Project 1: Web Page Search

Internet search engines such as Google (http://www.google.com) return a sequence of web pages (URLs) based on keywords provided by a user searching for information on the web. One or more keywords are provided and the web pages returned are typically ordered according to some likelihood of importance. Some preprocessing is often done on current web content to ensure that succeeding search operations are both efficient and fairly accurate.

Assignment:

For this assignment, you will write a C++ program that implements a simplified Web search engine. There will be one keyword per search and the relative importance of a web page is predetermined. The program will take two input files:

**Web page data file.** This file represents current web content and the program will begin by reading in this file and incorporating all the data in its search engine (the preprocessing stage). It will contain several lines of text, where each line contains a web page URL, followed by keywords that are associated to that web page. The URLs appear in this file according to their order of importance. The format of a line in this file is summarized below:

```
<URL> <keyword-1> <keyword-2> ... <keyword-i>
```

The URL and the keywords are strings of characters and they are separated in the text line by one or more spaces. Note that it is possible to have no keywords associated to a URL; i.e., no words follow the URL field in the data line.

**Search file.** This file will have lines containing exactly one keyword each. Each keyword will represent a search performed on the engine. For each keyword read, the program will print a sequence of all web pages that are associated to that keyword.

Output will be directed to a file. Each line in the search file will correspond to a sequence of lines in the output file. The format for this sequence of lines is as follows:

```
<keyword>: <j> web pages found, <stats> keyword lookups.
<URL-1>
<URL-2>
...
<URL-j>
```

where `<keyword>` is a string indicated in the search file, and `<URL-1>` through `<URL-j>` are the associated web pages. The associated web pages must be in the same relative order as they appear in the web page data file. The `<stats>` field indicates the number of keyword lookups performed during the search (described in the next section). Finally, after processing all keywords in the search file, a summary line is printed that provides the total number of keywords processed, the total number of web pages returned, and the total number of keyword lookups performed. The format for this last line is given below:

```
TOTALS: <k> keywords, <w> web pages returned, <l> keyword lookups.
```
Figure 1: The keyword array on the left has a maximum size of 1000 and is sorted by keyword. The web page list on the right contains the pages as they appear in the data file. Each list in the center is associated with a keyword and its nodes link to the web page nodes on the right.

Data Structures:

You are required to use the following data structures for this project:

- You will use an array to store the keywords, and these will be stored in sorted order, so that you can later perform binary search on the array as you process the search file. This means that as a new unique keyword is encountered in the web page datafile, it is “inserted” into the array, so that correct order is preserved. This will be relatively inefficient but tolerable in practice since it will occur during the preprocessing stage. You may assume a maximum of 1000 distinct keywords. The <stats> field for a keyword in the output file is the number of array lookups that are required to find the keyword in the binary search.

- Each array element above will be associated with web pages and these will be represented as a linked list of nodes ordered according to importance (the order in which the web pages occur in the data file). Since it is common that a web page is associated to more than one keyword, you should have a separate linked list that stores all unique web pages, so that the list associated with a keyword have nodes that just point to the appropriate node in this separate list (to avoid having duplicate web page data).

Refer to the figure for a description of these data structures and their interrelationships. You may not use any of the C++ standard templates to implement the data structures specified although you may use the code found in the textbook.
Program Invocation:

The program is invoked as follows:

websearch <datafile> <searchfile> <outputfile>

If the command line contains an insufficient number of arguments or if any of the specified input files does not exist, the program should print an appropriate error message and exit.

Sample Input and Output Files:

In the example below, out1.txt will be produced from the input files data1.txt and search1.txt.

data1.txt

http://courses.cs.vt.edu/~cs2604/ structures algorithms
http://www.sorting.net/ algorithms bubble heap
http://www.buildings.com/ cement structures building architecture
http://www.snpp.com/ simpsons cartoon
http://geeks.net/ cartoon algorithms

search1.txt

cartoon
algorithms
data
building
heap

out1.txt

cartoon: 2 web pages found, 1 keyword lookups.
   http://www.snpp.com/
   http://geeks.net/

algorithms: 3 web pages found, 3 keyword lookups.
   http://courses.cs.vt.edu/~cs2604/
   http://www.sorting.net/
   http://geeks.net/

data: 0 web pages found, 3 keyword lookups.

building: 1 web pages found, 4 keyword lookups.
   http://www.buildings.com/

heap: 1 web pages found, 2 keyword lookups.
   http://www.sorting.net/

TOTALS: 5 keywords, 7 web pages returned, 13 keyword lookups.
Submitting Your Program:

This program is due on September 23, 11:59pm. You will submit a gzipped tar file containing all source files and a makefile to the Curator System (read the Student Guide), and it will be archived until you demo it for one of the GTAs. Instructions for submitting are contained in the Student Guide. Follow the instructions there carefully; it is very common for a student to suffer a loss of points (often major) because she or he failed to include the necessary source code files or the makefile. It is amazingly common for students to omit required header or cpp files or to submit the wrong version of their program. In such a case, it is obviously impossible to perform a test of the submitted program unless the student is allowed to supply the missing files. When that happens, to be fair to other students, we must assess the late penalty that would apply at the time of the demo. To avoid such problems, once you have prepared your gzipped tar file for upload, copy it to a new location, untar it in a new directory, build an executable, and test that executable. If you do that you can at least be sure you’re not submitting an old, incomplete version. You will be allowed up to five submissions for this assignment, in case you need to correct mistakes. Test your program thoroughly before submitting it. If you discover an error, you may fix it and make another submission. Your last submission will be graded, so fixing an error after the due date will result in a late penalty.

Ten early bonus points applies if you submit the program by 11:59pm, September 22. A late penalty deduction of up to 20 points applies if you submit beyond the due date. No late submissions will be accepted after September 25, 11:59pm. You are advised not to submit close to the cutoff times to ensure you receive the proper credit.

Programming Standards:

The GTAs will be carefully evaluating your source code on this assignment for programming style, so you should observe good practice. See the Programming Standards page on the course web site for specific requirements that should be observed in this course. As always, you should practice proper object-oriented design and implementation.

Evaluation:

You will schedule a demo with your assigned GTA. At the demo, the TA will supply your submitted project, and you will perform a build and run your program on the supplied test data. The GTA will evaluate the correctness of your results. In addition, the GTA will evaluate your project for good internal documentation and software engineering practice.

Pledge

Each of your program submissions must be pledged to conform to the Honor Code requirements for this course. Specifically, you must include the pledge statement (provided in the website) in the header comment for your main source code file.