Assigned: Sunday, 16 November 2014, **Design Check Due: Monday, 24 November, 11:59pm.**
Final Assignment Due: Sunday, 7 December, 11:59pm

Song Search

Rick Astley

1. Overview

Ever hear a song on the radio and wonder what it is? What is the song title and artist? You've heard a few of the lyrics — you know, "love something something heartbreak" — but how do you find the song? Worry no longer! We will solve this problem by creating a search engine for song lyrics.

But how many songs are there? Well, we have constructed a data file with lyrics for ***170,000*** songs with a total of ***40 million*** words. The data file itself is 200 megabytes.

Soooo...the efficiency, both time and memory, of your search data structure is critical. Since searches are based on words, you can consider N to be 40 million. Even linear searches will be painful, but anything that is $O(N^2)$ will take hours or days to complete.

2. Project Description

Given a word, your program will identify the songs that contain that word. For the basic version of the program you only need to search for a single word. **Note: you will not be matching against the title or artist.**

You will run your program using the command-line, and you will provide the song database as in input file. We will provide you a small test database and you will also be able to test on the large database as well. But, it will probably take minutes to load and process the large database, so most of your testing should be on smaller databases.

For each matching song, your program should print out the artist, title, and **context** of the match. The context should consist of a fragment of the lyrics containing the word, in the following form: print the five words before the matching word, then print the matching word, and then print the five words after the matching word (see the example below).

Clearly, there might be many songs that contain a given word. Part of your job is to **rank** the matching songs and present only the top 10 best matching songs. Matching songs will be ranked by the number of times the word occurs in the lyrics of a particular song (e.g., if a word occurs more times in one song than in another, the first song is ranked higher).

Here is an example run:

```
user_name$  ./songsearch lyrics_fulldb.txt
massachusetts  <—— user input (i.e., the program does not print any prompts)
Massachusetts
Title: Massachusetts
Artist: Bee Gees
Context: Feel I'm goin back to Massachusetts Somethings telling me I must

Title: Massachusetts
Artist: Bee Gees
Context: lights all went out in Massachusetts The day I left her

Title: Massachusetts
Artist: Bee Gees
Context: lights all went out in Massachusetts They brought me back to
```

```
Title: Massachusetts
Artist: Bee Gees
Context: Talk about the life in Massachusetts Speak about the people I

Title: Massachusetts
Artist: Bee Gees
Context: lights all went out in Massachusetts And Massachusetts is one place

Title: Massachusetts
Artist: Bee Gees
Context: went out in Massachusetts And Massachusetts is one place I have

Title: Massachusetts
Artist: Bee Gees
Context: have seen I will remember Massachusetts (I will remember Massachusetts) (repeat

Title: Massachusetts
Artist: Bee Gees
Context: remember Massachusetts (I will remember Massachusetts) (repeat to fade)

Title: Homecoming King
Artist: Guster
Context: sooner you're home Back in Massachusetts To your golden age where

Title: Homecoming King
Artist: Guster
Context: you give in Back to massachusetts To your golden age where

Title: The Flow
Artist: Akrobatik
Context: yo It's Akrobatik y'all, Boston Massachusetts Who's up next? Who's up

Title: Holiday For States
Artist: Allan Sherman
Context: District of Columbia, New Hampshire, Massachusetts, Alabama, South Dakota, North Dakota,

Title: Nifty Fifty United States And Capitals, Yakko, Ani
Artist: Animaniacs
Context: then we go north, to Massachusetts Boston and Albany New York.

Title: Millworker
Artist: Bette Midler
Context: no good millworking man from Massachusetts who died from too much

Title: When The Weight Comes Down
Artist: Black Stone Cherry
Context: fog rollin' on to Boston, Massachusetts Who's got the smokin' gun

Title: 924
Artist: Fifteen
Context: a food pantry in Springfield, Massachusetts In the Mellow House Garage

Title: Amy
Artist: Gary B & The Notions
Context: roam, From the shores of Massachusetts It's how you tuck your

Title: Millworker
Artist: James Taylor
Context: no good millworking man From Massachusetts Who dies from too much

<END-OF-REPORT>
```

In this example, the word "massachusetts" appears 8 times in the Bee Gees song, but only 2 times in “Homecoming King", so the Bee Gees song should appear earlier in the results. In the other eight songs, the word only appears once (and notice that the same song by different artists appears both times, e.g., *Millworker* by Bette Midler and also by James Taylor). Also note: the order for the songs that have the same number of occurrences does not matter. **Only the top ten songs should be printed, but all lyrics that match in those top ten songs should be printed.**

3. Implementation Specifics

The key problem in this program is managing the scale of the data. As with our previous programs, you will first need to read in the data from the lyrics file. The difference in this program is that you will need to store the information in an efficient structure that (a) doesn't require tons of memory, and (b) makes answering user queries fast.

You will probably use a variety of data structures: hash tables (or a trie), sets, and sequences. You must use the structures you have built yourself in class, or that we have provided to you in other assignments. However, since time is short, you may use any of the existing data structures in the standard template library. The only data structure you are allowed to use from the standard library is the **`vector<>`** data structure, which we will discuss in class. You may **not** use any other data structure from the standard template library (e.g., hashmaps, queues, etc.).

4. Design Check

Before or during your regularly scheduled lab time next week, you must present your proposed design to one of the TAs and discuss the pros and cons (you do not need to set up a meeting, but you need to go to office hours or speak with your TA during or after lab). Your design should describe:

- A high-level description of how you will store the lyric information in terms of hash tables, vectors, sequences, etc.
- A sketch of the song search algorithm, showing how you will identify and rank the matching songs.
- You will need to Provide your design document by Monday**, 24 November, 11:59pm.**

Things to think about:

- What kind of data structure will make it efficient to search based on words?
- How can you represent and retrieve both the song information (title and artist) and the context (part of the lyrics themselves).
- We will need to keep track of multiple instances of a word, both within one song and across multiple songs. How can you do this?

5. Running and Testing Your Code

When you run your code from the command-line, you will be able to enter search words as in the example above. In order to stop the program, you type **`<BREAK>`** as the search word.

Keep in mind that your program should not print any prompts to the screen to request information from the user. However, if you do want prompts that only show up when a user is entering your program from the command line (vs from a script), you can create a command-line only version of the prompt as follows:

```
if ( isatty(0) ) {
        cout << "Enter a word: ”;
}
```

6. Input Data

Full data file: `/comp/15/public_html/assignments/hw6/lyrics_full/lyrics_fulldb.txt`

Small data file: `/comp/15/public_html/assignments/hw6/files/rick_db.txt`

Basic code for reading the file: `/comp/15/public_html/assignments/hw6/files/read_lyrics.cpp`

The full data file is very large (200MB), so you should not try to copy it to your home directory. I strongly recommend you develop the program using the small data file, which has the same format, but only contains a few songs. When you do want to use the huge data file, you should create a "symlink" in your working directory, as follows:

```
ln -s /comp/15/public_html/assignments/hw6/other_lyrics/lyrics_fulldb.txt lyrics_fulldb.txt
```

The symlink will allow you to read from the data file as if it was in your working directory, without taking up 200MB of your own disk space. (Note: there are also three other relatively small databases in the **`other_lyrics`** folder: **`13300Songs.txt`**, **`2316Songs.txt`**, and **`poems.txt`**.

The format of the files is very simple. The first line is the artist name, the second line is the song title, and subsequent lines contain the lyrics. Songs are separated by a special token "`<BREAK>`" on a line by itself. Here is a snippet:

```
Rick Astley
Cry For Help
She's taken my time
convinced me she's fine
but when she leaves I'm not so sure.
It's always the same: she's playing her game
and when she goes I feel to blame.
Why won't she say she needs me?
...etc...
<BREAK>
Rick Astley
Hold Me In Your Arms
We've been trying for a long time to say what we want to say
But feelings don't come easy to express in a simple way.
But we all have feelings
we all need loving
And who would be the fool to say that if you -
Hold me in your arms
...etc...
```

We have provided you with code that reads the file, but does not do anything with the data. You will need to augment this code with your own code to store the information.

7. Other Comments

- The `Makefile` (provided) will create one executable file:
  - songsearch

- We have provided you a `read_lyrics.cpp` file (and .h file), and a simple `main.cpp` file. We have also provided a hash function (not a hash table — you would need to create that yourself). See section 8 below for details. You will need to create the rest of the interface yourself. You may find it easiest to simply copy the contents of `read_lyrics.cpp` into a class file and make the read function (and "alpha only" function) part of your class.

- We have also provided (in the **`read_lyrics.cpp`** file) an "**`alpha_only()`**" function. This function converts a string with uppercase letters and punctuation into a string that is only lowercase and does not have punctuation. Your searches will be based on the lowercase, non-punctuated lyric, although you will have to return the true (possibly punctuated and uppercase) lyric for your result. Example: If a lyric was "We're going back to good 'ol California!", then a search for "we're" and "were" and "ol" and "'ol" and "california" would all match the line.

- The output from your program should include `<END-OF-REPORT>` on a line by itself, in order to indicate that the output has completed.

- As always, direct any specific questions to the Piazza class site.

- Hash function for strings

Bob Jenkins is a programmer who has developed a number of nice, widely-used hash functions for strings. We have provided version of his code that will work nicely with C++. You can use it for your own hash table implementation. You can find the files here:

```
/comp/15/public_html/assignments/hw6/files/hashfunc.cpp
/comp/15/public_html/assignments/hw6/files/hashfunc.h
```

The only function you need to use is hash_string. You give it a string and it returns a large hash code. You must then mod or mask this number to get it into the range of the size of your hash table.

```
uint32_t hash_string(const std::string & str);
```

Note that the `uint32_t` data type is just an abbreviation for an unsigned 32-bit integer. You can declare a variable of type `uint32_t` and then mod it down to a regular `int`:

```
uint32_t hashcode = hash_string(a_word);
int index = hashcode % size;
```

8. Low Level Details

Getting the files

The files are located at **`/comp/15/public_html/assignments/hw6/files`**

To get the files: **`mkdir hw6 && cd hw6 && cp /comp/15/public_html/assignments/hw6/files/* .`**

Using Eclipse (assuming you have followed the steps above to get the files):

1. See online video https://www.youtube.com/watch?v=BbnLZ2oqrD0
2. If in the lab, simply open Eclipse, and then start following from step 6 below.
3. If at home, ensure that you have installed (Mac) XQuartz, or (Windows) Cygwin (see here: https://www.youtube.com/watch?v=BbnLZ2oqrD0) or (Win) putty and XMing (http://sourceforge.net/projects/xming/).
4. ssh to the homework server using the -X argument:
   ssh -X your_cs_username@homework.cs.tufts.edu
5. start eclipse by typing “eclipse” (no quotes)

6. Use the following steps to set up your project (you’ll get used to the steps quickly, steps A & B only need to be done once at the beginning of the course):

A. Default location for your workspace is fine
B. Click on “Workbench”, then click “Window->Open Perspective->Other”, and choose C++
(note: if C++ is not listed, close Eclipse and re-open by typing Eclipse)
C. Window->Preferences
General->Editors->Text Editors
Displayed tab width (8) (recommended)
Show print margin: 80 col
Show line numbers
C/C++ -> Code Style
Select K&R [built-in] and then “Edit”, then rename to “K&R 8-tab”
Change Tab size to 8 (recommended)
Click “Apply” then “Ok”

D. File->New C++ Project
Fill in project name (no spaces!) IMPORTANT: the name CANNOT
be exactly the same as your folder name. I suggest to
simply put name_project (with the _project) for all project
names (e.g., hw6_project)
Use default location should be checked.
Select Empty Project->Linux GCC
Click Next
Click "Advanced Settings"
(there is a bug in the setup that means you MUST follow steps
1 and 2 below in order):
1. On the left, under C/C++ General->Paths and Symbols (left hand side)
(you may have to click on the triangle next to C/C++ General)
Select "[Debug]" at the top for Configuration
Source Location Tab
Click "Link Folder"
Check "Link to folder in the file system"
Browse to find your folder.
(should be something like h/your_username/hw6)
Click OK
Click Apply

2. C++ Build->Settings (may have to click on the triangle)
For Configuration (at the top), select [All configurations]
GCC C++ Compiler: Command should be "clang++" (no quotes)
GCC C Compiler: Command: clang
GCC C++ Linker: Command: clang++
Click "Apply"
Click OK

Click Finish

E. Click on the white triangle next to your project name.
The folder you linked to should be listed. Click on its
triangle.
The files in that folder should be listed.

F. Test the build by clicking on the hammer in the icon bar.
You should see some compiling messages in the Console window
at the bottom. (Things like, "Building file...; clang++, etc.)

Compiling and providing

1. Create your *own* `Makefile`, using the examples we have had in class. We have given you a template Makefile with the files themselves removed.

   - To make:
     ```
     make
     ```

2. To submit your design document: at the command prompt, enter "`make provide_design`"

3. To submit your completed project, enter "`make provide`"

9. FAQ

1. How many songs do we return?
   a. You return **up to** 10 songs, if there are at least 10 songs that have lyrics that match the search term.

2. How many lyrics per song do we return?
   a. All lyrics for each of the top ten songs.

3. What if there are 11 songs that match?
   a. Only return the top 10 songs

4. If any of our Top 10 results have the search term in them multiple times (which presumably most of them will), do we need to print out every hit within each song, or just one? If just one, does it matter which one?
   a. Yes. Print out each hit, even if it happens more than once on each line. See question 8 below — you can think of the song as one long list of words, and it is not broken up by lines.

5. What if the 10th and 11th song both have one match? Which one do we return?
   a. It doesn't matter — you are free to return either song as the 10th song.
   b. In other words, ties between songs don't matter — one person may return one song, and the other may return the second matching song with the same number of hits.
   c. This is also true for earlier matches — let's say that there are fifteen songs that match lyrics, and each one only has one match. You can return any of those songs, in any order (but you have to return 10 of them).

6. What do you mean, "print the five words before the matching word, then print the matching word, and then print the five words after the matching word"? What if the word was at the beginning of the song?
   a. See the example above.
   b. If the word you are searching for comes near the beginning or end of the song (e.g., if it is the first or second word of the song), you only need to print the words back to the beginning (or to the end) of the song. E.g.:

      A song (Rick Astley, *Together Forever)* starts as follows:
      If there's anything you need
      All you have to do is say

      If you search for "anything", you would return the following:

```
Title: Together Forever
Artist: Rick Astley
Context: If there's anything you need All you have
```

Notice that you only print out two words before "anything" because there are only two words in the lyrics before "anything". You do print five words after "anything."

7. Do we return results based on partial searches? E.g., if we search for "any" and the lyrics have the word "anything", is that a hit?
   a. No. Only search on full words.
   b. Remember, though, punctuation is ignored. So, a search for "any" would match "any", "Any", "Any!", "any," etc.

8. So the lyrics of the song are not broken up by line?
   a. The song gets read in word-by-word, and lines are ignored. So, you can think of a song as one long string of words.

9. Is our program supposed to work in a loop?
   a. Yes. You should continue to accept searches until the user types "**<BREAK>**", at which point your program should clean up its memory and exit.
   b. The basic flow is as follows:
      i. Read in the lyrics
      ii. Start a **while(!done)** loop that waits for a search term from the user.
      iii. Find the lyrics based on the search term
      iv. Continue with the while loop
      v. When the user types "**<BREAK>**", exit the program.

10. Can you explain more about how to use hash_func.cpp?
    a. Here is a high-level overview of creating a hash function for your program (using a linear probe):

```
    void insert(string key,obj *value) { // key is a string, val is a pointer to some object
                                         // that contains the key as well

    if (load_factor() > LOADMAX) expand_table();

    uint32_t hash = hash_string(key) % table_size;

    while (table[hash] != NULL) {
            hash = (hash + 1) % table_size; // linear probe
    }

            table[hash] = value;
    }

obj *find(key) {

            uint32_t hash = hash_string(key) % table_size;
            while (table[hash] != NULL && table[hash]->key != key) {
                    hash = (hash + 1) % table_size; // linear probe
            }
            return table[hash];
}

void expand_table() {
            1. create new table with twice table_size
            2. initialize the new table to NULLs
            3. swap the new table pointer and old table pointers
            4. change table_size to twice the table_size
            5. rehash all the old keys (which you get by walking through the
               old table
            6. delete [] the old table
}
```

11. In the rick_db example, there are some lines which are not actually lyrics, such as (CHORUS). We can remove the punctuation for this using **`alpha_only()`**, but is it okay to treat this like any other word in the lyrics?
    a. Yes, treat ALL words as lyrics. The database is not perfectly clean.

12. I was just wondering what we should expect our program memory usage to be because mine seems to use about 3gb when working with the complete data base. Is that too large? Should I try to make it smaller?
    a. 3GB sounds about right -- my smallest attempt takes 2GB, and the largest takes 4GB.

13. What if the user searches for “Astley”. Do we return all Rick Astley songs?
    a. No — you only search for lyrics. An “above and beyond” could search for song titles or artists, but do not do that for your regular submission.

14. Can a user search for a phrase, e.g., “I will always”?
    a. No. Searches should be for single words only. Hyphenated words will match the whole hyphenated word. E.g., if a lyric was “high-tech” then “hightech” and “high-tech” would both match.

15. How does alphaOnly handle numbers?
    a. The "alpha" in "alphaOnly" will not strip digits (so I guess it should be called “alphaNumOnly”). You should be able to search for “123”, for instance, and it would return a hit for “1up”.

16. Is it legal to have a pointer be the return type of a function?
    a. Absolutely — you should not be passing around your database by value (it will be WAY too slow). You can use either pointers or references.

17. If I use a Trie to hold the lyrics, do I have to account for numbers or uppercase letters, or punctuation?

   a. You will need to account for numbers, but not punctuation or uppercase letters. You will also have to store the full lyrics with punctuation, but you should not do that in a Trie.

18. What is the "**`show_progress`**" variable all about in **`read_lyrics`**?
   a. If you call **`read_lyrics()`** with **`show_progress`** as true (e.g., **`read_lyrics("rick_db.txt",true)`**), the **`read_lyrics`** function will print out the song and artist regularly, but not every time (change the "**`% 10000`**" to something smaller if you are using a small database). This is a good debugging tool — you can see how far along your database has been read.

19. What's the desired behavior if our search returns no matches?
   a. The program should print "**`<END-OF-REPORT>`**" and a newline (no quotes).

20. How quickly should we expect our data structures to create? I know we want our searches to be very fast, but how long should we be expected to wait when we run our program until we are able to enter input.
   a. I've written the program about five times now, and my best time for going through the whole database on the Homework server is about two and a half minutes. The worst is about five minutes. If it's longer than about ten minutes, we won't be able to grade it.

21. Are there other small- or medium-sized databases we can test our code on?
   a. Yes. See section (6) above ("Input Data").