Data Structures and Algorithms I
Spring 2017
Programming Assignment #2

You are going to write a program that sorts the nodes of a linked list. Each program will load data from a file and create a linked list of pointers to data objects using the provided C++ list class. Each data object will consist of a single C++ string, and each string will represent a floating-point value. After creating the list, the program will sort the list according to the values represented by the strings. The input and output files will have the same format. The first row will be in integer indicating how many rows follow. Each row after that represents a single data object and contains a string of ASCII characters followed by a Unix-style newline character ('\n'). Other than the newline characters, all the characters will be digits or decimal points. Each string will contain exactly one decimal point with at least one digit to the left of the decimal point and at least one digit to the right of the decimal point. The nodes in the linked list are to be sorted according to the values represented by the strings. Note that if two strings are identical, then they represent the same value, and they are indistinguishable; in this case, it does not matter which one is listed first in the output (i.e., you do not have to create a stable sort).

I am providing you with code that handles most aspects of the program, and *you may not make any changes to the provided code or its behavior* (this will be explained further in class). The provided code includes the implementation of a simple class to store the data objects, the file loading routine that loads the data from an input file, and the file saving routine that writes the sorted data to an output file. There is also a call to a sort routine that you must fill in. You may also add additional functions, additional class definitions, or additional global variables if you wish, but all the added code must be included below a specific comment which indicates that the code above the comment may not change. (You should even be able to include additional provided header files below the comment if you wish. If you want to do this but your compiler does not support it, then include them at the top of the file, and mention this in your e-mail when you submit the program.) I will use the "diff" command to make sure you have not changed any code above the comment.

Your program will be tested on four test cases, which we will call $T_1 \ldots T_4$. For each of these test cases, a single file will exist using the format specified earlier (the first row will indicate the number of rows to follow, and each additional row contains a single string). The contents of the four test cases will also adhere to the following specifications:

- T_1 will contain approximately (within 1 percent of) 100,000 data objects. Each data object will store a randomly generated string representing a floating-point value. For each string, 20 randomly digits will be generated to the left of the decimal point, and 20 random digits will be generated to the right of the decimal point. After that, leading zeros and trailing zeros, if any, will be stripped (unless all digits left or right of the decimal point are 0, in which case a single 0 will be displayed). A sample file with this format will be provided on the course home page; the provided file will not be the same T_1 that will eventually be used to test your programs, but it will be generated the same way.
- T_2 will contain approximately 1 million data objects. Each data object will store a randomly generated string representing a floating-point value, and each string will be generated the

same way as for T_1. The only difference between the processes for creating T_1 and T_2 is that more strings are generated for T_2.

- T_3 will contain approximately 1 million data objects. Each data object will store a randomly generated short string (relative to T_1 and T_2) representing a floating-point value with a more limited range. For each string, three random digits will be generated left and right of the decimal point. As with T_1 and T_2, leading and trailing zeros will be stripped, but if all digits left or right of the decimal point are 0, a single 0 will be displayed.
- T_4 will contain approximately 1 million data objects in close-to-sorted order. The first object will contain a randomly generated string with 20 random digits before and after the decimal point, generated the same way as T_1. Each successive data object will be generated based on the current data object as follows. First, a randomly generated string representing a fraction between zero and one will be generated with 20 random digits after a decimal point. Then, a random integer from 1 to 10 will be chosen. If 1 through 9 are chosen, the fraction will be added to the current number (represented as a string) to obtain the new number (also represented as a string). If 10 is chosen, the fraction will be subtracted.

Every working program will be assigned a score that is based on the CPU times that the program takes to sort each of the four test cases. If $time_1 \ldots time_4$ are the CPU times required by the program when tested on T_1 through T_4, respectively, the overall score for the program will be $10 * time_1 + time_2 + time_3 + time_4$. Assuming that it works (i.e., it generates the correct output for all four test cases), *your program will be graded almost entirely based on this overall score just described*. I expect working programs; penalties for non-working programs will be discussed in class. I may take off up to a few points for poor indentation (which does not affect speed, since it is ignored by the compiler), but otherwise, you will not lose points for lack of elegance. Almost anything goes, so long as your write the program individually and do not violate any of the rules described here or in class. You may use any provided classes or routines to which you have access, including the provided sort member function of the C++ list class (which provides an implementation of merge sort) or the provided sort function from the C++ algorithm library (which provides an implementation of a modified quicksort). However, you may not write code that changes the behavior of the provided code – this will be explained further in class.

If you wish, your program may try to figure out which test case it is dealing with and use different strategies for each. You don't really have to get that perfect – if you can determine that your code has less than a one in a million chance of guessing the format of the file wrong, that is safe. (In general, if your output is not correct, and you can convince me that it was a fluke with less than a one in a million probability, I will generate new test data and run it again – but, of course, if you are wrong, the program will not get the second chance.) Your program does not have to work for any test case that does not follow one of the above specifications.

Submit your program to me by e-mail. Send the code to *CarlSable.Cooper@gmail.com* as an attachment. I will compile all programs using the g++ compiler under Cygwin, and I will test the executable under this environment; no compiler optimization options will be used for any program. (I will add –std=c++11 if required, but no other command line options will be used.) Your program is due before midnight on the night of Wednesday, April 26. If you submit the program to me early, I will test it and let you know roughly how it is doing. There is a lot of room for creativity with this assignment, so have fun with it!