Overview:

The program expects there to be 3 command line arguments, one being the file path, followed by sort type (either -i for insertion sort and -q for quicksort). In the event that the number of arguments is insufficient or the arguments are incorrect in some way this will output an error message with the type of error. If the command line is formatted correctly then the program will access the file-to-be-sorted. The program is only capable of handling EITHER integer data types OR character data types not both. Both integer and character files with empty values such as “,,,” will be interpreted accordingly so long as the type of data is disambiguated by the end of reading in. The parser reads in one byte at a time as to easily decide whether or not the byte is storable information such as integers or characters OR useless information such as spaces, tabs, and new lines. Hitting a comma will result in termination of the current parsing procedure and induce the creation of a new node which will be attached in the linked list. If a comma is detected prior to EOF the comma will terminate whatever came before it but will not induce the creation of an empty string or 0 (so long as prior to the comma there was actual storable data). Once the parsing procedure is complete the sort corresponding to the sort type input will decide if the list is sorted by the insertion sort algorithm or the quick sort algorithm. In order to operate quickly, the information in the nodes are loaded into an array and sorted using the array (for constant time access) to then be copied back into their corresponding linked lists. Upon return from the sort type the nodes are passed to a print function, printed, then passed to a free function. Once all heap allocated data has been freed, the main returns 0.

Function descriptions:

void qsInts(int * number,int start,int end, int (*comparator)(void*, void*));

- Is only invoked upon being called by the “quickSort” function. Accepts a numerical array as input, a start index, and an end index, as well as a function pointer “comparator”. Starting with the pivot being the first item, the data in the number array is then partitioned and recursively sorted using the classical quickSort algorithm. Once the sort is complete, the array is sorted and can be used in the parent “quickSort” function.

void qsStrings(char **strings,int start,int end, int (*comparator)(void*, void*));

- Is called by the “quickSort” function. Accepts a 2D array of characters, a start index, an end index, and the “comparator” function pointer. Starting with the pivot being the first item, the strings in the 2D character array are then partitioned and recursively sorted using the quicksort algorithm. Once the sorting is complete, the array can be used by the “quickSort” function.

int insertionSort(void* toSort, int (*comparator)(void*, void*));

- Accepts a void pointer and function pointer as its parameters. The void pointer is then casted as the proper structure type. If the file contains integers, an array of integers is created and the integer values from the linked list are copied over to the array. Then the insertion sort algorithm sorts the array using the comparator function to compare strings for lexicographical order. The sorted array values are then copied back into the linked list in the correct order (overriding whatever was there previously). If the file contains strings, a 2D array of characters is created. The strings from the linked list are then copied over to the array which is then sorted by the insertion sort algorithm, once again using the comparator function to compare two integers. The sorted array of strings is then copied back into the linked list in the now sorted order. Upon success, it returns 0.

int quickSort(void* toSort, int (*comparator)(void*, void*));

- Accepts a void pointer and function pointer as its parameters. The void * toSort is casted as the proper structure type. If the file contains integers, an array of integers is created and the integer values from the linked list are copied over to the array. Then the quick sort algorithm sorts the array by calling the "qsInts" function. Then the sorted array values are copied back into the linked list now in the correct order. If the file contains strings, a 2D array of characters is created. Then the strings from the linked list are copied over to the array which is then sorted by the quicksort algorithm, using the "qsStrings" function. The sorted array of strings is then copied back into the linked list (overriding information previously stored there) in the now sorted order. Upon success, it returns 0.

int comparator(void * a, void * b);

- Accepts two void pointers as its parameters. If the file contains integers, two integer pointers are created and set equal to a and b to point to their memory addresses. Then the dereferenced values are compared to each other and the appropriate value is returned. If the file contained strings, then the function "simpleStrCmp" is called with a and b casted as char pointers as its parameters.

int simpleStrCmp(char * a, char * b);

- Accepts two character arrays as its parameters that represent strings. If string 'a' comes later alphabetically than string 'b', a positive value is returned. If string 'b' comes later alphabetically than string 'a', a negative value is returned. If the strings are the same, 0 is returned.

void printFinal(struct nodeData * list, char t);

- Accepts the head of the linked list and the character that holds the type of file that is being read. It first checks to make sure the list is not empty and then initiates a do-while loop that prints out the linked list of integers or strings depending on what the original text file contained.

void freeList(struct nodeData * list);

- Initiates a while loop that frees the linked list