|  |  |
| --- | --- |
|  | **Practical** |
| **Program:2.1** | A local construction company frequently deals with rectangular plots and structures of varying dimensions. To streamline their planning and estimation processes, the company requires a simple system to manage and analyze rectangular shapes efficiently. The system must be able to handle multiple rectangles, each with distinct dimensions. For each rectangle, the length and width need to be defined and stored securely. Additionally, the company needs the ability to calculate two key metrics for any given rectangle:  The area, to estimate usable space or material coverage. The perimeter, to estimate boundary lengths or material requirements for edges. To make this system functional, there should be a way to define or update the dimensions of any rectangle as required. The system should be capable of creating and managing multiple rectangle records, performing calculations for each, and displaying the results clearly for analysis and planning purposes. |
| **Code** | #include<iostream> using namespace std;    class Rectangle { private:  float length, width;    public:  void setDimensions(float l, float w) { length = l; width = w;  }    float calculateArea() { return length \* width;  }    float calculatePerimeter() { return 2 \* (length + width);  } |
|  | void displayDetails() { cout << "Length: " << length << ", Width: " << width << endl; cout << "Area: " << calculateArea() << ", Perimeter: " << calculatePerimeter() << endl;  }  };    int main() {  Rectangle rect1; rect1.setDimensions(10, 5); rect1.displayDetails();    cout << "Name: Heet Parikh" << endl << "ID: 24CS058" << endl; return 0;  } |
| **Output** |  |

|  |  |
| --- | --- |
|  | **Practical** |
| **Program:2.2** | A university managing academic data for its students. The administration seeks to digitize the records of student performance, including personal and academic details. The system must store each student's roll number, name, and marks in three subjects. Additionally, it should provide functionalities to calculate and display the average marks for performance analysis. The institution's IT team proposed a solution where student records could be initialized in two ways: either with default values for testing purposes or by accepting specific input details for each student. Once initialized, the system would allow for viewing comprehensive student details, including their roll number, name, marks, and calculated average. This functionality was designed to help faculty and administrators track individual student performance efficiently. To simulate real-world usage, the team decided to create multiple student records. They planned to populate the system with a mix of students initialized using both default and specific values. The system's ability to accurately calculate averages and display detailed student information was to be tested with this data. |
| **Code** | #include<iostream> using namespace std;    class Student { private:  int rollNumber; string name; int marks[3];    public:  void setDetails(int r, string n, int m1, int m2, int m3) { rollNumber = r; name = n; marks[0] = m1; marks[1] = m2; marks[2] = m3;  } |
|  | float calculateAverage() { return (marks[0] + marks[1] + marks[2]) / 3.0;  }    void displayDetails() { cout << "Roll No: " << rollNumber << ", Name: " << name << endl; cout << "Marks: " << marks[0] << ", " << marks[1] << ", " << marks[2]  << endl; cout << "Average Marks: " << calculateAverage() << endl;  }  };    int main() { Student s1; s1.setDetails(101, "Chota Bheem", 85, 90, 80); s1.displayDetails();    cout << "Name: PREET" << endl << "ID: 24CE091" << endl; return 0;  } |
| **Output** |  |

|  |  |
| --- | --- |
|  | **Practical** |
| **Program:2.3** | In a growing city, a newly established bank sought to streamline its operations by implementing a digital system to manage customer accounts. The management envisioned a system where each account would store details about the account holder, their unique account number, and the current balance. To ensure flexibility and accuracy, they required  functionalities for creating accounts with varying initial balances, depositing money, withdrawing funds (with checks for sufficient balance), and generating detailed account summaries. The bank's IT team faced the challenge of developing a robust solution. They began by sketching out the essential features of the system. The team highlighted that new accounts could be created in two ways: one with no initial balance, and another with specified account details and a starting amount. Additionally, they recognized the need for reliable mechanisms to handle deposits and withdrawals, ensuring proper validation to prevent overdrafts. Displaying account details in a clear format was also prioritized for customer communication. To ensure scalability, the team decided to simulate the system by creating multiple accounts using the proposed methods. They tested various scenarios, such as depositing and withdrawing different amounts, handling edge cases like insufficient funds, and verifying that the account summaries were accurate. This iterative approach helped them refine the system and ensure its readiness for deployment. |
| **Code** | #include <iostream> #include <string> using namespace std;    class Account { private:  int accNo; string name; |

|  |  |
| --- | --- |
|  | float bal;    public:  // Default Constructor Account() { accNo = 0; name = "N/A"; bal = 0.0;  }    // Parameterized Constructor with Initial Balance Account(int a, string n, float b) { accNo = a; name = n; bal = b;  }    // Create Account with No Initial Balance void createAcc(int a, string n) { accNo = a; name = n; bal = 0.0;  }    // Create Account with Initial Balance void createAcc(int a, string n, float b) { accNo = a; name = n; bal = b;  }    // Deposit Money void deposit(float amt) { if (amt > 0) { bal += amt; cout << "Amount deposited successfully.\n";  } else { cout << "Invalid amount!\n";  }  }    // Withdraw Money void withdraw(float amt) { if (amt > 0 && amt <= bal) { |
|  | bal -= amt; cout << "Amount withdrawn successfully.\n";  } else if (amt > bal) { cout << "Insufficient balance!\n";  } else { cout << "Invalid amount!\n";  }  }    // Display Account Details void showDetails() { cout << "\nAccount Number: " << accNo << endl; cout << "Account Holder: " << name << endl; cout << "Account Balance: $" << bal << endl;  }    // Get Account Number for Search int getAccNo() { return accNo;  }  };    int main() { int n;  cout << "Enter number of accounts to create: "; cin >> n;    Account acc[100]; // Array to hold multiple accounts    // Create Multiple Accounts for (int i = 0; i < n; i++) { int a; string name; float b; int choice;    cout << "\nEnter details for account " << i + 1 << ":\n"; cout << "Account Number: "; cin >> a; cin.ignore(); // Ignore newline after cin cout << "Account Holder Name: "; getline(cin, name); cout << "Do you want to add initial balance? (1-Yes, 2-No): "; cin >> choice; |
|  | if (choice == 1) { cout << "Enter Initial Balance: "; cin >> b; acc[i].createAcc(a, name, b);  } else { acc[i].createAcc(a, name);  }  }    int choice, searchNo; do {  cout << "\nMenu:\n";  cout << "1. Display All Account Details\n2. Deposit Money\n3. Withdraw Money\n4. Search Account by Account Number\n5. Exit\n"; cout << "Enter your choice: "; cin >> choice;    switch (choice) { case 1:  for (int i = 0; i < n; i++) { acc[i].showDetails();  cout << "--------------------------\n";  } break;  case 2:  cout << "Enter Account Number to deposit money: "; cin >> searchNo; for (int i = 0; i < n; i++) { if (acc[i].getAccNo() == searchNo) { float amt; cout << "Enter amount to deposit: "; cin >> amt; acc[i].deposit(amt); break;  } } break;  case 3:  cout << "Enter Account Number to withdraw money: "; cin >> searchNo; for (int i = 0; i < n; i++) { |

|  |  |
| --- | --- |
|  | if (acc[i].getAccNo() == searchNo) { float amt; cout << "Enter amount to withdraw: "; cin >> amt; acc[i].withdraw(amt); break;  } } break;  case 4:  cout << "Enter Account Number to search: "; cin >> searchNo; for (int i = 0; i < n; i++) {  if (acc[i].getAccNo() == searchNo) { acc[i].showDetails(); break;  } } break;  case 5: cout << "Exiting the system...\n"; break;  default:  cout << "Invalid choice! Please try again.\n";  }  } while (choice != 5);    cout << "Name: PREET" << endl << "ID: 24CE091" << endl; return 0;  } |
| **Output** |  |

|  |  |
| --- | --- |
|  | **Practical** |
| **Program:2.4** | A mid-sized retail store faced challenges in efficiently managing its inventory of items. The store's management sought to build a system that could keep track of individual items, including details like a unique item ID, item name, price, and the quantity available in stock. The need for a streamlined process arose due to frequent stock discrepancies, which led to issues with customer satisfaction and operational efficiency. To address this, the store hired a team of developers to create a digital inventory management system. The envisioned solution would allow for the initialization of item details, either with default values or specific attributes like ID, name, price, and starting quantity. This system needed to handle operations like increasing stock levels when new shipments arrived and decreasing stock when items were sold, ensuring sufficient inventory was available for each transaction. Additionally, the system would provide clear, detailed summaries of each item's status, aiding in decision-making and reporting. The developers faced real-world scenarios where they had to manage multiple inventory items simultaneously. They planned to design an array of inventory items and simulate common tasks such as adding stock, processing sales transactions, and displaying the current inventory details. Handling edge cases, such as attempting to sell more items than available in stock, became a critical part of the implementation to ensure reliability. |
| **Code** | #include <iostream> #include <string> using namespace std;    class Item { |

|  |  |
| --- | --- |
|  | private:  int itemID; string itemName; float price; int quantity;    public:  // Default Constructor Item() { itemID = 0; itemName = "N/A"; price = 0.0; quantity = 0;  }    // Parameterized Constructor  Item(int id, string name, float p, int qty) { itemID = id; itemName = name; price = p; quantity = qty;  }    // Add stock void addStock(int qty) { quantity += qty;  }    // Sell items and reduce stock bool sellItem(int qty) { if (quantity >= qty) { quantity -= qty; return true;  } else { cout << "Insufficient stock for Item ID: " << itemID << endl; return false; |
|  | }  }    // Display item details void displayItem() { cout << "Item ID: " << itemID << endl; cout << "Item Name: " << itemName << endl; cout << "Price: $" << price << endl; cout << "Available Stock: " << quantity << endl;  }    // Get item ID int getItemID() { return itemID;  }  };    int main() { int n;  cout << "Enter the number of items: "; cin >> n;    Item inventory[100]; // Array to hold inventory items    // Initialize inventory for (int i = 0; i < n; i++) { int id, qty; string name; float price;    cout << "\nEnter details for item " << i + 1 << endl; cout << "Item ID: "; cin >> id; cin.ignore(); // Ignore newline after cin cout << "Item Name: "; getline(cin, name); |
|  | cout << "Price: "; cin >> price; cout << "Initial Quantity: "; cin >> qty;    inventory[i] = Item(id, name, price, qty);  }    int choice, itemID, qty; do {  cout << "\nMenu:\n"; cout << "1. Add Stock\n2. Sell Item\n3. Display All Items\n4. Exit\n"; cout << "Enter your choice: "; cin >> choice;    switch (choice) { case 1:  cout << "Enter Item ID to add stock: "; cin >> itemID; cout << "Enter quantity to add: "; cin >> qty;  for (int i = 0; i < n; i++) { if (inventory[i].getItemID() == itemID) { inventory[i].addStock(qty); cout << "Stock added successfully!\n"; break;  } } break;  case 2:  cout << "Enter Item ID to sell: "; cin >> itemID; cout << "Enter quantity to sell: "; |

|  |  |
| --- | --- |
|  | cin >> qty;  for (int i = 0; i < n; i++) { if (inventory[i].getItemID() == itemID) { inventory[i].sellItem(qty); break;  } } break;  case 3:  cout << "\nDisplaying All Items:\n"; for (int i = 0; i < n; i++) { inventory[i].displayItem(); cout << "--------------------------\n";  } break;  case 4: cout << "Exiting the system...\n"; break;  default:  cout << "Invalid choice! Please try again.\n";  }  } while (choice != 4);    cout << "Name: PREET" << endl << "ID:24CE091" << endl; return 0;  } |
| **Output** |  |

|  |  |
| --- | --- |
|  | **Practical** |
| **Program:2.5** | A regional banking institution sought to improve its loan management process by developing a system that could efficiently handle loan details for applicants. The system was expected to streamline the calculation of monthly EMIs (Equated Monthly Instalments) and provide detailed loan summaries for customers. This initiative aimed to enhance customer experience by offering accurate and transparent information about their loans.  To meet these requirements, the bank's technology team was tasked with designing a solution. They envisioned a system where each loan would be uniquely identified by a loan ID, and additional details such as the applicant's name, total loan amount, annual interest rate, and loan tenure in months would be stored. The team also emphasized the importance of accurately calculating the EMI using a standard formula based on the loan amount, interest rate, and tenure. The formula incorporated compound interest principles to determine the fixed monthly payment for the loan term.  EMI= 𝑃 ∗ 𝑅 ∗ (1 + 𝑅)^𝑇/((1+𝑅)^𝑇) −1  As part of the implementation, the developers planned to initialize loans either with default values for testing purposes or with actual customer data provided at the time of application. The system needed to include a feature to display comprehensive loan details, including the calculated EMI, in a customer- friendly format. This functionality was intended to aid both customers and bank staff in managing loan-related queries effectively. |
| **Code** | #include <iostream>  #include <cmath>  #include <string> |

|  |  |
| --- | --- |
|  | using namespace std;    class Loan { private:  int id;  string name; float p; // Principal/Loan Amount float r; // Annual Interest Rate int t; // Loan Tenure in Months float emi;    public:  // Default Constructor Loan() { id = 0; name = "N/A"; p = 0.0; r = 0.0; t = 0; emi = 0.0;  }    // Parameterized Constructor  Loan(int i, string n, float amount, float rate, int tenure) { id = i; name = n; p = amount; r = rate; t = tenure; calcEMI(); // Calculate EMI after initialization  }    // Calculate EMI using formula: EMI = P \* R \* (1 + R)^T / ((1 + R)^T -  1) void calcEMI() { float monthlyRate = r / (12 \* 100); // Monthly interest rate |
|  | float num = p \* monthlyRate \* pow(1 + monthlyRate, t); float den = pow(1 + monthlyRate, t) - 1; if (den != 0) { emi = num / den;  } else { emi = 0.0;  }  }    // Display loan details void showDetails() { cout << "Loan ID: " << id << endl; cout << "Applicant Name: " << name << endl; cout << "Loan Amount: $" << p << endl; cout << "Annual Interest Rate: " << r << "%" << endl; cout << "Loan Tenure: " << t << " months" << endl; cout << "Calculated EMI: $" << emi << endl;  }    // Get Loan ID int getID() { return id;  }  };    int main() { int n; cout << "Enter number of loans to initialize: "; cin >> n;    Loan loans[100]; // Array to hold loan details    // Initialize loans for (int i = 0; i < n; i++) { int id, t;  string name; |
|  | float p, r;    cout << "\nEnter details for loan " << i + 1 << endl; cout << "Loan ID: "; cin >> id; cin.ignore(); // Ignore newline after cin cout << "Applicant Name: "; getline(cin, name); cout << "Loan Amount: "; cin >> p; cout << "Annual Interest Rate (%): "; cin >> r; cout << "Loan Tenure (months): "; cin >> t;    loans[i] = Loan(id, name, p, r, t);  }    int choice, searchID; do { cout << "\nMenu:\n";  cout << "1. Display All Loan Details\n2. Search Loan by Loan ID\n3. Exit\n"; cout << "Enter your choice: "; cin >> choice;    switch (choice) { case 1:  cout << "\nDisplaying All Loan Details:\n"; for (int i = 0; i < n; i++) { loans[i].showDetails(); cout << "--------------------------\n";  } break;  case 2: |

|  |  |
| --- | --- |
|  | cout << "Enter Loan ID to search: "; cin >> searchID;  for (int i = 0; i < n; i++) { if (loans[i].getID() == searchID) { loans[i].showDetails(); break;  } } break;  case 3: cout << "Exiting the system...\n"; break;  default:  cout << "Invalid choice! Please try again.\n";  }  } while (choice != 3);    cout << "Name: PREET" << endl << "ID: 24CE091 << endl; return 0;  } |
| **Output** |  |