Project 5: Address Book With Hash Table

Junchen Lu, Varun Mehrotra, Udit Subramanya, Calvin Khiddee-Wu, Sankaet Cheemalamarri

Fall 2022

Contents

1 Overview

In this assignment, you will be implementing an address book using a hash table.

2 Hash Table Implementation

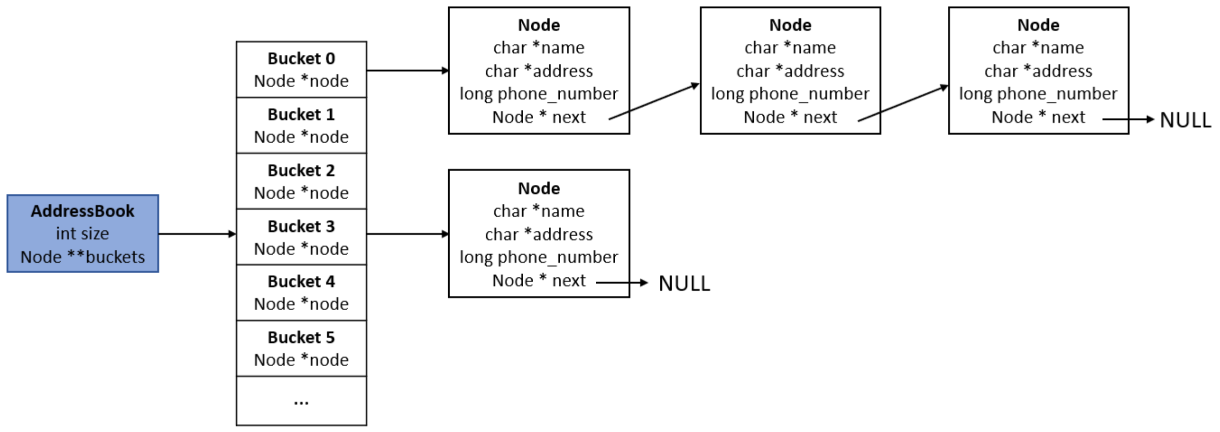
2.1 What is a hash table?

The hash table you will be implementing is based on an array of linked lists. A hash table stores its data in the form of a (key, value) combination. When provided with a key, the key is passed through a hash function that converts the key into an integer corresponding to a "bucket" (i.e. the hash table maps the key to a "bucket"). When multiple keys are mapped to the same bucket, they are stored in a linked list. **We usually resize the base array of the hash table to ensure efficiency. However, since this is not a data structure course, you do not have to worry about the load factor and resizing the base array.**

2.2 The address book

In your address book, the names of the residents will be the keys and their names, addresses and phone numbers are stored as values.

Here is a visual representation of the hash table:



You are given an `AddressBook` which contains a pointer `buckets` to an array of `Node*`. `Bucket[index]` is just a name that we give to each element in the array. There is no `struct Bucket`.
Each `Node` contains a string representing the name of the resident, a string representing an address, a `long` representing the phone number, and a `Node*` to the next node in the linked list. Valid phone numbers should have a fixed length of 10 (e.g. 0123456789, 9999999999, etc.).

3 Instructions

3.1 General instructions

You have been given one C file - **book.c** in which to implement the data structure. Each function has a block comment that describes exactly what it should do and there is a synopsis included below. **NOTE: You cannot assume data passed to a function is always valid. Our Unit Tests will test invalid arguments.**

You can use any function included in stdlib.h, string.h and stdio.h, and they are already included for you. **Importing any other library is forbidden.**

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. The autograder catches memory leaks with Valgrind. See Section 4.1 for details on Valgrind.

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

3.2 Functions you need to implement

- `char *dynamic_string_copy(const char *str)`: Allocate a new string, copy the passed-in string over to it, and return the new string. This function might be helpful when implementing later functions.
- `AddressBook *create_book(void)`: Allocate a new `AddressBook`, and return a pointer to it. You should initialize `buckets` to point to an array of length `BUCKET_CAPCCITY` and `size = 0`
- `Node *create_node(const char *name, const char *address, long phone_number)`: Allocate and initialize a new `Node` with a **copy** of the given data that's passed in as arguments, and return a pointer to it. Don't forget to set the `next` member for this new `Node` to `NULL`.
- `int put(AddressBook *book, const char *name, const char *address, long phone_number)`: Create a new `Node`, put it into the `AddressBook` and increment the size of `AddressBook`. The bucket to put the node can be computed with `hash(char *)`. Add the node to the front of the bucket's linked list. If a node with the same name already exists, don't make any changes to the table. Return SUCCESS if the function is successful and FAILURE if the function fails.
- `char *remove_node(AddressBook *book, const char *name)`: Remove the `Node` with the given name. Assign the predecessor's next `Node` to be the successor of the removed `Node`, and return the removed `Node`'s name.
- `Node *get(AddressBook *book, const char *name)`: Return a pointer to the `Node` with the given name. Return `NULL` if the given book is `NULL`, the given name is `NULL`, or a node with the given name is not found.
- `char *update_node(AddressBook *book, const char *name, const char *address, long phone_number)`: Update the data of the `Node` with the given name. Return a pointer to the old name.
- `void destroy_book(AddressBook *book)`: **Completely** destroy the given `AddressBook`. This means freeing **everything** that the `AddressBook` references (all `Node`s).
- `void destroy_bucket(AddressBook *book, int bucket)`: Destroys all nodes in the linked list of a bucket.

3.3 Functions provided

Do not modify any function listed in this section.

- `int hash(char *name)`: Computes a hash from the string provided and returns an integer.

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 `main.o` and one called `book_suite.o`. These contain the 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 your own `main.c` file: `make student`:
3. To run the tests without valgrind or gdb: `make run-tests`
4. To run a specific test-case: `make TEST=test_name run-case`
5. To run your tests with valgrind: `make run-valgrind`
6. To debug a specific test with valgrind: `make TEST=test_name run-valgrind`
7. 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 `book.c` **and** `collaborators.txt`.

The autograder uses the original header file that we provided. Any local changes to `book.h` **will not be reflected in Gradescope.**

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 above.

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 Is collaboration allowed?

From the syllabus:

- You must submit an assignment or project as your own work. No collaboration on answers is permitted. Absolutely no code or answers may be copied from others. Such copying is Academic Misconduct.
- Using code from GitHub, via Googling, from Stack Overflow, etc., is Academic Misconduct (Honor Code: Academic Misconduct includes "submission of material that is wholly or substantially identical to that created or published by another person").
- Publishing your assignments on public repositories accessible to other students is unauthorized collaboration and thus Academic Misconduct.

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

7.5 Syllabus Excerpt on Academic Misconduct

The goal of all assignments in CS 2110 is for you to learn. Learning requires thought and hard work. Copying answers thus prevents learning. More importantly, it gives an unfair advantage over students who do the work and follow the rules.

1. **As a Georgia Tech student, you have read and agreed to the Georgia Tech Honor Code.** The Honor Code defines Academic Misconduct as "***any act that does or could improperly distort Student grades or other Student academic records.***"

2. You must submit an assignment or project as your own work. **No collaboration on answers is permitted. Absolutely no code or answers may be copied from others. Such copying is Academic Misconduct.**

3. Using code from GitHub, via Googling, from Stack Overflow, etc., is Academic Misconduct (Honor Code: Academic Misconduct includes "*submission of material that is wholly or substantially identical to that created or published by another person*").

4. Publishing your assignments on public repositories accessible to other students is unauthorized collaboration and thus Academic Misconduct.

5. Suspected Academic Misconduct will be reported to the Division of Student Life Office of Student Integrity. It will be prosecuted to the full extent of Institute policies.

6. Students suspected of Academic Misconduct are informed **at the end of the semester**. Suspects receive an *Incomplete* final grade until the issue is resolved.

7. We use accepted forensic techniques to determine whether there is copying of a coding assignment.

8. **If you are not sure about any aspect of this policy, please ask Dr. Conte.**