COSC 2320: Data Structures

Homework 5: Hash Tables

Deadline: **Nov. 22, 2014**

1. INTRODUCTION

In this homework you are asked to create a C++ program that acts as a primitive search engine, first searching for keywords in a series of files, storing those keywords in a hash table (indexing) and implementing a quick way of determining what files contain certain keywords (searching).

1.1 Keywords

To find the keywords in a text document you will first need to load the file into a linked list in the memory, as was done in previous homework. From this list you should delete words that cannot be keywords such as "a", "the", "is", etc. This list of non-keywords, commonly called *stop words*, will be provided for you in a special file. The remaining words will be the keywords in your file. This can be added to a common **repository** in memory.

1.2 Indexing

You will need to create a special data structure to hold your results. This structure should have a field for a single keyword, which will act as the index, and a list of the files where this keyword is present. For instance if the word "blue" can be found only in text files "rainbow.txt" and "colors.txt", your data structure might look like:

```
{keyword='blue', files = {'rainbow.txt', 'colors.txt'}}
```

This data structure should be stored in a hash table and you should allow for more files to be added to the structure. In essence after you process the words in a file (as in 1.1) you will search your hash table for each keyword. If the word is found, then you should add the file to the list. If it is a new keyword, you should add it to the hash table.

1.3 Searching

The last part of the program should be a function to efficiently search the hash table and answer conjunctive queries such as searching for all the files where the words "blue" AND "red" are present. Since the results of a keyword search will be a list of filenames, stored as words, you are encouraged to use the **intersection** function that was developed previously.

2. INPUT & OUTPUT

The input of this program is a file, whose name is given in the command-line argument. This file will contain multiple commands to be executed by the program, one command per line. The commands are defined as follows:

- **Stop_word [*file name*]**
  This command specifies the file that contains all the stop words (defined in section 1.1). Stop words are separated by spaces or carriage returns. You should read all the words in this file into a list, and use the list to filter out all the non-keywords in the documents. There will be only one "Stop_word" command in each input file, and it is guaranteed to appear in front of all the "Add_doc" commands in that file.
  The "stop words" can be any combination of numbers and letters, neither punctuations nor special characters will be included.

- **Add_doc [*file name*]**
  Scan all the words in the given file and compare with the stop word list. Discard the word if it can be found in the stop word list (non-keyword), otherwise search the word in your hash table. If it can be found in the hash table, then you should add the file to its list. If it is a new keyword, you should add a new entry in the hash table for the word then add the file to its list.

- **Search [*number of the words*] [*word 1*] [*word 2*] … [*word n*]**
  The first argument is the number of the words to be searched n (n is integer, $n > 0$). The next *n* arguments are the *n* words to be searched. When executing this command, you should search for the files in your hash table that contain **all** the *n* words, then print out the file names you found that can meet the criteria in alphabetical order.
  (It is acceptable that you sort the output by the ASCII codes of the word's characters because it's easy).
  The search is case-insensitive.
  You should print a blank line after all the file names, but if there is no file can meet the searching criteria, print nothing.

3. PROGRAM AND ARGUMENT SPECIFICATION

The main program should be called "**keywordsearch**". The program should be able to take the first argument as the input command file.

The call syntax will be like:

```
keywordsearch.exe cmd.txt
```

Note that the file name will not necessarily be the same every time, so your program shouldn't have the input command file name hard coded.

4. A SAMPLE TEST CASE

"< EOF >" just signals the end of the file, it is not literally a string in the file.

rainbow.txt:

```
red orange yellow
green
blue indigo
violet
< EOF >
```

roses.txt:

```
Roses are red,
Violets are blue,
Sugar is sweet,
And so as you.
< EOF >
```

puppy.txt:

```
I’ve got a yellow puppy,
And I’ve got a speckled hen,
I’ve got a lot of little
Spotted piggies in a pen.
< EOF >
```

remove.txt:

```
a the is was will would be
not if then else to it they
you and or in
< EOF >
```

cmd.txt:

```
Stop_word remove.txt
Add_doc rainbow.txt
Add_doc roses.txt
Add_doc puppy.txt
Search 1 blue
Search 2 blue red
Search 1 amber
< EOF >
```

Program call:

```
keywordsearch.exe cmd.txt
```

Output:

```
rainbow.txt
roses.txt

rainbow.txt
roses.txt

< EOF >
```

5. SUBMISSION REQUIREMENTS

Your submission should be well tested before submitting under Visual Studio 2010 or later versions. You can get a copy of Visual Studio from the UH website using your cougarnet username and password. The URL is http://uh.edu/infotech/php/software/list.php.

We use the UH blackboard system to collect your homework submissions. Before you submit your homework, please make sure to **put everything in a ZIP file** named in the form of **LastName_PeopleSoftID_HW5.zip.**

For example: Zhang_1234567_HW5.zip

The instructions about how to use the blackboard system can be found on the TA’s webpage for this course: http://www2.cs.uh.edu/~yzhang/cosc2320-f2014/

6. GRADING

The maximum grade for this homework is 100.

You will get 15 pts for submitting the homework in time, 10 pts if your program can be successfully compiled.

We will test your program with 5 easy test cases and 5 hard test cases. Each easy test case will worth 10 pts, and each of the hard ones will worth 5 pts.

When testing, we will compare your program's output with the standard output. Therefore **do not print any content on the screen unless required**, avoid any prompt information like "Please enter the input file name:", "The elements in the doubly linked list are:" etc.

Last but not least, **no cheating or plagiarism will be tolerated in any graded submissions.**