CSE 303, Spring 2005, Assignment 6
Due: Thursday 26 May, 9:00AM

Last updated: May 25

Summary: You will work with two other group members to build an application that tests simple statistical models for the likelihood of short phrases in English text. You will use your solutions to the previous assignment — you will have to write new code, but you may not change the interface to the code you already have. You must use `make` and `cvs` to help manage your development (or at least to turn it in).

Behavior Specification:

1. Create an application `phrase_chance` that takes four command-line arguments. The first is a filename, the others are words containing only letters (upper- or lower-case) and hyphens.
2. If the input file cannot be opened, print an appropriate error message and exit.
3. The input file contains a sequence of words. A word is a sequence of letters and hyphens. Any other character is not a part of a word. Words are separated by one or more other characters. You may assume the last character in the file is a `'\n'`.
4. Adjacent words form phrases. For example, if the first word is "foo" and the second word is "bar", then this is the two-word phrase "foo" "bar" (regardless of what non-word characters are between "foo" and "bar").
5. Optionally, you may decide that a `'\n'` immediately after a `'-'` does not end a word. In this case the `'\n'` is not part of the word, but the word may continue on the next line.
6. Print to standard output text of the form:

   `1-word model:` n_1
   `2-word model:` n_2
   `3-word model:` n_3

Each number is an "estimation" of the probability that a three-word phrase drawn randomly from the input-file is the phrase described by command-line arguments 2–4:

- The first estimation is the "independent word model": The probability is $(w_1 \cdot w_2 \cdot w_3)/t^3$ where w_1 is the number of times the first word appears, w_2 the number of times the second word appears, w_3 the number of times the third word appears, and t is the total number of words.
- The second estimation is the "independent pair model": The probability is $(p_{12} \cdot p_{23})/(t \cdot w_2)$ where p_{12} is the number of times the two-word phrase `argv[2] argv[3]` appears, p_{23} is the number of times the two-word phrase `argv[3] argv[4]` appears, and the other variables are defined above.
- The third estimation is the "exact model": The probability is p_{123}/t where p_{123} is the number of times the three-word phrase `argv[2] argv[3] argv[4]` appears, and the other variables are defined above.

Note: We are assuming that t is large enough that the difference between t and $t-2$ is insignificant.

Implementation Specification:

1. You must use the files you turned in for assignment 5. You can fix bugs you discover, but do not change the interface (i.e., the types of your functions).
2. Do not "reveal" type definitions: only one file should "know" what fields `struct InputInfo` has, only another file should "know" what fields `struct Entry` has, and only a third file should "know" what fields `struct WordInfo` has.
3. *Do not concatenate strings together into longer strings.*

4. Use a Makefile. The first target of your Makefile must create the `phrase_chance` application (so you can just type `make`). The sources for your application should be only C files and header files.

5. Put all your source files and your Makefile in a CVS repository, in a module named `hw6`. No other files should be in the module.

Advice:

1. You will have to write new code (in new files) that "connects" the code you already have written. That is normal; do not shy away from it. In particular, you will have to write the functions that assignment 5B assumed existed.

2. Use 3 tries. For a trie that holds phrases, just enter phrases as one "word" that uses space characters to separate the "real words".

3. Use a pointer to this `struct` as a "function-pointer environment argument:"

```
struct WordEnv {
  int currWord;
  int currPos;
  int numWords;
  char ** words;
};
```

4. Some of the functionality in the code you already wrote is not necessary. That is normal; do not bemoan it.

5. For interesting test files already available in plain-text, see `http://www.gutenberg.org/catalog/`.

6. Do not try to run your application on files with more than a 10^5 distinct phrases or so unless you do Extra Credit 2 – or better yet, do try it, but see that you consume too much memory to complete quickly and then figure out why. An apparent infinite loop may actually just be a program using lots of memory.

Extra Credit 1: Turn in code for an application `phrase_chance_extra1`.

- This application is like `phrase_chance` except it takes any number of command-line arguments greater than or equal to 2.
- All arguments except the first one form a phrase. If the phrase is n words long, you should output the probability of seeing that phrase for every "word model" using pharses of length between 1 and n.
- Note you will need an unknown number of tries; use an array of them.
- Use a changed version of your "counter" code (assignment 5B), but still do not change the "trie" code or the "I/O" code.
- Running `make extra1` should create this executable.
- You should not have unnecessary copies of code files; your two applications should be built from some of the same sources.

Extra Credit 2: Turn in code for a third application `phrase_chance_extra2`.

- This application is like `phrase_chance` except it should work even for very large files with lots of distinct phrases.
- Do not add a phrase to a trie unless one of the words in the trie is one of the words on the command-line. (Note this optimization should not affect your results.)

- You may change the interface to functions, even those written for assignment 5, but do *not* use global variables.
- Running `make extra2` should create this executable.
- You should not have unnecessary copies of code files; your two applications should be built from some of the same sources.

Turn-in Instructions:

- Email Ben a message with subject `cse303: hw 6`. The body of the message should be nothing except an (absolute-path) directory d on `attu`.
- The directory d should hold a `cvs` repository. The permissions for the directory should allow access only to members of the operating-system "group" we have given you. The files in the repository should be readable by everyone.
- Ben should be able to execute `cvs -d` d `co hw6; cd hw6; make` and get an executable `phrase_chance` that works as described above.

Acknowledgment: Sarah Schwarm gave your instructor the idea to examine these natural-language models and helped him with the statistics. Thanks Sarah!