Winter Semester 2020, Final Project

Seneca Valet Application

Milestone 2 - Parking

V0.9

(Please watch the project repo for possible corrections)

Milestone 2:

Read the whole document before starting to implement milestone 2.

The Parking module:

The Parking class overview:

Parking module is the skeleton of your final project and will be revisited at the final stage of the application development. In milestone 2 you will build the main structure of the application that gives the user all the options needed for managing a valet parking.
For now, when these options are selected you will only print the name of the action. In following milestones, you will develop the facilities needed to actually provide the requested actions by the user.

The **Parking** module runs as follows:
It loads a data file from the hard disk that holds the information of the parked cars in the **Parking**. *(the actual loading will be implemented in the last milestone)* Then it will display the following menu:

```
Parking Menu, select an action:
1- Park Vehicle
2- Return Vehicle
3- List Parked Vehicles
4- Close Parking (End of day)
5- Exit Program
>
```

If user selects the first option a submenu as follows with one indentation to right will be displayed:

```
    Select type of the vehicle:
    1- Car
    2- Motorcycle
    3- Cancel
    >
```

For now, print the following messages `(in Red)` based these selections `(in Green)`:

```
Park Vehicle / Car:
    "Parking Car"
```

```
Park Vehicle / Motorcycle:
    "Parking Motorcycle"

Park Vehicle / Cancel:
    "Cancelled parking"

Return Vehicle:
    "Returning Vehicle"

List Parked Vehicles:
    "Listing Parked Vehicles"

Close Parking (End of day):
    "Closing Parking"
```

Selecting this option ends the parking application for the day.

```
Exit Program:
```

If exit program is selected first display the message:

```
    "This will terminate the program!"<NEW LINE>
    "Are you sure? (Y)es/(N)o: "
```

And wait for the user to enter either “Y” or “N” (lowercase or uppercase) and exit if the response is yes.
If user enters an invalid response then print:

```
    "Invalid response, only (Y)es or (N)o are acceptable, retry: "
```

And repeat until proper response is entered.

To accomplish the above Parking module functionality, implement the following:

The Parking class implementation:

Properties: (member variables and objects)

Filename:
Parking holds a **C-style** string with unknown size to hold the data file name of the application.

Parking menu:
Parking has a **Menu** object for the main menu of the application.
Populate the **Parking Menu** with the following *(see constructor for more detail)*:
Indentation:

```
    0
```

Parking Menu Title:

```
    "Parking Menu, select an action:"
```

Parking Menu Items:

```
    "Park Vehicle"
    "Return Vehicle"
```

```
"List Parked Vehicles"
"Close Parking (End of day)"
"Exit Program"
```

Vehicle Selection menu:

Parking has a **Menu** object for the **Vehicle selection menu** that is displayed as a submenu when option one of the **Parking Menu** is selected.
Populate the Vehicle Menu with the following *(see constructor for more detail)*:
Indentation:

```
1
```

Vehicle Menu Title:

```
"Select type of the vehicle:"
```

Vehicle Menu Items:

```
"Car"
"Motorcycle"
"Cancel"
```

Other properties:
Add other properties if, or when needed.

Constructor implementation:

- Parking can be created by a constant **C-style** character string with unknown size, to be used to set the name of the data file.
If the string is invalid (**null**) or empty then **Parking** is set to an **invalid empty state**.
Call the **Load Data File** function *(see list of member functions)* and if it returns true, then populate the **Parking and Vehicle menus**.
If **Load Data File** function returns false, then print:

```
"Error in data file"<NEWLINE>
```

and set the class to an **invalid empty state**.

Destructor implementation:

- Call the **Save Data File** function *(see list of member functions)*, and Make sure there is no memory leak when the **Parking** goes out of scope.

Copy and Assignment:

Copy and **assignment** are **denied** in the Parking class. (**Parking** class can not be copied or assigned to another **Parking** object)

Private Member function implementations:

- **isEmpty() function**
  Write an **isEmpty** function that returns **true** if the **Parking** is in an **invalid empty State** or **false** if parking is valid and usable.

- **Parking Status function**
  This function does not receive or return anything and only prints:
  `"****** Seneca Valet Parking ******"<NEWLINE>`
- **Park Vehicle function**
  This function does not receive or return anything.
  The function displays the submenu **Vehicle Selection menu** and then based on the users selection; **Car**, **Motorcycle** or **Cancel**, will print `"Parking Car"`, `"Parking Motorcycle"` or `"Cancelled parking",` respectively. Then it goes to new line.
- **Return Vehicle function**
  This function does not receive or return anything and only prints:
  `"Returning Vehicle" <NEWLINE>`
- **List Parked Vehicles function**
  This function does not receive or return anything and only prints:
  `"Listing Parked Vehicles" <NEWLINE>`
- **Close Parking function**
  This function does not receive anything and returns a Boolean. Print the following:
  `"Closing Parking" <NEWLINE>`
  and then return true.
- **Exit Parking App Function**
  This function does not receive anything and returns a Boolean. Print the following:
  `"This will terminate the program!"<NEW LINE>`
  `"Are you sure? (Y)es/(N)o: "`
  And wait for the user to enter either "Y" or "N" (lowercase or uppercase) and exit if the response is yes or no.
  If user enters an invalid response then print:
  `"Invalid response, only (Y)es or (N)o are acceptable, retry: "`
  and repeat until proper response is entered.
  At the end return true, if the user's response is Yes and false if the user's response is No.
- **Load Data File function**
  This function does not receive anything and returns a Boolean.
  If the **Parking** is not in an **invalid empty state**, Print the following:
  `"loading data from "`
  then print the name of the **data file**, `<NEWLINE>` and finally return true.
  If the Parking is in an invalid empty state, return false.
- **Save Data File function**
  This function does not receive or return anything and if the **Parking** is not in an **invalid empty state** it prints:
  `"Saving data into "`
  then prints the name of the **data file** and a `<NEWLINE>`.

The Run Public Member function implementation:

- `int Parking::run();`
  Run is the only public member function in **Parking** and runs the whole Parking Application exactly as stated in **The Parking class overview** section using the following functions calls and logic:

  If the **Parking** is not in an **invalid empty state**:
  A- The **Run** function calls the **Parking Status function** and then displays the **Parking menu** and waits for the user's response.
  B- If options **1**, **2** or **3** are selected, it will call one of the **Park Vehicle**, **Return Vehicle** or **List Parked Vehicles** functions respectively and goes back to stage **A**.
  C- If option **4** is selected it will call **Close Parking** function. If **Close Parking** Function returns true it will exit and end the **Run** function. Otherwise it goes back to step **A**.
  D- If option **5** is selected it will call **Exit Parking App** function and exits the **Run** function only if a true value is returned. Otherwise it goes back to step **A**.

  When **Run** exits, it should return **1** if it the **Parking** is in an invalid empty state, otherwise it should return **0**.

  > *Note: As you see options **4** and **5** can both exit the program. In future implementations there will be a difference between the two.*
  > ***Closing Parking** is an **end of the day** action; which results in towing all the remaining vehicles out of the parking, since this is not an overnight parking application.*
  > ***Exiting program** acts like a pause in the application execution. In this case, the program (in future implementations) will save all its information in a data file and when it is executed again it will load all the data back and continue where it left off las time.*

Other member functions:

- Add other member functions to the **Parking** if needed.

Milestone 2 Duedate:

This milestone is due by Tuesday March 17; 23:59;
~profname.proflastname/submit 200/NXX/MS2/parking -due<ENTER>

Milestone 2 submission:

To test and demonstrate execution of your program use the data provided in the execution sample below.

If not on matrix already, upload **Menu.cpp**, **Menu.h, Parking.cpp, Parking .h , ms2_ParkingApp.cpp** programs to your matrix account.
Compile and run your code and make sure that everything works properly.
Then, run the following command from your account (use your professor's Seneca userid to replace profname.proflastname, and your section ID to replace NXX, i.e., NAA, NBB, etc.):

```
~profname.proflastname/submit 200/NXX/MS2/Parking <ENTER>
```

and follow the instructions generated by the command.

IMPORTANT: Please note that a successful submission does not guarantee full credit for this workshop. If the professor is not satisfied with your implementation, your professor may ask you to resubmit. Resubmissions will attract a penalty.