COP4338

Fall 2016

Caryl Rahn

## Assignment #1: Sorting with Binary Search Tree (due by 11:55 PM on 10/19).

Through this programming assignment, the students will learn to do the following:

1. Know how to process command line arguments.
2. Perform basic file I/O.
3. Use structs, pointers, and strings.
4. Use dynamic memory.

This assignment asks you to sort the lines of an input file (or from standard input) and print the sorted lines to an output file (or standard output). Your program, called **bstsort** (binary search tree sort), will take the following command line arguments:

% bstsort [-c] [-o output\_file\_name] [input\_file\_name]

If -c is present, the program needs to compare the strings case sensitive; otherwise, it's case insensitive. If the output\_file\_name is given with the -o option, the program will output the sorted lines to the given output file; otherwise, the output shall be the standard output. Similarly, if the input\_file\_name is given, the program will read from the input file; otherwise, the input will be from the standard input. You must use getopt() to parse the command line arguments to determine the cases. All strings will be no more than 100 characters long.

In addition to parsing and processing the command line arguments, your program needs to do the following:

1. You need to construct a binary search tree as you read from input. A binary search tree is a binary tree. Each node can have at most two child nodes (one on the left and one on the right), both or either one can be empty. If a child node exists, it's the root of a binary search tree (we call subtree). Each node contains a key (in our case, it's a string). If the left subtree of a node exists, it contains only nodes with keys less than the node's key. If the right subtree of a node exists, it contains only nodes with keys greater than the node's key. You can look up binary search tree on the web or in your Data Structure textbook. Note that you do not need to balance the binary search tree (that is, you can ignore all those rotation operations) in this assignment.
2. Initially the tree is empty (that is, the root is null). The program reads from the input file (or stdin) one line at a time; If the line is not an empty line, it should create a tree node that stores (a copy of) the string (you shall remove the trailing line feed) and then insert the tree node to the binary search tree. An empty line would indicate the end of input for stdin, an empty line or end of file would indicate the end of input for an input file.
3. You must develop two string comparison functions, one for case sensitive and the other for case insensitive. You must not use strcmp() and strcasecmp() functions provided by the C library. You must implement your own version.
4. Once the program has read all the input (when EOF is returned), the program then performs an in-order traversal of the binary search tree to print out all the strings one line at a time to the output file or stdout. If there are duplicates than include all duplicates.
5. Before the program ends, it must reclaim the tree! You can do this by performing a post-order traversal, i.e., reclaiming the children nodes before reclaiming the node itself. Make sure you also reclaim the memory occupied by the string as well.
6. It is required that you use getopt for processing the command line and use malloc/free functions for dynamically allocating and deallocating nodes and the buffers for the strings.  It is required that you implement your own string comparison functions instead of using the corresponding libc functions.

Here's an example:

bash$ cat myfile

bob is working.

david is a new hire.

alice is bob's boss.

charles doesn't like bob.

bash$ ./bstsort myfile

alice is bob's boss.

bob is working.

charles doesn't like bob.

david is a new hire.

One can also use './bstsort -o sortedfile < myfile' to take input from stdin (in this case, it's replaced by a file by the shell) and output the sorted lines in the output file (called sortedfile in the example).

Please submit your work through moodle as one zip file. Follow the instructions below carefully (to avoid unnecessary loss of grade):

To start, first create a directory for this homework and name it FirstnameLastnameA1 (of course, you'd use your real name here).  You should place the source code and the Makefile in the directory. One should be able to create the executable by simply 'make'. The Makefile should also contain a 'clean' target for cleaning up the directory (removing all temporary files, object files and executable files). Make sure you don't include intermediate files: \*.o, executables, \*~, etc., in your submission. (There'll be a penalty for including unnecessary intermediate files). Only two files should be included unless permission is given for more, those would be bstsort.c and Makefile.

Please make sure you submit homework before the deadline.

Due 10/19/16 – extension given until 10/26/16