Liam King
Asst0: String Sorting

Program Description:

Design/Data Structue:
My string sorting program utilizes a linked list to store and sort the input strings. To implement this data structure, I created a structure called "node" that contains a char pointer that keeps track of the node's string and a node pointer that keeps track of the next node in the linked list. I also implemented a "create_node(char* arr)" function that takes a string as a parameter and initialized a new node with the given string and NULL as its field's values.

Runtime/Execution Walkthrough:
Upon being executed, the program first checks to make sure it has been given the correct number of parameters. If it receives anything other than one parameter, an error message is output and the program terminates. Once the program determines it's been given the correct number of arguments, it then runs a loops to determine the length of the input string. This value is stored in the variable "length". Next, a for loop that iterates length-number of times begins parsing the input string one character at a time. Inside the for loop, a while loop iterates as long as the current character is an alphabet character, incrementing the variable 'int j" by one after each iteration to keep track of the length of the current string being discovered. Once a non-alphabetic character is reached, the while loop terminates. Then an if statement checks if "int j" is greater than zero. If it is equal to zero, it means there is no string found at that time and the for loop iterates to the next character. If "j" is found to be greater than zero, a viable string has been found. In this case, memory is requested for (j+1) characters (plus one to leave room for the null terminator character). Then a for loop iterates "j" times and copies the found string one character at a time into the "char* word" and finally the null terminator into the last index. The found word is then given to an "insert" function that creates a new node with the found word and then puts that node into the linked list, comparing the current string to each string in the linked list to ensure the list stays sorted as each node is inserted. Finally, an if statement checks to make sure the head of the linked list isn't NULL (at least one string has been found) and then calls a "print" function that iterates through the linked list, printing out each nodes string and freeing that node on each iteration. And since my "insert" function sorts as it inserts, printing out the linked list in order will always result in a sorted list of strings. Finally, the "char* word" pointer is freed and the function terminates.