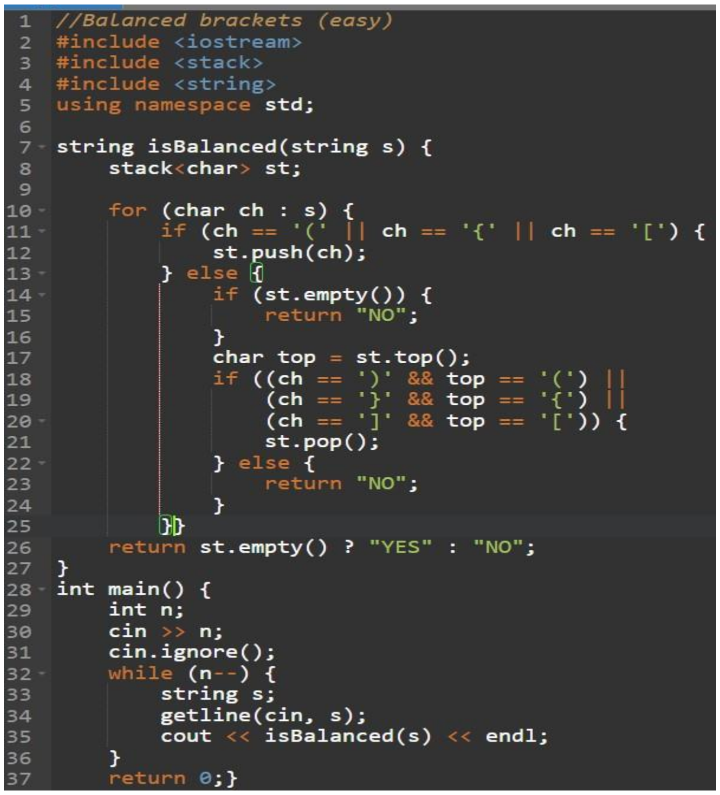
WINTER DOMAIN CAMP

NAME: Vikram Kumar **UID:22BCS12220**

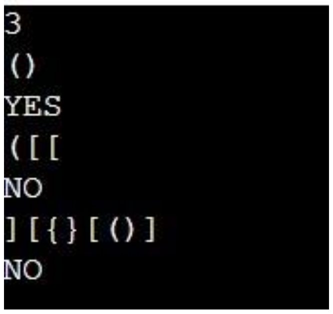
SECTION: KPIT-901-B **DATE: 24/12/2024**

Q1. Balanced brackets(Easy) Sol.



```cpp
//Balanced brackets (easy)
#include <iostream>
#include <stack>
#include <string>
using namespace std;

string isBalanced(string s) {
    stack<char> st;

    for (char ch : s) {
        if (ch == '(' || ch == '{' || ch == '[') {
            st.push(ch);
        } else {
            if (st.empty()) {
                return "NO";
            }
            char top = st.top();
            if ((ch == ')' && top == '(') ||
                (ch == '}' && top == '{') ||
                (ch == ']' && top == '[')) {
                st.pop();
            } else {
                return "NO";
            }
        }}
    return st.empty() ? "YES" : "NO";
}
int main() {
    int n;
    cin >> n;
    cin.ignore();
    while (n--) {
        string s;
        getline(cin, s);
        cout << isBalanced(s) << endl;
    }
    return 0;}
```

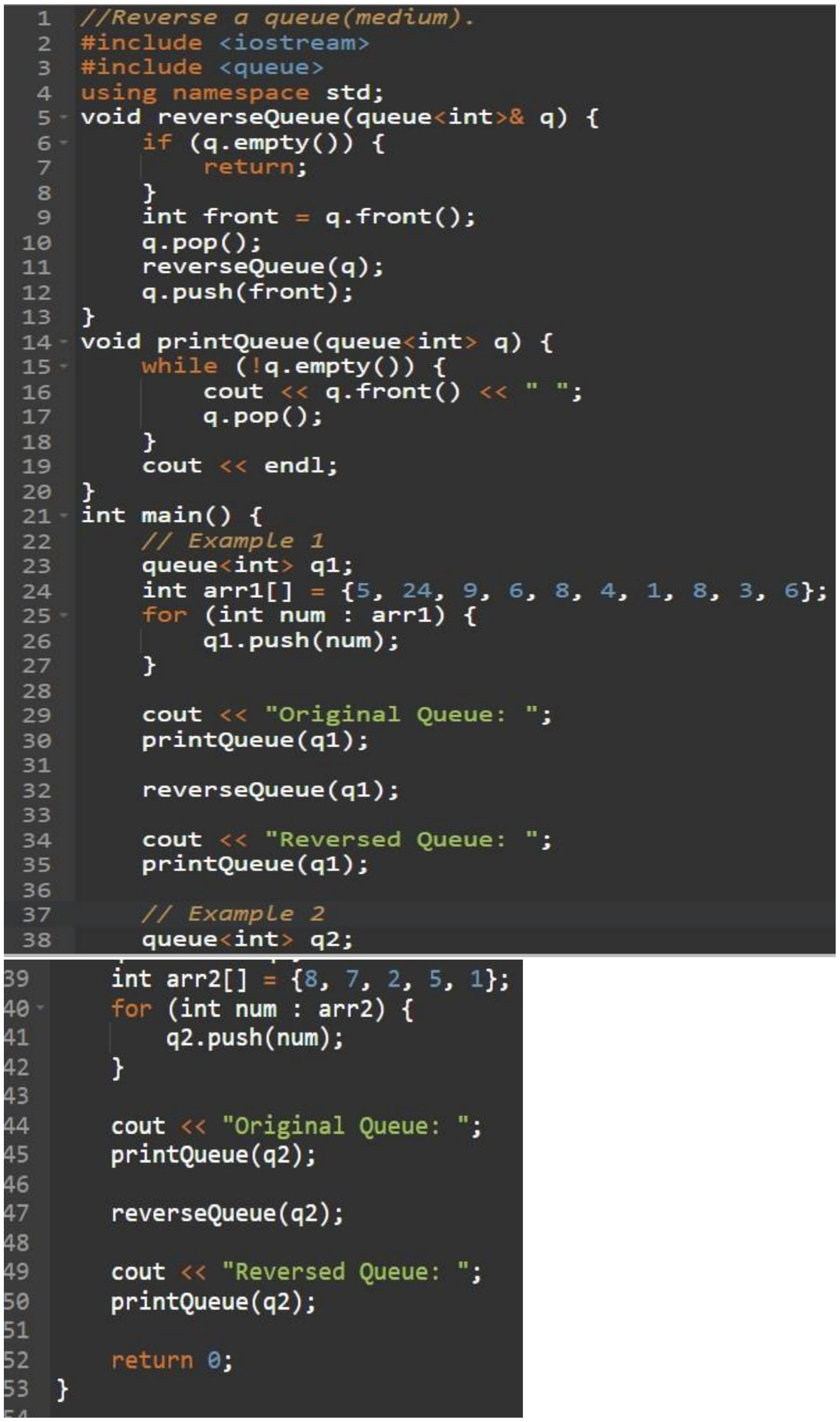
OUTPUT:



```
3
()
YES
([[
NO
][{}[()]
NO
```

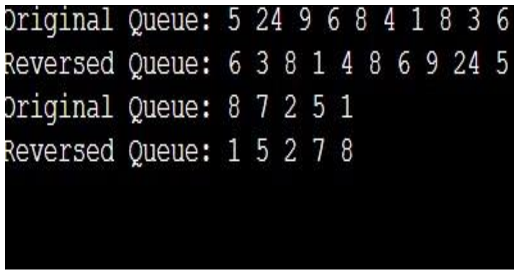
Q2. Reverse a queue(medium)

Sol.



```cpp
//Reverse a queue(medium).
#include <iostream>
#include <queue>
using namespace std;
void reverseQueue(queue<int>& q) {
    if (q.empty()) {
        return;
    }
    int front = q.front();
    q.pop();
    reverseQueue(q);
    q.push(front);
}
void printQueue(queue<int> q) {
    while (!q.empty()) {
        cout << q.front() << " ";
        q.pop();
    }
    cout << endl;
}
int main() {
    // Example 1
    queue<int> q1;
    int arr1[] = {5, 24, 9, 6, 8, 4, 1, 8, 3, 6};
    for (int num : arr1) {
        q1.push(num);
    }

    cout << "Original Queue: ";
    printQueue(q1);

    reverseQueue(q1);

    cout << "Reversed Queue: ";
    printQueue(q1);

    // Example 2
    queue<int> q2;
    int arr2[] = {8, 7, 2, 5, 1};
    for (int num : arr2) {
        q2.push(num);
    }

    cout << "Original Queue: ";
    printQueue(q2);

    reverseQueue(q2);

    cout << "Reversed Queue: ";
    printQueue(q2);

    return 0;
}
```

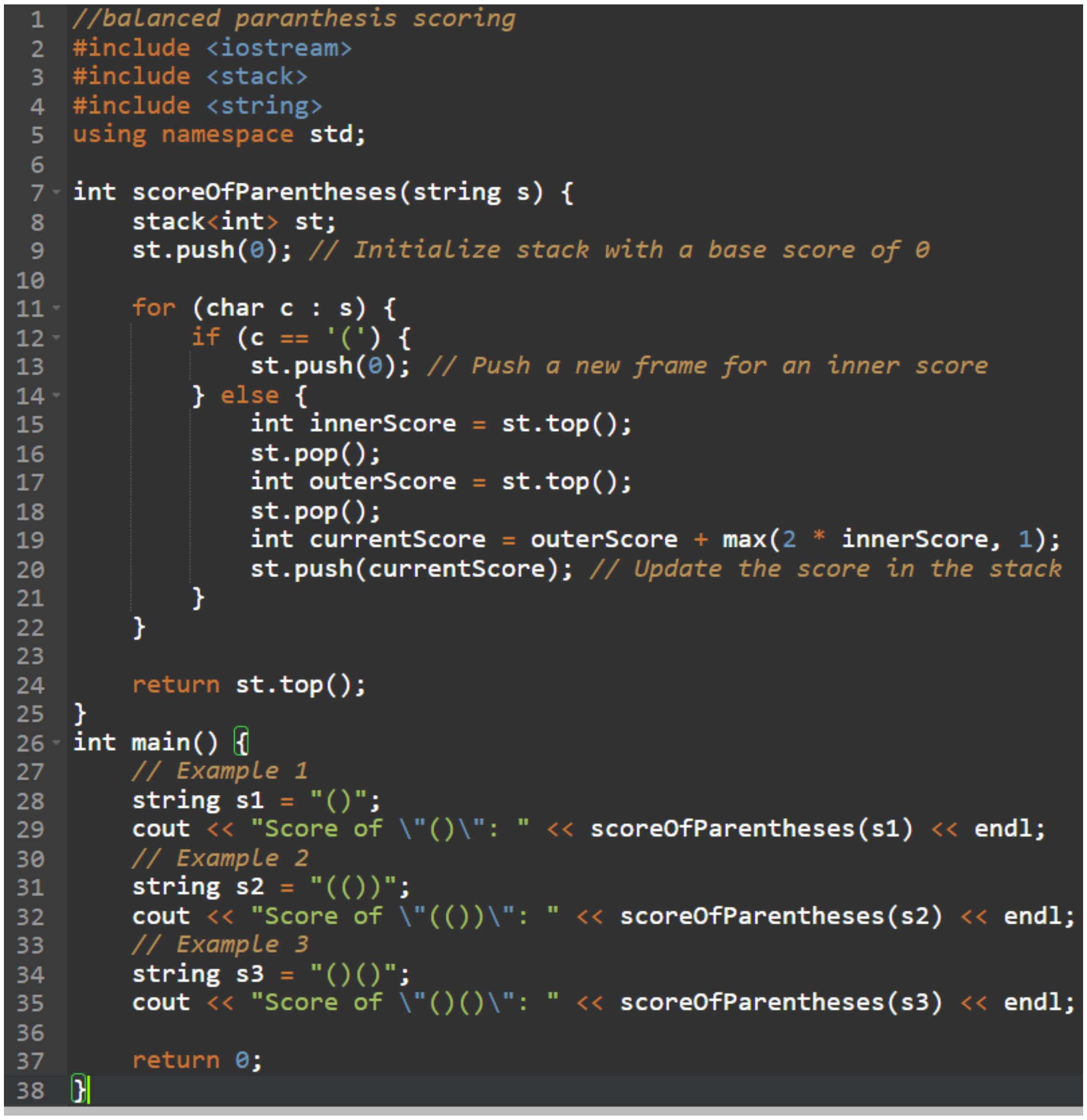
OUTPUT:



```
Original Queue: 5 24 9 6 8 4 1 8 3 6
Reversed Queue: 6 3 8 1 4 8 6 9 24 5
Original Queue: 8 7 2 5 1
Reversed Queue: 1 5 2 7 8
```

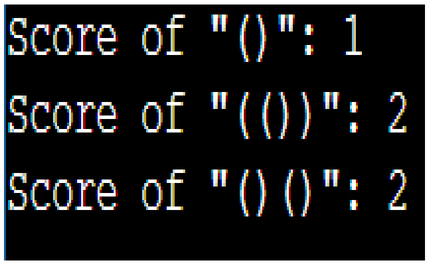
Q3. Balanced parenthesis scoring (medium)

Sol.



```cpp
//balanced paranthesis scoring
#include <iostream>
#include <stack>
#include <string>
using namespace std;

int scoreOfParentheses(string s) {
    stack<int> st;
    st.push(0); // Initialize stack with a base score of 0

    for (char c : s) {
        if (c == '(') {
            st.push(0); // Push a new frame for an inner score
        } else {
            int innerScore = st.top();
            st.pop();
            int outerScore = st.top();
            st.pop();
            int currentScore = outerScore + max(2 * innerScore, 1);
            st.push(currentScore); // Update the score in the stack
        }
    }

    return st.top();
}
int main() {
    // Example 1
    string s1 = "()";
    cout << "Score of \"()\": " << scoreOfParentheses(s1) << endl;
    // Example 2
    string s2 = "(())";
    cout << "Score of \"(())\": " << scoreOfParentheses(s2) << endl;
    // Example 3
    string s3 = "()()";
    cout << "Score of \"()()\": " << scoreOfParentheses(s3) << endl;

    return 0;
}
```

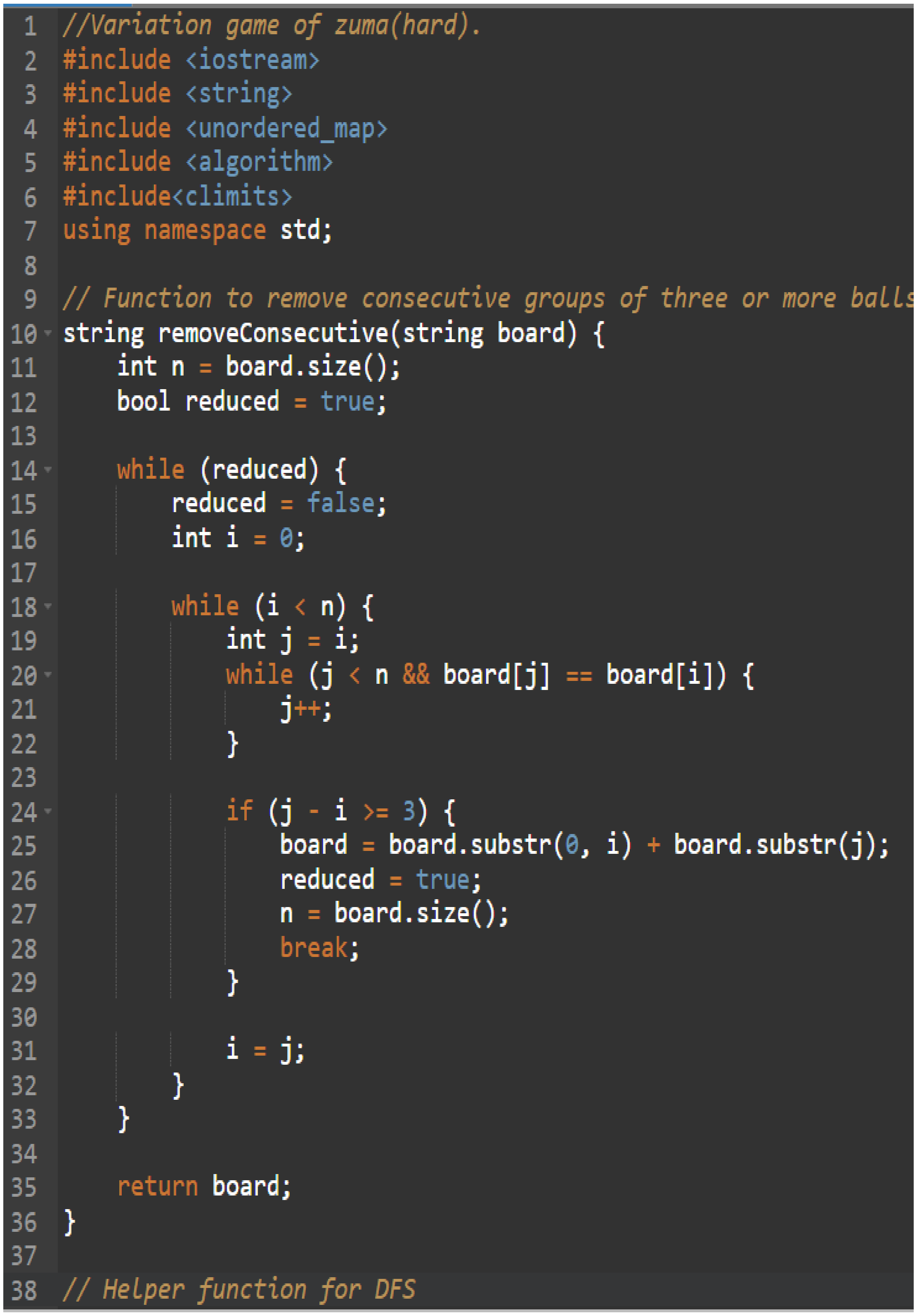
OUTPUT:



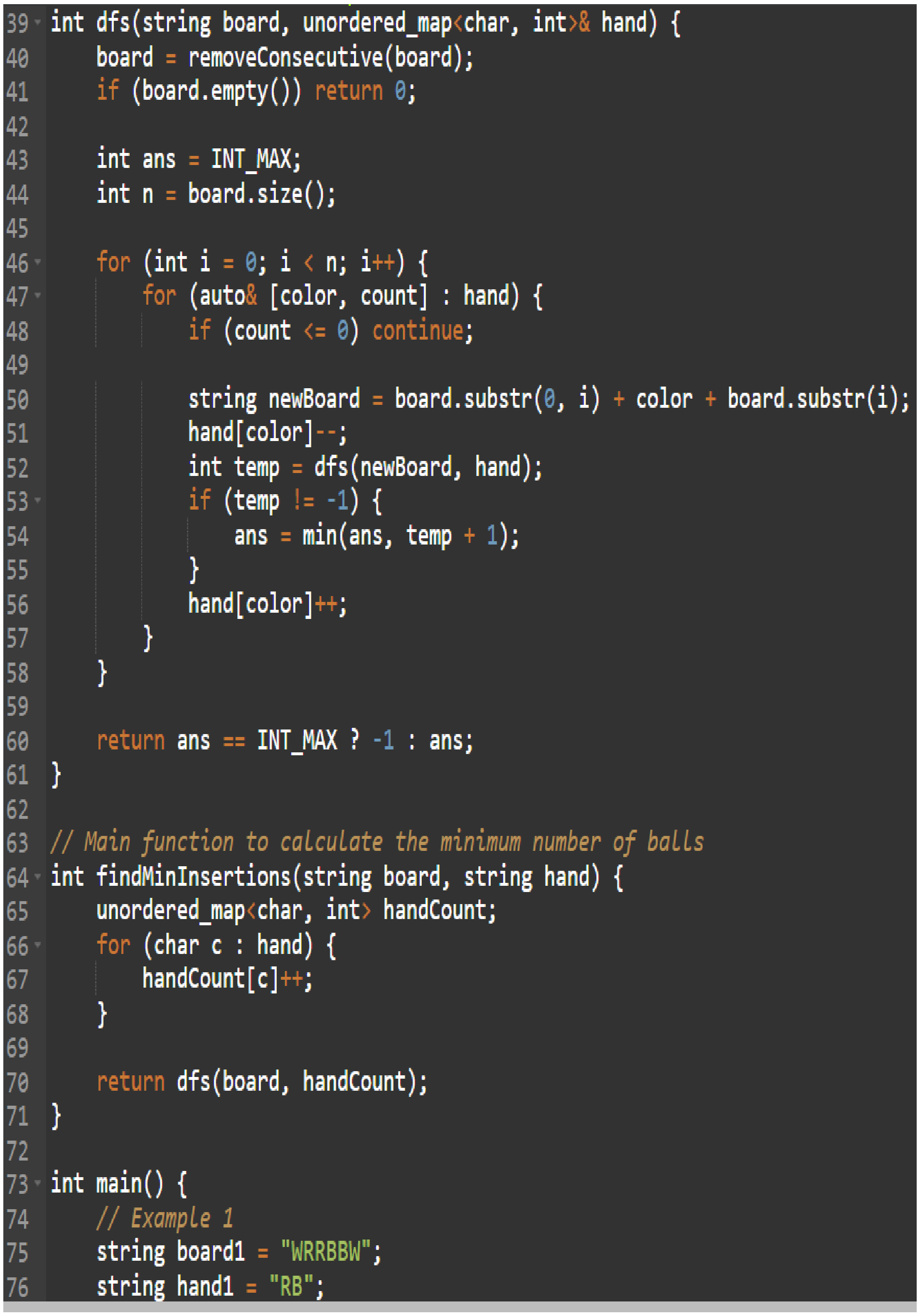
```
Score of "()": 1
Score of "(())": 2
Score of "()()": 2
```

Q4.Variation game of zuma(hard)

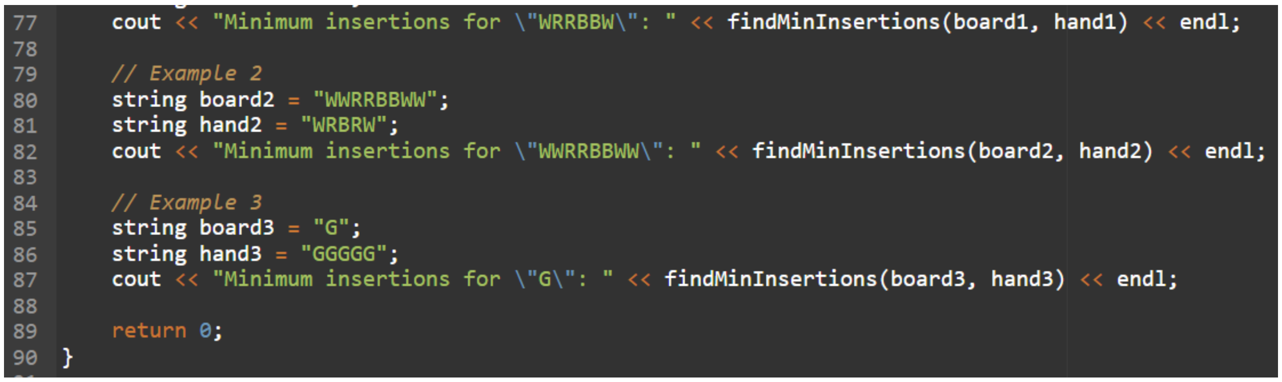
Sol.



```cpp
//Variation game of zuma(hard).
#include <iostream>
#include <string>
#include <unordered_map>
#include <algorithm>
#include<climits>
using namespace std;

// Function to remove consecutive groups of three or more balls
string removeConsecutive(string board) {
    int n = board.size();
    bool reduced = true;

    while (reduced) {
        reduced = false;
        int i = 0;

        while (i < n) {
            int j = i;
            while (j < n && board[j] == board[i]) {
                j++;
            }

            if (j - i >= 3) {
                board = board.substr(0, i) + board.substr(j);
                reduced = true;
                n = board.size();
                break;
            }

            i = j;
        }
    }

    return board;
}

// Helper function for DFS
```

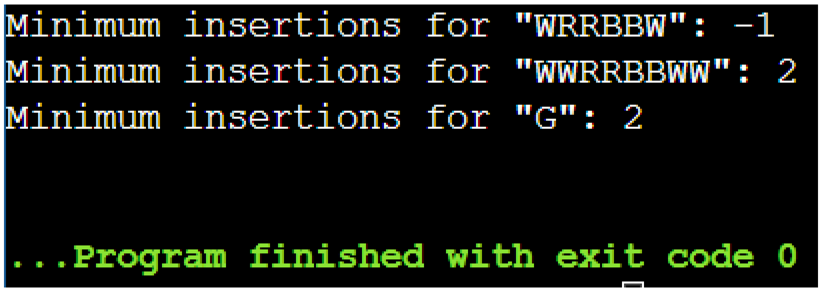


```cpp
int dfs(string board, unordered_map<char, int>& hand) {
    board = removeConsecutive(board);
    if (board.empty()) return 0;

    int ans = INT_MAX;
    int n = board.size();

    for (int i = 0; i < n; i++) {
        for (auto& [color, count] : hand) {
            if (count <= 0) continue;

            string newBoard = board.substr(0, i) + color + board.substr(i);
            hand[color]--;
            int temp = dfs(newBoard, hand);
            if (temp != -1) {
                ans = min(ans, temp + 1);
            }
            hand[color]++;
        }
    }

    return ans == INT_MAX ? -1 : ans;
}

// Main function to calculate the minimum number of balls
int findMinInsertions(string board, string hand) {
    unordered_map<char, int> handCount;
    for (char c : hand) {
        handCount[c]++;
    }

    return dfs(board, handCount);
}

int main() {
    // Example 1
    string board1 = "WRRBBW";
    string hand1 = "RB";
```



```cpp
    cout << "Minimum insertions for \"WRRBBW\": " << findMinInsertions(board1, hand1) << endl;

    // Example 2
    string board2 = "WWRRBBWW";
    string hand2 = "WRBRW";
    cout << "Minimum insertions for \"WWRRBBWW\": " << findMinInsertions(board2, hand2) << endl;

    // Example 3
    string board3 = "G";
    string hand3 = "GGGGG";
    cout << "Minimum insertions for \"G\": " << findMinInsertions(board3, hand3) << endl;

    return 0;
}
```

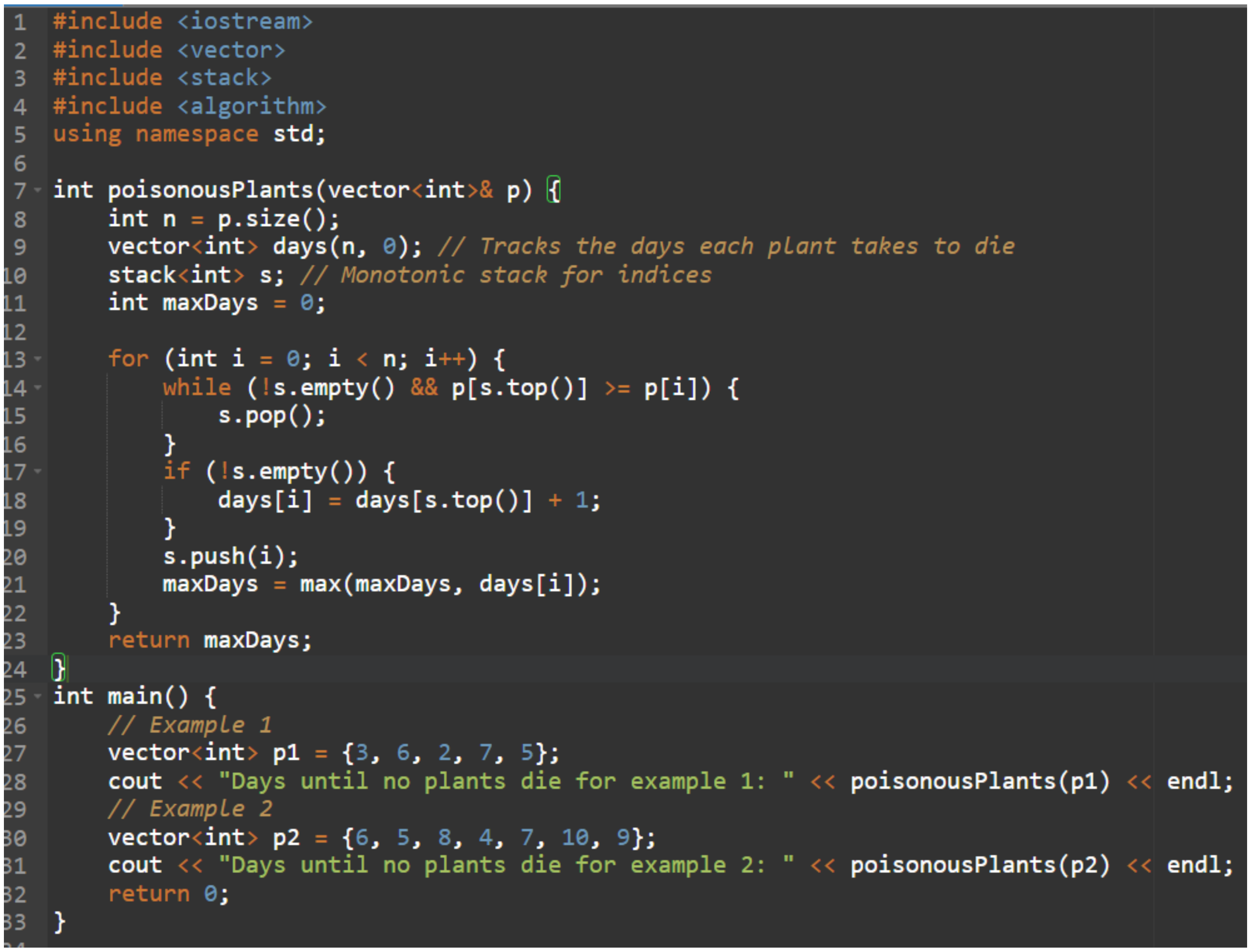
Output:



```
Minimum insertions for "WRRBBW": -1
Minimum insertions for "WWRRBBWW": 2
Minimum insertions for "G": 2


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

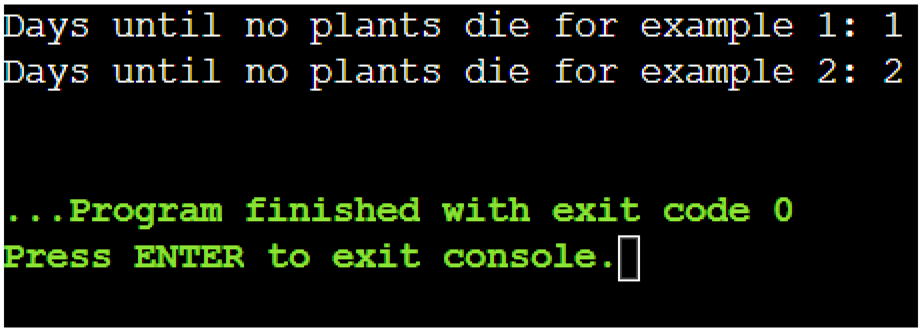
Q5.Poisonous plant.(very hard)

Sol.



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

int poisonousPlants(vector<int>& p) {
    int n = p.size();
    vector<int> days(n, 0); // Tracks the days each plant takes to die
    stack<int> s; // Monotonic stack for indices
    int maxDays = 0;

    for (int i = 0; i < n; i++) {
        while (!s.empty() && p[s.top()] >= p[i]) {
            s.pop();
        }
        if (!s.empty()) {
            days[i] = days[s.top()] + 1;
        }
        s.push(i);
        maxDays = max(maxDays, days[i]);
    }
    return maxDays;
}
int main() {
    // Example 1
    vector<int> p1 = {3, 6, 2, 7, 5};
    cout << "Days until no plants die for example 1: " << poisonousPlants(p1) << endl;
    // Example 2
    vector<int> p2 = {6, 5, 8, 4, 7, 10, 9};
    cout << "Days until no plants die for example 2: " << poisonousPlants(p2) << endl;
    return 0;
}
```

OUTPUT:



```
Days until no plants die for example 1: 1
Days until no plants die for example 2: 2


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