CS 2110 Timed Lab 5
Memory Allocation in C

Shannon Kek, Josh Viszlai, Ausbin Adams, Cem Gokmen, Vivian De Sa Thiebaut

November 7, 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 7:15 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 result in you being 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 or submit after 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 turn 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. There are 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 Gradescope. Under no circumstances whatsoever will we accept any email submission of an assignment. Note: if you were granted an extension, you will still turn in the assignment over 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

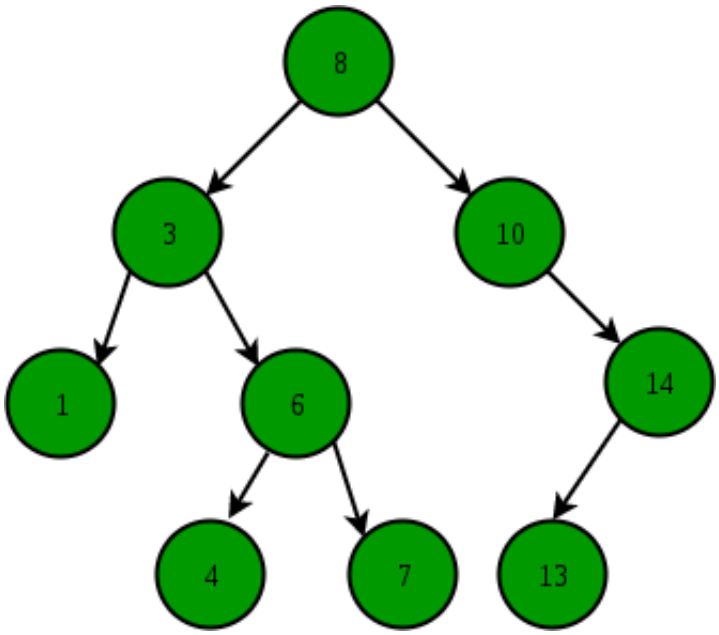
For this assignment, you will implement a binary search tree using memory allocation in C. There are four methods to write: `bst_add`, `bst_contains`, and `bst_destroy`.

Pseudocode and detailed descriptions have been defined for each method inside of the javadocs in `binary_search_tree.c`.

3.1 Binary Search Trees

A binary search tree is a node-based data structure with the following properties:

- each node has a maximum of two children.
- the left subtree of a node contains only nodes with values less than the value of the original node.
- the right subtree of a node contains only nodes with values greater than the value of the original node.
- the left and right subtrees must also be binary search trees.



In the example bst, you will notice that all children to the left of a parent node are of smaller value than the parent itself, and the children to the right of a parent node are of greater value than the parent itself. The basic process for memory allocation for a bst is not unlike the process used with linked lists, so feel free to reference Homework 8.

4 Instructions

You will be editing ONLY `binary_search_tree.c`. We have defined the `bst_node` struct in the `binary_search_tree.h` file. Instructions for each method have been provided in the javadocs for each method in `binary_search_tree.c`.

All `#include` directives have been included for you. Do not add any more includes. Doing so will result in point deductions that the tester will not reflect.

4.1 Dependencies

There are some dependencies for running the tests.

If you are using Docker, re-run `cs2110docker.sh` to stop the old container, download updates to the image (which include these dependencies), and restart the container.

If you are using a VM, run

```
$ sudo apt install build-essential gdb valgrind pkg-config check
```

5 Testing Your Work

5.1 Debugging in C

We have provided you with a test suite to check your linked list that you can run locally on your very own personal computer. You can run these using the Makefile.

The given test cases are the same as the ones on Gradescope. Note that you will pass some of these tests by default. Your grade on Gradescope may not necessarily be your final grade as we reserve the right to adjust the weighting. However, if you pass all the tests and have no memory leaks according to valgrind, you can rest assured that you will get 100 as long as you did not cheat or hard code in values.

You will not receive credit for any tests you pass where valgrind detects memory leaks or memory errors. Gradescope will run valgrind on your submission, but you may also run the tester locally with valgrind for ease of use.

Printing out the contents of your structures can't catch all logical and memory errors, which is why we also require you run your code through valgrind.

We certainly will be checking for memory leaks by using valgrind, so if you learn how to use it, you'll catch any memory errors before we do.

Your code must not crash, run infinitely, nor generate memory leaks/errors.
Any test we run for which valgrind reports a memory leak or memory error will receive no credit.

If you need help with debugging, there is a C debugger called gdb that will help point out problems. See instructions in the Makefile section for running an individual test with gdb.

5.2 Makefile

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

1. To clean your working directory (use this command instead of manually deleting the .o files): `make clean`
2. To run the tests without valgrind or gdb: `make run-tests`
3. To run your tests with valgrind: `make run-valgrind`
4. To debug a specific test with valgrind: `make TEST=test_name run-valgrind`
5. To debug a specific test using gdb: `make TEST=test_name run-gdb`

   Then, at the `(gdb)` prompt:

   (a) Set some breakpoints (if you need to — for stepping through your code you would, but you wouldn't if you just want to see where your code is segfaulting) with `b suites/bst_suite.c:420`, or `b binary_search_tree.c:69`, or wherever you want to set a breakpoint

   (b) Run the test with `run`

   (c) If you set breakpoints: you can step line-by-line (including into function calls) with `s` or step over function calls with `n`

   (d) If your code segfaults, you can run `bt` to see a stack trace

   For more information on gdb, please see one of the many tutorials linked above.

To get an individual test name, you can look at the output produced by the tester. For example, the following failed test is `test_bst_add_left_simple`:

```
suites/bst_suite.c:38:F:test_bst_add_left_simple:test_bst_add_left_simple:0: Assertion failed...
                          ^^^^^^^^^^^^^^^^^^^^
```

Beware that segfaulting tests will show the line number of the last test assertion made before the segfault, not the segfaulting line number itself. This is a limitation of the testing library we use. To see what line in your code (or in the tests) is segfaulting, follow the "To debug a specific test using gdb" instructions above.

6 Rubric

NOTE: For the `bst_contains` function, you must pass all the test cases in order to get credit for them.

The output of the Gradescope autograder is an approximation of your score on this assignment. The tool is provided so you can evaluate whether your submission fulfills the assignment expectations.

However, we reserve the right to run additional tests, fewer tests, different tests, or potentially change individual tests – your final score will be determined by your instructors, and there is no guarantee your score will correlate with the tester output.

7 Deliverables

Please upload the following file to the assignment on Gradescope:

1. `binary_search_tree.c`

Your files must compile with our Makefile, which means it must compile with the following gcc flags:

```
-std=c99 -pedantic -Wall -Werror -Wextra -Wstrict-prototypes -Wold-style-definition
```

Please check your submission after you have uploaded it to gradescope to ensure you have submitted the correct file. **Do NOT upload an archive; upload the file individually.**

Be sure to check your Gradescope test score before you leave the room.