EC327 - Spring 2025 - Homework 2

Background

This homework assignment will begin our first foray into C/C++. All of the problems on this assignment will require solutions to be implemented in C/C++.

Note on C vs C++

Since after covering the foundational pieces of C, we will move onto object oriented programming, we will be spending more time in this class using C++ features like classes and the standard template library (STL).

To make this transition easier for us, we will use the C++ compiler and create .cpp files (instead of .c) from the beginning, well before we introduce the more advanced C++ concepts that we will cover later (e.g. classes and standard template library). In addition to making the transition easier, this has an additional benefit of allowing us to use some basic C++ constructs, like the std::string data type earlier on, allowing us to focus on going deeper on other new concepts like structs and functions.

More specifically, we will use C++ version 17 on the lab machines.

For this particular assignment, as part of your solution implementations, unless otherwise called out, you are welcome to use any code that compiles happily with C++ version 17. However, you should not have to reach for the advanced concepts (classes and STL) until we cover them and have explicit assignments aimed at their use.

Important

Please make sure your code compiles and runs as intended on the engineering grid. Code that does not compile will NOT be graded and will receive a 0.

Submission Instructions

You will submit this assignment as single .zip file with the following name:

<first-name>_<last-name>-hw2.zip

So for example:

ed_solovey-hw2.zip.

Your zip file should contain one file per problem in this assignment. Those files should be named like this:

<first-name>_<last-name>-hw2-<problem-number>.cpp
So for example, my problem one file would be:

ed_solovey-hw2-1.cpp

Each of the problems below will have specific instructions around what the text file for that problem should look like.

Note on Collaboration

You are welcome to talk to your classmates about the assignment and discuss high level ideas. However, all code that you submit must be your own. We will run code similarity tools against your submissions and will reach out with questions if anything is flagged as suspicious.

The closed book exams for this class will mostly cover material very similar to what you do on the homework assignments. The best way to prepare for the exams is to do the assignments on your own and internalize the learnings from that process. Cheating on the assignments will very likely result in you not doing well on the exams.

HW2 Structs File

The following file should be called **hw2_structs.h** and should included from all of your solutions in this assignment.

```
#ifndef HW2_STRUCTS_H
#define HW2_STRUCTS_H

#include <string>

/**
* Representation of a line in one dimension, defined by two points in one
dimensional space.
*/
struct Line {
   int pointOne;
   int pointTwo;
};

/**
* @param pointOne one end of a one dimensional line.
* @param pointTwo the other end of a one dimensional line.
* @return newly created Line.
*/
inline Line createLine(int pointOne, int pointTwo) {
   Line line;
   line.pointOne = pointOne;
   line.pointTwo = pointTwo;
   return line;
}

/**
* Representation of a single point in two dimensional Cartesian space.
*/
struct Point {
   int x;
   int y;

   // Overload the == operator
   bool operator==(const Point& other) const {
       return x == other.x && y == other.y;
   }
};

/**
* @param x x coordinate of a point
* @param y y coordinate of a point
* @return newly created point with the given coordinates.
*/
inline Point createPoint(int x, int y) {
   Point point;
   point.x = x;
   point.y = y;
   return point;
}

inline std::string describePoint(Point point) {
   return "(" + std::to_string(point.x) + ", " + std::to_string(point.y) + ")";
}

/**
* Representation of a rectangle as defined by its lower left corner and upper
right corner.
*/
struct Rectangle {
   Point lowerLeft;
   Point upperRight;

   // Overload the == operator
   bool operator==(const Rectangle& other) const {
       return lowerLeft == other.lowerLeft && upperRight == other.upperRight;
   }
};

/**
* @param lowerLeft lower left corner of a rectangle
* @param upperRight upper right corner of a rectangle
* @return newly created rectangle as defined by its lower left and upper right
corners
*/
inline Rectangle createRectangle(Point lowerLeft, Point upperRight) {
   Rectangle rectangle;
   rectangle.lowerLeft = lowerLeft;
   rectangle.upperRight = upperRight;
   return rectangle;
}

inline std::string describeRectangle(Rectangle rectangle) {
   return
       "Rectangle bottom left coordinates: " +
describePoint(rectangle.lowerLeft) +
           " and top right coordinates: " +
describePoint(rectangle.upperRight);
}

#endif //HW2_STRUCTS_H
```

Problem 1 - Rectangle Translation (15 points)

Submission Instructions

Your solution to this problem should contribute a single **cpp** file to your overall **hw2** zip. The **cpp** file should follow the following format:

<first-name>_<last-name>-hw2-1.cpp

So for example, my file would be:

ed_solovey-hw2-1.cpp

Actual Problem

Create a header file called **hw2_problem1.h** with the following contents:

```
#ifndef HW2_PROBLEM1_H
#define HW2_PROBLEM1_H
#include "hw2_structs.h"

/**
* Given a rectangle, return a new rectangle that represents the translation of
the original one by
* xTranslation along the x-axis and yTranslation along the y-axis.
*
* @param rectangle original rectangle
* @param xTranslation translation along the x-axis
* @param yTranslation translation along the y-axis
* @return a new rectangle representing the translation of the original one.
*/
Rectangle translate(Rectangle rectangle, int xTranslation, int yTranslation);

#endif //HW2_PROBLEM1_H
```

Your **<first-name>_<last-name>-hw2-1.cpp** should implement the above **translate** function. You can test your implementation from a **main** function or from tests you set up otherwise. Your submission will be tested against multiple inputs and you should convince yourself that your solution works for the general case.

Problem 2 - Rectangle Scaling (15 points)

Submission Instructions

Your solution to this problem should contribute a single **cpp** file to your overall **hw2** zip. The **cpp** file should follow the following format:

<first-name>_<last-name>-hw2-2.cpp

So for example, my file would be:

ed_solovey-hw2-2.cpp

Actual Problem

Create a header file called **hw2_problem2.h** with the following contents:

```
#ifndef HW2_PROBLEM2_H
#define HW2_PROBLEM2_H

/**
* Given a rectangle, modify it by scaling it around its bottom left corner.
That is, that corner should remain
* unchanged, however the top right corner should be adjusted to reflect a
scaling of its horizontal and vertical
* size by the passed in factors.  The upper right coordinates should be
returned to integer form by truncating the
* scaled result.
*
* @param rectangle original rectangle
* @param xScale horizontal scaling factor
* @param yScale vertical scaling factor
*/
void scale(Rectangle *rectangle, double xScale, double yScale);

#endif //HW2_PROBLEM2_H
```

Your **<first-name>_<last-name>-hw2-2.cpp** should implement the above **scale** function. You can test your implementation from a **main** function or from tests you set up otherwise. Your submission will be tested against multiple inputs and you should convince yourself that your solution works for the general case.

Problem 3 - Function Return Types (10 points)

Submission Instructions

Your solution to this problem should contribute a single **txt** file to your overall **hw2** zip. The **txt** file should follow the following format:

<first-name>_<last-name>-hw2-3.txt

So for example, my file would be:

ed_solovey-hw2-3.txt

Actual Problem

The functions you implemented for the first two problems of this assignment had a different style of returning results of their computation.

In problem 1, the function returned an explicit **Rectangle** struct because the **rectangle** argument to it could not be modified. Where as in problem 2 the function return type was void and the **rectangle** argument was directly modified.

What are the names of each of these commonly used techniques for passing arguments to functions?

Explain the benefits and drawbacks of each of the approaches.

Problem 4 - Binary Fractions (25 points)

Submission Instructions

Your solution to this problem should contribute a single **cpp** file to your overall **hw2** zip. The **cpp** file should follow the following format:

<first-name>_<last-name>-hw2-4.cpp

So for example, my file would be:

ed_solovey-hw2-4.cpp

Actual Problem

Create a header file called **hw2_problem4.h** with the following contents:

```
#ifndef HW2_PROBLEM4_H
#define HW2_PROBLEM4_H
#include <string>

/**
* Converts a non-negative decimal, potentially fractional, number to binary.
The result should be a string of up to
* 32 bits, where
* * the period is not counted towards the 32 limit
* * the string should not have leading zeros to the left of the decimal point
* * the string should not have trailing zeros to the right of the decimal
point
* * whole numbers should not include a decimal point at all
* * bit should be used up to represent the whole number part first and
remaining bits can then be used to
* *   represent the fractional part as accuratedly as possible.
* * numbers larger than the maximum integer that can be represented with 32
bits should return the string:
*      "Input number too large!".
* * negative input numbers should return the string:
*      "Negative numbers not supported!".
*
* @param num decimal number to convert to a binary fraction.
* @return a string with a maximum of 33 characters - 32 bits (1/0)
representing whole and fractional parts of a number
* separated by a "." chracter as appropriate.
```

```
*/
std::string convertToBinaryFraction(double num);

#endif //HW2_PROBLEM4_H
```

Your **<first-name>_<last-name>-hw2-4.cpp** should implement the above **convertToBinaryFraction** function. You can test your implementation from a **main** function or from tests you set up otherwise. Your submission will be tested against multiple inputs and you should convince yourself that your solution works for the general case.

Problem 5 - Rectangle Overlap (35 points)

Submission Instructions

Your solution to this problem should contribute a single **cpp** file to your overall **hw2** zip. The **cpp** file should follow the following format:

<first-name>_<last-name>-hw2-5.cpp

So for example, my file would be:

ed_solovey-hw2-5.cpp

Actual Problem

Create a header file called **hw2_problem5.h** with the following contents:

```
#ifndef HW2_PROBLEM5_H
#define HW2_PROBLEM5_H

#include "hw2_structs.h"

/**
* This function determines if any part, including edges, of two rectangles
overlap.
*
* If the rectangles do overlap, this function indicates that.
* If the rectangles do not overlap, this function returns the distance between
their lower-left corners.
*
* @param rectangleOne one of the two rectangles defined by its lower-left and
upper-right corners.
* @param rectangleTwo the other rectangle defined by its lower-left and
upper-right corners.
* @return "Rectangles overlap." if the rectangles overlap.  Otherwise, return
* "Rectangles do not overlap.  Distance between lower corners is: NN.NN."
where the distance includes up to 2
* digits after the decimal point.
*/
std::string describeOverlap(Rectangle rectangleOne, Rectangle rectangleTwo);

#endif //HW2_PROBLEM5_H
```

Your **<first-name>_<last-name>-hw2-5.cpp** should implement the above **describeOverlap** function. You can test your implementation from a **main** function or from tests you set up otherwise. Your submission will be tested against multiple inputs and you should convince yourself that your solution works for the general case.