COMP2404 -"Introduction to Software Engineering"

Tutorial 2

- No late tutorials will be accepted.
- If there are specific instructions for making a function, please follow them exactly. That means that
  - function names
  - function return types
  - parameter types and order

  should all be **EXACTLY** as described. If the script can't read it, you will receive 0 for that part.
- Your code should be neat, readable, and documented. In the event that myself or a TA must mark this manually, you may be evaluated on how easy the code is to understand.

---

1 Submission Instructions

You will write 6 files, `main.cc`, `battle.h`, `battle.cc`, `Character.h`, `Character.cc`, and `Makefile`, and put them in a directory named "tutorial2". You will submit this tutorial to the submission server at http://134.117.133.122:3000/. If you are off campus, you MUST use the Carleton VPN to access the server.

You MUST submit a single .zip file. This file should unzip into a folder, `tutorial2`, and directly inside that folder should be your files (be careful as a Mac will sometimes insert an additional `tutorial2` folder inside, which means the server will not find your files and you will receive an error message and a mark of 0).

One way to zip your files from the command line. Open a terminal in the folder that contains `tutorial2`. Use the command `zip -r tutorial2.zip tutorial2`. **Note the -r. It will not work without -r.** This will zip your file and update it if you change the contents. Submit "tutorial2.zip" to the submission server by the deadline. You will receive your mark immediately, and you may submit as many times as you like, but you must wait 5 minutes between submissions. This is partly to ensure the server does not get overwhelmed, but also to encourage you to write your own tests, rather than rely on the server for feedback.

2 Learning Outcomes

This tutorial introduces you to classes and namespaces.

3 Instructions

3.1 Overview

In this tutorial you will simulate a battle between two `Characters`, an orc from Mordor and a fighter from Gondor. If they fight in Gondor, the fighter will have the advantage, and if they fight in Mordor, the orc will have the advantage. These two places will be defined by their namespaces.

You will make a class `Character` to represent the orc and the fighter. This will include both `Character.h` and `Character.cc` files. In addition you will have files `battle.h` and `battle.cc`. These two files should define two global `fight` functions in different namespaces. Note: `battle` is **not** a class. The `battle.h` file contains the function prototypes and the `battle.cc` contains the function implementations. Finally you will have a `main.cc` file with a `main` function to run the battle.

3.2 Makefile

Your `Makefile` should make two object files, `Character.o` and `battle.o`, and link these object files into your executable file `t2`. For example, to make the `Character.o` file you could use the commands:

```
Character.o:  Character.h Character.cc
    g++ -c Character.cc
```

In addition your Makefile should contain an `all` command that creates the `t2` executable and a `clean` command that removes all executables and object files.

3.3 Character Class

Remember to put header guards in the header file. All member variables are `private` unless otherwise specified. All member functions are `public` unless otherwise specified.

1. Member variables:
   (a) `string name`
   (b) integers for `maxHealth`, `currentHealth` and `damage`.
2. A three argument constructor `Character(const string&, int maxHealth, int damage)`. Use the parameters to initialize the member variables. Initialize `currentHealth` to be equal to `maxHealth`.
3. Make a getter for the `name` member variable.
4. Member functions:
   (a) `void takeDamage(int damage)`. Subtract the parameter value from the `currentHealth`. If `currentHealth` drops below 0, reset it to 0.
   (b) `int strike()`. This is essentially a getter for the `damage` parameter. Return the `damage` member variable.
   (c) `void print()`. Print the name and current health of the character to the console in whatever format you like.

3.4 `battle` global functions

Be sure to include header guards in this header file. You will write two global functions, each in their own namespace. Both have the same function signature.

`void fight(Character& fighter, Character& orc)`.

One should be in the `Gondor` namespace, and one should be in the `Mordor` namespace. Put the function prototypes in `battle.h` and put the function implementations in `battle.cc`.

Every time the characters fight, they each strike one blow. So use `strike` to retrieve the damage and apply that (`takeDamage`) to the other character. If they fight in `Gondor` (that is, using the namespace `Gondor`), the `fighter` should add 1 to their damage and the `orc` should subtract 1 from their damage. If they fight in `Morder` then the `orc` should add 1 to their damage and the `fighter` should subtract 1 from their damage. Write out an appropriate description. For example:

```
>Snarl hits Thor for 3 damage!
```

3.5 main.cc

Make a `main` function. It should do the following:

1. Prompt the user to input the name, max health, and damage for a fighter character (in that order).
2. Initialize a fighter `Character` with those parameters.
3. Prompt the user to input the name, max health, and damage for an orc character
4. Initialize an orc `Character` with those parameters.
5. Print out both characters.
6. Have the characters fight in `Gondor` (make sure the fighter character is the first parameter).
7. Print out both characters.
8. Have the characters fight in `Mordor` (make sure the fighter character is the first parameter).
9. Print out both characters.