Develop the Customer Support Ticket System
Theme: Stacks, Queues, and Linked Lists

GitHub:

https://github.com/mayank24000/University_Assignments/tree/main/Ds_%20Assignments/Lab_Assignments/Lab_Assignment_2

Code:

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

// Simple Customer Support Ticket System
// Made using Linked List, Stack, Priority Queue and Circular Queue
// Written in a beginner, student style

// --- Ticket Structure ---
struct Ticket {
    int id;
    string name;
    string issue;
    int priority;
    Ticket* next;

    Ticket(int i, string n, string is, int p) {
```

```
        id = i;
        name = n;
        issue = is;
        priority = p;
        next = NULL;
    }
};

// --- Linked List for Tickets ---
class TicketList {
public:
    Ticket* head;
    TicketList() {
        head = NULL;
    }

    void addTicket(int id, string name, string issue, int pr) {
        Ticket* t = new Ticket(id, name, issue, pr);
        if (head == NULL) {
            head = t;
            return;
        }
        Ticket* temp = head;
        while (temp->next != NULL)
            temp = temp->next;
        temp->next = t;
    }

    bool deleteTicket(int id) {
```

```
    if (head == NULL) return false;
    if (head->id == id) {
        Ticket* del = head;
        head = head->next;
        delete del;
        return true;
    }
    Ticket* prev = head;
    Ticket* curr = head->next;
    while (curr != NULL) {
        if (curr->id == id) {
            prev->next = curr->next;
            delete curr;
            return true;
        }
        prev = curr;
        curr = curr->next;
    }
    return false;
}

Ticket* getTicket(int id) {
    Ticket* t = head;
    while (t != NULL) {
        if (t->id == id)
            return t;
        t = t->next;
    }
    return NULL;
```

```
    }

    void showAll() {
        if (head == NULL) {
            cout << "No tickets to show.\n";
            return;
        }
        Ticket* t = head;
        cout << "Tickets List:\n";
        while (t != NULL) {
            cout << "ID: " << t->id << " | Name: " << t->name << " |
Priority: " << t->priority << "\n";
            cout << "Issue: " << t->issue << "\n";
            t = t->next;
        }
    }
};

// --- Stack for Undo Operation ---
struct Action {
    string type;
    int id;
    string name;
    string issue;
    int priority;
};

class UndoStack {
    vector<Action> st;
```

```
public:
    void push(Action a) { st.push_back(a); }
    bool empty() { return st.empty(); }
    Action top() { return st.back(); }
    void pop() { st.pop_back(); }
};

// --- Priority Queue ---
struct PQItem {
    int id, priority;
    string name, issue;
};

struct Compare {
    bool operator()(PQItem a, PQItem b) {
        return a.priority > b.priority; // lower number = higher
priority
    }
};

// --- Circular Queue ---
class CircularQueue {
    vector<int> arr;
    int front, rear, size, cap;
public:
    CircularQueue(int c = 10) {
        arr.resize(c);
        front = 0;
        rear = -1;
```

```
    size = 0;
    cap = c;
}
bool isEmpty() { return size == 0; }
bool isFull() { return size == cap; }

void enqueue(int x) {
    if (isFull()) {
        cout << "Queue is full.\n";
        return;
    }
    rear = (rear + 1) % cap;
    arr[rear] = x;
    size++;
}

bool dequeue(int &x) {
    if (isEmpty()) return false;
    x = arr[front];
    front = (front + 1) % cap;
    size--;
    return true;
}

void show() {
    if (isEmpty()) {
        cout << "No IDs in round robin.\n";
        return;
    }
```

```
        cout << "Round robin IDs: ";
        for (int i = 0, j = front; i < size; i++, j = (j + 1) % cap)
            cout << arr[j] << " ";
        cout << "\n";
    }
};

// --- Polynomial Linked List for Billing ---
struct Term {
    int coeff, pow;
    Term* next;
    Term(int c, int p) { coeff = c; pow = p; next = NULL; }
};

class Polynomial {
public:
    Term* head;
    Polynomial() { head = NULL; }

    void insert(int c, int p) {
        Term* t = new Term(c, p);
        if (head == NULL || head->pow < p) {
            t->next = head;
            head = t;
            return;
        }
        Term* temp = head;
        while (temp->next != NULL && temp->next->pow >= p)
            temp = temp->next;
```

```
        t->next = temp->next;
        temp->next = t;
    }

    bool same(Polynomial &b) {
        Term* a1 = head;
        Term* a2 = b.head;
        while (a1 && a2) {
            if (a1->coeff != a2->coeff || a1->pow != a2->pow)
                return false;
            a1 = a1->next;
            a2 = a2->next;
        }
        return (a1 == NULL && a2 == NULL);
    }

    void print() {
        if (head == NULL) { cout << "0\n"; return; }
        Term* t = head;
        while (t != NULL) {
            cout << t->coeff << "x^" << t->pow;
            if (t->next != NULL) cout << " + ";
            t = t->next;
        }
        cout << "\n";
    }
};

// --- Main System Class ---
```

```
class SupportSystem {
public:
    TicketList list;
    UndoStack undo;
    priority_queue<PQItem, vector<PQItem>, Compare> pq;
    CircularQueue cq;
    Polynomial b1, b2;
    int nextId;

    SupportSystem() {
        nextId = 1;
        cq = CircularQueue(15);
    }

    void addTicket(string n, string i, int p) {
        list.addTicket(nextId, n, i, p);
        Action a = {"ADD", nextId, n, i, p};
        undo.push(a);
        pq.push({nextId, p, n, i});
        cq.enqueue(nextId);
        cout << "Ticket created with ID " << nextId << "\n";
        nextId++;
    }

    void undoLast() {
        if (undo.empty()) {
            cout << "Nothing to undo.\n";
            return;
        }
```

```
    Action a = undo.top();
    undo.pop();
    if (a.type == "ADD") {
        list.deleteTicket(a.id);
        cout << "Undo: Ticket " << a.id << " removed.\n";
    }
}

void processPriority() {
    if (pq.empty()) {
        cout << "No urgent tickets.\n";
        return;
    }
    cout << "Processing urgent tickets:\n";
    while (!pq.empty()) {
        PQItem x = pq.top();
        pq.pop();
        cout << "ID " << x.id << " (" << x.name << ") done.\n";
        list.deleteTicket(x.id);
    }
}

void processRoundRobin(int count) {
    cout << "Round robin start:\n";
    for (int k = 0; k < count; k++) {
        int id;
        if (!cq.dequeue(id)) break;
        Ticket* t = list.getTicket(id);
        if (t != NULL)
```

```
                cout << "Cycle " << (k + 1) << ": Working on ID " <<
id << " - " << t->name << "\n";
        }
    }

    void compareBilling() {
        cout << "Billing Record 1: "; b1.print();
        cout << "Billing Record 2: "; b2.print();
        if (b1.same(b2)) cout << "Records are same.\n";
        else cout << "Records are different.\n";
    }
};

// --- Driver ---
int main() {
    SupportSystem sys;
    sys.addTicket("Diana", "Login not working", 1);
    sys.addTicket("Ethan", "App crashes", 2);
    sys.addTicket("Frank", "Payment issue", 3);

    cout << "\nAll Tickets:\n";
    sys.list.showAll();

    cout << "\nUndo last ticket:\n";
    sys.undoLast();

    cout << "\nRemaining Tickets:\n";
    sys.list.showAll();
```

```
    cout << "\nPriority processing:\n";
    sys.processPriority();

    cout << "\nRound robin demo:\n";
    sys.cq.show();
    sys.processRoundRobin(2);

    // Billing records check
    sys.b1.insert(40, 2);
    sys.b1.insert(10, 1);
    sys.b2.insert(40, 2);
    sys.b2.insert(10, 1);
    cout << "\nBilling comparison:\n";
    sys.compareBilling();

    return 0;
}
```

OUTPUT:

```
Ticket created with ID 1
Ticket created with ID 2
Ticket created with ID 3

All Tickets:
Tickets List:
ID: 1 | Name: Diana | Priority: 1
Issue: Login not working
ID: 2 | Name: Ethan | Priority: 2
Issue: App crashes
ID: 3 | Name: Frank | Priority: 3
Issue: Payment issue

Undo last ticket:
Undo: Ticket 3 removed.

Remaining Tickets:
Tickets List:
ID: 1 | Name: Diana | Priority: 1
Issue: Login not working
ID: 2 | Name: Ethan | Priority: 2
Issue: App crashes

Priority processing:
Processing urgent tickets:
ID 1 (Diana) done.
ID 2 (Ethan) done.
ID 3 (Frank) done.

Round robin demo:
Round robin IDs: 1 2 3
Round robin start:

Billing comparison:
Billing Record 1: 40x^2 + 10x^1
Billing Record 2: 40x^2 + 10x^1
Records are same.
```