CSE 109: Systems Programming

Fall 2018

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

Checkpoint Due: **Due on Sunday, October 28th at 9pm via checkpointer.**

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*, *reader*, *Node.h* and *ListFile.h* from the *prog5student* folder.
3. You will be writing a *ListFile.cpp* and *Node.cpp* for this assignment.

   You can write a *prog5.c* file, but we won't use it.
4. Do not alter the header files. Only the originals will be used during grading.
5. 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. class *Node_t* will be created with the following behaviors:

   (a) Default construction is removed. You may not defaultly create *Node_t* objects.

   (b) Explicit constructor(const string& name, void* data, size_t size, Node_t* next), initializes a *Node_t* object with the values given. The *Node_t* object will have complete ownership on the elements and will be responsible for malloc/free, new/delete.

   (c) Destructor deallocates all of the elements contained within the *Node_t* object. It does not free the actual *Node_t* object, delete is responsible for that.

   (d) Accessors and mutators: *getName, setName, getData, setData, getNext, setNext, getNodeSize.* The mutators will not return anything. The accessors will be const.

       The accessor, *getName*, must return a copy to prevent accidental changes to the names of each *Node_t* object.

       The mutator, *setData*, will take a *void** followed by a *size_t* reflecting the size of the data.

       There will be no direct *setNodeSize* method, it is set by the behavior of *setData*

2. class *ListFile_t* will be created with the following behaviors:

   (a) *ListFile_t* will contain a pointer to a *Node_t* object that contains the first element of the list and points to successive elements (head node).

   (b) *ListFile_t* will also contain a size_t to indicate how many elements are contained within the linked list.

   (c) Each element of the list (*Node_t*) will contain dynamically allocated space for the *name* of the element (via std::string) and the *data* of the element - it will have its own copies of these, it will not share them. It will contain whatever other members that you need to maintain the properties of the *Node_t*.

       name will be guaranteed to be a proper std::string.

   (d) Default constructor initializes an empty *ListFile_t* object to refer to an empty list.

   (e) Copy constructor that causes the new *ListFile_t* object to refer to a deep copy of some existing *ListFile_t* object.

(f) operator= that causes a *ListFile_t* object to refer to a deep copy of some existing *ListFile_t* object.

    A subtle difference from the copy constructor, this method assumes the *ListFile_t* object is already initialized. It also returns a reference to itself.

(g) Destructor deallocates the contents of the *ListFile_t* object. It does not free the actual *ListFile_t* object, delete is responsible for that.

(h) *readFile* that replaces the contents of an existing *ListFile_t* object with the contents of the data read in from a file (insert in-order). The filename will be provided as a std::string. If, in any way, the file is not valid do not alter the contents of the existing *ListFile_t* object and return -1. Otherwise, the existing *ListFile_t* object now reflects the contents of the file and we return 0.

(i) *appendFromFile* appends (in-order, so not literally at the end) the elements contained by the file referred to from the std::string given to *appendFromFile*. If, in any way, the file is not valid do not alter the contents of the existing *ListFile_t* object and return -1. Otherwise, the existing *ListFile_t* now contains 0 or more additional elements. Return the number of additional elements that were successfully added.

(j) *saveToFile* files (overwrites) into a file specified by a std::string the contents of the *ListFile_t* object. Returns -1 is unsuccessful, otherwise returns 1.

(k) A const accessor called *getSize* that returns a size_t representing the number of elements within the *ListFile_t* object.

(l) A const accessor called *getElementName* Returns a std::string referring to the name of the *nth* element of the *ListFile_t* object.

(m) A const accessor called *getElementData* Returns a *void** referring to the data of the *nth* element of the *ListFile_t* object.

(n) A const accessor called *getElementSize* Returns a *size_t* referring to the size of the data of the *nth* element of the *ListFile_t* object.

(o) A const index operator that returns a const reference to the*nth* *Node_t* element of the *ListFile_t* object.

(p) A non-const index operator that returns a non-const reference...

(q) void *clear* that empties the *ListFile_t* object.

(r) bool *exists* (const) that returns 1 if there is an element in the *ListFile_t* object that shares the same name as the name referred to by the std::string given to *find*. Otherwise, returns 0.

(s) size_t *count* (const) that returns the number of elements that have matching data.

(t) *removeByName* similiar to *exist* but also removes the element with the matching name. Return 1 if a matching element was removed, otherwise return 0. Insertion rules will prevent duplicates.

(u) *insert(const string& name, void* data, size_t size)*, inserts a *name/data/size* into the list as a *Node_t* object. Returns 1 if successful. Returns 0 if there is a conflicting *name*. This inserts the data into the list in alphabetical order (use strcmp).

(v) *insertInternal* is private, you do not have to use it. The answer key uses it. You do not have to.

Output File Format:

1. The first eight bytes of the file signify the number of elements contained within the list.
2. After the first eight bytes, each element is stored, successively.
3. Each element consists of four pieces, written in the following order without any gaps or spaces:
    - Eight bytes to signify the length of the node name (*nameLength*).
    - Eight 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.

Checkpointing:

1. The *Node_t* object, in its entirety (Node.c and Node.h) are due at checkpoint time.
2. You may call other functions as long as everything is included in your checkpoint submission.

3. To submit your checkpoint:

```
cp Node.h checkpoint5.c
cat Node.cpp >> checkpoint5.c
~jloew/CSE109/submitCheckpoint.pl 5
```

   That is lowercase PL, followed by the number 5.

4. You may submit your Checkpoint up to five times before the deadline. You may not submit it afterwards. You may lose credit for submissions past five.

5. Ideally, it will tell you which functions are incorrect. It is possible that the functions are incorrect but pass the checkpoint. Although, they should be mostly correct or completely correct if they pass the checkpoint.

6. The checkpoint will not check for memory corruption or leaks. You will need to handle that yourself.

Style:

For assignments, we follow the Allman style of braces and indentation.

1. **Review the Style document on Coursesite**

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. The lists generated from this are not in alphabetical order!

   ./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 2018
    <Your Name>
    <Your user id (Email ID)>
    <Program Description>
    Program #5
*/
```