Green University of Bangladesh
Department of Computer Science and Engineering (CSE)

Faculty of Sciences and Engineering

Semester: (Fall, Year: 2024), B.Sc. in CSE (Day)

Lab Report NO 05

Course Title: Data Structures Lab

Course Code: CSE 206   Section: 232 D10

Lab Experiment Name: Implementation of Stack using Array & Implementation of Queue using Linked List

Student Details

Name		ID
1.	Md.Rabby Khan	213902037

Lab Date	**: 08-11-2024**
Submission Date	**: 16-10-2024**
Course Teacher's Name	**: Md. Ashiqussalehin**

Lab Report Status

Marks: …………………………………   Signature:.....................

Comments:...   Date:..............................

1. TITLE OF THE LAB REPORT EXPERIMENT

Implementation of Stack using Array & Implementation of Queue using Linked List

2. OBJECTIVES/AIM

The primary objective of this lab experiment is to:

- Implement a **stack** using an **array**, allowing operations such as push, pop, and display.
- Implement a **queue** using a **linked list**, enabling operations such as enqueue, dequeue, and display.
- Understand the theoretical concepts of stack and queue as linear data structures and their practical applications.

3. PROCEDURE / ANALYSIS / DESIGN

Problem 1
Stack Algorithm using Array**:**

```
1. Initialize stack with a fixed size.
2. Set top = -1.
3. To push:
    - If top < SIZE - 1, increment top and insert value at stack[top].
4. To pop:
    - If top >= 0, return stack[top] and decrement top.
5. To display:
    - Traverse the stack from top to bottom and print each element.
```

Problem 2
Queue Algorithm (Using Linked List)**:**

```
1. Initialize front and rear as NULL.
2. To enqueue:
    - Create a new node.
    - If the queue is empty, set front and rear to the new node.
    - Else, link the current rear node to the new node, and update rear.
3. To dequeue:
    - If front is not NULL, remove the node at the front and update front.
4. To display:
    - Traverse from front to rear and print each node’s data.
```

4. IMPLEMENTATION

Problem 1

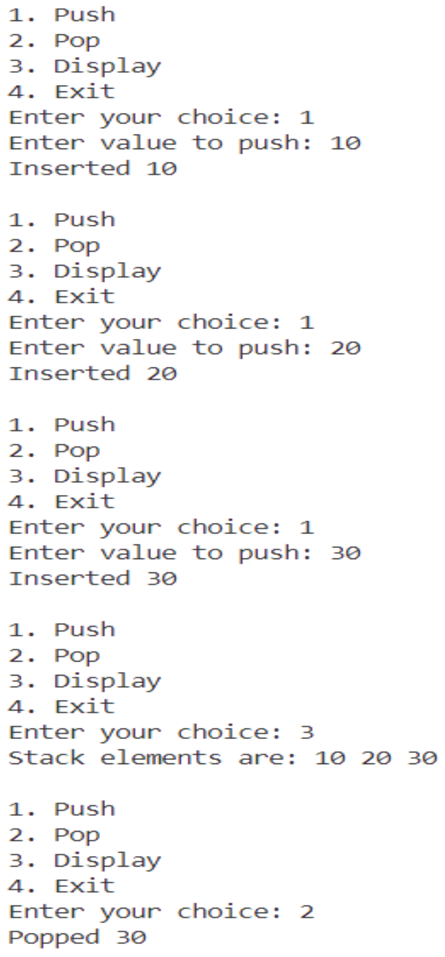
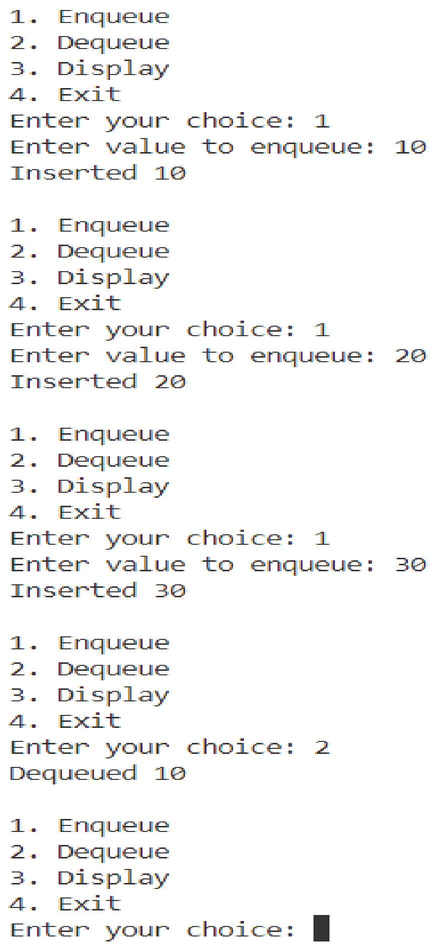
```
#include <iostream>
using namespace std;

#define SIZE 5  // Defining the size of the stack

class Stack
{
private:
    int stack[SIZE];
    int top;

public:
    Stack()
    {
        top = -1;  // Initialize top to -1 indicating the stack is empty
    }

    // Push function to add an element to the stack
    void push(int value)
    {
        if (top >= SIZE - 1)
        {
            cout << "Stack Overflow!" << endl;
        }
        else
        {
            stack[++top] = value;
            cout << "Inserted " << value << endl;
        }
    }
    // Pop function to remove an element from the stack
    void pop()
    {
        if (top == -1)
        {
            cout << "Stack Underflow!" << endl;
        }
        else
        {
            cout << "Popped " << stack[top--] << endl;
        }
    }

    // Function to display the elements of the stack
    void display()
    {
        if (top == -1)
        {
            cout << "Stack is empty!" << endl;
        }
        else
        {
            cout << "Stack elements are: ";
            for (int i = 0; i <= top; i++)
            {
                cout << stack[i] << " ";
            }
            cout << endl;
        }
    }
};
int main()
{
    Stack s;
    int choice, value;

    do
    {
        cout << "\n1. Push\n2. Pop\n3. Display\n4. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice)
        {
        case 1:
            cout << "Enter value to push: ";
            cin >> value;
            s.push(value);
            break;
        case 2:
            s.pop();
            break;
        case 3:
            s.display();
            break;
        case 4:
            cout << "Exiting..." << endl;
            break;
        default:
            cout << "Invalid choice!" << endl;
        }
    }
    while (choice != 4);

    return 0;
}
```

Problem 2

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

// Node structure to represent each element in the queue
struct Node
{
    int data;
    Node* next;
};
class Queue
{
private:
    Node* front;
    Node* rear;
public:
    Queue()
    {
        front = rear = nullptr;
    }
    // Enqueue function to add an element to the queue
    void enqueue(int value)
    {
        Node* newNode = new Node;
        newNode->data = value;
        newNode->next = nullptr;

        if (rear == nullptr)
        {
            front = rear = newNode;
        }
        else
        {
            rear->next = newNode;
            rear = newNode;
        }
        cout << "Inserted " << value << endl;
    }
    // Dequeue function to remove an element from the queue
    void dequeue()
    {
        if (front == nullptr)
        {
            cout << "Queue Underflow!" << endl;
        }
        else
        {
            Node* temp = front;
            front = front->next;
            cout << "Dequeued " << temp->data << endl;
            delete temp;
        }
    }

    // Function to display the elements of the queue
    void display()
    {
        if (front == nullptr)
        {
            cout << "Queue is empty!" << endl;
        }
        else
        {
            Node* temp = front;
            cout << "Queue elements are: ";
            while (temp != nullptr)
            {
                cout << temp->data << " ";
                temp = temp->next;
            }
            cout << endl;
        }
    }
};
```

```
int main()
{
    Queue q;
    int choice, value;
    do
    {
        cout << "\n1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;
        switch (choice)
        {
        case 1:
            cout << "Enter value to enqueue: ";
            cin >> value;
            q.enqueue(value);
            break;
        case 2:
            q.dequeue();
            break;
        case 3:
            q.display();
            break;
        case 4:
            cout << "Exiting..." << endl;
            break;
        default:
            cout << "Invalid choice!" << endl;
        }
    }
    while (choice != 4);

    return 0;
}
```

5. TEST RESULT / OUTPUT

Problem 1	Problem 2
 1. Push 2. Pop 3. Display 4. Exit Enter your choice: 1 Enter value to push: 10 Inserted 10 1. Push 2. Pop 3. Display 4. Exit Enter your choice: 1 Enter value to push: 20 Inserted 20 1. Push 2. Pop 3. Display 4. Exit Enter your choice: 1 Enter value to push: 30 Inserted 30 1. Push 2. Pop 3. Display 4. Exit Enter your choice: 3 Stack elements are: 10 20 30 1. Push 2. Pop 3. Display 4. Exit Enter your choice: 2 Popped 30 **Result Explanation:** Result shows a stack with values 10, 20, and 30 pushed on, then displayed, and finally 30 is popped off.	 1. Enqueue 2. Dequeue 3. Display 4. Exit Enter your choice: 1 Enter value to enqueue: 10 Inserted 10 1. Enqueue 2. Dequeue 3. Display 4. Exit Enter your choice: 1 Enter value to enqueue: 20 Inserted 20 1. Enqueue 2. Dequeue 3. Display 4. Exit Enter your choice: 1 Enter value to enqueue: 30 Inserted 30 1. Enqueue 2. Dequeue 3. Display 4. Exit Enter your choice: 2 Dequeued 10 1. Enqueue 2. Dequeue 3. Display 4. Exit Enter your choice: **Result Explanation:** The result shows a queue where 10, 20, and 30 are enqueued, then 10 is dequeued, leaving [20, 30].

6. ANALYSIS AND DISCUSSION

The stack correctly follows the **LIFO** (Last In, First Out) principle, while the queue adheres to the **FIFO** (First In, First Out) principle. Both data structures were implemented and tested successfully. The main challenges were managing memory dynamically for the queue and handling stack indices. Despite these difficulties, the hands-on experience solidified my understanding of stack and queue operations. I learned how these structures work and gained experience with dynamic memory management and linked lists. The objectives were successfully met by implementing the stack using arrays and the queue using linked lists.

7. SUMMARY

In this experiment, I implemented the stack and queue data structures—stack using an array and queue using a linked list. The stack followed the **LIFO** principle, and the queue followed the **FIFO** principle. The operations were tested successfully, enhancing my understanding of these data structures.