CMPUT 201 - Winter 2023 Lab 6 Henry Tang – University of Alberta

Concepts covered:

- Pointers
- Pointers and arrays
- Using `gdb`

Prep before the lab:

- Revise slides and reading material from Week 7.
- Revise and understand all demo programs used in class.
- You will need to review slides on `gdb`. If you need to pass arguments to the program, pass them to `run` from inside the debugger.

Exercise overview: This lab exercise has 3 parts. Part 1 is worth 2 points, Part 2 is worth 3 points, and Part 3 is worth 2 points.

General instructions: As with all labs and assignments in this course, your code **must compile on the lab machines without any warnings or errors** using `gcc -Wall -std=c99 ...` Please check the "Labs & Lab Demo Guidelines" page on eClass for further important details for all labs.

Part 1

This exercise is Exercise 5 from Ch11 (p255).

Write a program called `split_time.c` that has the following function:

```
void split_time(long total_sec, int *hr, int *min, int *sec)
```

`total_sec` is a time represented as the number of seconds since midnight. `hr`, `min`, and `sec` are pointers to variables in which the function will store the equivalent time in hours [0,23], minutes [0,59], seconds [0,59] respectively.

Your program should have a `main` function that takes `total_sec` as an argument and then calls `split_time`. The main function should then print the converted time as `HH:MM:SS`.

If the total time exceeds 23:59:59, your program must print the following error message to `stderr`: "Total time limit exceded".

```
henry@ug12:~>./split_time 3720
Converted time = 01:02:00
```

You must create a Makefile to compile your program. As usual, your Makefile must have a `clean` target.

Files that need to be on GitHub:

- `part1/split_time.c`
- `part1/Makefile`

Possible Demo Tasks:

- Show and explain the `split_time` function in your code to your TA
- Compile your program using `make`
- Your TA will tell you what input to run your program with (multiple runs)
- Run `make clean`

Grading:

2 marks for Part 1

- **0.5 marks** for a correct `Makefile`
- **1.5 marks** for proper functionality of `split_time` on all test cases.

Part 2

Use the `gdb` tool to find and fix the problem of the given program, `average.c`. It should output:

```
The average is 864.5
```

when run with the given `input.txt` file as `./average input.txt`. Create a `Makefile` to compile the program.

To show how you used `gdb`, you must create 2 screenshots. One showing the problem you discovered (name the sreenshot 1.png). Two showing the code executing successfully (name the sreenshot 2.png).

Files that need to be on GitHub:

- corrected version of `average.c` (you must call it `part2/average_corrected.c`)
- `part2/Makefile` to compile the program
- folder `part2/screenshots` with the screenshots showing how **you** used `gdb` to debug the program.

Possible Demo Tasks:

- Run `make` on the new program and show your TA that your program behaves correctly now
- Explain which `gdb` features you used

Grading:

3 marks for Part 2 - **0.5 marks** for a correct `Makefile` - **0.5 mark** for a corrected `average_corrected.c` program - **2 marks** for a screenshots on debugging with `gdb`

Part 3

This exercise is based on a variation of Ch12, Programming Project 1, p275.

Write a program called `reverse_half.c` that reads a message, then prints the reversal of the **first half of the message**. Given a message of size *n* (every character in the message counts), the first half of the array would end at index *n/2* (regardless of whether the array is an odd or even size). A message will have at most 100 characters.

Examples:

```
Enter a message: Don't get mad, get even.
Reversal of first half: am teg t'noD

Enter a message: abcdef
Reversal of first half: cba

Enter a message: abcde
Reversal of first half: ba
```

*Your program MUST use pointers and NOT integer array indices (i.e., **do NOT use** a[1], a[5] etc.) to process the character array.

You can NOT use strings in this program.

You will get a 0 on this part if you used integer array indices (a[i]) to loop through the character array. You will also get a 0 on this part if you used strings.

You must create a `Makefile` to compile your program. As usual, your `Makefile` must have a `clean` target.

Files that need to be on GitHub:

- `part3/reverse_half.c`
- `part3/Makefile`

Possible Demo Tasks:

- Run `make clean`
- Run `make` – should not have any errors or warnings
- Show your code to your TA and explain how you used pointers to process the array
- Your TA will tell you what input to run your program with (multiple runs)

Grading:

2 marks for Part 3

- **0.5 marks** for acorrect `Makefile`
- **1.5 marks** for proper functionality of `reverse_half` on all test cases (must use pointers to process the array).