Assignment 4: Dynamic List

Topics:	Singly-linked lists, Dynamic Memory, Valgrind, Procrastination
Date assigned:	Wednesday, October 5, 2016
Date due:	Part 1: Wednesday, October 12, 2016 Part 2: Wednesday, October 19, 2016
Points:	40

For this assignment, you are to implement the List ADT in a file called **dlist.c** using the header file **dlist.h**. You can find this header file on zeus in **/home/CS300Public/2016/04Files**. All of the data structures and function prototypes are defined in dlist.h. Further, each function prototype has been described to the point that you should be able to implement each list function in the file dlist.c.

In addition to implementing the list data structure, you must provide a Makefile and test driver (**dlistdriver.c** that produces an executable named **dlistdriver**) that thoroughly tests your list functions. The dlistdriver must display to the screen a series of SUCCESS or FAILURE messages with enough description that a user can quickly spot broken list functionality.

You may add any helper static functions you need to dlist.c. You may not alter dlist.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 dlistdriver as well as a clean, valgrind, and tarball targets. Typing **make** on the command line must build dlistdriver 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)
   dlist.c (do not print dlist.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.
6. You are to use the coding guidelines from V6.3 of the coding standards.

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 dlist.h header file as well as a list of error codes that each function can produce are part of dlist.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 highest 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

```
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");
```

Run Results of Sample Code

```
Program Start
List size = 0
List size = 1
List size = 2
A
B
Program End
```

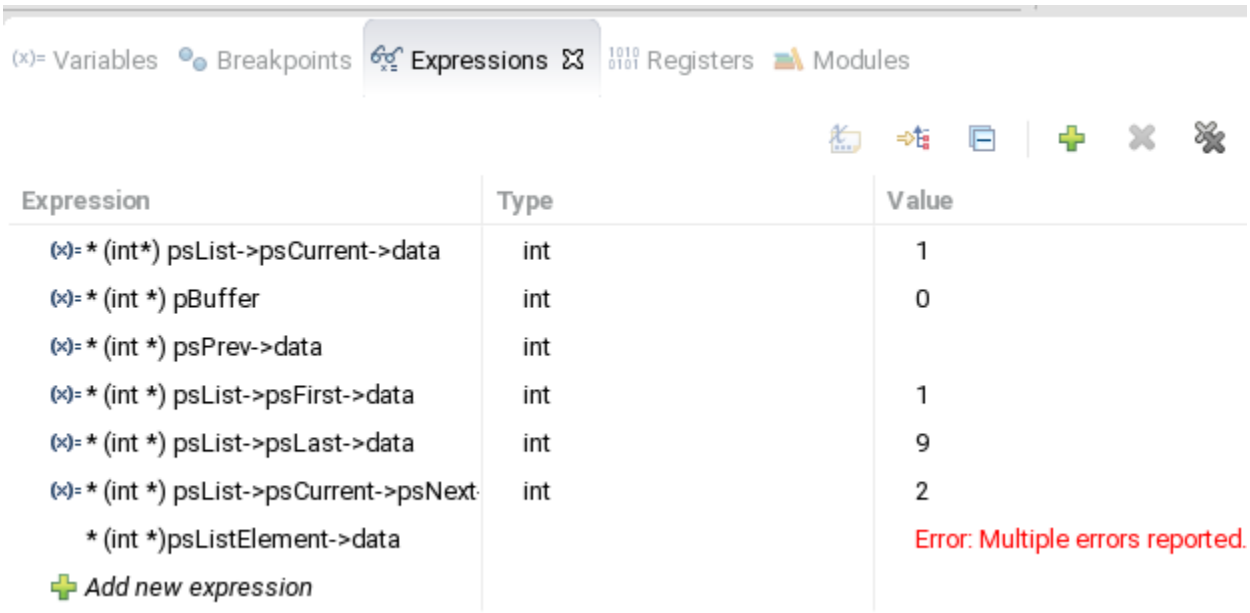
Part A: Here is a list of the functions that must be completed for Part A and the order in which I recommend you implement each function:

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: Complete the entire assignment. 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:



If you have Makefile problems, see me early. I do not want to see more code in your project than dlist.h, a single function lstCreate in dlist.c, and a single call to lstCreate in dlistdriver.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**.