Assignment 4: Dynamic List

Topics: Singly-linked list, Dynamic Memory, Valgrind, Procrastination,
Reading Comprehension
Date assigned: Wednesday, October 3, 2018
Date due: Part 1: Wednesday, October 10, 2018
Part 2: Wednesday, October 17, 2018
Points: 40

For this assignment, you are to implement the List ADT in a file called **list.c** using the header file **list.h**. You can find this header file on zeus in **/home/CS300Public/2018/04Files**. All of the data structures and function prototypes are defined in list.h. Further, each function prototype has been described to the point that you should be able to implement each list function in the file list.c.

In addition to implementing the list data structure, you must provide a Makefile and test driver (**listdriver.c** that produces an executable named **listdriver**) that thoroughly tests your list functions. The listdriver must display to the screen a series of SUCCESS or FAILURE messages with enough description that a user can quickly spot broken list functionality. Code your driver for SUCCESS tests. No test in your driver should Fail. If a FAILURE happens, your program is to terminate.

You may add any helper static functions you need to list.c. You may not alter list.h in anyway.

1. Your code is to be written in C using Eclipse. Programs written in other environments will not be graded. Create an Eclipse project named **GenericDynamicList**. This project must contain the directories: src, include, and bin.

2. The Makefile must contain the necessary targets to build the listdriver as well as a clean, valgrind, and tarball targets. Typing **make** on the command line must build listdriver in the bin directory.

3. Your program must not have any Valgrind errors.

4. Submit a color, double-sided, stapled packet of code by that same deadline. The packet must be in the following order:

    List Driver (.h then .c if you have both, otherwise just .c)
    list.c (do not print list.h)
    Any extra .h/.c modules
    Makefile

5. Test one function at a time and profile with Valgrind. This will lessen your level of frustration greatly. Commit after writing and testing a significant function. You will not be sorry.

6. You are to use the coding guidelines of the coding standards on the CS300 Web page.

Goals for this assignment:

1. Code and test your program one function at a time.
2. Write efficient/clean code
3. Use the debugger and Valgrind to effectively develop a correct solution
4. Thoroughly test your code.
5. Fully understand Makefiles.
6. Continue to use Subversion.

The list.h header file as well as a list of error codes that each function can produce are part of list.h. Further, the error codes are listed in order of precedence. If a function can produce multiple errors, the function must return the error code first on the list.

Since the interface for the list may be hard to understand at first, here is a very small example of how to walk a list and print out every element. For brevity, no error checking is done.

Sample Code	Run Results of Sample Code

<pre>lstLoadErrorMessages ();
lstCreate (&sTheList);
 	
printf ("List size = %d\n", lstSize (&sTheList));
lstInsertAfter (&sTheList, &charValue, sizeof (char));
//assert(lstSize(&sTheList) == 1, "Size is 1", "Size i:
 	
printf ("List size = %d\n", lstSize (&sTheList));
charValue = 'B';
lstInsertAfter (&sTheList, &charValue, sizeof (char));
//assert(lstSize(&sTheList) == 2, "Size is 2", "Size i:
 	
printf ("List size = %d\n", lstSize (&sTheList));
 	
size = lstSize (&sTheList);
lstFirst (&sTheList);
 	
for (i = 0; i < size; ++i)
{
  lstPeek(&sTheList, &charValue, sizeof(char));
  printf ("%c\n", charValue);
 	
  lstNext (&sTheList);
}
lstTerminate (&sTheList);</pre>
	<pre>Program Start
	List size = 0
	List size = 1
	List size = 2
	A
	B
	Program End</pre>

Part A: Here is a list of the functions that must be completed for Part A and the order in which I recommend you implement each function:

1. lstLoadErrorMessages
2. lstCreate
3. lstInsertAfter
4. lstTerminate
5. lstSize
6. lstIsFull
7. lstIsEmpty
8. lstFirst
9. lstNext
10. Don't forget to build a driver that tests each of these functions!

Submit your assignment as **cs300_4A_punetid.tar.gz**

Part B: Implement all functions in list.h. Add code to your driver to test all of the list functions. Submit your solution as **cs300_4B_punetid.tar.gz**.

lstInsertBefore and **lstDeleteCurrent** are difficult.

To view data using a void*, you will need to use the **Expressions tab**. In the debugger, you will need to enter expressions in the **Expressions tab**. (Window | Show View | Expressions) The Expressions tab is shown below.

You can click **Add new expression** and type in a C code expression such as:
 (int)psListElement->pData

(x)= Variables ●◉ Breakpoints 6⚿ Expressions ⛶ 1010 0101 Registers ▣ Modules		
Expression	Type	Value
(x)=*(int*)psListElement->pData	int	2
➕ Add new expression		

I expect you to start this project early. **This code will be reused in subsequent assignments.** Coding this last minute will cause headaches for much of the rest of the course!

To be on schedule, you should have your Eclipse project build and Makefile done by Friday, Oct 5!

Go back and read the assignment carefully one more time!

Use this page to draw yourself some Lists!