Homework 1, TCSS 333 B Spring 2016
Due: Wednesday, Apr. 20, 9:00 a.m.

OBJECTIVE

The objective of this assignment is to give you practice with the basic shell commands in Unix, with constructing a program in C that reads and writes basic data types, and with binary operations and representations of those basic data types.

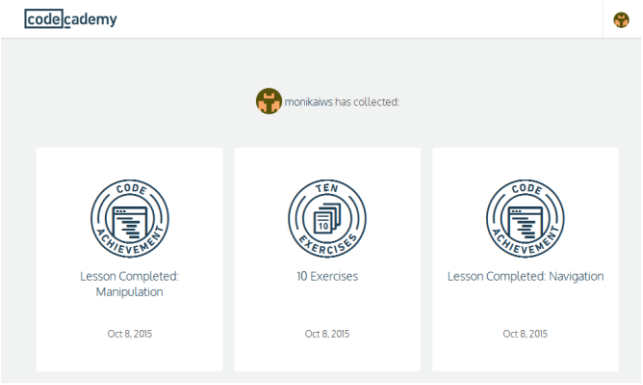
ASSIGNMENT SUBMISSION

To get credit for this assignment, you must

- ✓ complete Unix tutorial (25%)
- ✓ write a program in C (65%)
- ✓ write a test plan (10%)
- ✓ submit your files through Canvas exactly as instructed (naming, compatibility, etc.)
- ✓ submit your assignment on time

UNIX TUTORIAL

Go to Code Academy website (https://www.codecademy.com/courses/learn-the-command-line) and complete two modules from *Learn the Command Line* tutorial: *Navigating the File System* and *Viewing and Changing the File System*. After you complete both tutorials, go to your profile and click on badges. Take a snapshot that shows your name and the badges you earned for completing the two tutorials - upload to Canvas as `jpg`, `gif`, `png` or `pdf`. Your file is to be named `badges.<extension>`. If you can find some other way of showing completed lessons that includes your name, you may take a snapshot of some other screen. For example, these are my badges:



C PROGRAM

Introduction

Error detection is the process of detecting data corruption that may occur during the transmission process. The general idea used in the process is to add some redundancy to the message being sent over some communications channel. Among many techniques used to detect issues, probably the two most basic ones involve the usage of parity bits and the usage of checksums.

A parity bit is simply a bit added to the message that indicates whether number of bits in the message with the value of 1 is even or odd. Let's assume even parity encoding, in which case the number of 1s in the message should always be even. Basically, we count the number of 1s in the original data, and if the result is an even number, then we append a 0 at either end of the message (we will append in front) as we already have an even number of 1s; else we append a 1 to make the total number of 1s an even number. We will use an 8-bit model for our examples:

Original data	Count of 1 bits	Even parity message
0000000	0	00000000
1010001	3	11010001
1000001	2	01000001

When a receiver gets the message, they know that the leftmost bit is used for parity purposes only. They count the number of 1s and if that number happens to be an even number, then they assume the transmission went well and they decode the remaining bits (in our case 7 bits are the actual message). If, however, the receiver finds the odd number of 1s, then they know the error occurred and may ask the sender to send the message again.

Checksum of a message is a value sent in addition to the original message that indicates the sum of the bits used in the message. It could be calculated in a variety of ways but the method that we will follow involves the calculation of a modular arithmetic sum of message 8-bit sub-components. Assuming we have original data comprising of 16 bits, we will first split the data into two 8-bit units and calculate a magnitude-only sum of each 8-bit unit. Then, we will add these sums and perform mod 255 on the resulting total.

Original data	Sum	Sum % 255
00000000 00000000	0 + 0 = 0	0
11111111 11111111	255 + 255 = 510	0
00000001 11111111	1 + 255 = 256	1

When a receiver gets the message, they calculate the checksum of the message and compare to the checksum transmitted with the message. If their sum and the message sum is the same, then they assume the transmission went well, and they decode the message. If it is different, then the receiver knows the error occurred and may ask the sender to send the message again.

Problem Statement

Create a program that asks the user to enter a character or an integer and then calculates ASCII character parity for the character entered by the user or integer checksum for the integer entered by the user. The program should start by giving the user a choice as to which data type / operation they want to perform, where value 1 indicates that a user wants to calculate parity of a character and value 2 indicates that a user wants to calculate the checksum of an integer.

If a user selects 1, then the program prompts for the character, echo prints it to the console, and displays

a) its underlying binary representation,
b) the number of bits of value 1
c) the binary representation of the constructed message

If a user selects 2, then the program prompts for the integer, echo prints it to the console, and displays

a) its underlying binary representation
b) the sum of 8-bit components
c) the checksum (sum % 255)
d) an 8-bit representation of the checksum

Then the programs prompts the user, if they want to repeat the program (use lowercase r as the indicator) and repeats the process as required.

This is a sample run of the program (highlighted parts are user entries):

```
What type of display do you want?
Enter 1 for character parity, 2 for integer checksum: 1
Enter a character for parity calculation:  b
Character: b, Bit representation: 01100010
Number of ones: 3
Even 1 parity for the character is: 11100010

Enter r to repeat, q to quit: r
What type of display do you want?
Enter 1 for character parity, 2 for integer checksum: 1
Enter a character for parity calculation:  Z
Character: Z, Bit representation: 01011010
Number of ones: 4
Even 1 parity for the character is: 01011010
```

```
Enter r to repeat, q to quit: r
What type of display do you want?
Enter 1 for character parity, 2 for integer checksum: 2
Enter an integer for checksum calculation:  -1
Integer: -1, Bit representation: 11111111 11111111 11111111 11111111
Sum of the number is: 1020
Checksum of the number is: 0, Bit representation: 00000000

Enter r to repeat, q to quit: r
What type of display do you want?
Enter 1 for character parity, 2 for integer checksum: 2
Enter an integer for checksum calculation:  1024
Integer: 1024, Bit representation: 00000000 00000000 00000100 00000000
Sum of the number is: 4
Checksum of the number is: 4, Bit representation: 00000100

Enter r to repeat, q to quit: q
```

Specs

- Your program has to use the same choice values (1, 2, r) and follow the same order of operations as shown in the sample run above, although your actual prompt wording could be different
- You have to use bitwise operators in your program
- You are only allowed to use basic primitive types, i.e. arrays and strings are NOT allowed
- Your program is to be written as a single c file named `hw1.c`
- **Your program must compile in gcc gnu 90 – programs that do not compile will receive a grade of 0**
- If you use a math library, then you will need to compile your program with the following command:
  `gcc hw1.c -lm`
- Your program has to follow basic stylistic features, such as proper indentation (use whitespaces, not tabs), whitespaces, meaningful variable names, etc.
- Your program should include the following comments:
  - Your name at the top
  - Comments explaining your logic
  - If your program does not run exactly as shown above, explain at the top how to run your program – you will not receive full credit but at least you may receive some credit rather than none

Test Plan

A test plan in this class is a document that shows the test cases you performed on the final version of your program (values entered to test the outcomes). A test plan in this case should include full code coverage, should list the actual values you entered to test your program, and should be submitted as the following table, in a pdf document named `test.pdf`:

Test case	Reason	Expected Outcome

b	Verifying proper parity count and adjustment	`Binary representation: 01100010`
		`Number of ones: 3`
		`Even 1 parity for the character is: 11100010`

…		
…		

Extra Credit (15%)

Open-ended: extend the program in some fashion – explain in comments at the top of your code. Make sure your extra credit does not modify the original processing but rather adds to it.

Program Submission

On or before the due date, use the link posted in Canvas next to Assignment 1 to submit your tutorial snapshot, your C code, and the test plan. Make sure you know how to do that before the due date since late assignments will not be accepted. Valid documentation file formats are: pdf, jpg, gif, png. Valid program format: a single file named hw1.c