CS303 Programming Assignment #2: "Pointers and Functions"

Out:	Jan 30, 2013. **Due**: Feb 8, 2013 by 7:00 pm on the CS303 Moodle site.
Total points:	100. approximately 20% of the total homework grade.
Lectures:	Lecture 3 includes file I/O; Lectures 4-8 practice using pointers.
Textbook:	Page 111 summarizes elementary string-processing operations; Page 112 has an example of reading characters from standard input using getchar(); you may want to use fscanf().

You must write a C program which reads a file and performs a basic auto-capitalization of the words in the file. It must have the following features (70 points):

1. It is invoked from the command line using exactly two parameters. Note that the name of the executable counts as the first parameter. The second parameter is the name of a file to be used as input.
2. If an incorrect number of parameters is used, it should print an error message and return EXIT_FAILURE.
3. If the correct number of parameters is used, it should:
   a. Attempt to open the file. If the file is missing, the program should print an error message and return EXIT_FAILURE.
   b. Examine the opened file.
      (1) If the file has 0 words, it should print an error message and return EXIT_FAILURE
      (2) If the file has greater than 25 words, it should print an error message and return EXIT_FAILURE.
   c. Return to the start of the file. This may be done by closing and then reopening the file.
   d. Read each word in the file, ignoring any whitespace (blank spaces, newline, and tab characters).
   e. Store each word in an array of characters.
   f. Store each array of characters in an array of character pointers (see figure below).
   g. Print all the words using a supplied **`printWords()`** function.
   h. Print a carriage return.

   i. Examine each word.
      (1) If the word is the first word in the file, it should capitalize the first letter in the word. All other letters should be converted to lowercase.
      (2) If the word appears *after* a period, it should capitalize the first letter in the word. All other letters should be converted to lowercase.
      (3) All other words' letters should be converted to lowercase.

   (Implement all steps i-(1) through i-(3) in a single user-defined function).

   j. Print all the words using a supplied **`printWords()`** function.
   k. Close the file, free memory and return EXIT_SUCCESS.

*

Step (i) *must* be implemented in a user-defined function. The remaining steps may be implemented up to the student's discretion. The program must work correctly for files containing 25 words or less and for words consisting of 20 characters or less. The program must work correctly for files containing both letters and any assorted punctuation: e.g., !, $, &, ?.

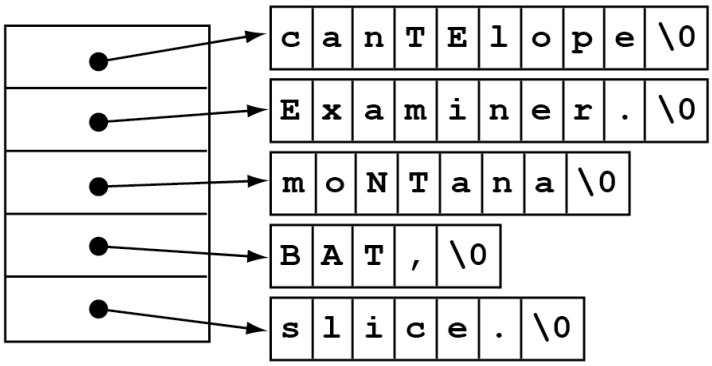
As stated in the steps above, error handling must be implemented for missing or empty input files. Error handling must be implemented if a file contains more than 25. The program does not need to handle words greater than 20 characters. The program does not need to handle non-printable ASCII characters.

Example

Suppose the input file, test1.txt, contains five words:

```
canTElope Examiner. moNTana BAT, slice.
```

After opening and parsing the file, the five words must be stored in a structure like so:



After counting and capitalizing letters, the five words must be printed like so:

```
Cantelope examiner. Montana bat, slice.
```

Example execution of test cases:

Test 1: Does the program execute using two command-line parameters? (2 points)	`$ ./a.out test1.txt` `...`

Test 2: Does the program print the words as they originally appear in the input file? (25 points)	`$ ./a.out test1.txt`
	`canTElope Examiner. moNTana BAT, slice.`
	`...`

Test 3: Does the program capitalize correctly? (21 points)	`$ ./a.out test1.txt`
	`canTElope Examiner. moNTana BAT, slice.`
	`Cantelope examiner. Montana bat, slice.`

Test 4: Does the program handle an incorrect number of command-line parameters? (2 points)	`$ ./a.out test1.txt test2.txt`
	`./a.out: Incorrect number of parameters.`
	`usage: ./a.out <input file>`

Test 4: Does the program handle an empty input file? (5 points)	`$ ./a.out test1.txt` `./a.out:  empty file`
Test 5: Does the program handle a missing input file? (5 points)	`$ ./a.out test.txt` `./a.out:  test.txt not found`

Test 6: Does the program handle an input file with greater than 25 words? (10 points)	`$ ./a.out test3.txt`
	`...`
	`./a.out: input file exceeds 25 words.`

To receive 100/100 points on this assignment, you must utilize good programming practice (30 points):

• Variables must have meaningful names and global variables must not be used.	2
• Preprocessor directives must be used for constant values.	2
• Code must be documented with useful comments and should use standard tabbing rules for good readability.	9
• Code should not be redundant. If two snippets of code have similar functionality, make a function or write a loop.	6
• A makefile must be used to compile the program and should be submitted with your homework submission. Note that a working makefile is provided in the starter code.	2
• All files opened by the program and all memory allocated to the program should be closed and freed before program exit.	5
• Only the main function and printX functions should use printf(); other functions should not. Instead, the main() function should print a message depending on the return value of a function.	4
Total	30

*

Students are given an opportunity for extra credit for implementing this additional feature (15 points)

The program must produce correct auto-capitalized output for words of arbitrary length.

*

To achieve maximum points on your submission, consider using this submission checklist before submitting your program to the Moodle course website:

- ☐ "I did not change the name of the source files in the starter code. The names of my program sources are main.c, donotedit.c and donotedit.h."
- ☐ **"**I did not change the makefile."
- ☐ "My program compiles successfully with the makefile provided in the starter code."
- ☐ "I ran all the tests (see above) to make sure my program executes correctly."
- ☐ "I followed the five pieces of guidance on commenting programs."
- ☐ "I compressed my source *files* into a zipfile named with my username, e.g., crenshaw13.zip"
- ☐ "I did **not** compress my source files using .rar, .z7, or some other proprietary compression program."
- ☐ "I did **not** compress a DIRECTORY of files."
- ☐ "I uploaded my zipfile to Moodle."