Course Code: UTA018

Course Name: Object Oriented Programming (Using C++)

Practice Questions

……………………………………………………………………………………….

Topics Covered:

Class and objects, constructors, array of objects, array of objects with constructors, static data member, static member function, friend function, inheritance.

Q.1 Develop a class `Student` for managing student records in a school. The `Student` class should include attributes like `name` (string), `rollNumber` (int), and `marks` (float array of size 5). You are required to:

a. Implement a default constructor that initializes the `name` to `"Unknown"`, `rollNumber` to `0`, and all `marks` to `0.0`.

b. Implement a parameterized constructor to initialize the `name`, `rollNumber`, and `marks` with the provided values. The constructor should validate that `marks` are within the range `0-100`.

c. Implement a `friend` function `calculateTotalMarks` that takes a `Student` object as an argument and returns the total marks as a float.

d. Create an array of `Student` objects (size 3) and dynamically allocate memory for it using `new`. Populate the array using the parameterized constructor and display the total marks for each student using the `calculateTotalMarks` function.

e. Implement a `displayDetails()` function in the `Student` class to display the name, roll number, and total marks. Use this function to display details of all students in the array.

Write a `main` function that demonstrates the creation and manipulation of student records, along with proper memory deallocation using `delete`.

Q.2 (a) Define a class named CoffeeOrder. Declare a private static field that holds the price of a cup of coffee as $1.25. Include private integer fields that you set to a flag value of 1 or 0 to indicate whether the order should have any of the following: cream, milk, sugar, or artificial sweetener. Include a public function receiveOrder() that takes a user’s order from the keyboard and sets the values of the four fields in response to four prompts. If the user indicates both milk and cream, turn off the milk flag to allow only cream. If the user indicates both sugar and artificial sweetener, turn off the artificial sweetener flag, allowing only sugar. Include another function that displays the user’s completed order. Write a main() function that declares a CoffeeOrder object and calls the data entry and display methods.

(b) Using the CoffeeOrder class, write a main() function that continues to ask a user for an order in a loop until the user indicates the order is complete or 10 orders have been placed, whichever comes first. After the user indicates that ordering is complete, display a recap of all the coffee orders, including the cream, milk, sugar, and sweetener status of each, as well as a count of the number of coffees ordered and the total price.

Q.3 Create a class **`Course`** and its derived classes **`UndergraduateCourse`** and `GraduateCourse`**.** The `Course` class should have attributes `courseID` (int), `courseName` (string), and `credits` (int). Implement the following:

a. Implement a constructor in the `Course` class that initializes the `courseID`, `courseName`, and `credits`. Ensure that `credits` are non-negative. Also implement a member function `displayCourseDetails()to show the details of course.`

b. Implement an `UndergraduateCourse` class that adds the attribute `level` (string) and `displayCourseDetails()` function to display all course details, including the level (e.g., "Beginner", "Intermediate", "Advanced").

c. Implement a `GraduateCourse` class that adds the attribute `researchComponent` (bool) and `displayCourseDetails()` function to display all course details, including whether a research component is required.

d. Implement a `friend` function `compareCredits(Course& c1, Course& c2)` that compares the credits of two courses and returns the course with more credits. If the credits are equal, return either course.

e. Implement a `static` member function `getTotalCourses()` in the `Course` class that returns the total number of courses created.

Write a `main` function to create multiple undergraduate and graduate courses, compare credits, display course details, and show the total number of courses created using the static function.

Q.4 Develop a class **`Order`** that manages customer orders for an online store. The `Order` class should have attributes like `orderID` (static int), `customerName` (string), `itemList` (array of strings), and `totalAmount` (double). Implement the following features:

a. Implement a constructor that initializes the `orderID` using a static counter, `customerName` to the given value, and `itemList` using array. The `totalAmount` should be calculated based on the items in the `itemList`.

b. Implement a `friend` function `applyDiscount(Order& order, double discountPercentage)` that reduces the `totalAmount` by the given discount percentage.

d. Implement a method `displayOrderDetails()` that displays all details of the order, including the order ID, customer name, item list, and total amount.

e. Implement a `static` member function `getNextOrderID()` that returns the next available order ID.

Write a `main` function to create and manage multiple orders, apply discounts, display order details, and show the next available order ID using the static function.

Q.5 Create a class **`Inventory`** that manages a store's inventory of products**.** Each `Product` object should have attributes like `productID` (int), `name` (string), `quantity` (int), and `price` (double). Implement the following:

a. Implement a constructor in the `Inventory` class that initializes an array of `Product` objects using user input. The constructor should also take an argument for the initial `capacity` of the inventory.

b. Implement a method `addProduct(const Product& newProduct)` that adds a new product to the inventory. If the inventory is full, generate the message the inventory is full.

c. Implement a method `removeProduct(int productID)` that removes a product from the inventory by its `productID`. Enter the "0" in place of removed product.

d. Implement a `friend` function `findMostExpensiveProduct(const Inventory& inventory)` that returns a pointer to the product with the highest price. If there are no products, return `nullptr`.

e. Implement a method `displayInventory()` in the `Inventory` class that lists all products currently in the inventory, showing their `productID, name, quantity, and price`.

Write a `main` function to demonstrate the addition, removal, search, and display of products in the inventory. Ensure proper dynamic memory management throughout the program.

Q.6 Design a class **`Vehicle`** and its derived classes **`Car`**, **`Bike`**, and **`Truck`**. The `Vehicle` class should have attributes `make` (string), `model` (string), and `year` (int). Implement the following features:

a. Implement a constructor in the `Vehicle` class that initializes the `make`, `model`, and `year` attributes.

b. Implement a `Car` class that adds the attribute `numberOfDoors` (int) and a `showDetails()` function that displays all the details of the car, including the number of doors.

c. Implement a `Bike` class that adds the attribute `type` (string) and a `showDetails()` function that displays all the details of the bike, including its type. Example of `type` is "petrol", "CNG", "EV" etc.

d. Implement a `Truck` class that adds the attribute `loadCapacity` (double) and a `showDetails()` function that displays all the details of the truck, including its load capacity.

e. Implement a `static` data member in the `Vehicle` class to keep track of the total number of vehicles created. Implement a `static` member function to return the total number of vehicles.

Write a `main` function to create multiple `Car`, `Bike`, and `Truck` objects, display their details, and show the total number of vehicles created using the static function.

Q.7 Create a class **`Rectangle`** that represents a rectangle in a 2D space**.** The class should include attributes `length` (double) and `width` (double). Implement the following functionalities:

a. Implement a default constructor that initializes `length` and `width` to `1.0`. Implement a parameterized constructor that allows setting custom `length` and `width`. Ensure that both dimensions are positive.

b. Implement a `friend` function `calculateArea(Rectangle& rect)` that calculates and returns the area of the rectangle.

c. Implement a `friend` function `calculatePerimeter(Rectangle& rect)` that calculates and returns the perimeter of the rectangle.

d. Create an array of `Rectangle` objects (size 3) using dynamic initialization. Use a loop to input the dimensions for each rectangle from the user.

e. Implement a `static` member function `compareArea(Rectangle& rect1, Rectangle& rect2)` that compares the area of two rectangles and returns the rectangle with the larger area. If the areas are equal, return either rectangle.

Write a `main` function that demonstrates the creation of rectangle objects, calculation of area and perimeter, comparison of areas, and dynamic memory management.

Q.8 Design a class **`Department`** that manages a department in a university**,** and a `Faculty` class representing faculty members within that department. Implement the following:

a. Implement a constructor in the `Department` class that initializes the department's `name` (string) and `numberOfFaculty` (int). bonus points if "Use dynamic memory allocation to create an array of `Faculty` objects".

b. Implement a method `addFaculty(const Faculty& newFaculty)` in the `Department` class that adds a new faculty member to the department. If the department is full, dynamically increase its capacity.

c. Implement a method `removeFaculty(int facultyID)` that removes a faculty member based on their `facultyID` and adjusts the array accordingly.

d. Implement a `friend` class `University` that has access to the `Department` class's private members. The `University` class should be able to change the name of the department and list all faculty members in a department.

e. Implement a method `displayDepartmentDetails()` in the `Department` class that displays the department's name and lists all faculty members.

Write a `main` function to create and manage departments and faculty members, add and remove faculty, and demonstrate the use of the `University` class to manage department details.

Q.9 Develop a class **`Employee`** that manages employee records for a company**.** The `Employee` class should include attributes like `name` (string), `employeeID` (int), `department` (string), and `salary` (double). Implement the following features:

a. Implement a constructor that initializes `name`, `employeeID`, `department`, and `salary`. Ensure that `salary` is a non-negative value.

b. Implement a `friend` function `giveRaise(Employee& emp, double percentage)` that increases the employee's salary by a given percentage.

c. Implement a `static` data member `totalEmployees` in the `Employee` class that keeps track of the total number of employees. Implement a `static` member function `getTotalEmployees()` to return this count.

d. Implement a `displayEmployeeDetails()` function that displays all details of the employee, including name, employeeID, department, and salary.

Write a `main` function to create multiple `Employee` objects, give raises, update departments, terminate employees, and display employee details. Also, display the total number of employees using the static function.

Q.10 Develop a class **`ConstructionProject`** that manages a construction project, and a `Contractor` class representing contractors involved in the project. Implement the following:

a. Implement a constructor in the `ConstructionProject` class that initializes the project name (string) and budget (double). Ensure that the budget is non-negative.

b. Implement a `Contractor` class that includes attributes `contractorID` (int), `name` (string), and `projectCost` (double). Implement a method `displayContractorDetails()` in the `Contractor` class that displays the contractor's details.

c. Implement a method `assignContractor(const Contractor& newContractor)` in the `ConstructionProject` class that adds a contractor to the project. Ensure that the total project cost does not exceed the budget.

d. Implement a `friend` function `calculateTotalCost(const ConstructionProject& project)` that returns the total cost of all contractors assigned to the project.

e. Implement a method `displayProjectDetails()` in the `ConstructionProject` class that displays the project details, including the name, budget, and all assigned contractors.

Write a `main` function to create and manage construction projects and contractors, assign contractors, calculate total costs, display project and contractor details, and ensure that the project stays within budget.

Q.11 Create a class **`Patient`** and its derived classes **`Inpatient`** and **`Outpatient`**. The `Patient` class should have attributes `patientID` (int), `name` (string), `age` (int), and `disease` (string). Implement the following:

a. Implement a constructor in the `Patient` class that initializes the `patientID`, `name`, `age`, and `disease`. Ensure that `age` is non-negative.

b. Implement an `Inpatient` class that adds attributes `roomNumber` (int) and `daysAdmitted` (int). Implement a `displayPatientDetails()` function that displays all patient details, including the room number and days admitted.

c. Implement an `Outpatient` class that adds an attribute `visitDate` (string). Implement a `displayPatientDetails()` function that displays all patient details, including the visit date.

d. Implement a `static` data member `totalPatients` in the `Patient` class that keeps track of the total number of patients (both inpatient and outpatient). Implement a `static` member function `getTotalPatients()` to return this count.

e. Implement a `friend` function `compareDaysAdmitted(Inpatient& p1, Inpatient& p2)` that compares the days admitted for two inpatients and returns the one with more days. If the days are equal, return either patient.

Write a `main` function to create and manage inpatients and outpatients, compare days admitted, display patient details, and show the total number of patients using the static function.