Computer Science II — CSci 1200
Lab 8
File I/O; File Merge

Introduction

Three major concepts are covered in this lab: command-line arguments, file input and output, and merging. The first two are covered in the first two checkpoints. The third is covered in the last two checkpoints.

Downloads

Download the following four files. The first two will be used in Checkpoint 1. The other two will be used in Checkpoints 2 and 3.

```
http://www.cs.rpi.edu/academics/courses/fall04/cs2/lab08/stats.cpp
http://www.cs.rpi.edu/academics/courses/fall04/cs2/lab08/numbers.txt
http://www.cs.rpi.edu/academics/courses/fall04/cs2/lab08/strings1.txt
http://www.cs.rpi.edu/academics/courses/fall04/cs2/lab08/strings2.txt
```

Once you have downloaded these files, you need not access the network any further. Turn off all network connections.

Checkpoints

1. Start by building a project for the program `stats.cpp` and compiling it. There shouldn't be any problems. Once you've finished consider the following.

   Thus far, when we've wanted to read from a file or write to a file, we've used re-direction. For example,

   ```
   stats.exe < numbers.txt > results.txt
   ```

   would attach the input stream verb*cin* to the file `numbers.txt` and would take what's written to the output stream `cout` and put it in the file `results.txt` (erasing what was already in the file). This works well much of the time, but not if you want to get input from two (or more) different files or write to two (or more) different files.

   A second possibility is illustrated in the program `stats.cpp`. Here the syntax of running the program is slightly different,

```
stats.exe numbers.txt results.txt
```

In this case, `numbers.txt` and `results.txt` become C-style strings that are passed as part of function call *arguments to the main function*. The main function takes these C-style strings, opens the files associated with the strings, and attaches these to stream objects.

At this point, please switch to reading the source code and especially the comments in `stats.cpp`. The explanation of command-line arguments and file streams is detailed in the comments.

To complete Checkpoint 1, you must do two things:

(a) Run the program from a command prompt.

(b) Make a list of the new C++ that appears in `stats.cpp`. Ask questions about what you don't understand from reading the code.

2. In the rest of this lab you will write a program that reads two files of strings and writes out a single file containing the resulting strings in lexicographic order, one string per line. If you look at the contents of the two files, you will notice that each is already sorted. You will ignore this fact in Checkpoint 2, but exploit it in Checkpoint 3 to write a more efficient program.

   In your Checkpoint 2 program, read the strings from each file into a single vector, sort the vector, and output it. The only tricky part to this is reading from the two files and writing to the single output. Focus on this, using the example in `stats.cpp`. Run your resulting program using the command line:

   ```
   merge.exe strings1.txt strings2.txt merged.txt
   ```

   (Use the name of your program executable in place of `merge.exe`.)

3. Create another version of the merge program that does not use sort and does not use vectors (or any other container!). It can only read one string at a time from each file. The resulting program should be faster than the sorting version of the program, although this is nothing you could measure on an example this small.

   Here's the idea of the merge. Read the first string from each of your two input streams. Compare these. Because the two input files are already sorted, the smaller of these two (in lexicographic order — the order imposed by `operator<` on strings) **must** be the smallest string in either file! Therefore, you can output it directly to the output stream and read in the next string from the stream where the smaller value was taken. Stop now and hand simulate this. Show the initial two strings read, which string should be output, and which new string is read.

This same idea can be applied again to the new string just read and to the previously read string that wasn't output. The smaller of the two is the smallest string from either of the two files that has not already been written out. Hand simulate this. The idea can be applied over and over until one of the streams is empty (eof). You can write a loop where each iteration through the loop outputs a single string. The loop should terminate when one of the streams is empty (end-of-file has been found). After the loop is completed, you will need code to input and then output (to the output stream) the remaining strings in the non-empty file.

Use this idea to write a new program to merge two files. Complete this checkpoint and the lab by showing your TA a working version of the program, including the source code. Be especially certain your program works by (a) reversing the order of the input files on the command-line, (b) running the program with one or two empty files.