CSC 322 Systems Programming Fall 2022
Lab Assignment L1: Cipher-machine
Due: Wednesday, September 14

1 Goal

In the first lab, we will develop a system program (called cipher-machine) which can encrypt or decrypt a code using C language. It must be an errorless program, in which user repeatedly executes pre-defined commands and quits when he or she wants to exit. For C beginners, this project will be a good initiator to learn a new programming language. Students who already know C language will also have a good opportunity to warm up their programming abilities and get prepared for the rest of labs.

2 Introduction

A. Basic Commands

The system program provides three basic commands shown in Table 1. The program runs a related function or exits when a user enters a command as seen in Figure 1. Note that commands need parameters. Misspelled command should be handled as an exception, and the program must show an error message. Note that program must be in running and shows the next prompt.

Table 1. Basic Commands

Command	Description
encrypt	Encrypt user's input based on Caesar's cipher. See the section II-B.
decrypt	Decrypt user's input based on Caesar's cipher. See the section II-B.
encode	Encode user's input to get a binary code based on ASCII code. See the section II-C. (extra)
decode	Decode user's binary input to get a character code. See the section II-C. (extra)
exit	Exit the program

The program **should run on Linux machine at *cs.oswego.edu***. Once it is compiled and executed, it draws user to its own computing environment by showing prompt following the format: **your_loginID $**. Figure 1 shows how the prompt looks when the logID is *jwlee*. When a command is correctly executed, output will be printed out immediately. The logID will be printed by using `printf` function.

```
jwlee@altair~ > cc lab1-jwlee.c -o lab1
jwlee@altair~ > ./lab1
jwlee $  exot
[Error] Please enter the command correctly!
jwlee $ encrypt("hello")
ebiil
jwlee $ exit
```

```
jwlee@altair~ >
```

Figure 1. Command Mode

B. Caesar's Cipher: `encrypt/decrypt`

Caesar's cipher is one of cryptography methods, which has a long history, back to Julius Caesar's Roman Empire. He used this simple yet efficient method in sending and receiving private messages with his subordinates and political comrades. The method is based on a substitution cipher technique in which substitute an alphabet letter to a different alphabet letter by a certain rule. Look at the list of alphabets in the Figure 2. In this plaintext each character has a certain positional number, for example "A" has 1, meaning the first character. In the same manner, "B" has 2, "C" has 3, and so on. To get the ciphertext, all positional number is shifted right by a proposed number, 3 in the Figure 2. By the shifting rule, "A" is now in the 4th position, "B" is in the 5th position.

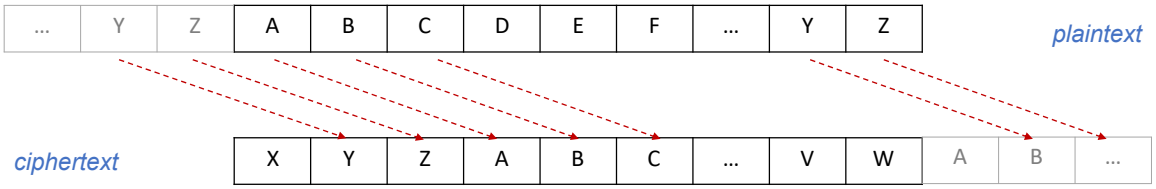

Figure 2. Shifts in Caesar's cipher method

The shifting rule has issues when it shifts alphabets at the end. See Y and Z. After shifting 3, Y which was in 25th position should move to 28th position, but it is out of range of alphabet. Caesar solved it by modular operation, which means the positional number after shifting is modularized by the size of alphabets. For simplicity, let say A is in 0, B is in 1, and so on. Then the encryption formula ***E*** to get cipher text ***C*** from plain text ***P*** is below:

$$C = E(k, P) = (P + k) \ (mod\ 26)$$

In the formula ***k*** stands for the number of shifts and ***k*** is 3 in our example. Similarly, a cipher text can be decrypted by the following formula ***D***:

$$P = D(k, C) = (C - k) \ (mod\ 26)$$

A cipher message can be decrypted by person who knows ***k***. It makes the method secure. For example, the following sentence is encrypted to a cipher message, which will be sent:

Actual message: SEE ME AFTER CLASS

Sending message: VHH PH DIWHU FODVV

Because you are given ***k*** above, you can understand the sending message, VHH PH DIWHU FODVV and will see me after class. But who don't know it cannot see me, right?

However, Caesar's Cipher covers only 26 alphabet letters. Numbers cannot be encrypted. In this program, thus you extend it to cover characters in ASCII code[1]. Each ASCII character is encoded to an 8-bit number (7 bits are used in the traditional code). For example, "A" is 65, "B" is 66, the number "0" is 48, "1" is 49. Lowercases are also differentiated, for example, "a" is 97, "b" is 98. Moreover, it covers other special characters such as "!", "+", and so on. Because it encrypts more than 26 characters, it definitely enhances the secure level!

In your program, you will encrypt a plaintext to get a ciphertext when ***k*** is 5 (NOTE that the ***k*** is 3 in the examples above). And you will decrypt a given ciphertext to get a plaintext. Your program will run as shown in Figure 3. Note that it must follow the correct syntax below:

encrypt(plaintext)
decrypt(ciphertext)

Any typos in entering the command should be caught and returns an error message. It includes the format of command.

```
jwlee $ encrypt(I have a key)
N%mf{j%f%pj~
jwlee $ encrypt(see me at 3)
xjj%rj%fy%8
jwlee $ decrpyt(|jfw%ns%gqzj)
wear in blue
jwlee $ encr(see you soon)
[Error] Please enter the command correctly!
jwlee $ decrypt "rndb"
[Error] Please enter the command correctly!
jwlee $
```

Figure 3. `encrypt/decrypt` example

After the output, the system must print prompt so that user enters a next command. Are you ready? LTTI QZHP!!

C. [Extra work] Encoding/Decoding: `encode/decode`

It is not mandatory but optional, which means those who develop this command will earn extra credits up to 6 points per command. The commands are executed when you want to encode a text to a binary code and decode a binary code to a readable text. The encoding and decoding are commonly used in editing video or audio data. The idea is to convert data (of image, video, audio, etc.) to a format which can be read and managed by machine. Since the machine is a passive device which can use two characters - 0 and 1, the format is usually written in the binary code. There is variation for the format, but you implement a simple encoder/decoder, which convert a character into its ASCII code's binary code. You can use the ASCII code in the last page. See the example of encode and decode commands in Figure 4. The text "key" has three ASCII codes of 107, 101, and 121. The encoder converts them into a binary number (01101011, 01100101, 01111001) and concatenate them. Note that the traditional ASCII code has 7 bits code, but you will consider 8-bit code which enables 8 bits Unicode. Thus, after convert 107 to 1101011, and make it **0**1101011 by putting **0** in the beginning.

```
jwlee $ encode(key)
011010110110010101111001
jwlee $ encode(7-eleven)
00110111001011010110010101101100011001010111011001100101011011 10
jwlee $ encod(location)
[Error] Please enter the command correctly!
```

```
jwlee $ decode “011100”
[Error] Please enter the command correctly!
jwlee $ decode(here)
[Error] Please enter the command correctly!
jwlee $
```

Figure 4. `encode/decode` example

The commands have the following syntax:

encode(text)
decode(binary code)

You should check misspelled commands and incorrect syntax. And you should check binary code input for decode command, since it decodes “binary code” to a text. Look at the last case of **decode(here)** in the **Error! Reference source not found.**. It is incorrect, because “**here**” is not a binary code. After the output, the system must print prompt so that user enters a next command.

Each command is worth 6 points, thus those who develop both commands will get 12 points.

3 Hints in Implementation

A. Reading user input

There are several ways to get your input: `scanf`, `sscanf`, `gets`, `fgets`, and so on. But the best way to handle exceptions such as non-numerical inputs and excess of inputs is to get a string and parse it into required forms. For needs, the good choice is `fgets` but not limited to use other functions in this lab. Find full descriptions of the functions using `man`.

B. Parsing user input

Once user enters a command, the system must parse it to recognize the type of command and input. Tokenizer enables to parse commands and return the information needed for further processing. String library provides necessary functions such as `strtok`. Find full descriptions of the functions using `man`.

4 Requirements

A. Developing environment

The program should be **implemented in C only**. The program in another language will not be graded. And the program should be executable in CS Linux machine. The program which cannot be compiled or executed in the CS Linux machine will be considered as incomplete program and will lose most points.

B. Handling exceptions

The program must detect errors while running. The errors may occur when user violates the syntax of the command or enters incorrect inputs (characters, more than required inputs, or see more in the section II). When the program detects any error cases, it should handle

properly by giving some error messages to the user, and still be runnable. **The program, which fails to detect errors and to properly handle them, will lose points**.

C. Readability of source code

The source code should be easy to read with **good indentation** and **proper amount of comments**.

D. Creating a readme file

The program should be submitted along with a readme file which has information such as your ID, name, etc. Students who work on the extra work should notify to instructor through the readme file. It also may provide a special instruction to run this program if exists. The source file and readme file should be named following the format below. **Those who are not following the naming rule will also be taken off penalty points**.

```
lab1-your_loginID.c
lab1-your_loginID_readme.txt
```

E. Submitting by due

The program should be submitted to Blackboard within the two weeks after the project is posted. **Due is September 14th**. All submission **by 11:59 PM** on that day will be accepted without any penalty. On the due date, Blackboard may be suffering of too much network traffics and be unstable. There is no excuse about the issue, therefore you are strongly recommended to submit earlier than the due date.

5 Grading

A. Grading criteria

The lab is assigned **30** points, which is 15% of the final grade. It will be graded by evaluating the requirement. Any missing and unsatisfiable criteria will take off points. The tentative and brief criteria are below.

- Compilation: **5** points
- Execution: **20** points
- Error detection & others: **5** points

The extra work will give you up to **12** bonus points, which are worth of 40%.

B. Late penalty

Late submission will take off **10% per week** after due date. **Thus, submission after 10 weeks will not be accepted in any circumstances**.

6 Academic Integrity

Any dishonest behaviors will not be tolerated in this class. Any form of plagiarism and cheating will be dealt with according to the guidelines on the Academic Integrity Policy on-

line at http://www.oswego.edu/integrity. For more information about university policies, see the following online catalog at:

http://catalog.oswego.edu/content.php?catoid=2&navoid=47#stat_inte_inte

Student who is against the honor code will not have any credits in this project.

[1] **ASCII code** with binary (not covering *UNICODE*)

Dec	Hex	Oct	Binary	Char	Dec	Hex	Oct	Binary	Char	Dec	Hex	Oct	Binary	Char	Dec	Hex	Oct	Binary	Char
0	00	000	0000000	NUL (null character)	32	20	040	0100000	space	64	40	100	1000000	@	96	60	140	1100000	`
1	01	001	0000001	SOH (start of header)	33	21	041	0100001	!	65	41	101	1000001	A	97	61	141	1100001	a
2	02	002	0000010	STX (start of text)	34	22	042	0100010	"	66	42	102	1000010	B	98	62	142	1100010	b
3	03	003	0000011	ETX (end of text)	35	23	043	0100011	#	67	43	103	1000011	C	99	63	143	1100011	c
4	04	004	0000100	EOT (end of transmission)	36	24	044	0100100	$	68	44	104	1000100	D	100	64	144	1100100	d
5	05	005	0000101	ENQ (enquiry)	37	25	045	0100101	%	69	45	105	1000101	E	101	65	145	1100101	e
6	06	006	0000110	ACK (acknowledge)	38	26	046	0100110	&	70	46	106	1000110	F	102	66	146	1100110	f
7	07	007	0000111	BEL (bell (ring))	39	27	047	0100111	'	71	47	107	1000111	G	103	67	147	1100111	g
8	08	010	0001000	BS (backspace)	40	28	050	0101000	(	72	48	110	1001000	H	104	68	150	1101000	h
9	09	011	0001001	HT (horizontal tab)	41	29	051	0101001	)	73	49	111	1001001	I	105	69	151	1101001	i
10	0A	012	0001010	LF (line feed)	42	2A	052	0101010	*	74	4A	112	1001010	J	106	6A	152	1101010	j
11	0B	013	0001011	VT (vertical tab)	43	2B	053	0101011	+	75	4B	113	1001011	K	107	6B	153	1101011	k
12	0C	014	0001100	FF (form feed)	44	2C	054	0101100	,	76	4C	114	1001100	L	108	6C	154	1101100	l
13	0D	015	0001101	CR (carriage return)	45	2D	055	0101101	-	77	4D	115	1001101	M	109	6D	155	1101101	m
14	0E	016	0001110	SO (shift out)	46	2E	056	0101110	.	78	4E	116	1001110	N	110	6E	156	1101110	n
15	0F	017	0001111	SI (shift in)	47	2F	057	0101111	/	79	4F	117	1001111	O	111	6F	157	1101111	o
16	10	020	0010000	DLE (data link escape)	48	30	060	0110000	0	80	50	120	1010000	P	112	70	160	1110000	p
17	11	021	0010001	DC1 (device control 1)	49	31	061	0110001	1	81	51	121	1010001	Q	113	71	161	1110001	q
18	12	022	0010010	DC2 (device control 2)	50	32	062	0110010	2	82	52	122	1010010	R	114	72	162	1110010	r
19	13	023	0010011	DC3 (device control 3)	51	33	063	0110011	3	83	53	123	1010011	S	115	73	163	1110011	s
20	14	024	0010100	DC4 (device control 4)	52	34	064	0110100	4	84	54	124	1010100	T	116	74	164	1110100	t
21	15	025	0010101	NAK (negative acknowledge)	53	35	065	0110101	5	85	55	125	1010101	U	117	75	165	1110101	u
22	16	026	0010110	SYN (synchronize)	54	36	066	0110110	6	86	56	126	1010110	V	118	76	166	1110110	v
23	17	027	0010111	ETB (end transmission block)	55	37	067	0110111	7	87	57	127	1010111	W	119	77	167	1110111	w
24	18	030	0011000	CAN (cancel)	56	38	070	0111000	8	88	58	130	1011000	X	120	78	170	1111000	x
25	19	031	0011001	EM (end of medium)	57	39	071	0111001	9	89	59	131	1011001	Y	121	79	171	1111001	y
26	1A	032	0011010	SUB (substitute)	58	3A	072	0111010	:	90	5A	132	1011010	Z	122	7A	172	1111010	z
27	1B	033	0011011	ESC (escape)	59	3B	073	0111011	;	91	5B	133	1011011	[	123	7B	173	1111011	{
28	1C	034	0011100	FS (file separator)	60	3C	074	0111100	<	92	5C	134	1011100	\	124	7C	174	1111100	|
29	1D	035	0011101	GS (group separator)	61	3D	075	0111101	=	93	5D	135	1011101	]	125	7D	175	1111101	}
30	1E	036	0011110	RS (record separator)	62	3E	076	0111110	>	94	5E	136	1011110	^	126	7E	176	1111110	~
31	1F	037	0011111	US (unit separator)	63	3F	077	0111111	?	95	5F	137	1011111	_	127	7F	177	1111111	DEL