This is Malay and Nick's stringsorter program. How it works is quite simple:

1. Check the number of inputs

Stringsorter makes two checks before continuing. First, it checks if there are zero inputs, in which it returns like the PDF says to. Then it checks if there was more than one input string, upon which it prints a usage string and returns. Both these checks test the value of argc.

2. Run through, tokenize, and add individual strings

Seeing that there is only one input string, stringsorter will now scan through each char, and while the char is not the null terminator, it changes the values of some variables. Ints "start" and "end" store the first and last indices of a potential string. So, if the scanned char is alphabetic, end increases by one. Otherwise, we have reached a delimiter, and we must add the string between start and end into strArr, a malloc'ed array that will store all strings.

We do so with the addString() function. It checks if start != end, meaning the string is not empty, and adds it to strArr. It copies the string from argv[1] into a char array, and puts the address of the array in strArr. This will make things easier when we sort strArr. If strArr is not big enough, we use realloc() to expand its size. Regardless of whether we added a string or not, both start and end are updated by addString(), as we passed the addresses of these variables to allow changing them. It also updates the contents of strArr by returning the address of the strArr in the function.

After this happens repeatedly with the while loop, it seems that we have reached the end of the input string and have no more strings to add, but this is NOT the case. We call addString() one more time after the loop in the case that there was no delimiter before the last token string in the input line (ex. test in "test" has no delimiter, there in "hello9there" as well, etc.).

3. Sorting, printing, and freeing

Now that strArr has all of the token strings, we will sort it with bubblesort. We realize this is not the best sorting algorithm, but it is easy to code. Because strArr essentially stores pointer addresses to char arrays, we simply swap memory addresses when we need to.

With strArr sorted, a simple for-loop prints each string.

Lastly, we free all of the strings in strArr array, and strArr itself. We did check if malloc() and realloc() fail in every instance they were called, making the program return in case they failed.

And that's it!