

Reminder, the **First Prelim Exam** will be on **Wednesday, Oct. 12**, *in class period*. It will be closed-book, closed-notes except for one notes sheet. It will cover the domain of the first five assignments and up through chapter 3 of the text—more specifically, up thru what was covered by the end of today’s lecture. This assignment has *only an online-submission part*, no hardcopy.

Reading. Same as last week: read/review Chapter 3, and read up through section 4.4 as it relates to concepts in Chapter 3. There is a lot of material in Chapter 4, and it will take up much of October. The lectures next week will by-nature draw from both chapters.

Mini-Project: Hot Phrase Scores

Short task statement: Do the task of Assignment 4, but searching for whole phrases with punctuation stripped off as in Assignment 5. Instead of just counting occurrences, compute two kinds of “hot scores”: the number of words matched, and the total number of characters in those words. We will use Rolling Stone’s list of 500 Top Songs for the hot-phrases, and will pull various texts off the Web. For now, “occurrence” will mean equality—see questions below for a preview of the idea (to be implemented later) of matching some of a phrase/song-title and/or some of an individual word. Finally sort the phrases (song titles) found to occur and print them with the highest number-of-words hot score first, and then an overall hot score defined to be the sum of those scores divided by the number of words in the text file, multiplying by 100.0 to make it look like a percentage. Do the same for the number-of-chars hot-score, dividing by the number of chars in the text file.

Other Rules: Your code for m hot-phrases and n text-words must run in time $o(mn)$ as demonstrated analytically by comments in your code (drawing on your Assignment 5 essay questions). You may assume that calls to the C++ standard library `sort` function, which requires including `<algorithm>`, on N items run in $O(N \log N)$ time. Modify your `Phrase` class as-needed and submit four files (or six with `StringWrap` too) as on Assignment 5, but with `HotPhrases.cpp` and `HotPhrases.make` for the files, and `hotphrases` for the named executable produced by the makefile (no `.exe` ending in UNIX). Now there must be an `operator<` function inside the `PhraseNNN.{h,cpp}` files but outside the `Phrase` class—it should do the same thing as your prior `lessThan`, and depending on how you code things, may-or-may-not need to be declared a `friend` by the class.

There is one particular “modularity” rule for the `Phrase` class. It must *not* maintain information about a particular hot-score, because that pertains to a particular text file that is not logically part of what the class is modeling. You are, however, welcome to derive a class `HotPhrase` from it to include this extra information. Alternatively, you can make `HotPhrase` a “wrapper” class for a phrase that adds this extra functionality—this is like what `StringWrap` does for `string`. The buzzword for the latter is “using composition not inheritance,” but it may involve more work because you may have to create some “delegating” versions of methods of the `Phrase` class from scratch, whereas with the derived-class approach you can just inherit them. Either way you must create the `HotPhrase` class (it can go in the `PhraseNNN.{h,cpp}` files) and describe in a report question how you handled it.

For the C++ library `sort` function there are issues that may differ between `timberlake` and your home compiler(s). These are illustrated by lines commented out in my answer-key code, where I’ve read the phrase file into a `vector<Phrase>` called `phrases`. First, below my `Phrase` class I have:

```
bool operator<(const Phrase& lhs, const Phrase& rhs); //== what you should do
bool lt(const Phrase& lhs, const Phrase& rhs);      //to fix syntactic hitch
```

Then in my client driver I have:

```
//sort(phrases.begin(), phrases.end(), Phrase::lessThan); //error
//Not allowed to pass a non-static member function.
//sort(phrases.begin(), phrases.end(), operator<); //error :-(
//sort(phrases.begin(), phrases.end(), less<Phrase>); //error :-( :-(
//sort(phrases.begin(), phrases.end(), lt); //OK, non-operator lt
sort(phrases.begin(), phrases.end()); //OK!---which IMHO is wrong
```

Similar considerations apply to your sorts on the hot-score criteria. Further particular instructions of a your-mileage-may-vary will be given, but this specifies the project. (90 pts. for code, 30 for having *a few good* INV/REQ/ENS comments and for the above-mentioned report question, for 120 points total).