Computer Science 1 - Program 5
Group Project: Database Queries
Assigned: 3/31/04 (Wednesday)
Due: 4/16/04 (Friday) at 11:55pm (WebCT time)

Objective

1. To implement a binary tree.
2. To implement a complicated linked list in conjunction with a binary tree to solve a common problem.

The Problem

You have recently opened a private library using donated books and funds. However, your library does not yet have a nice program to allow users to make queries to find the best books in the library to suit their research needs. It's important that the search work quickly, and provide reasonable "hits." Luckily, you already have on file some pertinent information about each book in your library. For each book, you have the author's first and last name, the title of the book, and a brief summary of the book that is about a paragraph long. Your searches will simply look for words from a query matching words in the titles of all the books in the library and matching words in the summaries of those books. The number of times the queried words match with words in the title and summary will determine the "score" of a match. As most search engines do, you would like to create a program that orders each "hit" based on its score with the query.

Here is a list of tasks your program must support:

1) Loading a list of books into the database.
2) Handling queries to the database.
3) Adding/Deleting sentences from a summary.

For 10 extra credit points (the program is out of 50), you may implement the following:

4) Using a cache of the last ten queries to speed up queries.

The Twist

In real-world projects, managers often change the specification for the project after a team has started working on the project. To simulate this idea, a small CHANGE to this specification will be posted on WebCT and the course web page on Thursday April 8th by 2pm. If your code has been designed in a good, modular manner, it should not be too difficult to incorporate this change into your code.

Required Data Structures

Each book must be stored in a struct that contains the following components:

1) Author's last name (less than 30 characters)
2) Author's first name (less than 30 characters)
3) Book Title (less than 80 characters - each title will not have any spaces in it. Instead, all book titles will have an underscore in the place of a space for convenient storage in a string.)
4) A binary tree that stores information about the book's summary.

The binary tree must have nodes that include the following components:

i) A string of less than 30 characters (storing a word from a book summary)
ii) An integer that corresponds to how many times the word stored in the string component of the node appears in a summary
iii) A left pointer to a node of the same type
iv) A right pointer to a node of the same type

The whole list of books will be stored in a linked list. The linked list will have the following two components:

i) a pointer to a structure storing a single book entry, described above
ii) a next pointer to a node of the same type

Division of Labor

Each group has two students. One student will be in charge of maintaining the binary tree. The functionality they have to provide is as follows:

1) Inserting a node into an existing tree. (Note that if a "word" is already stored in the binary tree, then rather than a new node being created, the integer counter for the existing node should be incremented.) Also, note that even though a string passed into the insert function may be in mixed case, you should store the string in the tree in **all lower case.**

2) Deleting a node from an existing tree. (Note that if a "word" has a count of more than 1, then this should not delete a physical node in the tree. Rather it should simply decrement the count of the appropriate node it the tree. If the count of a "word" is 1, then the node should be deleted.)

3) Searching for a node in an existing tree. For this function, an integer should be returned, representing the number of times the word being searched for appeared in the summary. (Clearly if no corresponding node is found in the tree, 0 should be returned, otherwise, the appropriate count should be returned.)

The other student will maintain the overall project. Their task involves the following:

1) Loading books into the database. (This involves reading in a file and storing the information for each book into a node of the linked list.)

2) Deciphering queries entered from the user and making the appropriate function calls to determine and display the results of the queries.

3) Editing the storage of summaries of books based upon additions and deletions to summaries. (*Note: you may assume that whenever a deletion is given, that the deleted sentence DID belong in the original summary and that you will never attempt to delete a word out of the tree storing the summary that is not already there.)*

I believe this to be a reasonably equal division of labor. If, for some reason, one person spends less time than the other on their task, they should NOT hesitate to help their partner on the other task.

Input File Format for Option #1

The files valid for this option have a different format than those valid for option #3. *Assume that users only enter in valid files for this option and the other option.* The first line on input will contain an integer, n, the number of new books described in the file. The information for each of the n books will follow this line. The format for each book record will be as follows:

The first line will contain the author's first name, followed by a space, then followed by the author's last name.

The second line will contain the title of the book.

The following several lines will contain the summary for the book. For your ease, these summaries will contain NO punctuation and will have a space and/or newline character between every word, but words may contain both lower and upper case letters. The end of the summary is indicated by the string `"END_BOOK_SUMMARY"`. You are guaranteed that this string does not appear in the text of any summary. Furthermore, you should NOT store this string in the tree for the book at all. The next book summary will start on the following line.

Here is a short sample input file:

```
3
Mark Twain
The_Adventures_of_Huckleberry_Finn
This classic American novel is about the adventures of Huckleberry Finn a young boy travelling along the Mississippi River Though the book has been banned in several school libraries for its diction the book has endured as a very popular piece of literature commonly studied in many classrooms END_BOOK_SUMMARY
Richard Dawkins
The_Selfish_Gene
This book analyzes natural selection for a different point of view than the usual view It describes genes as being selfish in the gene pool and how thinking about genes as attempting to survive in many copies in the gene pool can do a better job of explaining the behavior of species than the usual selfish individual or group model can
END_BOOK_SUMMARY
Dan Brown
The_Da_Vinci_Code
This historical fiction thriller chronicles the adventures of Robert Langdon a Harvard professor who gets caught up in the mystery involving the Priory of Sion an organization reputed of keeping a secret with regards to the very foundations of Christianity
END_BOOK_SUMMARY
```

Query Format

Each query should contain a set of 5 words or less that are connected by one of two logical connectors, either "and" or "or." Here is an example of how a query should proceed:

```
How many words are in your query?
4
What are the four words in your query?
novel
literature
selfish
thriller
Which type of query would you like, (1)and or (2)or?
2
```

You do not need to follow this format exactly, but this is the information that must be gathered from a query. Notice that the mixing of and's and or's is not allowed. Thus, a query either requests books that contain ALL of the words in their summary, or AT LEAST ONE of the words in their summary.

File for Adding/Deleting a sentence from a Book Summary

This type of file has a different format than the file described for option #1. Unlike option one which contains information for more than one book, all files used for this option will contain information for exactly one existing book in the database. The first line of the file will contain the first and last name of the author of the book in question. The second line in the file will contain the title of the book in question. The third line in the file will contain either the string "ADD" or the string "DELETE". The following lines will contain a sentence or set of sentences to either be added or deleted from the summary of the given book based upon the third line in the file. The end of the additions/deletions will be indicated by the string "END_OF_CHANGES". ***You may assume that any file entered for this option refers to a book already in the database.***

Here is a sample input file:

```
Dan Brown
The_Da_Vinci_Code
ADD
The Priory of Sion is an actual society that had Leonardo
Da Vinci as one of its members
END_OF_CHANGES
```

Scoring Matches

An "or" query will be scored as follows:

1) For each time a queried word appears in the title of a book, add 10 points to the score for that book. (Notice that you'll have to take care of this search in a special manner since the original title is NOT stored with blanks!!! Also, just search through the title sequentially, attempting to match each word with each searched word. This is a reasonable technique because we know the title only has a few words.)
2) For each time a queried word appears in the summary of a book, add 1 point to the score for that book.

An "and" query will be scored as follows:

1) If each word in the query appears in the title, add 10 points to the score for that book. (Thus, either 0 or 10 will be added to the score based on this criteria.)
2) Count the number of times each word in the query appears in the summary of the book. Take the minimum of these and add that to the score of the book. For example, if the three words in the "and" query were "hat", "cat", and "the", and "hat" appeared in a particular summary 10 times, "cat" appeared 3 times, and "the" appeared 47 times, then 3 should be added to the score of that book.

Outputting Matches

Only the top five matching books should be outputted for a query. If less than 5 books match the query with a positive score, then only the books matching with a positive score should be outputted. If no books match with a positive score, then a message to that effect should be displayed. For each match the following information should be displayed:

1) The title of the book
2) The first and last name of the author
3) The score of the match between the book and the query

You may decide other issues for displaying the output, along with formatting issues.

***Cache Details (Extra Credit)**

These details are mostly left to you. The information stored in the cache should be quickly accessible, thus, it probably makes most sense to store all the necessary information for a query in an array of some sort. The necessary information for a query is as follows:

1) A list of the words for the query
2) whether the query was an and or or query
3) A list of the book titles, authors and scores for each match

Your cache should store the results of the LAST TEN queries. Whenever you get a query, you must "know" where the tenth to the last query made is stored in the cache and replace it. If ten queries have yet to be made, just add the query in question to the cache, not writing over any previous queries.

Extra Enhancements

You are welcome to make extra enhancements to this program above and beyond the specifications given here. The first natural enhancement to make would be to gracefully handle improper input files and do some error checking that isn't required in the specification. If you make some of these enhancements, some more extra credit may be awarded. This will be up to the disgression of your TA.

Program Write-Up

Unlike other assignments where you just turned in a single .c file, in this assignment you will turn in a document for the TA to read, explaining to them how to use your application. There are some ambiguities in this write up and it is your job to deal with those in a way that you see fit. In your write-up you must describe how you've dealt with issues that are not specified in this problem description. The TA will use your write-up to compile and execute your program. (Thus, you should specify to the TA exactly how to compile your code, and then run the code, telling them how to interact with your program.) Also, your write-up should contain a description of any oddities or known bugs about your application. Please save your write-up in either the file "readme.doc" or "readme.txt"

How to Work in a Group of Two

Though your TA will answer more in depth questions about this topic. Here are some guidelines:

1) Before any coding is done. Sit down with your partner and plan and agree upon the following items:

a) The exact composition of every data structure you will use.
b) The function prototype of every function you plan on writing, as well as a description of what each of these functions do.
c) Where and how some of these functions will be incorporated into the overall flow of the program
d) What separate files will be in the application and which code will be included in which function.

2) Have one or both team members come up with files for extensive test cases before any of the code is written based upon the specification. This usually ensures that more thorough test data is written.

3) Once you agree upon these, start implementing your portion of the code. In order to test your function, you may need to use stub functions. A stub function is one that doesn't actually perform the task of a given function, but syntactically returns a dummy value of the proper type. They allow you to test the flow of the design of a program when functions have yet to be written that must be called during that program. Test functions you have written with driver programs. Driver programs simply make calls to functions with various inputs and check to see if the function calls had the desired effect.

4) Get together and integrate your code. Though this should be seamless if you've planned well, but usually some problems in the code you have written will emerge that you weren't able to find. Also, sometimes compilation issues (for multiple files) may emerge. Make sure you leave some time to deal with these. Once all the basics seem to be integrated, run your program with the test data created in step #2, debug as problems arise and continue to come up with new test cases.

Restrictions

Although you may use other compilers, your program must compile and run using gcc or cygwin. In particular, state which compiler you used in your readme file and how you compiled your code. Your file in your program should include a header comment with the following information: both group member's names, course number, section number, assignment title, and date. You should also include comments throughout your code.

Deliverables

All of your source files (.c files and .h files) and a read me file named "readme.doc" or "readme.txt" should be turned in through WebCT by Friday April 16th by 11:55pm. NO LATE PROGRAMS WILL BE ACCEPTED. So really plan on turning in the program by 9pm on Friday.