CSCI165 Computer Science II
Lab Assignment
Simple Cryptography: Substitution and Keyword Ciphers

Objects:

- Read instructions carefully and pay attention to details
- File Reading/Writing
- Sequence processing via iteration
- String comparisons vs character comparisons
- Managing data types properly

In cryptography, a substitution cipher is a method of encrypting in which units of ***plaintext*** are replaced with ***cipher text***, according to a fixed system; the "units" may be single letters (the most common), pairs of letters, triplets of letters, mixtures of the above, and so forth. The receiver deciphers the text by performing the inverse substitution.

Substitution ciphers can be compared with transposition ciphers. In a transposition cipher, the units of the plaintext are rearranged in a different and usually quite complex order, but the units themselves are left unchanged. By contrast, in a substitution cipher, the units of the plaintext are retained in the same sequence in the cipher text, but the units themselves are altered.

There are a number of different types of substitution ciphers. If the cipher operates on single letters, it is termed a simple substitution cipher; a cipher that operates on larger groups of letters is termed polygraphic. A monoalphabetic cipher uses fixed substitution over the entire message, whereas a polyalphabetic cipher uses a number of substitutions at different positions in the message, where a unit from the plaintext is mapped to one of several possibilities in the cipher text and vice versa.

ROT13

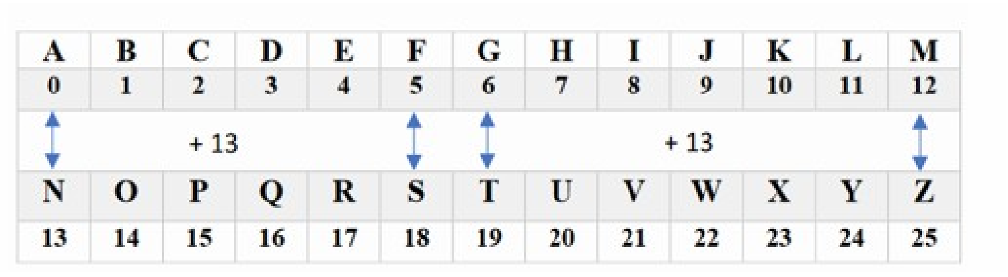
Have you ever been on an internet forum and seen a spoiler that has been encoded into an unreadable message? If you have then the chances are you've already come across ROT13. It's a cipher that is commonly used for disguising non-sensitive information such as puzzle solutions or NSFW (not suitable for work) messages. But this code has its roots all the way back in Roman history.

ROT13 (rotate by 13 places) is a simple letter substitution cipher that replaces a letter with the 13th letter after it, in the alphabet. ROT13 is a special case of the Caesar cipher which was developed in ancient Rome. Because there are 26 letters (2×13) in the basic Latin alphabet, ROT13 is its own inverse; that is, to undo ROT13, the same algorithm is applied, so the same action can be used for encoding and decoding. In other words, two successive applications of ROT13 restore the original text. The algorithm provides virtually no cryptographic security and is often cited as a canonical example of weak encryption, but it is a fun programming challenge.

Applying ROT13 to a piece of text requires examining its alphabetic characters and replacing each one by the letter 13 places further along in the alphabet, wrapping back to the beginning if necessary. A becomes N, B becomes O, and so on up to M, which becomes Z, then the sequence continues at the beginning of the alphabet: N becomes A, O becomes B, and so on to Z, which becomes M.

ROT13 Substitutions

What makes ROT13 unique is that it is its own inverse. Because the alphabet is 26 letters, and the shift is 13 letters, A translates to N and vice versa. However, it doesn't encode numbers or punctuation, which gives it some limitations.

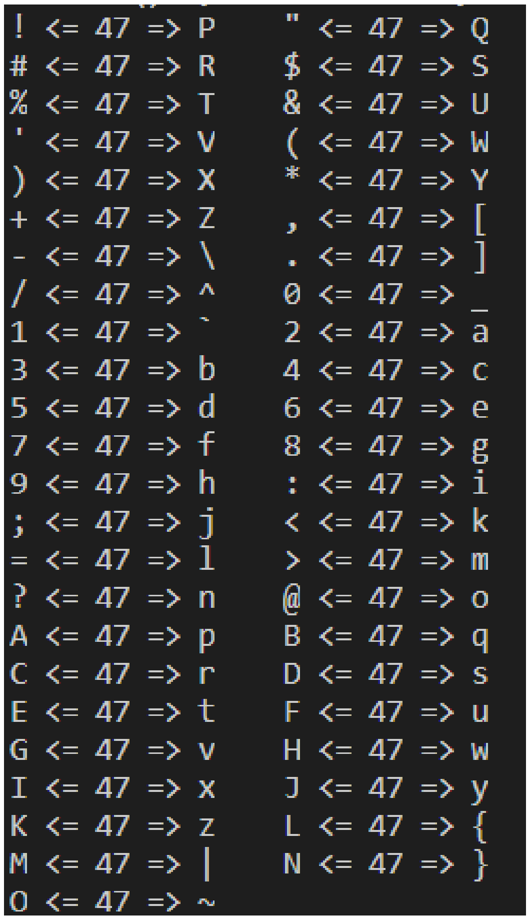


A	B	C	D	E	F	G	H	I	J	K	L	M
0	1	2	3	4	5	6	7	8	9	10	11	12
↕		+ 13			↕	↕		+ 13				↕
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
13	14	15	16	17	18	19	20	21	22	23	24	25

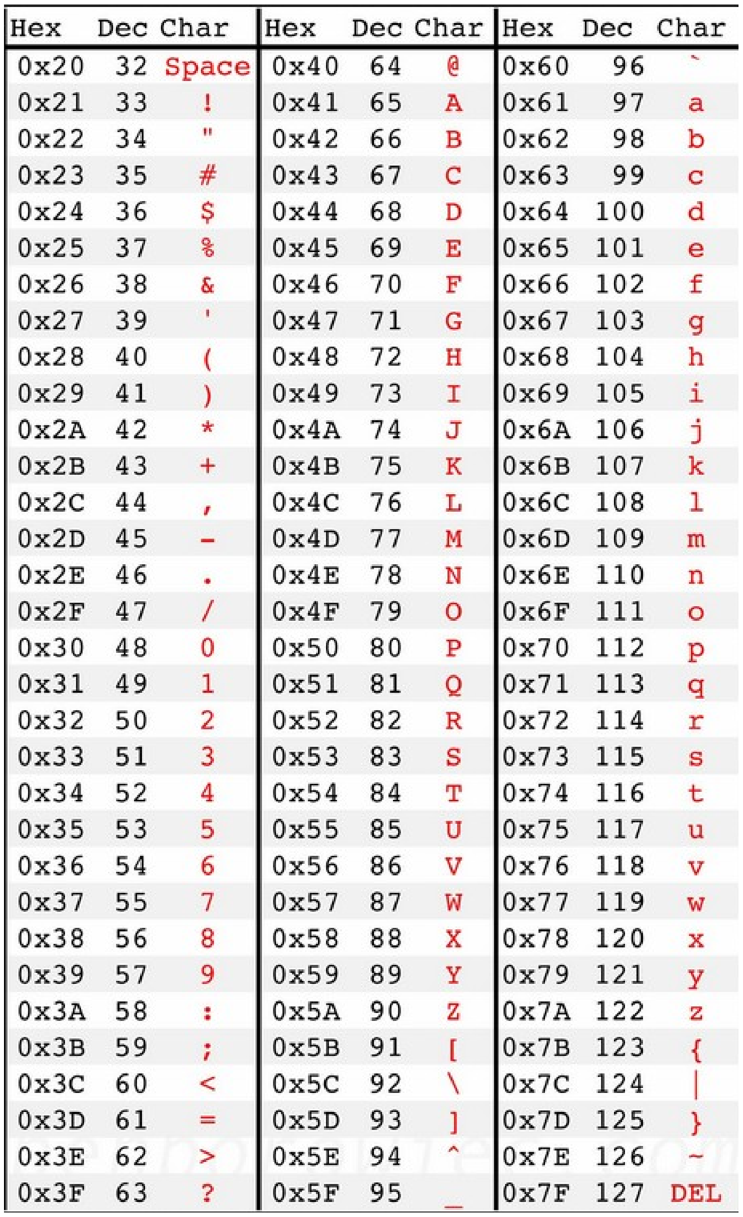
ROT47

ROT47 overcomes the inherent limitations in ROT13 by incorporating the full QWERTY keyboard range by using ASCII characters 33 – 126 (decimal). The same inversion concept as ROT13 applies here accept the ROT distance is 47 based on a character set that has 94 entries

The ROT47 substitutions are



```
! <= 47 => P    " <= 47 => Q
# <= 47 => R    $ <= 47 => S
% <= 47 => T    & <= 47 => U
' <= 47 => V    ( <= 47 => W
) <= 47 => X    * <= 47 => Y
+ <= 47 => Z    , <= 47 => [
- <= 47 => \    . <= 47 => ]
/ <= 47 => ^    0 <= 47 => _
1 <= 47 => `    2 <= 47 => a
3 <= 47 => b    4 <= 47 => c
5 <= 47 => d    6 <= 47 => e
7 <= 47 => f    8 <= 47 => g
9 <= 47 => h    : <= 47 => i
; <= 47 => j    < <= 47 => k
= <= 47 => l    > <= 47 => m
? <= 47 => n    @ <= 47 => o
A <= 47 => p    B <= 47 => q
C <= 47 => r    D <= 47 => s
E <= 47 => t    F <= 47 => u
G <= 47 => v    H <= 47 => w
I <= 47 => x    J <= 47 => y
K <= 47 => z    L <= 47 => {
M <= 47 => |    N <= 47 => }
O <= 47 => ~
```



Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char
0x20	32	Space	0x40	64	@	0x60	96	`
0x21	33	!	0x41	65	A	0x61	97	a
0x22	34	"	0x42	66	B	0x62	98	b
0x23	35	#	0x43	67	C	0x63	99	c
0x24	36	$	0x44	68	D	0x64	100	d
0x25	37	%	0x45	69	E	0x65	101	e
0x26	38	&	0x46	70	F	0x66	102	f
0x27	39	'	0x47	71	G	0x67	103	g
0x28	40	(	0x48	72	H	0x68	104	h
0x29	41	)	0x49	73	I	0x69	105	i
0x2A	42	*	0x4A	74	J	0x6A	106	j
0x2B	43	+	0x4B	75	K	0x6B	107	k
0x2C	44	,	0x4C	76	L	0x6C	108	l
0x2D	45	-	0x4D	77	M	0x6D	109	m
0x2E	46	.	0x4E	78	N	0x6E	110	n
0x2F	47	/	0x4F	79	O	0x6F	111	o
0x30	48	0	0x50	80	P	0x70	112	p
0x31	49	1	0x51	81	Q	0x71	113	q
0x32	50	2	0x52	82	R	0x72	114	r
0x33	51	3	0x53	83	S	0x73	115	s
0x34	52	4	0x54	84	T	0x74	116	t
0x35	53	5	0x55	85	U	0x75	117	u
0x36	54	6	0x56	86	V	0x76	118	v
0x37	55	7	0x57	87	W	0x77	119	w
0x38	56	8	0x58	88	X	0x78	120	x
0x39	57	9	0x59	89	Y	0x79	121	y
0x3A	58	:	0x5A	90	Z	0x7A	122	z
0x3B	59	;	0x5B	91	[	0x7B	123	{
0x3C	60	<	0x5C	92	\	0x7C	124	|
0x3D	61	=	0x5D	93	]	0x7D	125	}
0x3E	62	>	0x5E	94	^	0x7E	126	~
0x3F	63	?	0x5F	95	_	0x7F	127	DEL

Tasks (15 points): Please understand that what follows ***is what will be looked for and graded***. If the instructions say that a method is required, then it is required and it needs to perform its task as specified, accepting the specified arguments in the specified order, and returning the proper type. You are not being asked to create this however you wish, or however you find it on the internet.

1. Create a class called **ROT**
2. Write the static method ***String rotCharacterSet(int)*** that accepts an integer and returns a String. The integer will represent either 13 or 47. The returned String will contain the collection of characters to use for ROT encoding.

   Use a for loop to build the ROT alphabet based on the following character sets

   a. **ROT47:** ASCII characters 33 – 126.
   b. **ROT13:** ASCII characters 65 – 90.
   c. **You must use a loop for this**, you cannot simply type it out. You know the ranges and you can easily convert an integer to its ASCII character.

This should return one of the following. ***You will get points deducted for copying this into your code or writing it out yourself.***

ROT47: !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~

ROT13: ABCDEFGHIJKLMNOPQRSTUVWXYZ

3. Write the static method ***String applyROT(String, int)*** that accepts a String and an integer and returns a ROT encoded String. The integer will represent either 13 or 47. This method should call the ***rotCharacterSet()*** method to get the alphabet, and then create and return a new ROT-N encoded version of the string argument. You must approach the ***wraparound*** substitution of the +13 or +47 ***without using if statements***. This requires modulus division and we worked through an example of this in class.

   YOU SHOULD THOROUGHLY TEST THIS METHOD BEFORE MOVING ON.

NOTE: When using a debugger on a program that uses command line arguments it may be easier to simply define the values and bypass having them passed in from the command line.

Application:

Complete the following:

1. Still in the **ROT class** define a main method. When running the program, ***use command line arguments*** to pass a file name, a string and a number. The file name will represent a file to encode/decode, the String will be either **e** or **d** (for encode or decode) and the number will be either **13** or **47**, representing the ROT encoding.
   a. **Validation:** If the proper quantity of command line arguments are not passed to the program, display a message that communicates this fact. The program should not continue.

   b. **Validation:** If the specified file does not exist, display a descriptive message echoing the file name and stating that it does not exist. The program should not continue
   c. **Validation:** If the **e** or **d** string is neither **e nor d** display the message "Invalid argument". The program should not continue
   d. **Validation:** If the integer is neither 13 nor 47 display a message communicating this fact and stating that the default encoding of ROT13 will be applied.

2. Encode the file using the specified ROT version and write the encrypted or decrypted contents to a new file. Your code should go both ways in and out of ROT encoding.

   a. **Encryption**: If the **e** option is specified the new file should be named: **original_file_name_encrypted.txt** this means, take the original file name and add the **-_encrypted** to the end before the extension.
   b. **Decryption**: If the **d** option is specified, the file name to write to should be the passed in file name with the **_encrypted** removed.
   c. **Don't worry about validating file names for the two options**. Just pretend that we live in a perfect world and the file names will be appropriate. You can trust that files with the **_encrypted** marker will only be passed in with the **d** option and vice versa. That is how I will be testing and grading your work. No curve balls, but I will be using files that you will not have access to.

Keyword Cipher

A keyword cipher is a form of monoalphabetic substitution where certain letters of the plain text alphabet are substituted with a cipher alphabet that has been configured based on a specified key. A text pattern is used as the key, and the key determines the letter matchings of the cipher alphabet to the plain text alphabet. Repeats of letters in the key are removed, then the cipher alphabet is generated with the key characters matching to A, B, C etc. until the keyword is used up. The remaining characters of the cipher text alphabet are used in alphabetical order, ***excluding those already used in the key*.**

Encryption

Let's say we use the text pattern **KRYPTOS** as the keyword. This pattern would be used to transform our alphabet from

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

Notice that A becomes K, B becomes R, C becomes Y etc . . . until all the characters of the key are substituted. Then the rest of the alphabet is used in alphabetical order ***except for*** any characters that were part of the key

Encrypting the message: "I love programming" using the keyword "KRYPTOS":

Encoded message: B EHUT ILHSLKFFBGS

The message has been uppercased and encoded using the cipher alphabet. When encoding using a keyword cipher all whitespace and punctuation are left unaffected. This is decidedly insecure. To add a level of security to our encoded messages do the following:

> ***After encoding using the keyword cipher, write the ciphered text to the file in columns of five characters, ignoring all whitespace and punctuation***.

“I love programming” encoded and written in this manner will yield

BEHUT ILHSL KFFBG S

This increases the security of the ciphered text because you cannot apply brute force pattern matching for common words like “I” and “the”. Obviously the deciphering of this message would be difficult to revert back to its original state, but it would be easily understandable by a human brain with a little effort.

For Example:

Message: we have the plans

Encoded: VTAKU TNATI EKGM

Decoded: WEHAV ETHEP LANS

Tasks (15 points):

1. Define the class **KeywordCipher**
2. Define the static method **String prepareKeyWord(String)** that accepts the keyword and removes any duplicate letters, returning a new String with only the unique characters. Preserve their original order.
3. Define the static method **String generateCipherAlphabet(String)** that accepts a String representing the keyword and returns a new String containing the new alphabet.
4. Define the static method **String encipher(String, String)** that accepts two Strings and returns a new String
    a. **String One**: The cipher alphabet
    b. **String Two**: The String to encipher
    c. **Returns**: The new enciphered String
5. The methods above are required. Feel free to define as many methods as you feel you need to aid in decomposing the problem.

Application:

Using command line arguments pass in the name of a file to encipher and a keyword. Using techniques similar to what was described in the above task, write the encrypted data to a new file in columns of five characters. I’ll allow you to be creative with the way this program functions. Just leave me some comments in the source so I’ll know exactly how to run the file.

Prove that you can also decrypt the file, without worrying about reverting to the exact word structure as the original.