|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Name:** | **:** | **Nilay Rajeshbhai Patel** | **Year** | **:** | **1** | **Semester** | **:** | **2** |
| **ID No.** | **:** | **24CE089** | **Div** | **:** | **2** | **Academic Year** | **:** | **2024-**  **2025** |
| **Subject code** | **:** | **CEUC102** | | | | |  | |
| **Subject name** | **:** | **Programming with C++** | | | | |  | |

|  |  |
| --- | --- |
|  | **Practical** |
| **Program: 1.1** | A bank wants to create a simple system to manage customer bank accounts. The system should allow customers to perform basic banking operations such as depositing money, withdrawing money, and checking their account balance.  Each bank account will need to have an account holder's name, a unique account number, and a balance.  Deposits should increase the account balance, while withdrawals should only be  allowed if there are sufficient funds in the account. If an attempt is made to withdraw more money than is available, an error message should be displayed. Customers should also have the ability to view their account balance whenever required.  The system must be designed using Object-Oriented Programming principles, focusing on creating a simple and efficient solution to manage the accounts effectively. The system should ensure that all account details are secure and accessible only through authorized methods. |
| **Code** | //This Program is prepared by 24CE089\_NILAY PATEL  #include<iostream>//Standard Input/Output library  using namespace std;  class BankAccount//Creating class  {  //Private data-members  char AccHolderName[10];  long Balance;  long AccNum;  //Making public functions for different tasks  public:  void AddAccDetails()//Function for entering details of bank account  {  cout << "Enter the account holder's name: ";  cin >> AccHolderName;//Taking input of account holder's name  cout << "Enter the account number: ";  cin >> AccNum;//Taking input of account number  Balance = 1000;  cout << "Account addition successful!" << endl;//Printing message  }  inline int GetAccNum()//Function for returning the value of account number  {  return AccNum;//Return statement  }  void DepositMoney()//Function to deposit  {  //Variable declaration  long Amount;  cout << "Enter the amount to be deposited: ";  cin >> Amount;//Taking input of amount to be deposited  Balance += Amount;//The amount to be deposit in account  cout << "Deposit successful!" << endl;//Printing message  }  void WithdrawMoney()//Function to withdraw  {  //Variable declaration  long Amount;  cout << "Enter the amount to be withdrawn: ";  cin >> Amount;//Taking input of amount to be withdrawn    //If wanted amount of withdrawal is greater than balance....  if(Amount > Balance-1000)  {  cout << "Insufficient balance!" << endl;//Printing error message  }  else//If not....  {  Balance -= Amount;//The amount to be withdraw from account  cout << "Withdrawal successful!" << endl;//Printing message  }  }  void DisplayAccBalance()//Function for displaying balance  {  cout<< "Current balance: " << Balance << endl;//Printing current balance  }  };  int main()//Main function  {  class BankAccount Acc[100];//Creating object  long AccFind;//Variable for matching account number  int Choice;//Variable for choice  int NumAcc = 0;//Variable for tracking the no. of account & initializing to 0  //Printing menu  cout << "Enter 1 to add account details." << endl;  cout << "Enter 2 to deposit." << endl;  cout << "Enter 3 to withdraw." << endl;  cout << "Enter 4 to display current balance." << endl;  cout << "Else to Exit." << endl;  cout << "===================================" << endl;  while(true)//Loop for doing various tasks  {  cout << "Enter the option which you want: ";  cin >> Choice;//Taking input of choice  bool Found = false;//Variable for signal if account no. matched  switch (Choice)//Switch case  {  case 1://Case for adding account details  {  Acc[NumAcc].AddAccDetails();//Calling AddAccDetails function  NumAcc++;//Increment in NumAcc Variable  break;//Break statement  }  case 2://Case for depositing money  {  cout << "Enter the account number: ";  cin >> AccFind;//Taking input of Account no. to search    //Loop for matching account no. with entered account no.  for (int i = 0; i < NumAcc; i++)  {  //If entered account matched with account no.  if (Acc[i].GetAccNum() == AccFind)  {  Acc[i].DepositMoney();//Calling DepositMoney function  Found = true;//Account no. matched with entered account no.  break;//Break statement  }  }  if(! Found)//If not...  {  cout << "Account not found!" << endl;//Printing error message  }  break;//Break statement  }  case 3://Case for withdrawing money  {  cout << "Enter the account number: ";  cin >> AccFind;//Taking input of Account no. to search  //Loop for matching Account no. with entered account no.  for (int i = 0; i < NumAcc; i++)  {  //If entered account matched with account no.  if (Acc[i].GetAccNum() == AccFind)  {  Acc[i].WithdrawMoney();//Calling WithdrawMoney function  Found = true;//Account no. matched with entered account no.  break;//Break statement  }  }  if(! Found)//If not...  {  cout << "Account not found!" << endl;//Printing error message  }  break;//Break statement  }  case 4://Case for displaying current balance  {  cout << "Enter the account number: ";  cin >> AccFind;//Taking input of Account no. to search  //Loop for matching Account no. with entered account no.  for (int i = 0; i < NumAcc; i++)  {  //If entered account matched with account no.  if (Acc[i].GetAccNum() == AccFind)  {  Acc[i].DisplayAccBalance();//Calling DisplayAccBalance function  Found = true;//Account no. matched with entered account no.  break;//Break statement  }  }  if(! Found)//If not...  {  cout << "Account not found!" << endl;//Printing error message  }  break;//Break statement  }  default://Default Case  {  cout << "24CE089\_NILAY PATEL." << endl;//Printing exiting message  return 0;//Return statement  }  }  }  } |
| **Output** |  |

**Sign: Grade:**

|  |  |
| --- | --- |
|  | **Practical** |
| **Program: 1.2** | A small retail store is facing challenges in managing its inventory effectively. The store sells a variety of products, each identified by a unique product ID, a name, the available quantity in stock, and the price per unit. To streamline their operations, the store needs a basic system to manage this inventory efficiently.  The system must provide the ability to add new products to the inventory, ensuring that each product has its ID, name, quantity, and price properly recorded. Additionally, the system should allow the store staff to update the quantity of any existing product, such as when new stock arrives or when items are sold.  Another essential feature of the system is the calculation of the total value of all products in the inventory, which is determined by multiplying the quantity of each product by its price and summing these values for all products.  The store management is exploring two approaches for this system: a procedural approach and an object-oriented approach. The goal is to evaluate these approaches by comparing their ease of implementation, code reusability, and overall complexity.  The system's design and implementation should consider these requirements and provide an effective solution (either procedural approach or an object-oriented approach) to the store's inventory management problems. |
| **Code** | //This program is prepared by 24CE089\_NILAY PATEL  #include<iostream>//Standard Input/Output library  using namespace std;  class Product//Creating class  {  //Private data-members  int ProdID,Quantity;  float Price;  char ProdName[20];  //Making public functions for different tasks  public:  void AddProd()//Function for adding new product  {  cout << "Enter product ID: ";  cin >> ProdID;//Taking input of product ID  cout << "Enter name of product: ";  cin >> ProdName;//Taking input of product's name  cout << "Enter price of product: ";  cin >> Price;//Taking input of product's price  Quantity = 1;//Initializing product's quantity to 1  cout << "Product is added successfully!" << endl;//Printing message  }  inline int GetID()//Function for returning product ID  {  return ProdID;//Returning product ID  }  void UpdateQuan()//Function for updating quantity  {  cout << "Enter no. of quantity to be update: ";  cin >> Quantity;//Taking input of product's quantity  cout << "Quantity is updated successfully!" << endl;//Printing message  }  inline int CalculateValue()//Function for returning total price of product  {  return Quantity\*Price;//Returning product of quantity & price  }  void DisplayBill()//Function for displaying bill  {  cout << ProdName << "\t";  cout << Quantity << "\t\t";  cout << Price << "\t" << CalculateValue() << endl;  }  };  int main()//Main function  {  class Product P[100];//Creating object  int Choice;//Variable for choice  int ProdFind;//Variable for matching product ID  int ProdCount=0;//Initializing variable for no. of product to 1  //Printing menu  cout << "Enter 1 to add new product." << endl;  cout << "Enter 2 to update quantity of product." << endl;  cout << "Enter 3 to display bill & exit." << endl;  cout << "======================================" << endl;  while(true)  {  cout << "Enter the option which you want: ";  cin >> Choice;//Taking input of choice  bool Found = false;//Variable for signal if account no. matched  switch (Choice)//Switch case  {  case 1://Case for adding new product  {  P[ProdCount].AddProd();//Calling AddProd function  ProdCount++;//Increment of ProductCount  break;//Break statement  }  case 2://Case for updating product's quantity  {  int ProdFind;//Variable for matching product ID  cout << "Enter product ID: ";  cin >> ProdFind;//Taking input of product id to search  //Loop for matching product id with entered product id  for(int i=0; i < ProdCount; i++)  {  //If product id is matched with entered product id...  if(P[i].GetID() == ProdFind)  {  P[i].UpdateQuan();//Calling UpdateQuan function  Found = true;//Product ID no. matched with entered product ID  break;//Break statement  }  }  if(! Found)//If not...  {  cout << "Product not found!" << endl;//Printing error message  }  break;//Break statement  }  case 3://Case for displaying bill  {  int TotalValue=0;//Initializing TotalValue variable to 0  cout << "----------------------------------------------" << endl;  cout << "Name\t\tQuantity\tPrice\tTotal" << endl;  cout << "----------------------------------------------" << endl;  for(int i=0; i < ProdCount ; i++) //Loop for calling displayBill function  {  P[i].DisplayBill();//Calling DisplayBill function  TotalValue += P[i].CalculateValue();  }  cout << "----------------------------------------------" << endl;  cout << "Total Bill: " << TotalValue << endl;//Printing total bill  cout << "24CE089\_NILAY PATEL." << endl;//Printing exiting message  return 0;//Return statement  }  default://Default case  {  cout << "Invalid input!" << endl;//Printing error message  break;  }  }  }  } |
| **Output** |  |

**Sign: Grade:**

|  |  |
| --- | --- |
|  | **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** | //This program is prepared by 24CE089\_NILAY PATEL  #include<iostream>//Standard Input/Output library  using namespace std;  class Plot//Creating class  {  //Private data-members  int Length,Width,Area,Perimeter;  public:  //Making public functions for different tasks  void AddPlotData()//Function for entering plot's details  {  cout << "Enter length of plot: ";  cin >> Length;//Taking input of length of plot  cout << "Enter width of plot: ";  cin >> Width;//Taking input of width of plot  cout << "Plot's data is successfully added!" << endl;//Printing message  Area = Length\*Width;//Calculating area of plot  Perimeter = 2\*(Length+Width);//Calculating perimeter of plot  }  void UpdatePlotData()//Function for updating plot's data  {  cout << "Enter new length of plot: ";  cin >> Length;//Taking input of new length of plot  cout << "Enter new width of plot: ";  cin >> Width;//Taking input of new width of plot  cout << "Plot's data is successfully updated!" << endl;//Printing message  Area = Length\*Width;//Again calculating area for updated data of plot  Perimeter = 2\*(Length+Width);//Again calculating perimeter for updated data of plot  }  void DisplayAllPlotData()//Function for displaying all plot's data  {  cout << "Length: " << Length << endl;//Displaying length of plot  cout << "Width: "<< Width<< endl;//Displaying width of plot  cout << "Area: "<< Area << endl;//Displaying area of plot  cout << "Perimeter: " << Perimeter << endl;//Displaying perimeter of plot  }  void DisplayAnyPlotData()//Function for displaying any one plot's data  {  cout << "Plot details:" << endl;  cout << "Length: "<< Length << endl;//Displaying length of plot  cout << "Width: "<< Width << endl;//Displaying width of plot  cout << "Area: "<< Area << endl;//Displaying area of plot  cout << "Perimeter: " << Perimeter << endl;//Displaying perimeter of plot  }  };  int main()//Main function  {  class Plot P[100];//Creating object  int Choice,PlotFind;//Variables for Choice & variable for matching plot no.  int NumPlot = 0;//Variable for tracking the no. of plot & initializing to 0  //Displaying menu  cout << "Enter 1 to add plot data." << endl;  cout << "Enter 2 to update plot data." << endl;  cout << "Enter 3 to display plot data." << endl;  cout << "Else to exit." << endl;  cout << "=============================" << endl;  next :  cout << "Enter your choice: ";  cin >> Choice;//Taking input of choice  switch (Choice)//Switch case  {  case 1://Case for adding plot's data  {  P[NumPlot].AddPlotData();//Calling AddPlotData  NumPlot++;//Increment in variable no. of plot  break;//Break statement  }  case 2://Case for updating plot's data  {  cout << "Enter plot no.: ";  cin >> PlotFind;//Taking input of plot no. to search  if(PlotFind <= NumPlot)//If no. of plot is greater than or equal to entered plot no. ...  {  P[PlotFind-1].UpdatePlotData();//Calling UpdatePlotData function  break;//Break statement  }  else//If not...  {  cout << "No plot found!" << endl;//Printing error message  break;//Break statement  }  }  case 3://Case for displaying plot's data  {  int Choice1;//Variable for another choice  cout << "Enter 1 to display all plot data & enter 2 to display any one plot data which you want: ";  cin >> Choice1;//Taking input for another choice  switch (Choice1)//Switch case  {  case 1://Case for displaying entered all plot's data  {  //Loop for displaying entered all plot's data  for(int i = 1, PlotNum = 0; PlotNum < NumPlot; i++, PlotNum++)  {  cout << "Plot " << i << " details:" << endl;  P[PlotNum].DisplayAllPlotData();//Calling DisplayAllPlotData function  }  if(NumPlot == 0)//If no. of plot is equal to 0...  {  cout << "No plot available to display!" << endl;//Printing error message  break;//Break statement  }  break;//Break statement  }  case 2://Case for displaying any one plot's data  {  cout << "Enter plot number: ";  cin >> PlotFind;//Taking input of plot no.  if(PlotFind <= NumPlot)//If no. of plot is greater than or equal to entered plot no. ...  {  P[PlotFind-1].DisplayAnyPlotData();//Calling DisplayAnyPlotData function  break;//Break statement  }  else//If not...  {  cout << "No plot found!" << endl;//Printing error message  break;//Break statement  }  }  default://Default case  {  cout << "Invalid input!" << endl;//Printing error message  break;//Break statement  }  }  break;//Break statement  }  default://Default case  {  cout << "24CE089\_NILAY PATEL." << endl;//Printing exiting message  return 0;//Return statement  }  }  goto next;//Goto statement  } |
| **Output** |  |

**Sign: Grade:**

|  |  |
| --- | --- |
|  | **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** | //This program is prepared by 24CE089\_NILAY PATEL  #include<iostream>//Standard Input/Output library  #include<cstring>//String functions library  #include<iomanip>//Input/Output manipulation library  using namespace std;  class Student//Creating class  {  //Private data-members  int RollNum;  char Name[10];  float Marks1,Marks2,Marks3,Average;  //Private member-function  //Function for calculating average marks of 3 subs.  inline void CalculateAverage()  {  Average=(Marks1+Marks2+Marks3)/3.0;//Calculating average of 3 subs.  }  public:  //Making public functions for different tasks  Student()//Default constructor  {  RollNum = 0;//Initializing roll no. to default value 0  strcpy(Name,"N/A");//Initializing name to default value N/A  Marks1 = 0;//Initializing marks of sub.1 to default value 0  Marks2 = 0;//Initializing marks of sub.2 to default value 0  Marks3 = 0;//Initializing marks of sub.3 to default value 0  Average = 0;//Initializing average to default value 0  }  Student(int StuRollNo,char StuName[],int StuMarks1,int StuMarks2,int StuMarks3)//Parameterized constructor  {  //Setting student's details with parameters  RollNum = StuRollNo;  strcpy(Name,StuName);  Marks1 = StuMarks1;  Marks2 = StuMarks2;  Marks3 = StuMarks3;  }  void AddStuDetails()//Function for adding student's details  {  cout << "Enter roll number: ";  cin >> RollNum;//Taking input of roll no.  cout << "Enter name: ";  cin >> Name;//Taking input of name  cout << "Enter marks of subject 1: ";  cin >> Marks1;//Taking input of marks of sub.1  cout << "Enter marks of subject 2: ";  cin >> Marks2;//Taking input of marks of sub.2  cout << "Enter marks of subject 3: ";  cin >> Marks3;//Taking input of marks of sub.3  CalculateAverage();//Calling CalculateAverage function  cout << "Student's details are successfully added!" << endl;//Printing message  }  void DisplayStuDetails()//Function for displaying all plot's data  {  cout << "Student's details:" << endl;//Displaying student's details  cout << "Roll number: " << RollNum << endl;//Displaying roll no.  cout << "Name: " << Name << endl;//Displaying name  cout << "Marks of subject 1: " << Marks1 << endl;//Displaying marks of sub.1  cout << "Marks of subject 2: " << Marks2 << endl;//Displaying marks of sub.2  cout << "Marks of subject 3: " << Marks3 << endl;//Displaying marks of sub.3  //Displaying average of 3 subs.  cout << "Average: " << setprecision(4) << Average << endl;  }  inline int GetRollNo()//Function for returning value of roll no.  {  return RollNum;//Returning roll no.  }  };  int main()//Main function  {  class Student S[100];//Creating object  int NumStu = 0;//Variable for tracking the no. of students & initializing to 0  int Choice,StuFind;//Variable for Choice & variable for matching roll no.  //Displaying menu  cout << "Enter 1 to add student's details." << endl;  cout << "Enter 2 to display student's details." << endl;  cout << "Else to exit." << endl;  cout << "=====================================" << endl;  next ://Label  cout << "Enter your choice which you want: ";  cin >> Choice;//Taking input of choice  switch (Choice)//Switch case  {  case 1://Case for adding student's details  {  S[NumStu].AddStuDetails();//Calling AddStuDetails function  NumStu++;//Increment in variable no. of student  break;//Break statement  }  case 2://Case for displaying student's details  {  cout << "Enter student's roll no.: ";  cin >> StuFind;//Taking input of roll no. to search  //Boolean type variable for switching between true/false & initializing it with false  bool Found = false;  //Loop for matching roll no. with entered roll no.  for(int i = 0; i < NumStu; i++)  {  if(S[i].GetRollNo() == StuFind)//If roll no. matched with entered roll no. ...  {  S[i].DisplayStuDetails();//Calling DisplayStuDetails function  //If roll no. matched with entered roll no. then Found variable switches to true  Found = true;  break;//Break statement  }  }  if(! Found)//If Found variable doesn't switch to true...  {  S[StuFind].DisplayStuDetails();//Calling DisplayStuDetails function  }  break;//Break statement  }  default://Default case  {  cout << "24CE089\_NILAY PATEL." << endl;//Printing exiting message  return 0;//Return statement  }  }  goto next;//Goto statement  } |
| **Output** |  |

**Sign: Grade:**

|  |  |
| --- | --- |
|  | **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** | //This Program is prepared by 24CE089\_NILAY PATEL  #include<iostream>//Standard Input/Output library  using namespace std;  class BankAccount//Creating class  {  //Private data-members  char AccHolderName[10];  long Balance;  long AccNum;  //Making public functions for different tasks  public:  void AddAccDetails()//Function for entering details of bank account  {  cout << "Enter the account holder's name: ";  cin >> AccHolderName;//Taking input of account holder's name  cout << "Enter the account number: ";  cin >> AccNum;//Taking input of account number  Balance = 1000;  cout << "Account addition successful!" << endl;//Printing message  }  int GetAccNum()//Function for returning the value of account number  {  return AccNum;//Return statement  }  void DepositMoney()//Function to deposit  {  //Variable declaration  long Amount;  cout << "Enter the amount to be deposited: ";  cin >> Amount;//Taking input of amount to be deposited  Balance += Amount;//The amount to be deposit in account  cout << "Deposit successful!" << endl;//Printing message  }  void WithdrawMoney()//Function to withdraw  {  //Variable declaration  long Amount;  cout << "Enter the amount to be withdrawn: ";  cin >> Amount;//Taking input of amount to be withdrawn  //If wanted amount of withdrawal is greater than balance....  if(Amount >= Balance-1000)  {  cout << "Insufficient balance!" << endl;//Printing error message  }  else//If not....  {  Balance -= Amount;//The amount to be withdraw from account  cout << "Withdrawal successful!" << endl;//Printing message  }  }  void DisplayAccDetails()//Function for displaying account's details  {  //Printing account holder's name  cout << "Account holder's name: " << AccHolderName << endl;  cout << "Account number: " << AccNum << endl;//Printing account no.  //Printing current balance of account  cout<<"Current balance: "<< Balance <<endl;  }  };  int main()//Main function  {  class BankAccount Acc[100];//Creating object  long AccFind;//Variable for matching account number  int Choice;//Variable for choice  int NumAcc = 0;//Variable for tracking the no. of account & initializing to 0  //Printing menu  cout << "Enter 1 to add account details." << endl;  cout << "Enter 2 to deposit." << endl;  cout << "Enter 3 to withdraw." << endl;  cout << "Enter 4 to display current balance." << endl;  cout << "Else to Exit." << endl;  cout << "===================================" << endl;  while(true)//Loop for doing various tasks  {  cout << "Enter the option which you want: ";  cin >> Choice;//Taking input of choice  bool Found = false;//Variable for signal if account no. matched  switch (Choice)//Switch case  {  case 1://Case for adding account details  {  Acc[NumAcc].AddAccDetails();//Calling AddAccDetails function  NumAcc++;//Increment in NumAcc Variable  break;//Break statement  }  case 2://Case for depositing money  {  cout << "Enter the account number: ";  cin >> AccFind;//Taking input of Account no. to search  //Loop for matching Account no. with entered account no.  for (int i = 0; i < NumAcc; i++)  {  //If entered account no. matched with account no.  if (Acc[i].GetAccNum() == AccFind)  {  Acc[i].DepositMoney();//Calling DepositMoney function  Found = true;//Account no. matched with entered account no.  break;//Break statement  }  }  if(! Found)//If Found variable doesn't switch to true...  {  cout << "Account not found!" << endl;//Printing error message  }  break;//Break statement  }  case 3://Case for withdrawing money  {  cout << "Enter the account number: ";  cin >> AccFind;//Taking input of Account no. to search  //Loop for matching Account no. with entered account no.  for (int i = 0; i < NumAcc; i++)  {  //If entered account no. matched with account no.  if (Acc[i].GetAccNum() == AccFind)  {  Acc[i].WithdrawMoney();//Calling WithdrawMoney function  Found = true;//Account no. matched with entered account no.  break;//Break statement  }  }  if(! Found)//If Found variable doesn't switch to true...  {  cout << "Account not found!" << endl;//Printing error message  }  break;//Break statement  }  case 4://Case for displaying account's details  {  cout << "Enter the account number: ";  cin >> AccFind;//Taking input of Account no. to search  //Loop for matching Account no. with entered account no.  for (int i = 0; i < NumAcc; i++)  {  //If entered account no. matched with account no.  if (Acc[i].GetAccNum() == AccFind)  {  Acc[i].DisplayAccDetails();//Calling DisplayAccDetails function  Found = true;//Account no. matched with entered account no.  break;//Break statement  }  }  if(! Found)//If Found variable doesn't switch to true...  {  cout << "Account not found!" << endl;//Printing error message  }  break;//Break statement  }  default://Default Case  {  cout << "24CE089\_NILAY PATEL." << endl;//Printing exiting message  return 0;//Return statement  }  }  }  } |
| **Output** |  |

**Sign: Grade:**

|  |  |
| --- | --- |
|  | **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** | //This Program is prepared by 24CE089\_NILAY PATEL  #include<iostream>//Standard Input/Output library  using namespace std;  class Inventory//Creating class  {  //Private data-members  int ItemID;  char ItemName[20];  float Price;  int Quantity;  //Making public functions for different tasks  public:  void AddItemDetails()//Function for adding item's details  {  cout << "Enter item ID: ";  cin >> ItemID;//Taking input of item ID  cout << "Enter item's name: ";  cin >> ItemName;//Taking input of item's name  cout << "Enter price of item: ";  cin >> Price;//Taking input of price  Quantity = 1;//Initializing quantity to 1  }  int GetItemID()//Function for returning item ID  {  return ItemID;//Returning item ID  }  void IncreaseItemQuan()//Function for increasing item's quantity  {  //Variable declaration  int NewQuantity;//Variable for increasing quantity  cout << "Enter quantity to be increased: ";  cin >> NewQuantity;//Taking input of increasing quantity  Quantity += NewQuantity;//Addition of present quantity & increasing quantity  //Printing message  cout << "Item's Quantity is successfully increased!" << endl;  }  void DecreaseItemQuan()//Function for decreasing item's quantity  {  //Variable declaration  int NewQuantity;//Variable for decreasing quantity  cout << "Enter quantity to be decreased: ";  cin >> NewQuantity;//Taking input of decreasing quantity  //If present quantity is greater than decreasing quantity...  if(Quantity >= NewQuantity)  {  //Subtraction of present quantity & decreasing quantity  Quantity -= NewQuantity;  //Printing message  cout << "Item's Quantity is successfully decreased!" << endl;  }  else//If not...  {  //Printing error message  cout << "Insufficient quantity for decreament!" << endl;  }  }  void DisplayItemDetails()//Function for displaying item's details  {  cout << "Item's details:" << endl;  cout << "ID: " << ItemID << endl;//Displaying item ID  cout << "Name: " << ItemName << endl;//Displaying item's name  cout << "Price: " << Price << endl;//Displaying price  cout << "Quantity: " << Quantity << endl;//Displaying present quantity  }  };  int main()//Main function  {  class Inventory I[100];//Creating object  int NumItem = 0;//Variable for tracking no. of items & initializing to 0  int ItemFind,Choice;//Variables for matching item ID & choice  //Displaying menu  cout << "Enter 1 to add item." << endl;  cout << "Enter 2 to increase quantity." << endl;  cout << "Enter 3 to decrease quantity." << endl;  cout << "Enter 4 to display item's details." << endl;  cout << "Else to Exit." << endl;  cout << "==================================" << endl;  while(true)//Loop for doing various tasks  {  cout << "Enter the option which you want: ";  cin >> Choice;//Taking input of choice  bool Found = false;//Variable for signal if account no. matched  switch(Choice)//Switch case  {  case 1://Case for adding items  {  I[NumItem].AddItemDetails();//Calling AddItemDetails function  NumItem++;//Increment in no. of items  break;//Break statement  }  case 2://Case for increasing quantity  {  cout << "Enter item ID: ";  cin >> ItemFind;//Taking input of item ID to search  //Loop for matching item ID with entered item ID  for (int i = 0; i < NumItem; i++)  {  //If entered item ID matched with item ID...  if (I[i].GetItemID() == ItemFind)  {  I[i].IncreaseItemQuan();//Calling IncreaseItemQuan function  Found = true;//Account no. matched with entered account no.  break;//Break statement  }  }  if(! Found)//If Found variable doesn't switch to true...  {  cout << "Item not found!" << endl;//Printing error message  }  break;//Break statement  }  case 3:  {  cout << "Enter item ID: ";  cin >> ItemFind;//Taking input of item ID to search  //Loop for matching item ID with entered item ID  for (int i = 0; i < NumItem; i++)  {  //If entered item ID matched with item ID...  if (I[i].GetItemID() == ItemFind)  {  I[i].DecreaseItemQuan();//Calling DecreaseItemQuan  Found = true;//Account no. matched with entered account no.  break;//Break statement  }  }  if(! Found)//If Found variable doesn't switch to true...  {  cout << "Item not found!" << endl;//Printing error message  }  break;//Break statement  }  case 4:  {  cout << "Enter item ID: ";  cin >> ItemFind;//Taking input of item ID to search  //Loop for matching item ID with entered item ID  for (int i = 0; i < NumItem; i++)  {  //If entered item ID matched with item ID...  if (I[i].GetItemID() == ItemFind)  {  I[i].DisplayItemDetails();//Calling DisplayItemDetails  Found = true;//Account no. matched with entered account no.  break;//Break statement  }  }  if(! Found)//If Found variable doesn't switch to true...  {  cout << "Item not found!" << endl;//Printing error message  }  break;//Break statement  }  default://Default case  {  cout << "24CE089\_NILAY PATEL." << endl;//Printing exiting message  return 0;//Return statement  }  }  }  } |
| **Output** |  |

**Sign: Grade:**

|  |  |
| --- | --- |
|  | **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.    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** | //This Program is prepared by 24CE089\_NILAY PATEL  #include<iostream>//Standard Input/Output library  #include<math.h>//Standard math library  using namespace std;  class LoanAcc//Creating class  {  //Private data-members  int LoanID;  long LoanAmount;  char AppName[20];  float InterestRate,LoanTenure;  float EMI;  inline void CalculateEMI()//Function calculating EMI  {  InterestRate = InterestRate/12/100;//Calculating Interest rate    //Calculating EMI  EMI = (LoanAmount\*InterestRate\*pow((1+InterestRate),LoanTenure))/(pow((1+InterestR ate),LoanTenure)-1);  }  //Making public functions for different tasks  public:  void AddAccData()//Function for adding account data  {  cout << "Enter loan ID: ";  cin >> LoanID;//Taking input of loan ID  cout << "Enter applicant's name: ";  cin >> AppName;//Taking input of applicant's name  cout << "Enter total loan amount: ";  cin >> LoanAmount;//Taking input of loan amount  cout << "Enter annual interest rate: ";  cin >> InterestRate;//Taking input of interest rate  cout << "Enter loan tenure in months : ";  cin >> LoanTenure;//Taking input of loan period in months  CalculateEMI();//Calling CalculateEMI function internally  }  void DisplayAccData()//Function for displaying account data  {  InterestRate = InterestRate\*12\*100;//Calculating interest rate  cout << "Account's details:" << endl;  //Displaying applicant's name  cout << "Applicant's name: " << AppName << endl;  //Displaying loan amount  cout << "Total loan amount: " << LoanAmount << endl;  //Displaying interest rate  cout << "Annual interest rate: " << InterestRate << "%" << endl;  //Displaying loan period in months  cout << "Loan tenure in months : " << LoanTenure << endl;  cout << "Total EMI of loan amount: " << EMI << endl;//Displaying Total EMI  }  inline int GetAccNum()//Function for returning account no.  {  return LoanID;//Returning loan ID  }  };  int main()//Main function  {  class LoanAcc L[100];//Creating object  //Variable for tracking no.of account will be created & initializing to 0  int NumAcc = 0;  int AccFind,Choice;//Variables for matching account no. & choice  //Displaying menu  cout << "Enter 1 to add loan account." << endl;  cout << "Enter 2 to display account data." << endl;  cout << "Else to Exit." << endl;  cout << "================================"<<endl;  while(true)//Loop for doing various tasks  {  cout << "Enter the option which you want: ";  cin >> Choice;//Taking input of choice  switch(Choice)//Switch case  {  case 1://Case for adding account data  {  L[NumAcc].AddAccData();//Calling AddAccData function  NumAcc++;//Increment in no. of accounts  break;//Break statement  }  case 2://Case for displaying account data  {  cout << "Enter Loan ID: ";  cin >> AccFind;//Taking input of account no. to search  bool Found = false;//Variable for signal if account no. matched  //Loop for matching item ID with entered loan ID  for (int i = 0; i < NumAcc; i++)  {  //If entered loan ID matched with loan ID...  if (L[i].GetAccNum() == AccFind)  {  L[i].DisplayAccData();//Calling DisplayAccData function  Found = true;//Account no. matched with entered loan ID no.  break;//Break statement  }  }  if(! Found)//If Found variable doesn't switch to true...  {  cout << "Account not found!" << endl;//Printing error message  }  break;//Break statement  }  default://Default case  {  cout << "24CE089\_NILAY PATEL";//Printing exiting message  return 0;//Return statement  }  }  }  } |
| **Output** |  |

**Sign: Grade:**

|  |  |
| --- | --- |
|  | **Practical** |
| **Program: 3.1** | A growing organization sought to enhance its payroll process to improve accuracy and efficiency in calculating employee salaries. The goal was to manage details such as the employee's name, basic salary, and bonus amount. While most employees would receive a default bonus, there was a need for flexibility to provide customized bonuses for exceptional performers or those in specialized roles.  To ensure real-time computation of each employee's total salary, the system would dynamically calculate the total salary by adding the basic salary and the bonus. An inline function was chosen for this task, allowing the system to compute the total salary instantly during processing. The system was designed to initialize employee records using a constructor, where the bonus could either be set to a default value or adjusted according to specific employee criteria.  As the organization expanded, managing employee records efficiently became a key consideration. The solution involved storing employee details dynamically, ensuring the system could handle a growing number of records while maintaining scalability and flexibility. Each employee record would include their personal details, basic salary, and bonus information, with the system displaying a breakdown of each employee’s details along with their total salary. |
| **Code** | //This program is prepared by 24CE089\_NILAY PATEL  #include<iostream>//Standard Input/Output library  using namespace std;  class Employee//Creating class  {  //Private data-members  char EmployeeName[10];  int BasicSalary,BonusAmount,TotalSalary;  //Private members-functions  inline void AddDefaultBonus()//Function for adding default bonus  {  BonusAmount = BasicSalary\*0.1;//Default bonus amount  TotalSalary = BasicSalary + BonusAmount;//Total salary  }  inline void AddAdditionalBonus()//Function for adding additional bonus  {  cout << "Enter additional bonus: ";  cin >> BonusAmount;//Taking input of additional bonus  TotalSalary = BasicSalary + (BasicSalary\*0.1) + BonusAmount;//Total salary  }  public:  //Making public functions for different tasks  void AddEmployeeData()//Function for entering data of employee  {  int Choice1;//Variable for another choice  cout << "Enter employee's name: ";  cin >> EmployeeName;//Taking input of employee's name  cout << "Enter employee's basic salary: ";  cin >> BasicSalary;//Taking input of basic salary  cout << "Enter 0 to add default bonus or 1 to add additional bonus: ";  cin >> Choice1;//Taking input of another choice  if(Choice1 == 0)//If entered choice is 0...  {  AddDefaultBonus();//Calling AddDefaultBonus function  }  else if(Choice1 == 1)//If entered choice is 1...  {  AddAdditionalBonus();//Calling AddAdditionalBonus function  }  else//If not...  {  cout << "Invalid input!" << endl;//Printing error message  }  cout << "Employee's Data is successfully added!" << endl;//Printing message  }  void DisplayEmployeeData()//Function for displaying employee's data  {  cout << "Employee details:" << endl;  cout << "Name: " << EmployeeName << endl;//Displaying name  cout << "Basic salary: " << BasicSalary << endl;//Displaying basic salary  //Displaying bonus amount  cout << "Bonus amount: " << BonusAmount << endl;  cout << "Total salary: " << TotalSalary << endl;//Displaying total salary  }  };  int main()//Main function  {  class Employee E[100];//Creating object  //Variable for tracking no. of employee & initializing to zero    int NumEmployee = 0;  int Choice;//Variable for choice  int EmployeeFind;//Variable for matching employee ID  //Displaying menu  cout << "Enter 1 to add employee's data." << endl;  cout << "Enter 2 to display employee's data." << endl;  cout << "Else to exit." << endl;  cout << "===================================" << endl;  next ://Label  cout << "Enter your choice which you want: ";  cin >> Choice;//Taking input of another choice  switch (Choice)//Switch case  {  case 1://Case for entering data of employee  {  //Calling AddEmployeeData function  E[NumEmployee].AddEmployeeData();    NumEmployee++;//Increment in no. of employee    break;//Break statement  }  case 2://Case for displaying employee's data  {  cout << "Enter employee's ID: ";  cin >> EmployeeFind;//Taking input of employee's ID to search  //If no. of employee is greater than entered employee's ID ...  if(EmployeeFind <= NumEmployee)  {  //Calling DisplayEmployeeData function  E[EmployeeFind-1].DisplayEmployeeData();    break;//Break statement  }  else//If not...  {  cout << "Employee not found!" << endl;//Printing error message    break;//Break statement  }  break;//Break statement  }  default://Default case  {  cout << "24CE089\_NILAY PATEL." << endl;//Printing exiting message    return 0;//Return statement  }  }  goto next;//Goto statement  } |
| **Output** |  |

**Sign: Grade:**

|  |  |
| --- | --- |
|  | **Practical** |
| **Program: 3.2** | A software development company was tasked with conducting a performance analysis of recursive algorithms versus their iterative counterparts. The specific focus was on calculating the sum of integers in an array, where the array's size and elements were to be provided dynamically by the user. To facilitate memory management and enable dynamic resizing of the array, the team decided to use a flexible container for storing the array elements.  The system was designed to first prompt users for the array's size and then request the input of individual elements. A recursive function was to be implemented to compute the sum by dividing the problem into smaller sub-problems, recursively summing subsets of the array until reaching the base  case. In addition to the recursive implementation, an iterative version of the function would be created for comparison.  The main objective of the study was to assess and compare the computational performance and implementation complexity of both recursive and non-recursive approaches. By evaluating execution times, memory usage, and code complexity, the team hoped to gain insights into the trade-offs between recursion and iteration, particularly in terms of efficiency and applicability to real-world problems. |
| **Code** | //This program is prepared by 24CE089\_NILAY PATEL  #include<iostream>//Standard input/output library  using namespace std;  int SumRecursive(int Array[],int n)//Function for recursive sum  {  if(n == 0)//If array size equals to 0  {  return 0;//Return statement  }  //Returning addition of entered two numbers  return Array[n-1] + SumRecursive(Array,n-1);  }  int SumIterative(int Array[],int n)//Function for recursive sum  {  int Total = 0;//Variable for total & initializing to to 0  for(int i = 0; i < n; ++i)//Loop for addition of entered numbers  {  Total += Array[i];//Adding entered numbers to total  }  return Total;//Returning total  }  int main()//Main function  {  int n;//Variable for array size  cout << "Enter the size of the array: ";  cin >> n;//Taking input of array size  int\* Array = new int[n];//Initializing new size to Array variable  for (int i = 0; i < n; ++i)//Loop for taking input of Array elements  {  cout << "Enter element " << i+1 << ": ";  cin >> Array[i];//Taking input of Array elements  }  //Variable for recursive sum & calling SumRecursive function  int RecursiveSum = SumRecursive(Array,n);    //Printing recursive sum  cout << "Sum using recursion: " << RecursiveSum << endl;  //Variable for iterative sum & calling SumIterative function  int IterativeSum = SumIterative(Array,n);    cout << "Sum using iteration: " << IterativeSum << endl;//Printing iterative sum  delete[] Array;//Releasing size of Array variable  cout << "24CE089\_NILAY PATEL." << endl;//Printing exiting message  return 0;//Return statement  } |
| **Output** |  |

**Sign: Grade:**

|  |  |
| --- | --- |
|  | **Practical** |
| **Program: 3.3** | A community bank sought to enhance its account management system with a digital solution to improve efficiency and provide better customer service. The system was required to manage the essential details of each account, including the account number, account holder’s name, and balance. Additionally, the bank wanted to provide a secure mechanism for transferring money between accounts, allowing customers to easily manage their funds.  The bank also needed a way to track the total number of accounts created, which would be important for generating reports and understanding the growth of their customer base. This feature was aimed at helping the bank maintain an overview of their account portfolio and analyse trends over time.  To ensure smooth and reliable operations, the system was designed to store account information in a way that would allow easy access and updates. When new accounts were created, they would be added to the system dynamically. The management team planned for future scalability and performance improvements by considering more efficient storage and retrieval methods after the initial system was built, ensuring that the bank could easily accommodate more accounts and transactions as the customer base grew. |
| **Code** | //This Program is prepared by 24CE089\_NILAY PATEL  #include<iostream>//Standard Input/Output library  using namespace std;  class BankAccount//Creating class  {  //Private data-members  char AccHolderName[20];  long long Balance;  long AccNum;  //Making public functions for different tasks  public:  void AddAccDetails()//Function for entering details of bank account  {  cout << "Enter the account number: ";  cin >> AccNum;//Taking input of account number  cout << "Enter the account holder's name: ";  cin >> AccHolderName;//Taking input of account holder's name  cout << "Enter the account balance: ";//Taking input of balance  cin >> Balance;//Taking input of balance  cout << "Account addition successful!" << endl;//Printing message  }  void DisplayAccDetails()//Function for displaying account's details  {  cout << "Account's details:" << endl;  cout << "Account no. : " << AccNum << endl;//Printing account no.  //Printing account holder's name  cout << "Account holder's name: " << AccHolderName << endl;  //Printing current balance of account  cout << "Current balance: "<< Balance << endl;  }  inline long GetAccNum()//Function for returning the value of account number  {  return AccNum;//Return statement  }  //Function for returning the value of account number  inline long long GetBalance()  {  return Balance;//Return statement  }  inline void WithdrawMoney(long long Amount)//Function to withdraw  {  Balance -= Amount;//The amount to be withdrew from account  }  inline void DepositMoney(long long Amount)//Function to deposit  {  Balance += Amount;//The amount to be deposited in account  }  };  //Function for finding account  int FindAccount(long AccNum,BankAccount Acc[], int NumAcc)  {  //Loop for matching Account no. with entered account no.  for (int i = 0; i < NumAcc; i++)  {  //If entered account matched with account no.  if (Acc[i].GetAccNum() == AccNum)  {  return i;//Returning index of matched account  }  }  return -1;//Returning false as 1  }  int main()//Main function  {  //Creating object  class BankAccount Acc[100];  int NumAcc = 0;//Variable for tracking the no. of account & initializing to 0  int Choice;//Variable for choice  long long Amount;//Variable for amount  //Printing menu  cout << "Enter 1 to add account details." << endl;  cout << "Enter 2 to transfer money." << endl;  cout << "Enter 3 to display no. of accounts." << endl;  cout << "Else to exit." << endl;  cout << "===================================" << endl;  while (true)//Loop for doing various tasks  {  cout << "Enter the option which you want: ";  cin >> Choice;//Taking input of choice  switch (Choice)//Switch case  {  case 1://Case for adding account details  {  Acc[NumAcc].AddAccDetails();//Calling AddAccDetails function  NumAcc++;//Increment in NumAcc Variable  break;//Break statement  }  case 2://Case for transferring money  {  long AccFindW,AccFindD;//Variable for matching account number  //Variables for taking values from FindAccount function  int ResultD,ResultW;  cout << "Enter the account no. for withdrawal: ";  cin >> AccFindW;//Taking input of Account no. to search  //Initializing ResultW with returned from FindAccount function  //Calling FindAccount function  ResultW = FindAccount(AccFindW,Acc,NumAcc);  if(ResultW == -1)//If returned value is -1  {  cout << "Account not found!" << endl;//Printing error message  break;//Break statement  }  cout << "Enter the account no. for deposit: ";  cin >> AccFindD;//Taking input of Account no. to search  //Initializing ResultD with returned from FindAccount function  //Calling FindAccount function  ResultD = FindAccount(AccFindD,Acc,NumAcc);  if(ResultD == -1)//If returned value is -1  {  cout << "Account not found!" << endl;//Printing error mesasage  break;//Break statement  }  cout << "Enter the amount be transfered: ";  cin >> Amount;//Taking input of amount  //If entered amount is greater that account balance  if(Amount > Acc[AccFindW].GetBalance())  {  cout << "Insufficient balance!" << endl;//Printing error message  break;//Break statement  }  Acc[ResultW].WithdrawMoney(Amount);//Calling WithdrawMoney  Acc[ResultD].DepositMoney(Amount);//Calling DepositMoney  cout << "Amount is successfully transfered!" << endl;//Printing message  cout << "Transaction summary:" << endl;  Acc[ResultW].DisplayAccDetails();//Printing withdrawal account's details  Acc[ResultD].DisplayAccDetails();//Printing deposit account's details  break;//Break statement  }  case 3://Case for displaying no. of accounts  {  //Printing no. of accounts  cout << "The no. of account created: " << NumAcc << endl;  break;//Break statement  }  default://Default Case  {  cout << "24CE089\_NILAY PATEL." << endl;//Printing exiting message  return 0;//Return statement  }  }  }  } |
| **Output** |  |

**Sign: Grade:**

|  |  |
| --- | --- |
|  | **Practical** |
| **Program: 3.4** | A technology firm aimed to develop a flexible and reusable solution for managing collections of various data types, including integers, floating-point numbers, and characters. The system was intended to perform fundamental operations on these collections, such as finding the maximum value, reversing the collection, and displaying all elements. To achieve versatility and avoid redundancy in code, the solution was designed to use function templates, allowing the same logic to be applied seamlessly to different data types.  The team recognized the importance of using dynamic arrays to store the collections, enabling efficient management of varying collection sizes. The design emphasized scalability and flexibility, ensuring that the system could handle different data types and their associated operations with minimal changes to the core logic.  In practice, the system allowed for the creation of collections for various data types, such as integers, floating-point numbers, and characters. The operations on these collections included determining the maximum value, reversing the order of elements, and printing the collection contents. |
| **Code** | //This program is prepared by 24CE089\_NILAY PATEL  #include<iostream>//Standard Input/Output library  using namespace std;  template<typename T>//Template declaration  void InputData(T\* Data,int Size)//Function for entering data elements  {  for (int i = 0; i < Size; ++i)//Loop for taking data elements  {  cout << "Enter element " << i+1 << ": ";  cin >> Data[i];//Taking input of data elements  }  }  template<typename T>//Template declaration  //Function for returning/handling entered data elements  T GetElements(int Size,T\* Data)  {  int Count;//Number of elements  //If Count variable is greater or equal to 0 & less than size of data elements...  if(Count >= 0 && Count < Size)  {  return Data[Count];//Returning data elements  }  }  template<typename T>//Template declaration  //Function for finding max value from set of data elements  T FindMax(int Size,T\* Data)  {  T MaxValue = Data[0];//Initializing first data entry to max value  for(int i = 1; i < Size; ++i)//Loop for searching max value  {  if(Data[i] > MaxValue)//If any data entry is greater than max value  {  MaxValue = Data[i];//Giving value to max value  }  }  return MaxValue;//Returning max value  }  template<typename T>//Template declaration  void ReverseData(int Size,T\* Data)//Function for reversing data elements  {  //Loop for replacing data elements with each-other  for(int Start = 0,End = Size-1; Start < End ; ++Start, --End)  {  //Replacing data elements with each-other  T Temp = Data[Start];  Data[Start] = Data[End];  Data[End] = Temp;  }  }  template<typename T>//Template declaration  void DisplayData(int Size,T\* Data)//Function for displaying data elements  {  for (int i = 0; i < Size; ++i)//Loop for displaying all data elements  {  cout << Data[i] << " ";//Printing data elements  }  cout << endl;//End statement  }  template<typename T>//Template declaration  //Function for performing all operations/calling function  void PerformOperations(int Size)  {  T\* Data;//Pointer variable declaration  Data = new T[Size];//Dynamic allocation by new  InputData(Data,Size);//Calling InputData function  cout << "Entered data: ";  DisplayData(Size,Data);//Calling DisplayData function  T MaxValue = FindMax(Size,Data);//Assigning value, returned from function  //Printing max value from set of data elements  cout << "Max Value: " << MaxValue << endl;  cout << "Reversed Data: ";  ReverseData(Size,Data);//Calling ReverseData function  DisplayData(Size,Data);//Calling DisplayData function  delete[] Data;//Free up Data named array of pointer  }  int main()//Main function  {  int Size;//Variable for no. of data elements  cout << "Enter the no. of elements: ";  cin >> Size;//Taking input of no. of data elements  char DataType;//Character variable for data-type  cout << "Enter the data type (i = int, f = float, c = char): ";  Next ://Label  cin >> DataType;//Taking input for data-type  if (DataType == 'i')//If entered character in i...  {  //Calling PerformOperations function for int value  PerformOperations<int>(Size);  }  else if (DataType == 'f')//If entered character in f...  {  //Calling PerformOperations function for float value  PerformOperations<float>(Size);  }  else if (DataType == 'c')//If entered character in c...  {  //Calling PerformOperations function for character value  PerformOperations<char>(Size);  }  else//If not...  {  cout << "Invalid data type! re-enter data type: ";//Printing error message  goto Next;//Goto statement  }  cout << "24CE089\_NILAY PATEL." << endl;//Exiting message  return 0;//Return statement  } |
| **Output** |  |

**Sign: Grade:**

|  |  |
| --- | --- |
|  | **Practical** |
| **Program: 3.5** | A data analytics company was tasked with developing a unique digital signature system based on the concept of "super digits." The system required finding a single-digit representation of a given number through recursive digit summation. The algorithm was defined as follows:  If the number has only one digit, it is its super digit. Otherwise, the super digit is the super digit of the sum of its digits, repeated recursively until a single digit is obtained.  The challenge involved an additional complexity—constructing the number by concatenating a given string representation of an integer multiple times. For example, if the number n was represented as a string and concatenated k times, the super digit of the resulting number needed to be calculated.  For instance:  Given n = 9875 and k = 4, the number is represented as 9875987598759875.  The sum of digits in this number is calculated recursively until a single digit remains:  9 + 8 + 7 + 5 + 9 + 8 + 7 + 5 + 9 + 8 + 7 + 5 + 9 + 8 + 7 + 5 = 116  1 + 1 + 6 = 8  The super digit is 8.  The system was required to handle large numbers efficiently by leveraging mathematical insights rather than explicitly constructing large concatenated strings. This case study called for implementing a recursive solution to calculate the super digit, supported by a mathematical approach to optimize the handling of repeated sums. |
| **Code** | //This program is prepared by 24CE089\_NILAY PATEL  #include<iostream>//Standard Input/Output library  using namespace std;  int CalculateSumOfDigits(int Number)//Function for calculating sum of digits  {  int Sum = 0;//Variable for sum & initializing with 0  while(Number > 0)//If no. is greater than 0...  {  Sum += Number % 10;//Calculating sum  Number /= 10;//Calculating no.  }  return Sum;//Returning sum  }  int CalculateSuperDigit(int Number)//Function for calculating super digit  {  if(Number < 10)//If number is less than 10  {  return Number;//Returning no.  }  int Sum = 0;//Variable for sum & initializing with 0  while(Number > 0)//If no. is greater than 0...  {  Sum += Number % 10;//Calculating sum  Number /= 10;//Calculating no.  }  //Returning/Recursing CalculateSuperDigit function  return CalculateSuperDigit(Sum);  }  int main()//Main function  {  int Num,NumRepeat;//Variables for no. & number of repetitions(k)  cout << "Enter number(n): ";  cin >> Num;//Taking input no.  cout << "Enter the number of repetitions(k): ";  cin >> NumRepeat;//Taking input of number of repetitions(k)  //Variable for sum  //Initializing with returned value from CalculateSumOfDigits function  int Sum = CalculateSumOfDigits(Num);  //Variable for total sum  int TotalSum = Sum\*NumRepeat;//Calculating total sum  //Variable for super digit  //Initializing with returned value from CalculateSuperDigit function  int SuperDigit = CalculateSuperDigit(TotalSum);  cout << "Super digit: " << SuperDigit << endl;//Printing super digit  cout << "24CE089\_NILAY PATEL." << endl;//Exiting message  return 0;//Return statement  } |
| **Output** |  |

**Sign: Grade:**

|  |  |
| --- | --- |
|  | **Practical** |
| **Program: 4.1** | A team of engineers was tasked with developing a program to calculate and manage the areas of multiple circles for a design project. To achieve this, they devised a solution using a structured, object-oriented approach. At the foundation of their solution was a base class that represented a generic shape, responsible for storing and managing the radius of the circle. Building upon this, a specialized class for circles was created to extend functionality by introducing a method for calculating the area of a circle based on its radius. Using this framework, the team designed a system to handle multiple circles, ensuring that the process of storing, calculating, and displaying the areas was efficient and adaptable. They explored two different approaches for managing the collection of circles—one focusing on flexibility and dynamic handling, while the other used a more static structure. By implementing and comparing these methods, the engineers gained insights into the benefits of using efficient techniques for organizing and processing geometric data, enhancing their problem-solving capabilities. |
| **Code** | //This program is prepared by 24CE089\_NILAY PATEL.  #include<iostream>//Standard input/output library  using namespace std;  class Shape//Creating class  {  //Private data-members  float Radius;  //Making public functions for different tasks  public:  void InputRadius()//Function for taking input of radius  {  cout << "Enter radius: ";  cin >> Radius;//Taking input of radius  }  int GetRadius()//Function for returning radius  {  return Radius;//Returning radius  }  };  class Circle : private Shape//Creating class with a parent class  {  //Private data-members  float Area;  //Making public functions for different tasks  public:  void CalculateArea()//Function for calculating area  {  InputRadius();//Calling InputRadius internally  Area = 3.14\*GetRadius()\*GetRadius();//Calculating area  }  void DisplayArea()//Function for displaying area  {  cout << "Area of circle: " << Area << endl;//Printing area  }  };  int main()//Main function  {  class Circle C;//Creating object  //C.InputRadius();  C.CalculateArea();//Calling CalculateArea function  C.DisplayArea();//Calling DisplayArea function  cout << "24CE089\_NILAY PATEL." << endl;  return 0;//Return statement  } |
| **Output** |  |

**Sign: Grade:**

|  |  |
| --- | --- |
|  | **Practical** |
| **Program: 4.2** | A growing organization wanted to develop a system to manage its hierarchy and represent its structure in a programmatic way. To achieve this, a multilevel approach was designed, reflecting the natural progression of roles within the organization. At the foundation, a class was created to represent a person, capturing the basic details such as name and age. Building on this, an intermediate level was introduced to represent employees, adding a unique identifier for each. Finally, at the topmost level, a class for managers was created, which included additional details such as the department they oversee.  The system needed to handle the initialization of all these attributes through constructors at each level, ensuring the proper propagation of information across the hierarchy. Additionally, the functionality to display details at every level was included to provide clear insights into the organization's structure. Two approaches were explored for managing multiple managers: one relied on an efficient method for retrieval and organization based on employee identifiers, while the other used a straightforward and static method for storage. |
| **Code** | //This program is prepared by 24CE089\_NILAY PATEL.  #include<iostream>//Standard input/output library  #include<cstring>//Standard string library  using namespace std;  class Person//Creating class  {  //Private data-members  char Name[10];  int Age;  //Making public functions for different tasks  public:  Person()//Constructor of Person class  {  //Initializing data members with default value  strcpy(Name," ");  Age = 0;  }  void AddPersonData()//Function for Adding person's data  {  cout << "Enter Name: ";  cin >> Name;//Taking input of Name  cout << "Enter Age: ";  cin >> Age;//Taking input of age  }  void DisplayPersonData()//Function for displaying person's data  {  cout << "Name: " << Name << endl;//Displaying Name  cout << "Age: " << Age << endl;//Displaying Age  }  };  class Employee : private Person//Creating class with a parent class  {  //Private data-members  int EmployeeID;  //Making public functions for different tasks  public:  Employee() : Person()//Constructor of Person class with a constructor of parent  {  //Initializing data members with default value  EmployeeID = 0;  }  void AddEmployeeData()//Function for displaying employee's data  {  AddPersonData();//Calling AddPersonData function internally  cout << "Enter employee ID: ";  cin >> EmployeeID;//Taking input of employee ID  }  void DisplayEmployeeData()//Function for displaying employee's data  {  DisplayPersonData();//Calling DisplayPersonData function internally  cout << "Employee ID: " << EmployeeID << endl;//Displaying employee's ID  }  };  class Manager : private Employee//Creating class with a parent class  {  //Private data-members  char Department[10];  //Making public functions for different tasks  public:  Manager() : Employee()//Constructor of Person class with a constructor of parent  {  //Initializing data members with default value  strcpy(Department," ");  }  void AddManagerData()//Function for displaying manager's data  {  AddEmployeeData();//Calling AddEmployeeData function internally  cout << "Enter department: ";  cin >> Department;//Taking input of department  }  void DisplayManagerData()//Function for displaying Manager's data  {  cout << "Entered data:~" << endl;  DisplayEmployeeData();//Calling DisplayEmployeeData function internally  cout << "Department: " << Department << endl;//Displaying department  }  };  int main()//Main function  {  class Manager M;//Creating object  M.AddManagerData();//Calling AddManagerData function  M.DisplayManagerData();//Calling DisplayManagerData function  cout << "24CE089\_NILAY PATEL." << endl;//Printing exiting message  return 0;//Return statement  } |
| **Output** |  |

**Sign: Grade:**

|  |  |
| --- | --- |
|  | **Practical** |
| **Program: 4.3** | A vehicle manufacturing company sought to create a robust system to organize and manage the details of various cars produced under its brand. To accomplish this, a hierarchical structure was conceptualized, reflecting the essential components of a vehicle. At the foundation, a class was designed to represent the type of fuel a vehicle uses. Another class was created to capture the brand name of the vehicle. These two foundational elements were then combined into a derived class specifically representing cars, integrating both fuel type and brand information.  Constructors were used at each level to ensure proper initialization of attributes, allowing seamless integration of all details. Additionally, the ability to display complete information about a car, including its fuel type and brand, was incorporated into the system. To simulate a real-world scenario such as a service queue, multiple cars were organized and processed sequentially using a structured approach. This not only streamlined the handling of cars but also provided an opportunity to compare different methods of managing the collection and processing of vehicle data. |
| **Code** | //This program is prepared by 24CE089\_NILAY PATEL.  #include<iostream>//Standard input/output library  #include<cstring>//Standard string library  using namespace std;  //Fixing input for fuel-type  enum FuelType {Diesel = 'D', Petrol = 'P', Hybrid = 'H', CNG = 'C'};  class Fuel//Creating class  {  //Private data-members  FuelType FType;  //Making public functions for different tasks  public:  Fuel()//Constructor of Fuel class  {  //Putting values in parameter  FType = Petrol;  }  void AddFuelType()//Function for adding fuel type  {  char FChoice;//Variable for choice  cout << "Enter fuel type(D for diesel,P for petrol,C for CNG & H for hybrid or petrol/CNG both): ";  while(true)//Loop for taking input & Initializing FType variable  {  cin >> FChoice;//Taking input of choice  switch(FChoice)//Switch case  {  case 'D'://Case for diesel  FType = Diesel;//Initializing Ftype variable with diesel  break;//Break statement  case 'P'://Case for petrol  FType = Petrol;//Initializing Ftype variable with petrol  break;//Break statement  case 'H'://Case for hybrid or petrol/CNG both  FType = Hybrid;//Initializing Ftype variable with hybrid  break;//Break statement  case 'C'://Case for CNG  FType = CNG;//Initializing Ftype variable with CNG  break;//Break statement  default://Default case  //Printing error message & re-taking input of fuel-type  cout << "Invalid input! Please enter a valid fuel type (D, P, H, C): ";  continue;//Continue statement  }  break;//Break statement  }  }  void DisplayFuelType() const//Function for displaying fuel fuel-type  {  switch(FType)//Switch case  {  case Diesel://Case for diesel  cout << "Fuel type: Diesel" << endl;//Printing fuel type : diesel  break;//Break statement  case Petrol://Case for petrol  cout << "Fuel type: Petrol" << endl;//Printing fuel type : petrol  break;//Break statement    case CNG://Case for CNG  cout << "Fuel type: CNG" << endl;//Printing fuel type : CNG  break;//Break statement  case Hybrid://Case for hybrid or petrol/CNG both  cout << "Fuel type: Hybrid" << endl;//Printing fuel type : hybrid  break;//Break statement  }  }  };  class Brand//Creating class  {  //Private data-member  char BrandName[10];  //Making public functions for different tasks  public:  Brand()//Constructor of Brand class  {  //Initializing data members with default value  strcpy(BrandName," ");  }  void AddBrandDetails()//Function for adding brand details  {  cout << "Enter brand's name: ";  cin >> BrandName;//Talking input of brand name  }  void DisplayBrandDetails()//Function for displaying brand details  {  cout << "Brand's name: " << BrandName << endl;//Printing brand name  }  };  class Car : private Fuel,Brand//Creating class with a parent class  {  //Private data-member  char CarName[10];  //Making public functions for different tasks  public:  //Constructor of Car class with a constructor of parent classes  Car() : Fuel(),Brand()  {  //Initializing data members with default value  strcpy(CarName,"");  }  void AddCarDetails()//Function for adding car details  {  cout << "Enter car name: ";  cin >> CarName;//Talking input of car name  AddBrandDetails();//Calling AddBrandDetails internally  AddFuelType();//Calling AddFuelType function internally  }  void DisplayCarDetails()//Function for displaying car details  {  cout << "Entered car details:~" << endl;  cout << "Car name: " << CarName << endl;//Printing car name  DisplayBrandDetails();//Calling DisplayBrandDetails function internally  DisplayFuelType();//Calling DisplayFuelType function internally  }  };  int main()//Main function  {  class Car C;//Creating object  C.AddCarDetails();//Calling AddCarDetails function  C.DisplayCarDetails();//Calling DisplayCarDetails function  cout << "24CE089\_NILAY PATEL" << endl;//Printing exiting message  return 0;//Return statement  } |
| **Output** |  |

**Sign: Grade:**

|  |  |
| --- | --- |
|  | **Practical** |
| **Program: 4.4** | In a bid to design an efficient and user-friendly banking system, a structure was proposed that mirrors the real-world operations of various account types. The foundation of the system is a base class representing a generic bank account, encapsulating essential details such as account number and balance. Building on this foundation, two specialized account types were created: a savings account, which includes an interest rate as an additional feature, and a current account, which allows an overdraft limit to accommodate specific customer needs.  To ensure proper initialization and cleanup of account objects, constructors and destructors were implemented. Essential banking operations such as deposits and withdrawals were made available for both account types, allowing users to perform and manage their transactions effectively. The system also accounted for the need to track and manage transaction history, enabling operations such as undoing the last transaction. This was achieved by simulating a mechanism to store a sequence of transactions for each account type, providing insight into different ways of managing and organizing data. |
| **Code** | //This Program is prepared by 24CE089\_NILAY PATEL  #include<iostream>//Standard Input/Output library  using namespace std;  class BankAccount//Creating class  {  protected:  //Protected data-members  long AccNum;  long Balance;  //Making public functions for different tasks  public:  BankAccount()//Constructor of BankAccount class  {  //Initializing data members with default value  AccNum = 0;  Balance = 0;  }  virtual ~BankAccount()//Virtual destructor  {  }  virtual void DisplayAccountDetails() = 0;//Virtual function  void AddAccDetails()//Function for entering details of bank account  {  cout << "Enter account number: ";  cin >> AccNum;//Taking input of account number  cout << "Enter balance: ";  cin >> Balance;//Taking input of balance  cout << "Account addition successful!" << endl;//Printing message  }  void DepositMoney()//Function to deposit  {  long Amount;//Variable for amount to be deposited  cout << "Enter the amount to be deposited: ";  cin >> Amount;//Taking input of amount to be deposited  Balance += Amount;//The amount to be deposit in account  cout << "Deposit successful!" << endl;//Printing message  cout << "Current balance: " << Balance << endl;//Printing balance  }  void WithdrawMoney()//Function to withdraw  {  long Amount;//Variable for amount to be withdrawn  cout << "Enter the amount to be withdrawn: ";  cin >> Amount;//Taking input of amount to be withdrawn  //If wanted amount of withdrawal is greater than balance....  if(Amount >= Balance)  {  cout << "Insufficient balance!" << endl;//Printing error message  }  else//If not....  {  Balance -= Amount;//The amount to be withdraw from account  cout << "Withdrawal successful!" << endl;//Printing message  cout << "Current balance: " << Balance << endl;//Printing balance  }  }  inline long GetAccNum()//Function for returning the value of account number  {  return AccNum;//Return statement  }  inline long GetBalance()//Function for returning the value of account number  {  return Balance;//Return statement  }  inline void SetBalance(double Blnc)//Function for storing value of balance  {  Balance = Blnc;//storing value of balance  }  };  class SavingsAccount : public BankAccount//Creating class with parent class  {  //Private data-member  float InterestRate;  //Making public functions for different tasks  public:  //Constructor of Person class with a constructor of parent  SavingsAccount() : BankAccount()  {  //Initializing data members with default value  InterestRate = 0;  }  void ApplyInterest()//Function for applying interest  {  double Interest;//Variable for interest  cout << "Enter Interest Rate: ";  cin >> InterestRate;//Talking input of interest  Interest = GetBalance()\*InterestRate/100;//Calculating interest  cout << "Applied interest: " << Interest << endl;//Printing interest  }  void DisplaySAccDetails()//Function for displaying account's details  {  cout << "Account number: " << GetAccNum() << endl;//Printing account no.  //Printing current balance of account  cout<< "Current balance: " << GetBalance() << endl;  cout << "Interest rate: " << InterestRate << "%" << endl;//Printing interest rate  }  //Function for overriding DisplayAccountDetails function  void DisplayAccountDetails() override  {  DisplaySAccDetails();//Calling DisplaySAccDetails  }  ~SavingsAccount() override//Overriding destructor  {  }  };  class CurrentAccount : public BankAccount//Creating class with parent class  {  //Private data-member  long OverdraftLimit;  public:  //Constructor of Person class with a constructor of parent  CurrentAccount() : BankAccount()  {  //Initializing data members with default value  OverdraftLimit = 0;  }  void ApplyODLimit()//Function for applying overdraft-limit  {  long Amount;//Variable for amount to be over-drafted  cout << "Enter Overdraft Limit: ";  cin >> OverdraftLimit;//Taking input of overdraft-limit  cout << "Enter amount to be withdrawn: ";  cin >> Amount;//Taking input of amount to be withdrawn  if (Amount > GetBalance() + OverdraftLimit)//If  {  cout << "Invalid overdraft limit!"<< endl;//Printing error message  return;//Return statement  }  SetBalance(GetBalance() - Amount);//Substituting Amount from Balance  cout << "Current balance:" << GetBalance() << endl;//Printing balnce  }  void DisplayCAccDetails()//Function for displaying account's details  {  cout << "Account number: " << GetAccNum() << endl;//Printing account no.  //Printing current balance of account  cout<< "Current balance: " << GetBalance() << endl;  cout << "Overdraft limit: " << OverdraftLimit << endl;  }  //Function for overriding DisplayAccountDetails function  void DisplayAccountDetails() override  {  DisplayCAccDetails();//Calling DisplayCAccDetails function  }  ~CurrentAccount() override//Overriding destructor  {  }  };  int main()//Main function  {  //Creating objects  class SavingsAccount\* S[100];  class CurrentAccount\* C[100];  char Choice1;//Variables for choices  int Choice2,Choice3;//Variables for choices  //Variables for tracking the no. of savings & current account & initializing to 0  int NumAcc1 = 0,NumAcc2 = 0;  long AccFind;//Variable for matching account number  bool Found;//Variable for signal if account no. matched  //Displaying menu  cout << "Enter S for Savings account." << endl;  cout << "Enter C for Current account." << endl;  cout << "Enter E to exit." << endl;  cout << "==================================" << endl;  while(true)//Loop for different tasks on savings account & Current account  {  cout << "Enter your choice: ";  cin >> Choice1;//Taking input of Choice1  switch(Choice1)//Switch case  {  case 'S'://Case for savings account  {  //Displaying menu  cout << "Enter 1 to add savings account's details." << endl;  cout << "Enter 2 to deposit money." << endl;  cout << "Enter 3 to withdraw money." << endl;  cout << "Enter 4 to apply interest." << endl;  cout << "Enter 5 to display account's details." << endl;  cout << "Enter your choice: ";  cin >> Choice2;//Taking input of Choice2  Found = false;//Variable for signal if account no. matched  switch(Choice2)//Switch case  {  case 1://Case for adding savings account's details  S[NumAcc1] = new SavingsAccount();//Calling AddAccDetails function  S[NumAcc1]->AddAccDetails();//Calling AddAccDetails function  NumAcc1++;//Increment in NumAcc Variable  break;//Break statement  case 2://Case for depositing money  case 3://Case for withdrawing money  case 4://Case for applying interest  case 5://Case for displaying account's details  cout << "Enter the account number: ";  cin >> AccFind;//Taking input of Account no. to search    //Loop for matching Account no. with entered account no.  for (int i = 0; i < NumAcc1; i++)  {  //If entered account no. matched with account no.  if (S[i]->GetAccNum() == AccFind)  {  if (Choice2 == 2)//If Choice2 is equal to 2  {  S[i]->DepositMoney();//Calling DepositMoney function  }  else if (Choice2 == 3)//If Choice2 is equal to 3  {  S[i]->WithdrawMoney();//Calling WithdrawMoney function  }  else if (Choice2 == 4)//If Choice2 is equal to 4  {  S[i]->ApplyInterest();//Calling ApplyInterest function  }  else if (Choice2 == 5)//If Choice2 is equal to 5  {  //Calling DisplaySAccDetails function  //Calling DisplaySAccDetails function  S[i]->DisplaySAccDetails();  }  Found = true;//Account no. matched with entered account no.  break;//Break statement  }  }  if(! Found)//If Found variable doesn't switch to true...  {  cout << "Account not found!" << endl;//Printing error message  }  break;//Break statement  default:  cout << "Invalid input!" << endl;//Printing error message  break;//Break statement  }  break;//Break statement  }  case 'C'://Case for current account  {  //Displaying menu  cout << "Enter 1 to add current account's details." << endl;  cout << "Enter 2 to deposit money." << endl;  cout << "Enter 3 to withdraw money." << endl;  cout << "Enter 4 to display account's details." << endl;  cout << "Enter your choice: ";  cin >> Choice3;//Taking input of choice  Found = false;//Variable for signal if account no. matched  switch(Choice3)//Switch case  {  case 1://Case for adding current account's details  //Creating object by dynamic allocation  C[NumAcc2] = new CurrentAccount();  C[NumAcc2]->AddAccDetails();//Calling AddAccDetails function  NumAcc2++;//Increment in NumAcc Variable  break;//Break statement  case 2://Case for depositing money  case 3://Case for withdrawing money  case 4://Case for displaying account's details  cout << "Enter the account number: ";  cin >> AccFind;//Taking input of Account no. to search  //Loop for matching Account no. with entered account no.  for (int i = 0; i < NumAcc2; i++)  {  //If entered account no. matched with account no.  if (C[i]->GetAccNum() == AccFind)  {  if (Choice3 == 2)//If Choice3 is equal to 3  {  C[i]->DepositMoney();//Calling DepositMoney function  }  else if (Choice3 == 3)//If Choice3 is equal to 3  {  C[i]->ApplyODLimit();//Calling ApplyODLimit function  }  else if (Choice3 == 4)//If Choice3 is equal to 4  {  //Calling DisplayCAccDetails function  C[i]->DisplayCAccDetails();  }  Found = true;//Account no. matched with entered account no.  break;//Break statement  }  }  if(! Found)//If Found variable doesn't switch to true...  {  cout << "Account not found!" << endl;//Printing error message  }  break;//Break statement  default://Default case  cout << "Invalid input!" << endl;//Printing error message  break;//Break statement  }  break;//Break statement  }  case 'E'://Case for exiting  {  for (int i = 0; i < NumAcc1; i++)//Loop for deleting array  {  delete S[i];//Free up S named array of pointer  }  for (int i = 0; i < NumAcc2; i++)//Loop for deleting array  {  delete C[i];//Free up C named array of pointer  }  cout << "24CE089\_NILAY PATEL." << endl;//Printing existing message  return 0;//Return statement  }  default://Default case  {  cout << "Invalid input!" << endl;//Printing error message  break;//Break statement  }  }  }  } |
| **Output** |  |

**Sign: Grade:**

|  |  |
| --- | --- |
|  | **Practical** |
| **Program: 4.5** | In an educational setting, an advanced grading system was conceptualized to accommodate the diverse evaluation criteria for students at different academic levels. At the heart of the system is an abstract base class that defines the grading framework. This class includes a protected member to store marks and declares a pure virtual function for computing grades, ensuring that specific grading logic is implemented by derived classes.  Two distinct derived classes were introduced to handle the unique grading needs of undergraduate and postgraduate students. Each class defines its own implementation of the grade computation method, reflecting the varying academic expectations for these groups. The system enables users to input student data, compute grades based on the respective criteria, and manage a collection of student records. |
| **Code** | //This Program is prepared by 24CE089\_NILAY PATEL  #include<iostream>//Standard Input/Output library  #include<cstring> //String library  #include<iomanip>//Input/Output manipulation library  using namespace std;  class GradingFramework//Creating class  {  protected:  //Protected data-members  char Name[20];  float Marks1, Marks2, Marks3;  //Making public functions for different tasks  public:  //Pure virtual function  virtual void AddStuDetails() = 0;//For adding student's details  virtual void CalculateAverage() = 0;//For calculating average  virtual void DisplayStuDetails() = 0;//For displaying student's details  };  class UGStudent : public GradingFramework//Creating class with parent class  {  //Making public functions for different tasks  public:  void AddStuDetails() override//Overriding function for adding student's details  {  cout << "Enter student's name: ";  cin >> Name;//Taking input of student's name  cout << "Enter marks of subject 1: ";  cin >> Marks1;//Taking input of marks of sub.1  cout << "Enter marks of subject 2: ";  cin >> Marks2;//Taking input of marks of sub.2  cout << "Enter marks of subject 3: ";  cin >> Marks3;//Taking input of marks of sub.3  cout << "===========================" << endl;  }  //Overriding function for calculating average marks  void CalculateAverage() override  {  //Printing average marks of three sub.s  cout << "The average marks of 3 subjects: " << fixed << setprecision(2) << (Marks1 + Marks2 + Marks3) / 3.0 << endl;  cout << "======================================" << endl;  }  //Overriding function for displaying student's details  void DisplayStuDetails() override  {  cout << "The student's details: " << endl;  cout << "Name: " << Name << endl;//Printing student's name  cout << "Marks of subject 1: " << Marks1 << endl;//Printing marks of sub.1  cout << "Marks of subject 2: " << Marks2 << endl;//Printing marks of sub.2  cout << "Marks of subject 3: " << Marks3 << endl;//Printing marks of sub.3  //If average marks of 3 sub.s is greater than or equal to 40...  if (((Marks1 + Marks2 + Marks3)) / 3 >= 40)  {  cout << "The student is Pass" << endl;//Printing pass  }  else//If not...  {  cout << "The student is Fail" << endl;//Printing fail  }  }  };  class PGStudent : public GradingFramework//Creating class with parent class  {  //Making public functions for different tasks  public:  void AddStuDetails() override //Overriding function for adding student's details  {  cout << "Enter name of student: ";  cin >> Name;//Taking input of student's name  cout << "Enter marks of subject 1: ";  cin >> Marks1;//Taking input of marks of sub.1  cout << "Enter marks of subject 2: ";  cin >> Marks2;//Taking input of marks of sub.2  cout << "Enter marks of subject 3: ";  cin >> Marks3;//Taking input of marks of sub.3  cout << "==========================" << endl;  }  //Overriding function for calculating average marks  void CalculateAverage() override  {  //Printing average marks of three sub.s  cout << "The average marks of 3 subjects: " << fixed << setprecision(2) << (Marks1 + Marks2 + Marks3) / 3.0 << endl;  cout << "======================================" << endl;  }  //Overriding function to display result and marks  void DisplayStuDetails() override  {  cout << "The student's details: " << endl;  cout << "Name: " << Name << endl;//Printing student's name  cout << "Marks of subject 1: " << Marks1 << endl;//Printing marks of sub.1  cout << "Marks of subject 2: " << Marks2 << endl;//Printing marks of sub.2  cout << "Marks of subject 3: " << Marks3 << endl;//Printing marks of sub.3  //If average marks of 3 sub.s is greater than or equal to 40...  if ((Marks1 + Marks2 + Marks3) / 3.0 >= 50)  {  cout << "The student is Pass" << endl;//Printing pass  }  else//If not...  {  cout << "The student is Fail" << endl;//Printing fail  }  }  };  int main() //Main function  {  //Creating objects  UGStudent UGS;  PGStudent PGS;  //Declaring pointers of base class  GradingFramework \*GF1,\*GF2;  //Assigning objects to base class pointers  GF1 = &UGS;  GF2 = &PGS;  int Choice;//Variable for choice  //Displaying menu  cout << "Enter 1 for UG student." << endl;  cout << "Enter 2 for PG student." << endl;  cout << "Else to exit." << endl;  cout << "=======================" << endl;  Label://Label  cout << "Enter your choice: ";  cin >> Choice;//Taking input of choice  if (Choice == 1)//If choice is equal to 1...  {  GF1->AddStuDetails();//Calling AddStuDetails function for UG  GF1->DisplayStuDetails();//Calling DisplayStuDetails function for UG  GF1->CalculateAverage();//Calling CalculateAverage function for UG  goto Label;//Goto statement  }  else if (Choice == 2)//If choice is equal to 2...  {K  GF2->DisplayStuDetails();//Calling DisplayStuDetails function for PG  GF2->CalculateAverage();//Calling CalculateAverage function for PG  goto Label;//Goto statement  }  else//If not...  {  cout << "24CE089\_NILAY PATEL." << endl;//Printing exiting message  return 0;//Return statement  }  } |
| **Output** |  |

**Sign: Grade:**