

**DSA (Data Structure and Algorithms) Lab**

**LAB REPORT # 4**

**Semester**: 3rdSemester

**Section**: C

**Submitted To:**

**Submitted By:**

**Name**: Faisal Khan

**Roll No**: 22-CS-039

**Task 1: Write difference between queue and stack.**

Stack and Queue are two fundamental data structures used in computer science and programming for managing collections of data. They have distinct characteristics that make them suitable for different types of tasks. Here are the key differences between a stack and a queue:

1. **Order of Data Access:**
   * **Stack:** Follows the Last-In-First-Out (LIFO) order. The most recently added item is the first one to be removed.
   * **Queue:** Follows the First-In-First-Out (FIFO) order. The item that has been in the queue the longest is the first one to be removed.
2. **Insertion and Removal:**
   * **Stack:** Items are inserted and removed from the same end, known as the "top" of the stack.
   * **Queue:** Items are inserted at one end (the "rear" or "back") and removed from the other end (the "front").
3. **Use Cases:**
   * **Stack:** Typically used for tasks that involve reversing or tracking function calls (call stack) and undo operations. It's suitable for situations where you need to access the most recently added item quickly.
   * **Queue:** Often used in scenarios where tasks are processed in the order they arrive, such as task scheduling, print spooling, and breadth-first search algorithms.
4. **Example Real-World Analogy:**
   * **Stack:** Think of a stack of plates. You add or remove plates from the top of the stack.
   * **Queue:** Think of people waiting in a line (queue) at a ticket counter. The person who arrives first is the first one to get served.
5. **Implementation:**
   * **Stack:** Can be implemented using arrays or linked lists with push (to add) and pop (to remove) operations.
   * **Queue:** Can be implemented using arrays or linked lists with enqueue (to add) and dequeue (to remove) operations.
6. **Complexity:**
   * Both stacks and queues generally have O(1) time complexity for adding and removing elements in most implementations.
7. **Variations:**
   * There are variations of both data structures, such as double-ended queues (deque) that allow insertion and removal from both ends, and priority queues where elements are processed based on their priority.

In summary, the choice between a stack and a queue depends on the specific problem you need to solve. If you need to access the most recently added item first, a stack is appropriate (LIFO). If you need to process items in the order they were added, a queue is the right choice (FIFO).

**Task 2:**

#include <iostream>

using namespace std;

class CircularQueue

{

private:

    int \*queue;

    int front;

    int rear;

    int maxSize;

public:

    CircularQueue(int size)

    {

        maxSize = size;

        queue = new int[size];

        front = rear = -1;

    }

    ~CircularQueue()

    {

        delete[] queue;

    }

    bool isEmpty()

    {

        return front == -1;

    }

    bool isFull()

    {

        return (front == 0 && rear == maxSize - 1) || (front == rear + 1);

    }

    void enqueue(int data)

    {

        if (isFull())

        {

            cout << "Queue is full. Cannot enqueue." << endl;

        }

        else

        {

            if (front == -1)

            {

                front = 0;

                rear = 0;

            }

            else

            {

                rear = (rear + 1) % maxSize;

            }

            queue[rear] = data;

            cout << "Enqueued: " << data << endl;

        }

    }

    void dequeue()

    {

        if (isEmpty())

        {

            cout << "Queue is empty. Cannot dequeue." << endl;

        }

        else

        {

            int data = queue[front];

            if (front == rear)

            {

                front = rear = -1;

            }

            else

            {

                front = (front + 1) % maxSize;

            }

            cout << "Dequeued: " << data << endl;

        }

    }

    void display()

    {

        if (isEmpty())

        {

            cout << "Queue is empty." << endl;

        }

        else

        {

            cout << "Queue elements: ";

            int i = front;

            while (true)

            {

                cout << queue[i] << " ";

                if (i == rear)

                    break;

                i = (i + 1) % maxSize;

            }

            cout << endl;

        }

    }

    int peek()

    {

        if (isEmpty())

        {

            cout << "Queue is empty. No element to peek." << endl;

            return -1;

        }

        else

        {

            cout << "Peeked element: " << queue[front] << endl;

            return queue[front];

        }

    }

};

int main()

{

    int size;

    cout << "Enter the size of the circular queue: ";

    cin >> size;

    CircularQueue cq(size);

    int choice, data;

    while (true)

    {

        cout << "\nCircular Queue Operations:" << endl;

        cout << "1. Enqueue" << endl;

        cout << "2. Dequeue" << endl;

        cout << "3. Display" << endl;

        cout << "4. Peek" << endl;

        cout << "5. Quit" << endl;

        cout << "Enter your choice: ";

        cin >> choice;

        switch (choice)

        {

        case 1:

            cout << "Enter data to enqueue: ";

            cin >> data;

            cq.enqueue(data);

            break;

        case 2:

            cq.dequeue();

            break;

        case 3:

            cq.display();

            break;

        case 4:

            cq.peek();

            break;

        case 5:

            exit(0);

        default:

            cout << "Invalid choice. Please try again." << endl;

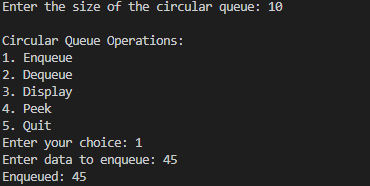
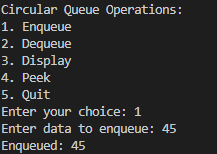
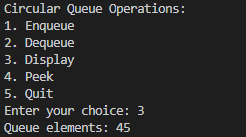
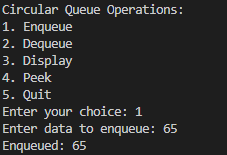
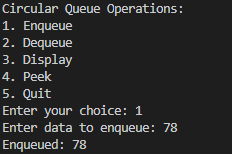
        }

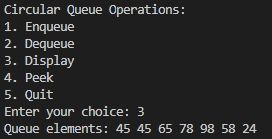
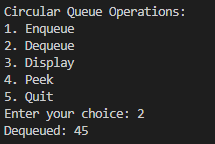
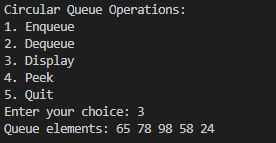
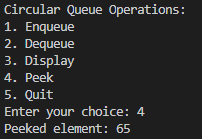
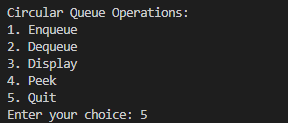
    }

    return 0;

}

**Output:**

**Task 3:**

#include <iostream>

using namespace std;

const int MAX\_SIZE = 100;

class Queue

{

private:

    int arr[MAX\_SIZE];

    int front;

    int rear;

public:

    Queue()

    {

        front = -1;

        rear = -1;

    }

    bool isFull()

    {

        return (rear == MAX\_SIZE - 1);

    }

    bool isEmpty()

    {

        return (front == -1);

    }

    void enqueue(int data)

    {

        if (isFull())

        {

            cout << "Queue is full. Cannot enqueue." << endl;

            return;

        }

        if (isEmpty())

        {

            front = 0;

        }

        rear++;

        arr[rear] = data;

        cout << "Enqueued: " << data << endl;

    }

    void dequeue()

    {

        if (isEmpty())

        {

            cout << "Queue is empty. Cannot dequeue." << endl;

            return;

        }

        int data = arr[front];

        if (front == rear)

        {

            front = rear = -1;

        }

        else

        {

            front++;

        }

        cout << "Dequeued: " << data << endl;

    }

    void display()

    {

        if (isEmpty())

        {

            cout << "Queue is empty." << endl;

            return;

        }

        cout << "Queue elements: ";

        for (int i = front; i <= rear; i++)

        {

            cout << arr[i] << " ";

        }

        cout << endl;

    }

    int count()

    {

        return (rear - front + 1);

    }

    int peek()

    {

        if (isEmpty())

        {

            cout << "Queue is empty. No element to peek." << endl;

            return -1;

        }

        cout << "Peeked element: " << arr[front] << endl;

        return arr[front];

    }

};

int main()

{

    Queue queue;

    int choice, data;

    while (true)

    {

        cout << "---------------------- MAIN MENU FOR QUEUE ----------------------" << endl;

        cout << "1. ENQUEUE" << endl;

        cout << "2. DEQUEUE" << endl;

        cout << "3. Display" << endl;

        cout << "4. Count" << endl;

        cout << "5. isEmpty" << endl;

        cout << "6. Peek" << endl;

        cout << "7. QUIT" << endl;

        cout << "Enter your choice: ";

        cin >> choice;

        switch (choice)

        {

        case 1:

            cout << "Enter data to enqueue: ";

            cin >> data;

            queue.enqueue(data);

            break;

        case 2:

            queue.dequeue();

            break;

        case 3:

            queue.display();

            break;

        case 4:

            cout << "Queue size: " << queue.count() << endl;

            break;

        case 5:

            if (queue.isEmpty())

            {

                cout << "Queue is empty." << endl;

            }

            else

            {

                cout << "Queue is not empty." << endl;

            }

            break;

        case 6:

            queue.peek();

            break;

        case 7:

            cout << "Exiting program." << endl;

            exit(0);

        default:

            cout << "Invalid choice. Please try again." << endl;

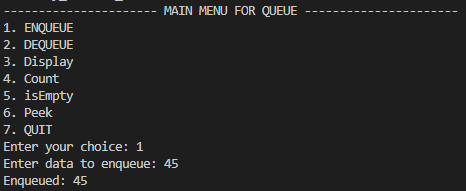
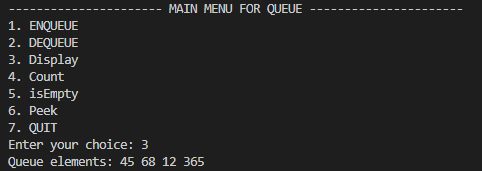
        }

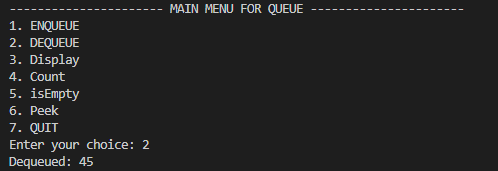
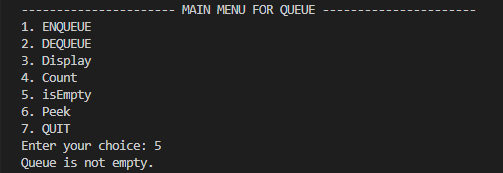
    }

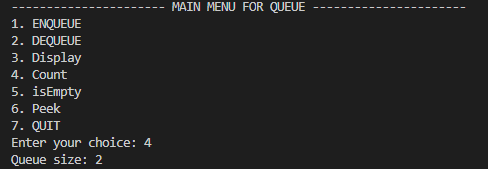
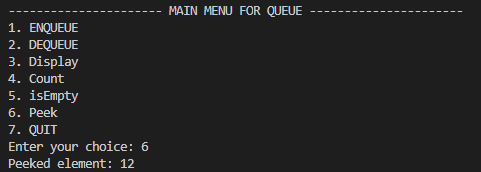
    return 0;

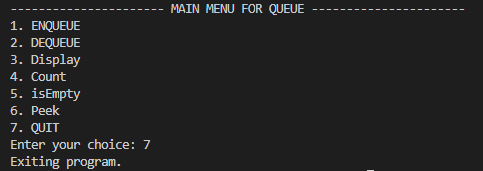
}

**Output:**

****

**Task 4:**

#include <iostream>

#include <stack>

using namespace std;

class QueueUsingStacks

{

private:

    stack<int> enqueueStack;

    stack<int> dequeueStack;

public:

    void enqueue(int data)

    {

        enqueueStack.push(data);

    }

    void transferToDequeueStack()

    {

        while (!enqueueStack.empty())

        {

            dequeueStack.push(enqueueStack.top());

            enqueueStack.pop();

        }

    }

    void dequeue()

    {

        if (dequeueStack.empty())

        {

            if (enqueueStack.empty())

            {

                cout << "Queue is empty. Cannot dequeue." << endl;

                return;

            }

            transferToDequeueStack();

        }

        int data = dequeueStack.top();

        dequeueStack.pop();

        cout << "Dequeued: " << data << endl;

    }

    void view()

    {

        if (dequeueStack.empty() && enqueueStack.empty())

        {

            cout << "Queue is empty." << endl;

            return;

        }

        transferToDequeueStack();

        stack<int> tempStack = dequeueStack;

        cout << "Queue elements: ";

        while (!tempStack.empty())

        {

            cout << tempStack.top() << " ";

            tempStack.pop();

        }

        cout << endl;

    }

};

int main()

{

    QueueUsingStacks queue;

    int choice, data;

    while (true)

    {

        cout << "---------------------- MAIN MENU FOR QUEUE ----------------------" << endl;

        cout << "1. ENQUEUE" << endl;

        cout << "2. DEQUEUE" << endl;

        cout << "3. VIEW" << endl;

        cout << "4. QUIT" << endl;

        cout << "Enter your choice: ";

        cin >> choice;

        switch (choice)

        {

        case 1:

            cout << "Enter data to enqueue: ";

            cin >> data;

            queue.enqueue(data);

            break;

        case 2:

            queue.dequeue();

            break;

        case 3:

            queue.view();

            break;

        case 4:

            cout << "Exiting program." << endl;

            exit(0);

        default:

            cout << "Invalid choice. Please try again." << endl;

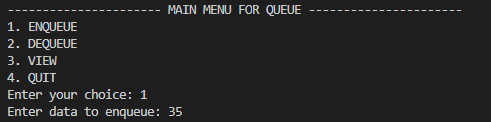
        }

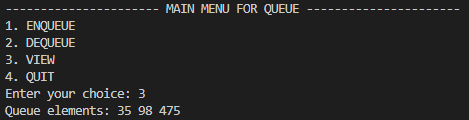
    }

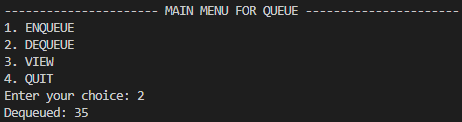
    return 0;

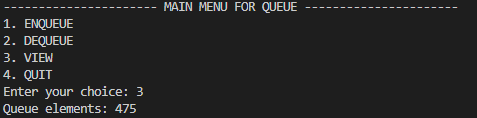
}

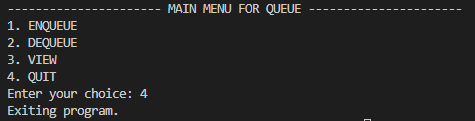
**Output:**

****

****

****

****

****