Lab 01: Introduction to C
CSE-031-01 **Points: 20**

Overview

Welcome to CSE31! This lab will help you be familiar with the C programming language and the tools available to you on Linux (and Linux-based) systems. In this exercise, you may do your work on the lab systems directly so to minimize any headaches in the beginning of this semester. To access the Linux lab computers, visit https://remotelab.ucmerced.edu/. It would be a good time this week for exploring your other options (e.g., setting up the coding environment in your own systems).

The goal of this lab is to refresh your programming skills and create simple programs that will display output to the console, read input from the user, use conditional and loop statements in C, and perform some simple error handling.

Note: You need to have a separate program for each of the parts of this lab. When you submit your assignment through CatCourses, make sure that ALL PARTS are included.

Getting Started

In this class, we will use the Linux terminal extensively, so you should get familiar with it as much as possible (there is plenty of information online). We recommend setting up a smart directory structure to save your assignments throughout the semester, but it is up to you about how you want to save your work. We will setup a CSE31 directory on the Desktop, a directory for the lab (i.e. Lab_1) inside it, and a directory inside the lab directory for each part of the lab (i.e. part1, part2, etc...). **Note that all the files shown in green below are the ones you will be submitting for this assignment.**

You must have a clear idea of how to answer the lab activities before leaving lab to receive participation score.

Tutorial in Linux

As a software engineer (or a normal human), it is impossible to remember all the commands for Linux terminal. It is particularly true for those that you don't use Linux often. So, where can you find the resources?

TPS (Think-Pair-Share) activity 1: Perform the following tasks while paired with your classmates assigned by your TA (you will be assigned to groups of **3-4 students**) and record your answers in a text file named `tpsAnswers.txt` under a section labelled "TPS 1" (***you will continue to use this file to record your answers to all the TPS questions that follow in the lab handout***):

1. Record your TPS partners' names.
2. Independently search the internet for 3 online tutorials on how to use Linux terminal.
3. Share your tutorials with your TPS partners.
4. Bookmark your results in the browser of your computer.

Create `Lab_1` directory

TPS activity 2: Discuss questions 1 – 4 with your TPS partners in your assigned group (10 minutes) and record your answers in `tpsAnswers.txt` under a section labelled "TPS 2":

1. How can you open a terminal from your Linux computer?
   a. Can you open more than 1 terminal at the same time?
   b. Why do you think you want to open more than 1 terminal at the same time?
2. In the terminal, how can you tell what are contents inside the current directory (i.e., what is the command)?
3. From your current directory, how can you navigate to `Desktop` directory?

4. While you are in `Desktop`, create a new directory called `CSE31`. How do you do this?

Your TA will "invite" one of you randomly after the activity to share what you have discussed. Now navigate to the directory you have just created (`CSE31`), create another new directory called `Lab_1` here (`Lab_1` is inside `CSE31`). You will be saving all the programs you create today in this directory.

Coding 1: Create `main.c`

Make sure you are in your `Lab_1` directory, type `gedit main.c` and press enter (if you get an error, you may need to install `gedit` by typing `sudo apt install gedit` on the terminal before using it). The `gedit` text editor will be displayed. Note that `gedit` is only available in a Linux system with graphic interface (e.g., if you are using the Linux systems through https://remotelab.ucmerced.edu/). You may want to use other text editors (`vi`, `nano`, `sublime`, etc.) if terminal is the only interface to access a Linux system, or if you are using Windows or a Mac. Ask your TA if you are not sure about this.

Copy the following code to your file:

```
#include <stdio.h>
int main()
{
   printf("Welcome to CSE 31!\n");
   return 0;
}
```

Save your file and exit `gedit`. Congratulations, you have just written your first C program!

The first line, `#include< stdio.h>`, includes a library that allows you perform simple Input/Output (I/O) operations. In this program, the library allows you to use the `printf` statement. In the next line, `int main()`, we start the `main` function, which is a mandatory function for any C program. This is the function that first executes when the program is launched. The contents of the function's code need to be surrounded by curly braces (lines 3 and 6). In line 4, we are printing the text `Welcome to CSE031!` to the screen, ending with a new line (`\n`). Finally, the last line returns from the `main` function, which will stop execution of the program.

Once you have created this file and understood its content, you want to compile the source file (`main.c`) into an executable so that you can run it on your computer. To compile, we will use GCC compiler (not G++ – it is used for C++).

TPS activity 3: Discuss questions 1 – 9 with your TPS partners in your assigned group (15 minutes) and record your answers in `tpsAnswers.txt` under a section labelled "TPS 3":

1. Independently find 2 online references on how to use GCC in a Linux terminal.
2. Share what you have found with your partners and save your results in the bookmark of your browser. You will refer to these references to answer the following questions.
3. What command do you type in the terminal to compile your `main.c`?
4. How do you know if your program has compiled successfully?
5. What does the `-c` flag do in `gcc`?
6. What does the `-g` flag do in `gcc`?
7. How do you change the executable name from `main` to `cselab1`?
8. What happens when you compile your code by typing `gcc main.c` only?
9. Now, let us run the program you have just compiled. What command do you use?

Your TA will "invite" one of you randomly after the activity to share what you have discussed.

Coding 2: Create `punishment.c`

A common punishment for school children is to write out the same sentence multiple times. **Individually** create a C program (`punishment.c`) that will write out the following sentence the number of times specified by the user (without the quotes, unless explicitly mentioned): "`C programming language is the best!`". Your program will ask the user for the number of times to output the punishment phrase using the following prompt (notice the space at the end of the prompt): "`Enter the number of repetitions for the punishment phrase: `". If an invalid value is entered (think about what would constitute an invalid value), your program should output `You entered an invalid value for the number of repetitions!` and continue asking the user for an input till a valid one is entered. To make this program "realistic", it will introduce a typo during a certain repetition. It will ask for the repetition during which to introduce a typo using the following prompt: "`Enter the repetition count when you wish to introduce a typo: `". Once again, you should check that the value entered by the user is valid (think about what would constitute an invalid value). If the value is invalid, display `You entered an invalid value for the typo placement!` and continue asking the user for an input till a valid one is entered. When both inputs are correct, you should display the punishment phrase the correct number of times (`C programming language is the best!`), making sure to change it to `C progranming languoge is the bezt!` (the typo) during the repetition count defined/input by the user.

You will need to use `scanf` to process user inputs. Look it up online to find out how to use it. Please see the sample runs below. Your program must produce an output that **exactly resembles the Sample Runs, including identical wording of prompts, spacing, input locations, etc**.

Sample Runs (user input shown in blue, with each run separated by a dashed line):

```
--------------------------------------------------------------------------------SAMPLE RUN 1
Enter the number of repetitions for the punishment phrase: 4

Enter the repetition count when you wish to introduce a typo: 1

C progranming languoge is the bezt!
C programming language is the best!
C programming language is the best!
C programming language is the best!
--------------------------------------------------------------------------------SAMPLE RUN 2
Enter the number of repetitions for the punishment phrase: 6

Enter the repetition count when you wish to introduce a typo: 3

C programming language is the best!
C programming language is the best!
C progranming languoge is the bezt!
C programming language is the best!
C programming language is the best!
C programming language is the best!
--------------------------------------------------------------------------------SAMPLE RUN 3
Enter the number of repetitions for the punishment phrase: -8
You entered an invalid value for the number of repetitions!
Enter the number of repetitions for the punishment phrase again: 0
You entered an invalid value for the number of repetitions!
Enter the number of repetitions for the punishment phrase again: 2

Enter the repetition count when you wish to introduce a typo: 0
You entered an invalid value for the typo placement!
Enter the repetition count when you wish to introduce a typo again: 3
```

```
You entered an invalid value for the typo placement!
Enter the repetition count when you wish to introduce a typo again: 2

C programming language is the best!
C progranming languoge is the bezt!
```

Coding 3: Create `averages.c`

Create a new program that will ask a user to enter a number repeatedly. This program will calculate 2 averages based on whether the inputted numbers are odd or even: `avg_even` (average of all even numbers) and `avg_odd` (average of all odd numbers). The program will stop when the user enters a '`0`'.

Before writing the program in C code, perform the TPS activity below to write a pseudocode to describe your approach to this problem.

TPS activity 4: Work with your TPS partners in your assigned group to write the pseudocode together (15 minutes) in a text file named `pseudoAverages.txt`. **Make sure to only discuss the pseudocode. You must write the C code individually.** Your TA will "invite" one of you randomly after the activity to share your pseudocode.

Your program must produce an output that **exactly resembles the Sample Runs, including identical wording of prompts, spacing, input locations, etc**.

Sample Runs (user input shown in blue, with each run separated by a dashed line):

```
--------------------------------------------------------------------------------SAMPLE RUN 1
Please enter an integer: 9
Please enter an integer: 4
Please enter an integer: -3
Please enter an integer: 7
Please enter an integer: -8
Please enter an integer: 0

Average of even numbers: -2.00
Average of odd numbers: 4.33
--------------------------------------------------------------------------------SAMPLE RUN 2
Please enter an integer: 3
Please enter an integer: -7
Please enter an integer: -1
Please enter an integer: 11
Please enter an integer: 0

Average of odd numbers: 1.50
--------------------------------------------------------------------------------SAMPLE RUN 3
Please enter an integer: 0

There is no average
```

Collaboration

You must credit anyone you worked with in any of the following three different ways:

1. Given help to
2. Gotten help from
3. Collaborated with and worked together

What to hand in

When you are done with this lab assignment, submit all your work through CatCourses.

Before you submit, make sure you have done the following:

- Your code compiles and runs on a Linux machine (without the need for special libraries).
- Attached `punishment.c`, `pseudoAverages.txt`, `averages.c`, and `tpsAnswers.txt`.
- Filled in your collaborator's name (if any) in the "Comments…" text-box at the submission page.

Also, remember to demonstrate your code to the TA or instructor before the end of the grace period.