PROG20799: Data Structures

Course Project

23 marks / 10%

Due Date: See SLATE

Prof. Georg Feil / Winter 2020

This project will help you get practice with pointer-based data structures and provide some real-life experience with algorithm complexity based on actual measurements. Do your work individually. This is not a group assignment. Sheridan's rules on academic integrity must be followed. If you have questions about this assignment please ask me in class, or come and see me!

You'll implement the following

- Filling a linked list containing 'Student' structures.
- Bubble Sort/Selection Sort/Insertion Sort of GPA values taken from a linked list.
- Quicksort of your GPA list using the C `qsort()` library function (you don't have to write code for the quicksort algorithm).
- Measurement of sort execution times using `clock()`.

Please do the following:

1. Download the files provided in the assignments dropbox for this project:
   student.c
   LinkedList.c
   LinkedList.h
   clock.c
   clock.h
   StudentList.txt
   StudentListHalf.txt
   In Dev-C++ create a project containing all source files (.c, .h). Get the program running so that it correctly reads the students file and prints "50000 lines read from the file".
   Important: You should only make changes in student.c, don't change any other files. If you make changes in the wrong places you risk losing marks. You may add new .c / .h files if you wish.

2. **Linked List:** Update the program so that it places the students read from the file into a linked list, using the linked list of students implementation provided (LinkedList.c, LinkedList.h). The C structure named Node holds one student's

data. Each student entry includes the following fields:
Last name, first name, ID, GPA, next pointer
Use initList() to initialize a List pointer either as a local or global variable.
Use createNode() to create a new node representing one student.
[4 marks]

3. **Sorting A:** Write a C function to sort an array of doubles in *descending* order using one of the "slow" sorting algorithms: Bubble Sort, Selection Sort, or Insertion Sort (your choice). Sort the student GPA values in descending order by calling your new function from the main function. Since these sorting algorithms require an array, you should first copy your linked list GPA values from the linked list into an array by *traversing* the linked list (see hints below). Print out the first 50 GPA values in the sorted array – the highest GPA values should appear first. **Timing measurement:** Your program should measure how long it takes to sort the list using the C library `clock()` function (described here). I've provided an improved version of the clock() function with this assignment that you should use on Windows. If you're working on Linux or Mac use the clock() function from your C library (see "Additional Instructions and Hints" below for more information). Count just the sorting algorithm in the time measurement, no I/O or printing, no data copying. Print out the time taken in seconds as a double. [7 marks]

4. **Sorting B:** Sort the student GPA values in descending order using the C library `qsort()` function. Since qsort requires an array, copy your linked list GPA values into an array by traversing the linked list. Important: Don't use the array from "Sorting A" since that's already sorted – copy the GPA values again. Print out the first 50 GPA values in the sorted array. Your program should measure how long qsort() takes. Count just the sorting algorithm in the time measurement, no I/O or printing, no data copying. Print out the time taken in seconds as a double. [7 marks]

5. **Sorting Complexity Report:** Try steps (3) and (4) above with half the student list (StudentListHalf.txt) and then the whole list (StudentList.txt). Record the time taken by the "slow" sort (Bubble/Selection/Insertion) and Quicksort in each case. Run each sort a few times and choose the best (smallest) time measurement since times can vary depending on what other programs your computer is running.

   Based on your timing values, what can you say about the performance (computational complexity, big "O") of the two sorting algorithms as 'n' changes? Write a report (Microsoft Word document) describing your findings including a table of your results and explanation of the numbers in terms of computational complexity. Use proper English sentences and paragraphs.
   [5 marks]

Additional Instructions and Hints:

- Array size: Don't hard-code the array size, use a variable or a constant.
- Traversing a Linked list: Looping through all the items in a linked list is called traversing the list. You don't want to remove the items, instead start at the head and follow "next" pointers until you reach the end (NULL). Use this to copy GPA values from the linked list into an array for sorting.
- Don't use the C library version of clock() on Windows/Cygwin since it has poor time resolution, use the clock.c and clock.h source files I've provided which have an improved version of the clock() function with microsecond resolution. On Windows don't #include <time.h> because this may create a conflict, #include "clock.h" instead.
- Here is an example of how you can measure time in seconds using clock():

```
clock_t t1 = clock();
// Code to be measured goes here
clock_t t2 = clock();
double time = (double)(t2 - t1)/CLOCKS_PER_SEC;  // Time taken in sec
```

Project Submission Instructions

Your project submission must follow all the requirements outlined below.

Submit your project using SLATE (click on the Assignments tab, then click on the Project link). Attach your source file(s) (.c, .h) and your report from question 6 (.docx) as individual files. You do not need to hand in LinkedList.c/h or clock.c/h. Do not use ZIP or RAR.

Your project must be submitted before the due date/time. Any projects submitted after this time are considered late. See our class plan on SLATE for rules governing late assignments.

Evaluation

Your program will be evaluated based on the following criteria:

Functionality: Program functions according to requirements and is structured as specified in the assignment description. A program that does not compile with gcc using the –std=c99 option will automatically receive a grade of ***zero***. If any warnings are displayed using the gcc -Wall option then one mark will be deducted. Output should be clearly labelled, don't just print numbers, print text along with numbers to indicate what they are. Print blank lines between sections of your output to make it easier to read.

Programming Style: Your code must be ***modular*** and ***organized***. It should be clear what parts of the code are for the linked list, binary tree, etc. Divide your program into more than one .c file (module) if appropriate. Use proper indentation, spacing, and comments. All identifiers shouls be descriptive (avoid one-letter variable names). Variables should be defined with appropriate data types and initialized correctly. Variables should use appropriate scope (global/module/local). For additional programming style rules see our class coding standards posted on SLATE.

Efficient Code: Program doesn't use too much repetitive code (e.g. uses functions instead of copy/pasting code). Program uses global variables where and only when necessary. Program doesn't define variables that are never used, nor does it use too many variables for unnecessary tasks. Program logic is written concisely and is not cluttered with unnecessary code.

External Documentation (if any): Good organization and clarity of written communication.

Other: All instructions regarding assignment submission and program specifications have been followed. Submission was completed as requested in a timely fashion. Techniques discussed in class have been used as appropriate.