1 Introduction

This practical assignment will require you to provide an implementation for various files provided to you. These files give an outline for the Composite and Decorator design patterns. After successful completion of the practical assignment, you will:

- have an understanding of the Composite design pattern (and by extension, the concept of a tree structure)
- have an understanding of the Decorator design pattern
- understand the similarities and differences between the Decorator and Composite patterns, as well as how they are capable of working together in one hierarchy
- practice your drawing of basic UML diagrams
- be able to do basic operations making use of vectors

2 Constraints

1. You must complete this assignment individually.
2. Teaching assistants will be available to help you during the practical times. They will however not give you the solutions.
3. Students should attend the practical that they booked for in order to get their practical tasks marked.

3 Submission Instructions

You are required to upload all your source files and the UML state diagram image file as a single tar.gz archive to the CS website before the deadline.
## Mark Allocation

<table>
<thead>
<tr>
<th>Task</th>
<th>Marks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Soldier and Main</td>
<td>1</td>
</tr>
<tr>
<td>Fighter</td>
<td>5</td>
</tr>
<tr>
<td>Party</td>
<td>14</td>
</tr>
<tr>
<td>Improved</td>
<td>3</td>
</tr>
<tr>
<td>Mage and Paladin</td>
<td>6</td>
</tr>
<tr>
<td>UML Diagram</td>
<td>13</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td><strong>42</strong></td>
</tr>
</tbody>
</table>
5 Assignment Instructions

In this practical, you will be required to provide the implementation for a variety of header files (please provide implementation in separate .cpp files for easier marking).

The scenario is as follows: There are monsters to be fought. You need to gather an army. You need a complex hierarchy in your army, so that you can be at the helm and manage sub-armies within your army (similarly capable of such structures). Visualizing a tree structure, you remember the Composite design. You also realise you are going to need more than just basic fighters to deal with tough monsters. What could be better for creating properly trained fighters than Decorator?

Take note: The implementation of the enemy (Boss) is unrelated to the design patterns and is thus provided for you.

The task of understanding the structure of the Design Patterns in question has been left to you (and will be tested thoroughly in your UML Diagram). You have been provided with the header files for each class. As such, this practical is split as follows

**Task 1: Soldier and Main** ................................................................. (1 marks)

Inspect soldier.h. Do not change it, simply be able to understand it, so you can give an explanation, should you be asked about any part of it.

Secondly, inspect the function of the main.cpp provided. You will not be questioned about this, however it will be greatly beneficial to your understanding of the scenario provided to you, as well as the manner in which it is intended to be implemented.

**Task 2: Fighter** ................................................................. (5 marks)

For Fighter, you must provide the implementation of the following functions

- **Constructor:** Takes an integer (the level of the fighter) and initializes the hp and atk member variables as follows - HP between 20 and 20+level, ATK between 10 and 10+Level. This should be done using randomization (you may use the rand and srand functions available in the Standard Library).

- The Constructor should also give output in the format "A new level [level] fighter with HP: [hp] and ATK: [atk] has been created"

- The attack function: Should deal damage equal to the Fighter’s atk to the target (you should be able to determine how from inspection of the given files)

- The takeDamage function: Removes an integer amount of health from the Fighter and returns true if the fighter is still alive, otherwise returns false.

**Task 3: Party** ................................................................. (14 marks)

For the Party class, you are required to provide a variety of function implementations, once again. Additionally, you must be able to work efficiently with vectors. The functions to be implemented are as follows:

- **Destructor:** Provide implementation as per prior knowledge of deallocation.

- The recruit function: Adds a generically typed Soldier to the current party. Should output the string "New member has been added to party".

- The oust function: Removes a given soldier from the party (remember to consider the consequences of removing a pointer from some manner of list). Should output the string "A fighter has been removed from the party". If the Soldier passed in is not part of the party, should output the string "An invalid party member removal has been requested. No action taken".

- The getMember function: Returns a pointer to the party member at a given index. If an invalid index is supplied, should output "Invalid member index specified".

- The attack function: Should cause every party member to attack. If there are no party members, output the string "No living party members, party cannot fight". (although the takeDamage function should ensure this output never occurs, it is purely for aesthetics and reusability).
• The takeDamage function: Should cause every party member to take a set amount of damage and, if a Soldier is no longer alive after he or she attacks, their cold, lifeless body should be removed from the party. If there are still party members remaining, the function returns true, otherwise it returns false.

**Task 4: Improved** ................................................................. (3 marks)
For the Improved class, you are required to provide a variety of function implementations, once again. Additionally, you must be understand a bit of a corner case regarding pure virtual functions with definitions (before you call me crazy, read this: [http://blogs.msdn.com/b/oldnewthing/archive/2013/10/11/10455907.aspx](http://blogs.msdn.com/b/oldnewthing/archive/2013/10/11/10455907.aspx)). This implementation may be relatively simplistic, but it is important that you understand the possibilities associated with it and the relevant design pattern (don't forget, you have to identify what design pattern this belongs to at the end).

- Constructor: Initialize member variable
- Destructor: Deallocate dynamically allocated memory
- The attack function: Cause the Soldier in question to attack the Boss passed in as a parameter
- The takeDamage function: Cause the Soldier in question to take the specified quantity of damage.

**Task 5: Mage and Paladin** ................................................................. (6 marks)
For the Mage and Paladin classes, you have to provide some very simplistic implementations as follows:

- Constructor: Calls base class constructor and output "This fighter is a Mage" for the Mage class and "This fighter is a Paladin" for the Paladin class.
- The attack function: This provides some behavioural changes on top of existing attack functions. The Mages attack function must use the base class attack function to attack twice, whereas the Paladin behaves normally.
- The takeDamage function: As with the attack function, this is a behavioural modification. The Mage in this case behaves normally, whereas the Paladin takes only half damage.

**Task 6: UML Diagram** ................................................................. (13 marks)
Create a complete class diagram from all the classes of this practical. Annotate the diagram with the roles the different classes have in the Composite and Decorator design patterns. Export the class diagram to a png image and upload it with your code.

**Software:** You can use Visual Paradigm to create your class diagrams. By this point, you should have access to Visual Paradigm and this should not be a problem.

**Note:** Here is some example output for creation of the party:
Do you want to add another member to this party? (y/n): y
Enter 1 to recruit a new soldier to your party. Enter 2 to add a new sub-party. (1/2): 1
What kind of fighter do you want? (1 - Paladin, 2 - Mage, 3 - Generic fighter): 
A new level 10 fighter with HP: 26 and ATK: 11 has been created
This fighter is a Mage
New member has been added to party
Do you want to add another member to this party? (y/n): y
Enter 1 to recruit a new soldier to your party. Enter 2 to add a new sub-party. (1/2): 1
What kind of fighter do you want? (1 - Paladin, 2 - Mage, 3 - Generic fighter): 
A new level 9 fighter with HP: 27 and ATK: 18 has been created
New member has been added to party
Do you want to add another member to this party? (y/n): y
Enter 1 to recruit a new soldier to your party. Enter 2 to add a new sub-party. (1/2): 2
New member has been added to party
A new sub-party has been created. It is now the active party
Do you want to add another member to this party? (y/n): y
Enter 1 to recruit a new soldier to your party. Enter 2 to add a new sub-party. (1/2): 1
What kind of fighter do you want? (1 - Paladin, 2 - Mage, 3 - Generic fighter): 
A new level 7 fighter with HP: 26 and ATK: 12 has been created
This fighter is a Paladin
Do you want to add another member to this party? (y/n): n
Active party has a parent party. Continue editing parent party? (y/n): y
Do you want to add another member to this party? (y/n): n
Enter 1 to recruit a new soldier to your party. Enter 2 to add a new sub-party. (1/2): 1
What kind of fighter do you want? (1 - Paladin, 2 - Mage, 3 - Generic fighter): 
A new level 3 fighter with HP: 22 and ATK: 11 has been created
This fighter is a Paladin
New member has been added to party