Design Plan - Wordsorter

James Viner

Project Summary

Wordsorter (ws) is a command line program design to take in a number of words delimited by white space and sort them according to given command line options.

Features Targeted

Case Insensitive Sorting

Allow the program to accept a '-i' option that sets the desired sorting algorithm to case-insensitive sorting. This should be easily accomplished through an option passed to the '-a' algorithm for lexicographical sorting that turns on an extra function to convert each compared character in both strings to lowercase before comparisons are done.

Filtering Last n Results

Allow a '-C' option that filters the results of the sorting algorithm to printing only the last n results from the list. Once the sorted lists have been assembled, this is a matter of pruning the results that print to the screen. One slight difficulty here is determining the order in which the pruning is done when combined with the '-c' option, but having a variable that tracks which one goes first seems like the best solution.

Scrabble Sorting W/ Validation

Allow a '-S' option that enables Scrabble sorting with validation based on what words are possible to form with the English Scrabble set of tiles. The best way forward here seems to be implementing a list of the number of tiles for each letter in the game and having the a function that runs before the scrabble sort count the instances of each letter, comparing after it's iterated through the entire word to see if it has exceeded its allotment of tiles for a given letter as well as all available blank tiles.

Architecture

Data

The maximum size for each given word to be sorted is unknown before program execution, as is the number of words to be sorted. It would be possible to get a line out of the file on some white space delimiter, then tokenize the string on the rest of the white space characters, allocating space for each string equal to its length and appending pointers to the start of each string to an array that gets dynamically resized when needed. The result should be that at the end of reading from each file there will be an array of pointers to the start of each individual word that can then be passed to sorting algorithms for processing.

Significant Functions

```
char **load_words(FILE **input_files)
```

Loads the contents of all given input files into another array, ignoring duplicate white space, but not otherwise doing any validation. Returns an array of pointers to strings that contains all processable words, split on white space.

```
char **prune_scrabble_words(char **words)
```

Function specifically made to accompany the scrabble sort that iterates through each word in the passed array of strings and uses a parallel array of booleans to mark strings for pruning. Strings that meet the requirements to be a Scrabble legal string according to the number of tiles given for each letter (and all blank tiles) will be marked, the rest will be pruned. The return value is a pointer to an array of strings that contains all Scrabble legal words.

Algorithms for qsort()

```
int num_sort(const char *str_1, const char *str_2)
```

Compares two strings on their numerical magnitude. The expected input for this function is two strings to be interpreted as numerical data, where non-digit

characters are ignored.

```
int len_sort(const char *str_1, const char *str_2)
```

Compares two strings based on which one has more characters.

```
int scrabble_sort(const char *str_1, const char *str_2)
```

Compares two strings based on their score in scrabble.

```
int ascii_sort(const char *str_1, const char *str_2)
```

Compares two strings lexicographically by ASCII code point.

User Interface

If given no files as input, similar to the behavior of the wc command, the program will drop into an interactive environment where the user can type any words they want to be sorted. Once an EOF has been received, the program will process all input and return the desired values, sorted.

General Approach

The general approach should be to begin by creating the framework for the program. Blocking out the ability to set flags for placeholder options that will later on determine which sorting algorithm to use seems like a good start. Then, building out the function that loads words so that data can be read in from multiple files makes the most sense, as it will be used later to ensure the functionality of the sorting functions. Finally, designing the algorithms that will sort the words and then calling them based on which flags were set using the previously established framework would bring the entire project together. After that, all that is left to do is ensure that the array of strings is iterated through and printed to the console appropriately.