Billing System: #include <iostream> #include <vector> #include <fstream> using namespace std;

```
class Item {
public:
   int id;
   string name;
   double price;
   int stock;
   Item(int id, string name, double price, int stock)
      : id(id), name(name), price(price), stock(stock) {}
};
class Bill {
public:
   int billID;
   string customerName;
   double total = 0;
   vector<string> details;
   Bill(int id, string name) : billID(id), customerName(name) {}
   void addItem(Item &item, int qty) {
      if (qty <= 0) {
         throw runtime_error("Quantity must be greater than 0.");
      }
      if (qty > item.stock) {
         throw runtime_error("Not enough stock.");
      }
      item.stock -= qty;
      double itemTotal = item.price * qty;
      total += itemTotal;
      details.push_back(item.name + " x " + to_string(qty) + " = Rs." + to_string(itemTotal));
   }
   void display() {
      cout << "\n--- Bill #" << billID << " ---\n";
      cout << "Customer: " << customerName << "\n";
      for (auto &d : details) {
         cout << d << endl;
      }      cout << "Total: Rs." << total << "\n";
   }
   void saveToFile() {
      ofstream fout("customer_bills.txt", ios::app);
      fout << "Bill #" << billID << " | Customer: " << customerName << " | Total: Rs." << total <<
"\n";
      fout.close();
   }
};
void viewPreviousBills() {
   ifstream fin("customer_bills.txt");
   if (!fin) {
      cout << "No previous bills found.\n";
  return;
```

```
    }
    string line;
    cout << "\n--- Previous Bills ---\n";
    while (getline(fin, line)) {
        cout << line << endl;
    }
    fin.close();
} int main() {
    try {
        viewPreviousBills();
        vector<Item> inventory = {
            {1, "Pen", 20, 10},
            {2, "Pencil", 10, 5},
            {3, "Notebook", 50, 3}
        };
        cout << "\nAvailable Items:\n";
        for (auto &item : inventory) {
      cout << item.id << ". " << item.name << " (Rs." << item.price << ", Stock: " << item.stock <<
")\n";
        }
        Bill bill(1, "Prathamesh");
        while (true) {
            int itemId, quantity;
            cout << "\nEnter item ID to buy (0 to finish): ";
            cin >> itemId;
if (itemId == 0) break;
            cout << "Enter quantity: ";
            cin >> quantity;
            bool found = false;
            for (auto &item : inventory) {
                if (item.id == itemId) {
                    bill.addItem(item, quantity);
                    found = true;
                    break;
                }
            } if (!found) {
                cout << "Item not found. Please try again.\n";
            }
        } if (bill.total == 0) {
            cout << "No items were purchased.\n";
        } else {
            bill.display();
            bill.saveToFile();
            cout << "Bill saved to file.\n";
        }
    } catch (exception &e) {
        cout << "Error: " << e.what() << endl;
    } return 0; }
```

Vehicle Management System:

```
#include <iostream> #include <string>#include <vector> #include <iomanip>using namespace std;
class Vehicle
{
protected:
   int vehicleNo; string vehicleName;float rentalRate;    bool available;
public:
   Vehicle(int id, const string &br, float rate)
      : vehicleNo(id), vehicleName(br), rentalRate(rate), available(true) {}
   virtual ~Vehicle() {}
   virtual void display() const
   {
      cout << "Vehicle Registration Number:" << vehicleNo << endl;
      cout << "Make and Model:" << vehicleName << endl;
      cout << "Rent Per Day: RS" << rentalRate << endl;
      cout << "Availability:" << available << endl;
   }
   virtual float calculateRentalCost(int days) const
   {
      return rentalRate * days;
   }
   bool getAvailability() const
   { return available;
   }
   int getID() const{
      return vehicleNo;
   }
   void rentVehicle()
   {if (available)
      { available = false;
         cout << "Vehicle rented successfully" << endl;
      }
      else  {
         cout << "Vehicle not available" << endl;
      }
   }
   void returnVehicle(){
      available = true;
      cout << "Vehicle returned successfully\n";
   }
};
class Car : public Vehicle{
public:
   Car(int id, const string &br, float rate)
      : Vehicle(id, br, rate) {}
   void display() const override
   {
      cout << "Car Details:" << endl;
```

```
        Vehicle::display();
    }
};
class Bike : public Vehicle
{
public:
    Bike(int id, const string &br, float rate)
        : Vehicle(id, br, rate) {}
    void display() const override
    {
        cout << "Bike Details:" << endl;
        Vehicle::display();
    }
};
int main()
{
    const int MAX_VEHICLES = 10;
    Vehicle *vehicle[MAX_VEHICLES];
 vehicle[0] = new Car(6400, "Hyundai-CRETA", 4000.0);
 vehicle[9] = new Bike(2700, "Kawasaki-Z900", 3000.0);
 cout << "Vehicle Management System:\n";
 for (int i = 0; i < MAX_VEHICLES; i++)
    { vehicle[i]->display();
        cout << "=====================\n";
    } int choice, days;
    cout << "Enter Vehicle Number to rent: ";
    cin >> choice;
 bool found = false;
    for (int i = 0; i < MAX_VEHICLES; i++)
    {
        if (vehicle[i]->getAvailability() && vehicle[i]->getID() == choice)
        {
            cout << "Enter number of days to rent: ";
            cin >> days;
            vehicle[i]->rentVehicle();
            cout << "Total Rent for " << days << " days is: Rs " << vehicle[i]->calculateRentalCost(days) <<
endl;
            found = true;
            break;
        }
    } if (!found)
    {
        cout << "Vehicle not available" << endl;
    }  for (int i = 0; i < MAX_VEHICLES; i++)
    {  delete vehicle[i];
    }
    return 0;
}
```

University Log:#include <iostream> <string> <vector> <fstream> <stdexcept> <memory>

```
using namespace std;
class person { //Abstract Class
   string name;
public:
   virtual void displayDetails(ofstream &out) = 0;
   virtual void getRole(ofstream &out) = 0;
   virtual void displayOnConsole() = 0;
   void setName(string n) {
      if (n.empty()) throw invalid_argument("Name cannot be empty!");
      name = n;
   }
   string getName() { return name; }
   virtual ~person() {}//virtual destructor
};
class student : public person { // Student Clas
   int RollNo;
   string enrolled_courses;
   int marks;
public:
   student(int Roll, string En, int m, string N) {
      if (Roll <= 0) throw invalid_argument("Roll number must be positive!");
      if (m < 0 || m > 100) throw out_of_range("Marks must be between 0 and 100!");
      if (En.empty()) throw invalid_argument("Course cannot be empty!");
      RollNo = Roll;
      enrolled_courses = En;
      marks = m;
      setName(N);
   } void displayDetails(ofstream &out) override {
      out << "Role: Student\n";
      out << "Roll No : " << RollNo << endl;
      out << "Name : " << getName() << endl;
      out << "Enrolled courses : " << enrolled_courses << endl;
      out << "Marks : " << marks << endl;
   } void getRole(ofstream &out) override { out << "Role: Student\n"; }
   void displayOnConsole() override {
      cout << "Role: Student" << endl;
      cout << "Roll No : " << RollNo << endl;
      cout << "Name : " << getName() << endl;
      cout << "Enrolled courses : " << enrolled_courses << endl;
      cout << "Marks : " << marks << endl;
   }
};class Faculty : public person {
   int employee_ID;
   string subjects;
public:
   Faculty(int id, string s, string n) {
      if (id <= 0) throw invalid_argument("Employee ID must be positive!");
```

```
        if (s.empty()) throw invalid_argument("Subject cannot be empty!");
        employee_ID = id;
        subjects = s;
        setName(n);
    }   void displayDetails(ofstream &out) override {
        out << "Role: Faculty\n";
        out << "Employee ID : " << employee_ID << endl;
        out << "Name : " << getName() << endl;
        out << "Subjects : " << subjects << endl;
    } void getRole(ofstream &out) override { out << "Role: Faculty\n"; }
    void displayOnConsole() override {
        cout << "Role: Faculty" << endl;
        cout << "Employee ID : " << employee_ID << endl;
        cout << "Name : " << getName() << endl;
        cout << "Subjects : " << subjects << endl;
    }
};
int main() {
    try {
        ofstream registerBook("register.txt", ios::app);
        if (!registerBook) throw runtime_error("Error opening register.txt file!");
        vector<unique_ptr<person>> registerList;
        int choice;
        do {
            cout << "\n===== Register Menu =====\n";
            cout << "1. Add Student\n";
            cout << "2. Add Faculty\n";
            cout << "3. View All Records (from memory)\n";
            cout << "4. Exit\n";
            cout << "Enter your choice: ";
            cin >> choice;
            try {
                if (choice == 1) {
                    int roll, marks;
                    string name, course;
                    cout << "Enter Roll No: ";
                    cin >> roll;
                    cin.ignore();
                    cout << "Enter Name: ";
                    getline(cin, name);
                    cout << "Enter Enrolled Course: ";
                    getline(cin, course);
                    cout << "Enter Marks: ";
                    cin >> marks;
                    auto s = make_unique<student>(roll, course, marks, name);
                    s->displayDetails(registerBook);
                    registerBook << "------------------" << endl;
                    registerList.push_back(move(s));
                } else if (choice == 2) {
                    int empId;
                    string name, subject;
                    cout << "Enter Employee ID: ";
                    cin >> empId;
                    cin.ignore();
                    cout << "Enter Name: ";
                    getline(cin, name);
                    cout << "Enter Subject: ";
                    getline(cin, subject);
                    auto f = make_unique<Faculty>(empId, subject, name);
                    f->displayDetails(registerBook);
                    registerBook << "------------------" << endl;
                    registerList.push_back(move(f));
                } else if (choice == 3) {
                    cout << "\n===== Records in Memory =====\n";
                    if (registerList.empty()) {
                      cout << "No records available yet.\n";
                    } else {
                      for (auto &p : registerList) {
                        p->displayOnConsole();
                        cout << "------------------\n";
                      }
                    }
                } else if (choice == 4) {
                    cout << "Exiting... Data saved in register.txt ✅" << endl;
                } else {
                    cout << "Invalid choice! Try again.\n";
                }
            } catch (exception &e) {
                cerr << "Error: " << e.what() << endl;
            }
        } while (choice != 4);
        registerBook.close();
    } catch (exception &e) {
        cerr << "Fatal Error: " << e.what() << endl;
    }
    return 0;
}
```