

*Final Term Project: Optimized Hot-Phrase Searcher*

**Prelim II** will be held on **Wednesday, Nov. 30**, in class period. It will cover up through Chapter 9 of the text, minus hash tables but including the `set` and `map` data structures, and assignments up through this one.

**Reading:**

For next week (Monday's lecture), read Chapter 9 focusing on hash tables.

**Short task statement:**

First compile a list of phrases into a `WordTree` data structure, technically called a `trie` but going by words rather than by individual characters. Every node holds a single word except the root, which has a large set of links (can use `vector` or `set`) to nodes holding the first words of phrases. Every child of the root has links to nodes holding the second word of phrases with the given first word (if any—a first word like `Satisfaction` might have no other phrases that begin with it), and so on. A node that completes a phrase will set a Boolean field saying so, and this will trigger operations that compute various kinds of hot-score for the matched phrase. Note that such a node can still have children, for instance there are books titled `The Greatest` and `The Greatest Generation`. One member of the team will program this class *individually*. The tree class must furnish an *iterator* class; the client will use iterators to track the progress of sequences of text words through the tree.

The second member of the team is individually responsible for a `PhraseHeap` class that is used to compile “Top  $k$ ” lists for various combinations of phrase file and text file and scoring criterion (number of words, number of chars as on Assignment 6, and also some partial-match criteria described later). This member is also individually responsible for upgrades to the `Phrase` and `HotPhrase` classes to use with the heap. The heap class must use an array implementation (i.e., `vector`) and must program the  $O(n)$  time `make_heap` routine in supplementary notes on the course webpage (covered in Wed. 11/16 lecture). (Calling the STL method by that name is OK for partial credit.) The “Heaper” will program non-member functions to use with the heap.

Team members will work *jointly* on the client file. This will first download text file(s) from the Web. For each word read, the client will *introduce* a new iterator if that word is a child of the root, and will *update* the already-present iterators (if any). For each already-present iterator, if the word read is a child of its node, the iterator is moved there, else it “dies” and is removed. If and when an iterator is moved to a node that was pre-compiled as completing a phrase, the score for that occurrence of the phrase is computed, and added to a running tally which should be maintained in the `HotPhrase` object for that phrase.

Upon finishing the text file(s), all leftover iterators are ignored, and the client goes about using the heap to compile lists of the top  $k$  phrase-textfile combinations, for  $k = 10, 20, 30$  and the several scoring criteria as above. Outputting these lists with some clear description completes the tasks.

**Points:** The `WordTree` class is 120 pts.; the `PhraseHeap` class is 90 pts. plus 30 for upgrading `Phrase` and `HotPhrase`. (The upgrade can be applied to your own or partner's versions or the answer-key versions.) The client file is 120 pts. applied to both members, and report questions to be detailed are 60 more, for a total of 300 pts. to each team member.