Lab 6: Review

CSE/IT 113L

NMT Department of Computer Science and Engineering

"A computer would deserve to be called intelligent if it could deceive a human into believing that it was human."

— Alan Turing

"We are showing people that engineering isn't just the physics of how a transistor works. It's using the technology, and being creative with it, and solving problems that people have or creating something beautiful with it. And I think that's opening up engineering to a whole new group of people that maybe never would have thought that engineering was for them."

— Limor Fried

"I don't like the feeling, but I've got to say that a little fear makes me a more focused, more responsible programmer."

— Kent Beck

Contents

1 Introduction

This lab focuses on arrays of structures.

2 Overview

This lab is an application of ideas – arrays, structures, array of structures, enumerations – learned in previous labs. If you run into problems, review the material from previous labs and/or the book.

3 Lab Specifications

The following subsections describe what you will write and submit for this lab. **Check that you have met all of the requirements for this lab before you submit it for grading, as the lab will be counted as late if you submit or resubmit it after the due date.**

3.1 Requirements

You must complete the Zombie Apocalypse program (Section 3.2). **Use a Makefile, similar to the one you made in Prelab 6 to compile your program in this lab.** Show the Makefile compiling your program with no errors in the **l6.script** file.

3.2 Zombie Apocalypse

This week you will write a program to record a *zombie apocalypse log*.

The log allows you, in the event of an actual zombie apocalypse, to keep track of the zombies that you encounter. You will enter whether you encountered a dead zombie or one that was still alive (or *undead* if you prefer). If you encounter a dead zombie, you will be asked to take note of how many toes it had. (Remember: zombies were most likely human before, who on average would have had around 10 total toes.) If the zombie was alive when you encountered it, you can record how many milliliters of blood oozed from its body after you eliminated it. (The average human has around 4,500 - 5,700 mL of blood in their body, I would imagine Zombies have less.)

You will also record the day and time that you encountered each zombie.

For this lab, you will be using a structure that looks like this:

```
1 struct zombie {
2         enum {MONDAY = 1, TUESDAY, WEDNESDAY, THURSDAY,
```

```
                FRIDAY, SATURDAY, SUNDAY} day;
        union {
            float blood;
            int toes;
        } note;
        int hour;
        int min;
        int sec;
        char dead;  /*'y' if dead, 'n' if alive*/
};
```

Your program should display the following menu when run:

```
1) Enter new zombie information
2) Display zombie information
3) Return to fighting zombies (exit)
>>
```

If you type 1 and hit enter, your program prompts you like this:

```
Was the zombie found dead? Y or N
>>
```

You can only enter one or the other because the memory we are using is very expensive. We only have room for one. At this point you must enter either Y or N. Then a corresponding question will follow. This is an example of the code running.

```
1) Enter new zombie information
2) Display zombie information
3) Return to fighting zombies (exit)
>> 1

Was the zombie found dead? Y or N
>> Y

Please enter the number of toes the zombie had:
>> 7

Please choose the day this zombie was encountered:
1) Monday
2) Tuesday
3) Wednesday
4) Thursday
```

```
5) Friday
6) Saturday
7) Sunday
>> 5

Enter the time this zombie was encountered.
Separate hours, minutes, and seconds with colons
(HH:MM:SS):
>> 12:33:42

1) Enter new zombie information
2) Display zombie information
3) Return to fighting zombies (exit)
>> 1

Was the zombie found dead? Y or N
>> N

Please enter the amount of blood that oozed from its body
after you eliminated it (in mL):
>> 1073.32

Please choose the day this zombie was encountered:
1) Monday
2) Tuesday
3) Wednesday
4) Thursday
5) Friday
6) Saturday
7) Sunday
>> 1

Enter the time this zombie was encountered.
Separate hours, minutes, and seconds with colons:
(HH:MM:SS)
>> 01:22:12

1) Enter new zombie information
2) Display zombie information
3) Return to fighting zombies (exit)
>> 2

1. This zombie was found DEAD!
This zombie had 7 toes.
This zombie was sighted Friday at 12:33:42.

2. This zombie was found ALIVE!
```

```
It was drained of 1073.32 mL of blood once eliminated.
This zombie was sighted Monday at 01:22:12.
```

```
1) Enter new zombie information
2) Display zombie information
3) Return to fighting zombies (exit)
>> 3

Stay alert! Keep a watch out for zombies!!

GOODBYE and GOOD LUCK!
```

Notice how when you print out the data, you print either blood or toes. You will need to account for this since the structure contains both variables. (Hint: Use `char dead`)

For this lab, you should only support having five entries (hint: use an array). However, the array is circular, meaning on the 6th entry, it is placed in the first element of the array of zombie structures, etc. You do this to save memory. **(Hint: To do this use the modulo operator).**

Zombie Apocalypse Requirements

Exercise 1 (lab6.c, zombie.c, zombie.h, Makefile, l6.script).
For this lab, you are required to:

- Implement the user interface described above.
  - Read the time stamp from the console – three integers separated by colons – using `fgets` and/or `scanf`.
- Use an array of structures to hold five data collections in a circular buffer.
- Use an enumeration to keep track of which day of the week the zombie was found.
- Use a structure to hold all of the zombie "stats" (see example above).
- Use `fgets` and/or `scanf` for input.
- Except for a few variable declarations, a few `fgets` statements and some `switch` statements for the menu logic, the majority of the code in `main` should be function calls.
- These functions are required for your lab.
  - A **input_time** function to input the day and the time.
  - A **input_toes** function to input the number of toes on a dead zombie

  - A **input_blood** function to input the amount of blood oozing from an undead zombie once it has been eliminated.
  - A **print_zombies** function that displays all of the zombie information you have entered into the database.

- Like last lab, you need to implement a header file for this lab. For this lab, your structure will go in `zombie.h` along with your function prototypes. All of your function implementation will go in `zombie.c`. Your main function goes in `lab6.c`.
- Capture some test output in a script file named `l6.script`. Feel free to use your own test data or simply recreate the code example run-through starting in Section 3.2: Zombie Apocalypse.

4 README

Exercise 2 (README).
Every lab you turn in will include a README file. The name of the file is README with no file extension.The README will always include the following two sections *Purpose* and *Conclusion*:

- **Purpose:** describe what the program does (i.e. what problem it solves). Keep this brief.
- **Conclusion:**
  - What did you learn. What new aspect of programming did you learn in this lab? This is the portion of the lab where you want to be analytical about what you learned.
  - Did the Pair Programming prelab help you in solving this solo lab?
  - Did you encounter any problems? How did you fix those problems?
  - Did your code still contain bugs in your final submission? If so, list those bugs.
  - What improvements might you make? In other words, if you were to refactor your lab code, what would you update?

The conclusion does not have to be lengthy, but it should be thorough.

Some labs will require a special **Pseudo-code** section. If a lab requires a pseudocode section, it will be explicitly mentioned in that lab.

5 Getting Help

If you need help with this Lab, please go to tutoring offered weekdays in Cramer 213 and weekends on discord. Please also ask questions in class to the TA's or instructor. These concepts are important for future labs and exams, so if you don't understand them now it is important to get help!
If you don't understand any part of this lab it is important to get help with it.

Submitting

You should submit your code as a tarball file that contains all the exercise files for this lab. The submitted file should be named (**note the lowercase**)

`cse113_firstname_lastname_lab6.tar.gz`

If you require a refresher on generating a tar archive file, instructions can be found in Section 6 - Creating Tar archives of Prelab 0.

Upload your `.tar.gz` file to Canvas.

List of Files to Submit

Exercises start on page 1.