

CST8130: Data Structures

Final Project: Concordance Generator

Overview

You and your partner will create a concordance generator. Your solution will be of your own design; it need not look like the solution that I provide on my website.

Your concordance generator must implement the following functionality.

- Read any arbitrary body of text from a text-oriented file. (Words are delimited by spaces and punctuation. Paragraphs are delimited by newline characters.) I downloaded several public domain titles from the net (e.g. **Dickens**: *A Christmas Carol, Oliver*; **Robert Louis Stevenson**: *Treasure Island*; **Shelly**: *Frankenstein*; **Swift**: *Gulliver's Travels*; **Shakespeare**: *Macbeth*; **God**: *The Bible, The Koran, The Bhagabad-gîtâ*.). Some are quite large and would exercise a **concordance generator**. The text files would probably have to be sanitized (that is, extraneous prefacing and trailing text might have to be deleted and paragraph boundaries normalized). I have already edited the novel *Oliver*.
- Build an index of all words. Each index entry will have a count of the number of words and a list of byte offsets for the start of the word. Set timers to see how quickly the **concordance generator** can be built (for races and the award of prizes). Also award prizes for the best text-oriented user interface, (TUI) and graphic user interface (GUI).
 - It could also track paragraph offsets (paragraphs numbered ordinally from 0) with a **vector** or **list** of these offsets.
- After the concordance is built, a user can type a word, be presented with options to pursue research on the text.

Detailed Design

The detailed design is up to you, but I have a few recommendations.

- Read the entire text file into a single dynamically allocated buffer. If the file is a megabyte in size, you can easily allocate an array of **char** large enough to read the entire file at once (that is, an array of **char** dynamically allocated with **new**). You can open the file, with the older FILE data type and the related functions: fopen(), fseek(), ftell(), fread(). To simplify, text files will be stored as **binary** to avoid the problem of text translation of CR/LF to **newline**.
- Use STL containers. The words can be stored in any of the following containers: a **set**, a **map**, a **vector** (sorted), or a **vector** (implemented as a **hash table**). Each instance of a word will have one or more occurrences of within the body of the text. These occurrences can be stored in a **list**, or a **set**, or a **vector**.

Use STL **set** and **list** to manage concordance. **Set** captures individual unique word entries. **List** captures each occurrence (in sequence) with pointer offset

- Open file, read length (fseek() and ftell()). To simplify, text files will be stored as **binary** to avoid the problem of text translation of CR/LF to **newline**.
- Allocate space for one massive buffer to read entire file.
- Read with fread() into buffer.
- Process one word at a time, using a pointer into massive buffer
 - Scan to end of next word
 - If not found in **set**
 - Create new **word** object (**word** members: **string:word**, **int:count**, **list:occurrences**. Each node in **list:occurrences** contains an **int:offset** (into buffer) and possibly an **int:paragraph_offset**.
 - First **list:occurrences** node added for current buffer offset (and possibly **paragraph_offset**).
 - Else (found in **set**)
 - increment **count**. Tracks count even with high frequency words.
 - if **count** over threshold
 - if **list:occurrences** populated
 - remove all nodes from **list:occurrences** (keep **word** object with its **count**)
 - Else (**count** not over threshold)
 - insert **list:occurrences** node with buffer offset (and possibly **paragraph_offset**).
 - Concordance now built. Program allows for user to research, looking up words and displaying matches. There are enormous opportunities for sophisticated enhancements.