CSE 109: Systems Programming

Fall 2018

Program 3: **Due on Sunday, October 7th at 9pm on CourseSite.**

Checkpoint Due: Monday, October 1st at 9pm, via Checkpoint submissions.

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 *Prog3* directory in your class folder.
2. Create the files *List.c*, *List.h*, *Node.c* and *Node.h*
3. Copy the provided *prog3.c*, *ListTraverse.o* and *ListTraverse.h* files from ∼jloew/CSE109/

   Do not edit these files unless told to do so. Do not edit them.
4. All your source code files must have the comment block as shown at the end of this document. All files must be contained in your *Prog3* directory.

Assignment:
You will be creating both header and implementation files for this assignment.

1. *Node.h*

(a) Use conditional compilation to prevent *Node.h* from being included more than once.

(b) Declare the *Node_t* struct.

    Do not use a typedef, nor an anonymous struct.

(c) Include as few *#includes* as possible.

    Node.c will contains more *#includes* since it is doing the real work.

(d) The Nodes will contain a link to the next Node as well as the value that the Node contains.

(e) Declare the following function prototypes (You may give names to the arguments if desired.)

    i. struct Node_t* makeNode1(struct Node_t *);
    ii. struct Node_t* makeNode2(struct Node_t *, int);
    iii. struct Node_t* makeNode4(struct Node_t *, int, struct Node_t * next);
    iv. void freeNode(struct Node_t *);
    v. int setData(struct Node_t *, int);
    vi. struct Node_t* setNext(struct Node_t *, struct Node_t* next);
    vii. int getData(struct Node_t *);
    viii. struct Node_t* getNext(struct Node_t *);

2. *Node.c*

    (a) Provide the implementation for all of the code specified by *Node.h*

    (b) If the *this/self/it* pointer provided is NULL, just use it anyway, the resulting crash is sufficient.

    (c) The default values for construction are the data is 0 and the pointer is NULL.

        i. struct Node_t* makeNode[124](struct Node_t*, ...): Constructs the *Node_t* object by initializing. Returns the pointer to the newly constructed *Node_t* object. Uses the values provided, if any. Otherwise, uses the default construction values.

        ii. void freeNode(struct Node_t*): Deallocates the data provided *Node_t* object. Does not deallocate anything else. The *Node_t* object is **not** deallocated. The user must still deallocate the object but now they can do so safely.

            This will be tested by the checkpoint. The final program might not end up using it due to the *Free List*.

   iii. Accessors and Mutators. Defined as written. You **must** use these and not attempt to directly access your instance/member variables (You will lose style points for not doing so).

3. *List.h*

   (a) Define the *List_t* struct. It will contain:

      i. A size_t to indicate the size of the list.
      ii. A size_t to indicate the size of the *Free List*.
      iii. A *Node_t* pointer that refers to the start of the *List_t*.
      iv. A *Node_t* pointer that refers to the start of the *Free List*.
      v. Associated metadata so you can deallocate the *Free List*.

   (b) Use conditional compilation to prevent *List.h* from being included more than once.

   (c) Declare the *List_t* struct.

      Do not use a typedef, nor an anonymous struct.

   (d) Include as few *#include*s as possible.

      List.c will contains more *#include*s since it is doing the real work.

   (e) Make sure to include *Node.h* as your the List will be using Nodes.

   (f) Declare the following function prototypes (You may give names to the arguments if desired.

      i. struct List_t* makeList(struct List_t *);
      ii. void freeList(struct List_t *);
      iii. size_t size(struct List_t *);
      iv. size_t freeSize(struct List_t *);
      v. struct Node_t* insert(struct List_t *, int);
      vi. size_t find(struct List_t *, int);
      vii. size_t removeItem(struct List_t *, int);
      viii. struct Node_t* getHead(struct List_t *);

   (g) You may have additional, private helper methods, these must be prototyped in *List.c* since they are not intended for public usage.

4. *List.c*

   (a) Define all the functions specified in *List.h*.

   i. struct List_t* makeList(struct List_t*): Creates a new *List_t* object and initializes it. Returns the pointer to the newly constructed *List_t* object.

      At this point, both pointers of the *List_t* object will be NULL and the size members will be 0.

   ii. void freeList(struct List_t*): Deallocates the data provided *List_t* object. Which includes deallocating all *Node_t* elements that are referred to by the List, as well as the *Free List*. The *List_t* object is **not** deallocated. The user must still deallocate the object but now they can do so safely.

   iii. size_t size(struct List_t*): Returns the size of the *List*.

   iv. size_t freeSize(struct List_t*): Returns the size of the *Free List*.

   v. struct Node_t* insert(struct List_t*, int): Adds the given value to the end of the *List* by acquiring a *Node_t* from the *Free List*. Return a pointer to the *Node_t* that refers to the inserted element.

   vi. size_t find(struct List_t*, int): Return the number of elements contained within the *List* that match the desired value,

   vii. You will be provided traverse via the file *ListTraverse.o*.

   viii. size_t removeItem(struct List_t*, int): Remove **all** occurrences of the given value from the *List* and returns those *Node_t*s to the *Free List*. Return the number of elements removed.

   ix. struct Node_t* getHead(struct List_t*): Returns a copy of the pointer to the beginning of the elements tracked by the *List_t* object.

(b) Free List: The behavior of the *Free List* is as follows.

   i. When you need to allocate a *Node_t* object, do not call malloc. Instead, use a *Node_t* object that the *Free List* points to.

   ii. If the *Free List* is empty and we need an *Node_t* object, then we need to malloc more space for our *Node_t* objects.

   iii. To malloc more space, allocate as close to, but not exceeding, 4096 bytes at a time. For example, if your *Node_t* objects require 32 bytes, you would take 4096/32 and end up allocating 128 *Node_t* objects, as a single malloc.

Your *Node_t* object should end up being 16 bytes in size (so you should be allocating 256 *Node_t* objects at a time).

iv. After mallocing more space, the *Free List* will point to the start of that space and have its size member increased appropriately.

v. After mallocing more space, you must have the individual *Node_t* objects point to each other, as if they were a large list.

Checkpointing

1. The Node object, in its entirety (*Node.h* and *Node.c*) are due for the checkpoint.

2. You may call other functions, etc, as long as everything is included in your checkpoint submission.

3. To submit your Checkpoint:

   ```
   cp Node.h checkpoint3.c
   cat Node.c >> checkpoint3.c
   ~jloew/CSE109/submitCheckpoint.pl 3
   ```

   That is lowercase PL, followed by the number 3.

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. *prog3.c* references the expected code in *List.c*, *List.h*, *Node.c* and *Node.h*. Look through it to figure out how the command line arguments can be used to test your code.

2. You will need to use multiple steps to compile your code since you will have more than one .c source file:

   ```
   module load gcc-7.1.0
   gcc -Werror -Wall -g -c prog3.c
   gcc -Werror -Wall -g -c List.c
   gcc -Werror -Wall -g -c Node.c
   gcc -Werror -Wall -g -o List prog3.o List.o Node.o ListTraverse.o
   ```

Submission:

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

   ```
   tar -czvf Prog3.tgz Prog3
   ```

   You must use this command in the directory that contains the *Prog3* folder, not within the directory.

2. Transfer *Prog3.tgz* to the Program 3 submission area of CourseSite.

Comment Block:

```
/*
    CSE 109: Fall 2018
    <Your Name>
    <Your user id (Email ID)>
    <Program Description>
    Program #3
*/
```