1 Purpose

In this lab you will practice sorting a vector of elements.

If you have not already done so, create a sub-directory `Lab10` within your `Labs` directory. This new directory is where you will do work for this lab.

2 Sorting a Vector of Fractions

In class we have talked about the insertion sort algorithm. In this lab you will write an insertion sort for a vector of Fractions. To do so you will need to read the fractions from the file `~cs203/Labs/Lab10/sample-fracs.dat`. The lines in the file contain numerator and denominator pairs, e.g., 1 2. Make no assumptions about the number of values in the file.

2.1 Copy the Files

Copy the files in `~cs203/Labs/Lab10/` into your `Lab10` directory. You will have a copy of the `Fraction` class, a skeleton for a class called `SortedFracs`, a main program, a data file, and a `Makefile`.

2.2 Read Fractions and Display Them

The file `SortedFracs.cc` contains *stubs* for the methods that you need to complete. A stub is the simplest form of a method that will compile. It is a common programming practice to create stubs (placeholders) for methods that you need. Then, fill them in one at at time, verifying that the program works at each step. This is the approach you will use in this lab. The program currently compiles but doesn't do much. Type `make` to compile and then run. After you complete a method, comment out the original `cout` in each of the stubs.

You will be doing all of your work in `SortedFracs.cc`. Begin by completing the method `readFromFile()`. This method should open the file `sample-fracs.dat`, read the fractions and save them in the vector `mFractions`. (Use the `vector` class' `push_back` method).

The `size()` method reports the size of `mFractions`, and `display()` will display its contents. Complete these also, compile the program, and convince yourself that it is working properly. Copy into your `handin.txt` file an execution of your program to demonstrate that the data is being read properly.

2.3 Sort the Fractions

When sorting a list of values there must be an ordering for the values in the list — in this case the ordering is the ordering of fractional values. While C++ has comparison operators defined for the basic data types (`int`, `char`, etc.), these comparison operators are not defined for `Fraction` objects. It is possible in C++, however, to extend the definition of an operator so that it will work for a new type. In our case we want the operator `<` to be defined for `Fraction` objects. If you look in the file `Fraction.cc` you will find the following definition at the end of the file.

```
bool operator<(const Fraction & inLeft,
               const Fraction & inRight) {
    return
        inLeft.getNumerator()*inRight.getDenominator() <
        inLeft.getDenominator()*inRight.getNumerator();
}
```

This definition treats the operator `<` as a function with two parameters — both `Fraction` objects — and returns a Boolean value.

With this capability to compare `Fraction` objects, complete the method `insertVal()`. This method will insert its input parameter at the correct place in the vector so that the vector is sorted. Then change `readFromFile()` so that it uses `insertVal()` to insert values into the vector.

Compile the program and run it to convincing yourself that it is working properly. Copy to your `handin.txt` file a copy of the input file and a copy of the output from your program to demonstrate that the insertion method works.

2.4 Write Results to a File

Implement the `writeToFile()` method. You should write the fractions to the file `sample-fracs2.dat`. Use the original file format for writing.

Compile and run the program and make sure it is writing the values correctly. Add to your `handin.txt` file a copy of the file `SortedFracs.cc`, a sample run, and a copy of `sample-fracs2.dat`.

3 What to Hand In

Print (using `a2ps`, of course) a copy of the `handin.txt` file you have constructed and turn it in as usual.