University of Toronto
APS105H1F — Computer Fundamentals
Lab 4: File I/O with Streams

Introduction

This lab is designed to give you practice with reading from and writing to files using stream I/O objects. At the time this lab is first posted, we will not have covered all the necessary material in lectures, but if you read Sections 6.1 and 6.3 in the text in addition to this handout, you will have everything you need. None of the programs is particularly difficult, and you will find when you are done your programs look very similar to the examples in the text. Please read the entire handout carefully before you start.

You should attempt to complete the assignment *before* your scheduled lab period, as many of you will find it will take more time than will be available in the lab. Please keep in mind that the marking process will begin approximately forty minutes before the end of the lab.

Part I

Write a program to read input from the keyboard, one character at a time, and echo it back to the display. Instead of using the extraction and insertion operators (>> and <<), use instead the `get()` and `put()` member functions, as described in Section 6.3 in the text. Also, use the `eof()` member function to determine when to stop reading the input and exit your program. For example, to use the `eof()` function with `cin`, you can call it as `cin.eof()`. Keep count of how many characters your program reads, and output the count before you exit your program. The count should be on a line by itself, and your count output should be formatted like "`234 characters processed.`". When running your program, type Ctrl-D on the keyboard to signal end-of-file to your program.
Another way to use this program is, instead of taking input from the keyboard, to use I/O redirection to get the input from a file. For example, if your executable is named `Lab4Part1`, then type

```
$ Lab4Part1 < someFile.txt
```

This will cause the contents of the file to be read and processed by your program. Compare the output to that given by

```
$ cat someFile.txt
$ wc -c someFile.txt
```

(use `man cat` and `man wc` to find out what these commands do, if you don't already know).
Finally, try a small experiment—what happens if you instead use >> instead of `get()` when reading from `cin`? Try this when reading the input from a file, as described above, and again compare the output with that of the `cat` command.

Part II

Make a copy of your program from Part I, and modify it. When your program begins, it will prompt the user for a filename and store the entered name in a `string` variable. You will open the named file, read all the characters in the file one at a time, all the while outputting them to the screen. You are to detect if for some reason the file cannot be opened, and print an error message when that happens. Use the `open()`, `fail()` and `close()` member functions, as described in Section 6.1 of the text. You will find that the `open()` function does not accept `string` variables directly, but (assuming your `string` variable is called `fileName`) you can use `open(fileName.c_str())`. This statement will be explained in a later lecture. You will still use the `eof()` function to determine when all the characters have been read from the input file.

Part III

The program you write for Part III will be similar to that of Part II, except you will be *writing* data to a file. Again, prompt the user for a file name, and also ask for a positive integer value (repeatedly if necessary until you get one). You may assume the user only enters integers. We will call this value `n`. You will then open/create the file named, and write all the prime numbers that are less than or equal to `n`, one per line, into the file. It is sufficient to use the insertion operator (<<) to do this, but you will need to devise an algorithm for determining the prime numbers. One simple method is to loop over all value in the range 2 to `n`, test each value, and if it is prime then write it into the file. Write a function `bool isPrime(int x)` to make your loop simpler to understand. Try to make your loop and the `isPrime()` function as efficient as you can (without making the code cryptic).

Warning: If, when testing this program, you input the name of an existing file, it will be overwritten (and its original contents will be lost). Be careful what names you input!

Requirements

Your programs must be in files named `Lab4Part1.c`, `Lab4Part2.c` and `Lab4Part3.c`. There **must** be comments at the start of each file indicating your name, student number, course code (APS105), assignment number (Lab 4), assignment part (1–3) and the date you began work on the file. There should be additional meaningful comments in your source code.

You must be able to compile each program and demonstrate it for the TA who marks your lab.

Submission

Before your lab is over you **must** submit your files using the unix `submitaps105f` command. For example, the command `submitaps105f 4 Lab4Part[1-3].c` may be used to submit all 3 files. The manual page for this command may be read by executing the unix `man submit` command.
The command `submitaps105f -l 4` may be used to confirm that you have successfully submitted your files.