CS 2110 Homework 9
Dynamic Memory

Prabhav Gupta, Richard So, Anisha Gurram, Alex Whitlock,
Rohan Bafna, Greg Gould, Kunal Daga, Samarth Kamat

Spring 2024

Contents

1 Overview **2**
- 1.1 Purpose . . . 2
- 1.2 Task . . . 2
- 1.3 Criteria . . . 2

2 Instructions **2**
- 2.1 Implementation Overview . . . 2
- 2.2 Implementation Tips . . . 5
- 2.3 Testing . . . 5

3 Deliverables **6**

4 Appendix **7**

5 Rules and Regulations **7**
- 5.1 Academic Misconduct . . . 7

1 Overview

1.1 Purpose

The purpose of this assignment is to introduce you to dynamic memory allocation in C. How do we allocate memory on the heap? How do we de-allocate it when it is no longer used?

You will learn how to manage memory in C with the malloc family of functions.

1.2 Task

Social media is on the rise! Slack is used by professionals across the world to communicate and interact with each other. In this assignment, you will be creating a Slack channel!
Your Slack channel will be represented by a `struct channel`. Each channel has a list of users subscribed to it along with the posts created by them. Each user has their own username and account ID. Each post has its own post ID, the ID of the user who posted it, the post's contents, and a list of reactions to it. Both account and post have their own struct. You can find details about these structs and more in the included **slack.h** file, as well as in this document. Your Slack channel will be able to add, remove, mutate, and query the data stored within it

You will be writing code in

1. slack.c
2. main.c

(main.c is only for your own testing purposes, you will not submit it.)

1.3 Criteria

You will be graded on your ability to implement the structs and data structures specified properly. Your implementation must be as described by each function's documentation. Your code must manage memory correctly, meaning it must be free of memory leaks.

Note: A memory leak is when a block of memory is allocated for some purpose, and never de-allocated before the program loses all references to that block of memory.

2 Instructions

2.1 Implementation Overview

You have been given one C file - **slack.c**. 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. The changes in the other files will not be reflected by the autograder, so you will not be able to use them.

Remember that the structs passed in to functions are already malloc-ed. It is key to note that this data is malloc-ed separately from the structs that contain it (a `struct channel` will hold `struct post`).

Forgetting to free this data when it is removed can cause memory leaks (memory that's no longer being used but not freed), but freeing it when it should be returned to the user can created dangling pointers (memory that's freed but still being referenced). Keep this in mind when writing functions that deal with removing elements.

You will create and allocate the following structs: `struct channel`, `struct post`, `struct account`, `struct reaction`, `struct linked_list`, and `struct node`. Below is an overview of what each struct contains.

Each `struct channel` has the following:

- `int numUsers`, the number of users in the channel.
- `Account** users`, a pointer to an array that holds pointers to `struct account`s. You will need to dynamically allocate space for this array.
- `LinkedList posts`, a linked list of all the posts in the channel

Each `struct post` has the following:

- `int postID`, a unique identifier for each post.
- `int senderID`, the accountID of the post's sender.
- `char* text`, a string containing the content of the post.
- `Reaction reactions[]`, an array of size MAX_REACTION_NUM (10) containing `struct reaction`s.
- `int numReactions`, the number of reactions on the post currently.

Each `struct account` has the following:

- `int accountID`, a unique identifier for each account.
- `char* username`, a string representing the username.

Each `struct reaction` has the following:

- `int userID`, the accountID of whoever posted the reaction.
- `enum ReactionType reaction`, an enum of various emoji options to react to a post with.

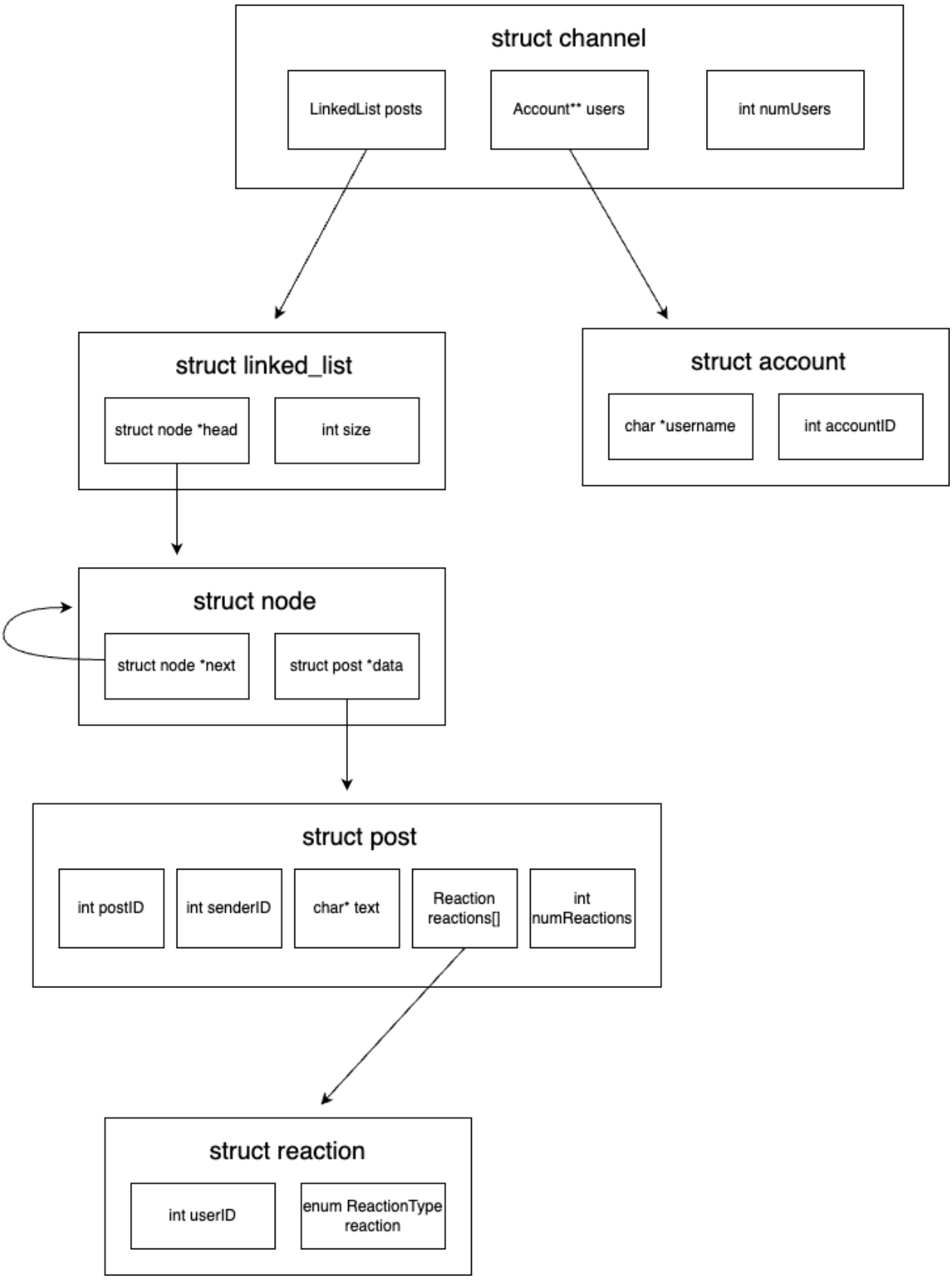
Each `struct linked_list` has the following:

- `struct node *head`, the head of the Linked List.
- `int size`, the number of nodes in the Linked List currently.

Each `struct node` has the following:

- `struct node *next`, a pointer to the next `struct node` in the list.
- `Post* data`, a pointer to a `struct post` that the node represents in the Linked List.

Refer to the following diagram for a visual representation of the interactions between `struct`s in this homework:


struct channel
LinkedList posts
Account** users
int numUsers
struct linked_list
struct node *head
int size
struct account
char *username
int accountID
struct node
struct node *next
struct post *data
struct post
int postID
int senderID
char* text
Reaction reactions[]
int numReactions
struct reaction
int userID
enum ReactionType reaction

Once you've implemented the functions in the slack.c file, or after implementing a few, compile your code using the makefile. You may test your functions manually by writing your own test cases in the provided `main.c`, or run the autograder. See the Testing section below for more information.

Please COMPILE OFTEN. The compiler will reveal many syntax errors that you would rather find early before making them over and over throughout your code. Waiting until the end to compile for the first time will cause big headaches when you are trying to debug code. We speak from experience when we say compile often. :)

2.2 Implementation Tips

- When creating structs, consider what parts of the struct need to be dynamically allocated with the use of malloc. We use dynamic allocation when amount of space needed to be allocated is unknown at compile time, so think about which members of the structs do not have a constant size.
- calloc and realloc are your friends. Using them instead of malloc in certain cases will help make your life easier and reduce the amount of code you'll need to write.
- When freeing structs, you will need to free any components of the struct that have been malloc-ed individually before you can free the struct itself. Keeping this in mind will help prevent memory leaks in your code.

2.3 Testing

First, make sure you are in the docker container environment using the `cs2110docker.sh` or `cs2110docker.bat` scripts, then change your directory to be on where your HW9 files are located.

To compile and test your code, use the Makefile given as follows:

To manually test specific functions in your code, fill in the main function in `main.c` with tests, and run

```
make hw9
```

This will create an executable called hw9. Then, run the main function using

```
./hw9
```

To run the autograder, run

```
make run-tests
```

The local autograder does not test for memory leaks, though the one on gradescope does. To test your code for memory leaks locally, you will need to use valgrind.

To test your code using valgrind, run

```
make run-valgrind

make run-valgrind TEST=<testname> # to run valgrind on a specific test case
```

To debug a certain test case with `gdb`, run the following make target:

```
make run-gdb TEST=<testname>
```

3 Deliverables

Please upload the following files to Gradescope:

1. `slack.c`

4 Appendix

5 Rules and Regulations

1. Please read the assignment in its entirety before asking questions.
2. Please start assignments early, and ask for help early. Do not email us the night the assignment is due with questions.
3. If you find any problems with the assignment, please report them to the TA team. Announcements will be posted if the assignment changes.
4. You are responsible for turning in assignments on time. This includes allowing for unforeseen circumstances. If you have an emergency please reach out to your instructor and the head TAs **IN ADVANCE** of the due date with documentation (i.e. note from the dean, doctor's note, etc).
5. You are responsible for ensuring that what you turned in is what you meant to turn in. 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. Email submissions will not be accepted.
6. See the syllabus for information regarding late submissions; any penalties associated with unexcused late submissions are non-negotiable.

5.1 Academic Misconduct

Academic misconduct is taken very seriously in this class. Quizzes, timed labs and the final examination are individual work. Homework assignments will be examined using cheat detection programs to find evidence of unauthorized collaboration.

You are expressly forbidden to supply a copy of your homework to another student. If you supply a 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 platform which would allow other parties to it (public repositories, pastebin, etc). If you would like to use version control, use a private repository on github.gatech.edu

Homework collaboration is limited to high-level 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.

High-level collaboration means 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, or providing other students any part of your code.

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.

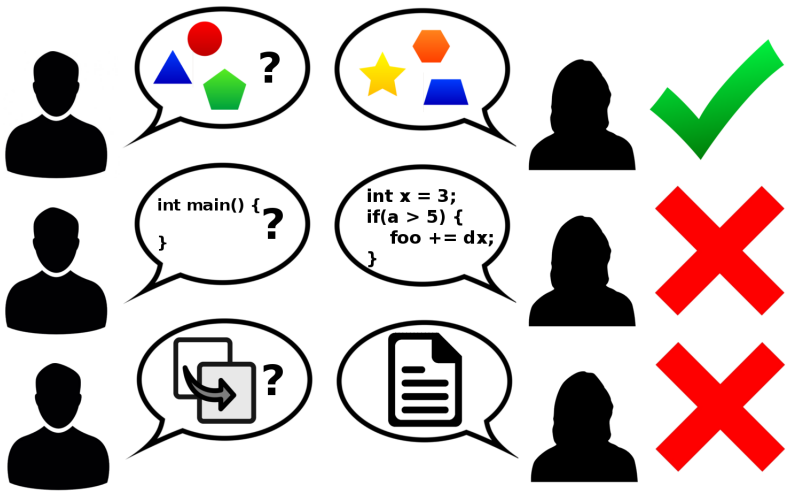


Figure 1: Collaboration rules, explained colorfully