PIMPRI CHINCHWAD EDUCATION TRUST's.

PIMPRI CHINCHWAD COLLEGE OF ENGINEERING

(An Autonomous Institute)

Class : SY BTech **Acad. Yr. 2025-26** **Semester : I**

Name of the student: Om Jitendra Khalane **PRN :** 124B1B040

Department: Computer Engineering **Division :** A

Course Name : Data Structures and Laboratory

Course Code: BCE23PC02

Completion Date : 12/08/2025

Problem Statement:

Design a **Ticketing System** using a **Circular Singly Linked List (CSLL)**. The system should allow customers to raise support tickets that are added to a queue.Support staff can dequeue and resolve tickets in order, while urgent tickets should be added at the front of the queue. Implement enqueue, urgent enqueue, dequeue, and display operations using a circular linked list structure.

Source Code :

https://github.com/omkhalane/DSAL-SY-PCCOE/blob/main/lab_assignments/assignment08.cpp

```cpp
#include <bits/stdc++.h>
 using namespace std;

 class Qqueue
 {
 public:
     int ID;
     string customerName;
     Qqueue *next;
```

```
    Qqueue(string name, int id)
    {
        customerName = name;
        ID = id;
        next = NULL;
    }
};

class TicketingSystem
{
    Qqueue *front;
    Qqueue *rear;

public:
    TicketingSystem()
    {
        front = rear = NULL;
    }

    void enqueue(string name, int id)
    {
        Qqueue *newNode = new Qqueue(name, id);

        if (front == NULL)
        {
            front = rear = newNode;
            rear->next = front;
            return;
        }

        rear->next = newNode;
```

```
        rear = newNode;
        rear->next = front;
    }

    // Urgent Enqueue
    void urgentEnqueue(string name, int id)
    {
        Qqueue *newNode = new Qqueue(name, id);

        if (front == NULL)
        {
            front = rear = newNode;
            rear->next = front;
            return;
        }

        newNode->next = front;
        front = newNode;
        rear->next = front;
    }

    void dequeue()
    {
        if (front == NULL)
        {
            cout << "No tickets to resolve. ; Queue is empty \n";
            return;
        }

        Qqueue *temp = front;

        if (front == rear)
```

```
            {
                cout << "Resolving ticket: " << temp->ID << " " <<
temp->customerName << endl;
                delete temp;
                front = rear = NULL;
                return;
            }

            cout << "Resolving ticket: " << temp->ID << " " <<
temp->customerName << endl;

            front = front->next;
            rear->next = front;
            delete temp;
        }

        void display()
        {
            if (front == NULL)
            {
                cout << "Queue is empty\n";
                return;
            }

            Qqueue *temp = front;
            cout << "\n Tickets in Queue:\n";
            do
            {
                cout << "Ticket ID: " << temp->ID << " || Customer Name: " <<
temp->customerName << endl;
                temp = temp->next;
            } while (temp != front);
            cout << endl;
```

```
    }
};

int main()
{
    int GlobalID = 1;
    TicketingSystem ts;

    while (true)
    {
        int choice;
        cout << "\n1. Add ticket"
            << "\n2. Add urgent ticket"
            << "\n3. Resolve ticket"
            << "\n4. Display tickets"
            << "\n5. Exit\n"
            << "Enter choice: ";
        cin >> choice;

        switch (choice)
        {
        case 1:
        {
            string name;
            cout << "Enter Name: ";
            cin.ignore();
            getline(cin, name);
            ts.enqueue(name, GlobalID++);
            break;
        }
        case 2:
        {
```

```
                string name;
                cout << "Enter Name: ";
                cin.ignore();
                getline(cin, name);
                ts.urgentEnqueue(name, GlobalID++);
                break;
            }
            case 3:
                ts.dequeue();
                break;
            case 4:
                ts.display();
                break;
            case 5:
                return 0;
            default:
                cout << "Enter valid choice!\n";
            }
        }
    }
```

Conclusion:

- A **Circular Singly Linked List (CSLL)** connects the last node back to the first node, forming a circular structure.
- It efficiently models continuous systems like queues in real-world applications (e.g., ticket booking, call centers).
- The **Ticketing System** simulates a queue where:
    - **Normal tickets** are added at the rear using `enqueue()`.

- **Urgent tickets** are inserted at the front using `urgentEnqueue()`.
- **Resolved tickets** are removed from the front using `dequeue()`.
- The **display()** function traverses and displays all tickets circularly.

Time Complexity (TC):

- **Enqueue:** O(1)
- **Urgent Enqueue:** O(1)
- **Dequeue:** O(1)
- **Display:** O(n)

Space Complexity (SC):

- **O(n)** — Proportional to the number of tickets (nodes) in the queue.