Richard Farrington netid: raf172
Anirudh Tunoori netid: at813

CS 214: Systems Programming Spring 2018
Professor Francisco
Assignment 0: String Sorting
~ Readme

Our program takes a single input string of unknown length and sorts the words found within it (if any exist). It achieves this objective through the execution of seven functions (including main).

The program begins by assessing that the user entered the correct number of arguments and printing a usage string if not. The usage sting simply provides the proper way to run the program. If the input string is empty, the program simply terminates without doing anything (as specified by the project specifications). It then checks to see if there is at least one character in the input string that is a letter, as otherwise the program need not do anything. This check is done in a loop and calls the isLetter function on each character. The isLetter function is used a few times during the duration of each "run": it takes a character and assesses if it is a letter by comparing the ascii value of the character and seeing if it falls in the appropriate range in order to be a letter. Our program makes use of a global struct as the primary data structure. The struct which we defined using typedef consists of a string array (char**) and an integer that specifies the number of words in our list (which is essentially what our struct is: a list of words). This struct is defined externally from the other functions, but the space for it is dynamically allocated (initiated) in the main function.

Our program then proceeds with the parsing process which is done through the use of a single loop. It took some experimentation to conduct the entire parsing process as efficiently as possible without involving implementation complexities. Notably our loop makes use of three integers a counter which is used to keep track of the current position in the input string and ensuring that we haven't left the scope/bounds of the string, a start and an end integer which is set with each iteration in order to find the next word in the input string. Every time we encountered an entire new word: the position after end contains a separator/non-letter character, we called our fill function. The fill function takes the start and end indexes along with the input string and essentially "creates/fills in" a new word in our list (struct). The fill function dynamically allocates enough space for a new word (string/char*) and makes use of strncpy from the string library to copy the desired word. This new word is then added into our existing list. The fill function dynamically allocates memory as needed so all words provided in the input can be of indeterminate length for an undetermined number of strings/words. Once the fill/parse process is completed in its entirety, a new list is dynamically allocated, and the old list is copied into it using strncpy. This new list is then sorted using a sorting procedure.

*A notable feature of our program is the use of a quicksort() algorithm to sort the individual words/strings. We favored the use of quicksort because it runs in O(nlog(n)) in the best case and the average case and although it runs in O(n^2) in the worst case, the space complexity is O(log(n)) in the worst case as the sorting is done in place. Its implementation is slightly trickier as we are trying to sort a "String" array (char**) than say an integer array another sorting algorithm such as insertion sort or selection sort.*

The sorting process is accomplished through the use of three functions. The swap function swaps the word found in two indices by utilizing a temp string (char*). The schism function accomplishes the partition procedure in the quicksort algorithm, it will iterate through

the list by calling strcmp on each word with respect to the designated pivot word. We used strcmp because it is effective in lexicographically comparing strings. The schism calls swap as needed (depending on the comparison). Finally, the quicksort function directs the overall process by finding the pivot (location) using schism and making recursive calls on everything less than the pivot followed by everything greater than it.

Once all the words in the list have been sorted, they are printed through the use of a loop by making use of %s in the printf calls. Finally, everything that was dynamically allocated was freed. We successfully tested our program for memory leaks by using valgrind.