Exercise 2 - Text-Based Artillery Game

DATE DUE: start of class 7
Hand in through the Dropbox
Make sure your files are working with Visual Studio 2010!

The turn-based artillery genre, popularized by the Worms series of games (which were based on the PC game *Scorched Earth* before them), involve players trying to shoot each other in a two-dimensional side-scrolling world with physics. On a player's turn, he or she chooses a direction to shoot at and the initial speed of the shot. The player's goal is to hit one of the other players. The earliest games of this genre were actually played on mainframes with text displays. We will create a simple version similar to these early games.

An online version of the game written in java can be played at the Scorched Earth 2000 website.

Single Shot, No Opponent

First we'll ask the player for an initial angle (of type *int*) and speed (of type *float*), and then calculate how far away the shot lands. We will need to make sure the player enters legal values for the numbers (this is called *validation*), and keep asking the player again until he or she gets it right. Specifically, the angle must be one of the following values: 15, 30, 45, 60 or 75 (smaller angles shoot straight ahead, while larger angles shoot straight up) and speed must be between 1 and 10.

HINT: You may use a do-while loop to ask for the input, and validate it in the while condition. The player may find it easier if you use two loops, one for angle and one for speed, so that one incorrect input doesn't force the player to re-enter both values. One way to validate angle, is to check for three conditions: greater than zero, less than 90, and *angle % 15* equals zero. For speed, check to make sure it is greater than 1.0 and less than 10.0.

Compile and run your program at this point to make sure your loops are working. It is much easier to find bugs, if there are any, when you have only written (or added) a small amount of code.

Next, compute the distance traveled. To compute this, use the formula:
speed * speed * value/ 10, where value is a float that depends on the angle according to the following relationship:

Angle	Value
15	0.5
30	0.866
45	1.0
60	0.866
75	0.5

HINT: A switch statement is ideal for looking up the value, given the angle.

Compute the distance and print it on the screen for the player to see. Sample output follows.

```
Enter angle (15, 30, 45, 60, or 75):  35
That is not a valid angle. It must be one of the five given values exactly.
Enter angle (15, 30, 45, 60, or 75):  45
Enter speed (between 1.0 and 10.0): 0
That is not a valid speed. It must be between 1.0 and 10.0.
Enter speed (between 1.0 and 10.0): 5
Your projectile flew a total distance of: 2.5
```

Multi Shot

Next, we will add a goal. A target is set a distance of 5 unit away, and the player keeps shooting until the player hits the target or gets close enough as determined by the program (say between 4.5 and 5.5). Add a statement at the beginning letting the player know how far away the target is. Statistics to keep track of include how many tries it takes so that it can be output at the end.

First, enclose all of Part 1 in a *game loop*. The loop continues until the total distance is between 4.5 and 5.5. It is your choice as to how you will implement this.

Optional: For bonus points, you can add a break out of the game loop if the player enters a 0 for either speed or angle.

Next, add an *int* variable to keep track of the number of tries that the player has made. After the player guesses correctly, print out the number of tries it took to win. Now a sample run may look like:

```
Try to get a distance of between 4.5 and 5.5.
Enter angle (15, 30, 45, 60, or 75):  45
Enter speed (between 1.0 and 10.0): 5
Your projectile flew a total distance of: 2.5
Enter angle (15, 30, 45, 60, or 75):  15
Enter speed (between 1.0 and 10.0): 10
Your projectile flew a total distance of: 5
You hit the target in 2 tries!
```

Variable Target

Finally, we will vary the target a small amount. We could do this with random numbers, but for this exercise we ask for the player's lucky number and then do something with it to make the distance to the target appear random.

Ask the player for a positive number (*int*), and then add up all the numbers between 1 and that number. This is a perfect use of a *for* loop. Now keep subtracting 9 from the number until it is between 1 and 9 inclusive (you can use a *while* loop for this part). Make the target range within .5 of the number you have computed.

Lastly, improve the text output so that if a player gets the target in a single try it does not print out "You got the target in 1 tries!". The computer is following instructions and so we must correct the instructions to make this grammatically correct. Change this so that it will use the word "try" for success in one try and "tries" if the player used more than one try. Now the output might look like:

```
Enter your lucky number: 4
The target is a distance of 1 away. Try to get a distance between 0.5 and 1.5.
Enter angle (15, 30, 45, 60, or 75):  45
Enter speed (between 1.0 and 10.0): 3
Your projectile flew a total distance of: 0.9
You hit the target in 1 try!
```

Make sure your program works on the above data – this test data is provided for you to ensure your program is working correctly and will be used with other data to test your submitted exercise.

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: *LastnameFirstnameExercise2.zip* (replacing "*LastnameFirstname*" with your names).
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

Design and Debugging (20 points)

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

Use of conditional statements (15 points)

- 15 points if the program uses conditionals where appropriate and with correct syntax
- 10 points if the program has bugs (for example, using = instead of == in an if statement) but uses the conditionals in the right places
- 0 points if the program does not use any conditional statements or uses them incorrectly in many different areas.

Use of Loops (while, do-while, and for) (15 points)

- 15 points if the program uses the correct type of loop where appropriate, with correct syntax, and with the correct initialization and termination conditions
- 10 points if the program has bugs (for example, using while instead of do-while or vice versa, or having incorrect termination conditions for some loops) but uses loops in the right places
- 0 points if no loops are used, used incorrectly, or contains an infinite loop that cannot be exited from.

Use of Switch Statements (10 points)

- 10 points if the program uses switch statements where appropriate and with correct syntax.
- 5 points if it is missing cases or break statements.
- 0 points if not used or used wrong**.**

Use of Blocks and Scope (10 points)

- 10 points if block structures are used properly, variables are in the proper scope
- 5 points if a few variables are declared in a larger scope than they need be.
- 0 points if there are issues of scope, variables declared in the wrong places, unnecessary block use

Coding Style (20 points)

Use of Comments (10 points)

Extra Credit (5 points)

- Extra points if the player can exit the game by entering zero for either the angle or speed, and this part works properly without bugs.