REPORT

ON

STUDENT RECORD MANAGEMENT SYSTEM (SRMS)

Submitted by:

Bhaumik Hinunia (AP24110010182)

Zeeshan Shaik Suhail (AP24110010183)

Amith Ratna Marisa (AP24110012110)

Prepared in the partial fulfillment of the

Project Based Learning of Course CSE 201 — CODING SKILLS - I

SRM UNIVERSITY AP

Acknowledgements
I express my sincere gratitude to SRM University AP and the Department of Computer Science and Engineering for providing guidance and support throughout this project work. I would also like to thank my friends and classmates who helped with testing and shared valuable feedback, which improved the overall quality of this project.

Abstract
This project presents a Queue Simulation system developed in C++ using the concept of a Circular Queue. The program simulates a real-life ticket counter where customers are added to the queue using enqueue and served using dequeue. The system displays the queue after every operation and includes features like viewing front and rear elements, checking overflow or underflow, and showing the current queue size. The project demonstrates array-based queue implementation, circular pointer movement, FIFO behavior, modular programming, and a menu-driven interface.

Table of Contents

1. Introduction

The purpose of this project is to simulate the working of a Queue using a Circular Queue implementation in C++. The queue is implemented using an array where the front and rear pointers move in a circular manner using modular arithmetic.
This project demonstrates FIFO (First In First Out), memory-efficient circular queue operations, pointer movement, menu-driven functionality, and structured programming.

2. Objectives

The main objectives of this project are:

- Simulate real-world queue behavior
- Implement Circular Queue using an array
- Perform enqueue and dequeue operations
- Display front, rear, and queue status
- Handle overflow and underflow cases
- Demonstrate modular programming with functions

3. System Overview

The Queue Simulation project is implemented using a circular queue where all operations—enqueue, dequeue, display, and front/rear access—are performed efficiently.
The program starts by taking the maximum queue size from the user. Each new customer is assigned a token number and added to the queue using enqueue. When a customer is served, the dequeue operation removes the front element.
Front and rear pointers automatically wrap around using the modulo operator, ensuring efficient use of memory and avoiding the need to shift elements.
The display function shows all customers currently waiting along with the front and rear indices. Overflow and underflow messages are shown when operations exceed queue limits.
The program uses a clean menu-driven structure, making it easy for users to interact with the system and understand how queues work internally.

4. Code Snippets

Project 1 > Simulation.cpp > main()

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

class CircularQueue {
private:
    int *arr;         // array to store queue elements
    int capacity;     // maximum capacity of the queue
    int front;        // index of front element
    int rear;         // index of last element
    int count;        // current size of the queue

public:
    CircularQueue(int size) {
        capacity = size;
        arr = new int[capacity];
        front = 0;
        rear = -1;
        count = 0;
    }

    ~CircularQueue() {
        delete[] arr;
    }

    bool isEmpty() const {
        return count == 0;
    }

    bool isFull() const {
        return count == capacity;
    }

    void enqueue(int x) {
        if (isFull()) {
            cout << "\nQueue Overflow! No more customers can wait.\n";
            return;
        }
        rear = (rear + 1) % capacity;
        arr[rear] = x;
        count++;
        cout << "Customer with token " << x << " joined the queue.\n";
    }

    int dequeue() {
        if (isEmpty()) {
            cout << "\nQueue Underflow! No customers to serve.\n";
            return -1;
        }
        int x = arr[front];
        front = (front + 1) % capacity;
        count--;
        cout << "Customer with token " << x << " has been served.\n";
        return x;
    }

```

```
    int getFront() const {
        if (isEmpty()) {
            cout << "Queue is empty.\n";
            return -1;
        }
        return arr[front];
    }

    int getRear() const {
        if (isEmpty()) {
            cout << "Queue is empty.\n";
            return -1;
        }
        return arr[rear];
    }

    int size() const {
        return count;
    }

    void displayQueue() const {
        if (isEmpty()) {
            cout << "\nQueue is empty. No customers waiting.\n";
            return;
        }

        cout << "\nCurrent Queue (front -> rear):\n";
        int i = front;
        for (int c = 0; c < count; c++) {
            cout << arr[i] << "  ";
            i = (i + 1) % capacity;
        }
        cout << "\n";
        cout << "Front index: " << front << ", Rear index: " << rear << "\n";
    }
};

int main() {
    int maxSize;
    cout << "====== TICKET COUNTER QUEUE SIMULATION ======\n";
    cout << "Enter maximum number of customers that can wait in queue: ";
    cin >> maxSize;

    CircularQueue q(maxSize);
    int choice;
    int nextToken = 1; // token number generator

    do {
        cout << "\n------------- MENU -------------\n";
        cout << "1. New customer arrives (Enqueue)\n";
        cout << "2. Serve next customer (Dequeue)\n";
        cout << "3. Show all waiting customers\n";
        cout << "4. Show front & rear customer\n";
        cout << "5. Show queue size\n";
```

Project 1 > Simulation.cpp > main()

```
110         cout << "6. Exit\n";
111         cout << "------------------------------\n";
112         cout << "Enter your choice: ";
113         cin >> choice;
114
115         switch (choice) {
116         case 1:
117             q.enqueue(nextToken);
118             nextToken++;
119             q.displayQueue();
120             break;
121
122         case 2:
123             q.dequeue();
124             q.displayQueue();
125             break;
126
127         case 3:
128             q.displayQueue();
129             break;
130
131         case 4: {
132             int f = q.getFront();
133             int r = q.getRear();
134             if (!q.isEmpty()) {
135                 cout << "Front customer token: " << f << "\n";
136                 cout << "Rear  customer token: " << r << "\n";
137             }
138             break;
139         }
140
141         case 5:
142             cout << "Number of customers waiting: " << q.size() << "\n";
143             break;
144
145         case 6:
146             cout << "Exiting simulation...\n";
147             break;
148
149         default:
150             cout << "Invalid choice! Please try again.\n";
151         }
152
153     } while (choice != 6);
154
155     return 0;
156 }
```

5. OUTPUT IMAGES

5.1 ENQUEUE

```
PS D:\My codes> cd "d:\My codes\" ; if ($?) { g++ cccproject1.cpp -o cccproject1 } ; if ($?) { .\cccproject1 }
====== TICKET COUNTER QUEUE SIMULATION ======
Enter maximum number of customers that can wait in queue: 5

------------- MENU -------------
1. New customer arrives (Enqueue)
2. Serve next customer (Dequeue)
3. Show all waiting customers
4. Show front & rear customer
5. Show queue size
6. Exit
--------------------------------
Enter your choice: 1
Customer with token 1 joined the queue.

Current Queue (front -> rear):
1
Front index: 0, Rear index: 0

------------- MENU -------------
1. New customer arrives (Enqueue)
2. Serve next customer (Dequeue)
3. Show all waiting customers
4. Show front & rear customer
5. Show queue size
6. Exit
--------------------------------
Enter your choice: 1
Customer with token 2 joined the queue.

Current Queue (front -> rear):
1  2
Front index: 0, Rear index: 1

------------- MENU -------------
1. New customer arrives (Enqueue)
2. Serve next customer (Dequeue)
3. Show all waiting customers
4. Show front & rear customer
5. Show queue size
6. Exit
--------------------------------
Enter your choice: 1
Customer with token 3 joined the queue.

Current Queue (front -> rear):
1  2  3
Front index: 0, Rear index: 2
```

```
------------- MENU -------------
1. New customer arrives (Enqueue)
2. Serve next customer (Dequeue)
3. Show all waiting customers
4. Show front & rear customer
5. Show queue size
6. Exit
--------------------------------
Enter your choice: 1
Customer with token 4 joined the queue.

Current Queue (front -> rear):
1  2  3  4
Front index: 0, Rear index: 3

------------- MENU -------------
1. New customer arrives (Enqueue)
2. Serve next customer (Dequeue)
3. Show all waiting customers
4. Show front & rear customer
5. Show queue size
6. Exit
--------------------------------
Enter your choice: 1
Customer with token 5 joined the queue.

Current Queue (front -> rear):
1  2  3  4  5
Front index: 0, Rear index: 4

------------- MENU -------------
1. New customer arrives (Enqueue)
2. Serve next customer (Dequeue)
3. Show all waiting customers
4. Show front & rear customer
5. Show queue size
6. Exit
--------------------------------
Enter your choice: 1

Queue Overflow! No more customers can wait.

Current Queue (front -> rear):
1  2  3  4  5
Front index: 0, Rear index: 4
```

5.2 DEQUEUE

```
PS D:\My codes> cd "d:\My codes\" ; if ($?) { g++ cccproject1.cpp -o cccproject1 } ; if ($?) { .\cccproject1 }

------------- MENU -------------
1. New customer arrives (Enqueue)
2. Serve next customer (Dequeue)
3. Show all waiting customers
4. Show front & rear customer
5. Show queue size
6. Exit
--------------------------------
Enter your choice: 2
Customer with token 1 has been served.

Current Queue (front -> rear):
2  3  4  5
Front index: 1, Rear index: 4
```

5.3 SHOW WAITING QUEUE

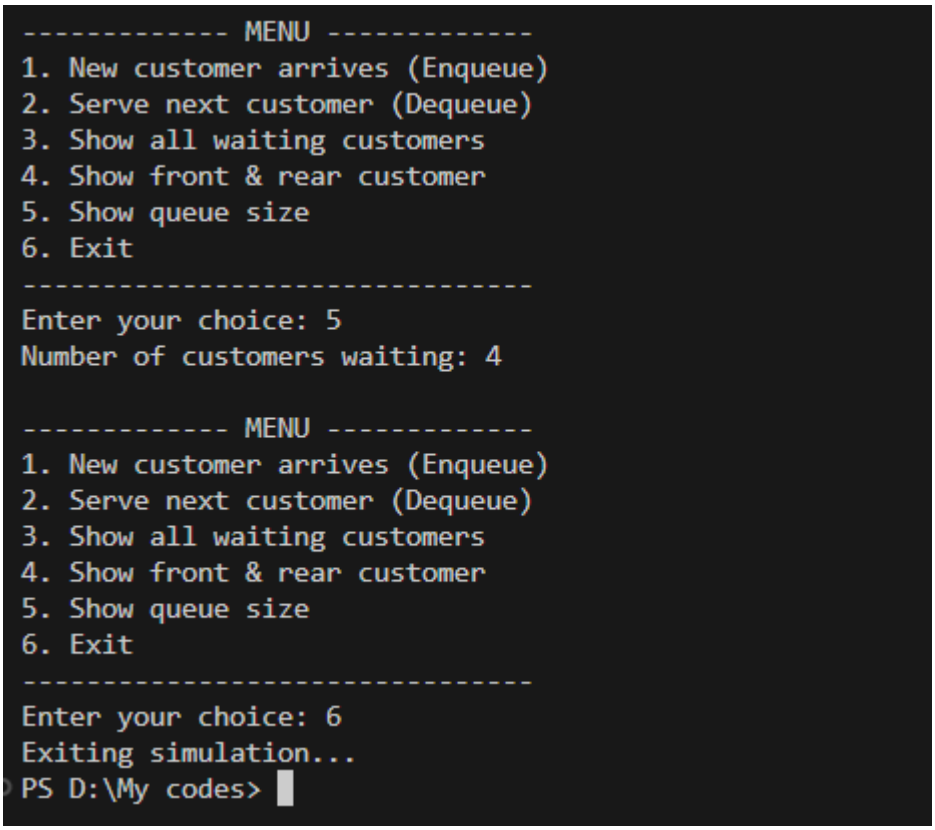
```
PS D:\My codes> cd "d:\My codes\" ; if ($?) { g++ cccproject1.cpp -o cccproject1 } ; if ($?) { .\cccproject1 }
------------- MENU -------------
1. New customer arrives (Enqueue)
2. Serve next customer (Dequeue)
3. Show all waiting customers
4. Show front & rear customer
5. Show queue size
6. Exit
--------------------------------
Enter your choice: 3

Current Queue (front -> rear):
2  3  4  5
Front index: 1, Rear index: 4
```

5.4 SHOW FRONT & REAR CUSTOMER

```
PS D:\My codes> cd "d:\My codes\" ; if ($?) { g++ cccproject1.cpp -o cccproject1 } ; if ($?) { .\cccproject1 }
------------- MENU -------------
1. New customer arrives (Enqueue)
2. Serve next customer (Dequeue)
3. Show all waiting customers
4. Show front & rear customer
5. Show queue size
6. Exit
--------------------------------
Enter your choice: 4
Front customer token: 2
Rear  customer token: 5
```

5.5 SHOW QUEUE SIZE



6. Results

- Enqueue and dequeue operations worked correctly across multiple test runs.
- The front and rear pointers correctly demonstrated circular behaviour.
- Overflow and underflow conditions were handled accurately.
- The queue displayed the correct order of customers (FIFO).
- The program proved efficient, reliable, and easy to use.

---

7. Conclusion and Recommendations

The Queue Simulation project successfully demonstrates the internal working of Circular Queue using C++. It clearly visualizes real-life queue operations and reinforces understanding of FIFO logic, circular pointer movement, and array-based memory management.

The modular approach used in the project makes the code organized and easy to extend. Future enhancements may include adding priority queues or graphical visualization.

---

8. References

- SRM University AP Course Materials – Data Structures in C++
- cppreference.com – C++ Standard Library Documentation