CPE 101 Program Assignment 2

Due Dates:

Program: **Friday, October 9th**, at 11:59pm
handin the file **Firetruck.c** to **zwood** in the **csc101p2** folder

Remember, **NO LATE PROGRAMS!**

The Assignment:

Your task is to write a program to help the emergency radio operator decide which fire truck should respond to an emergency phone call. The information the operator will input, is the location of the emergency (fire, accident, etc.) as a two dimensional coordinate. Then the operators will input the location of the two fire trucks and identify which is the closer truck to the emergency. Your program will compute the distance between each fire truck and the emergency in order to help the operator know which fire truck to send to respond to the emergency. The location of the two fire trucks will also be two dimensional coordinates. Your program will need to compute the distance in two different ways. First, compute the distance "as the crow flies", i.e. the shortest distance= a line (also called Euclidean distance) between their current location and the location of the emergency. Second, compute the distance using "Manhattan distance", i.e. the shortest distance a car would have to drive in a city laid out in blocks. Your program will print out both of these types of distances. Your program will then identify the closest truck using the Manhattan distance computation and direct that truck to respond to the emergency. Note that all 2D coordinated will be integer values and all computed distances should be positive values.

Note that although the coordinates of the fire truck and emergency are all integer values, the Euclidean distance between them is not necessarily an integer.

This program includes strict requirements about your solution. You must write all the specified functions as a part of your solution.

Development:

When writing this program, you will need to write test cases to confirm the correctness of your solution. It is always best to start with one piece of a program and make sure it works before moving onto the next. Please develop a function called **EuclidDistance2D** which takes in two 2D coordinates (in other words four different integers) and returns the Euclidean distance between the two points.
Next use the provided **compareNumber** in order to test your **EuclidDistance2D** function.

```
/* compare two numbers – necessary to compare floating
point numbers */
void compareNumber(float a, float b){
  float EPSILON;
```

```
  EPSILON=1e-8;
  if (fabs(a-b)< EPSILON) {
    printf("comparison succeeded\n");
  }
  else {
    printf("comparison failed!\n");
  }
}
```

For this program, you are required to include this test function in your program and you are required to test your Euclidean distance. In your main function you will include the following code to test your **EuclidDistance2D** function. For example:

```
int main(void) {
/* the distance from the origin to the point (3, 4) should
be 5 units away */
 compareNumber( EuclidDistance2D(0, 0, 3, 4), 5.0 );
/* if we translate the above situation by (1, 2) it should
be the same answer*/
 compareNumber(EuclidDistance2D (1, 2, 4, 6), 5.0 );
/* A final test, should be sqrt(2)*/
 compareNumber(EuclidDistance2D (3,4,4, 5), sqrt(2.0) );
}
```

Be sure that these test cases succeed. Once this works, develop a **ManDistance2D** function, which likewise takes in two 2D coordinates and returns the Manhattan distance between the two points. Use the **compareNumber** function to test your results (strictly speaking you do not need to use an epsilon for comparing integer values, but it will not hurt to use this function on integers). For example, now your main would now also include:

```
  compareNumber( ManDistance2D(3,6, 1,7), 3);
  compareNumber( ManDistance2D(1,7, 3, 6), 3);
```

Once you have completed your testing you may comment out your tests so that your final program does not print out the test results, but I expect to find at least three tests per distance functions in your file that you tested which are commented out. Now to complete the program you just need to fill in the correct code to prompt the user to input the data, use your **EuclidDistance2D** and your **ManDistance2D** functions and print out the results!

Here is some example runs with the input and output underlined for clarity only. Note that all 2D coordinates must be positive integer values!:

Example 1:
Sample prompt and input:

Welcome to the emergency response assistance tool.

Please enter the location of the emergency as a 2D coordinate.
Enter the X value of the 2D coordinate: 5
Enter the Y value of the 2D coordinate: 1
Please enter the location of the first firetruck as a 2D coordinate.
Enter the X value of the 2D coordinate: 2
Enter the Y value of the 2D coordinate: 3
Please enter the location of the second firetruck as a 2D coordinate.
Enter the X value of the 2D coordinate: 0
Enter the Y value of the 2D coordinate: 6

Sample output:

The first firetruck is 5 blocks (or 3.605551 units) away from the emergency.
The second firetruck is 10 blocks (or 7.071068 units) away from the emergency.

Please contact the first truck.

Your output should look exactly like the above example. Use the same wording, format and spacing. Note that the output, i.e. the distances of the firetrucks, should appear directly below the text requesting the input as shown in example 2 and 3 below.

Example 2:

Sample promp, input and output:

Welcome to the emergency response assistance tool.
Please enter the location of the emergency as a 2D coordinate.
Enter the X value of the 2D coordinate: 2
Enter the Y value of the 2D coordinate: 2
Please enter the location of the first firetruck as a 2D coordinate.
Enter the X value of the 2D coordinate: 2
Enter the Y value of the 2D coordinate: 4
Please enter the location of the second firetruck as a 2D coordinate.
Enter the X value of the 2D coordinate: 5
Enter the Y value of the 2D coordinate: 6
The first firetruck is 2 blocks (or 2.000000 units) away from the emergency.
The second firetruck is 7 blocks (or 5.000000 units) away from the emergency.
Please contact the first truck.

Example 3:

Sample prompt, input and output:

Welcome to the emergency response assistance tool.
Please enter the location of the emergency as a 2D coordinate.
Enter the X value of the 2D coordinate: 4
Enter the Y value of the 2D coordinate: 3
Please enter the location of the first firetruck as a 2D coordinate.
Enter the X value of the 2D coordinate: 6
Enter the Y value of the 2D coordinate: 2
Please enter the location of the second firetruck as a 2D coordinate.
Enter the X value of the 2D coordinate: 4

Enter the Y value of the 2D coordinate: 3
The first firetruck is 3 blocks (or 2.236068 units) away from the emergency.
The second firetruck is 0 blocks (or 0.000000 units) away from the emergency.
Please contact the second truck.

Turn in your C code (the SOURCE code!) on Friday. Be sure your source code has a comment statements at the top. **In addition please have your source code follow the specified code style guidelines.**

Steps:
1) Compile your program using **gcc -pedantic -Wall –ansi** You must get a clean compile or you will loose credit.
2) Turn in your program to the appropriate handin account using the following procedure. The following example shows how to turn in the program:

1. Transfer the file **Firetruck.c** (NOT the file **a.out** !) to your **vogon** account using **SSH** or **SCP**.
2. Login to your **vogon** account using **SSH vogon.calpoly.edu** .
3. Change to the Unix directory (folder) where you placed the files from **SSH**.

   Do this by typing the Unix command (where you should substitute the directory name you've chosen):
   ```
   > cd cs101
   ```

4. Run the handin script program as you did for Lab 1 (you must type **EXACTLY** as shown, **except replace the X with your section number**):

   ```
   > handin zwood csc101p2 Firetruck.c
   ```

Grading:
The following breakdown will be used to grade your program:

30% correct Manhattan distance computation
30% correct Euclidean distance computation
25% correct input and output
5% code style
10% clean compile (no warnings)

If your code does not compile, it will automatically be awarded a grade of 0.

Addendum to Program 2

Please note that for your Program 2, you must exactly match the example output.

This means that your output must look exactly like the examples provided above. In addition, you will be given test input files and an executable which you can use to compare your output to.

You will use file redirection to input one of the sample input files to your executable:

- firetruck < emergency-test01 > e_test_out01

Then you can use the "diff" program to test the difference between your output and the expected output.

>diff e_test_out01 instruct_e_test_out01