Name: Nandita Dutta   **UID:** 22BCS15313

Section: 22BCS_IOT_620-A   **Date:** 24-12-24

DOMAIN WINTER WINNING CAMP-Day(4)

1) **MinStack**

   Code:

```cpp
#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() ==
```

```cpp
      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:
```

```cpp
        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;
}
```
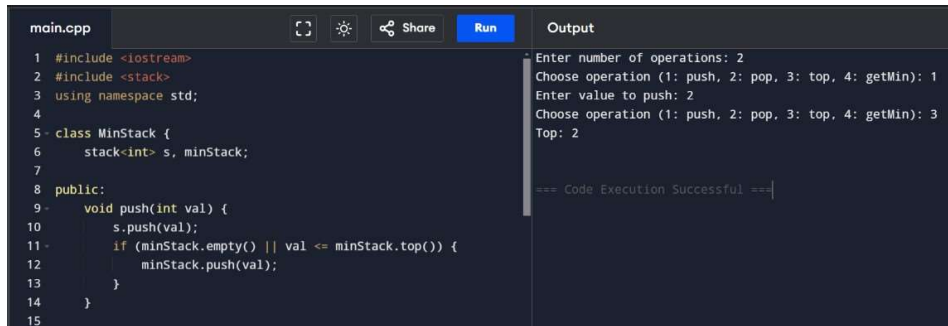
Output:

2) Balanced Brackets

```cpp
#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:


```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             }
```

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


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

3) Evaluate Reverse Polish Notation

```cpp
#include <iostream>
#include <stack>
#include <vector>
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 if (token ==
    "/") st.push(a / b);
    } else {
```

```cpp
            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:


```
1  #include <iostream>
2  #include <stack>
3  #include <vector>
4  using namespace std;
5
6  int evalRPN(vector<string>& tokens) {
7      stack<int> st;
8      for (string& 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);
```

Output:
```
Enter number of tokens: 5
Enter the tokens: 2
1
+
3
*
Result: 9

=== Code Execution Successful
```

4) Longest Valid Parentheses

```cpp
#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);
```

```cpp
        } 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:

5) Poisonous Plants

```cpp
#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]);
```

```cpp
    }
    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:

```cpp
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  int poisonousPlants(vector<int>& p) {
6      vector<int> days(p.size(), 0);
7      vector<int> stack;
8      int maxDays = 0;
9
10     for (int i = 0; i < p.size(); ++i) {
11         int day = 0;
12         while (!stack.empty() && p[stack.back()] >= p[i]) {
13             day = max(day, days[stack.back()]);
14             stack.pop_back();
15         }
```

Output:
```
Enter number of plants: 7
Enter pesticide levels: 6
5
8
4
7
10
9
Days until no plants die: 2

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

6) Implement Queue Using Stacks

```cpp
#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() {
```

```cpp
        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;
}
```
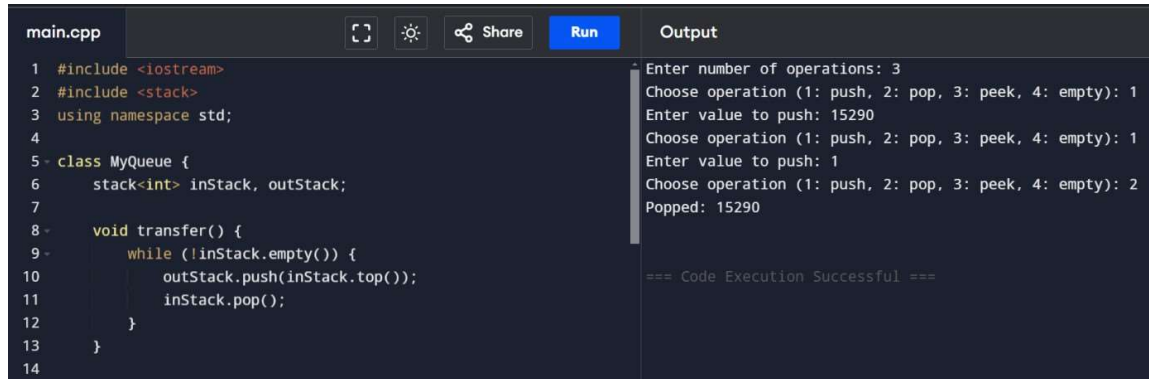
Output :


```
main.cpp                                          Output
1  #include <iostream>                            Enter number of operations: 3
2  #include <stack>                               Choose operation (1: push, 2: pop, 3: peek, 4: empty): 1
3  using namespace std;                           Enter value to push: 15290
4                                                  Choose operation (1: push, 2: pop, 3: peek, 4: empty): 1
5  class MyQueue {                                Enter value to push: 1
6      stack<int> inStack, outStack;              Choose operation (1: push, 2: pop, 3: peek, 4: empty): 2
7                                                  Popped: 15290
8      void transfer() {
9          while (!inStack.empty()) {
10             outStack.push(inStack.top());      === Code Execution Successful ===
11             inStack.pop();
12         }
13     }
14
```

7) Reverse a Queue Using Recursion

```cpp
#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 :

```cpp
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

```cpp
#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 :

9) Circular Queue Simulation

```cpp
#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();
```

```cpp
}

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;
}
```
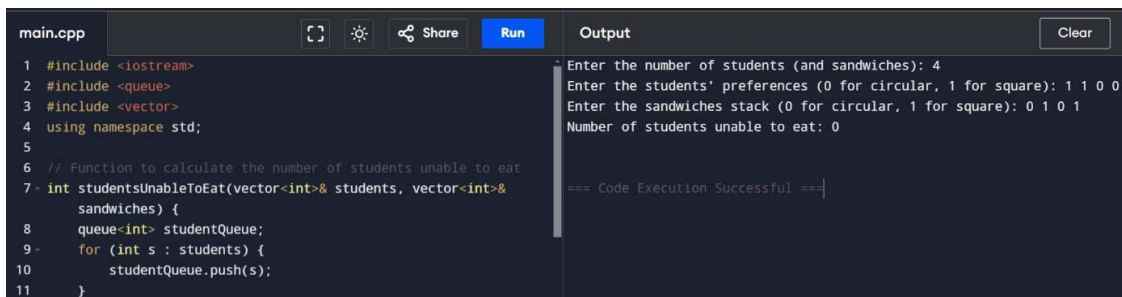
Output :

10) Zuma Game

```cpp
#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++;
```

```cpp
            }

            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 :

```cpp
#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;
```

```
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 ===
```