SPRING 2020

ECE 103 ENGINEERING PROGRAMMING

EXPANDED HOMEWORK HW-4

ECE DEPARTMENT
PORTLAND STATE UNIVERSITY

ECE 103 Expanded Homework HW-4

Problem List

50 points

Text encoding/decoding problem

General Instructions

- Your program must conform to either the ISO C99 or ISO C90 standard.
- Good programming style is important. Comment your program properly.
- Put a title block at the top of each source code file. It should include the course number, homework number, your full name, and a short program description.
- Each function that you write should have a short title block that describes the function's purpose.

- What you need to submit for grading:
  - Program Design Report (format: Word .doc/.docx or pdf)
  - Source code with this file name: **`hw4.c`**
- Store your completed files (source code, reports, etc.) in a single ZIP archive file.
  - Do **not** include the executable file (e.g., a.out or a.exe) or IDE project files.
  - WARNING: If you do not zip your files, you will receive a point deduction.
- Upload your ZIP file to the appropriate D2L Submission Folder by the deadline.

NOTE

For an example of what a typical Program Design Report should look like, refer to the file "Sample_Program_Design_Report.pdf", which is stored on D2L in the *Syllabus and Course Info* module.

Problem Statement

For as long as people have held important secrets, there has been a desire to send encrypted messages that cannot be deciphered by anyone except by the intended recipient. While modern encryption schemes are relatively secure, they are also mathematically highly complex, and writing a correct computer implementation can be very challenging. So instead of subjecting you to that degree of coding difficulty, you will instead investigate a simple "substitution" cipher for protecting a secret message.

- You need to develop a program that repeatedly allows the user to encode a single message, decode a single message, decode an entire protected file, or quit. The cipher will be based on the Lewis Carroll method.

Background

Suppose the original message is encoded using a set of symbols such as the English alphabet[1]. In the substitution method, each symbol in the alphabet is assigned a different symbol[2] according to some predetermined scheme. The encoding process consists of replacing each character in the original message by its associated substitution value. The encoded message can then be sent to the recipient, who can unscramble the message by performing a reverse lookup, but only if they know how the encoding table was constructed.

Example:

Orig	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Subs	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B

In the simple table above, the original alphabet symbols form the top row. The substitution symbols are placed in the second row. They are the same set of characters, but are shifted over by two positions to the left, with the A and B wrapping around on the right side.

- *To encode:* In the first row of the table, find the desired character from the original message, then look downward in the same column to determine its substitution character.

  Original message: `PORTLAND` (P to be replaced by R, O to be replaced by Q, etc.)
  Encoded message: `RQTVNCPF`

- *To decode:* In the second row of the table, find the desired character from the encoded message, then look upward in the same column to determine its original character.

  Encoded message: `RQTVNCPF` (R to be replaced by P, Q to be replaced by O, etc.)
  Original message: `PORTLAND`

[1] English alphabet symbols are the upper case *A* thru *Z* and lower case *a* thru *z* characters. The alphabet can be extended to include things like the numeric symbols 0 thru 9 and certain punctuation marks.
[2] The substitution symbols could be from the same alphabet or from an entirely different one.

If the message is intercepted before it reaches the intended recipient, then as long as that person does not know the exact encoding/decoding table, the message will remain secure. Of course, the extremely simple substitution shown in the previous example could be figured out by a young child and offers little or no security. To get around this problem, more sophisticated substitution algorithms have been developed, including one attributed to noted author Charles L. Dodgson, better known as Lewis Carroll.

For this assignment, a 2-dimensional "lookup" table of substitution values will be used along with a passcode. We also need to handle numeric digits and punctuation in the original message. To do this, we will refer to the ASCII chart shown below.

Table 1: ASCII characters (32 - 126) collating sequence

Dec	Char	Description	Dec	Char	Description	Dec	Char	Description
32		Space	64	@	At	96	`	Grave accent
33	!	Exclamation mark	65	A	Upper case A	97	a	Lower case a
34	"	Quotation Mark	66	B	Upper case B	98	b	Lower case b
35	#	Hash	67	C	Upper case C	99	c	Lower case c
36	$	Dollar	68	D	Upper case D	100	d	Lower case d
37	%	Percent	69	E	Upper case E	101	e	Lower case e
38	&	Ampersand	70	F	Upper case F	102	f	Lower case f
39	'	Apostrophe	71	G	Upper case G	103	g	Lower case g
40	(	Open parentheses	72	H	Upper case H	104	h	Lower case h
41	)	Close parentheses	73	I	Upper case I	105	i	Lower case i
42	*	Asterisk	74	J	Upper case J	106	j	Lower case j
43	+	Plus	75	K	Upper case K	107	k	Lower case k
44	,	Comma	76	L	Upper case L	108	l	Lower case l
45	-	Dash	77	M	Upper case M	109	m	Lower case m
46	.	Full stop	78	N	Upper case N	110	n	Lower case n
47	/	Slash	79	O	Upper case O	111	o	Lower case o
48	0	Zero	80	P	Upper case P	112	p	Lower case p
49	1	One	81	Q	Upper case Q	113	q	Lower case q
50	2	Two	82	R	Upper case R	114	r	Lower case r
51	3	Three	83	S	Upper case S	115	s	Lower case s
52	4	Four	84	T	Upper case T	116	t	Lower case t
53	5	Five	85	U	Upper case U	117	u	Lower case u
54	6	Six	86	V	Upper case V	118	v	Lower case v
55	7	Seven	87	W	Upper case W	119	w	Lower case w
56	8	Eight	88	X	Upper case X	120	x	Lower case x
57	9	Nine	89	Y	Upper case Y	121	y	Lower case y
58	:	Colon	90	Z	Upper case Z	122	z	Lower case z
59	;	Semicolon	91	[	Open bracket	123	{	Open brace
60	<	Less than	92	\	Backslash	124	|	Pipe
61	=	Equals sign	93	]	Close bracket	125	}	Close brace
62	>	Greater than	94	^	Caret	126	~	Tilde
63	?	Question mark	95	_	Underscore			

The amount of available symbols is adequate to handle the content of most messages. What is important to note here is that the symbols are *ordered*, so that each symbol can be associated with a unique and sequentially ascending number.

Lewis Carroll Algorithm

The encoding and decoding table for our version of the Lewis Carroll substitution algorithm is constructed like this:

Table 2: Encoding/Decoding Table

← Column Index →

↓ Row Index →	**sp**	**!**	**"**	**#**	**.**	**.**	**.**	**|**	**}**	**~**
sp	sp	!	"	#	.	.	.	|	}	~
!	!	"	#	$	.	.	.	}	~	sp
"	"	#	$	%	.	.	.	~	sp	!
#	#	$	%	&	.	.	.	sp	!	"
.						.				
.						.				
.						.				
|	|	}	~	sp	.	.	.	y	z	{
}	}	~	sp	!	.	.	.	z	{	|
~	~	sp	!	"	.	.	.	{	|	}

Across the top and along the left side are, in order, the ASCII characters SPACE (**sp**) through TILDA (**~**). Each row in the interior of the table is the same as the previous row, except that each character is shifted one position to the left and the last character of a row is the first character of the preceding row. If you want to see what the full table looks like, download the file named *Lookup_Table.pdf* that is in the HW-4 content module on D2L.

Encoding

To encode text, you choose an arbitrary *passcode* string, which is used as a key to lock or unlock the secret message. As an example, suppose the chosen passcode is **Walrus** and the message to encode is:

Meet me in St. Louis!

Write the passcode, repeated as often as necessary, on top of the text to be encoded:

WalrusWalrusWalrusWal

Meet me in St. Louis!

The pairs of characters **WM**, **ae**, **le**, …, one on top of the other, serve as indexes into the table. The encoded text results from finding the entries in the table that correspond to these pairs. Thus, the entry at row **W** and column **M** is **%**, so the first character of the encoded text is **%**. The entry at row **a** and column **e** is **G**; the entry at row **l** and column **e** is **R**, and so on.

Thus the original message is encoded as

%GRgua=aVauGLol?eiAUm

and can now be transmitted to someone else. To be able to decode this, a person must know the passcode that was used during the encoding phase.

Decoding

The original text can be recovered from the encoded message by using the passcode string to perform a reverse table lookup:

WalrusWalrusWalrusWal

%GRgua=aVauGLol?eiAUm

Once again, pairs of characters are considered, such as **W%**, **aG**, **lR**, and so on. The first character of the pair is still used as the row index in the lookup table. This time, though, the position of the second character *within that row* determines the column index. The character in the first row of the table at that column index is the original character.

As another example, suppose the message is **!#!** and the passcode is **XYZ**. The encoded string will then be: **Y\[**

If you were sent the encoded string, how would you decode it? Set it up like this:

Passcode: **XYZ**
EncString: **Y\[**

Pair **XY** Go to row **X**, find location of **Y**, original value is: **!**
Pair **Y** Go to row **Y**, find location of ****, original value is: **#**
Pair **Z[** Go to row **Z**, find location of **[**, original value is: **!**

	sp	!	"	#	$
X	X	Y	Z	[	\
Y	Y	Z	[	\	]
Z	Z	[	\	]	^

Fragment of full lookup table.

Some Issues

In the Lewis Carroll method, both the sender and receiver must be able to access the same lookup table. The table's construction may be highly deterministic, if the row and column indices must align with a known ordered sequence of symbols. In addition, the security of this technique also depends greatly on the choice of passcode. Ideally, the combination of message and passcode should cover a large area of the table, but if the passcode is too short, or if it contains too many repetitions of the same characters, then clustering will occur. This means the lookup is confined to just a few rows, which makes it easier for an interloper to crack the message. Another problem is that if you can guess some part of the passcode, then a portion of the encoded text will be revealed, which may give enough of a clue to determine the rest of the passcode.

Primary Requirements

The specific requirements for the program are:

1. Declare a 2-D array to hold the table lookup values. Since the array is so big, it is not practical to declare and initialize it at the same time. Instead, call a function (that you will write) in *main* to perform the actual initialization.
    - Option #1: You can make the lookup table array global, if you wish. In this case, you do not need to pass the table to your other functions but just access it directly.
    - Option #2: Alternatively, you can declare the table in *main* and pass it to your functions.
2. Declare the message, passcode, and encoded strings in *main* and pass them to functions as needed.
3. Display a menu of these options (Encode, Decode, Read File, Quit).
    - Prompt the user to enter a letter that corresponds to one of the available options ('e', 'E', 'd', 'D', 'r', 'R', 'q', 'Q' are the acceptable responses).
    - Verify that the entered response is valid.
        - As long as invalid input is entered, print an error message and ask again for input.
4. If the encoding option (E) is selected:
    - Prompt the user for the original message and the passcode.
    - Call a function (that you will write) to perform the actual encoding.
    - Display the encoded message and then redisplay the menu.
5. If the decoding option (D) is selected:
    - Prompt the user for the encoded message and the passcode.
    - Call a function (that you will write) to perform the actual decoding.
    - Display the recovered message and then redisplay the menu.
6. If the read file option (R) is selected:
    - Prompt the user for the name of a file that contains the encoded text.
    - Open the input file for reading
        - If it cannot be opened, then print an error message and exit the program.
    - Read, decode, and display each line that is stored in the file.
7. If the quit option (Q) is selected:
    - Exit the program.

Mandatory functions that you *must* write

Implement each of these functions:

1. **Function that initializes the lookup table**

   Fill each row in the lookup table array with the appropriately shifted ASCII characters. *Hint:* Loops are useful here.

2. **Function that implements the encoding operation**

   Encode a message string by using the passcode and the lookup table. This function should not prompt for input or print out anything.

3. **Function that implements the decoding operation**

   Decode an encoded string by using the passcode and the lookup table. This function should not prompt for input or print out anything.

The *main* function should call (invoke) these functions as needed.

Optional functions

You may write as many additional functions you believe are needed to make your code more efficient, modular, or readable. The choice is up to you.

Additional Requirements

- Wherever appropriate, use macro definitions or const variables for constants.
- The only global variable you may have (if you wish) is the lookup table array.
- The message string, passcode string, encoded string, etc. must be passed as arguments to each of your functions. This will give you practice in declaring parameter variables and passing arguments to functions.
- For the strings, you may assume a maximum size of 160 characters.
- Very briefly describe your encoding/decoding process in your Program Design Report.
- Save your program using this filename: **`hw4.c`**

Sample Run #1

```
Available options:
[e] Encode
[d] Decode
[r] Read file
[q] Quit

Please enter your choice: e

Enter the original message: Walt Disney World
Enter the passcode: StarTrek

Encoded message is: +VNgT7O_BZ[r,bXX8

Available options:
[e] Encode
[d] Decode
[r] Read file
[q] Quit

Please enter your choice: d

Enter the encoded message: +VNgT7O_BZ[r,bXX8
Enter the passcode: StarTrek

Decoded message is: Walt Disney World

Available options:
[e] Encode
[d] Decode
[r] Read file
[q] Quit

Please enter your choice: d

Enter the encoded message: +VNgT7O_BZ[r,bXX8
Enter the passcode: Portland

Decoded message is: Zf[rgU`zqjh}?!isg

Available options:
[e] Encode
[d] Decode
[r] Read file
[q] Quit

Please enter your choice:

Enter the encoded message: +VNgT7O_BZ[r,bXX8
Enter the passcode: StarWars

Decoded message is: Walt|U\kney T!edd

Available options:
[e] Encode
[d] Decode
[r] Read file
[q] Quit

Please enter your choice: q
```

Sample Run #2

```
Available options:
[e] Encode
[d] Decode
[r] Read file
[q] Quit

Please enter your choice: r

Enter name of encoded file: junk
Error: Cannot open file.
```

Sample Run #3

For this run, the file “testfile.txt” contains the following encoded text, and the passcode was “Walrus”:

```
%GRgua=aVauGLol?eiAUm
 GY_e WY\ebXe
(Q_gbUFFlFjULGlHd]NG_f_hQ
```

Here is the actual run output:

```
Available options:
[e] Encode
[d] Decode
[r] Read file
[q] Quit

Please enter your choice: r

Enter name of encoded file: testfile.txt
Enter the passcode: Walrus

Meet me in St. Louis!
Hello, world.
Portland State University

Available options:
[e] Encode
[d] Decode
[r] Read file
[q] Quit

Please enter your choice: q
```

Sample Run #4

```
Available options:
[e] Encode
[d] Decode
[r] Read file
[q] Quit

Please enter your choice: z


That choice is not valid.

Available options:
[e] Encode
[d] Decode
[r] Read file
[q] Quit

Please enter your choice: 9


That choice is not valid.

Available options:
[e] Encode
[d] Decode
[r] Read file
[q] Quit

Please enter your choice: q
```