Homework, Week 7

December 13, 2013

1 Requirements

This homework requires you to write a program that encrypts and decrypts text. The encryption method is a substitution cipher. It works by substituting the plain text character for another character. There are three versions.

These are your requirements: (1) write a program that allows the user to input clear text to be encrypted, a single character encryption key, and an instruction whether the encryption is to be continuous or not, and that outputs an encrypted string, (2) write a program that allows the user to input clear text to be encrypted, a multi-character encryption key, and an instruction whether the encryption is to be continuous or not, and that outputs an encrypted string, and (3) write a program the same as (2) but that uses the multi-character key once, and then appends the plain text and uses that as the key. Further requirements: (4) write a program that accepts an encoded text, a single character key, and decrypts the text, (5) write a program that accepts an encoded text, a multi-character key, and decrypts the text, and (6) the same as (4) but uses a multi-character, continuous key.

2 Explanation of requirements

Simple substitution ciphers are based on the fact that each letter of the alphabet has a numerical value, for example, A = 1, B = 2, C = 3 ... X = 24, Y = 25, and Z = 26. The encoding wraps around, so that 26 + 1 = 1, not 27, and 23 + 5 = 2, that is W + E = B. See the example in the first version. For decryption, the key is subtracted from the encoded text, for example 2 - 5 = 23, that is, B - E = W. In a computer, alphabetical characters are encoded as digits. In ASCII, A = 65, B = 66, C = 67 ... X = 88, Y = 89, and Z = 90.

The first version is a simple substitution. The code is a one letter character. For example, if the key is 'e', each letter is shifted five places. For example, 'a' becomes 'f', 'b' becomes 'g', and 'z' becomes 'e'. Here is an example.

Plain Text	W	E	A	T	T	A	C	K	A	T	D	A	W	N

ASCII text	87	69	65	84	84	65	67	75	65	84	68	65	87	78
Single key	E	E	E	E	E	E	E	E	E	E	E	E	E	E
ASCII key	5	5	5	5	5	5	5	5	5	5	5	5	5	5
ASCII code	66	74	70	89	89	70	72	80	70	89	73	70	66	83
Encryption	B	J	F	Y	Y	F	H	P	F	Y	I	F	B	S

The second version uses a multi character key. For example, if the key were ‘maze’, the first character would be shifted 13 places, the second character would be shifted one place, the third character would be shifted 26 places, and the fourth character would be shifted five places, and the pattern would repeat. Here is an example.

Plain Text	W	E	A	T	T	A	C	K	A	T	D	A	W	N
ASCII text	87	69	65	84	84	65	67	75	65	84	68	65	87	78
Multi-char key	M	A	Z	E	M	A	Z	E	M	A	Z	E	M	A
ASCII key	13	1	26	5	13	1	26	5	13	1	26	5	13	1
ASCII code	74	70	65	89	71	66	67	80	78	85	68	70	74	79
Encryption	J	F	A	Y	G	B	C	P	N	U	D	F	J	O

The third version uses a multi character continuous key. For example, if the key were ‘maze’, the first four characters of the plain text would be shifted as in the second version. The fifth character would be shifted the value of the FIRST character of the plain text, the character after that would be shifted the value of the SECOND character of the plain text, and so on. Here is an example.

Plain Text	W	E	A	T	T	A	C	K	A	T	D	A	W	N
ASCII text	87	69	65	84	84	65	67	75	65	84	68	65	87	78
Multi-char key	M	A	Z	E	W	E	A	T	T	A	C	K	A	T
ASCII key	13	1	26	5	23	5	1	20	20	1	3	11	1	20
ASCII code	74	70	65	89	81	70	68	69	85	85	71	76	88	72
Encryption	J	F	A	Y	Q	F	D	E	U	U	G	L	X	H

I have dribbled a run of my code. I have called my main functions `(encrypt-1)` and `(decrypt-1)` for the reason that this is the second version, I started with `(encrypt)` and `(decrypt)`. In my code, I have printed the inputs and outputs. This is a good practice when you are writing code, but these kind of print statements should be commented out in production code. Here is an example of my code.

```
;; Dribble of #<IO TERMINAL-STREAM> started on 2013-12-16 11:13:56.
#<OUTPUT BUFFERED FILE-STREAM CHARACTER #P"cipher.txt">
[17]> (encrypt-1)
Enter a line of text to be encrypted: We attack at dawn.
Enter text to be your key: e
Continuous key? (Y|N): n
Text: [(87 69 65 84 84 65 67 75 65 84 68 65 87 78)],
   Full-Key: [(5 5 5 5 5 5 5 5 5 5 5 5 5 5)],
   Continuous: NIL,
   Cont-Key: (5 23 5 1 20 20 1 3 11 1 20 4 1 23 14)
"BJFYYFHPFYIFBS"
[18]> (encrypt-1)
Enter a line of text to be encrypted: We attack at dawn.
Enter text to be your key: maze
Continuous key? (Y|N): n
Text: [(87 69 65 84 84 65 67 75 65 84 68 65 87 78)],
   Full-Key: [(13 1 26 5 13 1 26 5 13 1 26 5 13 1 26 5)],
   Continuous: NIL,
   Cont-Key: (13 1 26 5 23 5 1 20 20 1 3 11 1 20 4 1 23 14)
"JFAYGBCPNUDFJO"
[19]> (encrypt-1)
Enter a line of text to be encrypted: We attack at dawn.
Enter text to be your key: maze
Continuous key? (Y|N): y
Text: [(87 69 65 84 84 65 67 75 65 84 68 65 87 78)],
    Full-Key: [(13 1 26 5 13 1 26 5 13 1 26 5 13 1 26 5)],
    Continuous: T,
    Cont-Key: (13 1 26 5 23 5 1 20 20 1 3 11 1 20 4 1 23 14)
"JFAYQFDEUUGLXH"
[20]> (decrypt-1)
Enter a line of text to be decrypted: BJFYYFHPFYIFBS
Enter text that is your key: e
Continuous key? (Y|N): n
Text: [(66 74 70 89 89 70 72 80 70 89 73 70 66 83)],
   Full-Key: [(5 5 5 5 5 5 5 5 5 5 5 5 5 5)],
   Continuous: NIL, Cont-Key: (5)
"WEATTACKATDAWN"
[21]> (decrypt-1)
Enter a line of text to be decrypted: JFAYGBCPNUDFJO
Enter text that is your key: maze
Continuous key? (Y|N): n
Text: [(74 70 65 89 71 66 67 80 78 85 68 70 74 79)],
   Full-Key: [(13 1 26 5 13 1 26 5 13 1 26 5 13 1 26 5)],
   Continuous: NIL, Cont-Key: (13 1 26 5)
"WEATTACKATDAWN"
[22]> (decrypt-1)
Enter a line of text to be decrypted: JFAYQFDEUUGLXH
Enter text that is your key: maze
Continuous key? (Y|N): y
Text: [(74 70 65 89 81 70 68 69 85 85 71 76 88 72)],
   Full-Key: [(13 1 26 5 13 1 26 5 13 1 26 5 13 1 26 5)],
   Cohtinuous: T, Cont-Key: (13 1 26 5)
"WEATTACKATDAWN"
[23]> (dribble)
;; Dribble of #<IO TERMINAL-STREAM> finished on 2013-12-16 11:16:15.
```

This homework is a multi-week homework assignment. Work on this a piece at a time, and apply everything you have learned so far. I have posted my code, but looking at my solution before you have sweated blood is cheating.

3 Code hints and examples

Here are some built in Lisp functions that you might find useful.

(format) You will want to prompt your user to input various items.

```
[24]> (format t "This is a user prompt: ")
This is a user prompt:
NIL
```

(read-line) You will want to read what your user inputs.

```
[25]> (read-line)
This is what I input.
```

```
"This is what I input." ;
NIL
```

(remove-if-not) You will want to remove all non-alphabetical characters, like spaces and punctuation, from the input.

```
[28]> (remove-if-not #'alpha-char-p "This string contains spaces and pumction.")
"Thisstringcontainsspacesandpumction"
```

(string-upcase) You will want to convert all lower case characters to upper case.

```
[29]> (string-upcase "This string contains lower case characters.")
"THIS STRING CONTAINS LOWER CASE CHARACTERS."
```

(coerce) You will want to turn a string into a list, and a list into a string.

```
[33]> (setf str "This is a string")
"This is a string"
[34]> str
"This is a string"
[35]> (setf lst (coerce str 'list))
(#\T #\h #\i #\s #\Space #\i #\s #\Space #\a #\Space #\s #\t #\r #\i #\n #\g)
[36]> lst
(#\T #\h #\i #\s #\Space #\i #\s #\Space #\a #\Space #\s #\t #\r #\i #\n #\g)
[37]> (coerce lst 'string)
"This is a string"
```

(char-code) and (code-char) You will want to turn a character into its ASCII code, and turn an ASCII code into a character.

```
[40]> (char-code #\A)
65
[41]> (code-char 90)
#\Z
```

((decf) and (incf) You will want to convert the ASCII code of a character to its normal alphabetical equivalent, and from its normal alphabetical equivalent into its ASCII code.

```
[45]> (setf a 65)
65
[46]> a
65
[47]> (decf a 64)
1
[48]> (setf z 26)
26
[49]> z
26
[50]> (incf z 64)
90
```

(if) You will want to make simple decisions.

```
[54]> (if t "This is true" "This is false")
"This is true"
[55]> (if () "This is true" "This is false")
"This is false"
```

(cond) You will want to make decisions that are not so simple.

(car), (cdr), and (cons) You will want to take lists apart and put them back together.

(list) You may want to turn a single character into a list.

```
[56]> (list 'a)
(A)
```

(append) You may want to add something to the end of an existing list.

```
[64]> (append '(a b c d) '(e))
(A B C D E)
```

(mapcar) One of the problems you will face almost immediately is how to apply the same action to each item in a list. You can do this with `(mapcar)`, which takes a function as the first argument, and one or more lists as the succeeding arguments, and applies the function to each item of the list (or lists) in turn.

```
[66]> (mapcar #'code-char '(65 66 67 88 89 90))
(#\A #\B #\C #\X #\Y #\Z)
[67]> (mapcar #'char-code '(#\A #\B #\C #\X #\Y #\Z))
(65 66 67 88 89 90)
```

(y-or-no-p) This very cool function only accepts a 'y' or a 'n', and won't let you go until it gets one, returning 'T' if it gets a 'y' and 'NIL' if it gets a 'n.'

```
[68]> (y-or-n-p)
y
T
[69]> (y-or-n-p)
n
NIL
[70]> (y-or-n-p)
a
Please answer with y or n : b
Please answer with y or n : c
Please answer with y or n : d
Please answer with y or n : e
Please answer with y or n : y
T
```

(ceiling) You may want to know how many times a shorter list will go into a larger list. This function returns two values, separated by a semi-colon. All you need is the first value, so you can ignore the second value.

```
[71]> (ceiling 20 5)
4 ;
0
[72]> (ceiling 20 4)
5 ;
0
[73]> (ceiling 20 3)
7 ;
-1
```

(loop) This is a very complicated function, and we have an entire chapter on just this one function later on. For now, this is an easy to repeat a action multiple times.

```
[74]> (loop repeat 4 appending '(E))
(E E E E)
[75]> (loop repeat 5 appending '(MAZE))
(MAZE MAZE MAZE MAZE MAZE)
```

(let) and (setf) You will want to assign values to symbols, and to use the symbols as variables.

4 Design

So here you are, scratching your head and feeling lost. I know the feeling. This is what you should do. First, *by hand* take one plain character, one key, and figure out how to substitute that plain character to an encrypted character using the key code as the numerical value to substitute. For example, if your plain character was W, and your key was E, you would convert the E to 5, its numeric value, and add 5 to W, wrapping around if necessary. Then, you would write a piece of code to do exactly the same thing.

Next, you would probably want to take the ASCII value of W, which is 87, and figure out how to convert it to a B (assuming E is your key), and write a piece of cod that does that. You might want to write a function that subtracts 24 from your plain character if the value exceeds 90, that is, 91 should be 65. You may also want to write a function that adds 24 to a character that is less than 60, that is, 64 should be 90, 63 should be 89, and so on.

Think your way through the process, one step at a time. At each step, write a piece of code that *duplicates exactly* the step you have just complete by hand. You *will* come to dead ends, and you *will* have to revise or throw away code that you have previously written. Just remember, if you can convert one step into code that works correctly, you will have completed that much more of the assignment. This, in fact, is how code is written . . . one function at a time.

A very smart man once said, “If faced with a problem you don’t know how to solve, solve any part that you can and look at it again.” Here is your chance to learn this lesson and learn how to apply this advice to a real world problem.