CS 2110 Timed Lab 6: Dynamic Memory Allocation

Hi Patrick, Sanjay Sood, Preston Olds

Spring 2018

Contents

1 Before You Begin

Please take the time to read the entire document before starting the assignment. We have made some important updates, and it is your responsibility to follow the instructions and rules.

2 Timed Lab Rules - Please Read

2.1 General Rules

1. You are allowed to submit this timed lab starting at the moment the assignment is released, until you are checked off by your TA as you leave the recitation classroom. Gradescope submissions will remain open until 6 pm - but you are not allowed to submit after you leave the recitation classroom under any circumstances. **Submitting or resubmitting the assignment after you leave the classroom is a violation of the honor code - doing so will automatically incur a zero on the assignment and might be referred to the Office of Student Integrity.**
2. Make sure to give your TA your Buzzcard before beginning the Timed Lab, and to pick it up and get checked off before you leave. **Students who leave the recitation classroom without getting checked off will receive a zero.**
3. Although you may ask TAs for clarification, you are ultimately responsible for what you submit. **The information provided in this Timed Lab document takes precedence.** If in doubt, please make sure to indicate any conflicting information to your TAs.
4. Resources you are allowed to use during the timed lab:
    - Assignment files
    - Previous homework and lab submissions
    - Your mind
    - Blank paper for scratch work (please ask for permission from your TAs if you want to take paper from your bag during the Timed Lab)
5. Resources you are **NOT** allowed to use:
    - The Internet (except for submissions)
    - Any resources that are not given in the assignment
    - Textbook or notes on paper or saved on your computer
    - Email/messaging
    - Contact in any form with any other person besides TAs
6. **Before you start, make sure to close every application on your computer.** Banned resources, if found to be open during the Timed Lab period, will be considered a violation of the Timed Lab rules.
7. We reserve the right to monitor the classroom during the Timed Lab period using cameras, packet capture software, and other means.

2.2 Submission Rules

1. Follow the guidelines under the Deliverables section.

2. You are also responsible for ensuring that what you turned in is what you meant to turn in. After submitting you should be sure to download your submission into a brand new folder and test if it works. No excuses if you submit the wrong files, what you turn in is what we grade. In addition, your assignment must be turned in via Canvas/Gradescope. Under no circumstances whatsoever we will accept any email submission of an assignment. Note: if you were granted an extension you will still turn in the assignment over Canvas/Gradescope.
3. Do not submit links to files. We will not grade assignments submitted this way as it is easy to change the files after the submission period ends.

2.3 Is collaboration allowed?

Absolutely NOT. No collaboration is allowed for timed labs.

3 Overview

For this assignment, you will be creating a "list array" in C. To create this data structure, you will be implementing the following functions in `list_array.c`:

1. `create_node`
2. `create_list`
3. `set_list`
4. `get_list`
5. `destroy_list_array`

We have provided you with `list_array.h` (**which you should not modify**), a `Makefile`, `verify.sh` for checking if `list_array.h` has been modified, and `test.c` where you can design and run your own tests.

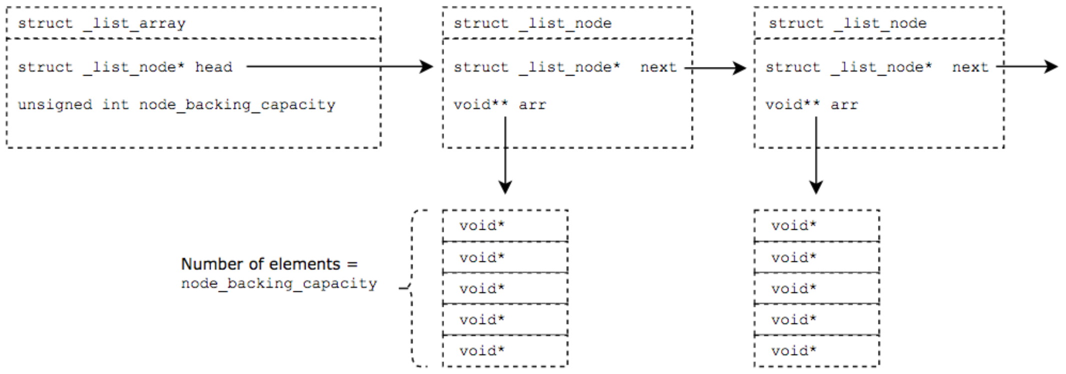
The **only** file you must edit and submit is `list_array.c`, but editing `test.c` could also be helpful.

PLEASE DOUBLE CHECK YOUR SUBMISSION BEFORE YOU LEAVE!

4 Instructions

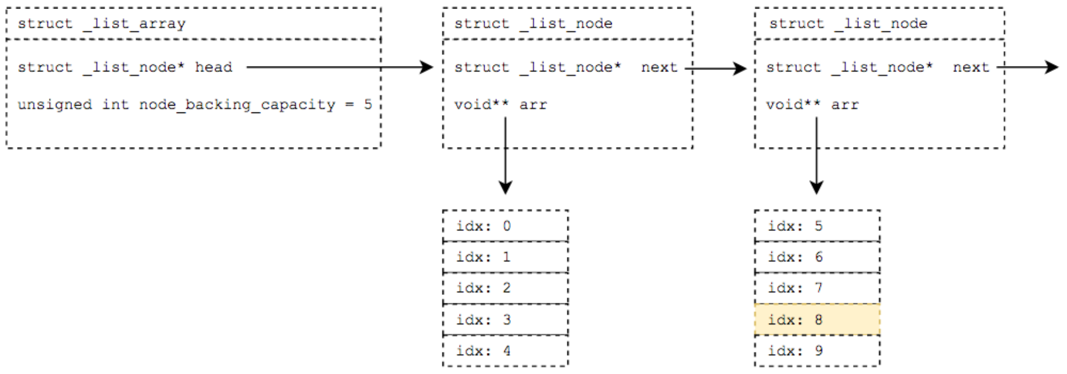
4.1 List Array

Consider the following diagram, illustrating an example of this data structure:



The `_list_array` struct has a `head` and `node_backing_capacity`. The `head` points to the first `_list_node` in the list array, and the `node_backing_capacity` indicates the number of elements in the backing array for each node. Each `_list_node` has a pointer to the `next` node and a pointer to a backing array of data.

Data in the `_list_array` is indexed like a traditional array: For example, say we have a `_list_array` with a `node_backing_capacity` of `5` and want to set data at index `8`. The data will be set in the fourth slot of the backing array pointed to by the second `_list_node`.



4.1.1 create_node

This function takes in a `backing_capacity` and creates a `list_node`, returning a pointer to the newly created node (or `NULL` if `malloc` fails). You'll want to ensure memory in the backing array is cleared before returning the `list_node`.

If `malloc` fails, be sure to clean up any partially allocated stuctures!

4.1.2 create_list

This function takes in a `backing_capacity` and creates a `list_array` with one node, pointed to by `head`, returning a pointer to the newly created list (or `NULL` if `malloc` fails). This `backing_capacity` will be the same for all nodes in the `list_array`.

If `malloc` fails, be sure to clean up any partially allocated stuctures!

4.1.3 set_list

This function assigns `data` to the specified `index`. If the `index` is out of bounds of any of the nodes, you must add more nodes to the `list_array` until it is within bounds.

4.1.4 get_list

This function gets `data` at the specified `index`.

4.1.5 destroy_list_array

This function destroys and frees all data associated with the `list_array`.

4.2 Restrictions

- Your code must not crash, run infinitely, or produce any memory leaks.
- Your code must compile with the Makefile we have provided.
- Your code must be optimal in terms of memory allocation operations (i.e. allocating more memory than needed is not optimal)

5 Testing Your Work

We have provided you with a file called `test.c` with which to test your code. Note that it contains no test cases. You must write your own.

We have provided a `Makefile` for this assignment that will build your project. Here are the commands you should be using:

1. To run the tests in `test.c`: `make run-test`
2. To debug your code using gdb: `make run-gdb`
3. To run your code with valgrind: `make run-valgrind`

Once done, you can test your assignment using the autograder

1. copy `tl6.tar.gz`, `autograde.tar`, `autograde-Makefile` to a new directory
2. change to that new directory and run `make -f autograde-Makefile` to run the grader

DO NOT USE THE AUTOGRADER TO DEBUG YOUR CODE. Running the autograder should be the last step before submitting, and you should debug by writing your own tests and running them instead of relying on the autograder.

The autograder is probably too complex to debug with during the time constraints of the testing period. You have been warned.

6 Rubric

The output of the autograder is an approximation of your score on this timed lab. It is a tool provided to students so that you can evaluate how much of the assignment expectations your submission fulfills. However, **we reserve the right to run additional tests, fewer tests, different tests, or change individual tests -** your final score will be determined by your instructors and no guarantee of tester output correlation is given.

7 Deliverables

When you are finished, run `make submit` and submit the following files to Autolab:

- `tl6.tar.gz`