Name: Arin Rai

UID: 22BCS11773

Section: KPIT_901

Day 4

1. Balanced Parentheses

Question:
Given a string s consisting of characters (,), {, }, [, and], determine if the string is balanced. A string is balanced if:

1. Every opening bracket has a corresponding closing bracket.
2. Brackets are closed in the correct order.

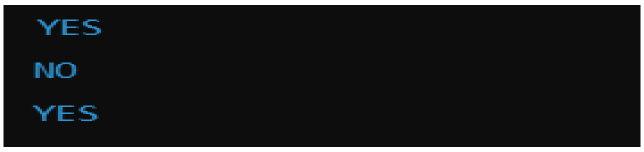
Return "YES" if the string is balanced, otherwise "NO".
Input Format:
The first line contains an integer n, the number of test cases.
Each of the next n lines contains a string s.

Output Format:
For each test case, print "YES" or "NO" based on whether the string is balanced.

Code:

```
#include <iostream>

#include <stack>

#include <string>

using namespace std;


bool isBalanced(string s) {

   stack<char> st;

   for (char c : s) {

      if (c == '(' || c == '{' || c == '[') {

         st.push(c);

      } else {
            if (st.empty()) return false;
            if ((c == ')' && st.top() != '(') ||
                (c == '}' && st.top() != '{') ||
                (c == ']' && st.top() != '[')) return false;
            st.pop();
        }
    }
    return st.empty();
}

int main() {
    int n;
    cin >> n;
    while (n--) {
        string s;
        cin >> s;
        cout << (isBalanced(s) ? "YES" : "NO") << endl;
    }
    return 0;
}
```

Output:



```
YES
NO
YES
```

4. Evaluate Reverse Polish Notation

Question:

You are given an array of strings representing an arithmetic expression in Reverse Polish Notation (RPN). Evaluate the expression and return the result.

The valid operators are +, -, *, and /. Each operand can be an integer or another expression.

- Division between two integers should truncate toward zero.
- It is guaranteed that the input is always a valid RPN expression.

Examples:

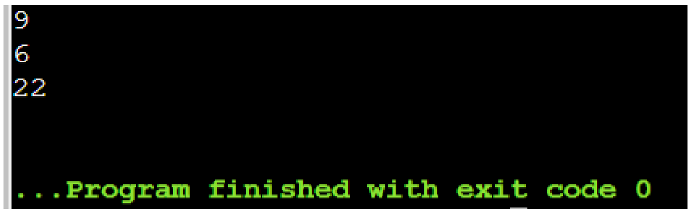
- Input: ["2", "1", "+", "3", "*"] → Output: 9 (Explanation: ((2 + 1) * 3))
- Input: ["4", "13", "5", "/", "+"] → Output: 6 (Explanation: (4 + (13 / 5)))

Code:

```
#include <iostream>

#include <vector>

#include <stack>

#include <string>

using namespace std;


int evalRPN(vector<string>& tokens) {

  stack<int> st;

  for (string& token : tokens) {

    if (token == "+" || token == "-" || token == "*" || token == "/") {

      int b = st.top(); st.pop();

      int a = st.top(); st.pop();

      if (token == "+") st.push(a + b);

      else if (token == "-") st.push(a - b);
```

```
            else if (token == "*") st.push(a * b);
            else st.push(a / b);
        } else {
            st.push(stoi(token));
        }
    }
    return st.top();
}

int main() {
    vector<string> tokens1 = {"2", "1", "+", "3", "*"};
    vector<string> tokens2 = {"4", "13", "5", "/", "+"};
    vector<string> tokens3 = {"10", "6", "9", "3", "+", "-11", "*", "/", "*", "17", "+",
"5", "+"};

    cout << evalRPN(tokens1) << endl;  // Output: 9
    cout << evalRPN(tokens2) << endl;  // Output: 6
    cout << evalRPN(tokens3) << endl;  // Output: 22
    return 0;
}
```

Output:



```
9
6
22


...Program finished with exit code 0
```

3. FIFO Queue Using Two Stacks

Question:
Implement a queue using two stacks. The queue should support the following operations:

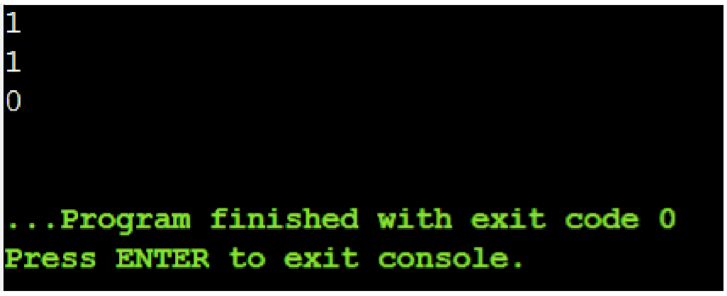
1. Push: Add an element to the end of the queue.
2. Pop: Remove the element from the front of the queue.
3. Peek: Return the front element of the queue.
4. Empty: Return whether the queue is empty or not.

Code:

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

class MyQueue {
  stack<int> stack1, stack2;

  void transfer() {
    while (!stack1.empty()) {
      stack2.push(stack1.top());
      stack1.pop();
    }
  }

public:
  void push(int x) {
    stack1.push(x);
  }
```

```
    int pop() {
        if (stack2.empty()) transfer();
        int top = stack2.top();
        stack2.pop();
        return top;
    }

    int peek() {
        if (stack2.empty()) transfer();
        return stack2.top();
    }

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

int main() {
    MyQueue q;
    q.push(1);
    q.push(2);
    cout << q.peek() << endl;  // Output: 1
    cout << q.pop() << endl;   // Output: 1
    cout << q.empty() << endl; // Output: 0
    return 0;
}
```

Output:



```
1
1
0


...Program finished with exit code 0
Press ENTER to exit console.
```

4. Simple Text Editor

Question:

You need to implement a simple text editor that performs the following operations:

1. **Append**: Add a string w to the end of the current text.
2. **Delete**: Remove the last k characters from the current text.
3. **Print**: Print the k-th character of the text (1-based index).
4. **Undo**: Revert the text to the state it was in before the last append or delete operation.

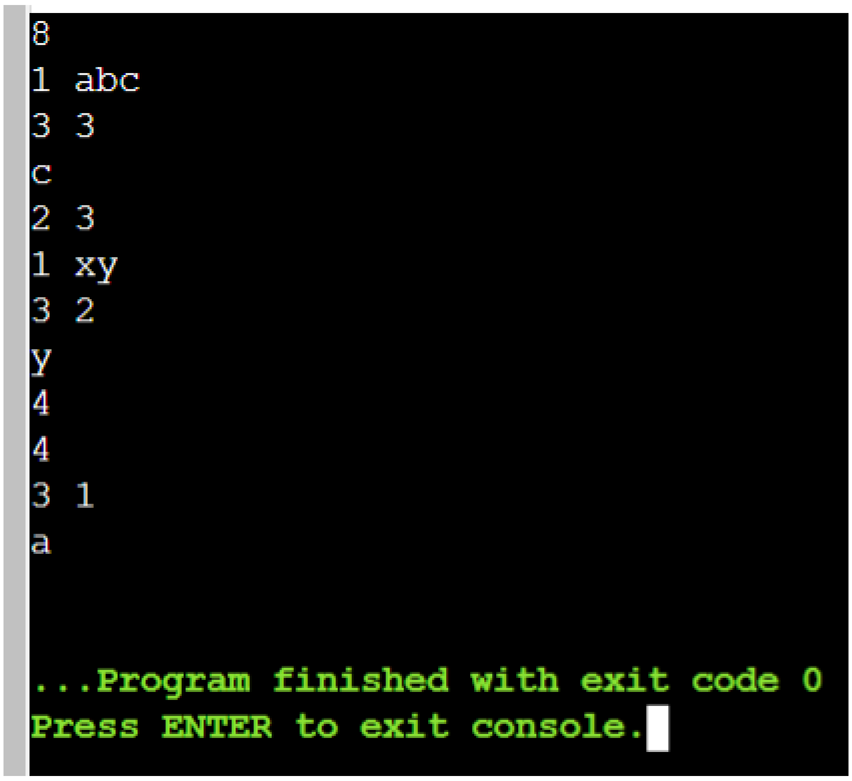
You will be given a sequence of operations to perform, and your task is to implement these efficiently.

Output:

```
#include <iostream>

#include <stack>

#include <string>

using namespace std;


int main() {

  stack<string> history;

  string s = "";

  int q;
```

```
    cin >> q;

    while (q--) {
        int type;
        cin >> type;
        if (type == 1) { // append
            string w;
            cin >> w;
            history.push(s);
            s += w;
        } else if (type == 2) { // delete
            int k;
            cin >> k;
            history.push(s);
            s.erase(s.size() - k);
        } else if (type == 3) { // print
            int k;
            cin >> k;
            cout << s[k - 1] << endl;
        } else if (type == 4) { // undo
            s = history.top();
            history.pop();
        }
    }
    return 0;
}
```

Output:



```
8
1 abc
3 3
c
2 3
1 xy
3 2
y
4
4
3 1
a


...Program finished with exit code 0
Press ENTER to exit console.
```

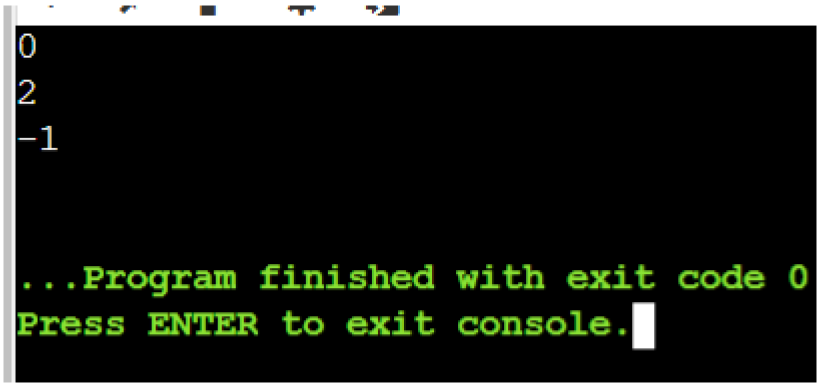
5.Question:
Given a string s, find the first non-repeating character in it and return its index. If it does not exist, return -1.

Code:

```
#include <iostream>

#include <string>

#include <unordered_map>

using namespace std;


int firstUniqueChar(string s) {

   unordered_map<char, int> freq;

   for (char c : s) freq[c]++;

   for (int i = 0; i < s.size(); i++) {

      if (freq[s[i]] == 1) return i;
```

```
    }
    return -1;
}

int main() {
    cout << firstUniqueChar("leetcode") << endl;      // Output: 0
    cout << firstUniqueChar("loveleetcode") << endl;  // Output: 2
    cout << firstUniqueChar("aabb") << endl;          // Output: -1
    return 0;
}
```

Output:



```
0
2
-1


...Program finished with exit code 0
Press ENTER to exit console.
```