CS 214: Systems Programming, Fall 2013
Programming Assignment 4: Search

1 Introduction

In this assignment, you will put everything that you have done together into a simple search tool. For now, your search tool will look much like the `grep` utility.

Your task is to implement a search tool that will load an inverted index produced by your indexer into memory and use it to answer users' search queries. Using the same example from the indexer assignment, if you are given the following set of files:

File Name	**File Content**
boo	A dog name name Boo
baa	A cat name Baa

you would use your indexer to generate the following inverted list and save it to an index file:

"a" → ("boo", 1), ("baa", 1)
"baa" → ("baa", 1)
"boo" → ("boo", 1)
"cat" → ("baa", 1)
"dog" → ("boo", 1)
"name" → ("boo", 2), ("baa", 1)

When you run your search tool, it should read the content of the index file into memory. Then, it should continuously poll for user queries and output the names of the files with matching content. For example, if the user gives the query `dog`, your search tool should output `boo`. If the user gives the query `name`, your search tool should output `boo, baa`.

2 Code Reuse

Your implementation will require the use of inverted-index files produced by the indexer you wrote for Assignment 4. If you could not get the previous assignment fully working, you must do so now. **The search tool program must work with the index files produced by both partners' indexers.** This means that, for example, if you did not get your indexer working, but your partner for the search tool project did, you must fix your indexer rather than simply relying on your partner's. Either indexer's output files should be able to support your search tool.

In order to do this, you should encapsulate the reading and parsing of an index file into a module that is separate from the rest of the search tool. This module should have a fixed interface. You

and your partner should each write a version of this module, conforming to the fixed interface, that reads and parses your own indexer output file.

You should write a makefile that efficiently compiles and links your search tool. The makefile must allow the search tool to be linked with either of the two index file parsing modules, at the user's option.

3 Implementation

Your program must support the following invocation interface:

```
search <inverted-index file name>
```

The first (and only) argument, <inverted-index file name>, gives the name of an index file that your search tool should read into memory. For now, you may assume that the entire index file will fit into memory (you will relax this assumption next week). You should design an in-memory data structure to make the search efficient.

Once `search` has successfully read and process the index file, it should go into a loop asking for queries. It should be able to respond to at least two commands:

sa <term> ...: search for files containing the given terms. A query may contain 1 or more terms. If there are more than 1 term, the search tool should return only files that contain *all* terms in the query. (The query is a "logical and" of all the given terms.)

so <term> ...: search for files containing the given terms. A query may contain 1 or more terms. If there are more than 1 term, the search tool should return any file that contains *any subset* of the terms in the query. (The query is a "logical or" of all the given terms.)

q: the search tool should gracefully shut itself down.

The following example shows how a user of your application would search for all files containing ALL of the words "cat", "dog", and "bird":

```
sa cat dog bird
```

As in the last assignment, you should carefully consider all possible exception cases, outline a strategy to deal with them, and implement your strategy.

4 What to turn in

- A writeup documenting your design, including exception handling **and paying particular attention to the memory requirements of your application**. Your writeup should detail the format with which the inverted index is written into a file.
- A file called hw5-testcases.txt that contains a thorough set of test cases for your code, including inputs and expected outputs.
- All source code (for search tool and both versions of all parts of the indexer) including both implementation (.c) and interface(.h) files.

- A makefile for producing an executable search tool, with multiple targets allowing selection of a compatible indexer.

Your grade will be based on:

- Correctness (how well your code is working).
- Testing thoroughness (quality of your test cases).
- Efficiency.
- Good design (how well written your design document and code are, including modularity and comments).
- Code reuse (that your index works with both indexers).