Batch: B2	**Roll No.:** 16010124107
Experiment / assignment / tutorial No. 4	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

Title: Implementation of Queue operations (Static and Dynamic implementation)- Queue, circular queue.

Objective: To implement Basic Operations of Queues

Expected Outcome of Experiment:

CO	Outcome
2	Apply linear and non-linear data structure in application development.

Books/ Journals/ Websites referred:

Introduction:

Queue is a linear structure that follows a particular order in performing operations. There are two ends. Data is inserted from one end called the rear end and removed from the other, called the front end.

Main queue operations:

Enqueue

- Add an element to the queue
- Happens at the rear end

Dequeue

- Remove an element from the queue

- Happens at the front end

Program source code:

operations to be performed for array(static) and Linked list(Dynamic)

1. **Enqueue**
2. **Dequeue:**
3. **Peek / Front:**
4. **isEmpty**
5. **isFull**

```
#include <bits/stdc++.h>

using namespace std;


int main() {

    int front = -1;

    int rear = -1;

    int cap = 0;


    cout << "Enter capacity of queue\n";

    cin >> cap;

    int q[cap];


    while (true) {

                cout << "Options:\n1. Enqueue\n2. Dequeue\n3. Peek\n4.
isEmpty\n5. isFull\n6. Exit\n";

        cout << "Enter choice\n";

        int choice;

        cin >> choice;
```

```
        switch (choice) {
            case 1: {
                cout << "Enqueue Operation\n";
                if ((front == (rear + 1) % cap)) {
                    cout << "Queue is full\n";
                } else {
                    int el;
                    cout << "Enter element\n";
                    cin >> el;
                    if (front == -1) { // first element
                        front = rear = 0;
                    } else {
                        rear = (rear + 1) % cap;
                    }
                    q[rear] = el;
                }
                break;
            }

            case 2: {
                cout << "Dequeue Operation\n";
                if (front == -1) {
                    cout << "Queue is empty\n";
                } else {
```

```
            cout << q[front] << endl;
            if (front == rear) { // last element removed
                front = rear = -1;
            } else {
                front = (front + 1) % cap;
            }
        }
        break;
    }

    case 3:
        cout << "Peek operation\n";
        if (front == -1)
            cout << "Queue is empty\n";
        else
            cout << q[front] << endl;
        break;

    case 4:
        if (front == -1)
            cout << "Queue is empty\n";
        else
            cout << "Queue is not empty\n";
        break;
```

```
            case 5:
                if ((front == (rear + 1) % cap))
                    cout << "Queue is full\n";
                else
                    cout << "Queue is not full\n";
                break;

            default:
                return 0;
        }
    }
}
```

Output Screenshots:

```
Enter capacity of queue
2
Options:
1. Enqueue
2. Dequeue
3. Peek
4. isEmpty
5. isFull
6. Exit
Enter choice
1
Enqueue Operation
Enter element
123
Options:
1. Enqueue
2. Dequeue
3. Peek
4. isEmpty
5. isFull
6. Exit
Enter choice
1
Enqueue Operation
Enter element
234
Options:
1. Enqueue
2. Dequeue
3. Peek
4. isEmpty
5. isFull
6. Exit
Enter choice
2
Dequeue Operation
123
Options:
```

```
PROBLEMS 15    OUTPUT

PS C:\Users\syeda\OneDr
xp6 }
123
Options:
1. Enqueue
2. Dequeue
3. Peek
4. isEmpty
5. isFull
6. Exit
Enter choice
1
Enqueue Operation
Enter element
898
Options:
1. Enqueue
2. Dequeue
3. Peek
4. isEmpty
5. isFull
6. Exit
Enter choice
3
Peek operation
234
Options:
1. Enqueue
2. Dequeue
3. Peek
4. isEmpty
5. isFull
6. Exit
Enter choice
4
Queue is not empty
Options:
1. Enqueue
2. Dequeue
3. Peek
```

```
3. Peek
4. isEmpty
5. isFull
6. Exit
Enter choice
5
Queue is full
Options:
1. Enqueue
2. Dequeue
3. Peek
4. isEmpty
5. isFull
6. Exit
Enter choice
2
Dequeue Operation
234
Options:
1. Enqueue
2. Dequeue
3. Peek
4. isEmpty
5. isFull
6. Exit
Enter choice
5
Queue is not full
Options:
1. Enqueue
2. Dequeue
3. Peek
4. isEmpty
5. isFull
6. Exit
Enter choice
6
PS C:\Users\sveda\OneDrive
```

Code for dynamic queue using linked list:-

```
#include <bits/stdc++.h>

using namespace std;


struct Node {

    int data;

    Node* next;

};


int main() {

    Node* front = NULL;

    Node* rear = NULL;

    int cap;

    int size = 0;


    cout << "Enter capacity of queue\n";

    cin >> cap;


    while (true) {

        cout << "Options:\n1. Enqueue\n2. Dequeue\n3. Peek\n4.
isEmpty\n5. isFull\n6. Exit\n";

        cout << "Enter choice\n";

        int choice;

        cin >> choice;
```

```
        switch (choice) {
            case 1: {
                cout << "Enqueue Operation\n";
                if (size == cap) {
                    cout << "Queue is full\n";
                } else {
                    int el;
                    cout << "Enter element\n";
                    cin >> el;
                    Node* temp = new Node();
                    temp->data = el;
                    temp->next = NULL;
                    if (front == NULL) { // first element
                        front = rear = temp;
                        rear->next = front; // circular link
                    } else {
                        rear->next = temp;
                        rear = temp;
                        rear->next = front;
                    }
                    size++;
                }
                break;
            }
```

```
        case 2: {
            cout << "Dequeue Operation\n";
            if (front == NULL) {
                cout << "Queue is empty\n";
            } else {
                cout << front->data << endl;
                if (front == rear) { // single element
                    delete front;
                    front = rear = NULL;
                } else {
                    Node* temp = front;
                    front = front->next;
                    rear->next = front;
                    delete temp;
                }
                size--;
            }
            break;
        }

        case 3:
            cout << "Peek operation\n";
            if (front == NULL)
```

```
                    cout << "Queue is empty\n";
                else
                    cout << front->data << endl;
                break;

            case 4:
                if (front == NULL)
                    cout << "Queue is empty\n";
                else
                    cout << "Queue is not empty\n";
                break;

            case 5:
                if (size == cap)
                    cout << "Queue is full\n";
                else
                    cout << "Queue is not full\n";
                break;

            default:
                return 0;
        }
    }
}
```

Output:-

```
Enter capacity of queue
2
Options:
1. Enqueue
2. Dequeue
3. Peek
4. isEmpty
5. isFull
6. Exit
Enter choice
1
Enqueue Operation
Enter element
234
Options:
1. Enqueue
2. Dequeue
3. Peek
4. isEmpty
5. isFull
6. Exit
Enter choice
1
Enqueue Operation
Enter element
421
Options:
1. Enqueue
2. Dequeue
3. Peek
4. isEmpty
5. isFull
6. Exit
Enter choice
2
Dequeue Operation
234
Options:
```

```
2
Dequeue Operation
234
Options:
1. Enqueue
2. Dequeue
3. Peek
4. isEmpty
5. isFull
6. Exit
Enter choice
2
Dequeue Operation
421
Options:
1. Enqueue
2. Dequeue
3. Peek
4. isEmpty
5. isFull
6. Exit
Enter choice
3
Peek operation
Queue is empty
Options:
1. Enqueue
2. Dequeue
3. Peek
4. isEmpty
5. isFull
6. Exit
Enter choice
5
Queue is not full
Options:
1. Enqueue
2. Dequeue
3. Peek
4. isEmpty
```

```
2. Dequeue
3. Peek
4. isEmpty
5. isFull
6. Exit
Enter choice
1
Enqueue Operation
Enter element
23
Options:
1. Enqueue
2. Dequeue
3. Peek
4. isEmpty
5. isFull
6. Exit
Enter choice
1
Enqueue Operation
Enter element
89
Options:
1. Enqueue
2. Dequeue
3. Peek
4. isEmpty
5. isFull
6. Exit
Enter choice
6
```

Conclusion:-

A queue is a powerful data structure to implement the real world structure of a queue. It demonstrates a first in first out principle for insertion and deletion of elements. It is very useful in applications where the sequence of elements matter, for example when queuing for a reservation in IRCTC trains. People who book first get seats first upon availability.

Post lab questions:

1. What are the key considerations in choosing between a circular array and a linked list for the implementation of a queue?

   When choosing between a circular array and a linked list for queue implementation, memory and size flexibility are important considerations. A circular array has a fixed size, which can waste space if underutilized or require costly resizing when full, while a linked list grows and shrinks dynamically but uses extra memory for pointers. Generally, circular arrays are preferred when the maximum size is known and fixed, while linked lists are better for unbounded or unpredictable queue sizes.

2. Describe how a deque can be implemented using both arrays and linked lists. What are the advantages and disadvantages of each approach?

   A deque (double-ended queue) can be implemented using both arrays and linked lists, each with its own pros and cons.

   - Using arrays, a deque is usually built with a circular array where front and rear indices wrap around. This allows insertion and deletion from both ends in constant time.

     Advantages: faster access

     Disadvantages: Limited flexibility, cannot resize

   - Using linked lists, a deque is implemented as a doubly linked list where each node has pointers to both its previous and next nodes. Insertions and deletions at both ends are constant time without the need for shifting or resizing.

     Advantages: Dynamic growth and shrinkage without waste of space

     Disadvantages: Higher memory overhead, slower access

3. Discuss how different types of queues can be used in real-world applications, such as job scheduling, CPU task management, and customer service systems.

Applications of queues:

- Job scheduling in operating systems
- CPU task management (ready queue, waiting queue)
- Customer service systems (call centers, bank counters)
- Printer task management (print queue)

In printer task management, a queue is used to store all print jobs sent by users. When multiple users send documents to the same printer, the jobs enter the print queue in order of arrival. The printer then processes them one by one, usually using the First-Come, First-Served (FCFS) principle. This ensures fairness so that no job is skipped.