Homework 4 - String Cypher Encryption

Due: Friday November 20th, 2015

Congratulations Agent, you've been recruited to the NSA. Ever since crackers broke into the State Department files we've needed to come up with a new means to keep their prying eyes off our prying eyes. (Note: the opinions in this document do not reflect the opinions of me or the staff or necessarily the US Government)

You have been tasked with creating a world class encryption program in the best language ever, Visual Basic. You mentors (Uttam and Ben) have devised the best encryption technique anyone could possible come up with. (Please never use this algorithm to actually secure anything. Please. Read up on cryptography. If you want an encryption algorithm consider looking into RSA. If you want a more secure hashing algorithm you can start with MD5 even though it was officially deemed insecure in 2008.) Here is an example of a weak encryption standard DES and a strong one PGP.

Encryption Technique

First: we will swap each pair of letters. Handle the end of the string by padding right with spaces to the correct length. This step requires some thinking on how to handle distances < 1

	Unencrypted	Reversed
Swap distance 1	ab	ba
Swap distance 1	abcdefgh	badcfehg
How to handle odd length strings (a # here means a space). Swap dist 1	abc#	ba#c
Swap distance 2	abcd	cdab
Swap distance 2	abcde	cdab##e#
Swap distance 2	abcdefgh	cdabghef
Swap distance 5	a	#####a####

Second: we will shift the letters by a certain amount. This step requires less problem solving on **how** to do it and more on “have I thought of everything?”

ABCDEFGHIJKLMNOPQRSTUVWXYZ

shifted by 3 becomes

XYZABCDEFGHIJKLMNOPQRSTUVW

which can be shifted by -3 or 23 to be

ABCDEFGHIJKLMNOPQRSTUVWXYZ

You **do** need to have alphanumeric symbols wrap around the end.

“A” + 1 = Z
“a” + 1 = z
“9” + 1 = 0

You do **not** need to **shift other** symbols or spaces but do need to **swap** them.

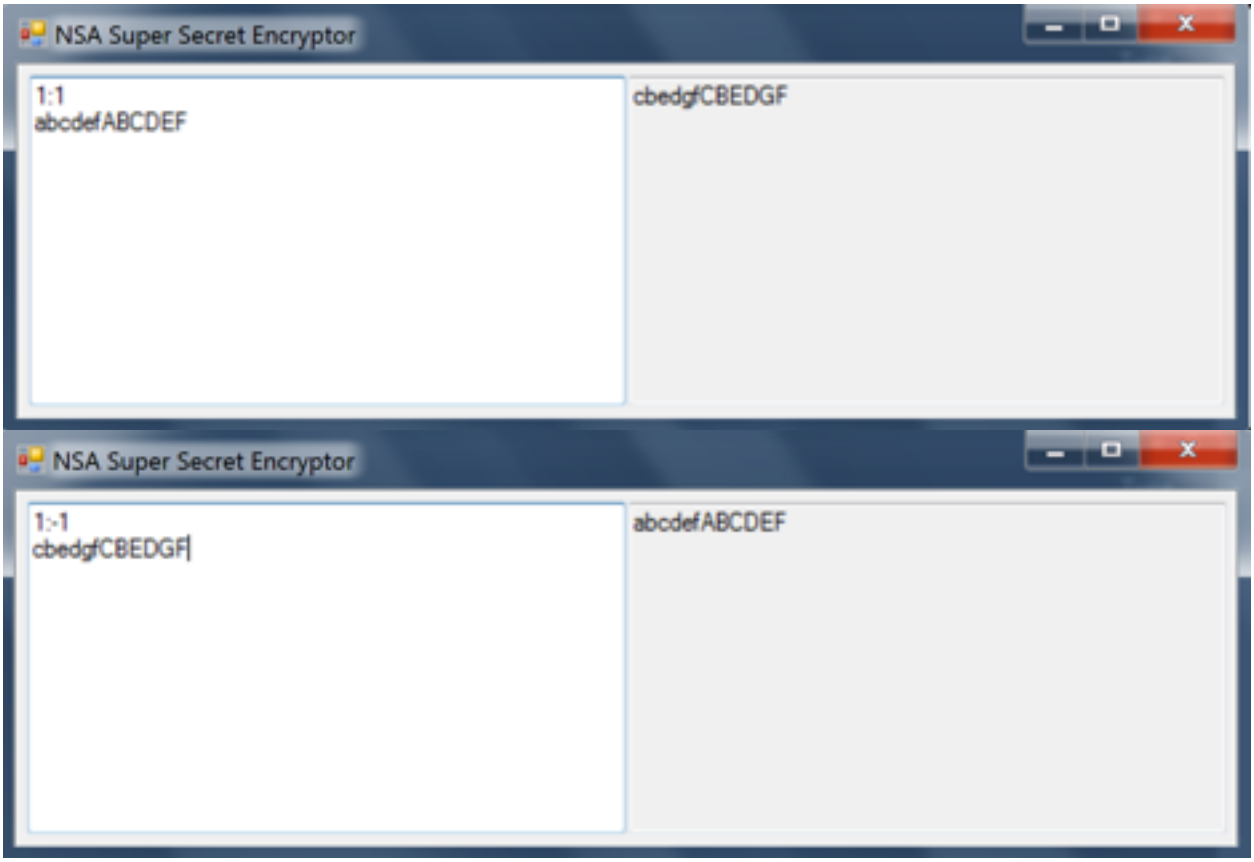
Test Cases

Here are some test cases you might consider up

1:1 AZ az 09 !@ #$ %^ &* () @`/ Hello this is Ben	AB ab 01 @! $# ^% *&)(`@ / fImm piutjj tfC o
2:1 AZ az 09 !@ #$ %^ &* () @`/ Hello this is Ben	BA ba 10 !@ #$ %^ &* () / @` mmIfuip jjtCft o
2:26 And Phibbus' car Shall shine from far And make and mar The foolish Fates.	d AnibPhs'buar c alShshl e inomfrar f d Ankemand aar m e Tholfoh isteFa s.
3:14 This is NOT what that ——> decodes into. It will decode into a very good quote.	bgfOayEouqd q uqzoa ye z mnadq oaagffqdybg e z Mftmazaefdue yk gf mnaqeofqxe abq esq-Qpuvwd Pm efd
4:1900 Wow such big shift, much wrap, so loop	uwejYqy ujk dkiowejhv, r, u ytcqr q nq

If you don't handle shifts ≥ 26 on alphabetic symbols ≥ 10 on numeric symbols and can explain why it would be necessary to in an actual application, minimal points will be taken off. Come to office hours if you don’t understand.

User Interface + Details



- There are two multiline textboxes. On the left is the original message, on the right is the encrypted message in a read-only textbox. You should use the text_changed event to call the encryption. I.E. it should encrypt live as you type.
- The first line of the input is the options/parameters. It comes in the form of a swap distance then a colon then the shift distance.
- You can see that a message can be decrypted by using the same swap distance and the negative shift distance.
- You do not need any sort of restrictive input validation. You should accept any input that the user gives. However if incorrect/no options are given, no output should be produced. Both options, separated by a colon must be given on the first line.
- You should convert the options to integers with Integer.TryParse, NOT Convert.toInt32