CSE 303, Winter 2006, Assignment 6
Due: Wednesday 1 March, 9:00AM

Last update: 15 February

Summary: With two other group members, you will build an application that takes a file f and two directories of files d_1 and d_2 and decides if f is "closer" to the files in d_1 or the files in d_2. This is a (sadly, very-ineffective) "spam-detector" if f is an email, d_1 holds examples of spam and d_2 holds examples of non-spam. You will use your solutions to the previous assignment — you will have to write new code (about 100 lines) but you should not change the (non-header) code you already have. You must use `make` and `cvs` to help manage your development (or at least to turn it in).

Requirements:

- Put all your source code and Makefile in a CVS repository you create.
- Write a Makefile that takes your source code and produces an executable `spam_detector` that takes 3 command-line arguments, two directory names and a file-name.
- If the directories or file cannot be accessed (e.g., they do not exist), print an appropriate error message and exit.
- You may assume the directories contain only files (no subdirectories).
- Compute the average "distance" of the file to the files in each directory (i.e., two separate averages). The distance is basically defined by the homework 5 code:
  - The "words" in a file are those returned by `next_multiletter_word`, i.e., ignore single-letter words, convert upper-case to lower-case, `'0'` to `'o'` and `'1'` to `'l'`.
  - The distance between two files is computed as described in Homework 5c, i.e., the square-root of the sum of the squares of the difference in word counts.
  - The average distance of a file to a set of files is the arithmetic mean of the distances to each file in the set.
- Print the distance to each directory and which directory is closer. An example execution might look like:

```
bash$./spam_detector listMail spamMail fromStudent
listMail: 77.2671, spamMail: 58.3604, closer: spamMail
```

- You may fix bugs in your homework-5 solutions, but you should not change the "interfaces" to the different solutions. For example:
  - Do not change the types of any functions you wrote.
  - Do not have your new code know the implementation of `struct WordCounter`.
  - Implement `how_many` and `does_longer_exist`; do not change function implementations to avoid calling these functions.
- You may need to change some header files to avoid files that define a `struct` twice or need to call a function that is not known to exist. This is fine, but large changes are not necessary.

Advice:

- This assignment gives less guidance on how to separate the task into helper functions and files, but style still matters.
- Use the library function `scandir` (see `man scandir`, particularly the example). Note this function returns entries for `.` and `..` which you should skip over. Avoid space leaks.
- It is probably easier and more space-efficient not to use `average_distance`. Rather, you can create a counter for one file, compute the distance to it, then deallocate the counter.
- The implementations of `how_many` and `does_longer_exist` are very easy. This sort of "adapter" code is very common when integrating separately developed units.
- The caller of `next_multiletter_word` cannot know how big a buffer it needs to pass, but it can pass a small buffer and use the return value to create a larger buffer. Avoid space leaks. It is more efficient *not* to create a new buffer for every word: use one buffer over and over again, making it larger only when necessary.

Extra Credit:

- Implement all three of these other ways to decide which directory is "closer" to the file f:
  - Report which directory has the single file that is closest to f.
  - Use the arithmetic mean *ignoring* the 10% of the files in each directory that are farthest away.
  - Use the geometric mean. The geometric mean of $a_i, \ldots, a_n$ is $(\prod_{i=1}^{n} a_i)^{1/n}$ but this could easily overflow a double. So instead, exploit that the mean is also $x^{(1/n)\sum_{i=1}^{n} \log_x a_i}$ for any x.
- Provide command-line options to use one of the above ways rather than the arithmetic mean. Your program should always report the result using exactly one method and it should use the arithmetic mean unless an option you choose is specified. Explain in a file called `README` how to use your application.

Assessment: Your solutions should be:

- Correct C code that compiles without warnings using `gcc -Wall` and does not have space leaks
- In good style, including indentation and line breaks
- A Makefile that rebuilds fairly precisely what is necessary
- Of reasonable size and efficiency

Turn-in (different than previous assignments!):

- Email Robert 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.
- Robert should be able to execute `cvs -d` d `co hw6; cd hw6; make` and get an executable `spam_detector` that works as described above.