Winter Semester 2020, Final Project

Seneca Valet Application

Milestone 1 - Menu

V0.91 (correct submission command)

Your task for the project for this semester is to create an application that keeps track of a Valet Parking that can park Cars, Motorcycles and Busses in a parking and retrieve them back when requested.

Milestones and due dates

The project will be done in minimum of four milestones and each milestone will have its own due date. The due date of each milestone will be announced when it is published, and it is based on the amount of work to be done for that milestone.

Citation, Sources

When submitting your work, all the files submitted should carry full student information along with a citation and sources information as follows.

```
/* Citation and Sources...
Final Project Milestone ?
Module: Menu
Filename: Menu.h
Version 1.0
Author John Doe
Revision History
-----------------------------------------------------------
Date      Reason
2020/?/?  Preliminary release
2020/?/?  Debugged DMA
-----------------------------------------------------------
I have done all the coding by myself and only copied the code
that my professor provided to complete my workshops and assignments.
-----------------------------------------------------------
OR
-----------------------------------------------------------
Write exactly which part of the code is given to you as help and
who gave it to you, or from what source you acquired it.
-----------------------------------------------------------*/
```

Compiling and testing your modules

Tester programs are provided with the milestones. Your code along with the teste programs should be compiled using this command on matrix:

g++ -Wall -std=c++11 -o ms

After compiling your code and testing the execution make sure the executable is tested as follows to check for possible memory leaks:
valgrind ms <ENTER>

Project Implementation notes: *Very Important, read carefully*

- All the code written in this project should be within the namespace **sdds**.
- You are free and encouraged to add any member variables or member functions you find necessary to complete your code. If you are not sure about your strategy for adding functionalities and properties to your classes, ask your professor for advice.
- A module called **Utils** is added to the project so, you can add your own custom code to the project. **Utils.h** will be included to all the testers of the milestones. Leave these files empty but available if you don't need to add any additional code.
- Unless you are asked for a specific definition, name the variables and functions yourself. Use proper names and follow the naming conventions instructed by your professor. Having meaningless and misleading names will attract penalty.
- Throughout the project, if any class is capable of displaying or writing itself, the member function will always have the following signature:
  The function will return a reference of an **ostream** and will receive a reference on an **ostream** as an optional argument. If this argument is not provided, the object "**cout**" will be passed instead.
- Throughout the project, if any class is capable of reading or receiving its content from a stream, the member function will always have the following signature:
  The function will return a reference of an **istream** and will receive a reference on an **istream** as an optional argument. If this argument is not provided, the object "**cin**" will be passed instead.
- When creating methods (member functions) make sure to make them constant if in their logic, they are not modifying their class.
- When passing an object or variable by address or reference, if they are not to be modified, make sure they are passed as constant pointers and references.
- An Empty state for an object is considered to be an "invalid" empty state and objects in this state should be rendered unusable.

Milestone 1:

The Menu module:

Menu module consists of two classes, **MenuItem** and **Menu** (these classes are both written in Menu.h and Menu.cpp)

Purpose:

Menu displays a menu title and several menu items with a row number attached to each, and lets the user make a selection. After user makes the selection, the row number of the selected menu item is returned to the caller program.

The MenuItem class:

Create a fully private class (no public members) called **MenuItem** that is owned by the **Menu** class. (i.e., **Menu** is a friend of **MenuItem**).

To make this possible (since Menu is not yet implemented), forward declare the **Menu** class to be able to make it a friend of **MenuItem** class.

MenuItem can hold a C-style character string with unknown length as its content.

Construction:

MenuItem can be created by a constant C-style character string to be used to set its content. If the provided string is invalid (null), then the object is set to an empty state.

Member function:

The **MenuItem** can display itself by writing its content on the screen and printing a new line.

If **MenuItem** is in an empty state, nothing is printed.

Copy and Assignment:

MenuItem cannot be copied or assigned to another **MenuItem.** This must be enforced in your code.

Destruction:

MenuItem has a destructor to make sure there are no memory leaks.

Other member functions:

Add other members to **MenuItem** if needed during the development of this module

The Menu class:

Before starting to implement the Menu class, create a constant int value in the Menu module called `MAX_NO_OF_ITEMS` and set it to **10**.

Properties: (member variables)

Title:
Menu item holds a C-styles string with unknown size to hold the title of the menu.

MenuItems:
Menu holds an array of **MenuItem** pointers with size of `MAX_NO_OF_ITEMS`. Throughout the program, as menu items are added to the **Menu**, these pointers are set to dynamically allocated individual **MenuItems** for each addition.
Make sure you keep track of the number of added **MenuItems** to the **Menu**.

Indentation:
when displaying the menu, the title and menu items may be indented to the right. For each indentation level (value of 1) indent the menu 4 spaces to the right.

Menu example without indentation (value of 0):

```
** The Menu **
1- Option one
2- Option Two
3- Option three
4- Option four
5- Exit
>
```

Menu example with indentation value of 2:

```
        ** The Menu **
        1- Option one
        2- Option Two
        3- Option three
        4- Option four
        5- Exit
        >
```

Other properties:
Add other properties if or when needed.

Construction:

Menu can be created by a constant C-style character string with unknown size, to be used to set its title and an optional integer value for indentation. If indentation is not provided, the value “0” will be passed instead.
if the string is invalid (**null**) then **Menu** is set to an invalid empty state.

Copy and Assignment:

Menu Should be safely copied and assigned to another menu.

Member functions, operator and cast overloads:

- **bool cast overload**
  overload the cast operator so if the **Menu** object is casted to a **bool**, it should return **true** if the **Menu** is valid and usable, otherwise it should return **false** if the Menu in an invalid empty state.

- **isEmpty() function**
  Write an **isEmpty** function that returns **true** if the **Menu** is in an invalid empty State or **false** if it is valid and usable.
  *Note that **isEmpty** works the opposite of the **bool cast overload**.*

- **display function**
  The **Menu** should **display** itself as follows:
  If the Menu is in an invalid empty state, it will print:
  `"Invalid Menu!"` and a new line
  If the Menu has no MenuItems, it will print:
  `"No Items to display!"` and a new line
  Otherwise:
  First it will print its **title** and go to new line.
  Then it will display **a row number**, starting with number one (**1**) and a **dash**.
  Afterwards it will print a **space** and then the first **MenuItem** in the array.
  It will continue printing **the row numbers** and the **MenuItems** to the number of menu items added to the **Menu**.
  At end it will print a **greater-than operator** (“**>**”) and a **space** as the prompt for user selection.

  Menu example:
  ```
  ** The Menu **
  1- Option one
  2- Option Two
  3- Option three
  4- Option four
  5- Exit
  >
  ```

- A **Menu** should be able to be **assigned to a constant character C-string**. This should reset the title of the **Menu** to the newly assigned string. If the string is invalid (**null**) the **Menu** is set to an invalid empty state.
  Assignment example:
  ```
  Menu lunchMenu("Lunch Menu");
  lunchMenu = "New Title"; //<<---- Assignment
  ```

- **add function:**
  write an **add** function that returns nothing and receives a **constant character C-string** to build a **MenuItem** and add it to the **MenuItems** of the **Menu**.
  If the string is invalid (**null**) then the **Menu** is set to an invalid **empty** state and nothing is added.
  The **add** function should create a dynamic **MenuItem** out of **the C-string argument** and add it the array of **MenuItem pointers**.
  If the array is full, then the function should silently ignore the addition and exit.
  Note that the **add** function will do nothing and exit silently if the **Menu** is in an invalid empty state.

- **Insertion operator overload ("<<")**
  Overload the **insertion operator** to insert a **constant character C-string** into the **Menu** as a **MenuItem**. This overload should work exactly like the **add** function and work as follows:
  Insertion example to add three MenuItems to the lunchMenu:

```
Menu lunchMenu("Lunch Menu");
lunchMenu << "Hamburger" << "Chicken Sandwich" << "Pasta and meatballs";
```

  Displaying the above menu should result the following:

```
Lunch Menu
1- Hamburger
2- Chicken Sandwich
3- Pasta and meatballs
>
```

- *Insertion example to add invalid MenuItems to the lunchMenu:*

```
Menu lunchMenu("Lunch Menu");
lunchMenu << nullptr << "Chicken Sandwich" << "Pasta and meatballs";
```

  Displaying the above menu should result the following:

```
Invalid Menu!
```

- **Menu class in action (run function):**
  Create a function called **run** that returns an integer. In this function **display**s the **Menu** and waits for the user's response to select an option by entering the row number of the **MenuItems**.
  If the user enters non-integer value print the following error message:
  `"Invalid Integer, try again: "`
  and wait for the user's response.
  If the user enters an integer, but out of the boundary of the available options print the

following error message:
`"Invalid selection, try again: "`
and wait for the user's response.
When user selects a valid option, end the function and return the selected option value.
If the **Menu** has no **MenuItem**s, no selection is made and **0** returned with no user interaction.

Run function example

```
Menu lunchMenu("Lunch Menu");
lunchMenu << "Hamburger" << "Chicken Sandwich" << "Pasta and meatballs";
int choice = lunchMenu.run();
cout << "You selected " << choice << endl;
```

- **Integer cast overload**
  After creating the **run** function, **overload the cast operator** so if the **Menu** object is casted to an integer, the **run** function is called, and its value is returned as the integer cast value.

  Integer cast example

```
Menu lunchMenu("Lunch Menu");
lunchMenu << "Hamburger" << "Chicken Sandwich" << "Pasta and meatballs";
int choice = lunchMenu;
cout << "You selected " << choice << endl;
```

Destruction:

Menu has a destructor to make sure there are no memory leaks.

Other member functions:

Add other member functions to the **Menu** if needed.

Milestone 1 Duedate:

This milestone is due by Thursday March 12 @ 23:59.
~profname.proflastname/submit 200/NXX/MS1/menu -due<ENTER>

Milestone 1 submission:

To test and demonstrate execution of your program follow the instructions in the tester program.

If not on matrix already upload **Menu.cpp**, **Menu.h, ms1_MenuTester.cpp**, **Utils.cpp**, **Utils.h** 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/MS1/menu <ENTER>
```

and follow the instructions generated by the command and your program.

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.

Milestone one tester program:

The tester program and the expected output have been supplied as separate files in the repository. **Do not modify these files in any way!**