Assignment 2

COMP 2401

Date: September 29, 2017

Due: on October 15, 2017 before 23:55

Submission: Electronic submission on cuLearn.

Objectives:

a. Understanding number representation of integers and floats

b. Bit manipulation of integers

The assignment is divided into two sections a written part, which solves problems with number representations, and a programming part, which looks at, bit manipulation.

Submission

Submission must be in cuLearn by the due date.

Part I – submission in cuLearn.

Part II - Program submission: submit a single tar file with all the .c and .h files and readme files.

Grading (100 pts):

You can earn points as you submit you assignment. Namely, marks will be added as you correctly complete the assignment.

1. (15 pts)
   1.1. (5 pts)Proper documentation, meaning variables,
   1.2. (5 pts) ease of reviewing program
   1.3. (5 pts) compiles with no errors and no warnings.
2. (15 pts) Functions in the bit_manipulation.c were correctly coded.
3. (25 pts) Transmit program is correct –
   3.1. correctly setting the parity bits
   3.2. Providing an example of the input and output of the program
   3.3. Providing a transmitReadMe file that describes how to compile and use the program.
4. (45) Receive program is correct –
   4.1. Correctly completing the function for error correction
   4.2. Correctly completing the function for converting from short integer to char
   4.3. Providing an example of the input and output of the program
   4.4. Providing a receiveReadMe file that describes how to compile and use the program.

Programming – Error Correction Code (3-7 hours)

First: A Note on The Hamming Code: Used for Error Detection and Correction

Typically data is transmitted along a transmission line by moving a byte to a register in a "serial interface chip" which converts the eight bit byte into a stream of bits which are transmitted from the serial port of the computer one at a time. At the other end the bits come into the serial port of the computer at the other end where there is a similar chip which collects the bits coming in serial fashion and then outputs the reconstituted byte to the CPU.

Unfortunately, during the transmission, there is a possibility of noise in the line which can result in bit flipping that is can result in a change in the value of the occasional bit. We assume that the noise level is such that at most one bit will be flipped.

Error Detection: The simplest approach to error detection is the parity check. An extra bit is added to the byte and is set so that the total number of ones in the pattern (including the parity bit) is, say, always even. This is called even parity. The other choice is odd parity, if the total numbers of ones in the pattern is odd. If after transmission, the total number of ones is found to be odd, then an error has been detected. However, it is impossible to say which bit has been flipped and so there is no correction possible.

Error Correction: An approach, which can both detect and correct errors, is called the Hamming code. It involves adding not one but four extra bits to the data to allow both error detection and correction. These bits are called check bits. Using this particular version of the code, one can encode up to 15 bits (including the 4 check bits which can also suffer errors in transmission). There are other versions of the hamming code which can encode larger numbers of bits using more check bits.

For purposes of the algorithm, the four check bits are labelled bits 1, 2, 4 and 8 although they can go anywhere. They are indicated by the letter "c" below. The data bits are labelled with the letter "d" below.

Left to right view of bits

	c	c	d	c	d	d	d	c	d	d	d	d	d	d	d
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Right to left view of bits

d	d	d	d	d	d	d	c	d	d	d	c	d	c	c	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

We are using the ASCII code which uses only eight bits. Therefore, bits in position 0. 13,14,15 are not used.

Setting the error detection bits: If we assume even parity, then the check bits are set using the following equations: The plus signs mean either (mod 2 addition) or (exclusive or). To encode, we calculate the 4 check bits using the following equations:

$X_8 = X_9 + X_{10} + X_{11} + X_{12}$

$X_4 = X_5 + X_6 + X_7 + X_{12}$

$X_2 = X_3 + X_6 + X_7 + X_{10} + X_{11}$

$X_1 = X_3 + X_5 + X_7 + X_9 + X_{11}$

[e.g., $X_8 = (X_9 + X_{10} + X_{11} + X_{12}) \% 2$

or $X_8 = (X_9 \wedge X_{10} \wedge X_{11} \wedge X_{12})$]

Error Detection and Correction: Now the entire group of 15 bits are assumed to be transmitted and received at the other end. And now one has to look for possible errors in transmission. To decode we see if the parity of each of the groups has been maintained. To decode, form the mod 2 sums or the EXCLUSIVE OR's of the following:

$X_8 + X_9 + X_{10} + X_{11} + X_{12}$ should be zero (checksum for X_8)

$X_4 + X_5 + X_6 + X_7 + X_{12}$ should be zero (checksum for X_4)

$X_2 + X_3 + X_6 + X_7 + X_{10} + X_{11}$ should be zero (checksum for X_2)

$X_1 + X_3 + X_5 + X_7 + X_9 + X_{11}$ should be zero (checksum for X_1)

Suppose the checksums for X_8 and for X_2 both give one rather then zero and the other two check sums give the correct value of zero. Then the bit in error is found by forming the following binary number:

X_8	X_4	X_2	X_1	
1	0	1	0	the incorrect bit is thus bit 10.

So to get the incorrect bit you just have to form the following formula:

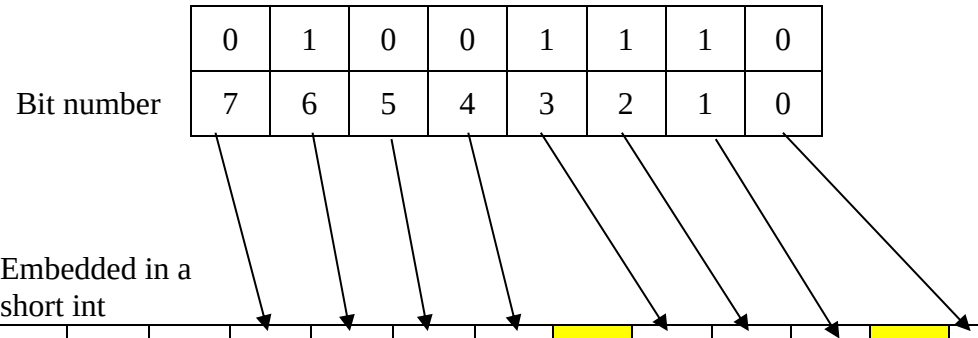
Incorrect bit = 8* (checksum for X_8) + 4* (checksum for X_4) + 2* (checksum for X_2) + (checksum for X_1)

If all check sums are zero then this gives a zero which means that all is well.

Example

The char is ‘N’ = 0x4E

Right to left view of bits



			0	1	0	0		1	1	1		0			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Setting the parity bits

$X_8 = X_9 + X_{10} + X_{11} + X_{12} = 0+0+1+0 = 1$

$X_4 = X_5 + X_6 + X_7 + X_{12} = 1+1+1+0 = 1$

$X_2 = X_3 + X_6 + X_7 + X_{10} + X_{11} = 0+1+0+1 = 0$

$X_1 = X_3 + X_5 + X_7 + X_9 + X_{11} = 0+1+1+0+1 = 1$

			0	1	0	0	1	1	1	1	1	0	0	1	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Assuming that bit 9 flipped during the transmission

			0	1	0	1	1	1	1	1	1	0	0	1	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Compute the parities after the transmission

$X'_8 = X_9 + X_{10} + X_{11} + X_{12} = 1+0+1+0 = 0$

$X'_4 = X_5 + X_6 + X_7 + X_{12} = 1+1+1+0 = 1$

$X'_2 = X_3 + X_6 + X_7 + X_{10} + X_{11} = 0+1+0+1 = 0$

$X'_1 = X_3 + X_5 + X_7 + X_9 + X_{11} = 0+1+1+1+1 = 0$

Compare the stored parities with the computed parities and determine the bit that needs to be flipped (if any)

X'_8 != X_8 → bitNum+=8

X'_4 == X_4 → do nothing

X'_2 == X_2 → do nothing

X'_1 != X_1 → bitNum+=1

bitNum == 9 and therefore the error occurred in bit number 9 which must be flipped.

			0	1	0	0	1	1	1	1	1	0	0	1	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Tasks

In this assignment you will write two short programs that would simulate the transmission a message over a communication line: one program would simulate the transmission of a message (an array of characters) and one that simulate the receiving of a message.

You will use the skeleton in the files transmit.c and receive.c. Helper functions that manipulate bit data should be written in a file bit_manipulation.c

Coding Instructions:

1) Commenting in Code – as provided in the slides given in class
2) No usage of global variables. All data must be passed or received via function parameters.
3) Write short and simple functions.
4) Suggestion – as you code your small helper functions write small test functions to ensure that the code is correct. This will allow you to focus on the logic of your program without worrying about the simple functions.

bit_manipulation file (15 pts)

The file contains some functions that are required by the main programs.

1. Review the functions in the file and their purpose
2. Complete the code for each of the functions.
3. Add other functions as needed

Transmit program (25 pts)

1. (10 pts) Add a function to bit_manipulation.c that takes as input an a short integer num and returns the number of bits that are set to 1. For example if num is 0x9 then the function returns 2. Here you need to declare the function and also add the prototype to the bit_manipulation.h header file.

2. (15 pts) Using the function that you created in 1 above add the code to the setParityBits function.

3. Use the functions in the bit_manipulation.c file

Receive program (45 pts)

The provided code can already read the encoded message that is provided into an array of integers. The program uses the output from the Transmit program as input to the receive program. Using the provided skeleton do the following:

1. (20 pts) Write a function to collects (unpacks) the bits from the encoded short integer into a character. The function is `int short2Char(short encodedChar, char *c)` which takes an encodedChar (a short integer) and updates the character with the correct bits. Namely, it maps bits 3,5,6,7,9,10,11,12 of the encodedChar onto bits 0,1,2,3,4,5,6,7 of c. Review the code of the function char2Short() in the Transmit program.

2. (25 pts) Complete the code of correctCode using the function that you coded in step 1 of the Transmit program and the functions in bit_manipulation.c
3. Use the functions in the bit_manipulation.c file

Compiling the programs.

Compile the transmit program

Here we have two files for each program (e.g., transmit.c and bit_manipulation.c). Compiling the program will be as follows:

gcc –o tran transmit.c bit_manipulation.c

Compiling for the debugger will be as follows:

gcc -g –o tran transmit.c bit_manipulation.c

Compile the receive program

Here we have two files for each program (e.g., transmit.c and bit_manipulation.c). Compiling the program will be as follows:

gcc –o recv receive.c bit_manipulation.c

Compiling for the debugger will be as follows:

gcc -g –o recv receive.c bit_manipulation.c

Execution:

a. Start the transmit program. The program will prompt the user to enter a message. The program will then print the transmitted message as a sequence of short integers.
b. Start the receive message. The program will prompt the user to enter the transmitted message. Here you will have to copy the output from transmit program. The program should output the uncorrected transmitted message and the corrected message.