Assignment 1

CS 360
Spring 2015

Summary: Write a program (in C) targeted at the Linux platform which reads words from one or more files, and prints out a list of the most frequently occurring *sequential pairs of words* and the number of times they occurred, in *decreasing* order of occurrence.

Example: If a file contained the text "*This is a nasty assignment and it will take me two weeks to do this nasty assignment. I should probably start on it early. I suspect it will take 300-500 lines of code*.", the word pairs "nasty assignment", "it will" and "will take" would each be detected as having 2 occurrences, while all the other sequential pairs have only one occurrence.

Program interface:

```
wordpairs <-count> fileName1 <fileName2> <fileName3> …

Where: count is the integer number of word pairs to print out
     and fileNameN are pathnames from which to read words.
     If no count argument is specified, ALL words are
     printed to stdout. (tokens enclosed in angular brackets
     are optional).
```

Specifications & restrictions:

1. All error output to be printed to `stderr`. If unrecoverable errors occur, the program exits with a non-zero exit code, otherwise it exits with a zero exit code.
2. All normal (expected) output (the list of word pairs and number of occurrences) are to be printed to `stdout`.
3. A procedure will be supplied by the instructor which reads successive words from an open file descriptor. Under the directory `/encs_share/class/cs360` you can find `getWord.h` and `libget.a` in subdirectories `include` and `lib`, respectively. *Do not copy these files, your makefile should utilize them in-place.* The source to `getWord` can be found in `/encs_share/class/cs360/src/getWord.c`, feel free to peruse it.
4. Use a hash table to store and count occurrences of sequences of words. Your hash table must keep track of how full it is and grow itself to a larger size (i.e. more buckets), as needed. Design and code your own data structures and procedures implementing a hash table for the purposes of this assignment. Use the technique known as *separate chaining* to implement your hash table buckets.
5. The hash table must evaluate some measure of its search performance. This can be average number of collisions, maximum collisions or some other reasonable measure which you will document in your comments. When this measure exceeds a threshold, your hash table will grow its number of buckets by at least a factor of 3. *Growth of the table will be transparent to the code using your hash table module.*
6. You will need a hashing function to hash the strings you insert and lookup in your hash table. You can research your own function, or use `/encs_share/class/cs360/src/crc64.c` If you develop your own function, document your hash algorithm in comments.
7. Use the standard library procedure "`qsort`()" for sorting, the unix man page should help with how to use it.
8. You are encouraged to use `assert()` where appropriate.

9. When your program outputs word pairs and their occurrence counts, output one word per line using the format `"%10d %s\n"`, where the decimal number is the number of occurrences and the string is the word pair (with one space between the words).
10. Use man pages to find the details of the C library routines you need.
11. Design your program to be robust, anticipating exceptional data or boundary conditions coming from the user or the data files.
12. Use good coding practices and make your code readable and understandable.
13. Break your code into at least two source files, one for the code implementing the hash table and at least one other for the implementation of the word pair counting. Design a good interface for your hash table implementation. The interface should not be specific to this application and must be re-entrant. The interface should be reflected in a header file included by both source programs and contains comments describing how each method (and its parameters) are used. The program using the hash table implementation must not need to comprehend the internal structure of the hash table, it should perform actions solely through the procedure interface declared in the hash table's header file.
14. Your program should manage the heap in such a way as to avoid memory leaks. If data structure components become unused, they must be freed before they become inaccessible. Consider this in the code associated with growing your hash table.
15. Do not surf the web for code or solutions to this assignment.
16. Think and design first, then implement. To the extent possible, implement and test components first, then integrate and test them together in a bottom up fashion. Start with simple tests and migrate to more strenuous test cases. You may test your program on the file `"/encs_share/class/cs360/lib/gettysburg".` You can find a large text file on which to test at `"/encs_share/class/cs360/lib/kjvbible.txt".` Diagramming your data structures and paper simulation of operations on your data structures are highly recommended as part of your design process.
17. Submit a tar ball (`.tar.gz` file, specifically) containing all of your source files and a **`Makefile`** *without absolute directory names or derived binary files*. Execution of make with no parameters should build the target program "`wordpairs`". Assume that the environment variable `GET_WORD` is defined as the pathname of a directory which contains directories "`include`" and "`lib`" containing `getWord.h` and `libget.a`, respectively. In your own build environment in the lab, you will want to define `GET_WORD` to be `"/encs_share/cs/class/cs360`".
18. While your program may build and execute correctly in other system environments, it must build and execute correctly in the ENCS laboratory's CentOS 6.5 environment.
19. Email your tar ball to the course's teaching assistant by the start of class on Monday, February 2th and include "CS 360" and "Assignment 1" in the subject line.