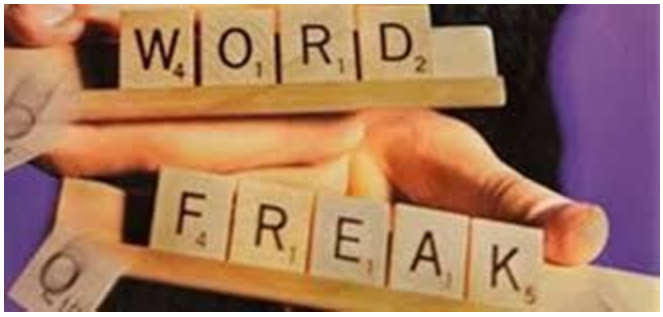

Word Freak

Project Profile

The goal of this project is to complete the implementation of a program that performs a simple glossary analysis. In particular, the program extracts “words” from a text file and prints to the console a mapping of the words found in the provided text file and their associated frequencies. You can download a bunch of example book text files to help with your implementation. For example, a working implementation will produce the following output (assuming your compiled program is called **wordfreak**) given a sequence of text files from the command line:

```
$ ./wordfreak aladdin.txt holmes.txt newton.txt
THE        :  1
ADVENTURES :  1
OF         :  1
ALADDIN    :  1
Once       :  1
upon       :  1
a          : 47
time       :  1
widow      :  5
had        : 21
an         :  6
only       :  4
son        :  6
...
```

A working implementation must also accept text files from standard input. For example, this is an alternate way a correct implementation can be executed (for more details on the bash pipe | operator see this tutorial):

```
$ cat aladdin.txt holmes.txt newton.txt | ./wordfreak
THE        :  1
ADVENTURES :  1
OF         :  1
ALADDIN    :  1
Once       :  1
upon       :  1
```

```
a          : 47
time       :  1
widow      :  5
had        : 21
an         :  6
only       :  4
son        :  6
...
```

A working implementation must also accept an environment variable called **WORD_FREAK** set to a single file from the command line to be analyzed:

```
$ WORD_FREAK=aladdin.txt ./wordfreak
THE        :  1
ADVENTURES :  1
OF         :  1
ALADDIN    :  1
Once       :  1
upon       :  1
a          : 47
time       :  1
widow      :  5
had        : 21
an         :  6
only       :  4
son        :  6
...
```

Furthermore, your submission is **restricted to only using** the following system calls: open(), close(), read(), write(), and lseek() for performing I/O. You are allowed to use other C library calls (e.g., malloc(), free()), however, all I/O is restricted to the Linux kernel's direct API support for I/O.

Video Demonstration

You must provide a link to a 2-minute video demonstration. Your video should be short and get to the point, showing your program being compiled and executed and demonstrating any requirements that are visible. You must use your voice to guide the watcher and demonstrate all required outputs.

Grading Breakdown

1. Project Requirements (40 points)
    a. Includes a Makefile that compiles your submitted code.
    b. Compiles a program that will accept 0 to n command line arguments. If your program receives 0 command line arguments it will read from standard input (using the bash pipe operator) for analysis. If your program receives 1 to n command line arguments it will open the provided files for analysis.
    c. Compiles a program that will use a specified WORD_FREAK environment variable to parse and analyze a text file and produce the proper output.
    d. Uses dynamic allocation and freeing (e.g., malloc(), free()).

    e. Uses structs and typedef.
    f. Uses pointers.
    g. Uses only system calls for I/O.
    h. Aligns output using only system calls for I/O.
    i. Provides error checking for all system and library calls.
2. Design and Implementation (30 points)
    a. Includes both C source files and header files. You can decide how best to organize your code and determine how many source and header files you require.
    b. Functions are declared and used properly.
    c. Data structures and data types are declared and used properly.
    d. Variable and function naming is clear and help with understanding the purpose of the program.
    e. Global variables are minimized, declared, and used properly.
    f. Control flow (e.g., if statements, looping constructs, function calls) are used properly.
    g. Algorithms are clearly constructed and are efficient. For example, there is no extra looping, unreachable code, confusing or misleading constructions, and missing or incomplete cases.
3. Coding Style (5 points)
    a. Code is written in a consistent style. For example, curly brace placement is the same across all if/then/else and looping constructors.
    b. Proper and consistent indenting is adhered to across the entire implementation.
    c. Proper spacing is used making the code understandable and readable.
4. Comments (5 points)
    a. Comments in the code are used to document algorithms.
    b. Variables are documented such that they aid the reader in understanding your code.
    c. Functions are documented to indicate the purpose of the parameters and return values.
5. Video (14 points)
    a. The video shows the code being compiled from the command line.
    b. The video shows the code being executed and satisfying the output requirements.
6. README.txt (5 points)
    a. The README.txt file is well written.
    b. The README.txt file provides an overview of your implementation.
    c. The README.txt file explains how your code satisfies each of the requirements.
7. GradeScope Submission (1 point)
    a. You automatically get a point for submitting to gradescope.

Submission

You must submit the following to Gradescope by the assigned due date:

- **`Makefile`** - the build file to use with the **`make`** program.
- **Source Files** - source and header files that provide the interface and implementation of the solution to this project.
- **`README.txt`** - this is a text file containing an overview/description of your submission highlighting the important parts of your implementation. You should also explain where in your implementation your code satisfies each of the requirements of the project or any requirements that you did not satisfy. The goal should make it easy and obvious for the person grading your submission to find the important rubric items. This text file should also clearly include a URL to your video for us to review. The video should not exceed 2 minutes in length and must provide a recorded voice to guide the viewer.

You must use the VM environment to write your code. Make sure you submit this project to the correct assignment on Gradescope!