CS 2110 Project 4: C Programming

Sainitin Daverpally, Christopher M Turko, Yuhan Li, Kartik Sinha, Sameer Suri

Fall 2022

Contents

1 Introduction **3**

2 Can I make my own functions? Can I use Operating System functions? **3**

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

3.1 Divide . . . 3
3.2 toLowercase . . . 3
3.3 GCD . . . 4
3.4 Fibonacci . . . 4
3.5 Count Ones . . . 4

4 Part 2 - RobbinU™ Implementation **5**

4.1 Introduction . . . 5
4.2 Overview . . . 5
4.3 More details on structs . . . 6
4.4 Program Conditions . . . 6
4.4.1 Open Account . . . 7
4.4.2 Close Account . . . 7
4.4.3 Create Acronym (Or Stock Ticker) . . . 7
4.4.4 Buy Stock . . . 7
4.4.5 Sell Stock . . . 8
4.4.6 Sort Stock . . . 8
4.4.7 Sort Stock Sector . . . 9
4.4.8 Total Investment . . . 9
4.4.9 Sector Investment . . . 9

5 Building & Testing **9**

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

6 Deliverables **11**

7 Demos **11**

8 Rules and Regulations **11**

8.1 General Rules 11

8.2 Submission Conventions 12

8.3 Submission Guidelines 12

8.4 Is collaboration allowed? 12

8.5 Syllabus Excerpt on Academic Misconduct 13

1 Introduction

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. In part 2, you will write an investment management program in C!

The project is due at 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 Ed Discussion or office hours.

2 Can I make my own functions? Can I use Operating System functions?

1. You **are** allowed to make your own helper functions in the `.c` project files if you so desire.
2. You can use `printf`, `strlen`, `strcpy` in your project but you shouldn't need any other OS functions.
3. Do NOT modify the provided `#include` statements or add your own.
4. In lecture you may have learned about dynamic memory allocation with `malloc`. Please note - **You do NOT need to use `malloc` or any other dynamic memory functions in this project**. If you attempt to use `malloc` the compiler will tell you to include `<stdlib.h>`. Including a new header file is NOT allowed and the autograder **will** get mad at you.

3 Part 1 - Assembly Functions in C

For this first section, you will be completing similar functions to those you wrote for Project 3, 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 Divide

You will implement a division function. You are given two parameters: `a` and `b`.

- return `0` if `b` is 0.
- return `a/b` otherwise.

Examples:

```
divide(5, 0) -> 0
divide(16, 2) -> 8
divide(22, 3) -> 7
```

3.2 toLowercase

You will implement a function `toLowercase`, that takes in a `char *str`, and converts all uppercase letters in the string to lowercase.

- Strings are essentially a contiguous array of ASCII values. In this case, the first character is stored at the address given by the parameter `str`.
- 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, `string[i]` will give the ith character in `string`. You can read from that character, as well as assign to it.

String representation in C is essentially identical to in assembly:

```
str = x4000
/--------------------------------------\
| 'h'   | 'A'   | 'h'   | 'A'   | '\0'  |
| x4000 | x4001 | x4002 | x4003 | x4004 |
\--------------------------------------/
```

- The string that you receive can contain any characters.
- You should change all letters to lowercase, and leave non-letters as is.
- The `str` input may be `NULL`.
- Note that the changes are done in-place.

Examples:

`toLowercase("Hello FRIEND")` will convert the input string to `"hello friend"`
`toLowercase("@@Zw!)` will convert the input string to `"@@zw!"`

3.3 GCD

You will implement an algorithm to find the greatest common divisor of two values. Your function will take in two integers, `a` and `b`, and return the largest integer that evenly divides both.

Examples:

```
gcd(4, 1) -> 1
gcd(-30, -15) -> 15
```

3.4 Fibonacci

You will implement a function that computes the n^{th} number in the Fibonacci sequence. The Fibonacci sequence is important in the field of computer science. It is a special sequence in which the current element is the sum of the previous two elements with $a_0 = 0$ and $a_1 = 1$.

Examples:

```
fib(2) -> 1
fib(7) -> 13
```

3.5 Count Ones

Given a binary number `num`, this function will return the number of bits set to 1 in the binary number. Recall that all numbers are binary numbers behind the scenes, even though they may be displayed to you as a decimal number.

Examples:

```
countOnes(7) -> 3
countOnes(-150) -> 28
```

4 Part 2 - RobbinU™ Implementation

The bulk of your work for this project will be implementing various functions for the RobbinU™ Investment Management App. 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-investments.c` and `part2-investments.h` files.

4.1 Introduction

Are you tired of traditional forms of investing? Well, we've got just the thing for you. With RobbinU™, you can make your favorite high-risk, high-return investments easier than ever. One slight issue, it does not exist yet...

CS 2110 TAs barely have enough time as it is, so we're giving YOU this unique opportunity to practice your newly acquired C skills by implementing the core behavior of the app.

4.2 Overview

There is a global `struct account` variable (found at the top of `part2-investments.c`). This structure contains an array of `struct stockProperties` called `stocksArr` which will hold everything related to the stocks in your investment app! You will be initializing it, adding stocks to it, removing stocks from it, and much more!

```
struct account myAccount;
```

The following `structs` and `enum` will be important (found in `part2-investments.h`).

```
enum sectorType {
    CRYPTO,
    ENERGY,
    TECHNOLOGY,
};

struct stockProperties {
    char name[MAX_NAME_LENGTH];
    char acronym[MAX_ACRONYM_LENGTH];
    float price;
    int numShare;
    enum sectorType sector;
};

struct account {
    char accountName[MAX_NAME_LENGTH];
    float totalBalance;
    int totalStocksBought;
    struct stockProperties stocksArr[MAX_NUM_STOCKS];
};

struct stockInfo {
    char name[MAX_NAME_LENGTH];
    float price;
    enum sectorType sector;
};
```

4.3 More details on structs

1. The `myAccount` struct is the main struct that will contain all of our app information.
2. In `myAccount`, the `stocksArr` will contain all the stock information of stocks you buy.
3. The `stockInfo` is the struct from which you recieve stock information either during buying or selling stocks.
4. The `stockProperties` is the struct we use to hold information about the stocks we have bought or hold in your account.

   For example, when you buy a stock, you receive stock information from `stockInfo` from which you create a `struct stockProperties` and add into your account's `stocksArr` if you can purchase the stock.

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

```
#define MAX_NAME_LENGTH 16      // Max length of a stock name
#define MAX_ACRONYM_LENGTH 4    // Max length of a created acronym (or Stock ticker) name
#define MAX_NUM_STOCKS 64       // Max number of stocks
#define ERROR 0                 // Code used to signal ERROR occurred
#define SUCCESS 1               // Code used to signal SUCCESS occurred
```

4.4 Program Conditions

Lastly, there are a few rules that RobbinU™ follows.

1. All stock names contain at least one vowel.
2. The length of all valid stock names is between 3 (inclusive) and 15 (inclusive), not including the `NULL` terminator.
3. There can be a max of 64 (`MAX_NUM_STOCKS`) unique stocks in our account at one time.
4. The length of all stock acronyms (stock tickers) is 3, not including the `NULL` terminator.
5. You cannot buy any shares of stock if you cannot afford the total value of the requested purchase. For example, imagine a stock priced at `10` that you want to buy `100` shares of. You currently only have `20.65` in your account. Therefore, you should not buy any shares of the stock, even though you could theoretically afford `2`.
6. Conversely, you cannot sell any shares if your account contains fewer shares the the requested sale. For example, imagine a stock is priced at `10` and you want to sell `100` shares of it. You currently only have `50` shares of that stock in your account. Therefore, you should not sell any shares of the stock, even though you could theoretically sell the `50` in your account.
7. Remember, each stock's total value is calculated by multiplying its `price` by its `shareCount`.
8. There are three sectors of stock you can buy on RobbinU™: `CRYPTO`, `ENERGY`, and `TECHNOLOGY`.

NOTE: You can also use `strlen` and `strcpy` when implementing the functions.

4.4.1 Open Account

```
int openAccount(char* name, float balance);
```

This function should open my account, initializing all the fields in the `account struct`.

You should initialize the account name and balance to the values provided as parameters. The new account starts with no stocks.

Recall that arrays (including strings) cannot be directly assigned in C. Instead, you must copy every character in the string. For example,

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

When copying the new string into the `name`, 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 `name` array.

This function should fail if the name or balance provided is invalid.

If you are successful in opening the account, then return `SUCCESS`, otherwise return `ERROR`.

4.4.2 Close Account

```
void closeAccount(void);
```

This function should close `myAccount`. To do this, you should clear the name, set the balance to 0, and empty the stocks array.

4.4.3 Create Acronym (Or Stock Ticker)

```
int createAcro(char* name, char* acro);
```

This function should create an acronym (stock ticker) for a stock using its name. **The format of a stock acronym is the following: stock name's first character + first vowel + stock name's last character.**

You can assume that all stock names on the RobbinU™ app have at least one vowel in them.

This function should fail if the stonk name provided is invalid. On success, create the acronym and write the result into the string pointed to by `acro`.

4.4.4 Buy Stock

```
int buyStock(struct stockInfo stock, int shareCount);
```

Buy a stock with the specified stock information and shares count. You will do this by adding it to your stock array and subtracting the total value of the stock purchase from your account's balance. If the stock already exists in your stock array, you only need to increase the `shareCount`. When buying a stock, you should use the passed in `price`, not the `price` in your `struct stockProperties`. Additionally, you should not edit the price in `struct stockProperties` if you already own the stock.

This function should fail and no edits to the account should be made if you can't afford the **entire** stock purchase, the specified share count is invalid, or your account is full.

If you are successful in buying the stock, then return `SUCCESS`, otherwise return `ERROR`.

4.4.5 Sell Stock

```
int sellStock(struct stockInfo stock, int shareCount);
```

Sell a stock with the specified stock information and shares count. You will do this by either reducing the share count or removing the corresponding stock (should the `shareCount` reach 0) and adding to your account balance the total value of the sale. When selling a stock, you should use the passed in `price`, not the `price` in your `struct stockProperties`. Additionally, you should not edit the price in `struct stockProperties`.

Removing a stock 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:

$$[e1, e3, e4, \_\_]$$

This function should fail and the no edits to the account should be made if you can't find the stock in your account or the specified sale share count is invalid.

If you are successful in selling the stock, then return `SUCCESS`, otherwise return `ERROR`.

4.4.6 Sort Stock

```
void sortStockInvest(void);
```

Sort the account's stock properties array by the stock's total value. The stock properties array should contain the stocks in the order of increasing stock value. In other words, the stock with the lowest total value should appear first in the sorted array.

Remember that each stock's total value is calculated by multiplying its price by its share count.

Example:

```
Unsorted:   [{sectorType: 'TECHNOLOGY', price: 40, numShares: 100},
             {sectorType: 'CRYPTO', price: 30, numShares: 50},
             {sectorType: 'CRYPTO', price: 100, numShares: 100},
             {sectorType: 'ENERGY', price: 60, numShares: 20},
             {sectorType: 'CRYPTO', price: 40, numShares: 70}]

Sorted:     [{sectorType: 'ENERGY', price: 60, numShares: 20},
             {sectorType: 'CRYPTO', price: 30, numShares: 50},
             {sectorType: 'CRYPTO', price: 40, numShares: 70},
             {sectorType: 'TECHNOLOGY', price: 40, numShares: 100},
             {sectorType: 'CRYPTO', price: 100,numShares: 100}]
```

4.4.7 Sort Stock Sector

```
void sortStockSector(void);
```

Sort the account's stock properties array by the type of sector. The ordering should be such that all `CRYPTO` stocks appear first in the sorted array, followed by all `ENERGY` stocks, and finally all `TECHNOLOGY` stocks at the end.

Example:

```
Unsorted:   [{sectorType: 'TECHNOLOGY', price: 40, numShares: 100},
             {sectorType: 'ENERGY', price: 30, numShares: 100},
             {sectorType: 'CRYPTO', price: 100, numShares: 100},
             {sectorType: 'ENERGY', price: 60, numShares: 100},
             {sectorType: 'CRYPTO', price: 20, numShares: 100}]

Sorted:     [{sectorType: 'CRYPTO', price: 100,numShares: 100},
             {sectorType: 'CRYPTO', price: 20, numShares: 100},
             {sectorType: 'ENERGY', price: 30, numShares: 100},
             {sectorType: 'ENERGY', price: 60, numShares: 100},
             {sectorType: 'TECHNOLOGY', price: 40, numShares: 100}]
```

4.4.8 Total Investment

```
float totalInvestment(void);
```

Return the total value of each stock in your account. If you have no stocks in your account, then return 0.

Remember, each stock's value is calculated by multiplying its price by its share count.

4.4.9 Sector Investment

```
float sectorInvestment(enum sectorType sector);
```

Return the total value of each stock of the provided sector type in your account. If you have no stocks of that type in your account then return 0.

Remember, each stock's value is calculated by multiplying its price by its share count.

NOTE: Check section 4.4 "Program 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**. In Interactive Terminal mode, you can access a terminal within Docker without having to go through your web browser.
2. Run the following command in your terminal to access Docker's terminal much more easily: `./cs2110docker.sh -it`.

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 the 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!

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

Executing `./tests` will run all the test cases and print out the percentage of tests passed (**not your score for the project**), 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
  ```

5.3 Write Your Own Tests

In your Project 4 directory there is a file named `main.c`. This file can be used to write your own tests. The `main` method provided can be executed with the following command.

`make student`

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

```
int main(void) {
    int a = 3;
    int b = 3;
    int test_value = divide(a, b);
    printf("Expected Return Value: 1\nActual Return Value: %d\n", test_value);

    return 0;
}
```

Then run `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

Submit the following files to Gradescope by the due date:

- `collaborators.txt`
- `part1-functions.c`
- `part2-investments.c`

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 (Ubuntu 20.04) 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.

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 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.

8.5 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.**