Exercise 1 – MMORPG Calculator
MMORPG == (Massively-Multiplayer Online Role Playing Game)

DATE DUE: start of class 5
Hand in through the Dropbox
Make sure your files are working with Visual Studio 2010!

Programs should be written in a modular fashion and built on working code. This first assignment is designed to step you through the process and build your work as you proceed. These are individual assignments – your work must be your own. Please observe the grading guidelines at the end of this document and ensure you have satisfied the requirements.

Exercise 1 – MMORPG
(Massively-Multiplayer Online Role Playing Game)

Calculating Damage Per Second

Most MMORPGs have a large number of statistics for the player: amount of damage per attack, quickness of the attack, chance of hit, damage range, armor reduction and all kinds of other things. One number that is very important to the characters in combat: damage per second (damagePerSecond).

For simplicity, we will assume that the player in one particular MMORPG has three statistics of interest:

- average damage dealt to an enemy for a successful attack
- chance of each attack being successful (between 0.0 for an always unsuccessful attack and 1.0 for an attack that always succeeds)
- amount of time in seconds to make an attack

  damagePerSecond can be calculated as:

  damagePerSecond = (averageDamagePerAttack * chanceOfSuccess) / secondsPerAttack

All of these values can be fractional and should be represented as float or double variables.

Start by writing a program that prompts the player to input the three statistics above, one at a time, and then responds with the calculated damagePerSecond.

A sample run of this program might look like this:

```
Enter the average damage for a successful attack: 40
Enter the chance of a successful attack: 0.75
Enter the number of seconds per attack: 2.0

Damage per Second for this character is 15.00000000
```

Calculating Experience

The reason to fight combats in most MMORPGs is to gain experience points and after accumulating enough you can “level up” and get more powerful. You can calculate the total amount of damage that a player must deal out in order to gain a level. The statistics of interest here are:

- amount of damage needed to defeat a single monster
- amount of experience points awarded from defeating a single monster
- total amount of experience points required to gain a level

The damage needed per level is calculated as:

damagePerLevel = (damagePerMonster * experienceGainLevel) / experiencePerMonster

All of the values in this section are integers.
For this part of the exercise, extend the program in Part 1 to ask the player to input these three values, and then have the program compute and output the damagePerLevel. The program might now look like this:

```
Enter the average damage for a successful attack: 5
Enter the chance of a successful attack: 0.95
Enter the number of seconds per attack: 0.5
Enter the damage to defeat a monster: 50
Enter the experience gained from defeating a monster: 10
Enter the experience required to level up: 250

Damage per Second for this character is: 9.5
Damage per Level for this character is: 1250
```

Final Part

Finally, we can compute the expected amount of time it will take to gain a level, assuming the player is constantly engaged in combat (let's call this secondsPerLevel). This is calculated as:

secondsPerLevel = damagePerLevel / damagePerSecond

Add this final calculation. Note that since damagePerLevel is an int and damagePerSecond is a float, the result will be a float.
HINT: when determining whether to multiply or divide when computing statistics, you can double check your reasoning with the units.
ie. (damage/level) / (damage/seconds) = damage/level * 1/(damage/seconds) = damage/level * seconds/damage
damage over damage cancels out and you are left with seconds/level.

Also prompt the player for his or her character name (as a string), and reformat the output to show the name and calculations in a report. A sample run of the program might now look like this:

```
Enter the character name: Skeeve
Enter the average damage for a successful attack: 16.5
Enter the chance of a successful attack: 1
Enter the number of seconds per attack: 1
Enter the damage to defeat a monster: 75
Enter the experience gained from defeating a monster: 20
Enter the experience required to level up: 400

Damage Report for Skeeve:
Damage per Second for this character is: 16.5
Damage per Level for this character is: 1500
Seconds per Level for this character is: 90.9091
```

Use the above scenario to test your code. **Your program will be tested with this sample data** as well as other data. Use this as a way to find problems with your code **before** you hand it in. **You should always test your code!**

Submission

1. Compress your Visual Studio 2010 project directory into a zip archive.
   (Under Visual Studio 2010/Projects/WhateverYouCalledIt, not the entire "Projects" directory but the WhateverYouCalledIt directory).
2. Rename your zip file as follows: *LastnameFirstnameExercise1.zip* (replacing "*LastnameFirstname*" with your name).
3. Copy this file to your dropbox.

DO NOT RENAME YOUR PROJECT DIRECTORY, just your zip file. The IDE uses the project name internally and your .sln file will not work properly if you rename the project even if your .cpp file is fine.

Grading Guidelines

The design specifications are very specific for this Exercise and you should feel free to use the suggested variable names and calculations that have been given in the design description.

Since this is your first Exercise in this class I have provided you with a more detailed description of what is looked for in programming assignments. The descriptions below emphasize the use of proper coding style and consistent, readable, working code.

Design and Debugging (30 points)

- 30 points if the program works exactly as stated in the design specifications given.
- 20 points if the program compiles, does not crash, displays some of the behavior but has some minor bugs or omissions.
- 10 points if the program compiles, but either crashes part way or has some major bugs or omissions.
- 0 points if the program does not compile, or compiles but crashes immediately.

Use of Variables (20 points)

- 20 points if the program declares variables of the appropriate type, uses variables in the correct calculations and outputs.
- 10 points if the program does not declare some variables appropriately or uses incorrect types.
- 0 points if the program does not declare any variables or has numerous and serious misdeclarations or misuse of variables.

Arithmetic Operators (20 points)

- 20 points if the program correctly calculates the three required outputs.
- 10 points if the program has minor bugs in calculations or is missing a calculation.
- 0 points if missing two or three of these calculations.

Coding Style (20 points)

- 20 points if variables used throughout the program are declared at the beginning, while single-use temporary variables are declared immediately before use, code for similar statements (such as prompting the player for input) are written in a consistent style, variables are named appropriately (you are welcome to use the suggested names given in the design specs), whitespace and indentation is used consistently to make the ***code easy to read***.
- 10 points if there are a few minor style errors. Things like variable names not making clear what they contain, lines written or indented inconsistently, different parts of the program not separated by line breaks. ***Code fairly easy to read***.
- 0 points if lacks naming conventions, does not use indentation and white space, is written inconsistently. ***Code difficult to read*.**

Use of Comments (10 points)

- 10 points if comments are used where appropriate, the program includes a comment block at the beginning stating the purpose of the program, along with its inputs and outputs, all comments reflect the reality of the code and the code contains no superfluous comments.
- 5 points if occasional comments are inaccurate, unnecessary, or necessary but missing, or the beginning comment block lacks some key details.
- 0 points if comments are not used, the comment block at the beginning is missing or lacks major ideas or the program contains many comments that are wildly inaccurate, misleading, or superfluous.