Spellchecker

Description

Write a spell checking program which uses a dictionary file to find misspelled words in a text document. Your program should prompt the user for the name of the input file and the name of the dictionary file, re-prompting if necessary (file does not exist). A valid dictionary file contains an alphabetized list of words, one word per line. A sample dictionary file is located in Canvas.

Functional requirements:

1. For each word in the input file, your program should search the dictionary for the given word. If the word is not in the dictionary, you program should print the message "Unknown word <word> found in line <line #> of the input file" to standard output. Hence you will need to read the text file one line at a time. You may find strtok and fgets useful:

   http://www.cplusplus.com/reference/cstring/strtok/ (http://www.cplusplus.com/reference/cstring/strtok/)
   http://www.cplusplus.com/reference/cstdio/fgets/

   (http://www.cplusplus.com/reference/cstdio/fgets/)
2. After traversing the entire input file, your program should print a message describing the total number of misspelled words in the file.
3. A dictionary file may contain words in uppercase, lowercase, or a mixture of both. You program should recognize words regardless of case. So if "dog" is in the dictionary file, the word "dOG" should be recognized as a valid word.
4. Within a line of the input file, a word is a white-space delimited string of characters. Any number of sequential non-alphabetic characters *at the end of a word* should be ignored. So "dog12" and "Dog!" are valid words (assuming the dictionary contains "dog") but "do-g" or "d.o.g." are not. You may find that the isalpha function from <ctype.h> makes this easier. Read more about isalpha here: http://www.cppreference.com/stdstring/isalpha.html (http://www.cppreference.com/stdstring/isalpha.html) .
5. Within a line of the dictionary file, only the first token on the line is recognized as a word. So a line " birch bark \n" would put the word "birch" in the dictionary, but bark or any subsequent text would be ignored.

Non-functional requirements:

1. You must implement a function char** parseDict(char* pathname) which opens the given file, counts how many dictionary entries are in the file, creates an array large enough to hold all the words, then reads the file a second time to load the words into the dictionary.
2. Implement and use a function int wordSearch(char* word, char** dict), which returns 1 if the word *word* is found in the array of strings, *dict*. The function returns 0 if the word is not found or if the dictionary array is not valid.
3. Implement and use a function int bad_word(char* word, int line, FILE* out) which prints the error message "Unknown word *word* found in line *line* of the input file" to the specified stream. The function returns true if the message is printed successfully, false if there is a problem (if, for example, out is a filestream not associated with a file opened for writing.) The header stdio.h defines stdin, stdout and stderr as filestreams for standard input, output and error.
4. You must put the functions above, and any other functions you write other than main, into spell_lib.c, while main goes in to spell.c. Both .c files should have associated .h files, and you should create a makefile that uses separate compilation to build spell.exe from your source code.
   Files (4): spell.c (contains main), spell.h (contains all of spell.c's #includes), spell_lib.c (contains all other function definitions), spell_lib.h (contains prototypes for functions in spell_lib.c, and all of spell_lib.c's #includes)

Rubric

	1	**2**	**3**
feature: word search works correctly	1	2	3
non-functional requirements	1	1	2
error reporting (line number)	1	2	2
efficiency/style	1	2	2
makefile	0	1	1

Submitting your program

You submit your program using the dropbox feature of Desire2Learn. The D2L dropbox will allow you to upload your program's source code file using a Web browser. For this assignment, your program's source code and make file must be saved in a directory called spell, and a zipped or tarred archive of this directory should be submitted to the Program 3 dropbox.