MAT 4370: Programming Assignment 8

Due: Friday, March 1

The Problem

We would like to implement a "poor man's" version of the Unix `sort` utility. Working on this program will consolidate your understanding of the following ideas:

- Processing strings
- Processing command line arguments
- Reading streams of data: line-by-line and character-by-character
- Allocation of memory via `malloc()`

Our implementation is going to be relatively simple, in the following ways:

1. The synopsis for our sort is:

   ```
   ./pmsort filename
   ```

   There will be just one command-line argument, which names the file of data to sort. Your program should verify that the correct number of arguments has been provided and that the specified file exists. Appropriate error messages should be provided.

2. We will assume the file has at most 1000 lines, with one word per line. Each word is assumed to be at most 10 characters in length. However, your program must still check that these assumptions apply and do something reasonable if either limit is exceeded.

3. Unlike the Unix utility, your sort program will not have any command-line switches. Instead, we will just sort according to how `strcmp()` compares strings.

4. Unlike the Unix utility, your sort algorithm does not need to be sophisticated. Since selection sort is among the easiest sort algorithms, use that. The time you save thinking about sorting can be invested in system-related tasks: opening and closing streams, reading streams, setting up an `argv`-like structure, using `malloc()`, etc.

What to Do

- Read the Unix `man` page for `sort`. Try it out on a simple text file.
- Study the three sample stream programs discussed in class. Refer to the appropriate sections in the book to reinforce your understanding of these programs — file i/o has a number of important subtleties!
- Compile and execute each of the three sample programs. Experiment with each to see if they behave the way you think they should.
- Re-implement the `readLine()` function so that it no longer makes use of `fgets()`. Instead, process the input on a given line character-by-character.
- Provide code which will create a data structure similar to `argv` to hold all of the words to be sorted. Use `malloc()` to ensure that each entry has just the required number of bytes needed to store the words. The final entry in your array should be the `NULL` pointer.

- Implement a `sort()` function which will rearrange the words in sorted order. To swap two words in your array, note that only a pair of pointers need to move. The strings themselves, once established, will never move.
- Put all the pieces together to form your sort utility. Carefully test your program to ensure it behaves correctly.

What to Submit

When your sort implementation works to your satisfaction, place it in a folder named `hw08`. (Source code only please; no other files need to be submitted.) Drag this folder onto the SUBMIT icon, found in the Applications folder.