Assignment 0

Karan Amin
Saavi Dhingra

This program reads in data from a file, sorts it, and outputs a sorted result based on two sorting functions: insertion sort and quicksort. It reads in all the tokens and determines whether the input consists of strings or numeric data while also ignoring new lines, spaces, tabs, etc.. Different levels of errors are taken into account based on unexpected situations that may or may not be easily resolved.

Files included:

fileSort.c
testplan.txt
readme.pdf

Compiling the program:

The user is allowed to select which sorting algorithm to use during run time. The first parameter of the code is set to a flag, which allows the user to select which sorting algorithm implementation (function) the code should apply. Use “-i” for insertion sort and “-q” for quick sort. The second command line parameter consists of a single file name, which the program opens and reads in all the tokens from. This is the file that represents the strings or numeric data, and in order to compile the program, the user must input in the following format.

→ ./fileSort -q test2.txt
→ ./fileSort -i test2.txt

How to use the program:

The program recognizes all ASCII character files, specifically numeric and string data. It takes into account all possible user input errors along with an error for empty files. It has implemented an error check that targets incorrect file path names inputted by the user. Depending on the three general levels of error, different error messages are outputted to the user. It is important to properly format the input.

./fileSort -q test7.txt ✔	./fileSort test7.txt ✗	./fileSort test7.txt -i ✗

How the code operates:

The program goes through the file and reads in the characters or integers. It separates the characters it has read up to a token that is not a character nor integer, as the it serves as a separation of the data content. The program also ignores all spaces, new lines, and tabs. Each of the values of the sorted contents are based on insertion sort or quicksort, which are outputted in their own line. The sorted data is resulted in increasing order with the help of the comparator function, which compares and typecasts the arguments in respect to the information presented.

Examples:

test7.txt: 9 0 8 2 4 , 3 , 9, 4 , 9 , , ,

./fileSort -q test7.txt	./fileSort -i test7.txt
Before Sorting: |90824| |3| |9| |4| |9| Using QuickSort After Sorting: |3| |4| |9| |9| |90824| End of File	Before Sorting: |90824| |3| |9| |4| |9| Using insertion Sort After Sorting: |3| |4| |9| |9| |90824| End of File

test1.txt: hi,there, every, one

Input: ./fileSort -q test1.txt	**Input:** ./fileSort -i test1.txt
Output: Before Sorting: |hi| |there| |every| |one| Using QuickSort After Sorting: |every| |hi| |one| |there| End of File	**Output:** Before Sorting: |hi| |there| |every| |one| Using insertion Sort After Sorting: |every| |hi| |one| |there| End of File