**Cryptol ASSIGNMENT**

**Assigned Week#1**

1. **Due at beginning of class during Week 2.**
2. **This is an individual assignment.**
3. **Students will need to install cryptol.net and follow the instruction below.**
4. **Take screen photographs of your output in hex and asci for key = 8.**
5. **Answer the questions.**
6. **E-mail your photographs and answered questions by due date.**

**Installing Cryptol -- Windows**

Download from <http://www.cryptol.net/downloads.html>

Run installation

Download Z3 from <https://github.com/Z3Prover/z3/releases>

Extract to some easy to find directory (you will need the directory path)

Include Z3s path to Windows PATH environment variable

1. Go to This PC
2. Right click and click on Properties
3. Click on Advanced System Settings
4. Click on Environment Variables
5. Select PATH and click on Edit
6. Click new, then insert the path to z3.exe (this will be in the \bin folder of the Z3 directory)

**RESTART YOUR COMPUTER**

You can now run Cryptol by opening command prompt and typing in "cryptol"

If the directions above do not work for some reasons, e-mail professor N. He has another way of doing it that might work.

**REFERENCE:**

Before you do the assignment below, I suggest that you skim the cryptol documentation which can be downloaded from link below:

<http://www.cryptol.net/documentation.html>

Later on, if you’d like to introduce Cryptol in your work, this document is very helpful.

Assignment below needs to be done and checked-off by Week 2 in class over Google Hangouts.

**ASSIGNMENT:**

**Creating a function -- The Caesar cipher**

1. Create and open a new file using vim, notepad, or any other text editor

Example using vim: vim caesar.cry

Cryptol is a dynamically typed language and like python it uses indentation to define code "blocks"

**It is very important to use consistent indentation**

Type the following code into your text editor:

caesar : {n} ([8], String n) -> String n

caesar (s, msg) = [ shift x | x <- msg ]

where map = ['A' .. 'Z'] <<< s

shift c = map @ (c - 'A')

**-----NOTE-----**

Copying and pasting the code may change the formatting, check to ensure that the indentation is correct

1. Save the file and exit
2. In command prompt, change to the directory where your .cry file is located. Load your function into cryptol by using the following command:

cryptol caesar.cry

1. Now, you can call your function through cryptol

caesar(25, "HELLO WORLD")

**-----NOTE-----**

The default output of this function will be in HEX

You can have cryptol output ASCII characters by using the following command:

:set ascii=on

**EXPLANATION OF THE CODE:**

line 1: This is the function signature, it tells cryptol the type of input, and the type of output to expect. The input is an 8 bit word, which is how cryptol would represent an unsigned short from C++ (any value between 0 and 255), and a string, outputting a string

line 2: This is the function header and body. "caesar (s, msg)" is the header, and is pretty self-explanatory.

"[ shift x | x <- msg ]" is the body. This will iterate through all the elements of msg, performing shift on each one.

The function shift is then defined locally within the where clause

line 3: "map = ['A' .. 'Z'] <<< s" creates an array called map, and stores the letters A through Z within it, then it circular left shifts this array by the input offset

line 4: This is the function body for shift. shift will calculate the index offset using (c - 'A'), where c is the input character

shift then selects the character at that index

**QUESTIONS:**

1. Caesar relies on having each character of the plaintext being evaluated as such:

ciphertext = (plaintext + key) % 26

1. How does this code ensure that the characters are properly offset when the plaintext + key is greater than 'Z'?

Assuming we are talking about the bit of code right above this text instead of the code mentioned earlier in the lab, the modulus 26 operation ensures that no value comes outside the range of 0 to 25.

The code chunk introduced in the beginning of the lab does not do this, resulting in the errors detailed in question 2.

1. How can we alter this code to instead decrypt the message?

The easiest way I can think of to decrypt instead of encrypt the message is to reverse the key value. To do this, we take a value of 26 and subtract the key value giving us the reversed key value. We then modulus by 26 so the key remains in the 0 to 25 range. The end code would look like this:

caesar : {n} ([8], String n) -> String n

caesar (s, msg) = [ shift x | x <- msg ]

where map = ['A' .. 'Z'] <<< (26 - s) % 26

shift c = map @(c - 'A')

1. What happens when the input message contains lower case letters or numeric values?

It fails due to values being outside the sequence index. I believe this sequence index is identified by the map @ (c - 'A') operation. If the value is outside the range of 0 to 25 there is no corresponding value in the map, and the input value doesn’t map to a valid amount. This also happens with any other ASCII symbols such as a space.

1. Can you think of a way to change this?

You could alter the operation to shift c = map @ ((c - 'A') % 26) which would ensure the input value is always inside the valid map range, but this makes the function not be deterministic. This means that a wide range of inputs could be used but they wouldn’t always be decoded to the proper values.

Instead, I propose that we check if the input values are within the ranges ‘a’ to ‘z’ or ‘A’ to ‘Z’. If they are not within these ranges, they simply wouldn’t be encoded. If they are within those ranges, we check if they are upper or lower case. If they are lower case, we convert them to upper case with an operation like:

c – ‘a’ + ‘A’

then run the altered lower case or normal upper case characters through the operation of:

shift c = map @ (c - 'A')

This sequence of operations would ensure that lower and upper case values would be converted to upper case, but that is an acceptable loss of information. If this was not acceptable, we would need a more complex cypher that allows for a range of 0 to 51 values instead of 0 to 25.

This could also be done by checking for the ‘a’ to ‘z’ range and converting it to the map like:

shift c = map @ (c - 'a')

Then if the character value was not in the ‘a’ to ‘z’ range, check if it is in the ‘A’ to ‘Z’ range. If it is, then convert it to the same map like:

shift c = map @ (c - 'A')

Due to the nature of the map converting all values to upper case all output would be in upper case. This may be a slightly more logically efficient method, but it requires two separate mapping operations which I believe are less efficient if implemented in hardware compared to the added subtraction and addition operations detailed in my previous method.