Tutorial 2 of 10 - Due Friday Sept. 24th, 11:59 pm

- 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". These should be in a directory titled "tutorial2". You will zip this directory into a file "tutorial2.zip". You must to this from the command line. Open a terminal in the folder that contains "tutorial2". Use the command `zip -r tutorial2.zip tutorial2`. This will zip your file and update it if you change the contents. Submit "tutorial2.zip" to Brightspace by the deadline. DO NOT USE .tar OR .tar.gz FILES. Use .zip only please.

2 Testing Your Tutorial With `t2test.py`

`t2test.py` is a test script that is very similar to what will be used to mark your tutorial (basically we will change the input and expected output). So the mark you see here should be the mark you receive, as long as you did not hard code output. To run `t2test.py`, paste or save this file in the directory that contains your "tutorial2" folder. Open a command line. You may have to make it executable, so type `chmod +x t2test.py`. You may run the script with the "unzip" step, or, if you have not zipped your files yet you may supply a "-nozip" argument.

To have the script unzip `tutorial2.zip` and then test your code, run `./t2test.py`. To skip the unzip step use `./t2test.py -nozip`. When your tutorial is being officially marked we expect a zipped file.

Running this script will generate a file "results.txt". This will have whatever was written to the console as well as the mark.

3 Learning Outcomes

This tutorial introduces you to classes and namespaces.

4 Instructions

4.1 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.

4.2 Makefile

Your Makefile should make two object files, `Character.o` and `battle.o`, and link these object files into your executable file `p1`. 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 `p1` executable and a `clean` command that removes all executables and object files.

4.3 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(string&, int maxHealth, int damage)`. Use the parameters to initialize the member variables.
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.

4.4 `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!
```

4.5 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
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.