

Project 5: Linked List

Gibran Essa, Siddhant Singh, Elton Pinto, Martin Heberling, Colin Sergi

Fall 2020

Contents

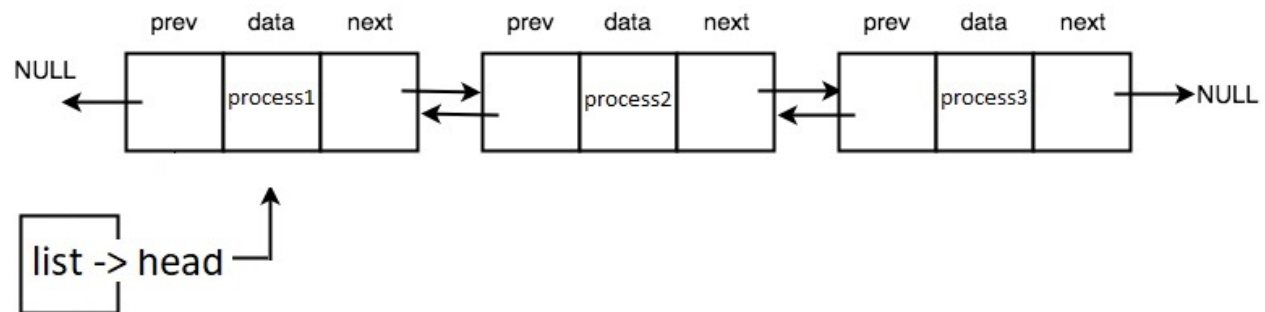
1	Overview	2
2	Linked List Implementation	2
3	Instructions	2
4	Checker/Makefile Usage	3
4.1	Testing & Debugging	3
4.2	Makefile	4
5	Deliverables	5
6	Demos	5
7	Rules and Regulations	5
7.1	General Rules	5
7.2	Submission Conventions	6
7.3	Submission Guidelines	6
7.4	Syllabus Excerpt on Academic Misconduct	6
7.5	Is collaboration allowed?	7

1 Overview

In this assignment, you will be implementing a list data structure whose underlying implementation is a doubly-linked list. If you have taken CS 1331/32 (don't worry if you haven't taken 32), then you should be familiar with this data structure.

2 Linked List Implementation

We are implementing a doubly-linked list. You are given a `list` which has a `head` pointer pointing to the first node in the list. Each `node` contains a `next` pointer to the next node in the list, a `prev` pointer to the previous node in the list and a `data` pointer to a `Process`. Refer to the following diagram for a visual representation to help you out when implementing your linked list!



3 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 and there is a synopsis included below.

- `Process *create_Process(char *, int)`: Takes in a string and an int. Create a new process with name equal to the passed in string, and priority equal to the passed in int. Assign a new PID one higher than the previous PID, with the first one being 0.
- `ListNode *create_node(Process *)`: Takes in a Process, creates a ListNode containing that Process as its data, and returns the ListNode.
- `List *create_list()`: creates an empty list and returns it
- `void push_front(List *, Process *)`: adds a ListNode containing the given process as data to the front of the List (the beginning of the list, so that this node becomes the new head of the list).
- `void push_back(List *, Process *)`: adds a ListNode containing the given process as data to the back of the List (the end of the list, so that this node becomes the last node in the list)
- `int remove(List *, Process **, int)`: finds and removes the Process whose PID is equal to the third input. The Process data will be returned via the Process** parameter and the ListNode containing the Process will be removed completely.
- `void destroy_Process(Process *)`: completely destroys all data in the Process and the Process itself.

- `void destroy(List *)`: destroys the list itself. This will destroy the list, all nodes within the list, and the Processes within the nodes.
- `int copy_Process(Process *, Process **)`: creates a deep copy of a Process and returns the deep copy through the Process ** parameter.
- `List *copy_list(List *)`: creates a deep copy of the list. This includes creating a new list, new nodes, and new Processes in the nodes.
- `int compare_PID(Process *, Process *)`: Compares the two Processes based off of PID.
- `int compare_name(Process *, Process *)`: Compares two Processes based on their name
- `int swap_nodes(ListNode *, ListNode *, List *)`: Swaps the two nodes in the list in place.
- `int sort(List *, int (*compare_func)(Process *, Process *b))`: Utilizes the compare function pointer to sort the list (in place) in ascending order.
- `int make_idle(Process *)`: Append " (idle)" to the end of the name of the passed in process. You must use `realloc` and any functions you may need from `string.h`.
- `int make_active(Process *)`: See if the process name ends with " (idle)" If it does, remove it. You must use `realloc` and any functions you may need from `string.h`.
- `int map_inplace(List *, int (*map_func)(Process *))`: Calls the map function pointer on each Process in the list.
- `Process **list_to_array(List *, int)`: Converts the linked list data structure to an array of pointers to Processes while removing the linked list structure itself so that only the process pointers remain.

You should not be leaking memory in any of the functions. For any functions utilizing malloc, if malloc returns null, return either 0 or null (based on the function header) and ensure that no memory is leaked.

Be sure not to modify any other files. Doing so may result in point deductions that the tester will not reflect.

4 Checker/Makefile Usage

4.1 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.

Note: There is a file called `test_utils.o` that contains some functions that the test suite needs. We are not providing you the source code for this, so make sure not to accidentally delete this file as you will need to redownload the assignment. Also keep in mind that this file does not have debugging symbols so you will not be able to step into it with `gdb` (which will be discussed shortly).

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 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.

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 in the Makefile section 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.

4.2 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/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.

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

```
suites/list_suite.c:906:F:test_list_size_empty:test_list_size_empty: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.

Note: The checker may not reflect your actual grade on this assignment. We reserve the right to update the checker as we see fit when grading.

5 Deliverables

Submit ONLY `list.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`

All non-compiling homeworks will receive a zero. If you want to avoid this, do not run gcc manually; use the Makefile as described below.

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

6 Demos

This project will not be demoed, as a special exception. There is no time at the end of the semester after this project's due date to demo it, so we will not hold demos.

The project grade is entirely composed of the autograder grade.

7 Rules and Regulations

7.1 General Rules

1. 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.
2. Please read the assignment in its entirety before asking questions.

3. Please start assignments early, and ask for help early. Do not email us a few hours before the assignment is due with questions.
4. 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.

7.2 Submission Conventions

1. 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. Both ways (uploading raw files or an archive) are exactly equivalent, so choose whichever is most convenient for you.
2. 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.

7.3 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. You are additionally responsible for ensuring that the collaborators list you have provided in your submission is accurate.
4. Projects turned in late receive partial credit within the first 48 hours. We will take off 30% of the points for a project submitted between 0 and 24 hours late, and we will take off 50% of the points for a project submitted between 24 and 48 hours late. We will not accept projects turned in over 48 hours late. This late policy is also in the syllabus.
5. You alone are responsible for submitting your project before the assignment is due; 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 the deadline.

7.4 Syllabus Excerpt on Academic Misconduct

Academic misconduct is taken very seriously in this class.

1. Students are expected to have read and agreed to the Georgia Tech Honor Code, see <http://osi.gatech.edu/content/honor-code>.
2. Suspected plagiarism will be reported to the Division of Student Life office. It will be prosecuted to the full extent of Institute policies.
3. A student must submit an assignment or project as his/her own work (this is what is expected of the students).

4. Using code from GitHub, via Googling, from Stack Overflow, etc., is plagiarism and is not permitted. Do not publish your assignments on public repositories (i.e., accessible to other students). This is also a punishable offense.
5. Although discussion among the students through piazza and other means are encouraged, the sharing of work is plagiarism. If you are not sure about it, please ask a TA or stop by the instructor's office during the office hours.
6. You must list any student with whom you have collaborated in your submission. Failure to list collaborators is a punishable offense.
7. TAs and Instructor determine whether the project is plagiarized. Trust us, it is really easy to determine this....

7.5 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 should not 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 Bluejeans, to help someone with debugging if you're not in the same room.

Any of your peers with whom you collaborate in the above fashion must be properly added to your **collaborators.txt** file. Collaborating with another student without listing that student as a collaborator is considered plagiarism.

