CS 2110

Timed Lab 6

Due Date and Time

Day: Wednesday, November 29th
Time: Before the end of your assigned lab section

Policy

Submission

TURN IN THIS ASSIGNMENT ELECTRONICALLY USING T-SQUARE

SUBMISSIONS WHICH ARE LATE WILL NOT BE ACCEPTED.

EMAIL SUBMISSIONS WILL NOT BE ACCEPTED UNDER ANY CIRCUMSTANCES!

IF YOU FORGET TO HIT THE SUBMIT BUTTON YOU WILL GET A ZERO.

Questions

If you are unsure of what questions mean, the TA's will clarify them to the best of their ability. We will not be able to answer any questions about how to reach a solution to the timed lab questions. You should know how by now!

What's Allowed

- The assignment files
- Your previous homework and lab submissions
- Your mind
- Blank paper for scratch work

What's Not Allowed

- The Internet (except the T-Square Assignment page to submit)
- Any resource on T-Square that is not given in the assignment.
- Textbook or notes on paper or saved on your computer.
- Dropbox (If your hard drive crashes we will let you retake it).
- Email/IM
- Contact in any form with any other person besides TAs

if you have any questions on what you may not use then assume you can't use it and ask a TA.

Other Restrictions

1. **You may not leave the classroom until we have verified that you have submitted the lab. If you leave the classroom without submitting you will receive a zero.**

2. **YOU MUST SUBMIT BY THE END OF YOUR LAB PERIOD**. Bear in mind that the clock on your computer may be a few minutes slow. You are supposed to have a full class period to work, and we are letting you use the 10 minutes between classes to make sure you have submitted your work. **WE WILL NOT ACCEPT LATE SUBMISSIONS**, be they 1 second or 1 hour late.

3. The timed lab has been configured to accept one submission. If you accidentally submit or submit the wrong version flag one of the TAs and we will reopen submission for you.

Violations

Failure to follow these rules will be in violation of the Georgia Tech Honor Code **AND YOU WILL RECEIVE A ZERO** and you will be reported to Bill and the Office of Student Integrity.

We take cheating and using of unauthorized resources **VERY SERIOUSLY** and you will be in serious trouble if you are caught.

Remember

1. **There is partial credit given, and some of it is just following the directions.**

2. **We allow you to use your homework assignment.**

3. **Please don't get stressed out during a timed lab. You have plenty of time; however, use your time effectively**

4. **Again, remember: Don't get stressed. Partial credit will be given for things you have done correctly. Do the best you can!**

5. **If you don't know something at least TRY. Do not just walk out of the lab or submit an empty file. Partial credit!**

6. **Remember what you can and can't use. If you don't know, then don't use it and ask a TA if you can use it. If we catch you with unauthorized resources we will give you a zero, so better to be safe than sorry.**

The Assignment

Overview

In this lab, you will be implemnting some functions for an **Array List** data structure. An Array List and a Linked List are both concrete implementations of the List abstract data type. As a result, they will offer the same services, but provide those services in a slightly different way (i.e the inner workings of the list implementations will be different). See the section below for some details on how an Array List works compared to a Linked List.

Array List

As you remember from your homework assignment, a Linked List works by creating a node for each piece of data, and linking these nodes together with next pointers. While this makes it very quick to add or remove elements from the front or back of the list (which is very useful in some applications), accessing elements in the middle of the list is very slow because to get to an element, you must iterate over all previous elements. Both of these properties are a result of the list being backed by a linked structure.

To solve this problem of slow access to the middle elements, an Array List uses an array as its backing structure, rather than some kind of linked structure. This allows it to access elements in the middle of the list in constant time, at the cost of some overhead when adding and removing.

Essentially, an Array List works like a resizeable array. It will have a length **N** array as its backing structure, and as elements are added, it will place them into this length **N** array. Note that the size of the Array List (i.e the number of elements actually stored within it) will usually be less than **N**. This just means that it is not using all of the space in the backing array. Elements will all be stored at the front of the backing array (i.e starting at index 0) without gaps in between them (e.g elements 2 and 3 will be right next to eachother in the backing array, and will have indices 2 and 3 respectively in the backing array).

It can continue adding without a problem untiil **S**, the size of the list (i.e the number of elements it is storing) is equal to **N**. This signifies that the array list is using up all of the space in its backing array. When this is happening, it means that on the next add attempt, there will not be enough space to store the new element. To rectify this, the Array List will create a new, bigger array, copy its contents into that bigger array, and start using the bigger array instead of the old one it was using (making **N** bigger). The new element can then be added to this bigger array without a problem, as there is now enough space.

As elements are removed from the list, they are removed from the backing array as well. However, the backing array should not have any empty space between elements. See the example below, where we want to remove element 2 from the list. **You will not implement remove in this timed lab, but this example is useful for seeing the structure of the array list.**

START:

Backing Array: [0, 1, 2, 3, 4, 5, EMPTY, EMPTY, EMPTY, EMPTY], **Size**: 6, **N**: 10

CORRECT AFTER

Backing Array: [0, 1, 3, 4, 5, EMPTY, EMPTY, EMPTY, EMPTY, EMPTY], **Size**: 5, **N**: 10

INCORRECT AFTER (check out the hints section for why this is *very* wrong)

Backing Array: [0, 1, NULL, 3, 4, 5, NULL, NULL, NULL, NULL], **Size**: 5, **N**: 10

Specifics

You are to provide implementations for all of the incomplete methods in array_list.c. You will have to dynamically allocate memory for your implemtation to work correctly. For any functions that cause memory leaks, you will not receive points. Check out array_list.h to see what you have been provided with.

Do not use calloc or realloc in your implementation!!!

Hints and Additional Implementation Details

- Note that the contents of the list are considered to be all elements in the backing array with an index **strictly less than** size (**S**). With this in mind, consider why the bad example for removing is so wrong. Also, does this tell you anything about the next available/unused slot in your backing array? If it does, this will probably simplify your add logic.

- Your implementation should support adding NULL data to the list (i.e, NULL is a valid element in the list). Again, consider why the incorrect removal example is wrong because of this.

- The initial size of the backing array (INIT_BACKING_SIZE) will always be a positive integer (pinky promise), so don't worry about input validation.

- You will not be graded on the time complexity of your solution (i.e we will not grade you on big O).

- Think back to all those times you had to make a resizeable array in 1331 :)

- Notice that <string.h> is included. That means you can use memcpy!

Deliverables

1. tl6.tar.gz

use the command:

make submit

To generate the file you have to submit

You may submit only the files listed above. We will not accept any internet links we want the files above and only these files!

Check over your submission after you submit it. If you submit the wrong file and leave the lab I will not be happy and we will grade what you submit so please check over what you submitted after you submit it!

Have fun and good luck!

Love,

Sanjay, Patrick, and Jim