Develop the Hospital Patient Record Management System
Theme: Linear Data Structures

GitHub:

https://github.com/mayank24000/University_Assignments/tree/main/Ds_%20Assignments/Theory_Assignments/Theory_Assignment_2

Code:

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

using namespace std;


// Beginner-level version by another student - Hospital Management System

// Uses: Linked List, Stack, Priority Queue, Circular Queue, Polynomial and Postfix Evaluation


struct PatientNode {

    int id;

    string name, admitDate, treatment;

    PatientNode* next;

    PatientNode(int pid, string pname, string pdate, string ptreat) {

        id = pid;

        name = pname;

        admitDate = pdate;
```

```
        treatment = ptreat;
        next = nullptr;
    }
};

class PatientRecords {
public:
    PatientNode* head;
    PatientRecords() { head = nullptr; }

    void addPatient(int id, string name, string date, string treat)
{
        PatientNode* newNode = new PatientNode(id, name, date,
treat);
        if (head == nullptr) {
            head = newNode;
            return;
        }
        PatientNode* temp = head;
        while (temp->next != nullptr) temp = temp->next;
        temp->next = newNode;
    }

    bool removePatient(int id) {
        if (!head) return false;
        if (head->id == id) {
            PatientNode* temp = head;
            head = head->next;
            delete temp;
```

```
            return true;
        }
        PatientNode* prev = head;
        PatientNode* cur = head->next;
        while (cur) {
            if (cur->id == id) {
                prev->next = cur->next;
                delete cur;
                return true;
            }
            prev = cur;
            cur = cur->next;
        }
        return false;
    }

    void showPatients() {
        if (!head) {
            cout << "No patients in record.\n";
            return;
        }
        PatientNode* t = head;
        while (t) {
            cout << "ID: " << t->id << " | Name: " << t->name << " |
Date: " << t->admitDate << "\n";
            cout << "Treatment: " << t->treatment << "\n";
            t = t->next;
        }
    }
```

```
};

struct Action {
    string type;
    int id;
    string name, date, treat;
};

class UndoHandler {
    stack<Action> undoStack;
public:
    void recordAdd(int id, string name, string date, string treat) {
        undoStack.push({"ADD", id, name, date, treat});
    }

    void undoLast(PatientRecords &records) {
        if (undoStack.empty()) {
            cout << "Nothing to undo.\n";
            return;
        }
        Action top = undoStack.top();
        undoStack.pop();
        if (top.type == "ADD") {
            if (records.removePatient(top.id))
                cout << "Undo successful: removed patient ID " <<
top.id << "\n";
            else
                cout << "Undo failed: patient not found.\n";
        }
```

```
    }
};

struct EmergencyPatient {
    int id, priority;
    string name;
    string condition;
};

struct ComparePriority {
    bool operator()(EmergencyPatient const &a, EmergencyPatient
const &b) const {
        return a.priority > b.priority;
    }
};

class RoundRobinQueue {
    vector<int> q;
    int front, rear, size, capacity;
public:
    RoundRobinQueue(int cap = 10) {
        capacity = cap;
        q.resize(cap);
        front = 0; rear = -1; size = 0;
    }
    bool enqueue(int id) {
        if (size == capacity) return false;
        rear = (rear + 1) % capacity;
        q[rear] = id;
```

```
        size++;
        return true;
    }
    bool dequeue(int &id) {
        if (size == 0) return false;
        id = q[front];
        front = (front + 1) % capacity;
        size--;
        return true;
    }
    void printQueue() {
        if (size == 0) {
            cout << "Round-robin queue empty.\n";
            return;
        }
        cout << "Patients in round-robin queue: ";
        for (int i = 0, j = front; i < size; i++, j = (j + 1) %
capacity)
            cout << q[j] << " ";
        cout << "\n";
    }
};

struct TermNode {
    int coeff, pow;
    TermNode* next;
    TermNode(int c, int p) { coeff = c; pow = p; next = nullptr; }
};
```

```
class BillPoly {
public:
    TermNode* head;
    BillPoly() { head = nullptr; }
    void addTerm(int coeff, int pow) {
        TermNode* node = new TermNode(coeff, pow);
        if (!head || head->pow < pow) {
            node->next = head;
            head = node;
            return;
        }
        TermNode* temp = head;
        while (temp->next && temp->next->pow >= pow) temp = temp-
>next;
        node->next = temp->next;
        temp->next = node;
    }
    bool equals(BillPoly &b) {
        TermNode *a1 = head, *b1 = b.head;
        while (a1 && b1) {
            if (a1->coeff != b1->coeff || a1->pow != b1->pow) return
false;
            a1 = a1->next;
            b1 = b1->next;
        }
        return a1 == nullptr && b1 == nullptr;
    }
    void display() {
        if (!head) { cout << "0\n"; return; }
```

```
        TermNode* t = head;
        while (t) {
            cout << t->coeff << "x^" << t->pow;
            if (t->next) cout << " + ";
            t = t->next;
        }
        cout << "\n";
    }
};

int evalPostfix(string exp) {
    stack<int> s;
    for (int i = 0; i < exp.size(); i++) {
        if (exp[i] == ' ') continue;
        if (isdigit(exp[i])) {
            int num = 0;
            while (i < exp.size() && isdigit(exp[i])) {
                num = num * 10 + (exp[i] - '0');
                i++;
            }
            i--;
            s.push(num);
        } else {
            if (s.size() < 2) return INT_MIN;
            int val2 = s.top(); s.pop();
            int val1 = s.top(); s.pop();
            int res = 0;
            switch (exp[i]) {
                case '+': res = val1 + val2; break;
```

```
                case '-': res = val1 - val2; break;
                case '*': res = val1 * val2; break;
                case '/': res = (val2 == 0) ? INT_MIN : val1 / val2;
break;
                default: return INT_MIN;
            }
            s.push(res);
        }
    }
    return (s.size() == 1) ? s.top() : INT_MIN;
}

class HospitalManager {
public:
    PatientRecords records;
    UndoHandler undoer;
    priority_queue<EmergencyPatient, vector<EmergencyPatient>,
ComparePriority> pq;
    RoundRobinQueue rr;
    BillPoly b1, b2;
    int idCounter;

    HospitalManager() {
        idCounter = 1;
    }

    void admit(string name, string date, string treatment, int
priority = 5) {
        int id = idCounter++;
        records.addPatient(id, name, date, treatment);
```

```
        undoer.recordAdd(id, name, date, treatment);
        pq.push({id, priority, name, treatment});
        rr.enqueue(id);
        cout << "Admitted patient " << name << " with ID " << id <<
"\n";
    }

    void processPriority() {
        if (pq.empty()) {
            cout << "No emergency patients.\n";
            return;
        }
        cout << "Emergency Handling by Priority:\n";
        while (!pq.empty()) {
            auto top = pq.top(); pq.pop();
            cout << "ID " << top.id << " (" << top.name << ") -
Priority " << top.priority << "\n";
            records.removePatient(top.id);
        }
    }

    void roundRobin(int cycles) {
        cout << "Round-robin emergency simulation:\n";
        for (int i = 0; i < cycles; i++) {
            int pid;
            if (!rr.dequeue(pid)) {
                cout << "Queue empty.\n";
                return;
            }
```

```
            cout << "Cycle " << (i + 1) << ": Handling patient ID "
<< pid << "\n";
        }
    }
};

int main() {
    HospitalManager hm;
    hm.admit("Karan Singh", "02/09/2025", "Fever observation", 2);
    hm.admit("Sneha Roy", "03/09/2025", "Minor fracture", 4);
    hm.admit("Rohit Jain", "04/09/2025", "Appendix surgery", 1);

    cout << "\nAll admitted patients:\n";
    hm.records.showPatients();

    cout << "\nUndoing last admission:\n";
    hm.undoer.undoLast(hm.records);

    cout << "\nAfter undo:\n";
    hm.records.showPatients();

    cout << "\nEmergency Queue by priority:\n";
    hm.processPriority();

    cout << "\nRound robin process demo:\n";
    hm.rr.printQueue();
    hm.roundRobin(2);

    hm.b1.addTerm(100, 2);
```

```
    hm.b1.addTerm(50, 1);
    hm.b2.addTerm(100, 2);
    hm.b2.addTerm(50, 1);
    cout << "\nBilling comparison:\n";
    hm.b1.display();
    hm.b2.display();
    cout << (hm.b1.equals(hm.b2) ? "Bills are same.\n" : "Bills
differ.\n");

    string exp = "5 3 + 2 *";
    cout << "\nPostfix Expression: " << exp << "\n";
    int res = evalPostfix(exp);
    if (res == INT_MIN) cout << "Invalid Expression\n";
    else cout << "Result: " << res << "\n";

    return 0;
}
```

OUTPUT:

```
Admitted patient Karan Singh with ID 1
Admitted patient Sneha Roy with ID 2
Admitted patient Rohit Jain with ID 3

All admitted patients:
ID: 1 | Name: Karan Singh | Date: 02/09/2025
Treatment: Fever observation
ID: 2 | Name: Sneha Roy | Date: 03/09/2025
Treatment: Minor fracture
ID: 3 | Name: Rohit Jain | Date: 04/09/2025
Treatment: Appendix surgery

Undoing last admission:
Undo successful: removed patient ID 3

After undo:
ID: 1 | Name: Karan Singh | Date: 02/09/2025
Treatment: Fever observation
ID: 2 | Name: Sneha Roy | Date: 03/09/2025
Treatment: Minor fracture

Emergency Queue by priority:
Emergency Handling by Priority:
ID 3 (Rohit Jain) - Priority 1
ID 1 (Karan Singh) - Priority 2
ID 2 (Sneha Roy) - Priority 4

Round robin process demo:
Patients in round-robin queue: 1 2 3
Round-robin emergency simulation:
Cycle 1: Handling patient ID 1
Cycle 2: Handling patient ID 2

Billing comparison:
100x^2 + 50x^1
100x^2 + 50x^1
Bills are same.
```