CS 2110 Homework 8
Linked List

Cliff Panos, Jim Harris, Skippy, Bharat Srirangam, Joshua Viszlai

Fall 2018

Contents

1 Overview

In this assignment, you will implement a list data structure whose underlying implementation is a singly-linked list. If you have taken CS 1332 (don't worry if you haven't), then you should be familiar with this data structure. Your linked list will be able to add and remove `person` structs, query the data that it stores, and even perform some additional library functions such as `reverse` and `copy`.

`https://en.wikipedia.org/wiki/Linked_list`

2 Instructions

You have been given one C file, **list.c**, in which to implement the data structure. Implement all functions in this file. Each function has a block comment that describes exactly what it should do.

Be sure not to modify any other files. Doing so will result in point deductions that the tester will not reflect. You may feel free to modify the files for the tester however you like though for debugging purposes (namely **list_suite.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.

2.1 Dependencies

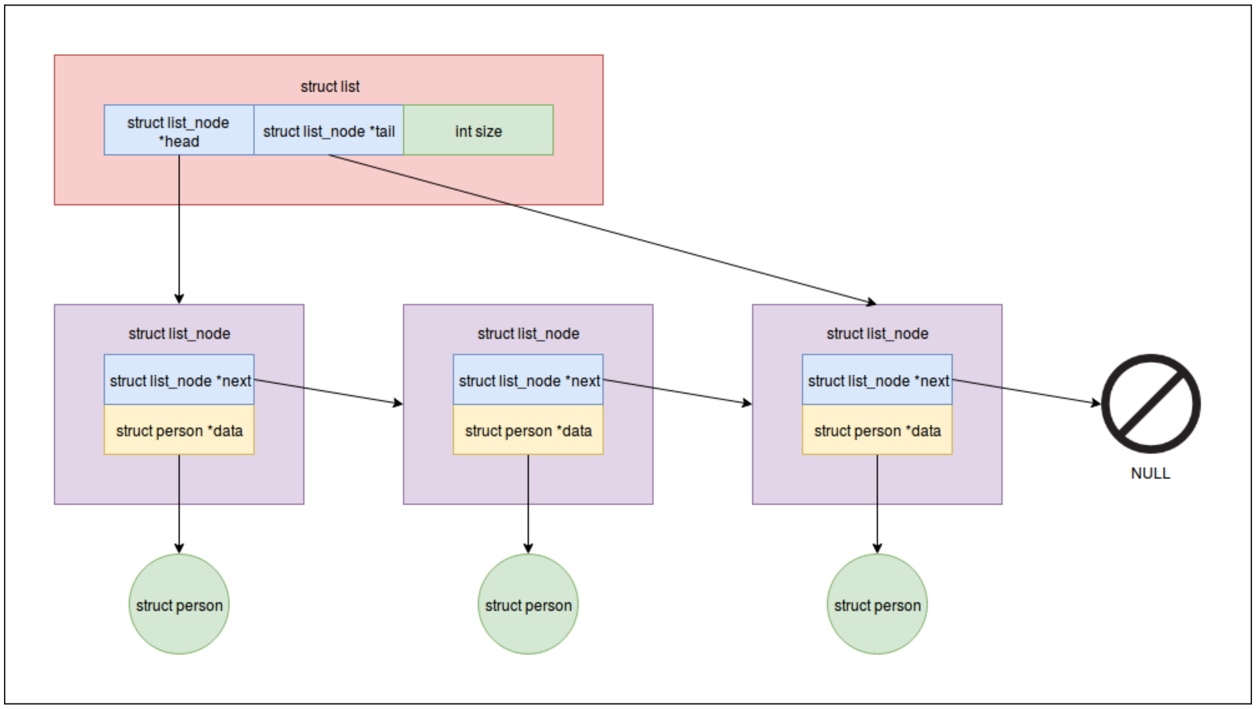
There are some dependencies for running the tests.

If you are using Docker, re-run `cs2110docker.sh` to stop the old container, download updates 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
```

2.2 Linked List Implementation



Refer to **list.c** Each `list` structure will contain its `size` as well as `head` and `tail` pointers that point to the nodes at either end of itself. Each `list_node` structure contains a `next` pointer and a `data` pointer; the `data` pointer references what is being stored, and the `next` pointer references the next `list_node` along the chain. **Do not** change the definition of the `list_node` struct. **Do not** use sentinel or dummy nodes in your list.

3 Deliverables

Submit ONLY submission.tar.gz, which is generated by running the command:

```
make submit
```

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

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

All non-compiling homeworks will receive a zero.

Please check your submission after you have uploaded it to gradescope to ensure you have submitted the correct file.

4 Background

4.1 Linked Lists, Continued

A linked list is an ordered list of nodes. Each `node` in the `list` is comprised of two consecutive values in memory: a pointer to the `next` node and a pointer to the `data` being stored by that node. Since both of these values are addresses, each value is 64 bits or 8 bytes on a 64-bit architecture.

Consider an example list with a `head` (starting `node`) at address `x4000`.

```
 Address | Contents | Comments
---------+----------+--------------
   x4000 |    x4010 | Node 1: next
   x4008 |    x0035 | Node 1: data
   x4010 |    x4030 | Node 2: next
   x4018 |    x0101 | Node 2: data
   x4020 |    x0000 | Node 4: next
   x4028 |    x9040 | Node 4: data
   x4030 |    x4020 | Node 3: next
   x4038 |    x7000 | Node 3: data
```

Observe that `Node 1` points to `Node 2` (at address `x4010`). `Node 2` points to `Node 3` (at address `x4030`). `Node 3` points to `Node 4` (at address `x4020`). Finally, `Node 4` points to address `x0000` (i.e. `NULL`), signaling that this node is the `tail` node at the end of the list!

Also notice that consecutive nodes need not be next to each other in memory. This quality is what makes it easy to dynamically allocate additional nodes without needing to reallocate the entire list when we need more space.

4.2 Data Structure Design

The design of this list library is such that the person using your library does not have to deal with the details of the implementation of your library (i.e. the `list_node` struct used to implement the `list`). None of these functions return a `struct list_node` nor do any of these functions take in a `struct list_node`. It is your responsibility to create the nodes and add them into the list yourself, not the user. When the user is done with the list, they will call empty_list which removes all of the person structs from the list and frees the nodes that contained pointers to the `person` structs. Finally, you must free the list structure yourself so that no memory is leaked by your function.

4.3 Testing & Debugging

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.

Your process for doing this homework should be to write one function at a time and make sure all of the tests pass for that function. Then, you can make sure that you do not have any memory leaks using valgrind. It doesn't pay to run valgrind on tests that you haven't passed yet. Further down, there are instructions for running valgrind on an individual test under the Makefile section, as well as how to run it on all of your tests.

The given test cases are the same as the ones on Gradescope. Note that you will pass some of these tests my 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. 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.

Here are tutorials on valgrind:
`http://cs.ecs.baylor.edu/~donahoo/tools/valgrind/`
`http://valgrind.org/docs/manual/quick-start.html`

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 further down for running an individual test with gdb.

If your code generates a segmentation fault then you should first run gdb before asking questions. We will not look at your code to find your segmentation fault. This is why gdb was written to help you find your segmentation fault yourself.
Here are some tutorials on gdb:
`https://www.cs.cmu.edu/~gilpin/tutorial/`
`http://www.cs.yale.edu/homes/aspnes/pinewiki/C%282f%29Debugging.html`
`http://heather.cs.ucdavis.edu/~matloff/UnixAndC/CLanguage/Debug.html`
`http://heather.cs.ucdavis.edu/~matloff/debug.html`
`http://www.delorie.com/gnu/docs/gdb/gdb_toc.html`

Getting good at debugging will make your life with C that much easier.

4.4 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 in test.c: `make run-tests`
3. To debug your code using gdb: `make run-gdb`
4. To run your code with valgrind: `make 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/list_suite.c:420`, or `b list.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.

6. To run a specific test using valgrind: `make TEST=test_name run-valgrind`

5 Rules and Regulations

5.1 General Rules

1. Starting with the assembly homeworks, any code you write must be meaningfully commented. You should comment your code in terms of the algorithm you are implementing; we all know what each line of code does.
2. Although you may ask TAs for clarification, you are ultimately responsible for what you submit. This means that (in the case of demos) you should come prepared to explain to the TA how any piece of code you submitted works, even if you copied it from the book or read about it on the internet.
3. Please read the assignment in its entirety before asking questions.
4. Please start assignments early, and ask for help early. Do not email us the night the assignment is due with questions.
5. If you find any problems with the assignment it would be greatly appreciated if you reported them to the author (which can be found at the top of the assignment). Announcements will be posted if the assignment changes.

5.2 Submission Conventions

1. All files you submit for assignments in this course should have your name at the top of the file as a comment for any source code file, and somewhere in the file, near the top, for other files unless otherwise noted.
2. When preparing your submission you may either submit the files individually to Canvas/Gradescope or you may submit an archive (zip or tar.gz only please) of the files. You can create an archive by right clicking on files and selecting the appropriate compress option on your system. Both ways (uploading raw files or an archive) are exactly equivalent, so choose whichever is most convenient for you.
3. Do not submit compiled files, that is .class files for Java code and .o files for C code. Only submit the files we ask for in the assignment.
4. Do not submit links to files. The autograder does not understand it, and we will not manually grade assignments submitted this way as it is easy to change the files after the submission period ends.

5.3 Submission Guidelines

1. You are responsible for turning in assignments on time. This includes allowing for unforeseen circumstances. If you have an emergency let us know **IN ADVANCE** of the due time supplying documentation (i.e. note from the dean, doctor's note, etc). Extensions will only be granted to those who contact us in advance of the deadline and no extensions will be made after the due date.
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. There is a 6-hour grace period added to all assignments. You may submit your assignment without penalty up until 11:55PM, or with 25% penalty up until 5:55AM. So what you should take from this is not to start assignments on the last day and plan to submit right at 11:54AM. You alone are responsible for submitting your homework before the grace period begins or ends; neither Canvas/Gradescope, nor

your flaky internet are to blame if you are unable to submit because you banked on your computer working up until 11:54PM. The penalty for submitting during the grace period (25%) or after (no credit) is non-negotiable.

5.4 Syllabus Excerpt on Academic Misconduct

Academic misconduct is taken very seriously in this class. Quizzes, timed labs and the final examination are individual work.

Homework assignments are collaborative, In addition many if not all homework assignments will be evaluated via demo or code review. During this evaluation, you will be expected to be able to explain every aspect of your submission. Homework assignments will also be examined using computer programs to find evidence of unauthorized collaboration.

What is unauthorized collaboration? Each individual programming assignment should be coded by you. You may work with others, but each student should be turning in their own version of the assignment. Submissions that are essentially identical will receive a zero and will be sent to the Dean of Students' Office of Academic Integrity. Submissions that are copies that have been superficially modified to conceal that they are copies are also considered unauthorized collaboration.

You are expressly forbidden to supply a copy of your homework to another student via electronic means. This includes simply e-mailing it to them so they can look at it. If you supply an electronic copy of your homework to another student and they are charged with copying, you will also be charged. This includes storing your code on any site which would allow other parties to obtain your code such as but not limited to public repositories (Github), pastebin, etc. If you would like to use version control, use github.gatech.edu

5.5 Is collaboration allowed?

Collaboration is allowed on a high level, meaning that you may discuss design points and concepts relevant to the homework with your peers, share algorithms and pseudo-code, as well as help each other debug code. What you shouldn't be doing, however, is pair programming where you collaborate with each other on a single instance of the code. Furthermore, sending an electronic copy of your homework to another student for them to look at and figure out what is wrong with their code is not an acceptable way to help them, because it is frequently the case that the recipient will simply modify the code and submit it as their own. Consider instead using a screen-sharing collaboration app, such as `http://webex.gatech.edu/`, to help someone with debugging if you're not in the same room.

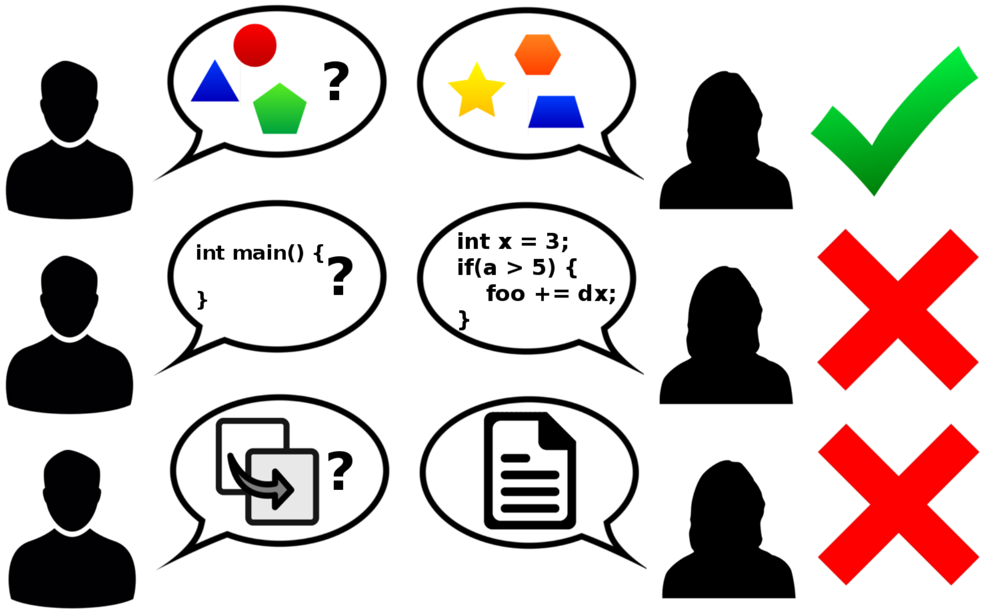


Figure 1: Collaboration rules, explained colorfully