Project – Word Frequency Analysis

Summary

For this project, you will create a C program which calculates the frequencies of all words in a text file and reports the results sorted by frequency. Here is a sample of the required output:

```
SUMMARY:
        27340 words
        2572 unique words

WORD FREQUENCIES (TOP 10):
the                             1644
and                             872
to                              729
a                               632
it                              595
she                             553
i                               545
of                              514
said                            462
you                             411
```

This output was obtained by running the solution program on a text file (alice.txt) which contained Lewis Carroll's "Alice's Adventures in Wonderland".   The first few lines provide a summary of the numbers of words found (total words and unique words). This summary is important because it indicates whether your program is processing the words correctly. Your program must produce such a summary. Following the summary is a list of words and their frequencies sorted by decreasing frequency. We see the word "the" was the most common and occurred 1644 times, followed by "and" occurring 872 times, etc.

Programming

You must use a circularly doubly linked list as the core data structure.

You must use the code given in the form below. Any alterations will result in lost marks.

To extract individual words from the text file, use the following function:

```
int getNextWord(FILE *fp, char *buf, int bufsize) {

      char *p = buf;

      char c;

      //skip all non-word characters

      do {

            c = fgetc(fp);

            if (c == EOF)

                  return 0;

      } while (!isalpha(c));

      //read word chars

      do {

            if (p - buf < bufsize - 1)

                  *p++ = tolower(c);

            c = fgetc(fp);

      } while (isalpha(c));

      //finalize word

      *p = '\0';

      return 1;

}
```

The above function extracts the next word from the file pointer “fp” (which must be open) and stores it in the character array “buf” (which must be allocated). The “bufsize” parameter indicates the size of the character buffer. All words are automatically converted to lower case to make it easier to compare them for equality – you may want to import a C library to make it work. The function returns 1 (true) if a word is successfully read, and 0 (false) if an end-of-file is reached.

Therefore, all the words can be read with a loop like this:

```
while (getNextWord(fp, buf, size)) {

      // do something with buf

}
```

You must use the following structure to represent a word:

```
#define MAX_WORD 32

typedef struct word {

      char str[MAX_WORD];

      int freq;

} Word;
```

Your linked list implementation should contain data comprised of the Word type.

Command Line Arguments

Your program must accept at least one command line argument, which is the name of the file to be processed. An optional (you must program it, but the user can choose to enter it) integer argument can also be provided which states how many of the most frequent words to list. If that argument is not provided, it must default to 10. Naturally, provide instructions to your users so that they can figure out how to run your program if they attempt to do so incorrectly.

Executing with the command-line argument “alice.txt” should produce the list shown on the first page. On the other hand, running your program with the arguments “alice.txt 3” should list only the three most frequent words (“the”, “and”, and “to”). The summary word counts will not change – you must read and count the entire file every time regardless.

Documentation

For this project, you are required to produce a document which describes the details of your solution. The document should be between 500 and 1000 words. This document is worth a lot of marks (as much as the code!), so spend some time on it and impress me! The document should have the following structure:

Algorithm Description

- *Outline the algorithms used in a few sentences*
- *Describe the linked list implementation you chose and its benefits*
- *Describe the time complexity of the program and state it in terms of Big O Notation. If it helps, you can consider the number of words in the file as n.*

Implementation Status

- *Say "Complete" if it works, or "Incomplete" if it does not.*
- *If it doesn't work, describe what remains to be done and why you think it is not working.*
- *Whether it works or not, describe anything you think could be improved*
- *If your program doesn't work with large files like alice.txt, but does work with a smaller test file you created, state so here and please include the test file with your submission*

Collaborations

- *This is an individual project. It is not ok to share code. However, it is ok to share ideas. In this section, list the person or people (if any) whose ideas were incorporated into your code.*
- *Likewise, list any people to whom you communicated your own ideas.*
- *Note: no-one will be busted for this section! It's just so you can get used to working alongside other programmers and so you can give credit where credit is due.*

Please submit your document in either Word or PDF format.

Submission Instructions and Marking Scheme

Submit all source code and the documentation in a single zip archive. Submit the zip by uploading it to the appropriate SLATE assignment.

This project is worth 10% of your final grade. It will be marked out of 50 possible marks using the following mark breakdown:

- Document [20 marks]
    - Document is complete and properly structured
    - Document is insightful, honest, and correct, and reflects your understanding of the problem
    - Document looks professional and is in a presentable format
- Code [20 marks]
    - Code is well commented and properly indented
    - Good coding practices (no memory leaks, checking for NULL pointers, etc.)
    - Proper handling of command line arguments
    - Code actually works
- Submission Quality [10 marks]
    - Everything submitted in a single zip file
    - No executables or object files included with the submission – just code and documentation
    - Code compiles without errors or warnings