1.124J Foundations of Software Engineering

Problem Set 3

Due Date: Tuesday 10/3/00

Reference Readings: *From C++ Primer*
(in addition to the reference reading from PS1 & PS2):

- *Chapters: 9, 10, 14*

Problem 1:[100%]

In this problem set you have to develop a menu driven program. No files are provided for you, but you are asked to follow a certain class hierarchy. You need to use certain files to implement the latter, acording to the instructions below.

The completed program should be a menu-driven application that allows the user, who may be an employee of a vehicle-rental company, to input information about a fleet of vehicles. Assume that the company rents motor vehicles, in particular cars and motorcycles, and bikes. The user should be able to add a vehicle, display the data and number of the already provided vehicles, save the vehicles to files, and quit the program, after making all the proper memory releases.

The options of the initial menu should be the following:

****

[A]: Add a vehicle

[S]: Show vehicles

[N]: Number of vehicles

[F]: Save vehicles to files

[Q]: Quit

****

The user should be prompted to give one of the above characters, upper or lower case, and the program should proceed accordingly. If any other character is given the program should insist on getting one of the specified characters.

The first option, *[A]: Add a vehicle*, should enable the user to give the information about a vehicle to be added in the database. In order to simplify the problem we ignore all other possible options, such as deletion of a vehicle in the database. However, you should set the pointers correclty, so that someone could extend your program. After the user enters 'a' or 'A' the program should give a new menu from where the user should select the kind of vehicle to be added. This is explained in detail below.

The second option *[S]: Show vehicles*, should display to the screen all the information about the vehicles that have been entered by the user. Details on the exact information to be displayed are provided below.

The third option, *[N]: Number of vehicles*, should display to the screen the numbers of vehicles, motor vehicles, cars, motorcycles, and bikes. An example of the format that is expected to be used is the following:

```
**************************************************

 [A]: Add a vehicle

 [S]: Show vehicles

 [N]: Number of vehicles

 [F]: Save vehicles to files

 [Q]: Quit

**************************************************

Your selection: n


Number of vehicles: 4

 Number of motorvehicles: 3
   Number of cars: 2
   Number of motorcycles: 1

 Number of bikes: 1

Press any button and the <Enter> to proceed...

**************************************************
```

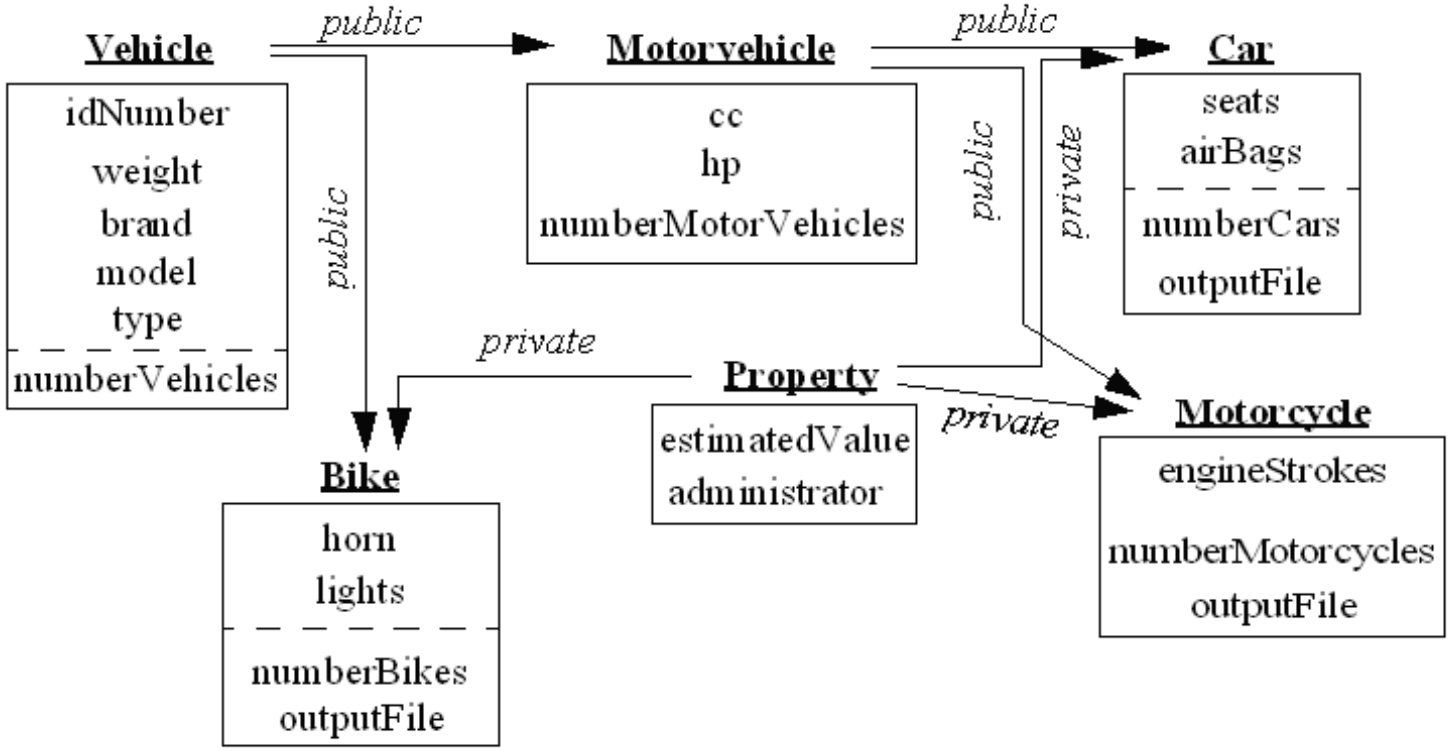
The forth option, *[F]: Save vehicles to files*, should save the information for all cars, motorcycles, and bikes into corresponding to these vehicles files with a certain format

given below.

Finally, the last option, *[Q]: Quit*, should release all the dynamically allocated memory and exit. You NEED to put printout statements in the destructors of the various classes that you will use in order to show that the destructors are visited prior to exiting the program. Make sure that there are no memory leaks.

Class hierarchy:

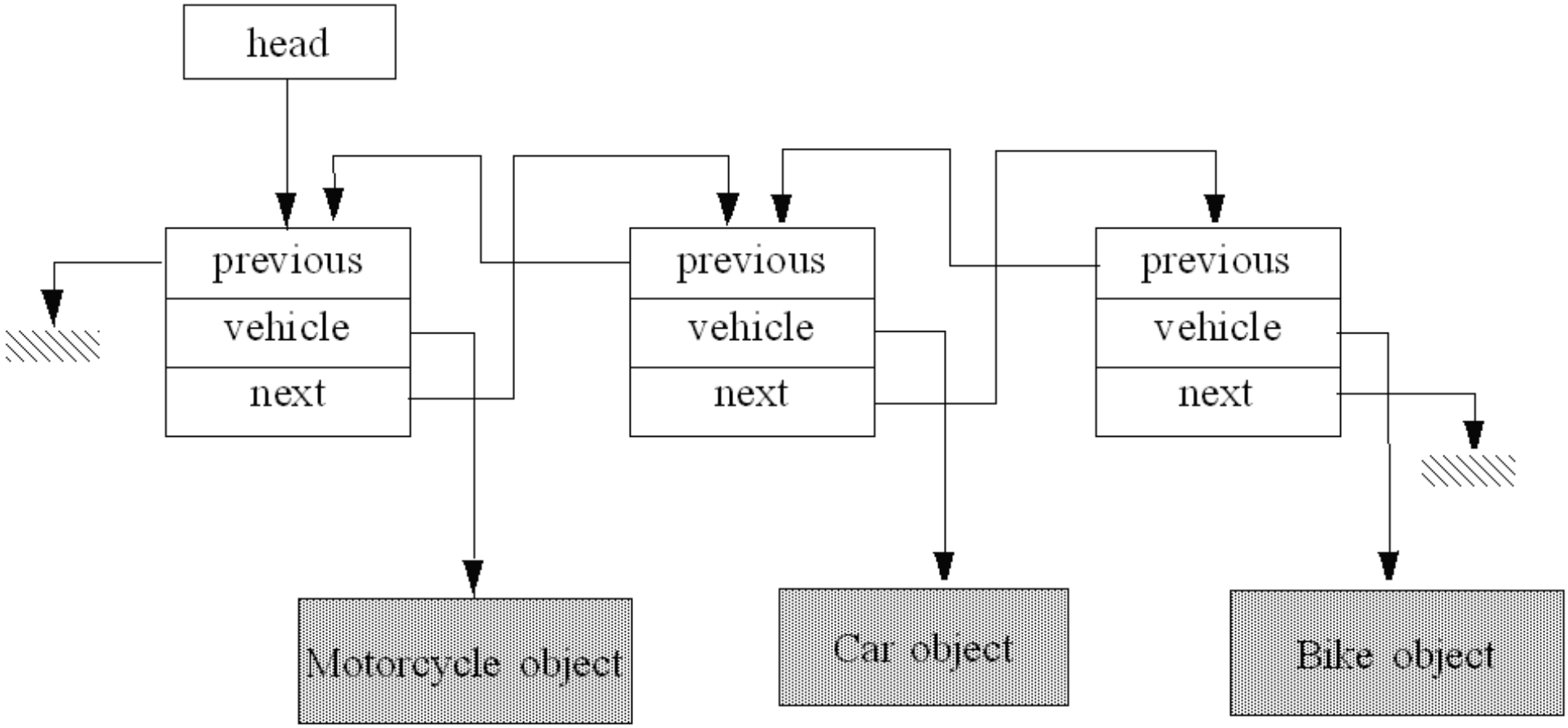
You should use the following class hierarchy, which corresponds to the types of vehicles that your program should be able to handle:



For each class, which will be described in detail next, you need to provide two (2) files, a header file with all the declarations, including the class definitions, and a source code files with all the definitions, such as the externally defined member functions.

Linked list description:

The vehicles should be stored using a linked lists as it is shown in the following figure:



Each new vehicle added by the user should be added to the linked list, using the class *VehicleNode* which has a pointer to a vehicle object, and, therefore, can point to any instance of the class *Vehicle* or any of its subclasses.

In particular, you need to provide the following files with the specified contents:

Contents	**source code file**	**header file**
main() program and general functions	ps3.C	ps3.h
Vehicle class	vehicle.C	vehicle.h
Motorvehicle class	motorvehicle.C	motorvehicle.h
Car class	car.C	car.h
Motorcycle class	motorcycle.C	motorcycle.h

Bike class	bike.C	bike.h
Property class	property.C	property.h
VehicleNode class	vehicleNode.C	vehicleNode.h

Description of each class:

The definition of each class should be provided in the corresponding header file (*.h) according to the above table. All member variables should be specified as **private**. The subclasses should be derived from their superclasses using the class hierarchy that was provided above with the specified derivations (e.g. class *Car* should be publicly derived from the class *Motorvehicle* and privately from the class *Property*). All classes should have destructors which should printout a statement to indicate their use, and release any dynamically allocated memory.

You can provide any member functions you think are necessary in order to achieve the requested functionality. Pointers to char should point to dynamically allocated memory that should be exactly as much characters as needed to store the corresponding string, e.g. the pointer *type*, below, should point to a memory of 7 *char* in order to fit the string *"Enduro"*.

class Vehicle:

This is the base class for all vehicles classes. It should have the following **private** member variables:

- int *idNumber*: ID of the vehicle (e.g. 17394)
- double *weight*: weight of the vehicle (e.g. 96kg)
- char **brand*: brand of the vehicle (e.g. Suzuki)
- char **model*: its model (e.g. DR600R)
- char **type*: its type (e.g. Enduro)
- int *numberVehicles*: current number of vehicles (static variable)

class Property:

This is a superclass of the classes *Car*, *Motorvehicle*, and *Bike*. It should have the following **private** member variables:

- double *estimatedValue*: to store the estimated value of the vehicle
- char **administrator*: to store the name of the person responsible for this vehicle.

class Motorvehicle:

This class should be publicly derived from the class *Vehicle*. It should have the following **private** member variables:

- int *cc*: cubic centimeters of the engine (e.g. 597)
- int *hp*; horsepower (e.g. 67)
- int *numberMotorvehicles*: current number of motor vehicles (static variable)

class Car :

This class should be derived publicly from the *Motorvehicle* class and privately from the *Property* class. It should have the following **private** member variables:

- int *seats*: number of seats
- int *airBags*: number of airbags
- int *numberCars*: current number of cars (static variable)
- ofstream *outputFile*: ofstream static object associated with the file *"cars.dat"* where cars should be stored when the "*[F]: Save vehicles to files"* option is selected and as long as there is at least one car available.

class Motorcycle:

This class should be derived publicly from the *Motorvehicle* class and privately from the *Property* class. It should have the following private member variables:

- int *engineStrokes*: number of strokes of the engine (e.g. 2)
- static int *numberMotorcycles:* current number of motorcycles (static variable)
- static ofstream outputFile: ofstream static object associated with the file *"motorcycles.dat"* where cars should be stored when the "*[F]: Save vehicles to files"* option is selected and as long as there is at least one motorcycle available.

class Bike:

This class should be derived publicly from the *Vehicle* class and privately from the *Property* class. It should have the following **private** member variables:

- bool *horn*: true (false) if it has (not) a horn
- bool *lights*: true (false) if it has (not) lights
- static int *numberBikes*: current number of bikes (static variable)
- static ofstream *outputFile*: ofstream static object associated with the file *"bikes.dat"* where cars should be stored when the "*[F]: Save vehicles to files"* option is selected and as long as there is at least one *bike* available.

class VehicleNode:

This class is used as a node for the linked list. each instance of the *VehicleNode* has a pointer pointing to a *Vehicle* object. You should take advantage of the polymorphic features of C++, e.g. virtual functions to achieve the requested functionality in the most efficient way. The class should have the following private member variables:

- static *VehicleNode* *head: a static variable pointing to the head of the linked list
- VehicleNode **previous*: a pointer to the previous VehicleNode object in the linked list
- Vehicle **vehicle*: a pointer to the vehicle object
- VehicleNode **next*: a pointer to the next VehicleNode object in the linked list

Example of program execution:

Initially the following menu should be provided to the user after cleaning the display from previously presented characters:

****

[A]: Add a vehicle

[S]: Show vehicles

[N]: Number of vehicles

[F]: Save vehicles to files

[Q]: Quit

****
Your selection: a

If the user selects to add a vehicle the program should give the following menu in order to allow the user to select what kind of vehicle to enter to the system:

****

[C]: Add a car

[M]: Add a motorcycle

[B]: Add a bike

[Q]: Quit operation

****
Your selection: m

Let's say that the user enters the character *m*, as above in order to add a motorcycle. Then, the user should be prompted to provide all the relevant information, e.g.:

****
Your selection: m
Adding a motorcycle....

Please give all the relevant information

Vehicle ID = 17
Weight = 96

Brand = Suzuki
Model = Dr600R
Type = Enduro

CC = 597
hp = 67

Estimated value = 2450.50
Administrator = Wilson

Number of strokes = 4

Press any button and the <Enter> to proceed...

****

The initial menu should be provided again. Then, the program should return to the initial menu giving the option to the user to select where to proceed. Let's assume that the user decides to enter a car:

****

[C]: Add a car

[M]: Add a motorcycle

[B]: Add a bike

[Q]: Quit operation

****
Your selection: c
Adding a car....

Please give all the relevant information

Vehicle ID = 33
Weight = 3675.5
Brand = Honda
Model = civic
Type = family

CC = 2350
hp = 136

Estimated value = 26550
Administrator = Edwards

Number of seats = 5
Number of airbags = 1

Press any button and the <Enter> to proceed...

Then, lets assume that the user decides to enter a bike:

****

[C]: Add a car

[M]: Add a motorcycle

[B]: Add a bike

[Q]: Quit operation

****
Your selection: b
Adding a bike....

Please give all the relevant information

Vehicle ID = 47
Weight = 24.5
Brand = BMX
Model = X12
Type = mountain

Estimated value = 325.75
Administrator = Thomas

Horn [Y/N] = y

Lights [Y/N] = n

Press any button and the <Enter> to proceed...

****

If the user at this point decides to select the option *"[N]: Number of vehicles"* to show on the display the number of vehicles, she/he will get the following:

****

[A]: Add a vehicle

[S]: Show vehicles

[N]: Number of vehicles

[F]: Save vehicles to files

[Q]: Quit

****
Your selection: N

Number of vehicles: 3
Number of motorvehicles: 2
Number of cars: 1
Number of motorcycles: 1
Number of bikes: 1

Press any button and the <Enter> to proceed...

****

If the user decides to select the option, [S]: Show vehicles, in order to display the vehicles the program should display something as the following providing all the relevant information and printing all vehicles in the linked list with an increasing number, starting from the lastly provided vehicle at 1, as below:

****

[A]: Add a vehicle

[S]: Show vehicles

[N]: Number of vehicles

[F]: Save vehicles to files

[Q]: Quit

****
Your selection: s

Showing vehicles' data...

Vehicles in the list

Vehicle 1: Bike
ID number= 47 Brand = BMX Model = X12 Type = mountain Weight = 24.5
Has Horn - Has No Lights
Estimated Value = 325.75 Administrator = Thomas

Vehicle 2: Car
ID number= 33 Brand = Honda Model = civic Type = family Weight = 3675.5
CC = 2350 Horsepower = 136
Seat = 5 Airbag = 1
Estimated Value = 26550 Administrator = Edwards

Vehicle 3: Motorcycle
ID number= 17 Brand = Suzuki Model = Dr600R Type = Enduro Weight = 96
CC = 597 Horsepower = 67
Engine strokes = 4 Estimated Value = 2450.50 Administrator = Wilson

---

If the user decides to select the *"[F]: Save vehicles to files"* option, then the program should:

- save the cars, only if there are any, into the file *"cars.dat"*, which is associated with the static *outputFile* variable of the class *Car*,
- save the motorcycles, only if there are any, into the file *"motorcycles.dat"*, which is associated with the static *outputFile* variable of the class *Motorcycle*,
- save the bikes, only if there are any, into the file *"bikes.dat"*, which is associated with the static *outputFile* variable of the class *Bike*,

****

[A]: Add a vehicle

[S]: Show vehicles

[N]: Number of vehicles

[F]: Save vehicles to files

[Q]: Quit

****

Your selection: f
Vehicles are stored in individual files

Press any button and the <Enter> to proceed...

The files should be created only if there are vehicles, at least one, of the corresponding type.

The first line of the files should describe what it will be saved below it for each of vehicle of that kind as follows:

"cars.dat"

Brand Model Type ID Weight CC HP Seats Airbags Value Administrator
---
1 ....
2 .... *etc.*

"motorcycles.dat"

Brand Model Type ID Weight CC HP Strokes Value Administrator

---

1 ....
2 .... etc.

"bikes.dat"

Brand Model Type ID Weight Horn Lights Value Administrator

---

1 ....
2 .... etc.

Finally, if the user selects to quit the program should exit after releasing all the dynamically allocated memory. The printout statements in the destructors should display similar to what follows:

****

[A]: Add a vehicle

[S]: Show vehicles

[N]: Number of vehicles

[F]: Save vehicles to files

[Q]: Quit

****

Your selection: q

Press any button and the <Enter> to proceed... d
releasing memory

Destroying a VehicleNode object

Deleting a Bike object
Deleting a Property object
Deleting a Vehicle object

Destroying a VehicleNode object
Deleting a MotorCycle object
Deleting a Property object
Deleting a Motorvehicle object
Deleting a Vehicle object

Destroying a VehicleNode object
Deleting a Car object
Deleting a Property object
Deleting a Motorvehicle object
Deleting a Vehicle object
Memory has been released

You should take advantage of object oriented programming to avoid repeating the same code at different places whenever possible. You should follow the above directions about the class hierarchy, the requirements for each class, the behavior of the menu driven application and not simplify the problem in any way.

You also need to provide a makefile named *makePS3* that can be used to compile your program. The name of the executable should be ***ps3***. You need to submit both hardcopies of your code and electronically turn in all files, including the makefile, which will also be graded to check that it works with the following command:

gmake -f makePS3 ps3
