CPSC 1070
Lab Project: Templated Functions
Nov. 25

Introduction

This is a one day exercise to give you your first practice with templated functions. You are instructed to write some templated code for a sorting routine. Demonstrate your program to a TA before leaving the lab.

Getting Started

Download the zip file `trysort.zip` that is linked to the lab schedule for Nov. 25 and unzip it. This will create a directory named `trysort`. Move into this directory. If you execute an `ls` command, it should display:

```
Makefile Sort.h   points.txt   trysort.cpp
```

First take a look at `trysort.cpp`. It is a program that is designed to load a list of floating point numbers from a text file, and store them in an array named `numbers`. After loading, the variable `n` will contain the number of numbers read.

Then, three more arrays are allocated of size `len` = `n/2`, each of a different 2D vector type: `VectorL1`, `VectorL2`, and `VectorInf`. Note that each of these vector types is a struct with `x` and `y` floating point coordinates. The program fills in each of these arrays with consecutive pairs of numbers from the array `numbers`. Each vector type overloads the `>` operator in a different way, because each measures distance in a different way.

A Bit About Vectors and Distance Metrics

A 2D vector is like a point, in that it has x and y coordinates, but instead of a position, a vector denotes an orientation and a length. In this exercise, we will only be interested in a vector's length. In geometry there are many ways to measure length (or distance), and these three vector types use three different approaches.

`VectorL1` uses the *Manhattan Metric* to measure distance. That is, how far you would have to travel if you first moved in the horizontal direction (East/West Streets in Manhattan) the distance x, and then moved in the vertical direction (North/South Avenues in Manhattan) the distance y.

`VectorL2` uses the *Euclidian Metric* for distance, which is the usual measure of distance as it is normally understood. Mathematically it is given by $\sqrt{x^2 + y^2}$.

`VectorInf` uses the so-called *Infinity* or *Maximum Metric*, which is simply the largest of the vector's two components.

Examine the code in `trysort.cpp` and make sure that you understand how the `>` operator is defined in terms of these three different norms for the three vector types.

Coding Project

Now, look at the code in `Sort.h`. In this file there is just a code stub where the `Sort()` routine should be defined. You can fill this in using any sorting method that you like. For example, the Wikipedia page on *Insertion Sort* describes an algorithm that will work well.

Your job is to make a template for this routine that can handle any data type for the array `table` for which the `>` operator is defined. For example, it should work for `int`, `float`, `char`, or the 3 `Vector` types. Once you complete the `Sort()` template, you should be able to compile everything using the `Makefile`, without any changes to the code in `trysort.cpp`. Test your code using the data in the input file `points.txt`. If it works correctly, the printout from a test run should look like this:

```
unsorted:
(2, 3)
(7, 6)
(3, 1)
(-2, -4)
(-1, 8)
(5, 5)

sorted, L1:
(3, 1)
(2, 3)
(-2, -4)
(-1, 8)
(5, 5)
(7, 6)

sorted, L2:
(3, 1)
(2, 3)
(-2, -4)
(5, 5)
(-1, 8)
(7, 6)

sorted, L3:
(2, 3)
(3, 1)
(-2, -4)
(5, 5)
(7, 6)
(-1, 8)
```