CIS*2500 - Intermediate Programming

Assignment 1: Dynamically Allocating Arrays

The question

Write a program that takes in a file of words (a regular text file with no punctuation) and prints out all the words from the file that start with 'A', all the words that start with 'B', etc.

How to write the program

For this assignment, the program has already been analyzed and broken down into the relevant functions (although you are free to create any other supporting functions that you please). Make sure that you create those functions exactly as specified below. In future assignments, you will be given more design latitude.

1. Copy the following function definition into your source code:

```
int file_size(FILE *fp)
{
   int sz = 0;
   fseek(fp, 0L, SEEK_END);
   sz = ftell(fp);
   rewind(fp);
   return sz;
}
```

   This function computes the file size (in bytes) of an opened file
   (see your CIS*1300 notes for opening and closing files).

2. Create a function `read_words( )` that takes in one argument, a File pointer, and then reads in words from that file into an array of words.
   - There will be at least one word per line. Other than that, the words in this file may be placed in any order (it will definitely not be ordered alphabetically) and can be any number of words.
   - For reading in a word from the file you may use a local character array of fixed size
     - You can assume that the longest word in English has 45 characters in it (a disease called Pneumonoultramicroscopicsilicovolcanoconiosis)
   - The maximum number of words that might be stored in the array of words can be computed from the file size (as opposed to assuming a maximum number as a constant).
     - As there can be no more words in a file then number of bytes in the file, use `file_size(fp)` to calculate the max number of words possibly in that file

        - note: this is an overestimate; for a single character to be a separate word you need a space or newline after the character … so the maximum word count is actually half the byte size of the file
    - The word array should also hold a NULL pointer after the entered words to indicate the end of the word list.
    - The array of words, once created, should be returned from the function.

3. Create a void function called `print_words( )` that takes a NULL terminated array of words and prints out the words, one word per line, stopping when it reaches the NULL pointer.
    - Make sure you handle the case of no words in the array.

4. Create a function called `alphabetical_word_count( )` that takes in the array of words and counts how many words begin with the letter 'a' or 'A', how many words begin with the letter 'b' or 'B' etc.
    - The results should be stored in a single array of int's and returned from the function.

5. Create a void function called `print_alphabetical_word_count( )` that prints out the array of integers and the letters that each count is associated with.
    - E.g.

      A = 23, B = 12, C = 0, D = 42, ... X = 0, Y = 0, Z = 1

6. Create a function called `create_alphabetical_array( )` that takes two parameters, a NULL terminated array of words, and the array of starting-letter word counts as returned by the `alphabetical_word_count( )` function. The function should:
    - Create 26 arrays, each one to stores all the words that start with a particular letter
        - For example, one of the array of words will contain all of the words that start with 'a' or 'A'.
        - There could be any number of words in a list. No maximums should be used.
        - The final element for each of these should be the NULL pointer (not the empty string or the \0 character).
        - If there are no words that start with the letter being considered, the array of words should have one element in it, which should be set to the NULL pointer.
    - The 26 arrays should be stored in an array called `alphabetical_array`, such that `alphabetical_array[0]` contains the array of all 'a' words, `alphabetical_array[1]` contains the list of all 'b' words etc.
        - This array should be the function's return value
        - You can use this array to store the 26 arrays while they are being created. No need to have separate variable names for each of the 26 word arrays.

7. Create a void function called `free_alphabetical_array( )` that takes in a single parameter of the same type and design as `alphabetical_array` and frees all of the memory in the array (remember to free the words first before freeing the array of words)

8. Create a void function called **printWordsAlphabetically** that prints out all of the words of alphabetical_array as follows

    'A'
    apple
    Anna
    apple

    'B'
    banana
    bounce
    Basket

    'C'
    car
    chase
    castle

    'D'
    There are no words that begin with the letter 'd'

    'E'
    elephant
    easy

    etc.

    hint: you should use the **`print_words( )`** function that you wrote for step 3

9. Create a main function (in a different file from the other functions so that we can use your functions with our own main()) that uses the above functions to
    - read in a file called **a1_words.txt** stored in the same directory as the program
    - print out the words in the file in the original order,
    - print out the number of words that start with 'A', 'B' etc.
    - print out all the words that start with 'A', 'B', etc.
    - read in a second file called **a1_moreWords.txt**
    - print out the words in the file in the original order,
    - print out the number of words that start with 'A', 'B' etc.
    - print out all the words that start with 'A', 'B', etc.

    make sure that all malloc'd memory is freed by the end of the program.