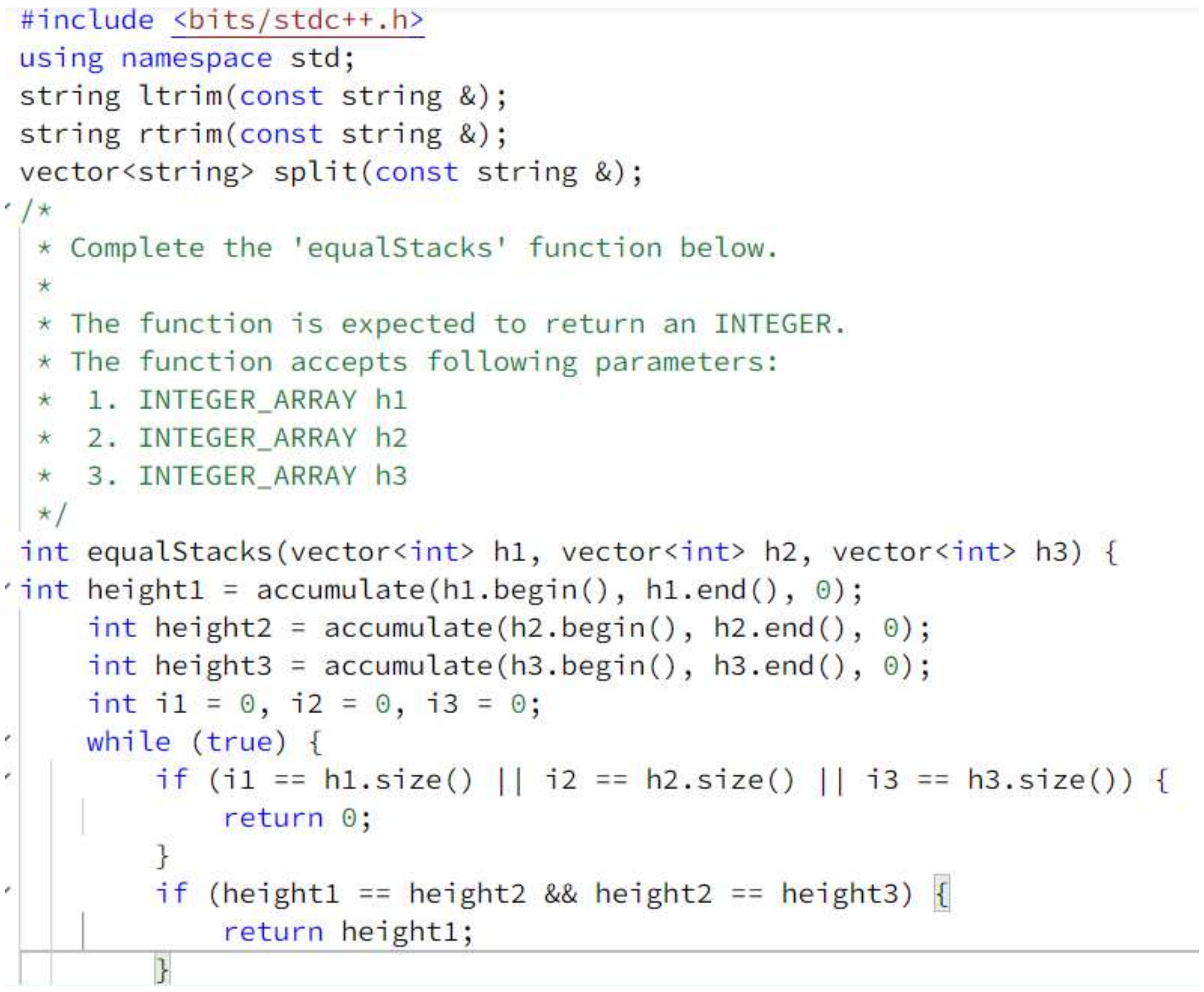
Experiment1.2

Student Name: Virat Samdarshi **UID: 22BCS12648**
Branch: B.E(CSE) **Section/Group: IOT_627-B**
Semester: Fifth **Date of Performance: 25/07/24**
Subject Name: AP LAB 1 **Subject Code: 22CSP-314**

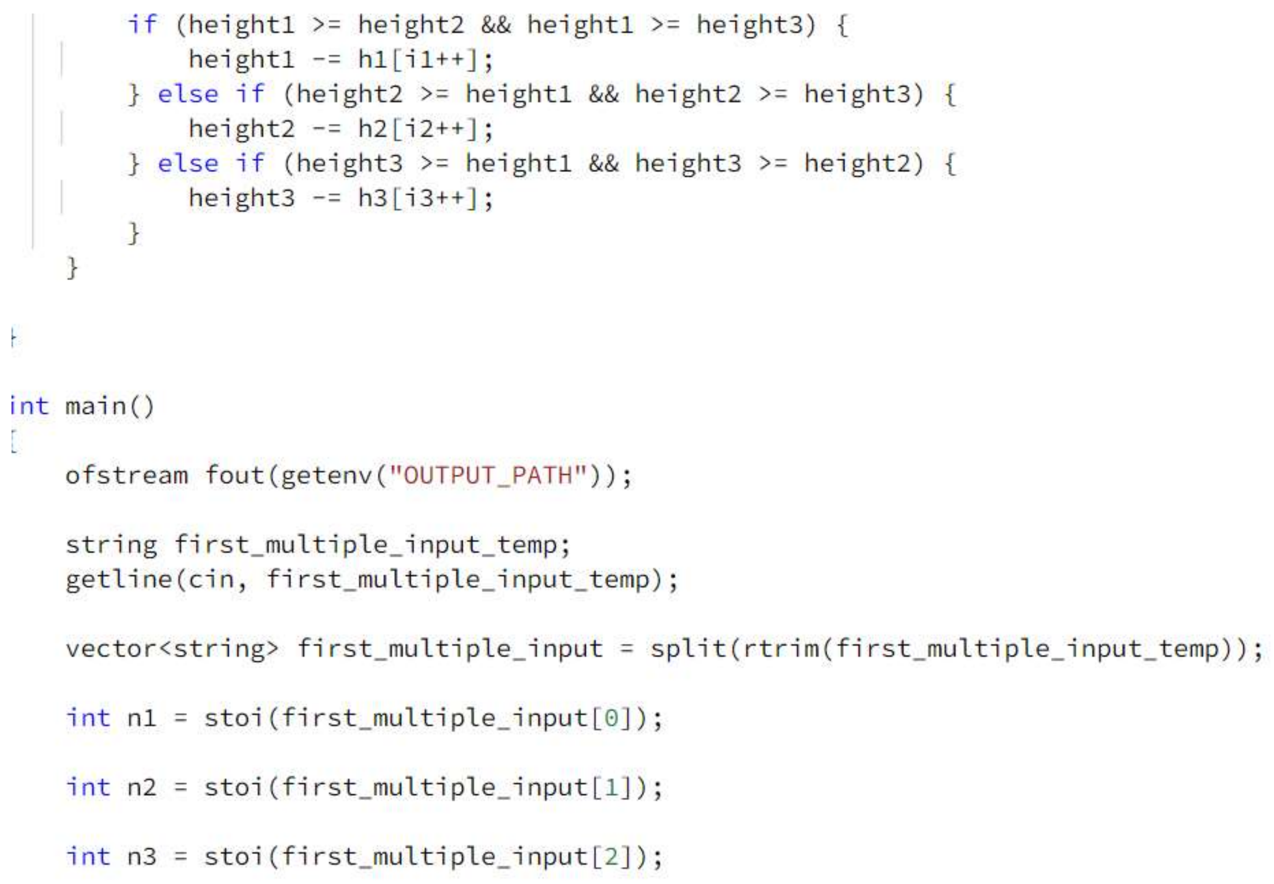
1. **Aim:** You have three stacks of cylinders where each cylinder has the same diameter, but they may vary in height. You can change the height of a stack by removing and discarding its topmost cylinder any number of times.
   Find the maximum possible height of the stacks such that all of the stacks are exactly the same height. This means you must remove zero or more cylinders from the top of zero or more of the three stacks until they are all the same height, then return the height.

2. **Objective:** Determine the maximum possible height such that all three stacks of cylinders are of equal height after removing any number of topmost cylinders from each stack.

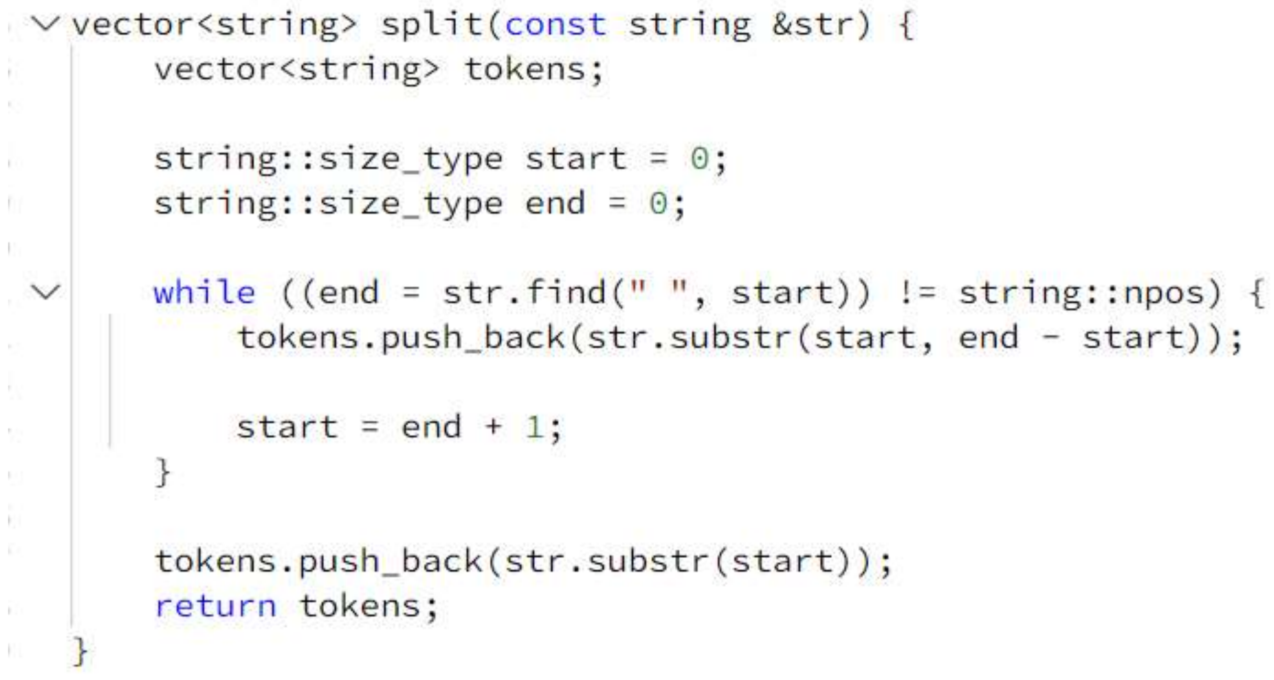
3. **Implementation/Code :-**



```
#include <bits/stdc++.h>
using namespace std;
string ltrim(const string &);
string rtrim(const string &);
vector<string> split(const string &);
/*
 * Complete the 'equalStacks' function below.
 *
 * The function is expected to return an INTEGER.
 * The function accepts following parameters:
 *  1. INTEGER_ARRAY h1
 *  2. INTEGER_ARRAY h2
 *  3. INTEGER_ARRAY h3
 */
int equalStacks(vector<int> h1, vector<int> h2, vector<int> h3) {
int height1 = accumulate(h1.begin(), h1.end(), 0);
    int height2 = accumulate(h2.begin(), h2.end(), 0);
    int height3 = accumulate(h3.begin(), h3.end(), 0);
    int i1 = 0, i2 = 0, i3 = 0;
    while (true) {
        if (i1 == h1.size() || i2 == h2.size() || i3 == h3.size()) {
            return 0;
        }
        if (height1 == height2 && height2 == height3) {
            return height1;
        }
```

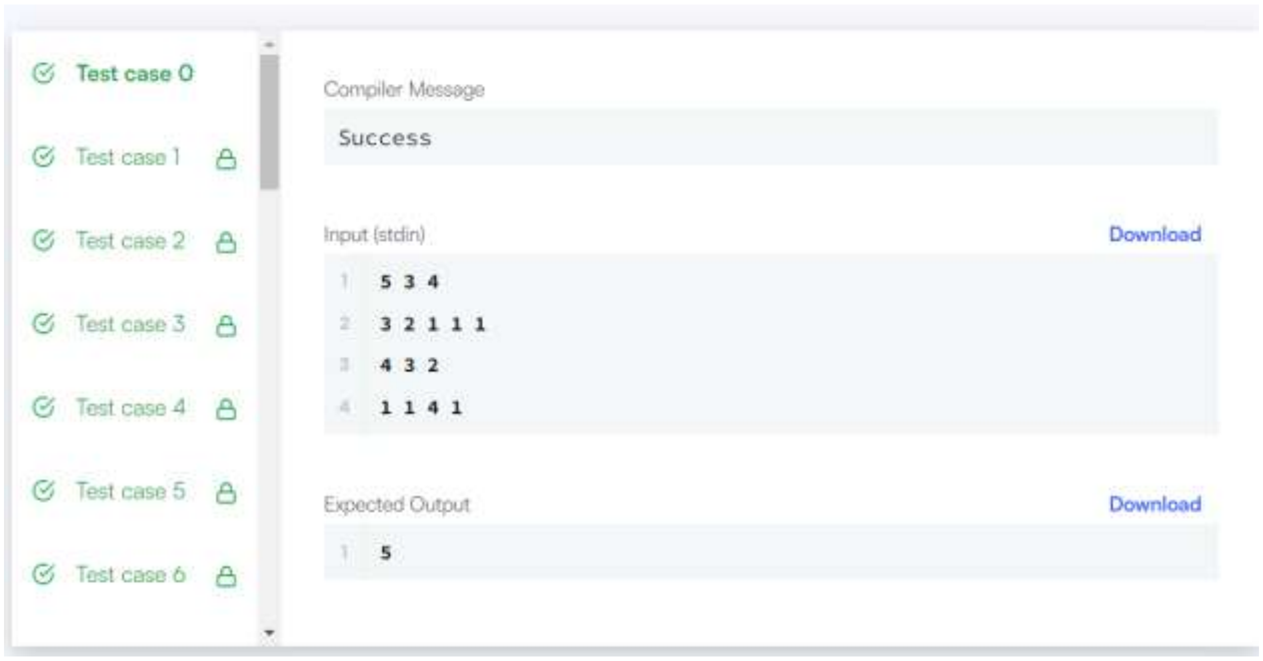


```
            if (height1 >= height2 && height1 >= height3) {
                height1 -= h1[i1++];
            } else if (height2 >= height1 && height2 >= height3) {
                height2 -= h2[i2++];
            } else if (height3 >= height1 && height3 >= height2) {
                height3 -= h3[i3++];
            }
        }

}

int main()
{
    ofstream fout(getenv("OUTPUT_PATH"));

    string first_multiple_input_temp;
    getline(cin, first_multiple_input_temp);

    vector<string> first_multiple_input = split(rtrim(first_multiple_input_temp));

    int n1 = stoi(first_multiple_input[0]);

    int n2 = stoi(first_multiple_input[1]);

    int n3 = stoi(first_multiple_input[2]);
```



```
vector<string> split(const string &str) {
    vector<string> tokens;

    string::size_type start = 0;
    string::size_type end = 0;

    while ((end = str.find(" ", start)) != string::npos) {
        tokens.push_back(str.substr(start, end - start));

        start = end + 1;
    }

    tokens.push_back(str.substr(start));
    return tokens;
}
```

4. Output :-



Test case 0
Test case 1
Test case 2
Test case 3
Test case 4
Test case 5
Test case 6

Compiler Message

```
Success
```

Input (stdin)

```
5 3 4
3 2 1 1 1
4 3 2
1 1 4 1
```

Expected Output

```
5
```

5.Time Complexity:- O(n)

PROBLEM 2

1. **Aim:** A bracket is considered to be any one of the following characters: (,), {, }, [, or].

Two brackets are considered to be a *matched pair* if the an opening bracket (i.e., (, [, or {) occurs to the left of a closing bracket (i.e.,),], or }) *of the exact same type*. There are three types of matched pairs of brackets: [], {}, and ().

A matching pair of brackets is *not balanced* if the set of brackets it encloses are not matched. For example, {[(])} is not balanced because the contents in between { and } are not balanced. The pair of square brackets encloses a single, unbalanced opening bracket, (, and the pair of parentheses encloses a single, unbalanced closing square bracket,].

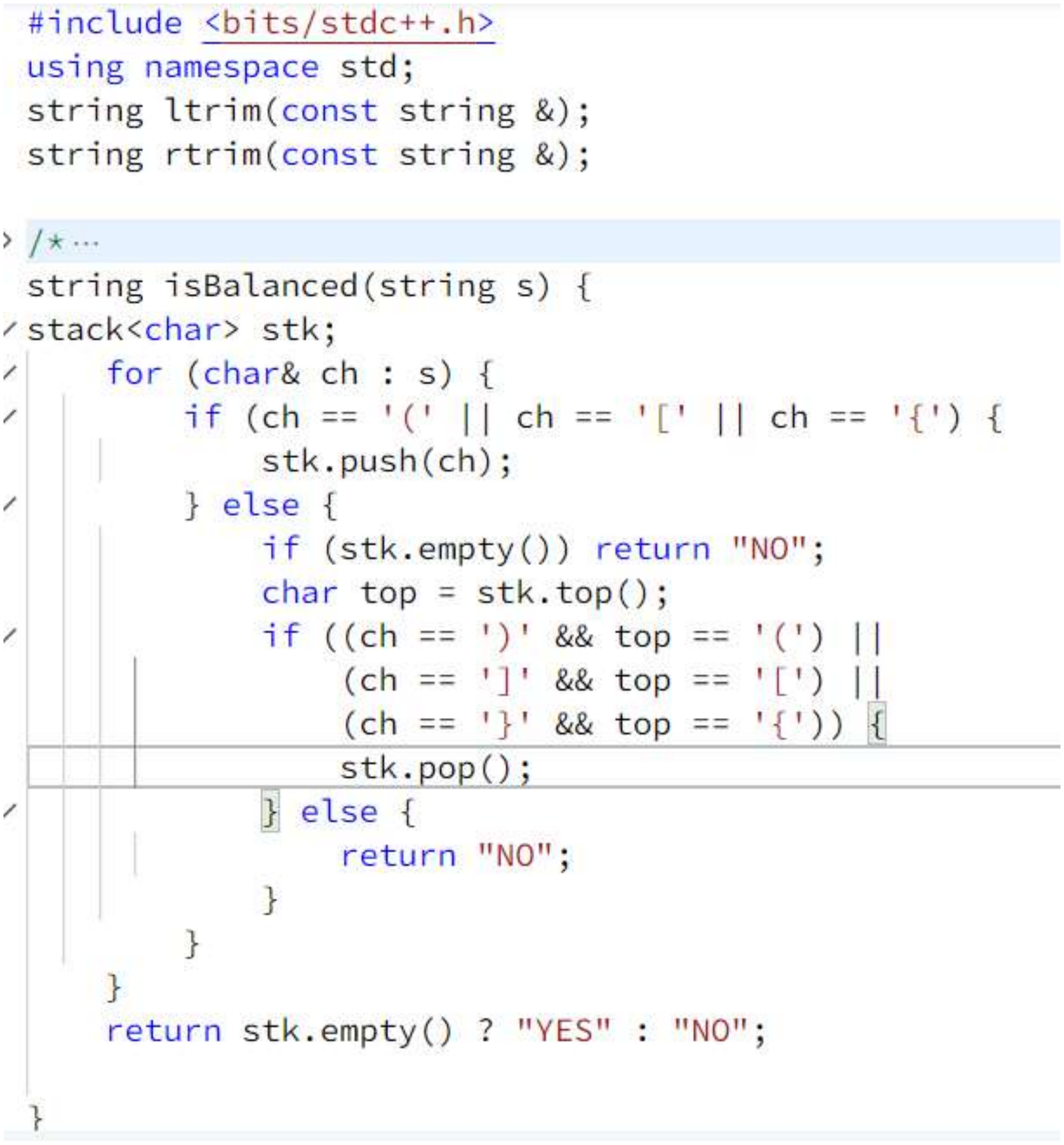
By this logic, we say a sequence of brackets is *balanced* if the following conditions are met:

- It contains no unmatched brackets.
- The subset of brackets enclosed within the confines of a matched pair of brackets is also amatched pair of brackets.

Given n strings of brackets, determine whether each sequence of brackets is balanced. If a string is balanced, return YES. Otherwise, return NO.

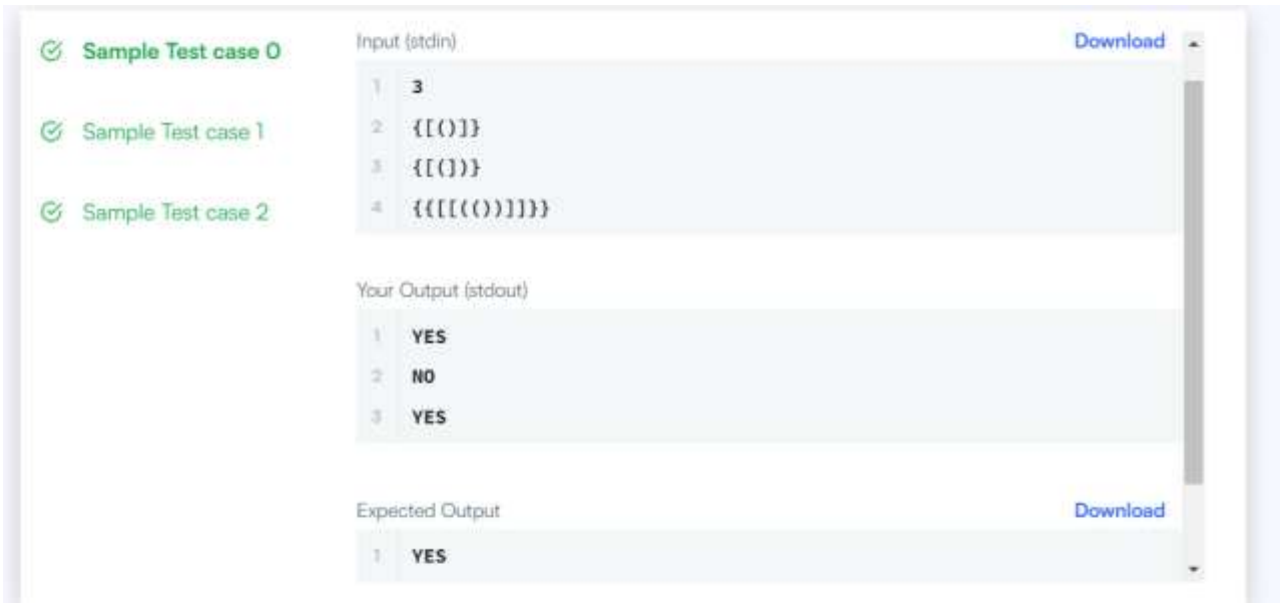
2.Objective: Determine whether each of the given sequences of brackets is balanced by checking if all opening and closing brackets match properly and if the enclosed subsets are also balanced.

3.Implementation/Code:



```
#include <bits/stdc++.h>
using namespace std;
string ltrim(const string &);
string rtrim(const string &);

/*...
string isBalanced(string s) {
stack<char> stk;
    for (char& ch : s) {
        if (ch == '(' || ch == '[' || ch == '{') {
            stk.push(ch);
        } else {
            if (stk.empty()) return "NO";
            char top = stk.top();
            if ((ch == ')' && top == '(') ||
                (ch == ']' && top == '[') ||
                (ch == '}' && top == '{')) {
                stk.pop();
            } else {
                return "NO";
            }
        }
    }
    return stk.empty() ? "YES" : "NO";

}
```

4.Output :-



Input (stdin)

```
3
{[()]}
{[(])}
{{[[(())]]}}
```

Your Output (stdout)

```
YES
NO
YES
```

Expected Output

```
YES
```

5.Time Complexity:- O(n)