ITGM 315 / Fowler/ Exercise 3 (Functions) – Improved Code/ Spring 2012

Improving the Text-Based Artillery Game with Functions – Spring 2012

DATE DUE: start of class 9
Hand in through the Dropbox
Make sure your files are working with Visual Studio 2010!

Improved Text-Based Artillery Game

Goal – introduce functions and modify existing code

The game from the last exercise had a few limitations. This assignment will make improvements to the game by introducing random numbers. The angle was limited to five options, instead of allowing the player to pick any angle. All the code was in main. Readability and re-use of code is enhanced by functions. This exercise will modify the code to remove these limitations.

Improved Distance Calculation

The formula to calculate distance was given as: speed * speed * value / 10, where value was a float that depended on the angle. The numbers from the table were equal to the trigonometric function sin(2 * angle). So for any angle, the *actual* formula is:

speed * speed * sin (2 * angle) / 10

Replace the distance calculation with this new formula. Also modify the code to input an angle so that it can accept any value between 0 and 90, not just increments of 15. Change angle to be of type *float* instead of type *int,* since it is being used in this calculation.

HINT: The formula above requires you to compute the sine of an angle. This can be done with the function *sin()* from the math library. Include this line of code at the start of your program in order to call this function.

```
#include <math.h>
```

Also include the following line:

```
const float M_PI = 3.14159265;
```

Note that sin() assumes the argument is given in radians, not degrees. Since you are asking the player to give the angle in degrees, you much convert the value to radians before calling sine. The formula is:

Radians = Degrees * M_PI / 180.0

Improved Randomness

Asking the player for his or her lucky number is going to be replaced. Remove that code and replace it with code that generates a random number (technically a pseudo-random number) between 1 and 9, inclusive.

Hint: To do this first include libraries, stdlib and ctime as shown in class.

The following program demonstrates how to produce random numbers for a dice game:

```
#include “stdafx.h”
#include <iostream>
#include <cstdlib>
#include <ctime>

using namespace std;

int main()
{
        // seed the random number generator using the time
        srand(  (int) time(  NULL ) );

        // divide a random number by 6, get the remainder and add one
        // to simulate a six-sided die
        int iDice  =  ( rand() % 6 ) + 1;
        cout  <<  “The dice roll is: “  <<   iDice   <<   endl;

        cout  <<  “Press a number to exit: “  <<  endl;
        int cPause;
        cin   >>  cPause;

        return 0;
}
```

I have provided a zip sample file called ITGM315-DiceRollSampleCode.zip in the _MATERIALS/E3-4-LoopExamples directory if you are having difficulty with the syntax. I have also included ITGM315-DiceRollLoopExample.zip to give you an example of this code but rolling the dice a number of times for those of you who are still getting comfortable with using loops.

Improved Code Readability and Reuse

Normally, the main() function contains just the main game loop and some very basic logic, and everything else is put in custom functions. Let’s modify the code to be a bit easier to understand and maintain.

First, notice that when we ask the player to input the angle and the speed, the code to prompt for, input and validate the angle is very similar to the code that does the same thing for the speed. This shared functionality suggest it may be better to put the code in a function.

Create a function with a heading similar to this one:

```
bool getPlayerInput( string sPrompt, string sError, float fMin, float fMax, float &fResult )
```

Arguments should include a string that is the text prompt to display on the screen, another string to be printed if the player enters an invalid number, and two floats that contain the minimum and maximum allowed values. The final argument should be the valid number entered by the player, *passed by reference* (what does this mean?). C++ provides a way to allow a function to modify the variable, by adding an & sign to the parameters. This is a way to change a value that is sent into the function and not use global variables. Lastly, the function should return a Boolean value, depending on whether the player entered zero (as a signal to stop and end the game) or a nonzero value that’s valid. (You may use an int instead of a bool type if that helps.)

Replace the logic in main() to call your new function when handling player input.

For examples of code using functions, take a look in the _MATERIALS/E3-FunctionExamples folder. You will find similar functions that will help. I have provided:

> ITGM315-FunctionExample.zip – a simple function
> ITGM315-FunctionExample2.zip – a function that uses pass by reference
> ITGM315-FunctionExample3.zip – as above, but also uses strings

Tip: When building code, you can always build a simple case test version in another file. This is a handy way to test syntax and functionality when building a program solution.

Optional: For extra credit, also create a function to calculate the shot distance. Determining function parameters and return values (if any) are up to you, but they should make sense.

Submission

1. Compress your Visual Studio 2010 project directory into two zip archives.
   (Under Visual Studio 2010/Projects/WhateverYouCalledIt, not the entire "Projects" directory but the WhateverYouCalledIt directory).
2. Rename your zip file as follows: *LastnameFirstnameExercise3.zip* (replacing "*LastnameFirstname*" with your names).
3. Copy these 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 for Part B

Design and Debugging (30 points)

- 30 points if the program works exactly as stated in the design specifications given.
- 15 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 library functions (20 points)

- 20 points if the program uses sin, srand and rand where appropriate and with correct syntax
- 10 points if the program has bugs (for example, not converting degrees to radians, or failing to use srand() correctly to initialize) but uses the function calls in the right places
- 0 points if the program does not use any library functions or uses them incorrectly

Use of User-Defined Functions (30 points)

- 30 points if getPlayerInput() is correctly defined, including declaration, definition, implementation and calls
- 15 points if the program has bugs but uses the correct ideas and concepts in the right places
- 0 points if the program does not include any user-defined functions, or uses them incorrectly.

Coding Style (10 points)

Use of Comments (10 points)

Extra Credit (5 points)

- 5 points if there is a function to calculate shot distance that is defined, used in main() instead of the calculation happening in main() directly, and the implementation is correct.