***A relative of yours is concerned with increasing electricity bills, and he knowing that you are on your way to become a software engineer has requested you to create a small programme for him so he can monitor and eventually control his monthly electricity bill. Since he is not good in calculations so he wants your program to determine his monthly units consumption and projected electricity bill. Each functionality he requires would be accessible through a menu.***

***Q1:***

***a- There are two types of reading from the meter, peak hours and regular hours. You need to take input of these two readings daily for complete month, number of days in a month can be 28, 29, 30 or 31. This number can be taken as input from user and the storage can be designed according to 31.***

* ***Implement function to take input of meter reading for peak and regular taken for meter for each day***
* ***Calculate and display daily consumption in peak and regular hours for each day***

***b- Determine the maximum consumption dates based on peak and regular hours units using loops***

***c- To reduce bill he has now installed a solar inverter, and now he have 4 readings due to net metering. Import Peak and import regular which is consuming electricity from Wapda, and export peak and export regular which is producing and giving it to Wapda. Create a new structure in separate function, without modifying part a, to store all these four readings on daily basis.***

* ***Implement function to take input of meter reading for peak and regular for both import and export taken for meter for each day***
* ***Calculate and display daily consumption and production in peak and regular hours for each day***

***d- Use single recursive function to determine the date of maximum reading difference of each category, peak import, peak export, regular import & regular export.***

***Sample Readings for any category:***

***89475.71***

***89491.84***

***89517.53***

***89526.7***

***89539.61***

***89562.27***

***89585.26***

***89603.73***

***(hint: consumption/ production is the difference in consecutive readings)***

|  |
| --- |
| #include <iostream>  const int MAX\_DAYS = 31;  // Structure to store daily readings  struct DailyReadings {  double peakHoursReading;  double regularHoursReading;  };  // Structure to store daily import/export readings  struct NetMeteringReadings {  double peakImportReading;  double peakExportReading;  double regularImportReading;  double regularExportReading;  };  // Function to take input of meter reading for peak and regular hours for each day  void takeDailyReadings(DailyReadings readings[MAX\_DAYS], int numDays) {  for (int i = 0; i < numDays; ++i) {  std::cout << "Enter peak hours reading for day " << i + 1 << ": ";  std::cin >> readings[i].peakHoursReading;  std::cout << "Enter regular hours reading for day " << i + 1 << ": ";  std::cin >> readings[i].regularHoursReading;  }  }  // Function to calculate and display daily consumption in peak and regular hours for each day  void calculateAndDisplayDailyConsumption(DailyReadings readings[MAX\_DAYS], int numDays) {  std::cout << "Daily Consumption:" << std::endl;  for (int i = 0; i < numDays; ++i) {  double peakConsumption = readings[i].peakHoursReading - (i > 0 ? readings[i - 1].peakHoursReading : 0);  double regularConsumption = readings[i].regularHoursReading - (i > 0 ? readings[i - 1].regularHoursReading : 0);  std::cout << "Day " << i + 1 << ": Peak - " << peakConsumption << " Regular - " << regularConsumption << std::endl;  }  }  // Function to determine the maximum consumption dates based on peak and regular hours units  void determineMaxConsumptionDates(DailyReadings readings[MAX\_DAYS], int numDays) {  double maxPeakConsumption = 0.0;  double maxRegularConsumption = 0.0;  int maxPeakDay = 0;  int maxRegularDay = 0;  for (int i = 0; i < numDays; ++i) {  double peakConsumption = readings[i].peakHoursReading - (i > 0 ? readings[i - 1].peakHoursReading : 0);  double regularConsumption = readings[i].regularHoursReading - (i > 0 ? readings[i - 1].regularHoursReading : 0);  if (peakConsumption > maxPeakConsumption) {  maxPeakConsumption = peakConsumption;  maxPeakDay = i + 1;  }  if (regularConsumption > maxRegularConsumption) {  maxRegularConsumption = regularConsumption;  maxRegularDay = i + 1;  }  }  std::cout << "Maximum Consumption Dates:" << std::endl;  std::cout << "Peak Hours - Day " << maxPeakDay << ", Regular Hours - Day " << maxRegularDay << std::endl;  }  // Function to take input of meter reading for peak and regular for both import and export for each day  void takeNetMeteringReadings(NetMeteringReadings netReadings[MAX\_DAYS], int numDays) {  for (int i = 0; i < numDays; ++i) {  std::cout << "Enter peak import reading for day " << i + 1 << ": ";  std::cin >> netReadings[i].peakImportReading;  std::cout << "Enter peak export reading for day " << i + 1 << ": ";  std::cin >> netReadings[i].peakExportReading;  std::cout << "Enter regular import reading for day " << i + 1 << ": ";  std::cin >> netReadings[i].regularImportReading;  std::cout << "Enter regular export reading for day " << i + 1 << ": ";  std::cin >> netReadings[i].regularExportReading;  }  }  // Function to calculate and display daily consumption and production in peak and regular hours for each day  void calculateAndDisplayNetMetering(DailyReadings readings[MAX\_DAYS], NetMeteringReadings netReadings[MAX\_DAYS], int numDays, int day = 0) {  if (day == numDays) {  return; // Base case for recursion  }  double peakConsumption = readings[day].peakHoursReading - (day > 0 ? readings[day - 1].peakHoursReading : 0);  double regularConsumption = readings[day].regularHoursReading - (day > 0 ? readings[day - 1].regularHoursReading : 0);  double peakProduction = netReadings[day].peakExportReading - netReadings[day].peakImportReading;  double regularProduction = netReadings[day].regularExportReading - netReadings[day].regularImportReading;  std::cout << "Day " << day + 1 << ": ";  std::cout << "Peak Consumption - " << peakConsumption << ", Peak Production - " << peakProduction << ", ";  std::cout << "Regular Consumption - " << regularConsumption << ", Regular Production - " << regularProduction << std::endl;  // Recursive call for the next day  calculateAndDisplayNetMetering(readings, netReadings, |

***A teacher wants you to develop a program which can maintain quizzes marks of his students along with their roll numbers and names. He came aware that your programming skills are good and you would be able to make a menu driven program which can fulfil his following requirements.***

***Q2:***

1. ***There are three types of inputs for an individual student, name, roll number and marks in quizzes. Each quiz is marked out of 10, and lets assume maximum number of quizzes can be 10, but you need to take input of number of quizzes actually conducted from the teacher at start along with number of students, maximum students can be 100 in a class.***

***Assume maximum length of name is 30 and roll number is 10 characters.***

* ***Implement a function to take input of each student***
* ***Display the information collected in form of a table, roll number, name, quiz 1 marks, quiz 2 marks, and so on.***

1. ***Determine the highest accumulative score, and display roll number and name. In case multiple students have same highest score show all of them through loops***
2. ***Teacher has decided to remove worst two performances for each student, the worst quiz for one student can be best for the other. Write a function which calculates accumulative score by excluding worst two performances.***

***d- Determine the highest accumulative score, and display roll number and name. In case multiple students have same highest score show all of them through recursion***

***Sample execution:***

***Number of students: 10***

***Number of quizzes: 5***

***Enter data for student 1:***

***Name : ABCD EFGH***

***Roll Number: BSSE23000***

***Quiz 1 Marks : 5***

***Quiz 2 Marks : 7***

***Quiz 3 Marks : 3***

***Quiz 4 Marks : 4***

***Quiz 5 Marks :***

***1 Enter data for student 2: . . .***

|  |
| --- |
| #include <iostream>  #include <iomanip>  #include <limits>  const int MAX\_STUDENTS = 100;  const int MAX\_QUIZZES = 10;  const int MAX\_NAME\_LENGTH = 30;  const int MAX\_ROLL\_NUMBER\_LENGTH = 10;  // Structure to store student information  struct Student {  char name[MAX\_NAME\_LENGTH];  char rollNumber[MAX\_ROLL\_NUMBER\_LENGTH];  int quizMarks[MAX\_QUIZZES];  int accumulativeScore; // To store accumulative score after removing worst two performances  };  // Function to take input of each student  void takeStudentInput(Student students[MAX\_STUDENTS], int numStudents, int numQuizzes) {  for (int i = 0; i < numStudents; ++i) {  std::cout << "Enter data for student " << i + 1 << ":" << std::endl;  std::cout << "Name: ";  std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n'); // Clear buffer  std::cin.getline(students[i].name, MAX\_NAME\_LENGTH);  std::cout << "Roll Number: ";  std::cin.getline(students[i].rollNumber, MAX\_ROLL\_NUMBER\_LENGTH);  for (int j = 0; j < numQuizzes; ++j) {  std::cout << "Quiz " << j + 1 << " Marks: ";  std::cin >> students[i].quizMarks[j];  }  }  }  // Function to display student information in a table  void displayStudentInfo(const Student students[MAX\_STUDENTS], int numStudents, int numQuizzes) {  std::cout << std::setw(15) << "Roll Number" << std::setw(30) << "Name";  for (int i = 0; i < numQuizzes; ++i) {  std::cout << std::setw(15) << "Quiz " << i + 1;  }  std::cout << std::endl;  for (int i = 0; i < numStudents; ++i) {  std::cout << std::setw(15) << students[i].rollNumber << std::setw(30) << students[i].name;  for (int j = 0; j < numQuizzes; ++j) {  std::cout << std::setw(15) << students[i].quizMarks[j];  }  std::cout << std::endl;  }  }  // Function to determine the highest accumulative score  void determineHighestAccumulativeScore(const Student students[MAX\_STUDENTS], int numStudents, int numQuizzes) {  int maxScore = -1;  for (int i = 0; i < numStudents; ++i) {  int accumulativeScore = 0;  for (int j = 0; j < numQuizzes; ++j) {  accumulativeScore += students[i].quizMarks[j];  }  students[i].accumulativeScore = accumulativeScore;  if (accumulativeScore > maxScore) {  maxScore = accumulativeScore;  }  }  std::cout << "Students with the highest accumulative score:" << std::endl;  for (int i = 0; i < numStudents; ++i) {  if (students[i].accumulativeScore == maxScore) {  std::cout << "Roll Number: " << students[i].rollNumber << ", Name: " << students[i].name << std::endl;  }  }  }  // Function to calculate accumulative score by excluding worst two performances  void calculateAccumulativeScoreWithoutWorstTwo(Student& student, int numQuizzes) {  // Sort quiz marks in ascending order  for (int i = 0; i < numQuizzes - 1; ++i) {  for (int j = 0; j < numQuizzes - i - 1; ++j) {  if (student.quizMarks[j] > student.quizMarks[j + 1]) {  std::swap(student.quizMarks[j], student.quizMarks[j + 1]);  }  }  }  // Calculate accumulative score excluding worst two performances  student.accumulativeScore = 0;  for (int i = 2; i < numQuizzes; ++i) {  student.accumulativeScore += student.quizMarks[i];  }  }  // Recursive function to determine the highest accumulative score  void determineHighestAccumulativeScoreRecursively(const Student students[MAX\_STUDENTS], int numStudents, int currentStudent, int maxScore, int numQuizzes) {  if (currentStudent == numStudents) {  return; // Base case for recursion  }  int accumulativeScore = 0;  for (int i = 0; i < numQuizzes; ++i) {  accumulativeScore += |