Assignment 2

COMP 2401

Date: October 2, 2021

Due: on October 17, 2021 before 23:55

Submission: Electronic submission on Brightspace.

Objectives:

a. Bit manipulation of integers (short int and char)
b. Using bit manipulation functions: and (&), or (|), xor (^), left shift (<<), right shift (>>)
c. Using functions - call by value and call by reference
d. Compiling and linking multiple files into a single program.

Submission

Program submission: submit a single tar file, assignment_2.tar, with all the .c and .h files and readme files.

Grading (100 pts):

(15 pts)

1.1. (5 pts) Proper documentation, meaningful variable names, proper comments,

1.2. (5 pts) Ease of reviewing program – program readability, single tar file, etc.

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

In this assignment you will simulate transmission and receiving of a message (ASCII message) over the network. The simulation consists of two parts: a transmit program and a receive program.

Transmit program:

The transmit program simulates the transmission of a user's message over a line (e.g., over the internet). The program obtains a message from the user and prepares it for transmission. In doing so the program the program encodes the message as integers while adding parity bits to ensure that simple transmission errors can be detected and corrected (see note below regarding parity bits). For example, when the user enters the message It is nice day, the program produces a sequence of integers which represent the message to be transmitted. Here each integer contains the corresponding character in the message as well as the required parity bits.

For example:

Transmitting the message:

It is a nice day

Produces the output:

2462 3924 5380 3226 3900 1300 3596 1284 3570 3218 3126 3672 1796 3158 3596 1944.

Here the number 2462 corresponds to the first character I.

Receive program:

The receive program simulates the receiving end of the message. As such the receive program obtains, as input, a sequence of integers that were produced by the transmit program and converts them back to a sequence of characters. If the transmitted message contains errors, then the program corrects the errors and produces the correct message.

In the above example the transmitted message (without correction) contains the following message: It**?**is q nhbu0dq9

After correction it contains the original message

It is a nice day

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 converting the eight-bit bytes into a stream of bits, using a "serial interface chip" which are then 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 receiving computer where a similar chip collects the bits coming in a 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, thus changing 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 even. This is called even parity. For example, the parity for the character 'C' is 1 because the value of 'C' is 0x43, i.ee., three of its bits are set to 1. Therefore, a parity value of 1 means that the total number of bits that are set to 1 is even. If after transmission, the total number of ones is found to be odd, then an error has been detected. However, it is impossible to determine 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.

Right to left view of bits

Bit value	d	d	d	d	d	d	d	c	d	d	d	c	D	c	c	
Bit #	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. Next, one has to look for possible errors in transmission. This is achieved by checking whether 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.

Alternatively, one can compute the parity bits of the transmitted bits again

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

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

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

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

Then comparing the stored parity with the newly computed parity and if the bits that are stored are different then add it to the incorrrectBit, which is initialized to 0.

If (X'_8 != X_8) then incorrrectBit +=8

If (X'_4 != X_4) then incorrrectBit +=4

If (X'_2 != X_2) then incorrrectBit +=2

If (X'_1 != X_1) then incorrrectBit +=1

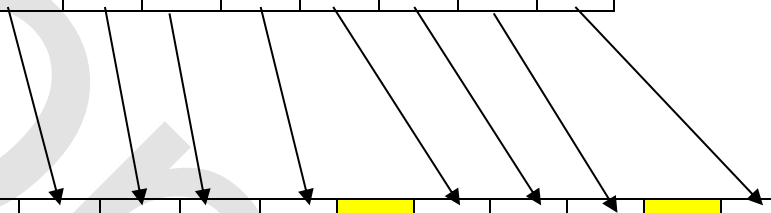
Example

Setting the parity bits

The char is ‘N’ = 0x4E

Right to left view of bits

Bit value	0	1	0	0	1	1	1	0
Bit #	7	6	5	4	3	2	1	0



Embedded in a short int

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

Setting the parity bits

$X_8 = X_9 + X_{10} + X_{11} + X_{12} = (0+0+1+0)\%2 = 1$

$X_4 = X_5 + X_6 + X_7 + X_{12} = (1+1+1+0)\%2 = 1$

$X_2 = X_3 + X_6 + X_7 + X_{10} + X_{11} = (0+1+1+0+1)\%2 = 1$

$X_1 = X_3 + X_5 + X_7 + X_9 + X_{11} = (0+1+1+0+1)\%2 = 1$

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

Correcting a transmission error

Assuming that bit 9 was corrupted during the transmission and its value changed from 0 to 1

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

Compute the parities after the transmission

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

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

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

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

Comparing the bits

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.

Bit value				0	1	0	0	1	1	1	1	1	0	0	1	
Bit #	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 for simulating 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.

The assignment is partially coded (i.e., solutions to some of the functions are provided). This was done to provide an example of approaching a solution. Note, that you do not have to follow the provided code and you can write your own code. However, you cannot change the function declarations.

You will use the skeleton in the files transmit.c and receive.c and bit_manipulation.c.

Files:

a. transmit.c – the file contains the program for transmitting the message. The program prompts the user to input a message and then prepares it for transmission. In doing so it introduces transmission errors.
b. receive.c – this file contains program for receiving the transmitted message. The program prompts the user to enter the transmitted message (you will have to copy the output of the transmission program and provide it as input to the receiving program). The program examines the transmitted message and corrects it if needed. In doing so it prints out the message before the message was corrected and after it was corrected.
c. bit_manipulation.c. this file contains some helper functions for manipulating bits. The helper functions are used by the two programs (transmit.c and receive.c). If you need additional helper functions, then code them in this file.

There are also two test programs: test_transmit and test_receive that you can use to either produce input for your receive program (namely use the test_transmit to produce output) or test the output of your transmit program (namely, by submitting it as input to the test_receive program).

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 six functions that are required by the main programs.

```
int isCharBitSet(char c, int bitNum);
void setCharBit(int bitNum, char *c);
int isShortBitSet(short num, int bitNum);
void setShortBit(int bitNum, short *num);
void flipBitShort(int bitNum, short *num);
int countBits(short num);
```

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. Using the functions in bit_manipulation.c complete the code to the setParityBits function.
2. If needed code additional helper functions in bit_manipulation.c

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) Code the function void short2Char(short encodedNum, char *c) for unpacking the bits from the encoded short integer into a character. Namely, it takes the short integers encodedChar and 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.

1. (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
2. Use the functions in the bit_manipulation.c file. If additional helper functions are needed code them in this 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., receiv.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.

   Here is an example of the transmit program execution (user input is shown in red)

   Please enter a message to transmit: It is a nice day

   Transmitted message (short integers):

   2462 3924 5380 3226 3900 1300 3596 1284 3570 3218 3126 3672 1796 3158 3596 1944

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.

   Here is an example of the receive program execution (user input is shown in red)

   Please enter the transmitted message: 2462 3924 5380 3226 3900 1300 3596 1284 3570 3218 3126 3672 1796 3158 3596 1944

   Transmitted message:

   It�is q nhbu0dq9

   Corrected Transmitted Message:

   It is a nice day