CSE 109: Systems Programming

Fall 2017

Program 5: **Due on Monday, October 30th at 9pm on CourseSite.**

Collaboration Reminder:

1. You must submit your own work.
2. In particular, you may not:
   (a) Show your code to any of your classmates
   (b) Look at or copy anyone else's code
   (c) Copy material found on the internet
   (d) Work together on an assignment

Assignment: Preparation

1. Make a *Prog5* directory in your class folder.
2. Copy the files *generator* and *reader* from the *p5student* folder.
3. You will be writing a *ListFile.c*, *ListFile.h*, *Node.c*, *Node.h* and a *prog5.c* file for this assignment.
4. All source code files must have the comment block as shown at the end of this document. All files must be contained in your *Prog5* directory.

Assignment:
In this assignment, you are making a singly linked list that supports in-order insertion (by name) and holds *raw* data. It also supports saving the contents of the linked list to disk and restoring them from disk.

1. struct *ListFile* will be created with the following behaviors:
   (a) *ListFile* will contain a pointer to a *head Node* object that contains the first element of the list and points to successive elements.

(b) Each element of the list will contain dynamically allocated space for the *name* of the element and the *data* of the element. It will contain whatever other members that you need to maintain the properties of a linked list.

    name will be guaranteed to be a proper C-string, more specifically, it won't have any whitespace or funny characters in it.

(c) An empty list constructor called *makeList* that returns an empty *ListFile* object.

(d) A copy list constructor called *copyList* that returns a copy of the *ListFile* object provided to the constructor.

(e) A destructor called *destroyList* that deallocates the *ListFile* object and returns NULL.

(f) *findByName(struct ListFile**, *char**), return a pointer to the corresponding *data* for an element with a matching *name*. Return NULL otherwise.

(g) *findByData(struct ListFile**, *void**, *int)*, returns the number of elements in the list that contain identical *data*.

(h) *removeByName(struct ListFile**, *char**) return 1 if a matching element was removed. Insertion rules will prevent duplicates.

(i) *removeByData(struct ListFile**, *void**, *int)*, returns the number of elements that were removed, since it can remove more than one element.

(j) *insert(struct ListFile**, *char**, *void**, *int)*, inserts a *name/data* pair into the list. Returns 1 if successful. Returns 0 if there is a conflicting *name*. This inserts the data into the list in alphabetical order (use strcmp).

(k) *readFile(char**), returns a *ListFile* object with the contents of the given filename. Note that if the file contains duplicate *name* elements, only accept the first element. Return NULL if the file is empty or otherwise fails.

(l) *appendFromFile(struct ListFile**, *char**), appends the elements from the file to the given list. Returns the number of elements added. Return -1 if the the file could not be opened for any reason.

(m) *saveToFile(struct ListFile**, *char**), saves the given list into the filename provided. Returns 0 if successful, -1 otherwise.

(n) *getElement(struct ListFile*, int)* returns a pointer to the *data* portion of the element at the given index. Returns a NULL if the index was invalid.

(o) *getElementName(struct ListFile*, int)* returns a pointer to the *name* portion of the element at the given index. Returns a NULL if the index was invalid.

2. *prog5.c* will contain a main function that will interact with your code in *ListFile*. Create something that uses all of the functions, specifically, it should read in at least 2 different data files and save another one: specifically, any input files should be named *input.0*, *input.1* and so on, output files *output.0*, *output.1*. This allow us to generate input files ourselves that your program will be aware of. Use the provided executables to create randomish data files to make sure your tester catches everything.

Output File Format:

1. The first four bytes of the file signify the number of elements contained within the list.
2. After the first four bytes, each element is stored, successively.
3. Each element consists of four pieces, written in the following order without any gaps or spaces:
    - Four bytes to signify the length of the filename (*nameLength*).
    - Four bytes to signify the length of the data (*dataLength*).
    - *nameLength* bytes containing the element name. No null terminator is to be stored.
    - *dataLength* bytes containing the data.

Testing:

1. You can use the provided executables to generate sample lists as well as read the contents of a correctly formatted list file.

    ./generator takes in two arguments. The first is the number of elements to make and the second is which generation *seed* to use.

    ./reader will take in a list file as standard input and output the contents of the list file.

2. You may want to generate your own in order to make more readable lists.

3. Remember that the sorting behavior is by strings, so it is a lexicographical sort, not a numeric sort.

Submission:

1. Once ready to submit, you can package up the assignment as a .tgz file

   ```
   tar -czvf Prog5.tgz Prog5
   ```

   You must use this command in the directory that contains the *Prog5* folder, not within the directory.

2. Transfer *Prog5.tgz* to the Program 5 submission area of CourseSite.

Comment Block:

```
/*
    CSE 109: Fall 2017
    <Your Name>
    <Your user id (Email ID)>
    <Program Description>
    Program #5
*/
```