# **OOP - Semester** 2 **- Group Assignment**

## **Brief outline**

This assignment is designed to demonstrate your understanding of object-oriented programming techniques presented in this module, with focus to C++ facilities introduced in semester 2, and your ability to implement documented UML designs. It is a group assignment that counts for 25% of the module. Groups of 3 or 4. Group changes must be approved by all group members and the teaching team. Individual solutions will be marked but cannot be awarded more than 40% unless agreed upon beforehand.

As well as this document, on blackboard you are also provided with an [initial implementation of the system](https://drive.google.com/open?id=1rA-Ndo36gD5wpwf0a-WYSY0WVAaM-DfD) (which you will use as a starting point for your work), and a couple of other partial code solutions.

The system being developed plans to allow users to purchase digital video games via a store and help them maintain a library of owned games.

The assignment is designed to simulate working with a real client who already has a semi working system but is unsure of the final requirements.

## Learning outcomes

This task assesses the following LOs: a) Identify, implement and use appropriate underlying data structures to store and manipulate data in an OO program. b) Identify, explain and apply the object-oriented features to implement efficient and reusable OO solutions.

## The current system

The system is made up of accounts. Much like netflix’s, spotify’s family model, etc., each account can have multiple users attached to it. There are different user types available - currently **Admin** and **Player**. Each account has exactly one admin user. In the current system admins behave exactly like standard players. Both have their own library of purchased games.The system includes a store - a collection of games which can be browsed by users. Each game has a name, description, rating, and a cost.

## Deadline

### Tuesday 2nd April

Submit two files: **Course-GroupNumber-Report.docx**, and **Course-GroupNumber-Code.zip** on Bb. Ensure all work includes your course, group number, and member names. Marks may be lost for not following these instructions. Make sure your program works on the university computers. All members must demonstrate your work in a **walkthrough session**. As well as **peer assessment**, your **use of git** will also affect your individual mark. Walkthrough times will be announced nearer the time.

## Task 1 - Understand the system - 10% - 2 marks each

To check everything is understood, please use this documentation and the code to answer the questions.

1. Why are **pointers** used to store the currently logged in user and account instead of instances of the **User** and **Account** classes?
2. How can we tell **Menu** is an **abstract** class? Why is it an abstract class? Which other class(es) in the proposed system in **appendix 2** could / should also be an abstract class?
3. **Accounts** are not currently set up to work with different types of users. What needs changing and why?
4. How is the **Menu** system set up to work polymorphically? Why are **virtual** functions being used? What would happen if they were not there?
5. Part of the improvements required is to represent dates correctly. Why would creating a **static** function **Date::CurrentDate()** be a good idea? How do **static** functions work?

## 

## Task 2 - Base improvements needed - 25%

The initial prototype was made as a way to explore and understand our idea. Now that we have a better idea of the system the following improvements are necessary before moving on to implementing more features.

### A more mature menu system

There will be lots of different views in the system so a more robust system for handling this is required. A diagram (**appendix 1**) and [prototype of the menu system](https://drive.google.com/open?id=1OXO_Bbn0uYZfjCQLwNxbEYmP50ulPmnk) are available. Merge the 2 systems to create a starting point for your work.

### Represent dates correctly

Dates are currently stored as **strings** in the format YYYY-MM-DD. Replace these with a custom **Date** class which provides all the required functionality and validity checks.

### Better handling of collections

Collections are currently **arrays** with hardcoded bounded sizes which restricts planned features. Replace all arrays with instances of the [custom generic class template **List<T>**](https://drive.google.com/open?id=1KKpMpgY0hVS_XBBpbCs3UB8eEEs_qKUh). To iterate over the **List** you will have to create a copy and repeatedly delete the first item, or overload the [] operator to allow a standard for loop.

### Utility functions

During development of the menu system it was deemed useful to maintain a collection of handy functions in a module called **Utils**. Continue to add useful functions to this as you develop the system.

## Task 3 - Core feature implementation - 35%

These are the core features which we’re sure we need in the finished system.

### User profiles

**Users** should be able to access their **profile page** after logging in. It should list the games they own. If the user is an **admin** it should also allow them to **create and delete player users** under the account they manage.

### Logging in to different accounts

Currently there is a single hardcoded **account** which is automatically logged in. Remove the automatic login and add an **account login option** to the main menu. Once logged in you can log in as a particular user.

### Persistent data

Test data is currently hardcoded into the source code. Replace this with the **text based file format** provided (**data.txt**). Create two new functions: **Application::Load()** runs at startup, reads the file, instantiates objects, and fills the relevant collections with these objects; **Application::Save()** runs just before closing, fully overwriting the file with the updated state of the system.

### Buying from the store

**Player** users need to be able to purchase games from the store. They will need **credits** adding to their data. They can only buy if they are logged in and have enough credits. On their profile page they should be able to see how many credits they have, and they should have an option to buy additional credits (simulate for now).

### Searching the store

Users should be able to search the store for games by name, price, and rating. Add a templated **Search** function to the **Store** class. For name searching create a **StartsWith** function in **Utils** which takes two **strings** as input (full text, search text) and returns a **bool**. Searching “br” would match both “Brothers”, and “Braid”, as would “BR”.

## Task 4 - Experimental features and improvements - 15%

The following are features and improvements we know we’d like in the finished system, but we’re currently unsure of the best way to approach them. We’d like you to implement these so we can test them.

### Aesthetics and consistency

Menu titles are currently in ALL CAPS **except** for on a game’s page. Make the **game menu** show the title in all caps. The data stored on the game object shouldn’t be changed. We’d like to analyse 2 different approaches to doing this so please create 2 versions of **ToUpper** in **Utils**. The functions should receive a **string** as input and return the uppercase equivalent of that **string**. Version A should use a standard **for** or **while** loop, whereas version B should be **recursive**.

### Playing games

**Users** should be able to select a game to play from their **profile page**. We cannot launch games yet so we’d like to simulate this by adding a random duration of 10 to 60 minutes of **play time** to a players’ records. Time played should be listed next to the game on their profile page and **formatted** in a user friendly way. For example, display exact minutes up to 1 hour of play time (e.g. 57 mins), after that show hours with 1 decimal point (e.g. 3.5 hrs), but after 5 hours of total play time just display hours with no decimal precision (e.g. 7 hrs). Create functions in **Utils** for getting a random number within a range, and for formatting a duration using the rules described.

### Library sorting and the standard List class

We’d like to evaluate the standard C++ **list** class and some of the surrounding features, so please replace Player’s **LibraryItem** collection with **std::list** and make the necessary changes in other areas of the codebase to accommodate this change. To test this properly we’d like to see it used with the built in sorting algorithm. Use **std::sort** from **<algorithm>** to allow users to sort the games displayed on their profile page. They should be able to choose to sort by game name or by date purchased.

## 

## Task 5 - Exploratory functionality - choose 1 - 15%

We have ideas of additional functionality which may add value to the system, but we are unsure. Please choose **one** to implement so we can test it and make a more informed choice.

### Friends

Player users should be able to maintain a list of friends. These must be other **Player** users. To do this you will need to make sure all **Player** usernames are **unique**. From their profile page users should be able to **sell a game** from their purchased games to one of their friends at ½ price. The friend must have enough credits to buy the game. If successful, after the transfer, the credits for each player are updated and the game has been transferred from an account to the other. The seller user will no longer own the game.

### Library management

From their **profile** player users should be able to **create and manage groups** to help **organise their library** of purchased games. For example they might create a “Currently Playing” list and add a few games so that they appear together. Their profile should now display their games in this format.

### Guest user

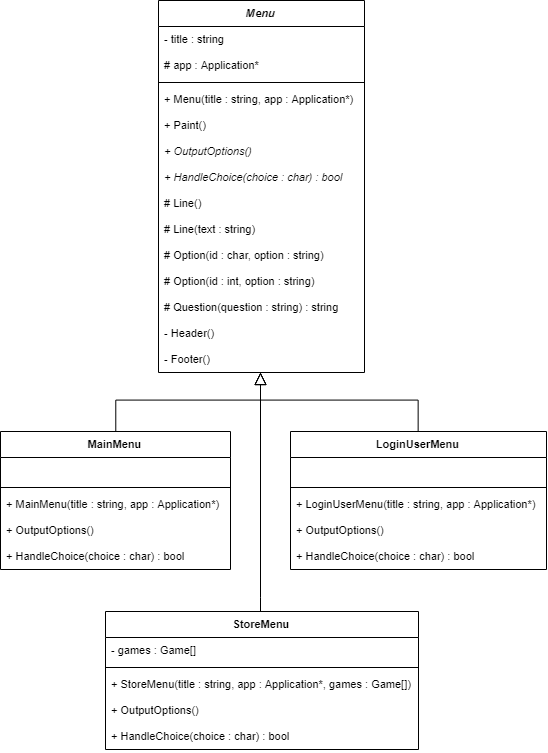
If you have a friend over and they do not have a registered user they should be able to sign-in as the **guest user**. Each account has a **single** guest user (**appendix 3**). The games available to a guest user are a **subset** of the games **owned by the admin user** for that account. From their profile page the admin can choose which of their games are available to guests. Guests can browse the store but cannot purchase games.

### Reviews

Player users should be able to **rate games** that they have purchased by going to their **profile page** and either giving it a **like or a dislike**. On the **store menu** games will now **show a percentage rating** next to their title based on this data. For example 4 likes and 1 dislike would be 80%. 15 likes and 35 dislikes would be 30%. Viewers of the store should be able to **sort** the display by rating.

## Appendices

### Appendix 1 - Partial class diagram for the Menu system



### Appendix 2 - Partial class diagram for the starting system

### 

### 

### Appendix 3 - Partial class diagram for the Guest hierarchy

# 

## Feedback and marking sheet

|  |  |  |
| --- | --- | --- |
| Group | Names | 100 |

### Task 1 - Understand the system - 10%

|  |  |  |
| --- | --- | --- |
| Questions | | 10 |

### Task 2 - Base improvements - 25%

|  |  |  |
| --- | --- | --- |
| Merge prototype and menu system | | 15 |
| Integrate custom Date class | | 5 |
| Integrate custom List<T> class | | 5 |

### Task 3 - Core features - 35%

|  |  |  |
| --- | --- | --- |
| User profile page | | 7 |
| Logging in | | 7 |
| Persistent data | | 7 |
| Purchasing from the store | | 7 |
| Searching the store | | 7 |

### Task 4 - Experimental features and improvements - 15%

|  |  |  |
| --- | --- | --- |
| Game titles + ToUpper | | 5 |
| Playing game simulation | | 5 |
| std::list<T> and std::sort | | 5 |

### Task 5 - Exploratory functionality - 15%

|  |  |  |
| --- | --- | --- |
|  | | 15 |