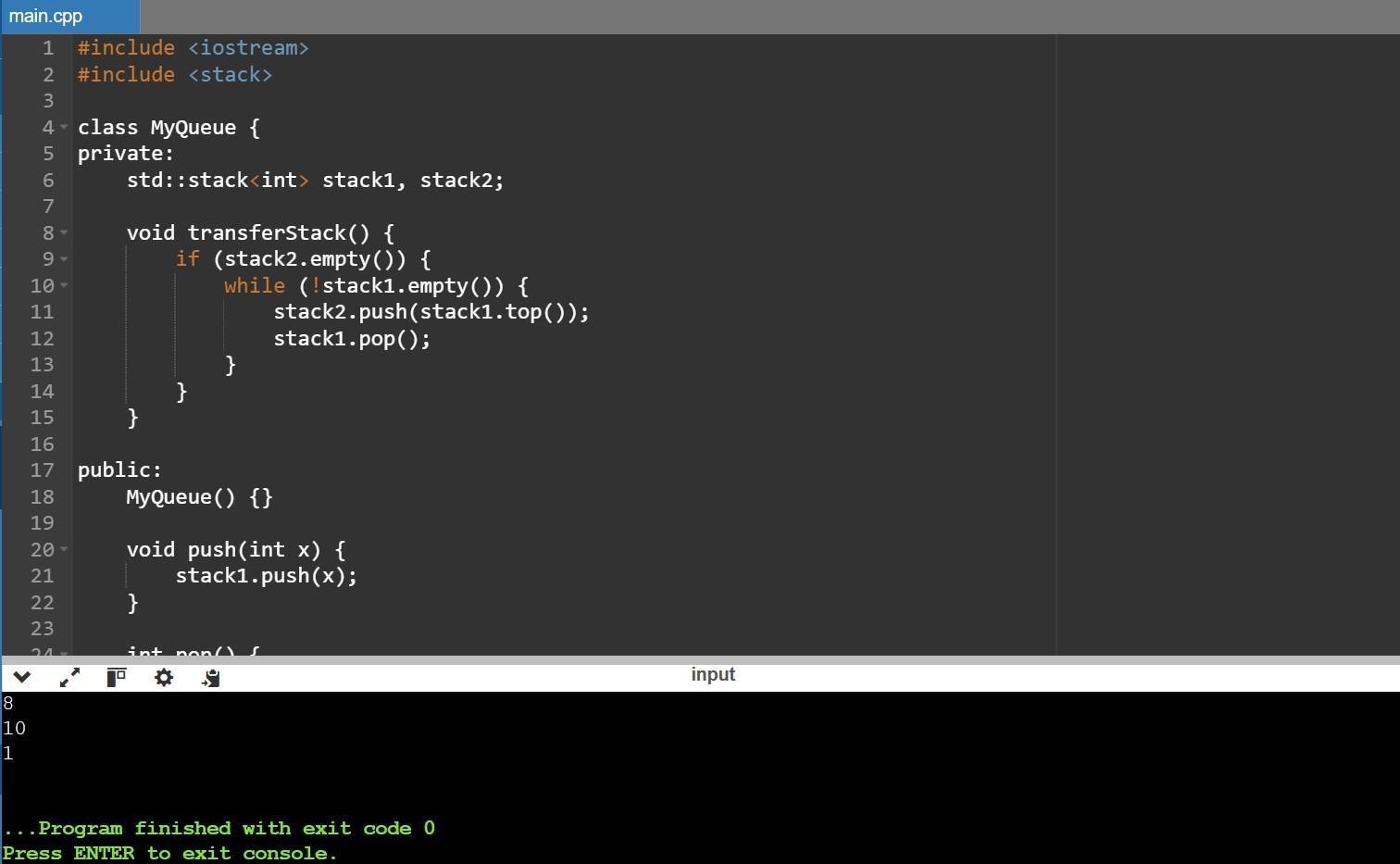
int main() {

MyQueue myQueue; myQueue.push(3); myQueue.push(5); myQueue.push(7); std::cout << myQueue.peek() << std::endl; std::cout << myQueue.pop() << std::endl; std::cout << myQueue.peek() << std::endl; std::cout << myQueue.empty() << std::endl;

MyQueue anotherQueue; anotherQueue.push(8); std::cout << anotherQueue.peek() << std::endl; anotherQueue.push(10); std::cout << anotherQueue.pop() << std::endl; std::cout << anotherQueue.pop() << std::endl; std::cout << anotherQueue.empty() << std::endl;

return 0; }

# Output

****

## 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

## Example 2:

**Input: tokens** = ["4","13","5","/","+"]

## Output: 6

**Explanation:** (4 + (13 / 5)) = 6

## Example 3:

**Input: tokens =** ["10","6","9","3","+","-11","\*","/","\*","17","+","5","+"]

## Output: 22

**Explanation:** ((10 \* (6 / ((9 + 3) \* -11))) + 17) + 5

= ((10 \* (6 / (12 \* -11))) + 17) + 5

= ((10 \* (6 / -132)) + 17) + 5

= ((10 \* 0) + 17) + 5

= (0 + 17) + 5

= 17 + 5

= 22

## Constraints:

* + 1 <= tokens.length <= 104
  + tokens[i] is either an operator: "+", "-", "\*", or "/", or an integer in the range [-200, 200].

## Approach

Using Fundamentals of STACK && LAMBDA...

**Time complexity:** O(n)

**Space complexity:** O(n)

# Code

#include <iostream> #include <vector> #include <stack> #include <string>

int evalRPN(std::vector<std::string>& tokens) { std::stack<int> stack;

for (const auto& token : tokens) { if (token == "+" || token

== "-" || token == "\*" || token == "/") { int b = stack.top(); stack.pop(); int a = stack.top(); stack.pop();

if (token == "+") stack.push(a + b); else if (token == "-") stack.push(a - b); else if (token == "\*") stack.push(a \* b); else if (token == "/") stack.push(a / b);

} else { stack.push(std::stoi(token));

}

}

return stack.top();

}

int main() { std::vector<std::string> tokens1 = {"2", "1", "+", "3", "\*"}; std::vector<std::string>

tokens2 = {"4", "13", "5", "/", "+"}; std::vector<std::string> tokens3 = {"10", "6", "9", "3", "+", "-

11", "\*", "/", "\*", "17", "+", "5", "+"};

std::cout << evalRPN(tokens1) << std::endl; std::cout << evalRPN(tokens2) << std::endl; std::cout << evalRPN(tokens3) << std::endl;

return 0;

}

**Output**

