Assignment 2

cpe 357 Fall 2010

```
        A program should be light and agile, its subroutines connected like a
strings of pearls.  The spirit and intent of the program should be retained
throughout.  There should be neither too little nor too much, neither needless
loops nor useless variables, neither lack of structure nor overwhelming
rigidity.
        A program should follow the 'Law of Least Astonishment'.  What is this
law?  It is simply that the program should always respond to the user in the
way that astonishes him least.
        A program, no matter how complex, should act as a single unit.  The
program should be directed by the logic within rather than by outward
appearances.
        If the program fails in these requirements, it will be in a state of
disorder and confusion.  The only way to correct this is to rewrite the
program.
                -- Geoffrey James, "The Tao of Programming"
```

— /usr/games/fortune

Due by 11:59:59pm, Friday, October 8th.
Due to the C quiz on Monday, no more than one late day may be used on this assignment. (Think of this as due Wednesday with two free late days.
This assignment may be done with a partner. (See "Notes on Partnership" below.)

Program: `fw`

Write a program that will return a sorted list of the *n* most common words in a file or set of files, sorted by order of frequency. In the case of a frequency tie, words later in the alphabet take precedence over words earlier. (That is, sort first on numbers, then on the words in lexographical order, but we're printing "bigger" things first so the words will wind up in reverse alphabetical order.)

While I am not going to specify any particular performance requirements, your program must be able to complete its task on a large input without causing undue emotional hardship.

Other things:

- So far as this program is concerned, a word is a string of alphabetic characters delimited by something that isn't. (That is, "cannot" is one word, never two.)

  See `isalpha(3)`, `isdigit(3)`, etc.
- Words should be output as a count followed by the word in all lowercase. (see below)
- You may not make any assumptions about the length of words, lines, or files.
- The command line of the program will consist of an optional argument of the form `-n <num>` to give the number of words, and an optional set of filenames. If the `-n` argument is not given, the program should default to 10, and if no filenames are given, the program should take its input from stdin.

  In case the argument to `-n` is not an integer, you should report the error, print a usage message, and stop.

- If, while processing files, you encounter a file that cannot be opened, print an error message explaining why, and continue with the rest.
- The output of this program will start with one line reading, "The top k words (out of n) are:" where k is the number requested and n is the number of unique words encountered in the input. (That is, a file consisting of the word "segfault" 5000 times has one unique word.)

  After that header line, the individual words are listed in descending order on individual lines as the count, right justified in a field nine characters wide, a space, then the word.

Tricks and Tools

- The program *does* require you to do both file IO and memory management. For help with these, you may want to look into some library functions:

  File IO Chapter 7 in K&R, as well as the manual pages for `fopen(3)`, `fclose(3)`, `fgetc(3)`, `fputc(3)`, (or `fprintf(3)` and `fscanf(3)`).

  Memory Chapters 5 & 6 in K&R, as well as the manual pages for `malloc(3)`, `free(3)`, `realloc(3)`, `calloc(3)`, etc.

- Be *very* careful with memory discipline. Be careful to `free()` everything you allocate when you're done with it, but only `free()` things you have allocated. Similarly, be sure only to `fclose()` files you have successfully `fopen()ed`. Solaris is fairly forgiving, but linux (where I grade these programs) will crash if you `free()` something you didn't allocate (or the same thing twice) or `fclose()` a `NULL` file pointer.

  Remember the first law of memory management:
  Free not that which thou hast not malloced, nor close that which thou hast not opened.

- Think about your data structures and overall program architecture up front. A poor choice can lead to dismal performance. I'd recommend either some sort of hashing scheme or a tree. This is an engineering decision left to you and your old data structures book. I do however reserve the right to apply the "undue emotional distress" criterion above. For example, a program that takes over a minute to process `/usr/share/dict/words` would not be acceptable.
- Be sure to test your program on pathalogical inputs. Consider the effect of `/usr/dict/words`, a file of 25 thousand discrete, alphabetized, words on a simple binary tree.
- Why reinvent the wheel? Look into the functions provided `ctype.h`, among them (in particular) `tolower(3)` and `isalpha(3)`.
- While converting strings to integers isn't all that hard, look into `atoi(3)` and `strtol(3)`.
- And, finally, if you don't know about it, look into `strcmp(3)`

Coding Standards and Make

See the pages on coding standards and make on the cpe 357 class web page.

Notes on Partnership

The goal of allowing you to work in partnerships on this assignment is to give you the opportunity to design, develop, and test with someone else while you are getting your balance with dynamic data structures in C. To that end, be sure to actually work together rather than simply dividing up tasks.

What to turn in

Submit via `handin` in the CSL to the `asgn2` directory of the `pn-cs357` account:

- your well-documented source files.
- A makefile (called `Makefile`) that will build your program.
- A README file that contains:
  - Your name(s). In addition to your names, please include your Cal Poly login names with it, in parentheses. E.g. `(pnico)`
  - Any special instructions for running your program.
  - Any other thing you want me to know while I am grading it.

  The README file should be **plain text,** i.e, **not a Word document**, and should be named "README", all capitals with no extension.

Sample runs

A working example of `fw` has been placed in `~pn-cs357/demos/fw`.

The first example below one took a minute or so, because `/var/man` on hornet is 95Mb of documentation...

Note, the expansion of wildcards is done by the shell. In this example `fw` saw an `argv` full of filenames. Note also that the error message changes as appropriate. (Remember `perror(3)`?)

```
% fw /usr/man/*/*
The top 10 words (out of 28390) are:
   193966 the
   191535 para
   156692 literal
   104872 refentrytitle
   104865 manvolnum
    97672 citerefentry
    90060 listitem
    88800 entry
    85343 term
    83944 varlistentry
% fw nonexistant
nonexistant: No such file or directory
The top 10 words (out of 0) are:
% fw -n fishsticks
usage: fw [-n num] [ file1 [ file 2 ...] ]
```

```
% wc main.c
     90     315    2189 main.c
% fw -n 1 main.c
The top 1 words (out of 101) are:
       13 infile
% fw nonexistant main.c
nonexistant: No such file or directory
The top 10 words (out of 101) are:
       13 infile
       13 fileidx
       12 the
       11 s
       11 argv
       10 words
       10 if
        7 n
        7 int
        7 argc
% chmod u-r main.c
% fw main.c
main.c: Permission denied
The top 10 words (out of 0) are:
%
```