

**Name:** Priya Chauhan

**UID:**22BCS15310

**Section:** 22BCS\_IOT\_620-A

**Date:** 24-12-24

## **DOMAIN WINTER WINNING CAMP-Day(4)**

### **1) MinStak**

#### **Code:**

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

class MinStack {
    stack<int> s,
    minStack;

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

            minStack.push(val);
        }
    }

    void pop() {
        if (s.top() ==
```

```
minStack.top()) {  
    minStack.pop();  
}  
s.pop();  
}
```

```
int top() {  
    return s.top();  
}
```

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

```
int main() {  
    MinStack minStack;  
    int n, operation, val;  
    cout << "Enter  
    number of  
    operations: ";  
    cin >> n;  
    for (int i = 0; i < n;  
        ++i) {  
        cout << "Choose  
        operation (1: push,  
        2: pop, 3: top, 4:
```

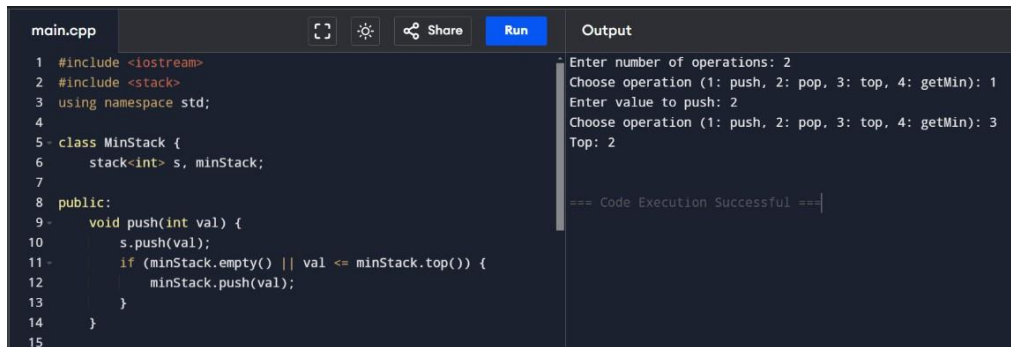
```

getMin): ";
    cin >> operation;
    if (operation == 1)
    {
        cout << "Enter
value to push: ";
        cin >> val;

        minStack.push(val);
    } else if (operation
== 2) {
        minStack.pop();
    } else if (operation
== 3) {
        cout << "Top: "
<< minStack.top()
<< endl;
    } else if (operation
== 4) {
        cout <<
"Minimum: " <<
minStack.getMin()
<< endl;
    }
}
return 0;
}

```

## Output:



The screenshot shows a C++ IDE with a file named `main.cpp`. The code defines a `MinStack` class with a `stack<int>` and a `minStack` variable. It includes a `push` method that pushes a value onto the stack and updates `minStack` if the new value is less than or equal to the current minimum. The output window shows the following interaction:

```
Enter number of operations: 2
Choose operation (1: push, 2: pop, 3: top, 4: getMin): 1
Enter value to push: 2
Choose operation (1: push, 2: pop, 3: top, 4: getMin): 3
Top: 2

=== Code Execution Successful ===
```

## 2) Balanced Brackets

```
#include <iostream>
#include <stack>
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;
    cout << "Enter number of bracket strings to check: ";
    cin >> n;
    for (int i = 0; i < n; ++i) {
```

```

    string s;
    cout << "Enter string: ";
    cin >> s;
    cout << (isBalanced(s) ? "YES" : "NO") << endl;
}
return 0;
}

```

## Output:

The screenshot shows a C++ IDE with a file named `main.cpp`. The code defines a function `isBalanced` that uses a stack to check if a string of brackets is balanced. The output window shows the program's execution: it prompts for the number of strings to check (1), then for a string (at), and outputs "NO". A success message "=== Code Execution Successful ===" is also displayed.

```

main.cpp
1 #include <iostream>
2 #include <stack>
3 using namespace std;
4
5 bool isBalanced(string s) {
6     stack<char> st;
7     for (char c : s) {
8         if (c == '(' || c == '{' || c == '[') {
9             st.push(c);
10        } else {
11            if (st.empty()) return false;
12            if ((c == ')' && st.top() != '(') ||
13                (c == '}' && st.top() != '{') ||
14                (c == ']' && st.top() != '[')) {
15                return false;
16            }
17        }
18    }
19    return true;
20 }

```

Output

```

Enter number of bracket strings to check: 1
Enter string: at
NO

=== Code Execution Successful ===

```

### 3) Evaluate

#### Reverse

#### Polish

#### Notation

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

int
    evalRPN(vector<stri
n g>& 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 if (token ==
"/") st.push(a / b);
    } else {
```

```

        st.push(stoi(token));
    }
}
return st.top();
}

```

```

int main() {
    int n;
    cout << "Enter number
        of tokens: ";
    cin >> n;
    vector<string>
    tokens(n);
    cout << "Enter the
        tokens: ";
    for (int i = 0; i < n; ++i) {
        cin >> tokens[i];
    }
    cout << "Result: "
        <<
        evalRPN(tokens)
        << endl;
    return 0;
}

```

## Output:

main.cpp	Output
<pre>1 #include &lt;iostream&gt; 2 #include &lt;stack&gt; 3 #include &lt;vector&gt; 4 using namespace std; 5 6 int evalRPN(vector&lt;string&gt;&amp; tokens) { 7     stack&lt;int&gt; st; 8     for (string&amp; token : tokens) { 9         if (token == "+"    token == "-"    token == "*"                token == "/") { 10             int b = st.top(); st.pop(); 11             int a = st.top(); st.pop(); 12             if (token == "+") st.push(a + b); 13             else if (token == "-") st.push(a - b); 14             else if (token == "*") st.push(a * b); 15             else if (token == "/") st.push(a / b);         }     } }</pre>	<pre>Enter number of tokens: 5 Enter the tokens: 2 1 + 3 * Result: 9  === Code Execution Successful ===</pre>

## 4) Longest Valid Parentheses

```
#include <iostream>
```

```
#include <stack>
```

```
using namespace std;
```

```
int longestValidParentheses(string s) {
```

```
    stack<int> st;
```

```
    st.push(-1);
```

```
    int maxLength = 0;
```

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

```
    { if (s[i] == '(') {
```

```
        st.push(i);
```

```
    } else {
```

```
        st.pop();
```

```
        if (st.empty()) {
```

```
            st.push(i);
```



```

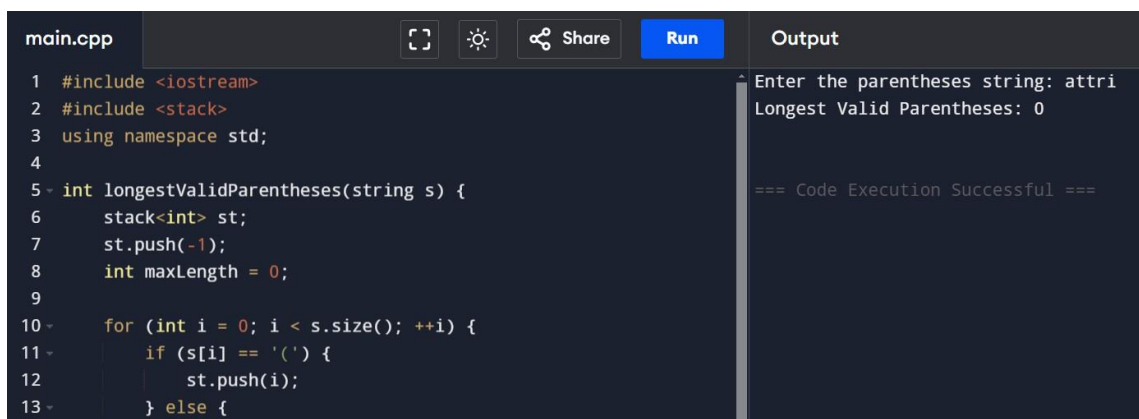
        } else {
            maxLength = max(maxLength, i - st.top());
        }
    }
}

return maxLength;
}

int main() {
    string s;
    cout << "Enter the parentheses string: ";
    cin >> s;
    cout << "Longest Valid Parentheses: " << longestValidParentheses(s) << endl;
    return 0;
}

```

### Output:



The screenshot shows a C++ IDE with a file named `main.cpp`. The code defines a function `longestValidParentheses` that uses a stack to find the longest valid parentheses substring. The `main` function prompts the user to enter a parentheses string. The output shows the user input "attri" and the program's response "Longest Valid Parentheses: 0". A status message at the bottom indicates "=== Code Execution Successful ===".

```

main.cpp
1  #include <iostream>
2  #include <stack>
3  using namespace std;
4
5  int longestValidParentheses(string s) {
6      stack<int> st;
7      st.push(-1);
8      int maxLength = 0;
9
10     for (int i = 0; i < s.size(); ++i) {
11         if (s[i] == '(') {
12             st.push(i);
13         } else {

```

Enter the parentheses string: attri  
Longest Valid Parentheses: 0

=== Code Execution Successful ===

## 5) Poisonous Plants

```

#include <iostream>
#include <vector>
using namespace std;

int
poisonousPlants(vector
<int>& p) {
    vector<int>
days(p.size(), 0);
    vector<int> stack;
    int maxDays = 0;

    for (int i = 0; i <
p.size(); ++i) {
        int day = 0;
        while
(!stack.empty() &&
p[stack.back()] >= p[i]) {
            day = max(day,
days[stack.back()]);

stack.pop_back();
        }
        if (!stack.empty()) {
            days[i] = day + 1;
        }
        stack.push_back(i);
        maxDays =
max(maxDays, days[i]);
    }
}

```

```
    }  
    return maxDays;  
}
```

```
int main() {  
    int n;  
    cout << "Enter  
number of plants: ";  
    cin >> n;  
    vector<int> p(n);  
    cout << "Enter  
pesticide levels: ";  
    for (int i = 0; i <  
        n;  
        ++i) cin >> p[i];  
    cout << "Days until  
no plants die: " <<  
    poisonousPlants(p) <<  
    endl;  
    return 0;  
}
```

## Output:

main.cpp	Output
<pre>1 #include &lt;iostream&gt; 2 #include &lt;vector&gt; 3 using namespace std; 4 5 int poisonousPlants(vector&lt;int&gt;&amp; p) { 6     vector&lt;int&gt; days(p.size(), 0); 7     vector&lt;int&gt; stack; 8     int maxDays = 0; 9 10    for (int i = 0; i &lt; p.size(); ++i) { 11        int day = 0; 12        while (!stack.empty() &amp;&amp; p[stack.back()] &gt;= p[i]) { 13            day = max(day, days[stack.back()]); 14            stack.pop_back(); 15        }</pre>	<pre>Enter number of plants: 7 Enter pesticide levels: 6 5 8 4 7 10 9 Days until no plants die: 2  === Code Execution Successful ===</pre>

## 6) Implement Queue Using Stacks

```
#include <iostream>
```

```
#include <stack>
```

```
using namespace std;
```

```
class MyQueue {
```

```
    stack<int> inStack, outStack;
```

```
    void transfer() {
```

```
        while (!inStack.empty()) {
```

```
            outStack.push(inStack.top());
```

```
            inStack.pop();
```

```
        }
```

```
    }
```

```
public:
```

```
    void push(int x) {
```

```
        inStack.push(x);
```

```
    }
```

```
    int pop() {
```

```
        if (outStack.empty()) transfer();
```

```
        int val = outStack.top();
```

```
        outStack.pop();
```

```
        return val;
```

```
    }
```

```
    int peek() {
```

```

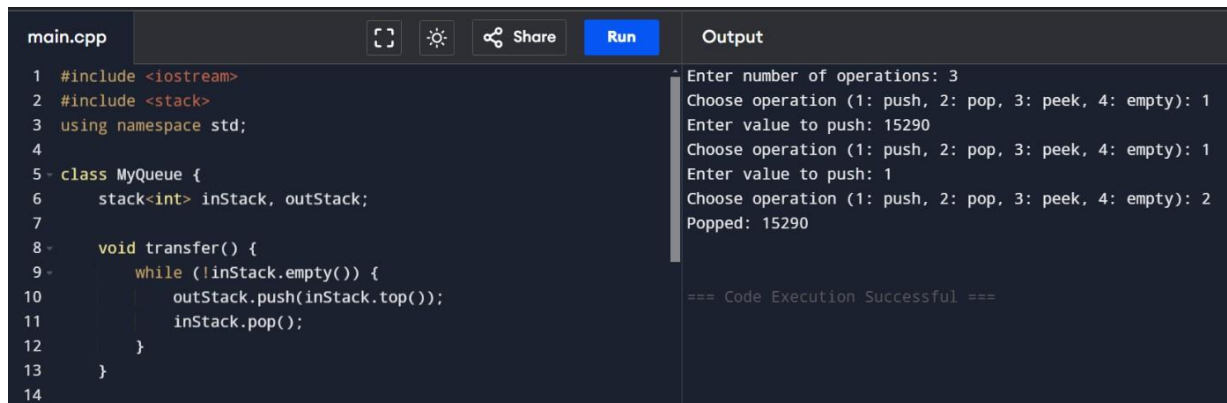
        if (outStack.empty()) transfer();
        return outStack.top();
    }

    bool empty() {
        return inStack.empty() && outStack.empty();
    }
};

int main() {
    MyQueue q;
    int n, op, x;
    cout << "Enter number of operations: ";
    cin >> n;
    for (int i = 0; i < n; ++i) {
        cout << "Choose operation (1: push, 2: pop, 3: peek, 4: empty): ";
        cin >> op;
        if (op == 1) {
            cout << "Enter value to push: ";
            cin >> x;
            q.push(x);
        } else if (op == 2) {
            cout << "Popped: " << q.pop() << endl;
        } else if (op == 3) {
            cout << "Front: " << q.peek() << endl;
        } else if (op == 4) {
            cout << "Empty: " << (q.empty() ? "Yes" : "No") << endl;
        }
    }
    return 0;
}

```

## Output :

A screenshot of a C++ IDE interface. The left pane shows a file named 'main.cpp' with the following code:

```
1 #include <iostream>
2 #include <stack>
3 using namespace std;
4
5 class MyQueue {
6     stack<int> inStack, outStack;
7
8     void transfer() {
9         while (!inStack.empty()) {
10             outStack.push(inStack.top());
11             inStack.pop();
12         }
13     }
14 }
```

The right pane shows the 'Output' window with the following text:

```
Enter number of operations: 3
Choose operation (1: push, 2: pop, 3: peek, 4: empty): 1
Enter value to push: 15290
Choose operation (1: push, 2: pop, 3: peek, 4: empty): 1
Enter value to push: 1
Choose operation (1: push, 2: pop, 3: peek, 4: empty): 2
Popped: 15290

=== Code Execution Successful ===
```

## 7) Reverse a Queue Using Recursion

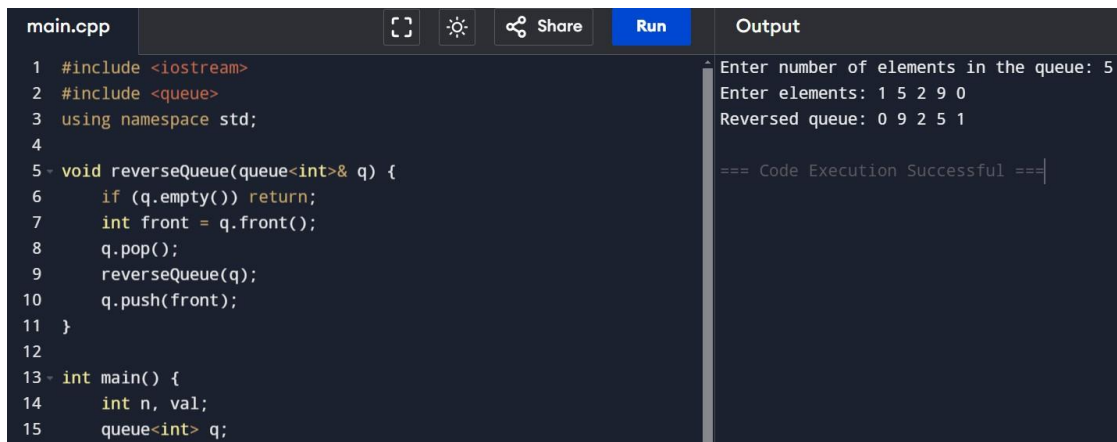
```
#include <iostream>
#include <queue>
using namespace std;
```

```
void reverseQueue(queue<int>& q) {
    if (q.empty()) return;
    int front = q.front();
    q.pop();
    reverseQueue(q);
    q.push(front);
}
```

```
int main() {
    int n, val;
    queue<int> q;
    cout << "Enter number of elements in the queue: ";
    cin >> n;
    cout << "Enter elements: ";
    for (int i = 0; i < n; ++i) {
        cin >> val;
        q.push(val);
    }
    reverseQueue(q);
    cout << "Reversed queue: ";
    while (!q.empty()) {
        cout << q.front() << " ";
        q.pop();
    }
    return 0;
}
```

```
}
```

**Output :**



```
main.cpp  [ ]  [ ]  [ ]  Share  Run  Output
1  #include <iostream>
2  #include <queue>
3  using namespace std;
4
5  void reverseQueue(queue<int>& q) {
6      if (q.empty()) return;
7      int front = q.front();
8      q.pop();
9      reverseQueue(q);
10     q.push(front);
11 }
12
13 int main() {
14     int n, val;
15     queue<int> q;
```

Enter number of elements in the queue: 5  
Enter elements: 1 5 2 9 0  
Reversed queue: 0 9 2 5 1  
=== Code Execution Successful ===

## 8) Sliding Window Maximum

```
#include <iostream>
#include <vector>
#include <deque>
using namespace std;
```

```
vector<int> maxSlidingWindow(vector<int>& nums, int k) {
    deque<int> dq;
    vector<int> result;

    for (int i = 0; i < nums.size(); ++i) {
        if (!dq.empty() && dq.front() == i - k) dq.pop_front();
        while (!dq.empty() && nums[dq.back()] < nums[i]) dq.pop_back();
        dq.push_back(i);
        if (i >= k - 1) result.push_back(nums[dq.front()]);
    }
    return result;
}
```

```
int main() {
    int n, k;
    cout << "Enter number of elements: ";
    cin >> n;
    vector<int> nums(n);
    cout << "Enter elements: ";
    for (int i = 0; i < n; ++i) cin >> nums[i];
    cout << "Enter window size: ";
    cin >> k;
    vector<int> result = maxSlidingWindow(nums, k);
    cout << "Sliding window maximums: ";
```

```

for (int x : result) cout << x << " ";
return 0;
}

```

## Output :

```

main.cpp
1 #include <iostream>
2 #include <vector>
3 #include <deque>
4 using namespace std;
5
6 vector<int> maxSlidingWindow(vector<int>& nums, int k) {
7     deque<int> dq;
8     vector<int> result;
9
10    for (int i = 0; i < nums.size(); ++i) {
11        if (!dq.empty() && dq.front() == i - k) dq.pop_front();
12        while (!dq.empty() && nums[dq.back()] < nums[i]) dq
            .pop_back();
13        dq.push_back(i);

```

Output

```

Enter number of elements: 5
Enter elements: 1 5 2 9 0
Enter window size: 4
Sliding window maximums: 9 9

=== Code Execution Successful ===

```

## 9) Circular Queue Simulation

```

#include <iostream>
#include <queue>
#include <vector>
using namespace std;

```

```

// Function to calculate the number of students unable to eat
int studentsUnableToEat(vector<int>& students, vector<int>& sandwiches) {
    queue<int> studentQueue;
    for (int s : students) {
        studentQueue.push(s);
    }

```

```

    int i = 0, count = 0;
    while (!studentQueue.empty() && count < studentQueue.size()) {
        if (studentQueue.front() == sandwiches[i]) {
            studentQueue.pop();
            ++i;
            count = 0;
        } else {
            studentQueue.push(studentQueue.front());
            studentQueue.pop();
            ++count;
        }
    }
}

```

```

return studentQueue.size();

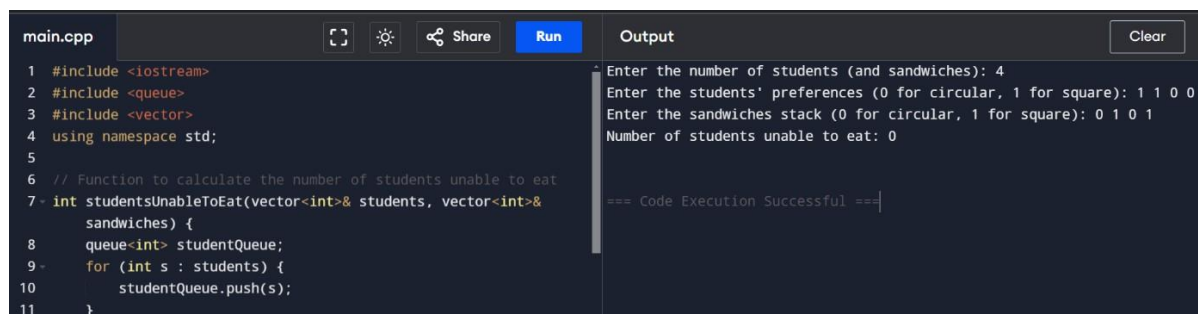
```



```
}
```

```
int main() {  
    int n;  
    cout << "Enter the number of students (and sandwiches): ";  
    cin >> n;  
  
    vector<int> students(n), sandwiches(n);  
    cout << "Enter the students' preferences (0 for circular, 1 for square): ";  
    for (int i = 0; i < n; ++i) {  
        cin >> students[i];  
    }  
  
    cout << "Enter the sandwiches stack (0 for circular, 1 for square): ";  
    for (int i = 0; i < n; ++i) {  
        cin >> sandwiches[i];  
    }  
  
    int result = studentsUnableToEat(students, sandwiches);  
    cout << "Number of students unable to eat: " << result << endl;  
  
    return 0;  
}
```

## Output :



The screenshot shows a C++ IDE with a file named `main.cpp`. The code defines a function `studentsUnableToEat` that takes two vectors of integers, `students` and `sandwiches`, and returns the number of students who cannot eat. The `main` function prompts the user to enter the number of students, their preferences (0 for circular, 1 for square), and the stack of sandwiches. The output shows the user input: 4 students, preferences 1 1 0 0, and sandwiches 0 1 0 1. The result is 0 students unable to eat. The IDE also shows a 'Run' button and a 'Clear' button in the output panel.

```
1 #include <iostream>  
2 #include <queue>  
3 #include <vector>  
4 using namespace std;  
5  
6 // Function to calculate the number of students unable to eat  
7 int studentsUnableToEat(vector<int>& students, vector<int>&  
    sandwiches) {  
8     queue<int> studentQueue;  
9     for (int s : students) {  
10         studentQueue.push(s);  
11     }
```

Output

```
Enter the number of students (and sandwiches): 4  
Enter the students' preferences (0 for circular, 1 for square): 1 1 0 0  
Enter the sandwiches stack (0 for circular, 1 for square): 0 1 0 1  
Number of students unable to eat: 0  
  
=== Code Execution Successful ===
```

## 10) Zuma Game

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

// Helper function to reduce the board by removing groups of 3 or more consecutive balls string
reduceBoard(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[i] == board[j]) {
                j++;
            }

            // If there are 3 or more consecutive balls, remove them if
            (j - i >= 3) {
                board = board.substr(0, i) + board.substr(j);
                n = board.size();
                reduced = true;
            } else
            { i = j;
            }
        }
    }
    return board;
}

// Helper function for DFS
int dfs(string board, unordered_map<char, int>& hand) {
    board = reduceBoard(board);
    if (board.empty()) return 0;

    int minSteps = INT_MAX, n = board.size();

    for (int i = 0; i < n; i++) {
        int j = i;
        while (j < n && board[i] == board[j]) {
            j++;
        }
    }
}
```

```

    }

    int need = 3 - (j - i);
    if (hand[board[i]] >= need) {
        hand[board[i]] -= need;
        int steps = dfs(board.substr(0, i) + board.substr(j), hand);
        if (steps != -1) {
            minSteps = min(minSteps, steps + need);
        }
        hand[board[i]] += need;
    }
}

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

// Main function to calculate the minimum steps to clear the board
int findMinStep(string board, string hand) {
    unordered_map<char, int> handCount;
    for (char c : hand) {
        handCount[c]++;
    }
    return dfs(board, handCount);
}

int main() {
    string board, hand;
    cout << "Enter the board string (e.g., WRRBBW): ";
    cin >> board;
    cout << "Enter the hand string (e.g., RB): ";
    cin >> hand;

    int result = findMinStep(board, hand);
    if (result == -1) {
        cout << "It is impossible to clear the board." << endl;
    } else {
        cout << "Minimum steps to clear the board: " << result << endl;
    }

    return 0;
}

```

## Output :

main.cpp	Output
<pre>1 #include &lt;iostream&gt; 2 #include &lt;unordered_map&gt; 3 #include &lt;string&gt; 4 #include &lt;vector&gt; 5 #include &lt;climits&gt; 6 using namespace std; 7 8 // Helper function to reduce the board by removing groups of 3   or more consecutive balls 9 string reduceBoard(string board) { 10     int n = board.size(); 11     bool reduced = true;</pre>	<pre>Enter the board string (e.g., WRRBBW): WRRBBW Enter the hand string (e.g., RB): RB It is impossible to clear the board.  === Code Execution Successful ===</pre>