CS 2110 Project 4: C Programming

Aarsh Patel, Ammar Ratnani, Sean Crowley, Charlie Gunn, Johnson Lu, Todd Hayes

Fall 2021

Contents

1 Introduction **2**

2 Am I allowed to make my own functions? **2**

3 Part 1 - Assembly Functions in C **3**

3.1 Compare . . . 3
3.2 Modulus . . . 3
3.3 String Manipulation . . . 3
3.4 Shift Right . . . 3
3.5 Shift Left . . . 3
3.6 Collatz . . . 4
3.7 Dot Product . . . 4

4 Part 2 - Calendar Implementation **5**

4.1 Overview . . . 5
4.2 Calendar Conditions . . . 6
4.3 Functions . . . 6
4.3.1 Initialize Calendar . . . 6
4.3.2 Check Event Overlap . . . 6
4.3.3 Add Event . . . 7
4.3.4 Remove Event . . . 7
4.3.5 Change Event Time . . . 7
4.3.6 Change Event Description . . . 7
4.3.7 Sort Events . . . 8
4.3.8 Destroy Calendar . . . 8

5 Building & Testing **9**

5.1 Helpful Info . . . 9
5.2 Unit Tests . . . 9
5.3 Write Your Own Tests . . . 10

6 Deliverables **10**

7 Demos **10**

8 Rules and Regulations **12**

8.1 General Rules 12

8.2 Submission Conventions 12

8.3 Submission Guidelines 12

8.4 Syllabus Excerpt on Academic Misconduct 13

8.5 Is collaboration allowed? 13

1 Introduction

Hello and welcome to Project 4. This project is separated into 2 parts. Part 1, consists of functions you have seen in project 3 but now you get to complete them in C! Part 2, you'll write a calendar program in C!

The project is due November 15th, 11:59 PM EST

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 or office hours.

2 Am I allowed to make my own functions?

Yes! You are allowed to make your own helper functions in the '.c' project files if you so desire.

3 Part 1 - Assembly Functions in C

For this first section you will be completing Project 3 again but this time you have the C language to help you out. This section is specifically designed to demonstrate the power of C.

3.1 Compare

You are given two parameters: 'a' and 'b'. The value you return depends on whether 'a' is greater than 'b'.

1. return -1 if b is larger
2. return 0 if equal
3. return 1 if a is larger

3.2 Modulus

Since there are different interpretations of modulus for negative numbers, you should take the absolute value of both of your parameters and then perform the operation. Return the result of the modulus operation.

3.3 String Manipulation

You will write a function that takes in a string stored in memory, changes any lowercase letters in the string to uppercase characters. Implement toUppercase.

* Strings are essentially a contiguous array of ASCII values. In this case, the first character is stored at the address given by the parameter 'string'.

* The string continues until the first instance of a null terminator, which has the value of 0.

* Memory addresses 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:
string = x4000
 -------------------------------------
| 'h'   | 'A'   | 'h'   | 'A'   | '\0'  |
| x4000 | x4001 | x4002 | x4003 | x4004 |
 -------------------------------------
```

The string that you receive can contain any characters, and you should change all letters to uppercase, and leave non-letters as is.

3.4 Shift Right

You will be given an original value, and an amount of bits to shift by **to the right**. Return the shifted value. You will not have to handle overflow.

3.5 Shift Left

You will be given an original value, and an amount of bits to shift by **to the left**. Return the shifted value. You will not have to handle overflow.

3.6 Collatz

You will be implementing a calculator for iterations of the Collatz Conjecture.

The Collatz Conjecture is a famous unsolved math problem, which operates under a few simple rules. We have our Collatz function, C(n), which takes in exclusively positive integers.

1. If n is an odd number, C(n) = 3n + 1
2. If n is an even number, C(n) = n/2

From here, we seek to calculate how many times we need to run n (and its subsequent results) through the Collatz function in order for it to reach the value 1.

3.7 Dot Product

The last method you will be writing is a dot product calculator. Given two arrays in memory, your job is to compute the dot product and return it back into memory. In mathematics, the dot product is calculated as the sum of multiplication between each pair of vector components.

If A = [a1 a2 a3] and B = [b1 b2 b3], then the dot product between A and B would be

$$(a1 * b1) + (a2 * b2) + (a3 * b3)$$

The values inside vectors can be positive or negative integers.

You are given three input values:

* ‘vecA‘ represents the address of the first vector in memory

* ‘vecB‘ represents the address of the second vector in memory

* ‘len‘ represents the length of each individual vector

4 Part 2 - Calendar Implementation

The bulk of your work for this project will be implementing various functions for the 2021 Calendar. The specifications for these functions are listed briefly in the code itself, but are explained with more detail in the sections below. This section specifically pertains to the part2-calendar.c and part2-calendar.h files..

4.1 Overview

Before the function explanation, **please notice that there is a global calendar array** (found at the top of part2-calendar.c). This array will hold everything related to the calendar! You will be initializing it, adding events to it, removing events from it, and much more.

```
date calendar_2021[NUM_DAYS];
```

The following structs will be important (found in part2-calendar.h).

```
typedef struct _time_ {
    int hour;
    int minute;
} time;

typedef struct _event_ {
    char description[SIZE_DESCRIPTION];
    time start;
    time end;
} event;

typedef struct _date_ {
    int month;
    int day;
    int year;
    int num_events;
    event events[MAX_NUM_EVENTS];
} date;
```

Many of these functions return an integer flag which indicates success or failure. We have provided the macros `#define ERROR -1` and `#define SUCCESS 0` to help you (found in part2-calendar.h)

```
#define MAX_NUM_EVENTS 10       // MAX number of events
#define SIZE_DESCRIPTION 100    // MAX size of event description array
#define NUM_DAYS 365            // Number of days in 2021

#define ERROR -1                // Code used to signal ERROR occurred
#define SUCCESS 0               // Code used to signal SUCCESS occurred
```

4.2 Calendar Conditions

Lastly, there are a few rules that the calendar follows.

1. Dates are strictly saved as integers. For example, January is 1, February is 2, ..., December is 12.
2. `'Military Time'`: Event times will be strictly in a 24 hour time scale. That is, the time division is now out of 24 hours instead of two 12 hour periods (am/pm). So, times before 1 pm are written the same way (9:01, 10:46, 12:39) but now 1:00 pm is written as 13:00 , 2:03 pm is 14:03, 3:24 pm is 15:24, 11:59 pm is 23:59 and finally 12 am is 00:00.
3. `'Event Overlap'`: Calendar event times **can NOT** overlap in any way. For example, if an Event A starts at 2:46 and ends at 13:05 and Event B starts at 12:03 and ends at 15:32; these events are considered to be overlapping since the start of Event B is within the time range of Event A. **Edge Case:** if an Event A ends at 4:45 and an Event B starts at 4:45 we consider this an overlap and this should not occur.
4. An events start time **can NOT** be after its end time. For example, an event with a start time of 17:23 and an end time of 16:44 is considered invalid and is not allowed to be added to your calendar. **Edge Case:** an event starting at 14:55 and ending at 14:55 **is** allowed.
5. Multi-day events are not possible.
6. The number of events (num_events in the date struct) gives the number of **valid** events on that date. For example, given some date, if num_events = 2 and the events array looks like:

   $$[e1, e2, e3, e4, e5]$$

   we only consider event1 and event2 as valid events. The remaining events are garbage data and can be overwritten. If num_events = 0 and the event array looks like:

   $$[e1, e2, e3, e4, e5]$$

   we consider e1, e2, e3, e4, and e5 to all be invalid and can be overwritten.

4.3 Functions

4.3.1 Initialize Calendar

```
void initialize_calendar(void);
```

Initialize the entire calendar with the correct dates. Set the month, day, year for every date in the calendar and initialize the num_events to zero.

For example, calendar_2021[364] should be assigned December 31st, 2021 - (12, 31, 2021).

NOTE: Check section 3.2 "Calendar Conditions" to determine edge cases.

4.3.2 Check Event Overlap

```
int check_event_overlap(event e1, event e2);
```

Check if events e1 and e2 have overlapping times. See Calendar Conditions 'Event Overlap' for the definition of overlap.

If the events overlap in anyway, return ERROR. Otherwise, return SUCCESS.

NOTE: Check section 3.2 "Calendar Conditions" to determine edge cases.

4.3.3 Add Event

```
int add_calendar_event(date d, event new_event);
```

Add an event on the date specified. Find the date in the calendar and add the event into the calendar. Remember that new_event must not overlap with any of the events currently found on that date.

NOTE: date d is used to locate the date's index in calendar_2021. Do not simply add new_event to date d as this will not accomplish anything.

If you are successful in adding the new_event to the calendar return SUCCESS, otherwise return ERROR.

NOTE: Check section 3.2 "Calendar Conditions" to determine edge cases.

4.3.4 Remove Event

```
int remove_calendar_event(date d, int event_index);
```

Remove an event on the date specified. Find the date in the calendar and remove the event, specified by event_index.

NOTE: date d is used to locate the date's index in calendar_2021. Do not try to remove the event from date d as this will not accomplish anything.

If the event is removed successfully return SUCCESS, otherwise return ERROR.

Removing an event successfully requires that every subsequent event is shifted 'up' in the array. For example, given the following event array of size 4:

$$[e1, e2, e3, e4]$$

If e2 is removed from this array the final array of size 3 will look like this:

$$[e1, e3, e4]$$

NOTE: Check section 3.2 "Calendar Conditions" to determine edge cases.

4.3.5 Change Event Time

```
int change_event_time(date d, int event_index, time start, time end);
```

Change an event's start and end time. Find the date in the calendar and modify the event specified by event_index. Change that event's start and end time to the start and end time passed into the function.

NOTE: date d is used to locate the date's index in calendar_2021.

NOTE: Check section 3.2 "Calendar Conditions" to determine edge cases.

4.3.6 Change Event Description

```
int change_event_description(date d, int event_index, char description[]);
```

Change an event's description. Find the date in the calendar and modify the event specified by event_index. Change the event's description string with the description string passed into the function. Recall string assignment requires you to copy every element in the string. Attempting to assign strings will not work. For example,

```
char str1[SIZE];
char str2[SIZE];
str1 = str2;        // This is invalid!
```

When copying the new string into the `description`, make sure not go out of bounds of the array. If supplied a string too long to fit, truncate the string to the longest possible length that can still fit entirely in the `description` array.

NOTE: date d is used to locate the date's index in calendar_2021.

NOTE: Check section 3.2 "Calendar Conditions" to determine edge cases.

4.3.7 Sort Events

```
void sort_events(date d);
```

Sort a date's event array by start time (earliest time to latest time). For example,

```
    Unsorted:   [7:42-9:12, 4:35-6:55, 3:05-4:00]

    Sorted:     [3:05-4:00, 4:35-6:55, 7:42-9:12] // Earliest Start Time First
```

Remember, this is CS 2110, we really don't care that much about efficiency at the moment. Feel free to sort the event array with any algorithm you'd like.

NOTE: date d is used to locate the date's index in calendar_2021.

NOTE: Check section 3.2 "Calendar Conditions" to determine edge cases.

4.3.8 Destroy Calendar

```
void destroy_calendar(void);
```

Set everything in the calendar to 0. Set every date's month, day, year, and num_events to zero. Remember there is no need to zero out the events array because any events remaining are considered garbage data and invalid (see Calendar Conditions).

NOTE: Check section 3.2 "Calendar Conditions" to determine edge cases.

5 Building & Testing

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

5.1 Helpful Info

1. **Use Docker's "Interactive Terminal"** Run the following command in your terminal to access Docker's terminal much easier: **./cs2110docker.sh -it**
2. From within the "Interactive Terminal" you should notice the "host/" directory when you type **ls**. Navigate to your Project diretory and run the unit tests using the commands mentioned later.
3. To exit Docker's "Interactive Terminal" simply type: **exit**.
4. **make** is a program to help you build your project (in other words, it helps you compile).
5. **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!

5.2 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)
```

5.3 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 method provided can be executed with the following command.

1. **make student**

For example, if you want to write a test for your "compare" function from Part 1 you can do the following in the main.c file:

```
int main(void) {

    int a = 2;
    int b = 3;
    int test_value = compare(a, b);
    printf("Expected Return Value: -1\nActual Return Value: %d\n", test_value);

    return 0;
}
```

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

```
Expected Return Value: -1
Actual Return Value: -1
```

6 Deliverables

Turn in the files `collaborators.txt`, `part1-porting_assembly.c` and `part2-calendar.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!!!

NOTE: The syllabus states the following requirement is met - "Your code must compile with gcc on Ubuntu 18.04 LTS. If your code does not compile, you will receive a 0 for the assignment." HOWEVER, for this project your code MUST compile on our Docker image otherwise you will receive a 0.

7 Demos

This project will be demoed. Demos are designed to make sure that you understand the content of the project and related topics. They may include technical and/or conceptual questions.

- Sign up for a demo time slot via Canvas **before** the beginning of the first demo slot. This is the only way you can ensure you will have a slot.
- If you cannot attend any of the predetermined demo time slots, e-mail the Head TA **before** the beginning of the first demo slot.
- If you know you are going to miss your demo, you can cancel your slot on Canvas with no penalty. However, you are **not** guaranteed another time slot. You cannot cancel your demo within 24 hours or else it will be counted as a missed demo.

- Your overall project score will be `((autograder_score * 0.5) + (demo_score * 0.5))`, meaning if you received a 90% on your autograder, but a 30% on the demo you would receive an overall score of 60%. **If you miss your demo you will not receive any of these points and the maximum you can receive on the project is 50%.**
- You will be able to makeup one of your demos at the end of the semester for half credit.

8 Rules and Regulations

8.1 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.

8.2 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.

8.3 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. The late policy is applied to both portions of the project; turning the code in late means that you will also lose points on the demo.
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.

8.4 Syllabus Excerpt on Academic Misconduct

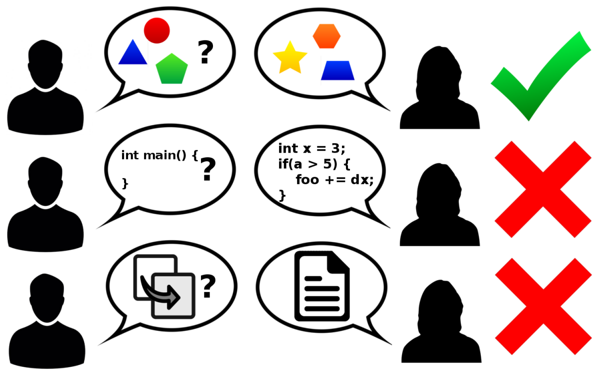
Academic misconduct is taken very seriously in this class.

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....

8.5 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.


?
int main() {
}
?
int x = 3;
if(a > 5) {
foo += dx;
}
?