CS 2110 Homework 7
Intro to C

Alice, Corey, Nick, Pranav, Izabela

Fall 2021

Contents

1 Overview

1.1 Purpose

The purpose of this assignment is to introduce you to basic C programming, building on your knowledge of assembly. This assignment will familiarize you with C syntax and how to compile, run, and debug C programs. You will become familiar with how to work with strings, arrays, pointers, and structs in C. You will understand the relationship in C between arrays and pointers, including pointer arithmetic (think about how arrays are stored in memory in assembly). You will also become familiar with how to use a Makefile to automate the compilation of your program.

1.2 Task

In this assignment, you will be helping to write new software for Ash Ketchum's Pokédex to help him keep track of and manage his Pokémon.

You will write C functions to add (catch) a Pokémon, remove (release) a Pokémon, update a Pokémon's information, compare Pokémon, as well as sort your Pokédex. Your job is to make sure that certain rules are being followed; for some functions you will have to signal whether or not it completed successfully. See the Detailed Instructions section for more details on the specific requirements for each function.

You will write your C code in two files, `my_string.c` and hw7.c. In `my_string.c`, you will write your own implementations of common C library functions for working with strings: `strlen()`, `strncmp()`, and `strncpy()`. In `hw7.c`, you will implement the functionality as mentioned above regarding the Pokédex. Please see the Detailed Instructions below for more, well, detailed instructions.

Take a look at the sections on Makefiles and Testing and Example Output for more info on how to compile and test your program.

While doing the homework, you may find it helpful to draw diagrams of memory locations, as you did with assembly programming. How are arrays represented in memory? How can you use pointers to find the address of `array[i]`? How are arguments passed to a function and results returned using the stack frame? Remember, C functions are pass by value (push copies of the values of arguments on the stack), just like subroutines in assembly.

1.3 Criteria

Your C code must compile without errors or warnings, using the provided Makefile. Your array of structs should be populated correctly at the end of the program. Your helper functions in `my_string.c` must all be implemented correctly, producing the same behavior for test cases as the equivalent library functions from `string.h`.

2 Detailed Instructions

2.1 "my_string.c" functions

The first part of this homework is to implement three very common C string library functions:

- `my_strlen`: compute the length of a null-terminated string
- `my_strncmp`: compare two strings, up to at most n characters
- `my_strncpy`: copy at most n characters from a source string to a target memory location

The caveat is that you must implement these functions **using only pointer notation**. That is, you cannot use array indexing notation, such as `str[i] = 'a'`. This restriction only applies in the `my_string.c` file. We recommend implementing these functions first so you are able to use these functions as you move on with the assignment.

You will notice that many of the arguments in the `my_string.c` and `hw7.c` files are of type `const char *`. This is a pointer to a char that is constant. This means that you cannot edit any of the characters to which a `const char *` points to. If you attempt to do so, using something like `*pointer = 'c'`, you will get a compile error. If `const` does not precede a `char *` declaration, you can edit the characters to which it points to.

In order to understand the functionalities of these three library functions, you need to take a look at their man page (i.e. manual page). See the section on Man Pages for more info.

Some notes:

1. **You are NOT allowed to use array notation in this file. All functions should be implemented using pointers only!** Think about how arrays and pointers correlate with each other in C. Again, this restriction only applies to this file.
2. You are **NOT** allowed to use any of the standard C string libraries (e.g. `#include <string.h>`).
3. Although expressions passed in to this program have a min/max length, the string functions should not have a boundary on any arguments passed in.
4. For `my_strncmp`, you do not need to return a specific number, as long as it follows the description in the man page.

2.2 hw7.c

The second part of this homework is to implement several C functions within the hw7.c file. You will be primarily interacting with the `pokedex` array as well as the global `size` variable. The `pokedex` array must not have gaps between Pokémon (must be contiguous). The size variable will be used to keep track of the number of Pokémon within the aforementioned array. You should update this variable whenever appropriate. Additionally, you may assume that when giving a Pokémon's species name, it will never exceed the maximum length as specified in the header file.

- `catchPokemon(const char *nickname, int pokedexNumber, double powerLevel, const char *speciesName)`:

  In this function, you will add a `pokemon` struct with the given `nickname`, `pokedexNumber`, `powerLevel`, and `speciesName` to the end of the class array. If the given nickname's length (including the null terminator) is above `MAX_NICKNAME_SIZE`, truncate the name to be `MAX_NICKNAME_SIZE`. Be mindful of how strings are represented in C! If the given speciesName's length is smaller than `MIN_SPECIES_NAME_SIZE`, or if the Pokémon with the given nickname already exists in the Pokédex array, or if adding the Pokémon

would cause the number of Pokémon to exceed the maximum size of the Pokédex, do not create and add the Pokémon. Return `SUCCESS` when you are able to successfully create and add the Pokémon, otherwise return `FAILURE`.

- `updatePokemonNickname(struct pokemon p, const char *nickname)`:

  In this function, you will update a Pokémon so that their nickname is updated to the nickname parameter passed into this function. The Pokémon to be updated will be the Pokémon in the Pokédex array with the same nickname as `p`. If the updated nickname's length (including the null terminator) is above `MAX_NICKNAME_SIZE`, truncate the nickname to be `MAX_NICKNAME_SIZE`. If you are able to successfully update the Pokémon's nickname, return `SUCCESS`. If you are unable to find the Pokémon, return `FAILURE`.

- `swapPokemon(int index1, int index2)`:

  In this function, you will swap the position of a Pokémon at index1 with the position of the Pokémon at index2. If the indices are invalid (such as a negative index or an index beyond the current size of the Pokédex) do not swap. On successful swap, return `SUCCESS`. Otherwise, return `FAILURE`;

- `removePokemon(struct pokemon p)`:

  In this function, you will remove a Pokémon with the same nickname as the passed in `pokemon` struct. You must maintain array contiguity when removing. This means that there must be no gaps between Pokémon within the `pokedex` array. On successful removal, return `SUCCESS`. If you cannot find the Pokémon with the given nickname, return `FAILURE`.

- `comparePokemon(struct pokemon p1, struct pokemon p2)`:

  In this function, you will compare two Pokémon by using Professor Oak's methodology for sorting Pokémon, Pokédex number and nickname. If the `pokedexNumber` of p1 is less than that of p2, return a negative number. If it is greater, return a positive number. If the `pokedexNumber` is the same, then compare the `nickname` of the two Pokémon as a tiebreaker. If the `nickname` of p1 is greater, return a positive number; if it is lesser, return a negative number, and if they are the same, return 0.

- `sortPokemon(void)`:

  In this function, you will sort the `pokedex` array of Pokémon by using any sorting algorithm of your choice. The result should be in ascending order. The ordering of the sorted array is determined by `comparePokemon`.

2.3 hw7.h

Within this file, you be provided with the function prototypes, the definition of the `pokemon` struct, as well as various macros that you can use in your implementation of the functions in `hw7.c`.

2.4 Makefiles and Testing

If you want to write manual tests of your functions, you are allowed to modify `main.c` to make custom invocations of your program. For example, you may want to create a new helper method that will print out the contents of the class array within `hw7.h`, implement it `hw7.c`, and call it in `main.c`. However, if you choose to create extraneous helper methods to help debug **make sure to remove them when submitting to the autograder**. Editing `main.c`, however, should not interfere with the autograder.

Since your program is connected to an autograder, it's a little difficult to compile it by hand. To help you out with compiling and running tests, we've provided you with a Makefile.

Make is a common build tool for abstracting the complexity of working with compilers directly. In fact, the PDF you're reading now was built with a Makefile! Makefiles let you define a set of desired targets, their prerequisites, and sets of directives to build those targets. In all of our C assignments (and also in

production level C projects), a Makefile is used to compile C programs with a long list of compiler flags. We have already provided you a Makefile for this homework, and although you are not being tested on it, we highly recommend that you take a look at this file and understand the `gcc` commands and flags used to understand how to compile C programs. If you're interested, you can also find more information regarding Makefiles here.

To test your code manually, compile your code using `make` and run the resulting object file with the command-line arguments of your choice.

Keep in mind that you should run all commands inside the Docker terminal. We highly recommend running the usual script as follows to immediately get a terminal:

```
./cs2110docker.sh -it
```

If you use your own Linux distribution/VM, make sure you have the `check` *unit test framework installed. However, keep in mind that your code will be tested on Docker.*

Below is an example of manually testing your code.

```
# Clean up all compiled output
$ make clean

# Recompile the hw7 executable
$ make hw7

# Run the hw7 executable
$ ./hw7
```

To run the autograder, see the Autograder section.

3 Useful Tips

3.1 Man Pages

The `man` command in Linux provides "an interface to the on-line reference manuals." This is a great utility for any C and Linux developer for finding out more information about the available functions and libraries. In order to use this, you just need to pass in the function name to this command within a Linux (in our case Docker) terminal.

For instance, entering the following command will print the corresponding man page for the strlen function:

```
$ man strlen
```

Additionally, the man pages are accessible online at:

```
http://man.he.net
```

NOTE: You can ignore the subsections after the "RETURN VALUE" (such as ATTRIBUTES, etc) for this homework, however, pay close attention to function descriptions.

3.2 Debugging with GDB and printf

We highly recommend getting use to "`printf` debugging" in C early on.

Moreover, If you run into a problem when working on your homework, you can use the debugging tool, GDB, to debug your code! Former TA Adam Suskin made a series of tutorial videos which you can find here.

Side Note: Get used to GDB early on as it will come in handy in any C program you will write for the rest of 2110, and even in the future!

When running GDB, if you get to a point where user input is needed, you can supply it just like you normally would. When an error happens, you can get a Java-esque stack trace using the backtrace(bt) command. For more info on basic GDB commands, search up "GDB Cheat Sheet."

4 Checking Your Solution

Important Notes:

1. All non-compiling homework will receive a zero (with all the flags specified in the Makefile/Syllabus).
2. **NOTE: DO NOT MODIFY THE HEADER FILES.**

   You must place any code elements you define (structs, macros, function declarations, etc.) in the C FILES. Usually placing those definitions in `.h` files would be good practice, but for this assignment you are not turning them in, and so the declarations would be lost when submitting.

To run the autograder locally (without GDB):

```
# To clean your working directory (use this instead of manually deleting .o files)
$ make clean

# Compile all the required files
$ make tests
```

```
# Run the tester executable
$ ./tests
```

This will run all the test cases and print out a percentage, along with details of the **failed test cases**.

Other available commands:

- To run tests without gdb:

```
# Run all tests
$ make run-case

# Run a specific test
$ make run-case TEST=testCaseName
```

- To run tests with gdb:

```
# Run all tests in gdb
$ make run-gdb

# Run a specific test in gdb
$ make run-gdb TEST=testCaseName
```

The output file will **ONLY** be graded on Gradescope.

TA Note: Since C autograders can sometimes print out a lot of info, it might be a good idea to pipe the output to a file and investigate the content of the file instead! Use Gradescope for a cleaner output or run tests individually when debugging as mentioned above.

Many test cases are randomly generated and your code should work every time we run the autograder on it, however, there's no need to submit to Gradescope multiple times once you get the desired grade.

We reserve the right to update the autograder and the test case weights on Gradescope or the local checker as we see fit when grading your solution.

5 Deliverables

Please upload the following files to Gradescope:

1. `my_string.c`
2. `hw7.c`

The homework is due Wednesday, November 3rdth at 11:59 P.M.

Note: Please do not wait until the last minute to run/test your homework; history has proven that last minute turn-ins will result in long queue times for grading on Gradescope. You have been warned.

6 Appendix

6.1 Appendix D: Rules and Regulations

6.1.1 General Rules

1. Starting with the assembly homeworks, any code you write should be meaningfully commented for your benefit. You should comment your code in terms of the algorithm you are implementing; we all know what each line of code does.
2. 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.
3. Please read the assignment in its entirety before asking questions.
4. Please start assignments early, and ask for help early. Do not email us the night the assignment is due with questions.
5. If you find any problems with the assignment, it would be greatly appreciated if you reported them to the TAs. Announcements will be posted if the assignment changes.

6.1.2 Submission Conventions

1. Unless otherwise noted, all files you submit for assignments should have your name somewhere near the top of the file as a comment.
2. When preparing your submission, you may submit the files individually to Canvas/Gradescope. You can create an archive by right clicking on files and selecting the appropriate compress option on your system. Both ways (uploading raw files or an archive) are exactly equivalent, so choose whichever is most convenient for you.
3. Do not submit compiled files (`.class` files for Java code or `.o` files for C code). Only submit the files we ask for in the assignment.
4. Do not submit links to files. The autograder will 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.

6.1.3 Submission Guidelines

1. You are responsible for turning in assignments on time. This includes accounting for unforeseen circumstances. If you have an emergency let us know **IN ADVANCE** of the due time, and provide 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 responsible for ensuring that you have turned in the correct files. After submitting, be sure to download your submission into a brand new folder and test that it works. What you turn in is what we grade; there are no excuses if you submit the wrong files. In addition, your assignment must be turned in via Canvas/Gradescope. Under no circumstances whatsoever will we accept any email submissions of assignments (Note: if you were granted an extension, you will still turn in the assignment over Canvas/Gradescope).

6.1.4 Syllabus Excerpt on Academic Misconduct

Academic misconduct is taken very seriously in this class. Quizzes, timed labs and the final examination are individual work.

Homework assignments are collaborative. In addition, many, if not all, homework assignments will be evaluated via demo or code review. During this evaluation, you will be expected to be able to explain every aspect of your submission. Homework assignments will also be programatically examined to find evidence of unauthorized collaboration.

What is unauthorized collaboration? Each individual programming assignment should be written by you. You may work with others, but each student should be turning in their own version of the assignment. Submissions that are essentially identical will receive a zero and will be referred 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.

You are expressly forbidden to supply a copy of your homework to another student via electronic means. This includes simply emailing it to them so they can look at it. If you supply an electronic 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 site which would allow other parties to obtain your code, including, but not limited to, public repositories (GitHub), Pastebin, etc. If you would like to use version control, use `github.gatech.edu`.

6.1.5 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 shouldn't 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.

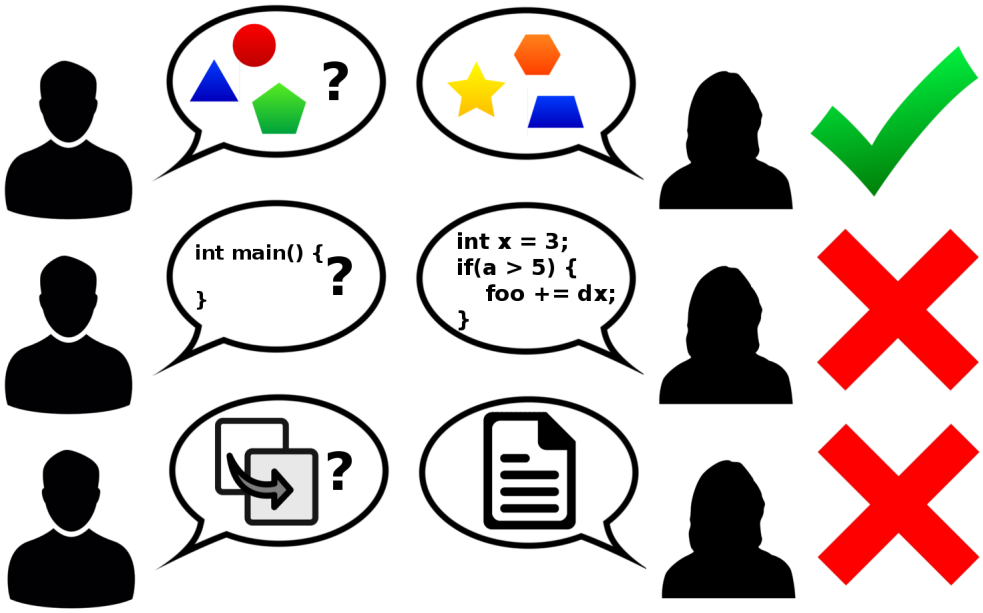

Figure 1: Collaboration rules, explained colorfully