

## Problem Set 4

Due before midnight on Tuesday, February 12, 2008.

### 1 Assignment Goals

1. Learn how to use command line arguments
  2. Learn how to open and read from a file.
  3. Learn how to report file system errors using `errno` and `strerror()`.
  4. Learn how to use `fgetc()` and `ungetc()` to peek at the next character of input without actually reading it, and to use the `scanf()` format code `%*s` to skip consecutive non-whitespace input characters.
  5. Learn how to write a multi-module program.
  6. Learn how to interface to a module written by someone else.
  7. Learn how to use dynamically-allocated storage.
  8. Learn how to use the provided flexible array package, `Flex`.
  9. Learn how to work with C strings and to use functions in `ctype.h` to process characters.
  10. Learn how to use control break logic to find and process matching groups of items in an input stream.
- 

### 2 Assignment

The editor of a newspaper wants to improve the writing of her reporters. She is tired of seeing overused words like “like” and long articles with limited vocabularaies and wants a tool to help her analyze writing style. To help her in her quest, you are to write a program that will read a text file containing a news article and produce an alphabetical list of words contained therein, where each word is preceded by a count of the number of times it occurs in the article.

To make the output more useful, the following conventions are to be observed:

1. Raw words are strings of non-whitespace characters separated by whitespace. Raw words longer than 30 characters are not expected; if any occur, they should be silently ignored.
2. Leading and trailing non-alphanumeric characters are to be discarded, but embedded non-alphanumeric characters are acceptable. Thus, the trailing period on “acceptable.” should be removed to yield “acceptable”, but the hyphen in “non-alphanumeric” should be preserved. Alphanumeric characters consist of the upper and lower case letters ‘a’, ..., ‘z’, and the digits ‘0’, ..., ‘9’, as defined by the `isalnum()` function. (See `ctype.h`.)

3. All words should be converted to lower case, so that “Leading” and “leading” are considered to be the same word.
4. Cleaned-up words of length less than 3 are not interesting and should be ignored.

## 3 Program Details

### 3.1 Method

Your program should read the file a raw word at a time. It should clean up each raw word and save each interesting word in a flex array (see below). After the file has been completely read, the flex array should be sorted alphabetically. This will bring like words together in the array. A pass over the array counts the number of occurrences of each word and prints out the count and the word.

### 3.2 Supplied modules

I have supplied two modules for your use. They are on the Zoo in the PS4 assignment directory `/c/cs223/assignments/ps4`.

The first, `util.c` and `util.h`, are slightly updated versions of the ones for PS3. In addition to `fatal()`, they contain functions `safe_malloc()` and `safe_realloc()`, which you will need.

The other, `flex.c` and `flex.h`, implement a flex array data structure. This is a container that stores a list of strings of arbitrary length. It allows you to add strings one at a time and “never” runs out of space (unless the computer runs out of memory).

You are required to use my code to store the list of words since part of the purpose of this assignment is to learn how to interface to other people’s code. You should copy the `.c` and `.h` files to your own directory and compile them there, but you should not modify them.

### 3.3 Coding style

Your own code should be in two modules that you create. The file `main.c` should contain only the main program `main()`; all other code goes in `wordfreq.c`. All that `main()` should do is to process its command line argument and open the input file. The processing of the contents of the file should be done by functions in `wordfreq.c`. You will need to write a header file `wordfreq.h` to be included in `main.c` that gives the prototype for the function that `main()` will call to process the file contents.

The function doing the processing should itself not be monolithic but should be split up into a number of smaller functions, each with a well-defined purpose. The processing naturally occurs in two phases: the first phase reads the file and builds the word list; the second phase sorts the word list and produce the output. Your code should reflect this two-phase structure by having separate functions for each phase. Note that the purpose of doing this is to make the code less complex and more manageable, not simply to avoid code duplication. In addition, smaller units of code with well-defined purposes should be split out as separate functions. For example, it is good to have a function that takes as its argument a raw word and returns a cleaned-up version of that word.

## 4 Testing

As usual, you should supply test data to verify that your program correctly implements the requirements spelled out in this assignment.

## 5 Submission

You should provide a `Makefile` that will build an executable file called `wordfreq` which expects one command line argument, the name of a text file to process. You should submit `Makefile`, `main.c`, `wordfreq.c`, and `wordfreq.h`. You do not need to submit copies of the source and header files for `util` and `flex` since we already have them, and your code will be built against our files, not yours. As always, you should also submit your test input files and the output produced by your program on those files.