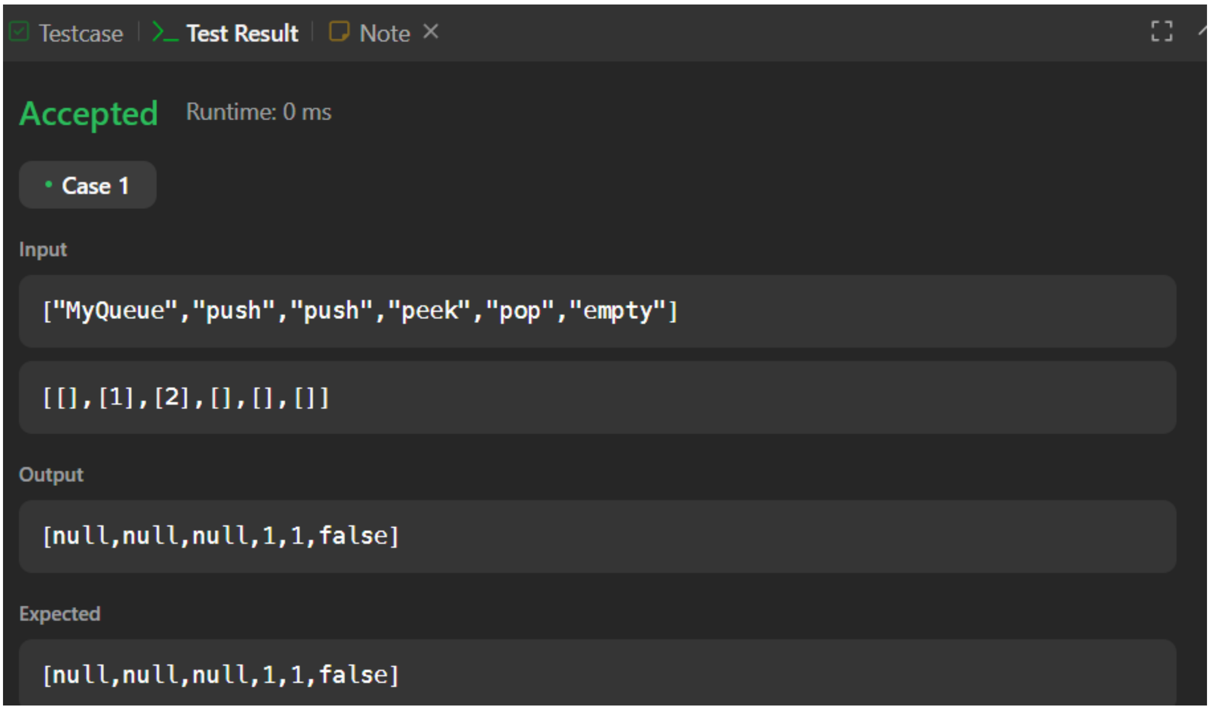
1. Implement Queue using Stacks

```
class MyQueue {
private:
    stack<int> s1, s2;


    void transferStack() {
        while (!s1.empty()) {
            s2.push(s1.top());
            s1.pop();
        }
    }

public:
    void push(int x) {
        s1.push(x);
    }

    int pop() {
        if (s2.empty()) {
            transferStack();
        }
        int front = s2.top();
        s2.pop();
        return front;
    }

    int peek() {
        if (s2.empty()) {
            transferStack();
        }
        return s2.top();
    }

    bool empty() {
        return s1.empty() && s2.empty();
    }
};
/**
 * Your MyQueue object will be instantiated and called as such:
 * MyQueue* obj = new MyQueue();
 * obj->push(x);
 * int param_2 = obj->pop();
 * int param_3 = obj->peek();
 * bool param_4 = obj->empty();
 */
```



Accepted Runtime: 0 ms

Case 1

Input

```
["MyQueue","push","push","peek","pop","empty"]
```

```
[[],[1],[2],[],[],[]]
```

Output

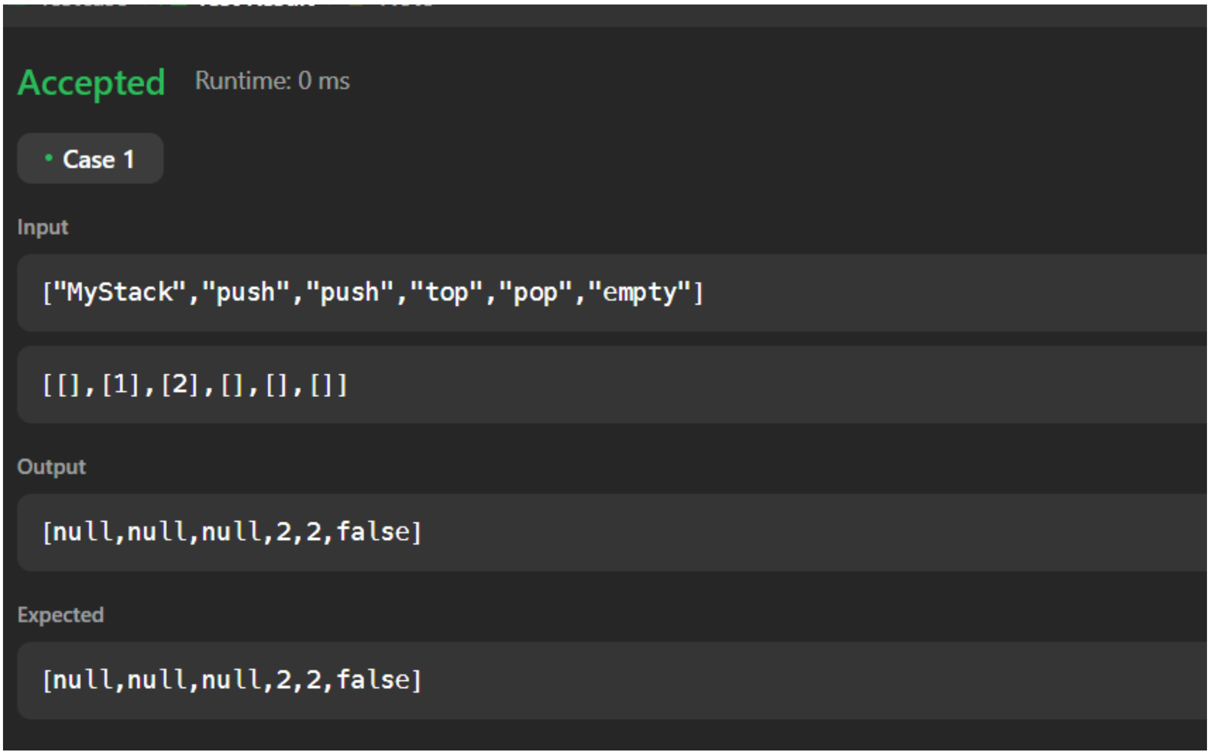
```
[null,null,null,1,1,false]
```

Expected

```
[null,null,null,1,1,false]
```

2. **Implement Stack using Queues**

```
class MyStack {
private:
    queue<int> q1, q2;

public:

    void push(int x) {
        q2.push(x);

        while (!q1.empty()) {
            q2.push(q1.front());
            q1.pop();
        }

        swap(q1, q2);
    }

    int pop() {
        int topElement = q1.front();
        q1.pop();
        return topElement;
    }

    int top() {
        return q1.front();
    }

    bool empty() {
        return q1.empty();
    }
};

/**
 * Your MyStack object will be instantiated and called as such:
 * MyStack* obj = new MyStack();
 * obj->push(x);
 * int param_2 = obj->pop();
 * int param_3 = obj->top();
 * bool param_4 = obj->empty();
 */
```



Accepted Runtime: 0 ms

• Case 1

Input

```
["MyStack","push","push","top","pop","empty"]
```

```
[[],[1],[2],[],[],[]]
```

Output

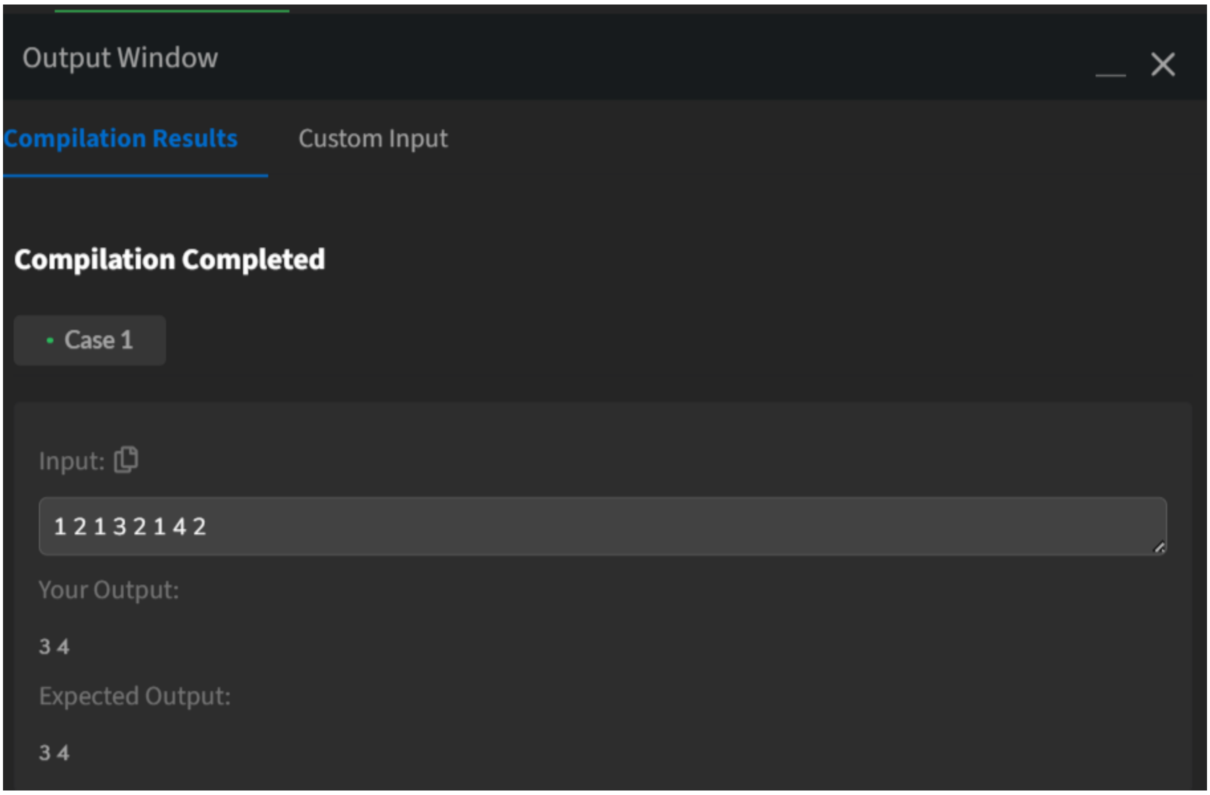
```
[null,null,null,2,2,false]
```

Expected

```
[null,null,null,2,2,false]
```

3. **Implement stack using array**

```
void MyStack ::push(int x) {
    if (top < 999) {
        top++;
        arr[top] = x;
    }
}

int MyStack ::pop() {
    if (top == -1) {
        return -1;
    }
    int popped = arr[top];
    top--;
    return popped;
}
```



Compilation Completed

Case 1

Input:

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

Your Output:

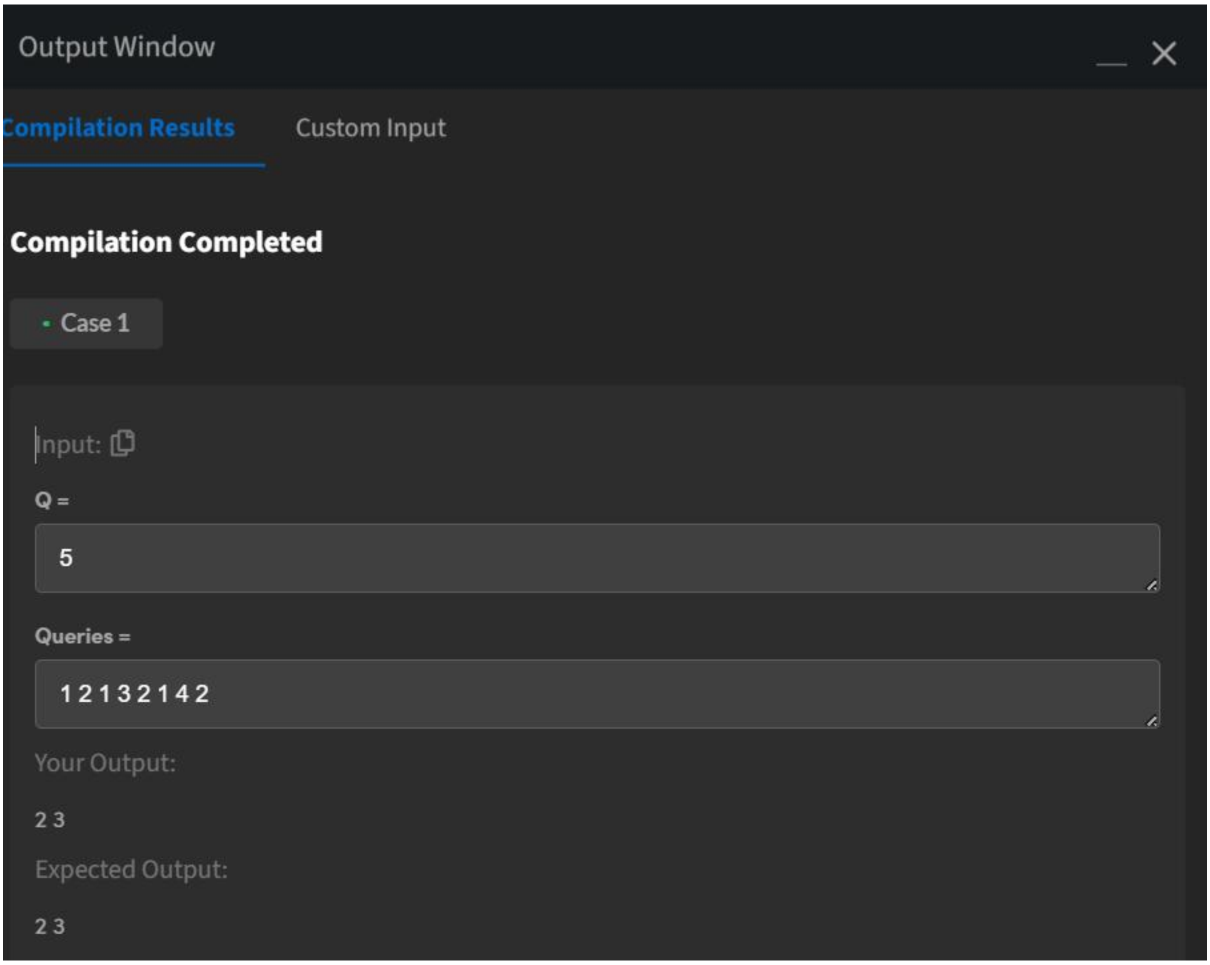
```
3 4
```

Expected Output:

```
3 4
```

4. Implement Queue Using Array

```
void MyQueue ::push(int x) {
    arr[rear++] = x;}
int MyQueue ::pop() {
 if (front == rear) return -1; // Queue is empty
   return arr[front++];
}
```



Compilation Completed

Case 1

Input:

Q =

```
5
```

Queries =

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

Your Output:

```
2 3
```

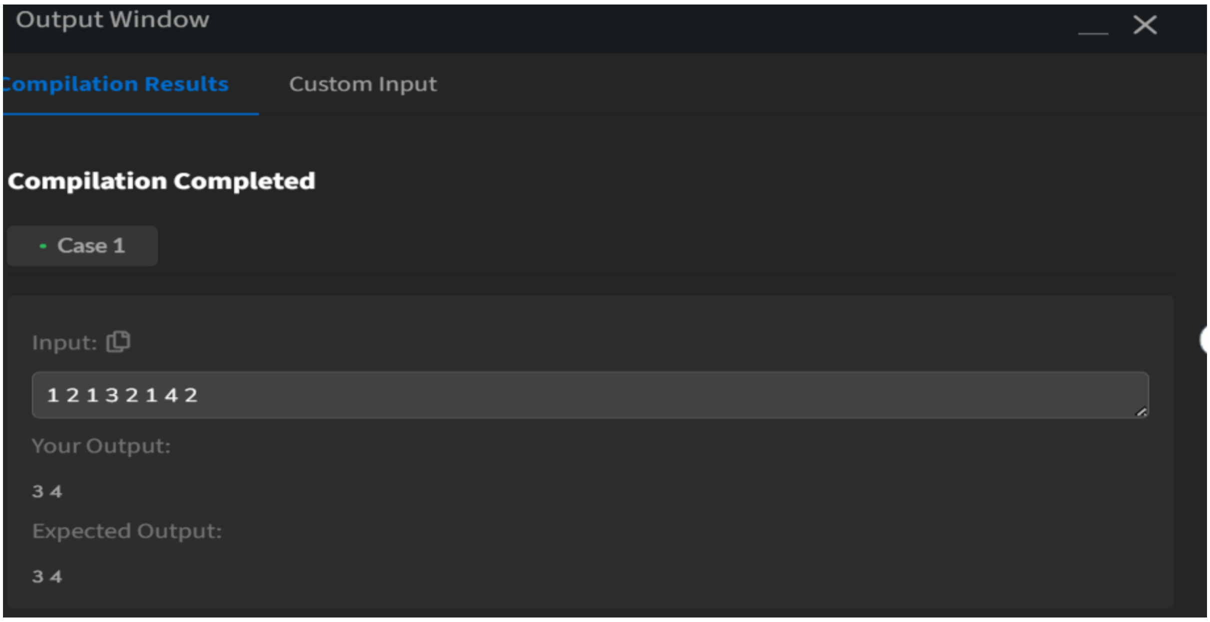
Expected Output:

```
2 3
```

5. Implement stack using linked list

```
class MyStack {
  private:
    StackNode *top;

  public:

    MyStack() { top = NULL; }
```

```
void push(int x) {
    StackNode* newNode = new StackNode(x);
    newNode->next = top;
    top = newNode;
}



int pop() {
    if (top == NULL) return -1;

    int popped = top->data;
    StackNode* temp = top;
    top = top->next;
    delete temp;

    return popped;
}
```



Compilation Completed

Input:

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

Your Output:

```
3 4
```

Expected Output:

```
3 4
```