Computer Science II — CSci 1200
Homework 7 — File Crawler
Due: November 15, 2004

This homework is worth 120 points. The solution is due by 11:59:59pm on Monday, November 15. Please note that Test 2 is Friday November 19, so it is in your best interest to start the assignment soon and finish it early. See the handout on homework and programming guidelines for style, submission instructions, and grading criteria. Remember to zip your files together prior to submission and then submit only the zip file.

Overview

Hypertext files, such as those coded in html, contain hyperlinks to other files. When you look at the source code for html files you will see text that looks like

```
<a href="http://www.google.com">Google</a>
```

HTML parsers recognize `<a href` as indicating that what follows is a "hyperlink" and `"http://www.google...` as specifying the hyperlink to a particular file on a particular computer using the hypertext transport protocol. The string `Google` is what will appear when the page is displayed is a browser. The `</a>` ends the hyperlink.

A web crawler scans html files for hyperlinks to other html files. It follows these hyperlinks to the actual files on the same or other computers. The crawler then repeats the process of scanning files, making lists of hyperlinks, and following hyperlinks. It keeps track of files that it has already visited so that it does not visit a file a second time. In addition to searching for hyperlinks, the crawler analyzes and summarizes the text of each file for the purpose of keyword searching.

Our goal in this assignment is to explore a drastically simplified version of the web crawling and keyword indexing problem.

Rensselaer Text Protocol — RenTP

The text protocol your program must work with is very simple. Links to other files will be specified in the text by the simple structure

```
< filename >
```

In other words, there will be a < symbol, 0 or more whitespace characters, a filename containing no whitespace characters, 0 or more whitespace characters and finally a > symbol. There will be no formatting mistakes in the hypertext, meaning in particular that the only < and > symbols appearing the file will indicate links. You will need to be careful of the situation where the < character is preceded immediately by an alphabetic character. See below for hints on this and on parsing in general.

Problem

With all of this as background, we are now ready to specify the actual programming problem. It has two parts. In the first part your program will "crawl" a set of RenTP files (all on your computer), starting from one or more file names specified on the command-line. It will output the names of the files it found (via a link) and successfully opened. This output should include the files that were originally specified on the command-line. The output should be in lexicographic order (order produced by sorting strings using the default `operator<`) with one file name per line. This output will be prefaced by a line:

```
FILES OPENED
```

If there are files that are referred to in a link (or on the command line) but that could not be opened, the program should output a blank line followed by the line

```
FILE LINKS FOUND THAT COULD NOT BE OPENED
```

Then it should output the names of these files, one per line, in lexicographic order.

In the second part your program will read a sequence of keywords from `cin`. For each keyword it will output the five files the keyword appears in most frequently. The order of output will be by number of occurrences of that keyword. Ties should be broken using lexicographic order. If the 5th file output has k appearances of the keyword and there are more files in which the keyword appears k times, then the program should output all of the names of these files. For example if keyword Lincoln appears in one file 8 times, in one file 4 times and in four other files 3 times, then all six of these file names should be output. If there are other files in which the keyword Lincoln appears fewer than three times, the names of these files should not be output. When a keyword appears in fewer than five files, your program

should only output the names of the files in which the keyword appeared. If the keyword appears in no files your program should output the line

```
-none-
```

Preface the output for a keyword by outputting a blank line followed by the line

```
keyword appears in files
```

where `keyword` is replaced by the actual keyword.

See the course homepage for examples and clarifications.

Definition of a Words and Keywords

For simplicity, your program should define words in files as consecutive alphabetical characters. Thus, `pre-computed` is two words. Input keywords (from `cin`) will be just sequences of alphabetical characters; there will be no punctuation and keywords will be separated by at least one whitespace character. Case in the letters of the keywords and in the words in the files you search is irrelevant. Case is relevant in the names of the files referenced in a link. Words in the files and in the keyword input that are less than four letters long should be completely ignored.

Program Design

The design of your program is a significant part of your grade. You must use functions or classes or both as appropriate to create a good, clear program design. Your main program must start with a paragraph-long explanation of the design of your program, including major classes and containers used. This explanation is worth 10 of the 120 points.

You may use any technique from Chapters 0-10 of the Koenig and Moo text, any example code provided in class, and any code from solutions to the first 9 labs and the first 6 homework assignments.

Hints on Parsing

Reading and parsing the input files is slightly different and more challenging than problems you've worked on thus far in the semester. We suggest that you should read and handle one character at a time. In doing so please be aware that the code

```
char c;
cin >> c;
```

skips over whitespace. This is quite dangerous in this case.

One option is to use the function `get`. Any input stream has this function. For example, the following code reads every character from file `foo.txt` and puts it in a string:

```
string s;
char c;
ifstream in_str("foo.txt");
while ( in_str.get(c) ) s.push_back(c);
```

Also, at times you may need to put a single character back on an input stream (when a word ends at `<`). The function `put_back` works nicely in this case:

```
in_str.put_back(c);
```

These stream functions are discussed in any C++ text.

Running Your Program

To keep things as simple as possible, place all of the example files to be searched (downloaded from the course web page) in the same directory as your program executable. This will avoid any problems with file names and paths.