Assignment 5

Due Monday August 8th, 2011 by 11:59pm

- Submit all the deliverables to the Course Management System: https://courses.cs.sfu.ca/
- Last possible day to submit is Tuesday August 9th **(only 1 day late is possible, for -10%).**
- This assignment is to be done individually. Do not show another student your code, do not copy code found online, and do not post questions about the assignment online. Please direct all questions to the instructor or TA:
  - For CMPT125, email cmpt-125-d2-help@sfu.ca;
  - For CMPT128, email cmpt-128-help@sfu.ca.
- See the marking guide for details on how each part will be marked.

General Requirements

All programs you submit must have good style:

- Use named constants appropriately: do not use magic numbers.
- Indent and format your code correctly.
- Correctly comment your code. Comment all functions, classes, and blocks of code.
- Conform to all C++ naming conventions discussed in class.

1. Object Oriented Design & UML

Perform Object Oriented analysis/design on the description below to identify the classes, attributes, and necessary member functions. This part is unrelated to the rest of the assignment.

> A planet has a name, and may have any number of moons. Each moon has a name, and is in orbit a certain average distance away from the planet. In addition, each planet may or may not have rings (for example, Saturn does, but Earth does not). Each object should be able directly to display (output) its information to the screen.

- Identify at least two classes, and each class should have at least two attributes.
- Give each class all appropriate accessor and mutator (setter and getter) functions, plus any other functions which are required (based on the description).
  - Hint: a class with a vector member variable should have functions to add an element to the vector, get the number of elements in the vector, and get an element by index.
  - For example, if the class `Course` has a vector of `Students`, then `Course` should include the following member functions (shown in C++ syntax):
    - `void addStudent(Student newStudent);`
    - `int getNumberStudents();`
    - `Student getStudentAtIdx(int index);`
- Create a UML diagram showing:
  - the classes in your design,
  - their attributes and methods
  - the relationships between the classes.
- Your diagram must be submitted electronically.
  - Visual Studios 2010 (full edition) has a built in UML diagram tool.
    - Create a new project and select "Modeling Projects".
    - Create a new item of type "UML Class Diagram".

        - Drag the class icon (on the very right side) into the diagram to create a new class.
        - You can interact with the class objects in the diagram to change its values.
    - You may draw the diagram on paper and then scan it in; however, this must be equally readable as a computer generated solution. Unclear submissions will not be marked.
    - Save your diagram as a PDF called `planetUML.pdf`.
- Do not add any additional features beyond what is described above.
- **Do not implement this in C++!** Just create the UML diagram!

2. Split into files

The rest of this assignment builds on your solution from Assignment 4. It is best to use your own solution, however, you may use the posted solution to assignment 4.

- Take your code from Assignment 4 and split it into four files:
    - `creature.h`, and `creature.cpp` holding the `Creature` class.
    - `menu.h`, and `menu.cpp` holding the `Menu` class.
    - Create a new (empty) `main.cpp` file.
        - The test functions which I provided for the previous assignment are not needed for this assignment; hence, we are starting with an empty client code.
- From now on, all classes will be assumed to be in their own `.h` and `.cpp` files.
    - Remember to put include guards in the `.h` files (`#ifndef`....)
    - Remember to include the header files in the necessary `.cpp` files.

3. Update the Menu Class

- The `Menu` class itself will not change, but into the `menu.h` file, move the prototypes for the `promptForText()` and `promptForYesNo()` functions.
- Place the implementation of each of these functions into the `menu.cpp` file. (For example, at the end of the file).
- It is common to have a library of functions with the prototype in the `.h` file, and the implementation in the `.cpp` file.
    - Here we can add these two functions to the related `Menu` class and cut down the number of files we have to work with. This is done to save time.

4. Update the Creature Class

Update your `Creature` class to match the UML diagram on the right.

Remember that one-line setter and getter member functions may be implemented as inline; all other member functions should be non-inline.

Creature
- name : string - hitPoints : int - experience : int - level : int
+ Creature( ) + Creature(newName : string, newLevel : int) + setName(newName : string) : void + getName( ) : string + getHitPoints( ) : int + getHitPointsMax( ) : int + takeDamage (damage : int) : void + isDead( ) : bool + healAllDamage( ) : void + displayInfo( ) : void + displayInfoShort( ) : void + setLevel(newLevel : int) : void + getLevel( ) : int + setExperience(newXP : int) : void + getExperience( ) : int + gainExperience(xpGain : int) : void + loseExperience(xpLoss : int) : void + getDamageDealt( ) : int

Changes:

- Add a member variable (and setter & getter member functions) for both:
  - the creature's current level, and
  - the creature's current experience.
- Change `initialize()` into a constructor. Implement both a default constructor, and the 2-parameter one shown.
- Change `getHitPointsMax()` to return 25 * (Creature's level).
- Add gaining and losing experience:
  - `gainExperience()` adds its parameter's value to the creature's current experience.
    - If the creature's experience is greater than 1000, then the creature levels up:
      - Increment the creature's level.
      - Heal all damage (a free heal).
      - Reduce its experience by 1000 (i.e., how far is it through the **next** level?)
      - Print out a message to the screen congratulating the user for leveling up. Note that the "650" in this example is calculated as: 1000 – (Creature's experience).

```
DING! Congratulations! You are now level 2.
In 650 more experience, you'll level up again!
```

  - `loseExperience()` subtracts its parameter's value from the current experience.
    - If it would go negative, make it 0.
    - The creature cannot lose a level due losing experience.
- Change `displayInfo()` to display all of the creature's information. For example:

```
Character info for Sir Awesome
Health      25 / 25
Level       1
Experience  0 / 1000
```

- Change `displayInfoShort()` to also show the level. For example:

```
Sir Awesome is level 1 with health [25 / 25]
```

- Implement the `getDamageDealt()` function. Call this function each time the creature attacks to find out how much damage it does this attack. It returns a random number between 1 and (10*level) inclusive.
  - level 1 creature can deal damage between 1 and 10,
  - level 2 does between 1 and 20 damage,
  - level 3 does between 1 and 30 damage, ...

5. Create the Game

The game allows the player to fight monsters to gain experience and level up. Once they reach level 4 they can start hunting for Dr. Evil, who is the end-boss of the game.

General Thoughts:

- Write the pseudo code first; write the C++ code second.
- Do all keyboard inputs using either the prompt functions or the `Menu` class (i.e., use the code in `menu.h` and `menu.cpp`). Do not add `cin` or `getline` statements in `main.cpp`.
- Randomize by the timer so each game is different. Only randomize once per game.
- Each switch statement (for example, ones that handle menu inputs) should use an assert statement to flag an error if an unexpected condition occurs.

Game Play

- At start-up ask the user's name (may be multiple words), and create the player's character (a `Creature` object). The user starts at level 1.
- The inn (main menu) has the following options:
  - **Seek adventure**: User fights a monster. See below for details.
  - **Rest**: Returns the player to full health.
  - **View info**: Displays the character's full info. (Hint: Which member function helps?)
  - **Quit**: Ask the user if they are sure. If so, exit the game.
- Fighting a creature
  - When seeking adventure, the user always enters combat against a monster.
  - The monster is always the same level as the player, and its name is randomly selected from a set of names.
    - To select a name, you must store the possible names in an **array** of strings.
    - Then, randomly choose a name from the array of names.
    - The names may be of your choosing; you must have at least 5 different names.
  - Each round of combat, the user is shown his/her health, and the health of the monster.
  - When in combat, the user may select one of the following three options:
    - **Attack**: Deals damage to the monster. The amount is found by calling the player's object's `getDamageDealt()` member function.
      - Note that the player attacks the monster before the monster can attack. If the monster dies due to the player's attack, it does not counter-attack (dead monster's don't attack).

      - When the monster dies, give the user 450 experience and return the user to the inn.
    - **Pick flowers**: Display some funny text, but does not attack. The monster still attacks.
    - **Run**: Leave combat, returning to the inn (main menu).
  - If the player does not run, and the creature is still alive, the monster attacks.
    - It deals damage to the player based on a call to the monster's `getDamageDealt()` function.
    - If the player dies, then the player loses 450 experience and is returned to the inn with 0 health. (Note that monsters do not gain or lose experience).
  - The user is shown the amount of damage he/she does to the monster, and how much the monster does to him/her.

Winning the Game

- When the user reaches level 4, display an extra item on the main menu: "Hunt for Dr. Evil".
- When this option is selected, the user will fight a level 5 creature named "Dr. Evil".
  - Dr. Evil will always be level 5, even if the player is a higher level.
  - Note that because you can find him at level 4, but he's level 5, the user may want to level up before fighting him.
- If the user kills Dr. Evil, the do the following:
  - Append to the user's name the title "the Heroic". So "Bob" becomes "Bob the Heroic".
  - Display a congratulations message:

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
   Congratulations!       You have killed Dr. Evil!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
You have been granted the title, Miss Awesome the Heroic!
You win! Now feel free to slay more evil monsters if you like!
```

  - Hide the "Hunt for Dr. Evil" option. (He's dead; you can't kill him again!).
  - The user can then continue playing the game.
- If the player dies fighting Dr. Evil, handle it exactly like dying during normal combat: lose 450 experience and return to the inn.

Sample test runs

- See online for sample game sessions:
  - One shows a complete game.
  - One shows how it should handle some incorrect user input.

Game Coding Suggestions

- Create a function to create a monster. Pass it the desired level of a monster, and have it pass back a complete monster object.
- Create a function to handle the inn's menu; pass in the player's object (a `Creature`). Remember to consider if any changes will be made to the object!
- Create a function to handle the combat menu; pass in the two creature objects involved in the combat (i.e., the monster and the player). Remember to consider if any changes will be made to the objects!

6. Deliverables

For this assignment, you will be submitting the planetUML.pdf file as an individual file, but **all your C++ source code files must be submitted in a single ZIP file.**

ZIP the following files and submit to the Course Management System: https://courses.cs.sfu.ca/

- `main.cpp`
- `creature.h, creature.cpp`
- `menu.h, menu.cpp`

Each C++ file must begin with a comment of :

```
// <Name>, <user ID>, student number
```

Once you have submitted, please double check online that all the correct files were submitted.

Please remember that all submissions will automatically be compared for unexplainable similarities. We will also be comparing assignments against what we can find on the internet. Please review the notes from lecture on the expectations for academic honesty.