Implementing the Linked List in Lab 4

Contents

Version 0

In lab 4 you will write the 4 functions that we use in the linked list library:

- insert
- iterate
- sort
- deleteSome

Signatures

Your code **must** use the exact signatures used by the library functions; same name, same parameters, same return type.

Your code must use the following declaration for a node:

```
typedef struct Node{
        struct Node *next;
        void *data;
        }Node;
```

The node declaration should **not** be visible outside your linkedlist.c file. This is the same declaration that the library code uses. If you use it, you can mix calls into your code and calls into the library code freely. If you use anything else, you risk going insane trying to track down mysterious pointer bugs in code that looks perfect.

Files

All of your linked list code should be in your linkedlist.c file. All supporting functions must be declared static (internal linkage), leaving only the functions listed above visible outside the file. The headers.sh script should work correctly (it doesn't create declarations for internal linkage functions).

Dynamic Memory

As before, allocation and free calls should be inside a function that has a static int to count the number of allocations and frees. So you might want an "allocate_node()" and "free_node()" function with appropriate parameters and return types. They must be in linkedlist.c, they must have internal linkage. Your code should print diagnostics in text mode:

```
DIAGNOSTIC: 1 nodes allocated.
DIAGNOSTIC: 1 nodes freed.
```

Your list code must tolerate allocation failures. If insert fails to allocate a node, return false and let the application deal with it. This *will* happen when you implement the unreliable memory code (See Unreliable).

Visibility

The list code must **not** #include the structs.h or equivalent file. In short, the list code must not know in any way what the data is. The list deals in void pointers and only void pointers for data.

No code outside linkedlist.c is allowed to know what a node is.

Only the four functions that provide the public interface to the list are allowed to be visible outside the list code.

Plan of Attack

Until you have all 4 required functions completed, retain the **`-llinkedlist`** item in your makefile rules. This lets you use your code (linkedlist.c) as well as the library (for functions you haven't written yet). Do the easy ones first:

1. iterate
2. insert
3. deleteSome

4. sort

You don't have to do any dynamic memory until you get to insert and deleteSome. You could get 4 prototypes done with the list code alone. Recycle some of your lab 3 list prototypes and point them at your code.

Have your list functions issue a debug print message so that you know when your code is being used and when the library is being used.

Once you have the 4 required functions written, **remove** the **`-llinkedlist`** item in your makefile and test to be sure that your code is being used.

There is a more elaborate plan of attack in the L4_00_Dates document.

Terse

You are allowed to use the terse insert code from the slides, properly modified, for your insert. If you use terse insert, you are **required** to use terse delete. Terse insert is quite short. Terse delete is something like 8 lines.

Ten-line Limit

If you use the long method for insert, your insert does not have to meet the ten-line limit. If you use the long form of deleteSome, that function does not have to meet the ten-line limit.

Sort

If you implement sorting, do not change nodes, swap data instead. I suggest bubble sort. Sort is not easy. Your sort must not infinite loop. You might want to code it in a way that deals with equality comparisons gracefully.