Assignment 2

COMP 2401

Date: October 8, 2018

Due: on October 21, 2018 before 23:55

Submission: Electronic submission on cuLearn.

Objectives:

a. Bit manipulation of integers
b. Using bit manipulation functions: and (&), or (|), xor (^), left shift (<<), right shift (>>)
c. Using functions - call by value and call by reference
d. Using strings
e. Using i/o functions (printf and scanf)
f. Compiling and linking multiple files into a single program.

1 Submission

Program submission: submit a single tar file with all the .c and .h files and readme files. The file name is a1.tar

2 Grading (100 pts):

1. (10 pts)
   1.1. (5 pts) Ease of reviewing program – program readability, single tar file, etc.
   1.2. (5 pts) Compiles with no errors and no warnings.
2. (50 pts) Functions in the bit_manipulation.c were correctly coded.
   2.1. (10pts) rotateLeft()
   2.2. (10pts) rotateRight()
   2.3. (10pts) translateChar()
   2.4. (10pts) revTranslateChar()
   2.5. (10pts) countBits()
3. (20 pts) Encrypt program is correct –

3.1. (10pts)Correctly coding the function for encryptChar()

3.2. (5 pts)Providing an example of the input and output of the program

3.3. (5 pts)Providing an encryptReadMe file that describes how to compile and use the program.

4. (20) Decrypt program is correct –

4.1. (10pts)Correctly coding the function for decryptChar()

4.2. (5 pts)Providing an example of the input and output of the program

4.3. (5 pts)Providing a decryptReadMe file that describes how to compile and use the program.

3 Program execution example

In this assignment you will code two programs: one that will encrypt a message and simulate transmission by printing to the screen; and one that will accept an the transmitted (encrypted) message, decode it and display the decrypted message. The message is an ASCII message.

3.1 Encrypt program:

The encrypt program obtains a message from the user, the encryption keys and then produces an encrypted message. For example, when the user enters the message It is nice day, the program produces a sequence of numbers which represent the message to be transmitted.

For example: if the message is This is a rainy day and the encryption key is 63 then the program would produce 6 49 -17 98 -71 -76 47 -30 50 60 -44 115 -105 -83 -27 84 12 18 74

Example of execution

Please enter a message to encrypt: This is a rainy day

Please enter the encryption key: 63

Encrypted message:

6 49 -17 98 -71 -76 47 -30 50 60 -44 115 -105 -83 -27 84 12 18 74

3.2 Decrypt program:

The decrypt program obtains the encrypted message, the encryption key and then prints the decoded message.

For example if the input is 6 49 -17 98 -71 -76 47 -30 50 60 -44 115 -105 -83 -27 84 12 18 74

and the key is 63 then program would produce This is a rainy day

Example of execution

Please enter the encrypted message: 6 49 -17 98 -71 -76 47 -30 50 60 -44 115 -105 -83 -27 84 12 18 74

Please enter the encryption key: 63

Transmitted message:

This is a rainy day

4 Overview and Rules

4.1 Encryption

The encryption program consists of three steps that are aimed at disguising the message: Character shifting, and two simple bit scrambling steps (circular bit shift and bit xor).

The user will provide a single key for the encryption.

a. **Character shifting** – here the program will "shift" the character by a computed amount specified by the user. This is known as Caesar Cipher (see https://en.wikipedia.org/wiki/Caesar_cipher). Here we will be using mode 256 which is provided for free by the system.

   Examples:
   1. If the shift amount is 1 then the character 'a' will be 'b', the character 'b' will be 'c' etc. Therefore the message "good bye" will be "hppe czf".

   2. If the shift amount is -1 then the message "good bye" will be "fnnc axd".

   3. The shift amount is large, e.g., 190. Since we are using ASCII code which is in the range of 0-255, then the result of adding the amount 190 to a letter will result in a number that is mod 255. For example: the letter 'a' has the value of 0x61 which is 97. Thus adding 190+97 is 287 which is greater than 255. In this case the letter 'a' will have the value of 32 which is 287 mod 255.

   4. The rules for shifting

      a. The shift amount is computed using the key and the array index of the character

(shift amount = 57*array index + key)

b. **Add** the shift amount to the character

b. **Circular rotation (rotates right or rotate left)** – This is one of first simple bit scrambling step. Here the program will rotate the bits (either right rotation or left rotation).

Examples are:

1. Rotate right by 3 bits– here the system will rotate the char so that three least significant bits at position 0, 1 and 2 will be at become bits at position 5, 6, and 7 respectively. If char bits are 10010101 (0x95) then the result of rotate right by 3 will be 10110010

2. Rotate left by 3 bits– here the system will rotate the char so that three most significant bits at position 5, 6 and 7 will be at become bits at position 0, 1, and 2 respectively. If char bits are 10010101 (0x95) then the result of rotate right by 3 will be 10101100

3. Rules for rotation

    a. If the number of bits that are set to 1 in the character is **odd** then perform a **rotate right** operation
    b. If the number of bits that are set to 1in the character is **even** then perform a **rotate left** operation
    c. The number of bits to rotate are computed as follows:
    (Character index in the array + number of bits set to 1) % 8

Example: Assume that the key is 57 and that the message is stored in the string *msg* and that msg[12] == 'a'.
The character 'a' is 0x61 == 01100001.

1. Here the number of bits that are set to 1 is 3, which is an odd number. Therefore the system has to perform rotate right operation.
2. The number of bits to rotated is computed as follows: (array index + key) % 8 which is (12+ 3) % 8 = 15 % 8 = 7. Therefore the system has to perform right rotation of 7 bits

01100001 yielding 11000010

c. Xor operation –
This is the second simple bit scrambling step. Here the program will try to disguise the bits by using an xor operation on the character. Here the xor operation will use the key.

Rules for xor operation

1. Xor the character with the key

Example: Assume that the key is 57 and that the message is stored in the string *msg* and that msg[12] == 'a'.

The system will perform msg[12] = msg[12] ^ key which is

msg[12] = msg[12] ^ 57 == 01100001 ^ 00111001 = 01011000

Example

This example shows the evolution of a message throughout each of the steps. Here we assume that the message is "CS", the key is 87 (0x57) and that the message is stored in the string msg.

Step 1 Character shifting

msg[0] = 'C' = 0x43 = 67

The character index is 0 and therefore the character will be translated to

67 + 57*0 + 87 = 154 (0x9A). Note that since the character is a signed integer the value may shown is -102

Therefore msg[0]=0x9A

msg[1] = 'S' = 0x53=83

The character index is 1 and therefore the character will be translated to

83 + 57*1 + 87 = 227 = 0xE3. Note that this number if store in a signed char and therefore the decimal value will be -29

Therefore msg[1]=0xE3

Step 2 Circular rotation

msg[0] = 0x9A

1. The number of bits that are set to 1 in 0x9A (10011010) is 4 and therefore the system has to perform a right rotation operation.
2. The number of bits that must be rotated is

   (0 + 4) %8 = 4%8 = 4

   Therefore the system has to perform left rotation of 4 bits

   0x9A = 10011010 → 10101001

   msg[0] = 0xA9

Msg[1] = 0xE3

1. The number of bits that are set to 1 in 0xE3 (11100011) is 5 and therefore the system has to perform a right rotation operation.
2. The number of bits that must be rotated is

   The character index is 1 and therefore the character will be translated to

   (1 + 5)%8 =6%8 = 6

   Therefore the system has to perform right rotation of 6 bits

   0x8D = 11100011 → 10001111

   msg[1] = 0x8F

Step 3 XOR opeation

msg[0] = 0xA9

msg[0]^key = 0xA9^0x57 = 10101001^01010111 = 11111110=0xFE

yielding msg[0] = 0xFE

Msg[1] = 0x8F

msg[1]^key = 0x8F^0x57 = 10001111^01010111 = 11011000=0xD8

yielding msg[1] = 0xD8

-2 -40

The program should print because the bit combination 0xFE and 0xD8 translate to -2 and -40

4.2 Decryption

The decryption program is the inverse of the encryption program. Therefore it has to operate in reverse order of the encryption program.

The decryption program consists of three steps that are aimed at disguising the message: two simple bit descrambling steps (xor and circular bit shift) and character shifting, and.

The user will provide a single key for the encryption.

a. **Xor operation** –
This is the first simple bit descrambling step. Here the program will try to disguise the bits by using an xor operation on the character. Here the xor operation will use the key.

Rules for xor operation

2. Xor the character with the key

Example: Assume that the key is 57 and that the message is stored in the string *msg* and that msg[12] == 0x58.

The system will perform msg[12] = msg[12] ^ key which is
msg[12] = msg[12] ^ 57 == 01011000^ 00111001 = 01100001 = 0x61

b. **Circular rotation (rotates right or rotate left)** –
This is one of first simple bit scrambling step. Here the program will rotate the bits (either right rotation or left rotation).

Examples are:

4. Rotate right by 3 bits– here the system will rotate the char so that three least significant bits at position 0, 1 and 2 will be at become bits at position 5, 6, and 7 respectively. If char bits are 10010101 (0x95) then the result of rotate right by 3 will be 10110010

5. Rotate left by 3 bits– here the system will rotate the char so that three most significant bits at position 5, 6 and 7 will be at become bits at position 0, 1, and 2 respectively. If char bits are 10010101 (0x95) then the result of rotate right by 3 will be 10101100

6. Rules for rotation

    a. If the number of bits that are set to 1 in the character is **odd** then perform a **left rotation** operation
    b. If the number of bits that are set to 1in the character is **even** then perform **a right rotation** operation
    c. The number of bits to rotate are computed as follows:
    (character index in the array + number of bits set to 1) % 8

Example: Assume that the key is 57 and that the message is stored in the string *msg* and that msg[12] == ‘a’.

The character ‘a’ is 0x61 == 11000010.

3. Here the number of bits that are set to 1 is 3, which is an odd number. Therefore the system has to perform left rotation operation.
4. The number of bits to rotate is computed as follows: (array index + key) % 8 which is (12+ 3) % 8 = 15 % 8 = 7. Therefore the system has to perform right rotation of 7 bits

11000010 yielding 01100001

c. **Character shifting** – here the program will “shift” the character by a computed amount specified by the user. Here we will be using mode 256 which is provided for free by the system.

1. The rules for shifting

    a. The shift amount is computed using the key and the array index of the character (shift amount = 57 * array index + key)
    b. **Subtract** the shift amount from the character

Example

This example shows the evolution of a message throughout each of the steps. Here we assume that the encrypted message is 0xFE 0xD8, the key is 87 (0x57) and that the message is stored in the string msg.

Step 1 XOR operation

msg[0] = 0xFE

msg[0]^key = 0xFE^0x57 = 11111110^01010111 = 10101001=0xA9

yielding msg[0] = 0xA9

Msg[1] = 0xD8

msg[1]^key = 0xD8^0x57 = 11011000^01010111 = 10001111

yielding msg[1] = 0x8F

Step 2 Circular rotation

msg[0] = 0xA9

3. The number of bits that are set to 1 in 0xA9 (10101001) is 4 and therefore the system has to perform a right rotation operation.
4. The number of bits that must be rotated is

(0+4) %8 = 4%8 = 4

Therefore the system has to perform right rotation of 1 bits

0xA9 = 10101001 → 10011010

msg[0] = 0x9A

Msg[1] = 0x8F

3. The number of bits that are set to 1 in 0x8F (10001111) is 5 and therefore the system has to perform a left rotation operation.
4. The number of bits that must be rotated is

   The character index is 1 and therefore the character will be translated to

   (1+5)%8 = 6%8 = 6

   Therefore the system has to perform left rotation of 5 bits

   0x8F = 10001111 → 11100011

   msg[1] = 0E3

Step 3 Character shifting

msg[0] = 0x9A = -102

The character index is 0 and therefore the character will be translated to

-102 – (57*0 +87) = -189 = 67=0x43

Therefore msg[0]=0x43 = 'C'

msg[1] = 0xE3=227=-29

The character index is 1 and therefore the character will be translated to

-29 – (57*1+87) = 0x53

Therefore msg[1]=0x53 = 'S'

5 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.

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

Files:

a. encrypt.c – the file contains the program for encrypting the message. The program prompts the user to input a message and a key. Once received the program encrypts the message and and transmits it (i.e., print it to the screen).

b. decrypt.c – this file contains program for receiving the transmitted message and decoding it. The program prompts the user to enter the transmitted message (you can either copy it or type it) and the key. The program decodes the message using the key prints out the message.

c. bit_manipulation.c. this file contains several helper functions for manipulating bits. The helper functions are used by the two programs (encypt.c and decrypt.c). If you need additional helper functions, then code them in this file.

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

5.1 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 helper functions **write short 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.

5.2 bit_manipulation file

The file contains several 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
4. Functions
    a. rotateLeft() – performs a circular rotation to the left
    b. rotateRight() – performs a cicurlar rotaqtion to the right
    c. translateChar() – translate a char by adding an amount
    d. revTranslateChar() – translates char by subtracting an amount
    e. encryptChar() – encrypts a char by a. ,translate operation, b. circular rotation, c. xor operation
    f. decryptChar() – decrypts a char by a. xor operation, b. circular rotation, c. translate operation
    g. countBits() – counts the number of bits that are set to 1 in a character
5.

5.3 Encrypt program

1. Complete the code I the main program as needed.

2. If additional functions are needed then code hose in the file bit_manipulation.c

5.4 Decrypt program

The provided code can already reads the encoded message that is provided into an array of integers.

You still need to add code in the main program. Review the main function in encrypt.c. The program uses the output from the Transmit program as input to the receive program. Using the provided skeleton do the following:

5.5 Compiling the programs.

Compile the encrypt 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 encrypt encrypt.c bit_manipulation.c

Compiling for the debugger will be as follows:

gcc –g –o encrypt encrypt.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 decrypt decrypt.c bit_manipulation.c

Compiling for the debugger will be as follows:

gcc –g –o decrypt decrypt.c bit_manipulation.c