**CIS 223 – Fall 2013**

**Lab # 7** - Car Dealership

Due: Beginning of class ( Tuesday Dec 10)

A used car dealership owner needs to needs to design a quick car inventory system to keep track of all the used cars on the lot.

Design a class called **Automobile** which stores the following information:

 Car ID number

* Car Make
* Car Model

Car Year

Car Mileage

Owner Cost (what he paid for it)

Design a class called **Dealership** that holds the automobiles the class holds the following information:

Actual number of cars on lot

*Array of Automobiles list[Capacity]*. The CAPACITY is the number of cars that can be in the dealership.

**// Dealership.h**

const int CAPACITY = 30;

**class Dealership**

{

public:

Dealership();

// constructor to set the number of cars to zero

bool addCar(Automobile aCar);

// a function to add an automobile to the list

// If the array is full, return false, else add the car to the list and return true

// If the car is already in the list, then replace it with aCar

bool deleteCar( Automobile aCar);

// a function to delete an automobile from the list

// if the array is empty, or the car is not in the array return false, else remove the

// car from the list and return true

bool retrieveCar( Automobile& aCar);

// a function that returns an Automobile from the list

// The input parameter aCar may hold only part or all of the information about

// the automobile, but must have the automobile id number( to be able to

// compare it with a car in the list)

// The function returns true if aCar was in the list, else it returns false

double total Investment();

// a function that returns the total cost that the dealership invested in cars

/\*\*\*\*\*\*

//print the dealership

printDealership()

// For each car print the Car ID number, car model, Car 

// mileage, owner car cost.

\*\*\*\*\*\*\*/

ostream& operator <<(ostream out, Dealership dealer);

// overload the << operator a function to print all the cars in the Dealership in

// ascending order by cost of the car\*.

// For each car print the Car ID number, car model, Car 

// mileage, owner car cost.

private:

numCars; // actual number of cars in the lot

Automobile CarList[CAPACITY]; // array of cars

void sort(); // a method to sort the automobiles in the

//dealership by cost of car

}

\* the operator << function will call private methods: **sort**

Use the following Automobile class, complete the class (implement each method):

//

// Automobile.h

//

// Class to keep track of a car on Dealer lot

// the class to set and get information about an automobile: the car Id, make, model, year

// of the car, the mileage on the car , and the cost of the car.

class Automobile

{

public:

void setId(string argId);

void setMake(string argMake);

void setModel(string argModel);

void setMileage(int argMileage);

void setCost(float argCost);

string getId();

string getMake();

string getModel();

int getMileage();

float getCost();

private:

string mId;

string mMake;

string mModel;

int mMileage;

float mCost;

};

1. create a program **(called Lab 7)** which reads from a file ("carInput.dat") a list of cars, one car per line

( look at the file bellow), and add each car to the Dealership list.

Input:

* Car ID number
* Car make
* Car Model

Car Mileage

Owner Cost (what he paid for it)

12345 Nissan Maxima 35379 7400

23456 Ford Escort 25667 6500

34567 Ford Taurus 55000 5500

# 45678 Pontiac Grandprix 60000 5000

2. In the program you will create a menu to:

* + add a new car
  + delete a car
  + retrieve a car and print the info about the car (if the car is not in the data base, give an appropriate message)
  + update info on the car ( cost)
  + print the inventory in ascending order

1. Your program should create an output file called **out.txt** with the following

information:

* 1. A list with all the cars ( in ascending order by cost of car)
  2. The total money invested in the Dealership(the total cost of all cars).

4. Create a function called **menu** that creates a menu for the following operations:

* 1. Create inventory ( from the provided file)
  2. Add a Car
  3. Delete a car
  4. Retrieve a car and print information about the car
  5. Update a car
  6. Print the inventory in ascending order
  7. Quit

Although you are creating a menu, this program will not be interactive program. Instead you will use main that is provided below.

Create a function called **menuCall** that will test your program.

**MenuCall** is called from **main**.

**MenuCall** will have one parameter - a string. The first character in the

string is the operation number in the menu, the rest of the string is the other parameters

that are required to make the correct call.

**Pay atention** for **delete a car** we only need the car id. Create an object Automobile with that id, and find if the car with that id is in the data base, if yes, then return the car from the data base.

int main()

{

Dealership dealer;

YOU CAN PUT HERE MORE CODE

out <<”Create the DataBase\n”;

out<<”-------------------------\n”);

out<<endl;

menuCall(“1, carInput.dat”, dealer);

out <<”add a car\n”;

out<<”------------\n”);

out<<endl;

menuCall ( “2, 56789,Mitsubishi, Galant, 25000, 7300”, dealer);

menuCall (“4 , 56789”, dealer);

out <<”delete a car\n”;

out<<”---------------\n”);

out<<endl;

menuCall (“ 3, 23456”, dealer);

out <<”delete a car\n”;

out<<”---------------\n”, dealer);

out<<endl;

menuCall (“ 4 , 23456”, dealer); // if the car is not in the data base, give an appropriate

//message

out <<”retrieve a car and print it\n”;

out<<”-------------------------------\n”);

out<<endl;

menuCall (“4 , 34567”, dealer);

out << “ update the car cost\n”;

out<<”------------------------\n”);

out<<endl;

menuCall (“5 , 45678, 90000”, dealer);

menuCall (“4 , 45678”, dealer);

out <<”Print the DataBase in ascending order \n ”;

out<<”-----------------------\n”);

out<<endl;

menuCall(6, dealer);

out <<”quit\n”;

out<<”-------\n”);

out<<endl;

menuCall(7,delear);

YOU CAN PUT HERE MORE CODE

}

**Evaluation by students**

**Prgram written by:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Program Evaluated by:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Criteria's for evaluations:**

1. Is the program working, check for correct results for any valid input

* Was a correct output created?
* Was the data displayed after the calls to retrieve and update operations?
* Was the data displayed in ascending order?
* Is the total money invested is output?

1. Design - use of existing classes, design of new classes if needed ( not all code is written in one method).

all the functions in **Dealership** are working? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

was a menu in **lab7** created? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **Documentation**: -

* each function is clearly described
* and comments inside the function for clarification

1. Robustness - detection and handling of errors (including user entry of simulation parameters)
2. User interface - clear format of output

6. Use meaningful identifiers