CSE 1321L: Programming and Problem Solving I Lab

Assignment 2 – 100 points

Solving Problems

What students will learn:

1) Problem solving
2) Concepts from assignment 1 (I/O, variables, intermediate calculations, operators)
3) Type casting

Overview: For this assignment, you're going to continue writing programs that solve problems, but it will be broken into two phases. Again, start early, practice, and ask a lot of questions.

Similar to the last assignment, you're going to start by writing pseudocode to solve the problem (see Module 2 of the Pseudocode Guide on the CCSE FYE Website for more information); this will be submitted as a single, separate assignment. Later (and very similar to the labs), you'll take that pseudocode to generate source code (in Java, C# or C++) and build an actual program. For this part of the assignment, you will submit three source files. If you need to see what that looks like, refer to the Appendix in Assignment 1.

Naming of your source code assignments is highly important. As such, you must name the file the name of the assignment (e.g. Assignment2A.java, Assignment2A.cpp, Assignment2A.cs). If you're writing in Java or C#, the name of the class must also be that name. For the Java folks, don't include the "package" statements for simplicity.

*When working with the autograder, we're going to award an extra +5 points if all tests work perfectly (max 65/60). Some of you asked us how the autograder works, and it's relatively simple. The autograder pushes "user" input into your program and writes out a file of your program's output. Then, it compares that against a solution file. If things match, you get full points for that assignment. Though there is *some* flexibility in the comparison, the autograder still requires you to match input/output exactly. However, we've simplified the input/output to make it easier to see where the output doesn't match the solution file when you submit.*

If you paste your pseudocode into your source code for comments, we recommend not using Word because it inserts "special" or "hidden" characters (like the quotes around the word special just now) that aren't UTF-8 (which means "standard"). If you want to use TextEdit or Notepad for your pseudocode and then copy/paste from there, that should work too.

Finally, remember that you need to learn everything you can, so don't cheat. While graduating from KSU may get you an interview, the more answers you get correct in that technical interview, the higher the salary is likely to be. You can also show off at nerd parties ☺

<u>Assignment2A:</u> *Hello, Autograder?* These are easy points, but write a program that prints out "Hello, Autograder!". When you write/submit the source code version of this, this is how you can test the autograder (i.e. you should get these points).

```
Sample Output #1:

Hello, Autograder!

Sample Output #2:

Hello, Autograder!
```

<u>Assignment2B:</u> *Yard Work.* If you've ever had to mow the lawn, you recognize that it's back-breaking work. One of the things that drives me crazy is how often you have to refill the lawnmower with gas. Good news, everyone! You just bought an electric mower. For this assignment, you're going to calculate how many charged batteries it will take to cut a rectangular yard based on its width and length and how many square feet you can mow per dot of charge.

Your program should behave like the examples below. **Bold characters** represent user input. NOTE: the printing of decimals may appear different (e.g. 13.5) depending on your language.

```
Sample Output:
Width:
15
Length:
90
Square feet per dot:
100
A yard of 1350 square feet will take 13.5 dots to cut.

Sample Output #2:
Width:
21
Length:
200
Square feet per dot:
75
An area of 4200 square feet will take 56 dots to cut.
```

Assignment 2C: *Can you still see the dots?* It's an interesting time in display technology. When we talk about displays, you most often hear the term "pixel", which is the smallest light-emitting speck on your screen. We use the term "resolution" when talking about the total number of pixels on the screen, but sometimes you want to know how dense the pixels are per inch. A 4K giant display is common, but a 4K display on your iWatch would be a seriously impressive feat! The real question is, if someone came out with a 16K television, would you be able to see the individual pixels? If you couldn't, would you still buy it?

So, in this assignment, we're going to calculate the pixels per square inch, very similar to an online calculator that does it. To calculate pixels per square inch, you need to know how many pixels there are horizontally (left-to-right), how many there are vertically (top-to-bottom), and the diagonal length of the display (upper-left to lower-right). According to the website, you have:

Diagonal Pixels = Square root of (Vertical Pixels^2 + Horizontal Pixels^2)

PPI = Diagonal Pixels / inches

Examples of how to calculate the square root of a number is in the Appendix below. We need to use the sqrt function, which returns a double. Your job is to write a program that reads in the horizontal and vertical resolution of the screen as well as its diagonal length (in inches) and print out the number of pixels per square inch. Note: you cannot have part of a pixel (e.g. 216.9), so you must round down to the nearest whole pixel. The easiest way to do this is type-cast from a float (or double) to an int.

```
Sample Output:
Horizontal pixels:
2880
Vertical pixels:
1800
Diagonal length in inches:
15.1
Pixels per inch: 224

Sample Output #2:
Horizontal pixels:
1920
Vertical pixels:
1200
Diagonal length in inches:
17.0
Pixels per inch: 133
```

```
Sample Output #2:
Horizontal pixels:
4096
Vertical pixels:
2160
Diagonal length in inches:
55
Pixels per inch: 84
```

Submission:

1. For the first week, you will submit a single pseudocode document that contains all three sub-assignments.
2. For the second week, you will submit 3 separate files – one for each of the assignments above.
3. Upload your files to the correct assignment submission folder in Gradescope. Do NOT submit homework in D2L.
4. You will receive two parts of a grade. Your pseudocode will be graded for the first 40%. The autograder will assign the remaining 60% of the grade. Remember, the output of your program must match exactly for the autograder to work.
5. We'll work with you on this assignment if something messes up, so long as you submit by the due date.

APPENDIX – Examples of testing string equality in Java, C# and C++

```java
//========= Java ==========
import java.util.*;

class Main {
  public static void main(String[] args) {
    Scanner scan = new Scanner (System.in);
    float userNumber = 0;
    double squareRoot = 0;
    System.out.println("Enter a number: ");
    userNumber = scan.nextFloat();
    squareRoot = Math.sqrt(userNumber);
    System.out.println("Square root is: " + squareRoot);
    System.out.println("Int version is: " + (int)squareRoot);
  }
}
```

```
//========= C# ============
using System;

class MainClass {
  public static void Main (string[] args) {
    float userNumber = 0;
    double squareRoot = 0;
    Console.WriteLine("Enter a number: ");
    userNumber = float.Parse(Console.ReadLine());
    squareRoot = Math.Sqrt(userNumber);
    Console.WriteLine("Square root is: " + squareRoot);
    Console.WriteLine("Int version is: " + (int)squareRoot);
  }
}


//========= C++ ============
#include <iostream>
#include <cstdlib>
#include <cmath>

using namespace std;

int main() {
  float userNumber = 0;
  double squareRoot = 0;
  cout << "Enter a number: " << endl;
  cin >> userNumber;
  squareRoot = sqrt(userNumber);
  cout << "Square root is: " << squareRoot << endl;
  cout << "Int version is: " << (int)squareRoot << endl;
}
```