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

To check everything is understood, use this documentation and the skeleton code to answer:

1. Why are **pointers** used to store the currently logged in user and account instead of instances of the **User** and **Account** classes?

A pointer references a location in a block of memory. You can easily manipulate the memory locations of the data for the logged in users, the amount of space allocated directly corresponds to the class. Instances of the User and Account classes will exist until it is deleted manually but, having the pointer to it means it will not be deleted once you exit the scope, so it can be used in other scopes and functions. The User or Account derived from the base classes can be accessed using a pointer no matter what type of derived class it is, it can all be stored in the base class’ pointer and dereferenced.

1. How can we tell **Menu** is an **abstract** class? Why is it useful for it to be an abstract class? Which other class(es) in the proposed system in **appendix 2** could / should also be an abstract class?

Menu is an abstract class as functions inside it are pure virtual functions, and they can be overridden by functions in the derived classes. It is useful for it to be abstract as it is a base class, multiple derived classes can override the pure virtual functions from it, so it functions differently for each derived class. The class that should be abstract is User as player and admin derive from the User class and will use some overridden pure virtual functions from User.

1. How are **Accounts** set up to work with different types of users? Which mechanism allows this?

Accounts are set up to work with Users, of which derived classes are Admin and Player. User is an abstract class, used as a base class with pure virtual functions, that are overridden for ‘Admin’ or ‘Player’ User types. When a User is created it can be created as a ‘Player’ or ‘Admin’, each may have different functionality for the overriding functions. Accounts receives the User object and it can then be added to a pointer of Users, whether it is an Admin or Player object.

1. How is the **Menu** system set up to work polymorphically? Why are **virtual** functions being used? What would happen if they were not virtual?

The Menu system is polymorphic as it contains virtual functions which can be overridden by derived classes. These virtual functions are used so that every time we make a new derivation of Menu we can alter the methods to get different result of OutputOptions() and HandleOptions(). If they were not virtual every new derivation of Menu would just be another Menu class.

1. 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?

The function Date::CurrentDate() set to static is a good idea as there will only ever be one copy of the static function. It is independent of any object of the class, thus restricts access to this function.

A static function has a scope limited to its class or object. Static functions are only visible in its class or object, it cannot be called in another class.