

Assignment 4: Dynamic List

Topics:	Singly-linked list, Dynamic Memory, Valgrind, Procrastination
Date assigned:	Wednesday, October 4, 2017
Date due:	Part 1: Wednesday, October 11, 2017 Part 2: Wednesday, October 18, 2017
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/2017/04Files**. 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. 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.

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

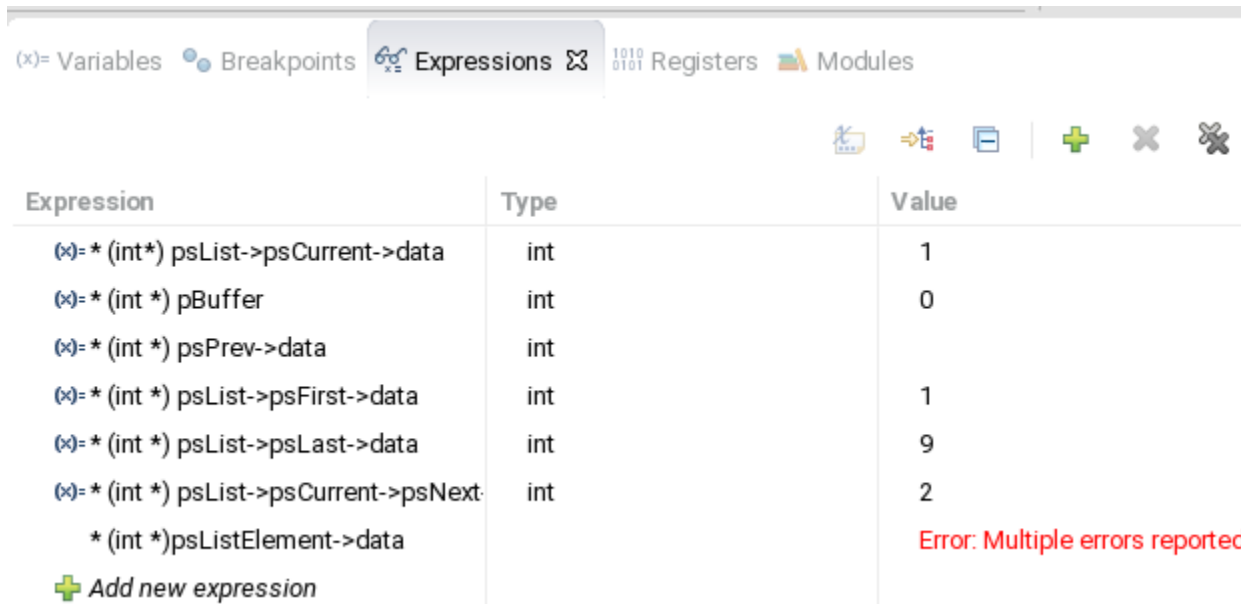
Part A: Here is a list of the functions that must be completed for Part A and the order in which I recommend you implement each function:

1. lstLoadErrorMessages
2. lstCreate
3. lstInsertAfter
4. lstTerminate
5. lstSize
6. lstIsFull
7. lstIsEmpty
8. lstFirst
9. lstNext

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

Part B: Implement all functions in list.h and write a driver that thoroughly tests all functions. Submit your solution as **cs300_4B_punetid.tar.gz**.

If you found Part A pretty easy, don't be fooled that Part B will be just as easy. Specifically, lstInsertBefore and lstDeleteCurrent are extremely hard. The debugging challenges for Part B are much harder than Part A. Over the years I have written a solution to this assignment multiple times, and this time I still had some difficult errors to find. What makes debugging especially difficult is void * is a typeless pointer. In the debugger, you will need to enter expressions in the Expressions tab. As I was debugging, here is an example of what my Expressions tab looked like:



The screenshot shows a debugger window with the 'Expressions' tab selected. The window has a toolbar with icons for variables, breakpoints, expressions, registers, and modules. Below the toolbar is a table with three columns: 'Expression', 'Type', and 'Value'. The table contains several entries, each with a red 'x' icon in the first column. The last entry has a red error message in the 'Value' column.

Expression	Type	Value
(x)- * (int*) psList->psCurrent->data	int	1
(x)- * (int *) pBuffer	int	0
(x)- * (int *) psPrev->data	int	
(x)- * (int *) psList->psFirst->data	int	1
(x)- * (int *) psList->psLast->data	int	9
(x)- * (int *) psList->psCurrent->psNext	int	2
* (int *)psListElement->data		Error: Multiple errors reported.
+ Add new expression		

If you have Makefile problems, see me early. I do not want to see more code in your project than list.h, a single function lstCreate in list.c, and a single call to lstCreate in listdriver.c. I'm not trying to be difficult, but the more code you have, the harder it is to fix a Makefile problem.

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!!!! **You've been warned.**