Daniel Cody

Professor Labouseur

Theory of Programming Languages

1 April 2014

**Prediction**

I predict that Scala will only take me about an hour or two since I have previous experience in it from the last assignment and also because when I had began to learn it, I had read abut making it functional. I expect just converting my algorithm to a functional base will be the only issue. I plan to first write it in sML because that is the functional language that I am most comfortable with since I wrote that in Language Study last year. Since I am so familiar with sML I expect it to only take me an hour. While writing it for the last assignment I had planned how I was going to do it in sML, so I already have a game plan and know the syntax well. My only concern is how well my algorithm will convert to a functional base, but that is why ill do it first in my most comfortable language.

I have yet to decide exactly which for the last two languages I will do. I plan to do LISP and Erlang with Haskell as a backup. I know my friends have done Haskell in their Language Study class so I believe it is easy to learn. I expect Erlang and LISP to take me a little longer. Based on the brief research I have done on them the syntax is far different from any I have seen before, LISP has too many parenthesis, and Erlang is just odd. I expect them to take me a few hours at least maybe even longer.

**3/3/14**

Today I did sML. I did it for language study last year so I was very comfortable with the language. All I needed to do was figure out the algorithm for the function. I also needed to figure out exactly the string functions needed. I also realized about half way through my program that it would be easier to write two helper functions for the character integer conversion which made me change the algorithm but made it a lot easier. Overall it took me about an hour and a half to do this program. I really enjoyed sML but it is the only one I have done that is functional so I am excited to try the other ones to see the differences between them.

**3/7/14**

Today I did Scala functionally. I wanted to take time in between doing the two different styles of Scala so I did not get confused as well as I was very comfortable with sML so I knew I could figure out the algorithm there easier. Also sML is very strict on type checking so I knew it would be easy to port it to another language. Scala is so nicely and smoothly written especially because I am a big fan of java it is extremely easy to write. I think Scala might be my favorite language I have ever used. It is so much like java which I love but also has the functional ability which is something I grew to like after using sML. Scala is just nicely written java, without all the hassle java causes. Println versus System.out.println is just a waste of typing as well as readability. All in all I believe Scala took me about an hour to write. The syntax I was familiar with due to writing it for the pervious assignment.

**3/10/14**

Today I started to do LISP. So far it is extremely annoying with all the parenthesis and how oddly it is written. A simple thing such as adding two numbers is just written illogically from how I see it. I have not been able to get the functions to work as I want them. I previously have done it recursively, but I know that it is possible to make it so you can use a map to do it. I believe that a map could be easier since everything is LISP is a list. I am going to try to keep working on doing it recursively just because I have only put a little over an hour into it so far.

As I continued to work on it, I only grew in frustration, because I was unable to get it to work. When I would try to call my sub functions inside of another function only errors occurred and I could not figure out why. I really do not like the error messages I get with LISP; they were not as easy to read as others were. I wish they gave me more detailed information with each message.

**3/14/14**

After taking some time away from LISP to clear my head since I was so frustrated with it, I was going to try to do it with a map versus recursively. I found the map to be a lot easier since it handles going through the entire string. The biggest issue that I faced was getting it to print out the entire string not just the single character after it was changed. After trying to change the output I realized that I can just print the entire map which will just print the entire string that it returns.

I faced an issue with spaces because I was doing the conversion with a mod sequence, but I figured out that you can put functions inside of the map. I was having an issues using my sub function to work with it, saying my map had incomplete arguments, so I realized I can put the body of my function in that spot which worked well for me.

After I had figured out all of these issues the only issue I faced after was trying to get a loop to go from 26 to 0 which is how I previously had done it. I realized though that if I go from 0 to 26 and I use decrypt instead of encrypt it works just fine.

3/21/14

Today I worked on Erlang, which was not easy. So far it was the most annoying language I have worked on so far for this assignment. It almost compared to COBOL for me. I worked on trying to get the syntax down for over an hour and it still gives me weird errors.

**Scala**

object encode {

//main method

def main(args: Array[String]) {

//test strings

val Estr : String = "THIS IS A TEST STRING FROM1122@";

val Dstr : String = "BPQA QA I BMAB ABZQVO NZWU1122@";

val Sstr : String = "HAL@";

//main

println(encode(Estr, 8));

println(decode(Dstr, 8));

println(solve(Sstr, 26));

}

//method to figure out char int for encode

def addCharInt(charInt : Int, moveAmt : Int) : Int = {

//check if letter

if(charInt <= 64)

return charInt;

//wrap around

else if((charInt + moveAmt) > 90)

return (64 + ((charInt + moveAmt) - 90));

//no wrap

else

return (charInt + moveAmt);

}

//method to figure out char int for decode

def subCharInt(charInt : Int, moveAmt : Int) : Int = {

//check if letter

if(charInt <= 64)

return charInt;

//wrap around

else if((charInt - moveAmt) < 64)

return (90 - (64 - (charInt - moveAmt)));

//no wrap around

else

return (charInt - moveAmt);

}

def encode(str : String, moveAmt : Int) : String = {

//vals for the char, rest of the string, char int and final char

val char : Char = str.charAt(0);

val rest : String = str.substring(1, str.size);

val charInt : Int = addCharInt(char.toInt, moveAmt);

val newChar : Char = charInt.toChar;

//check for recurseive call

if(rest.size == 0){

return newChar.toString;

}

else{

newChar.toString + encode(rest, moveAmt);

}

}

def decode(str : String, moveAmt : Int) : String = {

//vals for the char, rest of the string, char int and final char

val char : Char = str.charAt(0);

val rest : String = str.substring(1, str.size);

val charInt : Int = subCharInt(char.toInt, moveAmt);

val newChar : Char = charInt.toChar;

//check for recursive call

if(rest.size == 0){

return newChar.toString;

}

else{

newChar.toString + decode(rest, moveAmt);

}

}

def solve(str : String, solves : Int) : String = {

//check for recursive call

if (solves == -1)

return "\n";

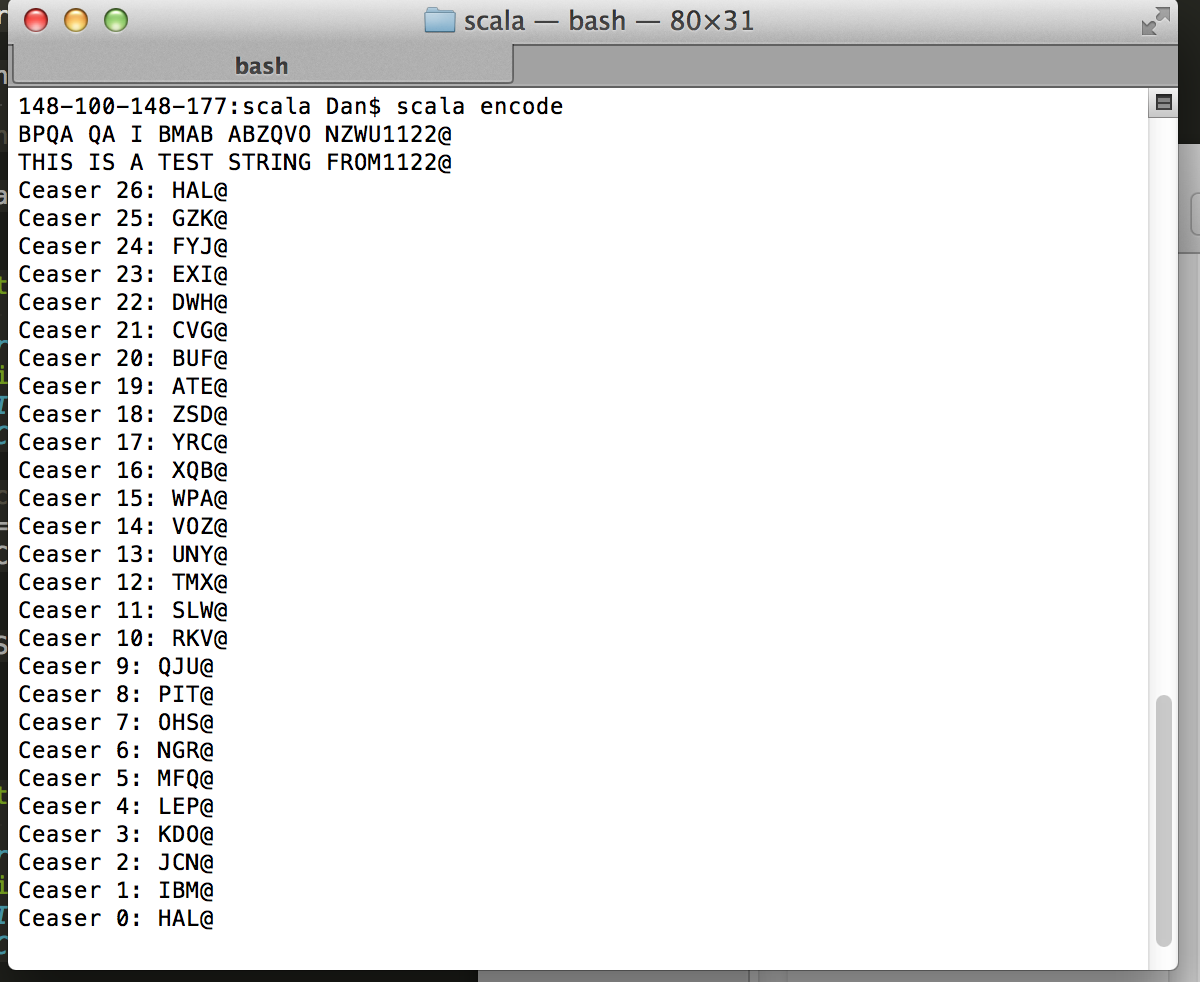
//call encode and recursive call

else

"Ceaser " + solves.toString() + ": " + encode(str, solves) + " \n" + solve(str, solves - 1);

}

}



**SML**

(\* Solving the char int based on the wrap around\*)

fun addCharInt(charInt, moveAmt) =

if(charInt < 64) then

charInt

else if((charInt + moveAmt) > 90) then

64 + ((charInt + moveAmt) - 90)

else

charInt + moveAmt;

(\* Solving the char int based on the wrap around\*)

fun subCharInt(charInt, moveAmt) =

if(charInt < 64) then

charInt

else if((charInt - moveAmt) < 64) then

90 - (64 - (charInt - moveAmt))

else

charInt - moveAmt;

(\* Encode function based on the move amount \*)

fun encode(str, moveAmt) =

let

val char = String.sub(str,0);

val str1 = String.substring(str, 1, String.size(str) - 1)

val charInt = addCharInt(ord(char), moveAmt)

in

if (String.size(str1) = 0) then Char.toString(chr(charInt))

else String.^(Char.toString(chr(charInt)), encode(str1, moveAmt))

end;

(\* decode function based on the move amount \*)

fun decode(str, moveAmt) =

let

val char = String.sub(str,0);

val str1 = String.substring(str, 1, String.size(str) - 1)

val charInt = subCharInt(ord(char), moveAmt)

in

if (String.size(str1) = 0) then Char.toString(chr(charInt))

else String.^(Char.toString(chr(charInt)), decode(str1, moveAmt))

end;

(\* Solve function to show all cases of the encode\*)

fun solve(str, solves) =

let

val csStr = "Ceaser " ^ Int.toString(solves) ^ ": "

in

if (solves = 0) then "\n"

else csStr ^ encode(str, solves) ^ " \n" ^ solve(str, solves - 1)

end;

(\*test cases \*)

val encodeTest = encode("123THIS IS A TEST STRING FROM", 8);

val decodeTest = decode("123BPQA QA I BMAB ABZQVO NZWU", 8);

val solveTest = solve("HAL", 26);

print(solveTest)

Executing the program....

$sml < main.sml

Standard ML of New Jersey v110.74 [built: Fri Jan 24 17:53:58 2014]

- val addCharInt = fn : int \* int -> int

val subCharInt = fn : int \* int -> int

[autoloading]

[library $SMLNJ-BASIS/basis.cm is stable]

[autoloading done]

val encode = fn : string \* int -> string

val decode = fn : string \* int -> string

[autoloading]

[autoloading done]

val solve = fn : string \* int -> string

val encodeTest = "123BPQA QA I BMAB ABZQVO NZWU" : string

val decodeTest = "123THIS IS A TEST STRING FROM" : string

val solveTest =

"Ceaser 26: HAL \nCeaser 25: GZK \nCeaser 24: FYJ \nCeaser 23: EXI \nCeaser#"

: string

= Ceaser 26: HAL

Ceaser 25: GZK

Ceaser 24: FYJ

Ceaser 23: EXI

Ceaser 22: DWH

Ceaser 21: CVG

Ceaser 20: BUF

Ceaser 19: ATE

Ceaser 18: ZSD

Ceaser 17: YRC

Ceaser 16: XQB

Ceaser 15: WPA

Ceaser 14: VOZ

Ceaser 13: UNY

Ceaser 12: TMX

Ceaser 11: SLW

Ceaser 10: RKV

Ceaser 9: QJU

Ceaser 8: PIT

Ceaser 7: OHS

Ceaser 6: NGR

Ceaser 5: MFQ

Ceaser 4: LEP

Ceaser 3: KDO

Ceaser 2: JCN

Ceaser 1: IBM

val it = () : unit

**LISP**

(defun encode (str moveAmt)

(print (map 'string (lambda (c)

(if(/= (char-code c) 32)

(if (>(+(char-code c) moveAmt) 90)

(code-char(+ 64 (-(+(char-code c) moveAmt) 90)))

(code-char(+ (char-code c) moveAmt)))

(code-char 32)))(string-upcase str)))

)

(defun decode (str moveAmt)

(print (map 'string (lambda (c)

(if(/= (char-code c) 32)

(if (<(-(char-code c) moveAmt) 65)

(code-char(- 91 (- 65 (-(char-code c) moveAmt))))

(code-char(- (char-code c) moveAmt)))

(code-char 32)))(string-upcase str)))

)

(defun solve (str solves)

(loop for num from 0 to solves do

(print (map 'string (lambda (c)

(if(/= (char-code c) 32)

(if (<(-(char-code c) num) 65)

(code-char(- 91 (- 65 (-(char-code c) num))))

(code-char(- (char-code c) num)))

(code-char 32)))(string-upcase str)))

)

)

(encode "This is A Test String" 8)

(decode "Bpqa qa i bmab ABZQVO" 8)

(solve "hal" 26)

**Executing the program....**  
$clisp main.lisp

"BPQA QA I BMAB ABZQVO"

"THIS IS A TEST STRING"

"HAL"

"GZK"

"FYJ"

"EXI"

"DWH"

"CVG"

"BUF"

"ATE"

"ZSD"

"YRC"

"XQB"

"WPA"

“VOZ"

"UNY"

"TMX"

"SLW"

"RKV"

"QJU"

"PIT"

"OHS"

"NGR"

"MFQ"

"LEP"

"KDO"

"JCN"

"IBM"

"HAL"

**Erlang**

-module(encode1).

-export([addInt/2, subInt/2, encode/2, decode/2, solve/2]).

-import(string, [to\_upper/1, concat/2]).

addInt(CharInt,MoveAmt) ->

if

CharInt < 64 ->

CharInt;

(CharInt + MoveAmt) > 90 ->

(64 + ((CharInt + MoveAmt) - 90));

true ->

(CharInt + MoveAmt)

end.

subInt(CharInt,MoveAmt) ->

if

CharInt < 64 ->

CharInt;

(CharInt - MoveAmt) < 64 ->

(90 - (64 - (CharInt - MoveAmt)));

true ->

(CharInt - MoveAmt)

end.

encode("", MoveAmt) -> "";

encode(Str1, MoveAmt) ->

TheCharInt = hd(Str1),

CharInt = addInt(TheCharInt, MoveAmt),

NewStr = tl(Str1),

if

length(NewStr) == 0 ->

[CharInt];

true ->

string:concat([CharInt], encode(NewStr, MoveAmt))

end.

decode("", MoveAmt) -> "";

decode(Str1, MoveAmt) ->

TheCharInt = hd(Str1),

CharInt = subInt(TheCharInt, MoveAmt),

NewStr = tl(Str1),

if

length(NewStr) == 0 ->

[CharInt];

true ->

string:concat([CharInt], decode(NewStr, MoveAmt))

end.

solve(Str1, 0) -> io:fwrite(encode(Str1, 0));

solve(Str1, Solves) ->

io:fwrite(encode(Str1, Solves)),

io:fwrite("\n"),

solve(Str1, (Solves -1)).

