Graded Lab 5B (18:15-19:45)

In this task you will be implementing fights between fantasy characters.

You are provided with all of the files needed to solve the task (no need to create any additional ones) - they just need to be filled with your code.

Each part contains a description of what needs to be implemented as well as example usages of this particular functionality in the main.cpp file. The main.cpp file cannot be modified (apart from uncommenting the code) and the functions you implement should fit the example usages in it. Your code should produce the same output as provided in the output.txt file.

IMPORTANT: In this task you need to avoid code duplication! You can do this by reusing already existing functions (from the same class or from the base class).

Part 0 - introduction

All objects representing any type of character have to provide the following interface:

- **void Attack(Character& character)** - attacks another character dealing 2 points of damage to it (only executed if the attacker has more than 0 health points)
- **void Rest(int hours)** - the character rests and restores <*hours*> health points to itself (health points cannot exceed the maximum number)
- **void TakeDamage(int damage)** - the character takes <*damage*> points of damage (health points cannot fall below 0)
- **void GetHealed(int heal)** - the character has <*heal*> health points restored (health points cannot exceed the maximum number)
- **operator<<** - outputs information about the character (just like in output.txt)

All of those functions may change their behaviour depending on the type of character executing them.

Important: The output operator has to only be defined in the base class.

Part 1 (1.5 pts)

Implement the Character class.

The header file is already fully provided to you. All of the Character functions need to behave in the same way as described in the introduction.

Part 2 (1 pt)

Implement the Warrior class.

Warrior extends the Character class and modifies the behaviour of the following functions:

- **void Rest(int hours)** - a Warrior first increases their maximum health by 1 and then performs a standard rest operation.
- **void TakeDamage(int damage)** - if <*shield_durability*> > 0 Warrior ignores the damage and decreases <*shield_durability*> by 1, otherwise they take damage as normal
- **operator<<** - apart from the standard Character information the operator also mentions that the objects is of class Warrior and prints the value of <s*hield_durability*> (just like in output.txt)

Part 3 (0.5 pts)

Modify the way a Warrior attacks.

When a Warrior attacks another Warrior they attack until they reduce the <*shield_durability*> of the target to 0 and then perform a standard attack.

Hint: use dynamic_cast

Part 4 (1.5 pts)

Implement the Spell, HealingTouchSpell and ArcaneMissilesSpell classes.

Spell is the base class for HealingTouchSpell and ArcaneMissilesSpell. The header file for it is fully provided. You only need to implement the output operator.

Each spell provides the following interface:

- **void Cast(Character& caster, Character& target)** - performs the operation of casting a spell by character <*caster*> on character <*target*>. The effects of casting a spell for each class are the following:
    - HealingTouchSpell - restores 8 points of health to the <*target*> character
    - ArcaneMissilesSpell - deals 1 point of damage to the <*target*> character 3 times
- **operator<<** - outputs the name of the spell (just like in output.txt).

Important: The output operator has to only be defined in the base class.

Part 5 (2.5 pts)

Implement the Warlock class.

Warlock extends the Character class. The Warlock class contains an array of spells with a constant size specified in the constructor (<*spell_slots*>). After creation a Warlock has no spells in the array (everything set to nullptr).

The Warlock class modifies the behaviour of the following functions:

- **void Rest(int hours)** - the Warlock first performs a standard rest operation and then additionally prepares a new spell and stores it at the first free place in the spells

array. If the Warlock rested for an even number of hours they prepare the ArcaneMissilesSpell, otherwise they prepare the HealingTouchSpell.

- **void Attack(Character& character)** - instead of attacking the Warlock casts the last spell that they have prepared (each spell is one time use only so it should get removed from the array). Casting a spell does not require having health points above 0. If a Warlock does not have any prepared spells they attack according to normal rules.
- **operator<<** - apart from the standard Character information the operator also mentions that the objects is of class Warlock and prints the list of prepared spells (just like in output.txt)

Part 6 (1 pt)

Implement a cloning function for the Warlock class. You can find the declaration for it in the header file.

The function should return a pointer to a new Warlock object that is an exact copy of the object the function was called on. Remember that you also need to create copies of the spells which the Warlock has prepared (the copy of the Warlock can't contain references to the same spell objects as the original Warlock).

Hint: You might need to add an additional function to the spell classes.