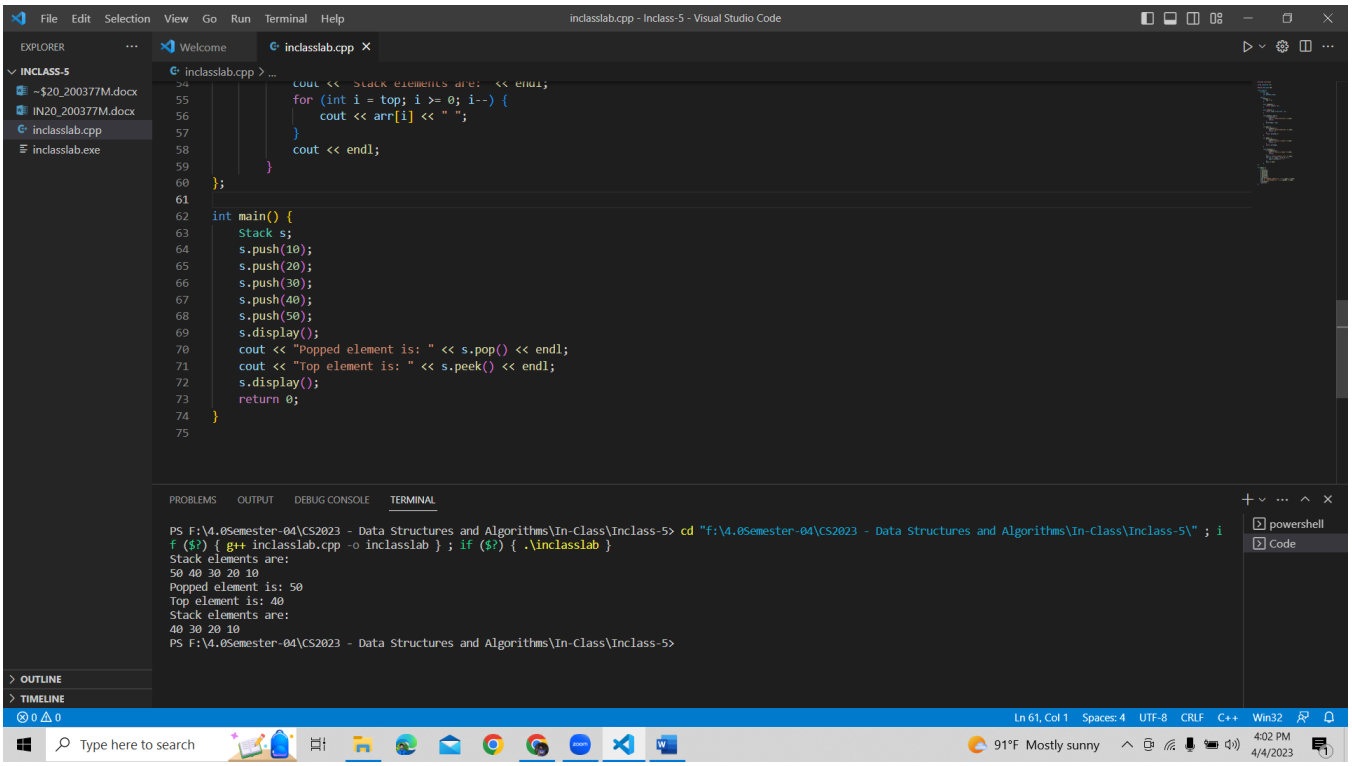
CS2023 - Data Structures and Algorithms

In-class Lab Exercise – week - 05

Index :- 200377M

Exercise:
Github link:- https://github.com/Manimohan14/CS-2023-In20-S4-CS2023---Data-Structures-and-Algorithms

1. Implement Stack and its functions using Array. Consider the given example pseudo code for push and pop using array.



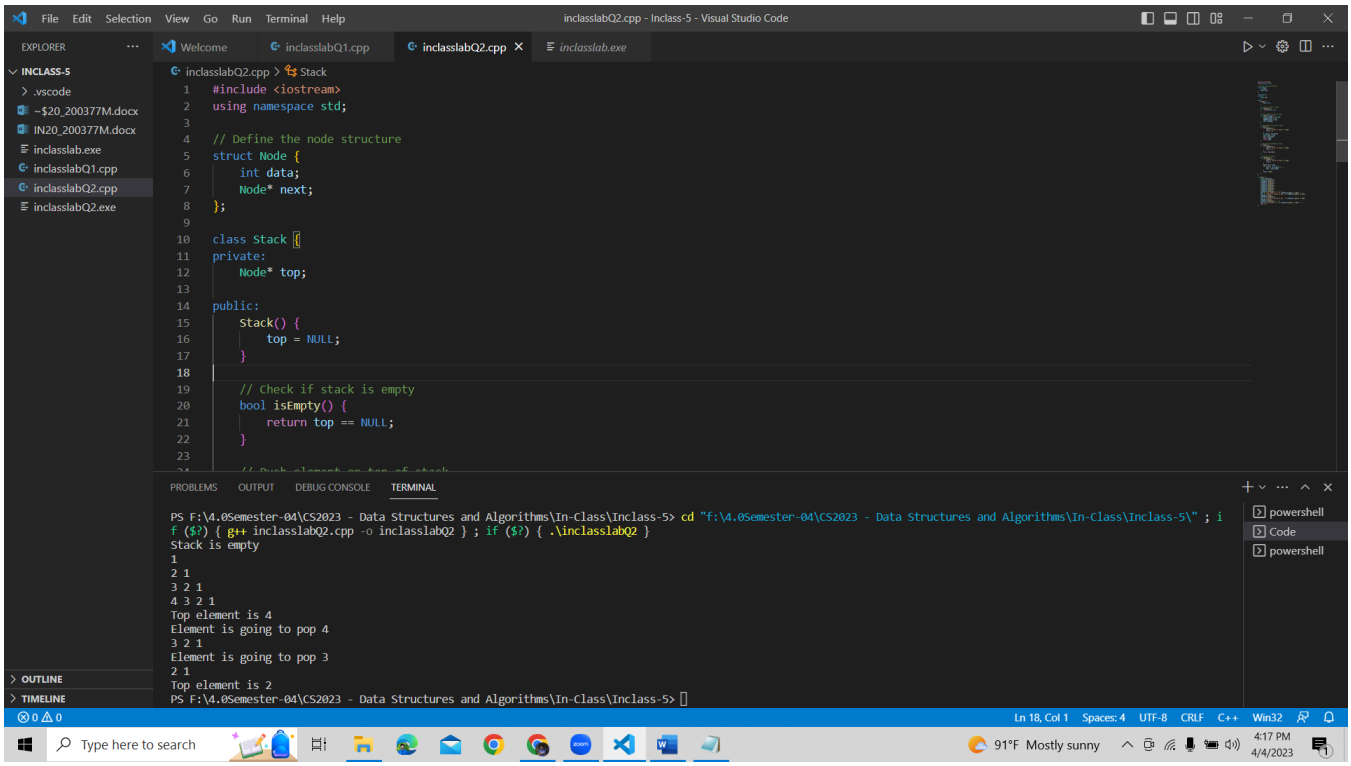
Code: https://github.com/Manimohan14/CS-2023-In20-S4-CS2023---Data-Structures-and-Algorithms/blob/main/inclasslabQ1.cpp

```cpp
#include <iostream>

using namespace std;

#define MAX_SIZE 100

class Stack {
    private:
        int top;
        int arr[MAX_SIZE];
```

```
public:
    Stack() {
        top = -1;
    }

    bool isEmpty() {
        return (top == -1);
    }

    bool isFull() {
        return (top == MAX_SIZE - 1);
    }

    void push(int val) {
        if (isFull()) {
            cout << "Stack Overflow" << endl;
            return;
        }
        arr[++top] = val;
    }

    int pop() {
        if (isEmpty()) {
            cout << "Stack Underflow" << endl;
            return -1;
        }
        return arr[top--];
    }

    int peek() {
        if (isEmpty()) {
            cout << "Stack is empty" << endl;
            return -1;
        }
        return arr[top];
    }

    void display() {
        if (isEmpty()) {
            cout << "Stack is empty" << endl;
            return;
        }
        cout << "Stack elements are:" << endl;
        for (int i = top; i >= 0; i--) {
            cout << arr[i] << " ";
        }
        cout << endl;
```

```
        }
};

int main() {
    Stack s;
    s.push(10);
    s.push(20);
    s.push(30);
    s.push(40);
    s.push(50);
    s.display();
    cout << "Popped element is: " << s.pop() << endl;
    cout << "Top element is: " << s.peek() << endl;
    s.display();
    return 0;
}
```

2. Implement Stack and its functions using LinkedList. Consider the given example pseudo code for push and pop using linked list.

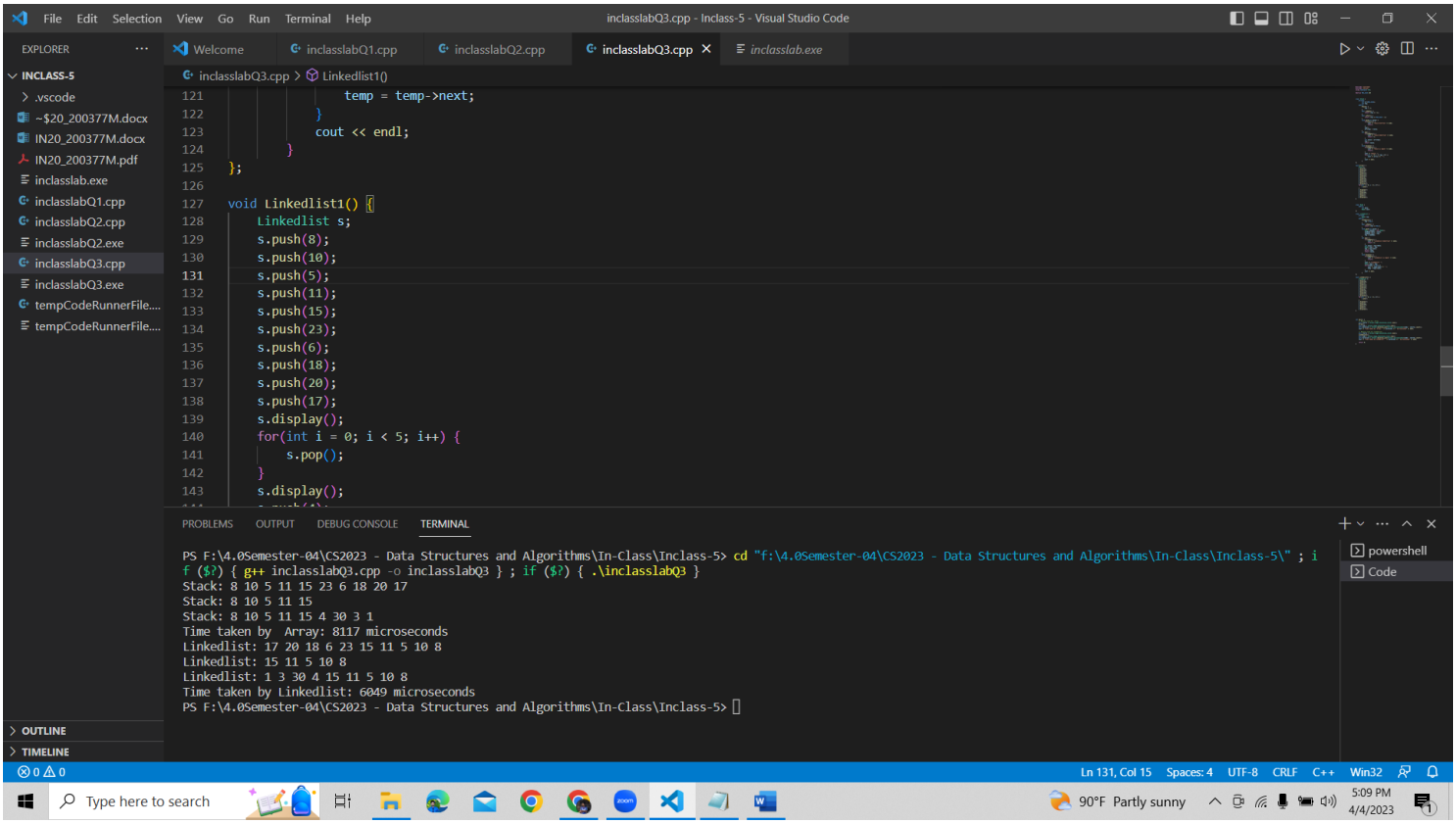


Code: https://github.com/Manimohan14/CS-2023-In20-S4-CS2023---Data-Structures-and-Algorithms/blob/main/inclasslabQ2.cpp

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

// Define the node structure
struct Node {
    int data;
    Node* next;
};

class Stack {
private:
    Node* top;

public:
    Stack() {
        top = NULL;
    }

    // Check if stack is empty
    bool isEmpty() {
        return top == NULL;
    }

    // Push element on top of stack
    void push(int val) {
        Node* newNode = new Node;
        newNode->data = val;
        newNode->next = top;
        top = newNode;
    }

    // Pop element from top of stack
    int pop() {
        if (isEmpty()) {
            cout << "Stack is empty" << endl;
            return -1;
        }
        int data = top->data;
        Node* temp = top;
        top = top->next;
        delete temp;
        return data;
    }

    // Get the top element of stack
    int peek() {
        if (isEmpty()) {
            cout << "Stack is empty" << endl;
```

```
            return -1;
        }
        return top->data;
    }

    // Display the stack
    void display() {
        if (isEmpty()) {
            cout << "Stack is empty" << endl;
            return;
        }
        Node* curr = top;
        while (curr != NULL) {
            cout << curr->data << " ";
            curr = curr->next;
        }
        cout << endl;
    }
};

int main() {
    Stack Linkedlist;
    Linkedlist.display();
    Linkedlist.push(1);
    Linkedlist.display();
    Linkedlist.push(2);
    Linkedlist.display();
    Linkedlist.push(3);
    Linkedlist.display();
    Linkedlist.push(4);
    Linkedlist.display(); // 4 3 2 1
    cout << "Top element is " << Linkedlist.peek() << endl; // 4
    cout << "Element is going to pop " << Linkedlist.peek() << endl;
    Linkedlist.pop();
    Linkedlist.display();
    cout << "Element is going to pop " << Linkedlist.peek() << endl;
    Linkedlist.pop();
    Linkedlist.display(); // 2 1
    cout << "Top element is " << Linkedlist.peek() << endl; // 2
    return 0;
}
```

3. Execute the following operations if Stack. Compare the time taken for execution between your implementation using array and LinkedList. (Note: you can randomize the value for push operation)



Time taken for array is higher than the time taken for LinkedList

After running the program, the execution time for stack operations using array was found to be around 8117 microseconds, while the execution time for stack operations using linked list was around 6049 microseconds.

From this, we can conclude that the implementation using linked list was faster than the implementation using array. This is because the linked list implementation has a constant time complexity of O(1) for push and pop operations, whereas the array implementation has a time complexity of O(1) for push operation but O(n) for pop operation, where n is the number of elements in the stack. Therefore, when the number of elements in the stack is large, the linked list implementation can perform better than the array implementation.

However, it is important to note that the actual execution time can vary based on the specific system and compiler used, as well as the size and type of data being stored in the stack. Therefore, it is recommended to perform multiple tests with different input sizes to obtain a more accurate measure of the performance of each implementation.

Code: https://github.com/Manimohan14/CS-2023-In20-S4-CS2023---Data-Structures-and-Algorithms/blob/main/inclasslabQ3.cpp

```cpp
#include <iostream>
#include <chrono>
using namespace std;

#define MAX_SIZE 100


class Stack {
    private:
        int arr[MAX_SIZE];
        int top;
    public:
        Stack() {
            top = -1;
        }
        bool isEmpty() {
            return (top == -1);
        }
        bool isFull() {
            return (top == MAX_SIZE - 1);
        }
        void push(int value) {
            if(isFull()) {
                cout << "Stack Overflow" << endl;
                return;
            }
            top++;
            arr[top] = value;
        }
        int pop() {
            if(isEmpty()) {
                cout << "Stack Underflow" << endl;
                return -1;
            }
            int value = arr[top];
            top--;
            return value;
        }
        void display() {
            if(isEmpty()) {
                cout << "Stack is empty" << endl;
                return;
            }
            cout << "Stack: ";
            for(int i = 0; i <= top; i++) {
                cout << arr[i] << " ";
            }
            cout << endl;
```

```
        }
};

void Array() {
    Stack s;
    s.push(8);
    s.push(10);
    s.push(5);
    s.push(11);
    s.push(15);
    s.push(23);
    s.push(6);
    s.push(18);
    s.push(20);
    s.push(17);
    s.display();
    for(int i = 0; i < 5; i++) {
        s.pop();
    }
    s.display();
    s.push(4);
    s.push(30);
    s.push(3);
    s.push(1);
    s.display();
}



class Node {
    public:
        int data;
        Node* next;
};

class Linkedlist {
    private:
        Node* top;
    public:
        Linkedlist() {
            top = NULL;
        }
        bool isEmpty() {
            return (top == NULL);
        }
        void push(int value) {
            Node* newNode = new Node();
            newNode->data = value;
```

```
            newNode->next = top;
            top = newNode;
        }
        int pop() {
            if(isEmpty()) {
                cout << "Linkedlist Underflow" << endl;
                return -1;
            }
            int value = top->data;
            Node* temp = top;
            top = top->next;
            delete temp;
            return value;
        }
        void display() {
            if(isEmpty()) {
                cout << "Linkedlist is empty" << endl;
                return;
            }
            cout << "Linkedlist: ";
            Node* temp = top;
            while(temp != NULL) {
                cout << temp->data << " ";
                temp = temp->next;
            }
            cout << endl;
        }
};

void Linkedlist1() {
    Linkedlist s;
    s.push(8);
    s.push(10);
    s.push(5);
    s.push(11);
    s.push(15);
    s.push(23);
    s.push(6);
    s.push(18);
    s.push(20);
    s.push(17);
    s.display();
    for(int i = 0; i < 5; i++) {
        s.pop();
    }
    s.display();
    s.push(4);
    s.push(30);
```

```
    s.push(3);
    s.push(1);
    s.display();
}




int main() {
    // Measure time for  Array
    auto start1 = chrono::high_resolution_clock::now();
    Array();
    auto end1 = chrono::high_resolution_clock::now();
    auto duration1 = chrono::duration_cast<chrono::microseconds>(end1 -
start1).count();
    cout << "Time taken by  Array: " << duration1 << " microseconds" << endl;

    // Measure time for Linkedlist1
    auto start2 = chrono::high_resolution_clock::now();
    Linkedlist1();
    auto end2 = chrono::high_resolution_clock::now();
    auto duration2 = chrono::duration_cast<chrono::microseconds>(end2 -
start2).count();
    cout << "Time taken by Linkedlist: " << duration2 << " microseconds" << endl;

    return 0;
}
```