**Programming II**

**Lab 7 Instructions**

Objectives:

* Practice problem solving/logic
* Practice defining/using classes

Lab Prep: A github link is provided on bboard (no zip file to download).

**Task 1**

Open Lab 5’s **game** project and enhance it to shoot at a moving target. Modify the code to do this as follows:

* The ship’s initial position should be near the bottom of the screen, centered horizontally.
* Prevent the user from moving beyond the left/right edges of the screen (i.e., don’t let them move any part of the ship off-screen).
* Prevent the user from moving higher than, say, the lower 20% of the screen.
* Create a new bit map file (in Visual Studio, click Right on the project, select Add->New Item. Next, select the Resource category in the upper left pane, then select Bitmap File (.bmp) from the right pane). Use the editor to draw something to represent an alien enemy. Don’t spend too much time on the art, a simple stick figure is fine. Aliens that resemble Professor Harbert’s face will result in a significant point deduction from your lab score. Save the file as “enemy.png”.
* When the game begins, the enemy should appear and move quickly back and forth across the top of the screen. The enemy doesn’t need to move vertically, just horizontally. Establish limits on the left and right between which the enemy moves (i.e., when it hits the right limit it should reverse direction).
* The enemy should move quickly enough so that it is a challenge to hit. If the missile hits the alien, your code should cout a message (you’ll need to have the console window visible so that you can see the output).
* Here is an easy way to check for a hit:

FloatRect missileBounds = missile.getGlobalBounds();

FloatRect enemyBounds = enemy.getGlobalBounds();

if (missileBounds.intersects(enemyBounds))

{

…we hit the enemy…

}

Note: Usually two sprites will intersect for several frames; in this case the ‘if’ will be true multiple times for one hit.

* When the enemy is hit, the missile should disappear, but the alien should never die (that’s right…there is no way to win this game).
* display (using cout) a “hit count”; ensure that the counter only increases once per hit.

Task 2

Use the Dice project for this task.

Part A: Study the code in the project so you understand what it’s doing (See also the discussion in section 13.14). Answer the questions on your worksheet. You do not need to change any Dice code.

Part B: Write a main function that will do the following:

1. Create a 6-sided die and a 12-sided die.
2. Write a loop to roll each of the dice 1 million times. Do not display the values. In main(), keep track of the frequency that each value occurred.
   1. HINT: Do not use a big ‘if’ or ‘switch’ statement. Use an array of counters (ints) to keep track of the frequencies. Make the array one bigger than you need (e.g., make 7 element array for 6-sided die) so that you can use the die value as an index into the array (e.g., array[1] will contain the number of 1’s, array[2] the number of 2’s, etc.)
3. Display the percentage that every die value occurred. (Hopefully, each possible value should occur at roughly the same percentage, otherwise our die isn’t a “fair” die). Define a “displayPct” function that main calls twice; once with the array of counters for the 6 sided die, and a second time with the 12 sided array of counters. Example output is shown below.

Stats for 6 sided die:

Side 1: 16.71%

Side 2: 16.65%

Side 3: 16.62%

Side 4: 16.66%

Side 5: 16.66%

Side 6: 16.71%

Stats for 12 sided die:

Side 1: 8.34%

Side 2: 8.36%

Side 3: 8.38%

Side 4: 8.35%

Side 5: 8.33%

Side 6: 8.34%

Side 7: 8.32%

Side 8: 8.30%

Side 9: 8.34%

Side 10: 8.30%

Side 11: 8.29%

Side 12: 8.35%

Task 3

Use the Retail project for this task. There are several parts:

Part A

Create a Retail class as described in problem 5 (RetailItem Class), on p. 803. Add a member function named getStockValue() that returns a float indicating the total value of the stock on hand (unitesOnHand \* price). Do not write the main yet, just create the class. For our purposes, you only need a constructor and getter functions; you do not need setters. Add projects files as needed.

Part B

Using Excel, create a csv file containing the data in the table at the bottom of page 803 (bring up Excel, enter the data, then save as a csv (comma-separated-format) file). Write a loadItems function (not a member function) that reads the data from the file. After it reads a line of data, it should instantiate a RetailItem object using the data it read, and then add the instance to a vector of RetailItems. Your program should work regardless of how many retail items are in the file. The loadItems function should accept the vector as a pass-by-reference parameter. (Note: the RetailItem class shouldn’t have any file I/O logic in it, do all of the file input in your loadItems function).

Hint:

* Remember that you used the getline function to read comma-separated data for your final project last semester.
* Remember that you used library functions in program 3 to convert a string to a number.
* Remember that we went over vector examples in class (this code is on bboard). See also 7.12 in your book.

Part C

Write a main that calls the loadItems function to load the data. The main should then display all of the information in the vector (including the stockValue of each item). After all data has been displayed, the program should display the grand total value of all items in stock. Sample output is shown below (align/format the data similar to that shown below):

Description Price On Hand Value

Jacket 59.95 12 719.40

Designer Jeans 34.95 40 1398.00

Shirt 24.95 20 499.00

Total inventory value: 2616.40