Name & surname: ……

Define the class `ConsumerOrder` representing a list of dishes a customer orders in a restaurant. The class should contain the following fields: a customer name, two arrays/lists of the same size (empty by default) storing respectively the names of dishes (strings) and their prices (real numbers), and a unique sequential number (ID) of the order (a positive integer). Implement the following public methods of the class :

- the default constructor with a string parameter representing a customer name,
- the copy-constructor and the assignment operator,
- a destructor, which (in addition to other actions if necessary) prints the sequential number of the order being destroyed,
- setCustomer, getCustomer and getID methods,
- addDish method, with two parameters representing a dish name (a string) and its price (a real value), adding this dish to the order as the last position,
- removeDish method, with a dish name as a parameter, removing from the order all the occurrences of this dish, together with their costs (or throwing an error if the dish is not in the order),
- length – returning the number of elements in the order,
- getOrderCount – returning the total number of customer's orders made so far
- the operator >, where by a "greater" order we mean a more expensive one,
- the indexing operator ([]) – returning the price of the dish on the position given as the parameter (1 means the first dish, 2 – the second dish etc; an error should be thrown if the value given is greater than the length of the order),
- the shift operator (<<) – printing the number of the order, the customer name, the numbered list of all the ordered dishes and their costs, and the total cost of the order.

Write a program which tests all the class capabilities, in particular the following code should be enabled

```
cout << ConsumerOrder::getOrderCount(); //should be 0
ConsumerOrder o1("John Smith");
cout << o1.getCustomer(); //should be John Smith
o1.addDish("vegetable soup", 20);
o1.addDish("apple pie", 19);
cout << o1.length(); //should be 2
o1[1] = 21; //changes the price of the first dish
cout << o1; //should print for example:
//Order no. 1; customer: John Smith
//1. vegetable soup, 21
//2. apple pie, 19
//Total cost: 40
ConsumerOrder oo("Andrew Taylor");
oo.addDish("tomato soup", 20);
oo.addDish("grilled chicken", 45);
oo.addDish("tomato soup", 20);
oo.addDish("ice cream", 15);
if (oo > o1)
cout << "order "<< oo.getID() << "is more expensive than " << o.getID();
      //the message 'order 2 is more expensive than 1' should be printed
oo.removeDish("tomato soup");
cout << oo; //should print for example:
//Order no. 2; customer:  Andrew Taylor
//1. grilled chicken, 45
//2. ice cream, 15
//Total cost: 60
cout << ConsumerOrder::getOrderCount(); //should be 2
```

Name & surname: ……

Define the class `PizzeriaOrder` representing a list of dishes a customer orders in a pizzeria. The class should contain the following fields: a customer name and two arrays/lists of the same size (empty by default) storing respectively the names of dishes (strings) and their prices (real numbers). Implement the following public methods of the class :

- the default constructor with a string parameter representing a customer name,
- the copy-constructor and the assignment operator,
- a destructor,
- setCustomer and getCustomer methods,
- addDish method, with two parameters representing a dish name (a string) and its price (a real value), adding this dish to the order as the last position,
- removeDish method, with a dish name as a parameter, removing from the order the dish together with its price (or throwing an error if the dish is not in the order),
- totalCost – returning the total cost of the dishes in the order,
- getDishesCount – returning the summary number of all the dishes in all the orders existing currently
- the operator <, where by a "smaller" order we mean a one with a shorter list of dishes,
- the indexing operator ([]) – with a dish name as a parameter, returning the price of the dish placed in the order (an error should be thrown if the dish is not present in the order)
- the shift operator (<<) – printing the customer name, the numbered list of all the ordered dishes and their costs, and the total cost of the order.

Write a program which tests all the class capabilities, in particular the following code should be enabled

```
cout << PizzeriaOrder::getDishesCount(); //should be 0
PizzeriaOrder o1("John Smith");
cout << o1.getCustomer(); //should be John Smith
o1.addDish("garlic bread", 10);
o1.addDish("Margherita pizza", 25);
cout << o1.totalCost(); //should be 35
o1["Margherita pizza"] = 23; //changes the price of the dish
cout << o1; //should print for example:
//customer: John Smith
//1. garlic bread, 10
//2. Margherita pizza, 23
//Total cost: 33
PizzeriaOrder oo("Andrew Taylor");
oo.addDish("fried zucchini", 15);
oo.addDish("tuna pizza", 35);
oo.addDish("beer", 5);
if (o1 < oo)
cout << "order "<< o1.getCustomer() << " ordered smaller number of dishes
          than " << oo.getCustomer(); //the message should be printed
oo.removeDish("fried zucchini");
cout << oo; //should print for example:
//customer:  Andrew Taylor
//1. tuna pizza, 35
//2. beer, 5
//Total cost: 40
cout << PizzeriaOrder::getDishesCount(); //should be 4
```

Name & surname: ……

Define the class `CarData` representing a history of the car in a car rental. The class should contain the following fields: car registration number, car brand (strings), and two arrays/lists of the same size (empty by default) storing respectively names of customers (strings) and lengths of periods for which they rented this car (in hours; integer numbers). Implement the following public methods of the class:

- the default constructor with two string parameters representing car registration number and brand,
- the copy-constructor and the assignment operator,
- a destructor which (in addition to other actions if necessary) prints the registration number of the car being destroyed
- setRegistrationNumber, getRegistrationNumber and getBrand methods,
- addRental - with two parameters representing a customer name (a string) and the rental period in hours (an integer), adding this rental data as the last position,
- removeRentals method, with a customer name as a parameter, removing from the car history all the rentals of this customer together with their periods (or throwing an error if the customer is not present there),
- averageTime – returning the average time for which the car was rented (arithmetic mean of the rental periods stored in its history currently),
- getRentalsCount – returning the summary number of all the rentals of all the cars from the beginning of the program (this should include removed rentals and cars which no longer exist)
- the operator <, where by a “smaller” car we mean a one with a shorter list of rentals,
- the indexing operator ([]) – with an integer as a parameter, returning the period of the renting placed on the position given as the parameter (1 means the first renting, 2 – the second renting etc; an error should be thrown if the value given is greater than the length of the list),
- the shift operator (<<) – printing the car data, the history of the rentals (customers and time periods), and the total time of renting this car.

Write a program which tests all the class capabilities, in particular the following code should be enabled

```
cout << CarData::getRentalsCount(); //should be 0
CarData c1(“EL12345”, “Toyota Yaris”);
cout << c1.getRegistrationNumber(); //should be  EL12345
c1.addRental(“John Smith”, 7);
c1.addRental(“Jonathan Taylor”, 10);
cout << c1.averageTime(); //should be 8.5
c1[1] = 12; //changes the period of the first rental
cout << c1; //should print for example:
//car: Toyota Yaris EL12345
//1. John Smith, 12
//2. Jonathan Taylor, 10
//Total time: 22 hours

CarData c2(“EL12233”, “Opel Astra”);
c2.addRental(“Mark Johnson”, 10);
c2.addRental(“Elizabeth Smith”, 3);
c2.addRental(“Mark Johnson”, 2);
if (c1 < c2)
cout << c1.getBrand() << “ is used less often that “ << c2.getBrand(); /
                        /the message should be printed
c2.removeRentals(“Mark Johnson”);
cout << c2; //should print for example:
//car: Opel Astra EL12233
//1. Elizabeth Smith, 3
//Total time: 12 hours
cout << CarData::getRentalsCount(); //should be 5
```

Name & surname: ……

Define the class `Racer` with the following fields: a name of a race car driver (being a string), a unique positive sequential starting number computed for each racer and an array/list of his lap times in seconds (with the accuracy of miliseconds). The given starting number cannot be changed since the Racer object creation. Implement the following public methods of the class:

- the constructor with a string (denoting a driver name) as a parameter,
- the copy-constructor and the assignment operator,
- the destructor,
- setName, getName and getNumber methods,
- addLapTime – with a time in seconds as a parameter, adding the next lap time for the racer (or throwing an error if the time is not positive),
- removeLapTime – removing the racer's lap time placed on the position given as a parameter (1 means the first lap, 2 means the second lap, etc., an error should be thrown for the position exceeding the number of laps stored),
- getLapCount – returning the number of lap times stored for the racer,
- getBestLapTime – returning the minimal lap time for the racer,
- the operator >, where by a "greater" racer we mean a one with the better average lap time,
- the indexing operator ([]) – returning the racer's lap time placed on the position given as a parameter (1 means the first lap, 2 means the second lap, etc., an error should be thrown for the position exceeding the number of laps stored),
- the shift operator (<<) – printing the racer name, his number, the list of all his lap times, and the best time.

Write a program which tests all the class capabilities, i.e. the following code should be enabled:

```
Racer racer1("Robert Kubica");
cout << racer1.getNumber(); //should be 1
racer1.addLapTime(79.534);
racer1.addLapTime(80.005);
cout << racer1.getLapCount(); //should be 2
cout << racer1.getBestLapTime(); //should be 79.534
racer1[1] = 79.122; //changes the first lap time
cout << racer1; //should print:
              //Name: Robert Kubica
              //Number: 1
               //Lap times: 79.122 80.005
               //Best time: 79.122

Racer racer2("Lawis Hamilton");
racer2.addLapTime(81.349);
racer2.addLapTime(79.967);
racer2.addLapTime(79.999);
racer2.addLapTime(79.202);
cout << racer2[2]; //should be 79.967
racer2.removeLapTime(1);
cout << racer2[2]; //should be 79.999
cout << racer2.getLapCount(); //should be 3
cout << racer2.getBestLapTime(); //should be 79.202
racer2.setName("Lewis Hamilton");
cout << racer2.getName(); //should be Lewis Hamilton
cout << racer2; //should print:
              //Name: Lewis Hamilton
              //Number: 2
               //Lap times: 79.967 79.999 79.202
               //Best time: 79.202
if (racer1 > racer2) cout << racer1.getName() << "has better average result
than " << racer2.getName(); ///the message should be printed
Racer racer3(racer2);
cout << racer3.getNumber(); //should be 3
cout << racer3.getName(); //should be Lewis Hamilton
racer3 = racer1;
cout << racer3.getNumber(); //should be 3
cout << racer3.getName(); //should be Robert Kubica
```