Project: Simple Point Of Sale System

Latest Release

MS1

Milestones

Milestone	Revision	Approximate Workload (days)	Overview	Comments
MS1	V0.9	5		
MS2		9		
MS3		10		
MS4		4		
MS5		14		

Your task for the project for this semester is to create simple Point of Sale (POS) application that keeps track of a small inventory of Goods to sell and can sell them at the cashier, issuing a bill of sale.

Milestones due dates

This project will be done in 5 milestones and each milestone will have its due date. The due date of each milestone is stated below, and it is based on the amount of work to be done for that milestone.

Final project mark and due dates

Milestone	Mark	Due date	Submission Policy
MS1	10%	Mar 10	gets full mark even if 1 week late. gets 0% afterwards
MS2	10%	Mar 20	gets full mark even if 1 week late. gets 0% afterwards

Milestone	Mark	Due date	Submission Policy
MS3	10%	Mar 29	gets full mark even if 1 week late. gets 0% afterwards
MS4	10%	Apr 2	gets full mark even if 1 week late. gets 0% afterwards
MS5	60%	Apr 16	See below

> To make the final submission of the project easier and to make it possible to partially submit a project we have divided the submission of milestone 5 into five small ones. Each submission is worth 12% of the project mark. Your project will be marked only if you have all four milestones and at least have one of the five submissions of milestone 5.

Milestone 5 Divided into five submission	Mark	Due date	Submission Policy
m51	12%	Apr 16th	10% penalty for each day being late up to 5 days
m52	12%	Apr 16th	10% penalty for each day being late up to 5 days
m53	12%	Apr 16th	10% penalty for each day being late up to 5 days
m54	12%	Apr 16th	10% penalty for each day being late up to 5 days
m55	12%	Apr 16th	10% penalty for each day being late up to 5 days

> The first 4 milestones will not be marked based on the code, but their success and their timely submissions. You may modify or debug your previous code as you are going through the milestones. The only milestone that is going to scrutinized based your code will be milestone 5. If you require any feedback on your first four milestones you need to ask your professor to do so.

> ⚠️ The first four milestones must be submitted successfully even if they are very late.
> Your project will receive a mark of zero if any of the first 4 milestones are not submitted by the rejection date (Apr 21) . For your project to be marked, you must submit all the 4 milestones and at least one of the 5 submissions of Milestone 5 (Rejection date for milestone 5 is also Apr 21)

You can check the due date of each milestone using the `-due` flag in the submission command:

```
~profname.proflastname/submit 2??/prj/m? -due
```

- replace **2??** with the subject code
- replace **m?** with the milestone number

> Different sections may have different due dates based on their section's progress. Always check the due date using this command. The due date returned by this command always have priority over the due dates stated above.

Citation, Sources

When submitting your work, all the files submitted should carry full student information along with the "citation and sources" information. See the following example:

If you have multiple submissions of the same milestone, please update the Revision History in each submission so your professor knows what changes to look for.

```
/* Citation and Sources...
Final Project Milestone ?
Module: Whatever
Filename: Whatever.cpp
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 project milestones.
-----------------------------------------------------------
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.
-----------------------------------------------------------*/
```

Failing to include the above citation to any of the files containing your work will cause the rejection of your project submission

See below for details:

For work that is done entirely by you (ONLY YOU)

If the file contains only your work or the work provided to you by your professor, add the following message as a comment at the top of the file:

> I have done all the coding by myself and only copied the code that my professor provided to complete my project milestones.

For work that is done partially by you.

If the file contains work that is not yours (you found it online or somebody provided it to you), **write exactly which part of the assignment is given to you as help, who gave it to you, or which source you received it from.** By doing this you will only lose the mark for the parts you got help for, and the person helping you will be clear of any wrongdoing.

> ⚠️ This Submission Policy only applies to the project. All other assessments in this subject have their own submission policies.

If you have helped someone with your code

If you have helped someone with your code. Let them know of these regulations and in an email write exactly which part of your code was copied and who was the recipient of this code.
By doing this you will be clear of any wrongdoing if the recipient of the code does not honour these regulations.

Compiling and Testing Your Program

All your code should be compiled using this command on `matrix`:

```
g++ -Wall -std=c++11 -g -o ms file1.cpp file2.cpp ...
```

- `-Wall` : the compiler will report all warnings
- `-std=c++11` : the code will be compiled using the C++11 standard

- `-g` : the executable file will contain debugging symbols, allowing *valgrind* to create better reports
- `-o ms` : the compiled application will be named `ms`

After compiling and testing your code, run your program as follows to check for possible memory leaks (assuming your executable name is `ms`):

```
valgrind --show-error-list=yes --leak-check=full --show-leak-kinds=all --track-origin
```

- `--show-error-list=yes` : show the list of detected errors
- `--leak-check=full` : check for all types of memory problems
- `--show-leak-kinds=all` : show all types of memory leaks identified (enabled by the previous flag)
- `--track-origins=yes` : tracks the origin of uninitialized values (`g++` must use `-g` flag for compilation, so the information displayed here is meaningful).

To check the output, use a program that can compare text files. Search online for such a program for your platform, or use *diff* available on `matrix`.

Adding alias to `.bash_profile` on matrix

You can add the following two lines to the end of your `.bash_profile` file in your matrix home directory to create two aliases for the above C++ compilation and valgrind testing so you don't have to type such a long command to compile and execute your programs:

```
alias msc+="g++ -Wall -std=c++11 -g -o ms"
alias vrun="valgrind --show-error-list=yes --leak-check=full --show-leak-kinds=all --
```

By adding the above to the '.bash_profile' (you must logoff and log back it so it takes effect) you can compile your c++ files as follows:

```
msc++ file1.cpp file2.cpp file3.cpp...<ENTER>
```

The above will compile your files and if successful it will create an executable called `ms`.

```
vrun ms<ENTER>
```

The above will execute your `ms` executable under the tester `valgrind` for detailed test on leaks and runtime errors.

> Make sure that all the debugging code and debugging comments are removed before submission.

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 attributes(member variables), functions and methods (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.
- If any methods are being added and they are not called outside the scope of the class, make sure they are private.
- 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 a penalty.
- When creating methods (member functions) make sure to make them constant if in their logic, they are not modifying their class.
- If an Empty state is required for an object, it is considered to be an “invalid” empty state, and objects in this state should be rendered unusable.
- A module called **Utils** is added to the project that can be used for your custom functions and classes in your implementation. Leave this module empty if you don't have any custom functionalities.
  You can add any custom code/class of your own to the **Utils** module to be used throughout the project.
- Throughout the project, if any class is capable of printing, 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 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.
- You may reuse and copy any code your professor provided for your workshops or functions you may have from previous work in this subject or other subjects and place it in the Utils module.

The Point Of Sale system

We will start the development of the project by creating its user interface (MS1, PosApp module). Through this, we will create a mockup application that represents what the application will look like and act without actually doing anything. In short, we are creating a prototype of the application (in MS1) then coding its engine (in MS2, MS3 and MS4) and finally adding functionality to the POS system (MS1) to make it fully functional (in ms5, parts 1 to 5).

Milestone 1

The PosApp Module

This Module contains one Module called `PosApp`. This module is responsible to offer the user a menu of features provided by the POS system.

The user, then, can select an option representing the feature and execute it. After the execution is complete, the system displays the menu again, until the user selects to exit the application.

The features of the POS system

1. List Items
2. Add Item
3. Remove Item
4. Stock Item
5. Point Of Sale

Implementation

For milestone 1 you are responsible to create a mockup module for the Point Of Sale that will demonstrate how the system is going to run (eventually) by only printing the name of the actions, instead of executing them. In later

stages of development, you will replace these messages with the proper logic to actually perform the action.

Note that the signature of many of the methods created now will be changed to accommodate what needs to be done.

The **PosApp** class

In module `PosApp` create a class with the same name (`PosApp`) having the following mandatory methods:

`menu`

This method will display the menu of the system, and receive the user's choice (in a foolproof way and return the choice). See below:

```
The Sene-Store
1- List items
2- Add item
3- Remove item
4- Stock item
5- Point of Sale
0- exit program
> a
Invalid Integer, try again: -1
[0<=value<=5], retry: > 6
[0<=value<=5], retry: > 1
Running listItems()
```

Mockup methods:

Create the following 7 methods that only print `Running` and their names.

1. addItem()
2. removeItem()
3. stockItem()
4. listItems()
5. POS()
6. saveRecs()

7. loadRecs()

Construction

`PosApp` is created by receiving the name of a file that is stored in character cString with a maximum length of 255 characters.

`PosApp` Can neither get copied nor assigned to another `PosApp` object. Your code must prevent such actions.

`run` method

> implement the following actions calling the corresponding mockup methods

This method first loads all the records and then displays the menu waiting for the user to make the selection. After the (foolproof) selection the proper action is executed and again the menu is displayed until the option exit is selected. In the latter case, all the records are saved and a `Goodbye!` message is displayed.

See the correct_output.txt file for sample execution.

The tester file for this output is main.cpp

MS1 Submission

files to submit

```
PosApp.cpp
PosApp.h
Utils.cpp
Utils.h
main.cpp
```

Data entry

```
abc
-1
6
1
2
```

```
3
4
5
0
```

Submission Process

> If you would like to successfully complete the project and be on time, **start early** and try to meet all the due dates of the milestones.

Upload your source code and the tester program to your `matrix` account. Compile and run your code using the `g++` compiler as shown in the introduction and make sure that everything works properly.

Then, run the following command from your account (replace `profname.proflastname` with your professor's Seneca userid):

```
~profname.proflastname/submit 2??/prj/m1
```

and follow the instructions.

- *2??* is replaced with your subject code

The submit program's options:

```
~prof_name.prof_lastname/submit DeliverableName [-submission options]<ENTER>
[-submission option] acceptable values:
  "-due":
       Shows due dates only
       This option cannot be used in combination with any other option.
  "-skip_spaces":
       Do the submission regardless of incorrect horizontal spacing.
       This option may attract penalty.
  "-skip_blank_lines":
       Do the submission regardless of incorrect vertical spacing.
       This option may attract penalty.
```

```
"-feedback":
     Check the program execution without submission.
```

Back to milestones

Milestone 2

> 🚧 under construction

MS2 Submission

> If you would like to successfully complete the project and be on time, **start early** and try to meet all the due dates of the milestones.

Upload your source code and the tester program to your `matrix` account. Compile and run your code using the `g++` compiler as shown in the introduction and make sure that everything works properly.

Then, run the following command from your account (replace `profname.proflastname` with your professor's Seneca userid):

```
~profname.proflastname/submit 2??/prj/m2
```

and follow the instructions.

- *2??* is replaced with your subject code

The submit program's options:

```
~prof_name.prof_lastname/submit DeliverableName [-submission options]<ENTER>
[-submission option] acceptable values:
  "-due":
       Shows due dates only
       This option cannot be used in combination with any other option.
  "-skip_spaces":
       Do the submission regardless of incorrect horizontal spacing.
```

```
        This option may attract penalty.
    "-skip_blank_lines":
        Do the submission regardless of incorrect vertical spacing.
        This option may attract penalty.
    "-feedback":
        Check the program execution without submission.
```

Back to milestones

Milestone 3

> 🚧 under construction

MS3 Submission

> If you would like to successfully complete the project and be on time, **start early** and try to meet all the due dates of the milestones.

Upload your source code and the tester program to your `matrix` account. Compile and run your code using the `g++` compiler as shown in the introduction and make sure that everything works properly.

Then, run the following command from your account (replace `profname.proflastname` with your professor's Seneca userid):

```
~profname.proflastname/submit 2??/prj/m3
```

and follow the instructions.

- *2??* is replaced with your subject code

The submit program's options:

```
~prof_name.prof_lastname/submit DeliverableName [-submission options]<ENTER>
[-submission option] acceptable values:
  "-due":
```

```
        Shows due dates only
        This option cannot be used in combination with any other option.
    "-skip_spaces":
        Do the submission regardless of incorrect horizontal spacing.
        This option may attract penalty.
    "-skip_blank_lines":
        Do the submission regardless of incorrect vertical spacing.
        This option may attract penalty.
    "-feedback":
        Check the program execution without submission.
```

Back to milestones

Milestone 4

> 🚧 under construction

MS4 Submission

> If you would like to successfully complete the project and be on time, **start early** and try to meet all the due dates of the milestones.

Upload your source code and the tester program to your `matrix` account. Compile and run your code using the `g++` compiler as shown in the introduction and make sure that everything works properly.

Then, run the following command from your account (replace `profname.proflastname` with your professor's Seneca userid):

```
~profname.proflastname/submit 2??/prj/m4
```

and follow the instructions.

- *2??* is replaced with your subject code

The submit program's options:

```
~prof_name.prof_lastname/submit DeliverableName [-submission options]<ENTER>
[-submission option] acceptable values:
  "-due":
       Shows due dates only
       This option cannot be used in combination with any other option.
  "-skip_spaces":
       Do the submission regardless of incorrect horizontal spacing.
       This option may attract penalty.
  "-skip_blank_lines":
       Do the submission regardless of incorrect vertical spacing.
       This option may attract penalty.
  "-feedback":
       Check the program execution without submission.
```

Back to milestones

Milestone 5

> 🚧 under construction

MS51 submission test

> 🚧 under construction

Data entry

Expected outcome

reflection

Create a file called `reflect.txt` and add the following:

- Your Citation if you have any borrowed code in your project.
- Any additional (extra) work done that needs your professor's attention.

- Your overall reflection on the project and work done in the 5 milestones.

MS51 Submission command

```
~profname.proflastname/submit 2??/prj/m51
```

MS52 submission test

> 🚧 under construction

Data entry

Expected outcome

MS52 Submission command

```
~profname.proflastname/submit 2??/prj/m52
```

Back to milestones

MS53 submission test

> 🚧 under construction

Data entry

Expected outcome

MS53 Submission command

```
~profname.proflastname/submit 2??/prj/m53
```

Back to milestones

MS54 submission test

> 🚧 under construction

Data entry

Expected outcome

m54-correct-output.txt

MS54 Submission command

```
~profname.proflastname/submit 2??/prj/m54
```

Back to milestones

MS55 submission test

Data entry

Expected outcome

MS55 Submission command

```
~profname.proflastname/submit 2??/prj/m55
```

Back to milestones