CPT323: Object Oriented Programming in C++
Study Period 3, 2011
Assignment 2: Development of a UI for the Online Purchasing Project

Due date –Monday November 21st, 11:59pm

Abstract

This assignment involves using the `Item/Book/Music` and any other classes implemented and tested in Assignment 1 and to write new classes to provide a `WimmeraStore` user interface. STL containers should be used to store objects. Also significant exception handling and significant input validation should be implemented .

The work involved is based on the C++ concepts and techniques learnt in the second six weeks of the course material.

The assignment is worth 20% (100 marks).

The Scenario

The representative from "YourSoftware", a software branch of RMIT, met again with the client to discuss and show the up-to-date class development and a demonstration of the current version. The client was very enthusiastic and discussed future development of the software package. One important requirement that was again stressed was that the software must be Object Oriented and developed as "pure" C++; that is, using only C++ libraries, and not a hybrid of C and C++.

The software designer knew that the STL library would be perfect for this package and commenced analysing which STL containers should be used. To be a robust package there should also be significant exception and validation implementation.

General Implementation Details

All initial data should be read from one data file**. For file testing purposes only you can assume only books and users**. The required formats are given in "`wimmera.dat`" and all output data should be printed to a new "*`wimmera.dat`*" in the same format with the updated values plus any Cart information for the Users when the "Save/Quit" menu option is chosen.

If the input is formatted incorrectly the execution should stop immediately and an appropriate error message should be displayed.

Marks will be allocated to your class design. You are required to modularise classes properly---i.e., to use abstract classes, inheritance, operator and function overloading as appropriate. No function should be longer than 50 lines.

Marks will be allocated to proper documentation and coding layout and style. Your coding style should be consistent with standard coding conventions

Your programs will be marked on yallara using "`g++ -ansi -pedantic -Wall`". Make sure you test your program on this machine before you submit.

Further hints, tips and details will be made via Blackboard.

Overall Specification

You will build out the Daintree system to manage multiple users purchasing different types of items, including discounts for multiple items.

Items to be Purchased

As in Assignment 1, a Book or CD can also be sold as a physical copy or as and ebook/emusic. You need to keep track of the physical copies of Books and CDs, and whether or not a title is available an online item. Books have a title and an author; music items have a title and an artist.

Each item is individually priced---i.e., the price depends on the title and whether it is a physical copy or ebook/emusic.

Purchasing Items

A User can buy any number of items (books, music, or a mix), adding one item at a time to their Shopping Cart. However, a User can only purchase up to a total of $100, unless they are a Member—if a non-Member User tries to add an item to their Shopping Cart that takes the total over their maximum then this is blocked. A Member has no limit.

Items can be added and removed from a Shopping Cart until Checkout. **When an Item is added to the Shopping Cart**, the system checks that there are enough copies of it available; if an Item is added or removed from the Shopping Cart, the number of copies available must be updated. Checkout clears the Shopping Cart.

Users

Users can add Items to their Cart, up to their allowed limit (i.e., their Shopping Cart cannot store a total greater than the limit). A User has an id (must be unique) and password (you do NOT need to make these encrypted or secure), as well as a name and email address. A Member is a special kind of user: a Member has no limit on value they can store in their Cart. Once a User has spent a total of 10% **more** than their limit **in total (this obviously must be over multiple Checkouts),** then they are offered to become a Member—this offer is made straight after they Checkout with the items that takes them to 10% over their limit.

An Administrator is a User that can perform special functions:

- add to the number of copies of a (physical) Book or CD;
- change the price of an item;
- print sales statistics: i.e., number of sales (physical and electronic) of each Item;
- add a new user—system must checked that the id is unique.

Other Users do not have these options on their menu.

A user must keep track of their previous purchases, grouped by Transaction—a Transaction is the set of items purchased at Checkout time.

Users can log in and out—they do not need to Checkout before logging out. However, only one user can be logged in at a time—the system allows something like “change user”. If a User logs back in, their Shopping Cart holds the same contents as before they logged out.

Overall Specification continued

Recommended Items and Discounts

Each item can store a list of *"if you liked this"* recommendations. If a User adds an Item to their Shopping Cart, then the system suggests other titles they may like. Only similar types of things are recommended—i.e., when a Book is added, other Books (not Music) are suggested.

At the time when a list of Recommended titles is given, the user has the option to add one of the recommended titles to their Shopping Cart. If a user adds the title, then they receive a discount of 15% off that second title (the first one is still full price); the User can add multiple recommended titles for 15% off each of them. If a Member adds the recommended title, then they get 25% discount off all the recommendations added**. Note**: when a recommended title is added, its recommendations are also shown, and are discounted if purchased at that time.

You are NOT required to handle the special case of updating discounts when a User removes recommendations from their Cart. However, there are **3 Bonus Marks** for this.

New Requirements of Your Online Purchasing System

Program Development

When implementing large programs, especially using object-oriented style, it is highly recommended that you build your program incrementally.

This assignment proposes a specific incremental implementation process: this is designed to both help you think about building large programs, and to help ensure good progress! You are not strictly required to follow the structure below, but it will help you manage complexity.

Requirement 1 (10 marks): Class Design Diagram

Define all the classes needed for the described system. In particular, you should try to encapsulate all the appropriate data and operations that a class needs. This may mean some classes refer to each other (e.g., the way a User ***"has-a"*** `ShoppingCart`).
At this point, you may just want to think about the data and operations and just write the definitions, not all the code.

- **You need to submit a class diagram (in pdf, gif or jpeg format with your final submission) .**

Requirement 2 Command-Line Arguments (5 marks)

You are to use command-line arguments to provide the name of your input and output files. For example, if your executable file is named assign2 and you read input from `wimmera.dat`, and your output is also `wimmera.dat`, then you would run your program as follows from the command-line on yallara:
>./assign2 `wimmera.dat wimmera.dat`

If you use Eclipse, then you specify the arguments under Run > Run Configurations; however, as always, your program will be marked by running it on yallara.

Requirement 3 File Input and Output (20 marks)

NOTE: For file handling <u>only</u> you can use Book and User data. Put initial Book and User information in a text file (the name of this file is the first command-line argument); your program should read this file at start-up and initialise inventory and create Users with the appropriate values. A sample input file will be provided on Blackboard: you must follow that format precisely, though you are welcome to include more information than in that sample.

Your program will be tested with different values (but with the same format) so make sure you can handle reading the file properly.
NOTE: You do not have to read in recommendations. Handling Cart data in the input file is tricky---leave this until the end of the assignment, it is worth 2 marks.

When the Save/Quit option is selected on the MENU list, you should write an output (text) file. The output file must use the identical format to input file format below. You should write out the same information as you read in with the updated values, plus any Cart information for the Users. Your saved output file must be able to be read back in as an input file.

Requirement 4 (10 marks): Main Program Menu

Your main program should be in the WimmeraStore class. (Of course, any class can contain a main(); this is useful for testing that class.) The main program contains menus that offer the following options in a loop.
Also, the main program will contain a menu that offers all the required options (these can be different for different Users!).
The system will allow a User to login by typing their id and password and will check that these match: if it does not then the menu prints an error; if they do match, then the system prints a welcome message with the user's name and shows them the appropriate menu.

The system must keep a list of all its Users: **this list must be efficient to look-up by User id**.

Sample menus

The menu for a standard User (i.e., a Shopper) should include the following options:

```
1. Add item to shopping cart
2. View shopping cart
3. Remove item from shopping cart
4. Checkout
5. List all items
6. Print previous purchases
7. Logout (change user)
0. Save/Quit
```

The menu for an Administrator should include the following options:

```
1. List all items (this option can include purchase statistics for
each title)
2. Add copies to item
3. Change price of item
4. Add new user
5. Logout (change user)
0. Save/Quit
```

* **SAMPLE RUNS and TEST DATA will be posted to Blackboard ***

Requirement 5 (25 marks): Implement Core Functionality

Implement the core functionality of the Wimmera Store system described above**, except for the recommendations, members, and discounts**. You should be able to implement the rest of the Daintree functionality described above, and run and test your system.

Requirement 6 (20 marks): Implement Recommendations, Members and Discounts

Implement the functionality of providing recommendations, users becoming and being members, and discounts.

Requirement 7 (5 marks) Exception classes and Data Validation

As always, marks will be awarded for coding style, documentation/comments, code layout and clarity, meaningful error and other messages, proper error handling, choice of data structures and other design decisions. For full marks you can use C++'s exception classes where appropriate or write your exception class. Significant data validation will lead to full marks. Define Exceptions to handle problems/errors. These may include: invalid menu options or inputs, etc.

Requirement 8 (5 marks) Makefile, Code Layout/Choice of Data Structures

As always, marks will be awarded for coding style, documentation/comments, code layout and clarity, meaningful error and other messages, proper error handling, choice of data structures and other design decisions.
1 mark for a makefile that runs your assignment on yallara.

Assignment 2 Question to be answered in the README FILE
Why did you choose the STL containers that you implemented?

Bonus (5 marks)

- 2 bonus marks for the use of template functions and/or template classes
- 3 bonus marks for correctly handling removal of recommended books from Cart—e.g., if a Member removes the first item then the 15/25% should be added back to the price of the recommended title, unless there are multiple recommendations linked to that title.
- All bonus mark code to be described in your README file

Further Details

- **You should use STL containers in this assignment.**
- **You should submit a working makefile that runs on yallara.**
- Every class must have its own interface file (e.g. a header file like user.h)
- You must implement all the functions in the interface, and you are allowed to add extra modifiers if needed e.g virtual.
- You are also allowed to add extra classes and functions to given classes such as extra constructors, and other functions that you believe are useful
- You are encouraged to overload operators e.g. the << and >> operators for particular objects/streams eg writing the attributes of an object to a file
- The code must use C++ libraries only – no specific C libraries must be used.
- You are allowed to use static_cast and dynamic-cast if required.
- You can divide the assignment into your own different namespaces if you wish
- Every file must compile using g++ without errors and warnings with the following compiler flags set: `-ansi -pedantic -Wall`
- All classes should be in their own separate file.
- Each class name must start with an uppercase letter, e.g. Item, and the code must e stored in a file starting with a lowercase letter, e.g. item.cpp
- Variables and constants must have meaningful names
- The code must not be tab indented - a simple 3-space indentation is sufficient.
- No line must be longer than 80 characters

Assignment Regulations

- You may refer to textbooks, notes, work in study groups etc. to discover approaches to problems, however the assignment should be your own individual work.
- Where you do make use of other references, please cite them in your work. Students are reminded to refer and adhere to plagiarism policies and guidelines:
  RMIT CS&IT Academic Integrity: http://www.cs.rmit.edu.au/students/integrity/

Submission

This assessment task is due at 11.59pm on Monday 21st November (the start of week 13).

Late submissions must first be approved by the Online Programs Administrator by completing the "Assignment Extension Form" at this URL:

http://oua.cs.rmit.edu.au/procedures/forms.html

Assignments may be accepted up to 4 days after the deadline has passed; a late penalty of 10% will apply for each day late. Submissions over 4 days late will not be marked.

Emailed submissions will not be accepted without exceptional circumstances.

Submission Format

Submission will take place through Weblearn. As your submission will comprise of multiple-files, please submit a compressed archive of your work in one of the following formats:

- ZIP compressed file (.zip; created by Windows XP, WinZIP Legacy, MacOS X, etc.)
- tar/gzip compressed file (.tar.gz; created by the Unix tar and gzip tools)

Important Note: other compression formats, such as (but not limited to) RAR/WinRAR (.rar) and proprietary WinZip (.zipx) will result in mark deductions.

Submit a README text file for each task. The README should inform the marker of the development environment used for completing the assignment, and how to run your project. Remember the marker will be using g++ with the flags `-ansi -pedantic -Wall`

Also, the README should state such things as how many containers you implemented, ,your function objects if any, your exception handling , data validation, bonus attempt etc..

Also, the README should state any problems you encountered that may help the marker.

For Linux users, please also submit your makefiles.