CS 223 – Data Structures and System Programming – Spring 2017

PEX 1 – Analysis of List Implementations– 90 Points

Due @ 2300 on Lesson 13
M-Day: Tue, 7 Feb 2017
T-Day: Wed, 8 Feb 2017

Help Policy:

AUTHORIZED RESOURCES: Any, except another cadet's program.
NOTE:

- Never copy another person's work and submit it as your own.
- Do not jointly create a program unless explicitly allowed.
- You must document all help received from sources other than your instructor or instructor-provided course materials (including your textbook).

Documentation Policy:

- You must document all help received from any source other than your instructor or instructor-provided materials, including your textbook (unless directly quoting or paraphrasing).
- The documentation statement must explicitly describe WHAT assistance was provided, WHERE on the assignment the assistance was provided, and WHO provided the assistance, and HOW it was used in completing the assignment.
- If no help was received on this assignment, the documentation statement must state "None."
- If you checked answers with anyone, you must document with whom on which problems. You must document whether or not you made any changes, and if you did make changes you must document the problems you changed and the reasons why.
- **Vague documentation statements must be corrected before the assignment will be graded and will result in a 5% deduction on the assignment.**

Turn-in Policies:

- On-time turn-in is at the specific day and time listed above.
- Late penalties accrue at a rate of 25% per 24-hour period past the on-time turn-in date and time. The late penalty is a cap on the maximum grade that may be awarded for late work.

OBJECTIVES

Upon completion of this programming exercise, students will:

- Have a better understanding of a list data structure that is implemented using arrays, i.e., ArrayLists.
- Have a better understanding of a list data structure that is implemented using pointers, i.e., LinkedLists.
- Understand what big-O notation means.
- Understand how to perform an empirical analysis of an algorithm, which determines the physical time it takes an algorithm to execute on various amounts of data.

1. DESCRIPTION

A data structure is a collection of related data elements. A data structure stores both the data elements and the relationships between the data elements. Given the design of a computer's memory, the relationships in a data structures can be implemented in one of two ways: using arrays or using pointers. There are trade-offs between each approach and as a software developer you must understand these trade-offs so that you can select (or implement from scratch) an appropriate data structure for the specific problem you are solving.

For this programming exercise you will analyze the run-time behavior of various functions that implement a list data structure.

To perform an empirical analysis of an algorithm, you do the following:

- Time how long it takes for an algorithm to execute on data structures of various sizes.
- Store the data structure size and the run-time as size-time pairs into a text file.
- Use Excel to plot theses size-time pairs in a scatter plot. The x-axis is the data size. The y-axis is the associated run-time. The resulting plot describes how the run-time behavior of an algorithm changes as the size of the data structure grows large.

2. FUNCTIONALITY REQUIREMENTS

In your source code repository, in your PEXs folder, create a folder called PEX1. Then save the following five (5) source code files into your PEXs/PEX1 folder. The files are:

- `ArrayList.h` – the definitions of an array list (your implementation from lab time)
- `ArrayList.c` – the implementation of an array list (your implementation from lab time)
- `LinkedList.h` –the definitions of a linked list (your implementation from lab time)
- `LinkedList.c` – the implementation of a linked list (your implementation from lab time)
- `PEX1.c` – the main program for PEX 1 (download this file from the course web site)

Your implementations for **both** data structures must implement the following functions:

```
ArrayList  *arrayListCreate(int initial_size);
void        arrayListDelete       (ArrayList *list);
void        arrayListAppend       (ArrayList *list, ElementType *element);
void        arrayListPrint        (ArrayList *list);
ElementType arrayListGetElement   (ArrayList *list, int position);
void        arrayListDeleteElement(ArrayList *list, int position);
void        arrayListInsertElement(ArrayList *list, int position, ElementType value);
void        arrayListChangeElement(ArrayList *list, int position, ElementType newValue);
int         arrayListFindElement  (ArrayList *list, ElementType value);
```

Use the "unit test" programs provided on the course web site to verify that your implementations are correct.

Your `PEX1.c` program must perform the following tasks:

- Time how long it takes to **create an array list** for various list sizes. (Note: You don't need to time how long it takes to create a linked list because it has no initial size.)

- Time how long it takes to **<u>fill an array list</u>** with values if its initial size is large enough to hold all the values it needs to store. Use *append* to fill the array.
- Time how long it takes to **<u>fill an array list</u>** with values if its initial size always starts at 10. Use *append* to fill the array.
- Time how long it takes to **<u>fill a linked list</u>** with values. Use *append* to fill the array.
- Time how long it takes to **<u>fill a linked list</u>** with values if each new value is *inserted* into the middle position of the list.
- Time how long it takes to **<u>sort an array list</u>** using an *insertion sort*. (You will have to add an insertion sort function to the `ArrayList` header and `ArrayList` source file.)
- Time how long it takes to **<u>sort a linked list</u>** using a *selection sort*. (You will have to add a selection sort function to the `LinkedList` header and `LinkedList` source file.)

Insert random values in the range [0, size) into the lists, where size is the size of the list. (See the HELPFUL HINTS section for how to create random values.)

Your program must meet the coding standards described in the "C Programming Standards" document on the course web site. Your code must be divided into functions, where each "timing task" is performed by a separate function.

Your final turn-in will include your code files, your *.csv data files that contain the raw data of your timing tests, and a single Excel spreadsheet that contains all of the timing data with associated scatter plots. Each timing test must be in a separate tab of the spreadsheet. Each tab must have an appropriate name.

3. GETTING STARTED

Verify that you have a correct `ArrayList` and `LinkedList` implementation by executing the "unit tests" provided for this assignment. If you have any problems with the "unit tests" or making corrections to your implementations, please get help from your instructor before proceeding with the rest of this assignment.

The first "timing task" has been solved for you. Study this code and make sure you understand how it works. You will probably want to copy and paste this code into separate functions to accomplish the other timing tasks. Refer to the HELPFUL HINTS section below for a description of the timing code and how to write values to a file.

You do not have to perform all of the timing tasks in a single execution. In fact, you are encouraged to perform each timing task in a separate execution. (This is "incremental code development" as you learned from CS210.) However, your final version of the `PEX1.c` must contain all of your timing functions and a call to each function from main, but the function calls in main can be commented out.

4. HELPFUL HINTS

To create a random integer in the range `[0, size)`, include "`stdlib.h`" and use

```
     randomValue = (int) ((float) rand() / (float) RAND_MAX * size);
```

Concerning the timing of a section of code:

- Include the `time.h` library definitions.

- Before you execute the code that you want to time, call `clock()` to get the current time of the computer's clock. The time is in milliseconds. The time value is of type `clock_t`, which is an *unsigned long integer*.
- After you execute the section of code you want to time, call `clock()` again and subtract the original starting time. This gives you the elapsed time for the code execution.
- To print an *unsigned long integer*, the format specifier is `"%lu"`. (The "`l`" means "long". The "`u`" means "unsigned.")

Concerning the writing of data to a file:

- To make sure your data files can be stored in the appropriate folder of your repository, include this line after your "project" statement in your `CMakeLists.txt` file:

  ```
  set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${CMAKE_CURRENT_SOURCE_DIR})
  ```

  When you build your program, all file paths will be relative to the root folder of your source code repository.

- The code to write data to a text file is almost identical to its equivalent Python code. You must open the file for "writing", write your data, and then close the file. Example code is included in the `PEX1.c` file and is shown here:

  ```
  fp = fopen("PEXs/PEX1/timesForArrayListCreate.csv", "w");

  fprintf(fp, "%d, %lu\n", size, elapsedTime);

  fclose(fp);
  ```

- To write data to a file, use the `fprintf()` function which takes a "file pointer" as its first argument. Otherwise the `printf()` and `fprintf()` functions are identical.

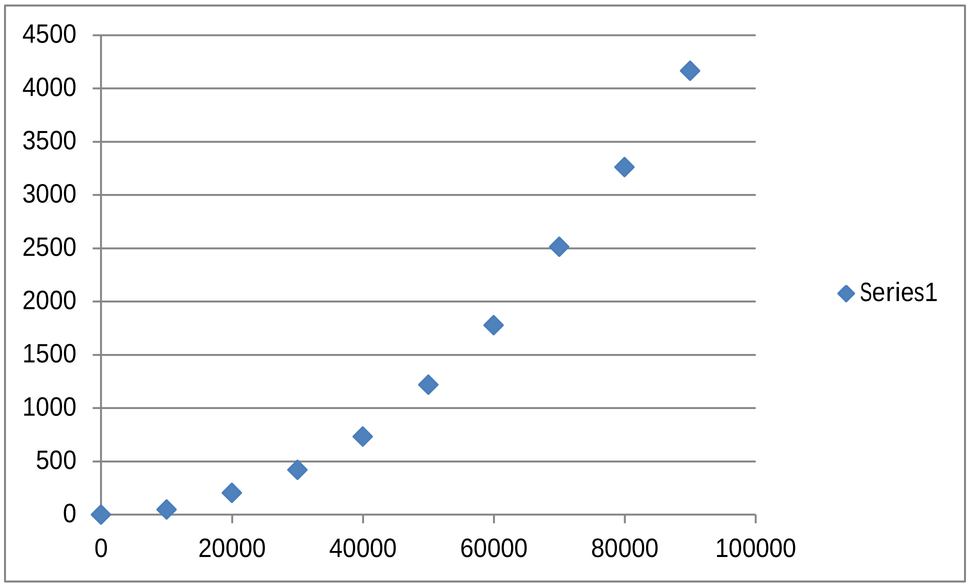
- You should name your file with a `.csv` file extension. A "csv" file is a "comma separated value" file that contains values separated by commas. Make sure you print a comma between each value on each line in the file. A "csv" file will open automatically in Microsoft Excel if you double-click on it.

- Your csv data file must contain a series of lines where each line represents a single timing test. Each line must contain two values: 2) the size of the data structure, and 2) the amount of time it took to process a data structure of that size. Here is an example output, where the size of the list is increasing by 10,000 on each timing test:

  ```
  0, 0
  10000, 47
  20000, 203
  30000, 421
  40000, 733
  50000, 1217
  60000, 1778
  70000, 2512
  80000, 3261
  90000, 4165
  ```

- Your final turn-in must include a single Excel Spreadsheet that contains all of your timing data, but each "timing" must be stored in a separate worksheet (tab). The easiest way to do this is to drag-and-drop the tab from the CVS file into your consolidated spreadsheet file. To be able to drag-and-drop worksheets, do not maximize the worksheets inside Excel.

- Each worksheet must include a scatter plot of your timing data. In Excel, select your data, then from the "tools ribbon" select Insert à Scatter à "Scatter with only markers". You should see a chart similar to the one below. (This specific example indicates that the time complexity of the timed algorithm grows at a polynomial rate based on the size of the data structure. We would say this algorithm has a run-time complexity of O(n^2).



5. Submission Requirements

- The **`PEXs/PEX1`** folder of your source code repository must contain the following files. (Make sure you have used `Git à Add` to make them part of your repository.
  - o Code files: `ArrayList.h, ArrayList.c,`
    `LinkedList.h, LinkedList.c,`
    `Pex1.c`
  - o Data files: Eight (8) CSV files, one for each timing problem.
  - o Final results: `AllTimings.xlsx` – a single spreadsheet that contains your data and scatter plots.

- Please make sure you *commit* and *push* your repository before the due date and time.

PEX 1 Grade Sheet **Name:**________________________________

	Points	
Criteria	Earned	Available
Coding standards	10%	
Your code follows the C coding standards.	**9**	
Functionality	80%	
Time the creation of an ArrayList. (This was done for you.)	**0**	
Time the filling of an ArrayList (initial size is big enough for all data).	**9**	
Time the filling of an ArrayList (initial size is always 10).	**9**	
Time the filling of a LinkedList using append.	**9**	
Time the filling of a LinkedList when inserting a new value in the middle of the list on each insertion.	**9**	
Implement insertion sort. Time an insertion sort on an ArrayList.	**18**	
Implement selection sort. Time a selection sort on a LinkedList.	**18**	
Algorithm Analysis	10%	
Your timing data and charts in the Excel Spreadsheet are correct.	**9**	
Subtotal	**90**	
Vague/missing documentation statement (-5%)	**- 4.5**	
Submission requirements not followed (-5%)	**- 4.5**	
Poor software development work habits (shown by your Git commit history)	**- 9**	
Late penalties – 25/50/75/100% per 24-hours past due date/time	**- 23/45/67/90**	
Total	**90**	

Comments: