Computational Problem Solving II

CPET-321

Design Project : Quidditch Team Carpool Reservation System

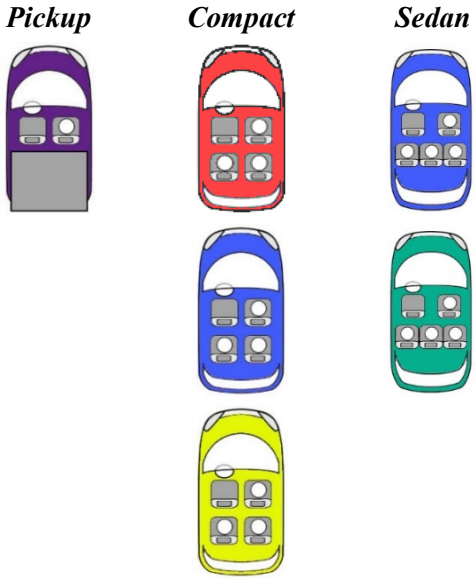
Design Overview:

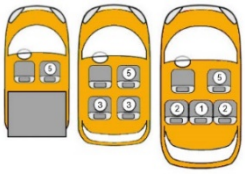
Your task is to design, code and fully test a carpool reservation system for RIT's Quidditch Team. Being a non-sanctioned NCAA sport (duh), the members of the Quidditch team must carpool to the regional tournaments. Amongst the 24 members, six (6) have cars and have agreed to provide transportation for the remaining 18 members.

These six team members will drive the following cars:

- One, 1-Passenger Pickup Truck
  - Total of (1) Available Seat
- Three, 3-Passenger Compact Cars
  - Total of (9) Available Seats
- Two, 4-Passenger Sedan Cars
  - Total of (8) Available Seats

Between the six vehicles, there are a total of 18 (1 + 9 + 8) available seats, the exact number needed to accommodate all remaining members of the Quidditch team.





To minimize the whining about who gets to ride with who and who gets to sit where in each vehicle (note: Quidditch players are notorious complainers ☺), all team members have agreed to use the carpool reservation system. Reservations are made on a week-to-week bases and are taken on a first-come first-serve bases.

At the beginning of the Quidditch season, each of the eighteen passengers are allocated 20 "seat credits". These credits are utilized to "purchase" a seat in a specific car each week of the eight week season. The value of each seat is determined by the following scheme:

- All front seats are worth five (5) points each.
- Back seats in compact cars are worth three (3) points each.
- Side-back seats in sedans are worth (2) points and middle-back seats are worth (1).
- See the ***Seat Value*** diagram shown above.

The design of your carpool reservation system must meet the following ***Design Specifications*** and ***Coding Requirements***.

***Design Specifications*:**

- The names of the twenty-four (24) team members are stored in a data file named **quidditch_team.dat**:

quidditch_team.dat

```
1 Pat Cooper
2 Jane Cox
3 Tim Taylor
4 Ben Butler
5 Art Campbell
6 Ann Edwards
7 Pam Allen 4
8 Tony James 4
9 Ron Jonesn 9
10 Will Rogers 1
11 Don Williams 3
12 Doug Long 2
13 Randy Carter 6
14 Adam Jenkins 8
15 Dan Scott 1
16 Ed Patterson 7
17 Sarah Ward 0
```

  - The first six names in the file are the names of the drivers for the following cars (in order):
    - Purple Pick-Up
    - Red Compact
    - Blue Compact
    - Yellow Compact
    - Blue Sedan
    - Green Sedan
  - The remaining names are those of the 18 members of the Quidditch team who will need to make a reservation. The number adjacent to the name are the player's remaining seat credits.
  - This file is posted on myCourses.
- The **quidditch_team.dat** data file should be read by the program when it starts and re-saved prior to the program ending. The information regarding the number of credits for each member must be maintained in an internal database and updated as members make, change, or delete reservations.
- The reservation system user interface should be menu driven and have the following options:
  - ***Create*** (C) a reservation:
    - Enter the passenger's name (first last). If the name is NOT in the database, print an error message and return to the menu.
    - If the name is in the database, report the number of seat-credits the passenger has remaining.
    - If the passenger has zero credits, they cannot make a reservation and must provide their own transportation. Print an error message and return to the menu.
    - After displaying a simple diagram showing the seat availability (see below), have the user select a seat.

```
  Truck    Compact      Sedan
 -------  -------     -------
 PURPLE     RED        BLUE
 (-)(5)   (-) (5)    (-)   (X)
          (X) (X)    (2)(1)(2)

            BLUE       GREEN
          (-) (X)    (-)   (5)
          (3) (X)    (X)(1)(2)

           YELLOW
          (-) (X)
          (X) (3)
```

In this diagram a (-) indicates the driver's seat, an (X) indicates the seat has been taken and a number (1,2,3, & 5) indicates the seat is available and the number of credits required to reserve this seat.

Note: Your diagram does not need to look exactly like this. Be creative, make it interesting and UNDERSTANDABLE.

    - The passenger can select a seat by category or by a specific seat in a specific vehicle. If they select by category, for example a front-seat, they will be assigned a front seat in the first available vehicle. Otherwise, they need to select

a specific seat in a specific vehicle. For example, Green Sedan, middle back-seat. Note: For seats with common positions and value (i.e. back seats in compact cars or side-back seats in sedans) the passenger does not (cannot) select the specific side.

- If the selected seat is available and the member has the required credits, complete the reservation, otherwise print an error message and re-prompt for a seat selection.
- Once the reservation process is complete, assigned a two-digit reservation number (0-18) to the reservation.

- ***Modify*** a reservation:
  - Enter the reservation number. If the reservation number exists, allow the user to reselect a different vehicle and seat. Be sure to update all the appropriate databases.
  - If the reservation number does NOT exist, print an error message and return to the menu.
- ***Delete*** a reservation:
  - Enter the reservation number. If the reservation number exists, delete the reservation and update all the appropriate databases.
  - If the reservation number does NOT exist, print an error message and return to the menu.
- ***Display*** vehicles:
  - Displays the seat assignment diagram (see above).
- ***Print*** vehicle assignments:
  - After selecting a specific vehicle (i.e. purple pickup, blue compact, green sedan, etc.) the passenger list for that vehicle will be displayed to the screen and printed to a data file. The file's name should be keyed to the vehicles description (for example **red_sedan.txt**).
- ***Reservation*** print:
  - This feature prints a complete passenger list (i.e. manifest) for the six vehicles to a file named **all_reservations.txt**. The manifest file contains a vehicle descriptor (i.e. purple pickup, blue compact, green sedan, etc.) the drivers name and a list of the passengers' names and their seat location. If the seat is not assigned, print "Unassigned".
  - This feature is only available to the system administrator and is password protected. The password is hardcoded into the program and cannot be changed. If the password is entered incorrectly, print an error message and return to the menu.

Code Requirements:

- Given the complexity of this project, your design must utilized a hierarchical implementation with one, or more, user defined libraries containing your code. These libraries will be included into the main program by means of a header file(s)
- The principle objective of your design (other than a functional reservation system) is to exploit the features of OOP. While the specifics will depend on your design, your code must utilize encapsulation and class inheritance.
    - For encapsulation, you may consider creating a class called **reservation**. When a reservation is created, an object of the class **reservation** would be instantiated. Thus you could make the functions that create, modify or delete a reservation member-functions of the class **reservation**.
    - For inheritance, you could create a parent class **Vehicle** and three children-classes, one for each of the car types (**pickup-truck**, **compact** and **sedan**).
- The specific data structures utilized for the databases used in the reservation system (i.e. array, vectors, etc.) are up to your discretion.

Grading:

Your grade for this assignment will be determined by the following factors:

- Adherence to the design specifications (40%)
- Code quality (i.e. style, commenting, etc.) (20%)
- Design Milestones (20%)
    - Milestone #1 : 9/23 or 9/25
        - IDE selection
        - Team meeting times & work assignments
        - Preliminary design & UML diagrams
    - Milestone #2 : 9/30 or 10/3
        - Code review with team & instructor.
        - Code should be functional with a majority (~75%) of the design specification implemented
- Class Presentations - Formal Code Review (10%) (10/7 or 10/9)
- Team-Members peer evaluations (10%)

Documentation & Due Date:

- All ****.cpp, *.h*** and data files. Be sure these files are properly commented and formatted in accordance to the ***C++ Style Guide***. The source code MUST be self-contained and self-explanatory. Do not assume the reader has ever read this handout.
- Formal presentation (PPT) documenting your design process final code.
- All documentation is due in the Design Problem #1 dropbox (assignments) on myCourses by midnight on **Tuesday 10/6.**