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.
- As usual 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.

**Commented [CB1]:** This is a resource that must be managed.

**Commented [FS2R1]:** I don't understand what you mean by this

## 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.

**Commented [CB3]:** This is misleading: suggest that the function always print to screen, while in the implementation it has a parameter where to do the printing.

**Commented [FS4R3]:** I don't know if I understand you correctly, but this means "prints itself wherever the cursor is".
Empty state is added, No other explanation is necessary.

## Copy and Assignment:

**MenuItem** can not get 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

**Commented [CB5]:** Later you say "out of boundary" integer – isn't the same thing?

**Commented [FS6R5]:** I meant non-integer value like "twenty"

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 Tuesday March 10; 23:59;
**~profname.proflastname/submit 244/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 244/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:

```
/* ----------------------------------------------------------
Final Project Milestone 1
Module: Menu
Filename: ms1_MenuTester.cpp
Version 1.0
Author Fardad Soleimanloo
Revision History
-----------------------------------------------------------
Date      Reason
2020/3/2  Preliminary release
-----------------------------------------------------------*/

#include <iostream>
#include "Utils.h"
#include "Menu.h"
using namespace std;
using namespace sdds;
int showSelection(int n);
void prompt(const char* message);
void pause();
void testMenus(Menu m, Menu sub1, const Menu& sub2);
void TT(const char* title);
int main() {
   TT("M1T1: constructors");
   Menu mainMenu("Main Menu");
   Menu subMenu1("Sub Menu One", 1);
   Menu subMenu2("Sub Menu", 2);
   Menu tempMenu("Temp");
   Menu invMenu("To test Invalid Menu");


   TT("M1T2: operator<< adding MenuItems");
   tempMenu << "One" << "Two" << "three";
```

```
   TT("M1T3: Menu::run()");
   prompt("Do the following: \nabc <ENTER>\n0 <ENTER>\n4 <ENTER>\n3 <ENTER>");
   cout << tempMenu.run() << " selected!" << endl;
   pause();

   TT("M1T4: Menu deep assignment with no items");
   tempMenu = subMenu2;

   TT("M1T5: Reseting Menu title and running an empty menu with no Items");
   tempMenu = "New Title"; //<<---- Assignment
   cout << tempMenu.run() << " returned by the Menu with no Items." << endl;
   pause();

   TT("M1T6: Menu::add() and operator<<");
   mainMenu.add("Option one");
   mainMenu.add("Option Two");
   mainMenu.add("Sub Options");
   mainMenu.add("Option four");
   mainMenu.add("Exit");
   subMenu1 << "Selection one" << "Sub Selections two" << "Selection three" << "Selection
four";
   subMenu2 << "The first" << "The second" << "The third";

   TT("M1T7: Deep assignment");
   prompt("Do the following: \n1 <ENTER>");
   tempMenu = subMenu2;
   cout << tempMenu.run() << " selected!" << endl;
   pause();


   TT("M1T7: Deep copying, passing const refrence and indentation display");
   testMenus(mainMenu, subMenu1, subMenu2);

   TT("M1T8: operator bool()");
   if (mainMenu) {
      cout << "The mainMenu is valid and usable." << endl;
   }

   TT("M1T9: Invalid Menu and its usage");
   mainMenu.add(nullptr);
   if (!mainMenu) {
      cout << "The mainMenu is invalid(empty) and not usable." << endl;
   }
   mainMenu.run();
   mainMenu.display();


   TT("M1T10: Assigning Menu to an empty Menu");
   subMenu1 = mainMenu;
   subMenu1.run();
   subMenu1.display();


   TT("M1T11: Adding invalid string to render Menu invalid");
   invMenu << nullptr << "This should not be added" << "This shouldn't be added either";
   invMenu.run();
   invMenu.display();
   return 0;
```

```
}
int showSelection(int n) {
   cout << "You selected " << n << "." << endl;
   return n;
}

void prompt(const char* message) {
   cout << message << endl;
}
void pause() {
   cout << "Press enter to continue..." << endl;
   cin.ignore(1000, '\n');
}
void testMenus(Menu m, Menu sub1, const Menu& sub2) {
   int selection;
   prompt("Do the following:\n2 <ENTER>\n3 <ENTER>");
   while ((selection = m.run()) != 5) {
      showSelection(selection);
      if (selection == 3) {
         prompt("Do the following:\n2 <ENTER>");
         if (showSelection(sub1) == 2) {
            prompt("Do the following:\n3 <ENTER>\n5 <ENTER>");
            showSelection(sub2);
         }
      }
   }
}
void TT(const char* title) {
   cout << "****************************************************************" << endl;
   cout << title << endl;
   cout << "==================>" << endl;
}

/* Execution Sample:
****************************************************************
M1T1: constructors
==================>
****************************************************************
M1T2: operator<< adding MenuItems
==================>
****************************************************************
M1T3: Menu::run()
==================>
Do the following:
abc <ENTER>
0 <ENTER>
4 <ENTER>
3 <ENTER>
** Temp **
1- One
2- Two
3- three
> abc
Invalid Integer, try again: 0
Invalid selection, try again: 4
Invalid selection, try again: 3
3 selected!
Press enter to continue...
```

```
****************************************************************
M1T4: Menu deep assignment with no items
==================>
****************************************************************
M1T5: Reseting Menu title and running an empty menu with no Items
==================>
** New Title **
No Items to display!
0 returned by the Menu with no Items.
Press enter to continue...

****************************************************************
M1T6: Menu::add() and operator<<
==================>
****************************************************************
M1T7: Deep assignment
==================>
Do the following:
1 <ENTER>
** Sub Menu **
1- The first
2- The second
3- The third
> 1
1 selected!
Press enter to continue...

****************************************************************
M1T7: Deep copying, passing const refrence and indentation display
==================>
Do the following:
2 <ENTER>
3 <ENTER>
** Main Menu **
1- Option one
2- Option Two
3- Sub Options
4- Option four
5- Exit
> 2
You selected 2.
** Main Menu **
1- Option one
2- Option Two
3- Sub Options
4- Option four
5- Exit
> 3
You selected 3.
Do the following:
2 <ENTER>
    ** Sub Menu One **
    1- Selection one
    2- Sub Selections two
    3- Selection three
    4- Selection four
    > 2
```

```
You selected 2.
Do the following:
3 <ENTER>
5 <ENTER>
        ** Sub Menu **
        1- The first
        2- The second
        3- The third
        > 3
You selected 3.
** Main Menu **
1- Option one
2- Option Two
3- Sub Options
4- Option four
5- Exit
> 5
*****************************************************************
M1T8: operator bool()
==================>
The mainMenu is valid and usable.
*****************************************************************
M1T9: Invalid Menu and its usage
==================>
The mainMenu is invalid(empty) and not usable.
Invalid Menu!
Invalid Menu!
*****************************************************************
M1T10: Assigning Menu to an empty Menu
==================>
Invalid Menu!
Invalid Menu!
*****************************************************************
M1T11: adding invalid string to render Menu invalid
==================>
Invalid Menu!
Invalid Menu!



*/
```