Homework 8 – CSC 230-001, Spring, 2007
Issued 4/11/07, due 4/19/07 4/20/07

Note: Same comments as made in Homework 1 apply.

Assumptions should be stated as comments in the body of the code.

1. Write a complete program that sorts strings, and prints them out in alphabetical order. Each string may be assumed to contain only lower and uppercase letters, and the sorting should be case-insensitive, but the strings should be printed out in original case, and if strings contain non-alphabetical characters, the program should fail gracefully, rather than crash.

Strings may also contain blanks or other punctuation. Blanks must be taken into account in sorting: a blank comes alphabetically before any letter of the alphabet. Punctuation can be ignored for the purpose of sorting, but must not abort the program.

Update: Strings may be expected to contain alphabetic characters, blanks, and commonly accepted English punctuation. (Do not confuse the term "punctuation" here with the C language source syntax. In terms of C, everything that we are talking about is part of a string, therefore a literal. No question of punctuation, identifiers, etc.) Accepted punctuation are periods, colon, semi-colon, commas, exclamation, query, quotation marks.

Any other character is unexpected, and if any string contains any other characters, it is okay to abort the sorting (but fail gracefully rather than crash).

The program should accept a command line argument -i (for “input”). If present, it indicates that the very next argument (which must be present) is the name of a file that contains the strings to be sorted.

If this option is not present, then the strings to be sorted must be read in from the keyboard, that is the user interactively types in the strings (separated by newlines, as before). After the last string, the user types in the \(\text{EOF}\) character (which for most UNIX systems is Ctrl-D).

The program should also support a -r option (for “reverse”) that indicates that the final sorted order should be reverse alphabetical. This option may be exercised for either the input file or the interactive options. Assume that the options may be supplied on the command line in any order, i.e. a.out -r -i input.txt and a.out -i input.txt -r are equally valid ways to run the program, and should result in exactly the same behavior.

The program should also support a -u option (for “unique”) that indicates that if two or more of the input strings are exactly the same (including non-sorting characters such as punctuation), then only one should be printed out. Again, allow any ordering of command line options.
Specifically state any assumptions you make about the length of the string, or any other assumption. Include as much error checking as possible so that violations of such assumptions are treated gracefully (e.g. program should print out error instead of crashing).

String library functions may be used if useful, but the program must behave according to the above specifications.