CS 2110 Project 4: C Programming

Prabhav Gupta, Rishi Nopany, Alex Whitlock, Max Ko

Fall 2023

Contents

1 Introduction

Hello and welcome to Project 4. You'll be implementing a student gradebook program in C! There are `gradebook.c` and `gradebook.h` files already provided for you, including the function prototypes and basic descriptions. More in-depth descriptions for each function will be provided through this document.

Please read through the entire document before starting. Often times we elaborate on instructions later on in the document. **Start early** and if you get stuck, there's always Piazza and office hours.

2 What functions can I use?

1. You **are** allowed to make your own helper functions in the '.c' project files if you so desire.
2. You can use the following library functions in your project: **printf, strlen, strcpy, strcmp**.
3. Do **NOT** modify or add your own include statements. Including a new header file is NOT allowed and the autograder will get mad at you. This includes **ANY** standard library headers not already provided.
4. In lecture, you will learn about dynamic memory allocation with **malloc**. Please note - **You do NOT need to use malloc or any dynamic memory functions in this project**. If you attempt to use malloc the compiler will tell you to include stdlib.h which is **NOT** allowed.2

3 Gradebook

The goal of this project is to implement a gradebook that is responsible for various operations. The `Gradebook` will be a struct containing several elements, most notably an array of `GradebookEntry`. These GradebookEntries will be where the students' grades are stored. Each entry will hold one student, the grades for that student, and the student's overall grade (a weighted average).

Note: All of the functions will be modifying a provided global variable `struct Gradebook gradebook;`

3.1 Overview

3.1.1 Gradebook Layout

Below is a visual representation of the `Gradebook` struct as well as the `GradebookEntry`. A `Gradebook`, as shown the code snippet below this, contains three arrays as well as a double and int. Comparing the struct definitions to this table may be helpful in trying to understand how the individual students piece into the overarching structs.

Gradebook Entry

Student	HW1	HW2	HW3	P1	P2	Student Average
`student`	`grades[]`					`average`

Gradebook

Alex	58.7	45.0	32.3	43.3	43.45	44.84
Max	43.02	89.97	99.54	2.62	18.23	60.74
Rishi	90.52	92.0	91.1	99.8	88.4	91.93
`more entries would be here...`						

`Size:`	`3`
`Course Average:`	`65.84`

Weights

HW1 Weight	HW2 Weight	HW3 Weight	P1 Weight	P2 Weight
0.25	0.25	0.25	0.125	0.125
`weights[]`				

Averages

HW1 Avg	HW2 Avg	HW3 Avg	P1 Avg	P2 Avg
64.08	75.86	74.31	48.57	50.03
`assignment_averages[]`				

3.1.2 Provided Structures and Definitions

The following structs and definitions will be important (found in `gradebook.h`).

```
#define ERROR -1
#define SUCCESS 0
#define INVALID_GTID -1

#define MAX_NAME_LENGTH 20
#define MAX_ENTRIES 50
```

```
enum Major { CS = 0, CE, EE, IE };

enum Assignment { HW1, HW2, HW3, P1, P2, NUM_ASSIGNMENTS };

struct Student {
  char name[MAX_NAME_LENGTH];
  int gtid;
  int year;
  enum Major major;
};

struct GradebookEntry {
  struct Student s;
  double grades[NUM_ASSIGNMENTS];
  double average;
};

struct Gradebook {
  struct GradebookEntry entries[MAX_ENTRIES];
  double assignment_averages[NUM_ASSIGNMENTS];
  double course_average;
  double weights[NUM_ASSIGNMENTS];
  int size;
};
```

3.1.3 Important Notes and Tips

1. **ALL** of the code that you will write will be in the `gradebook.c` file. You should use the `gradebook.h` file as reference for all of the structs and definitions that you will use in the `gradebook.c` file.
2. It will be helpful to go through and review the purpose of **ALL** of the functions before starting. Some of the later functions might help implement earlier functions.
3. If your inputs are NULL in any of the functions, return `ERROR`.
4. For all students within the gradebook, names and GTID's must be unique. You will be responsible for ensuring that this property is maintained.

3.2 Add Student

```
int add_student(char *name, int gtid, int year, char *major);
```

This function will take in several parameters as defined in the function prototype above (and also in the `gradebook.h` file). The goal of this function is to create a `GradebookEntry` and populate the `Student` struct with the parameters defined above. Then add this entry into the end of the gradebook. The other elements in the entry (grades and average) will be set to 0.

The student's major is given as a char*. You will need to determine which enum value the char* corresponds to. For example, if the char* major equals "CS", you know that it corresponds to the CS value in the enum. If you receive a char* that does not correspond to one of the enum values, return `ERROR`.

This function is also responsible for updating the overall course averages (assignment and course averages) as well as incrementing the size (number of gradebook entries) in gradebook.

If the gradebook isn't able to hold any more students (the gradebook is full) or if there is a duplicate student (duplicate name or gtid), return `ERROR`. In addition, if the name is invalid (too long), return `ERROR`. Otherwise, return `SUCCESS`.

3.2.1 More on Strings

- Strings are essentially a contiguous array of ASCII values. In this case, the first character is stored at the address given by the parameter 'name'.
- The string continues until the first instance of a null terminator, which has the value of 0. **Remember to account for the null terminator when calculating the length of a string.**
- Pointers in C can be treated as arrays. In particular, writing 'string[i]' will give the i-th character in 'string'. You can read from that character, as well as assign to it.

```
Example:
name = "Chris"
Stored at address x4000
 ----------------------------------------------
| 'C'   | 'h'   | 'r'   | 'i'   | 's'   | '\0'  |
| x4000 | x4001 | x4002 | x4003 | x4004 | x4005 |
 ----------------------------------------------
Note: length = 6
```

3.3 Update Grade

```
int update_grade(char *name, enum Assignment assignment_type, double new_grade);
```

This function will be responsible for updating/adding the specified grade of the student. Given a name, this function will search through the gradebook, and when the student's name is matched, update the student's grade (specified by `assignment_type`) with `new_grade`.

This function will also be responsible for updating the student's average based on the new grade input (see if there are any functions that already do this). In addition, this function should ensure that the overall course averages are still up-to-date after the grade updates.

If the student is not found, return `ERROR`. Otherwise, return `SUCCESS`.

3.4 Add Student with Grades

```
int add_student_with_grades(char *name, int gtid, int year, char *major, double *grades);
```

This function is responsible for adding a student to the gradebook and filling each of the students' grades in with the values provided in `grades`. You may find it helpful to review the defined functions to determine if they could be useful helper functions.

The student's major is given as a char*. You will need to determine which enum value the char* corresponds to. For example, if the char* major equals "CS", you know that it corresponds to the CS value in the enum. If you receive a char* that does not correspond to one of the enum values, return `ERROR`.

Additionally, this function is responsible for updating the overall assignment averages and course average as well as incrementing the size (number of gradebook entries) in gradebook.

Also, as a helpful reminder, the length of `double *grades` is equal to `NUM_ASSIGNMENTS`.

If the gradebook is full or unable to have their grades set, return `ERROR`. In addition, if the name is invalid (too long), return `ERROR`. Otherwise, return `SUCCESS`.

3.5 Calculate Average

```
int calculate_average(struct GradebookEntry *entry);
```

The function will take in the parameter `entry` as shown above which will be a pointer to a `GradebookEntry`. The specified entry's grade average will then be calculated based on the `weights[]` table held in the `Gradebook` struct and the individual assignment's grade pointed to by `*entry`.

If the average isn't able to be calculated (ie. pointer is null) return `ERROR`. Otherwise, return `SUCCESS`.

3.6 Calculate Course Average

```
int calculate_course_average();
```

The course average calculated by **averaging the averages of all students** and storing in `course_averages`.

This function will also update `assignment_averages[]` with each assignment's respective average score using the same function.

Essentially, take the average of each column in the gradebook.

Note: Calculating the average using the overall averages field might lead to different numbers before rounding due to how floating point numbers are handled, so for consistency use each `StudentEntry`'s average and take the average of that sum. In a proper implementation of storing grades, floating point numbers wouldn't be used as they aren't precise enough to handle calculations. The use of a double is just to simplify the project.

If the average isn't able to be calculated (ie. no students in Gradebook) set all of the averages' values to 0 and return `ERROR`. Otherwise, return `SUCCESS`.

3.7 Search Student

```
int search_student(char *name);
```

Given the student's name, return the index where the student is located at. If their name isn't found, return `ERROR`, otherwise return `SUCCESS`.

3.8 Withdraw Student

```
int withdraw_student(char *name);
```

This function is responsible for removing a student from the gradebook. Much like updating the grade, the function will search through the gradebook, searching for the first name (not firstname) that matches. This entry will be deleted from the gradebook, **maintaining the current ordering of the gradebook** while removing the gap created by the entry.

Additionally, this function is responsible for updating the overall assignment averages and course average as well as decrementing the size (number of gradebook entries) in gradebook.

If the student is not found, return `ERROR`.

If the gradebook is empty after removing the student, averages should be updated to reflect the now empty state (set values to 0) and `SUCCESS` should still be returned.

The example below illustrates the idea of how order should be maintained. Its important to note, the elements in each array are purely illustrative. The actual layout in memory is dependent on the data stored in the array (pointer, a struct, etc). In this case, `GradebookEntry entries[]` is going to hold a `GradebookEntry` in each index.

```
Example:
gradebook = x4000
toRemove = ‘M’
 ------------------------------------------------
|   ‘A’   |   ‘M’   |   ‘R’   |   ...   |   END   |
|  x4000  |  x4001  |  x4002  |  x4003  |  x4004  |
 ------------------------------------------------
 ------------------------------------------------
|   ‘A’   |   ...   |   ‘R’   |   ...   |   END   |
|  x4000  |  x4001  |  x4002  |  x4003  |  x4004  |
 ------------------------------------------------
 ------------------------------------------------
|   ‘A’   |   ‘R’   |   ...   |   ...   |   END   |
|  x4000  |  x4001  |  x4002  |  x4003  |  x4004  |
 ------------------------------------------------
```

3.9 Top 5 GTIDs

```
int top_five_gtid(int *gtids);
```

Given an array of integers of size 5, fill in the array with the gtid's of the top 5 students in the class (based upon average) The array should be filled in descending order by average. Essentially, the gtid of the student with the highest average is the first gtid in the array (index 0), the 5th best student is the last gtid in the array (index 4).

If there aren't enough entries to fully populate the array, fill in the remaining values with `INVALID_GTID`.

Return `SUCCESS` if the entries are able to be found and `ERROR` if the gradebook is empty.

3.10 Sort by Name

```
int sort_name();
```

Sort the gradebook by name in alphabetical order (first last) using bubble sort and modify the gradebook to reflect this sorted value.

Return `SUCCESS` if the gradebook is sorted, and `ERROR` if the gradebook is empty.

3.11 Sort by Averages

```
int sort_averages();
```

Sort the gradebook in descending order by student average score. This sort should be stable (ie. if student A has a 92.2 at index 4 and student B has a 92.2 at index 5, student A should remain ahead of student B regardless of the final position).

Return `SUCCESS` if the gradebook is sorted, and `ERROR` is the gradebook is empty.

3.12 Print Gradebook

```
int print_gradebook();
```

This function will print the entire gradebook with each value being separated by a comma. It should follow the form below. Notice that there isn't a space after each comma.

All of the floats that you print must be manually rounded to 2 decimal places.

Example for a gradebook with 2 students:

```
student_1,major,grade1,grade2,...,student_average\n
student_2,major,grade2,grade3,...,student_average\n

Overall Averages:\n
grade1_average,grade2_average,...,course_average\n
```

Note 1: The `\n` shouldn't print, just represents the newline for this example.

Note 2: There is an empty line above the "Overall Averages:" line.

Return `SUCCESS` if the gradebook is printed and `ERROR` if the gradebook is empty.

4 Building & Testing

All of the commands below should be executed in your Docker container terminal, in the same directory as your project files.

4.1 Helpful Info

1. **For issues with Docker** please refer to the 2110 Docker Guide for C
2. **Use Docker's "Interactive Terminal"** Run the following command in your terminal to access Docker's terminal much easier: `./cs2110docker-c.sh` on Linux/Mac and `./cs2110docker-c.bat` on Windows.
3. From within the "Interactive Terminal" you should notice the "host/" directory when you type `ls`. Navigate to your Project directory and run the unit tests using the commands mentioned later.
4. To exit Docker's "Interactive Terminal" simply type: **exit**.
5. **make** is a program to help you build your project (in other words, it helps you compile).
6. **GDB** is a very useful debugger however, it may be a bit confusing at first. Please refer to the following GDB Cheatsheet or ask a TA for help!

4.2 Unit Tests

These are the same tests that will run on Gradescope

To run the autograder locally (without GDB):

1. **make clean** - Clean your working directory
2. **make tests** - Compile all the required files
3. **./tests** - Run the unit tests
4. **make** - Compile all the required files and Run the unit tests (previous two steps with one command)

Executing **./tests** will run all the test cases and print out a percentage, along with details of the **failed test cases**.
Other available commands (after running make tests):

- To run a specific test case (to avoid all printing output/debug messages for all test cases):

```
make run-case TEST=testCaseName

Example: make run-case TESTS=test_compare
```

- To run a test case with gdb:

```
make run-gdb TEST=testCaseName (or no testCase to run all in gdb)
```

4.3 Write Your Own Tests

In your Project 4 directory there is a file named “main.c”. This file can be used to make your own tests. The main function provided can be executed with the following command.

1. **make student**

For example, if you want to write a test for your “add_student” and “search_student” functions, this is an example of how you could write the tests.

```
int main(void) {
  // Add your own test cases here to test your functions :)
  char *student_name = "John Doe";
  int gtid = 903123456;
  int year = 2025;
  char *major = "CS";

  // Add a new student
  int add_result = add_student(student_name, gtid, year, major);
  printf(
      "Adding Student - Expected Return Value: %d, Actual Return Value: %d\n",
      SUCCESS, add_result);

  // Verify the student was added by searching for them
  int search_add_result = search_student(student_name);
  printf("Searching Added Student - Expected: Index >= 0, Actual: %d\n",
         search_add_result);

  if (search_add_result == ERROR) {
    printf("Student not found after add. Exiting test.\n");
    return ERROR;
  }
  return SUCCESS;
}
```

Then execute **make student** inside Docker. Then, assuming your function is correct, you should receive an output similar to this:

```
Adding Student - Expected Return Value: 0, Actual Return Value: 0
Searching Added Student - Expected: Index >= 0, Actual: Index_Value
```

Where Index_Value is the index the student is located at (should be 0). Keep in mind, in this code `SUCCESS` is defined as 0 and `ERROR` is defined as -1.

5 Deliverables

Turn in the file, `gradebook.c` to Gradescope by the due date.

Please do not wait until the last minute to run/test your project, history has proved that last minute turn-ins will result in long queue times for grading on Gradescope. You have been warned!!!

6 Rules and Regulations

6.1 General Rules

1. Although you may ask TAs for clarification, you are ultimately responsible for what you submit.
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.

6.2 Submission Conventions

1. In order to submit your assignment, submit the files individually to the Gradescope assignment.
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.

6.3 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 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 Gradescope.
3. Projects turned in late receive partial credit within the first 48 hours, as defined in the syllabus. Between 0 and 24 hours late, you can receive a maximum score of 70%. Between 24 and 48 hours late, you can receive a maximum score of 50%. We will not accept projects turned in over 48 hours late.
4. 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.

6.4 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 or persons").

- Publishing your assignments on public repositories, github, etc, that is accessible to other students is unauthorized collaboration and thus Academic Misconduct.

6.5 Coding Guidelines C/C++ code

1. You must turn in ALL files specified in the "Deliverables" section of the assignment instructions. We reserve the right to impose a penalty on submissions that do not follow the given submission directions.
2. You must provide a Makefile that compiles and links your code by default. If you are given a Makefile with the project, we expect your code to compile with make. (don't worry, we'll explain what all this means later)
3. Your code must compile with gcc on Ubuntu 22.04 LTS. If your code does not compile, you will receive a 0 for the assignment.
4. You will be penalized if your code produces warnings when compiled with the given Makefile, or the following flags if no Makefile is provided: gcc -Wall -Wextra -Wstrict-prototypes -pedantic -O2
5. Code should be well commented and use a clean, consistent (readable) style (i.e., proper indenting, etc.). We reserve the right to impose style requirements, and deduct for non-conforming solutions. This is not the obfuscated C code competition!

6.6 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 Academic Honor Code.** The Academic 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 or persons*").
4. Publishing your assignments on public repositories, github, etc, that is 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. Submitting an assignment with code or text from an AI assistant (e.g., ChatGPT) is academic misconduct.
9. **If you are not sure about any aspect of this policy, please ask your lecturer**.

Using AI Assistants

Anything you did not write in your assignment will be treated as an academic misconduct case. If you are unsure where the line is between collaborating with AI and copying AI, we recommend the following heuristics:

Heuristic 1: Never hit "Copy" within your conversation with an AI assistant. You can copy your own work into your own conversation, but do not copy anything from the conversation back into your assignment. Instead, use your interaction with the AI assistant as a learning experience, then let your assignment reflect your improved understanding.

Heuristic 2: Do not have your assignment and the AI agent open at the same time. Similar to the above, use your conversation with the AI as a learning experience, then close the interaction down, open your assignment, and let your assignment reflect your revised knowledge. This heuristic includes avoiding using AI directly integrated into your composition environment: just as you should not let a classmate write content or code directly into your submission, so also you should avoid using tools that directly add content to your submission.

Deviating from these heuristics does not automatically qualify as academic misconduct; however, following these heuristics essentially guarantees your collaboration will not cross the line into misconduct.