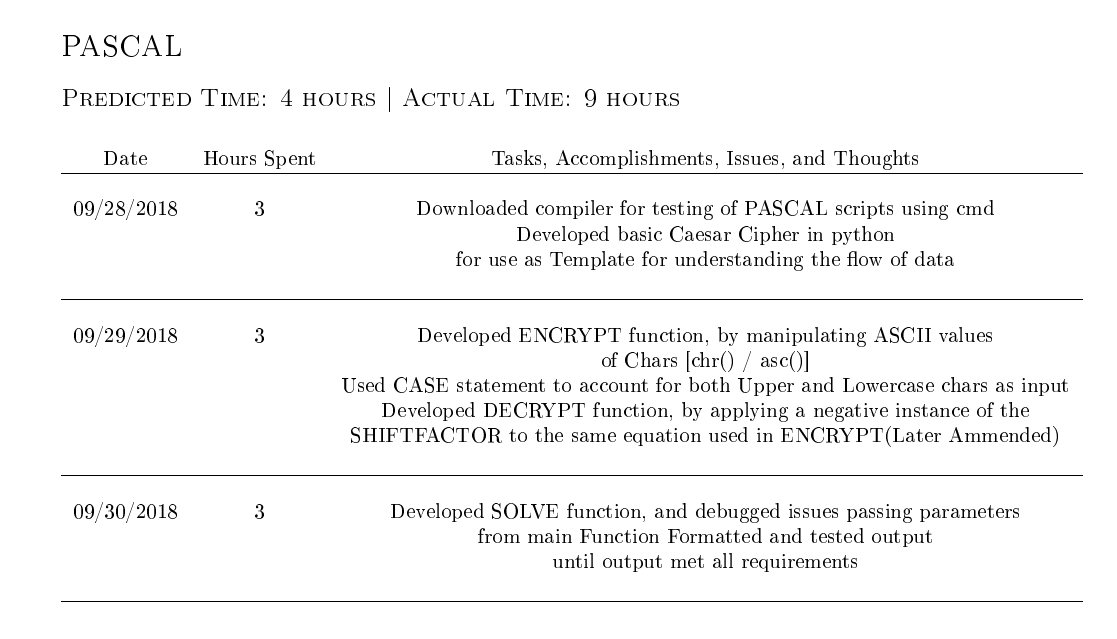
Jeffrey Lupia

Theory of Programming Languages

Professor Labouseur

*Programming in the past*

**PASCAL Programming language:**

****

CODE:

///////////////////////////////////////////////////////////////////

// Jeffrey Lupia

// Caesar Cipher

// Theory Of Programming languages

// Programming in the past

////////////////////// PASCAL \\\\\\\\\\\\\\\\\\\\\\\

program CaesarCipher;

var

S, S1: string;

shiftFactor: Integer;

maxShiftValue: Integer;

/////////////////////////////////////////////

function encrypt(S: string; shiftFactor: Integer): string; //returns STRING-encrypt

var

index: Integer;

begin

encrypt:= S;

for index:=1 to length(encrypt) do

case encrypt[index] of

'A'..'Z':encrypt[index] := chr(ord('A') + (ord(encrypt[index]) - ord('A')+ shiftFactor)mod 26);

'a'..'z':encrypt[index] := chr(ord('a') + (ord(encrypt[index]) - ord('a')+ shiftFactor)mod 26);

end;

end;

////////////////////////////////////////////

function decrypt(S1: string; shiftFactor: Integer): string; //returns STRING-decrypt

var

index: Integer;

begin

decrypt:= S1;

for index:=1 to length(decrypt) do

case decrypt[index] of

'A'..'Z':decrypt[index] := chr(ord('A')+ (ord(decrypt[index]) - ord('A')- shiftFactor + 26)mod 26);

'a'..'z':decrypt[index] := chr(ord('a')+ (ord(decrypt[index]) - ord('a')- shiftFactor + 26)mod 26);

end;

end;

////////////////////////////////////////////////

procedure solve(S: string; maxShiftValue: Integer);

var

index: Integer;

solved: string;

begin

solved := S;

writeln('Caesar ', maxShiftValue, ': ', solved);

for index := 1 to maxShiftValue do

begin

solved := encrypt(solved, 1);

writeln('Caesar ', (maxShiftValue - index), ': ', solved);

end;

end;

/////////////////////////////////////////

Begin {Main Program Block}

maxShiftValue:=26;

//Were this compiler able to accept prompts from the command line, the syntax

//would be:

//writeln('Tell me about yourself....:')

//readln(S);

//writeln('And your favorite number???');

//readln(shiftFactor);

//prompting the user for input, then using that string to go through each function

shiftFactor:=4;

S:='Pack my box with five dozen liquor jugs';

S1:= encrypt(S, shiftFactor);

writeln(S);

writeln(S1);

writeln(decrypt(S1, shiftFactor));

solve(S, maxShiftValue);

end. { end of main program block }

OUTPUT:

**$main**  
Pack my box with five dozen liquor jugs

Tego qc fsb amxl jmzi hsdir pmuysv nykw

Pack my box with five dozen liquor jugs

Caesar 26: Pack my box with five dozen liquor jugs

Caesar 25: Qbdl nz cpy xjui gjwf epafo mjrvps kvht

Caesar 24: Rcem oa dqz ykvj hkxg fqbgp nkswqt lwiu

Caesar 23: Sdfn pb era zlwk ilyh grchq oltxru mxjv

Caesar 22: Tego qc fsb amxl jmzi hsdir pmuysv nykw

Caesar 21: Ufhp rd gtc bnym knaj itejs qnvztw ozlx

Caesar 20: Vgiq se hud cozn lobk jufkt rowaux pamy

Caesar 19: Whjr tf ive dpao mpcl kvglu spxbvy qbnz

Caesar 18: Xiks ug jwf eqbp nqdm lwhmv tqycwz rcoa

Caesar 17: Yjlt vh kxg frcq oren mxinw urzdxa sdpb

Caesar 16: Zkmu wi lyh gsdr psfo nyjox vsaeyb teqc

Caesar 15: Alnv xj mzi htes qtgp ozkpy wtbfzc ufrd

Caesar 14: Bmow yk naj iuft ruhq palqz xucgad vgse

Caesar 13: Cnpx zl obk jvgu svir qbmra yvdhbe whtf

Caesar 12: Doqy am pcl kwhv twjs rcnsb zweicf xiug

Caesar 11: Eprz bn qdm lxiw uxkt sdotc axfjdg yjvh

Caesar 10: Fqsa co ren myjx vylu tepud bygkeh zkwi

Caesar 9: Grtb dp sfo nzky wzmv ufqve czhlfi alxj

Caesar 8: Hsuc eq tgp oalz xanw vgrwf daimgj bmyk

Caesar 7: Itvd fr uhq pbma ybox whsxg ebjnhk cnzl

Caesar 6: Juwe gs vir qcnb zcpy xityh fckoil doam

Caesar 5: Kvxf ht wjs rdoc adqz yjuzi gdlpjm epbn

Caesar 4: Lwyg iu xkt sepd bera zkvaj hemqkn fqco

Caesar 3: Mxzh jv ylu tfqe cfsb alwbk ifnrlo grdp

Caesar 2: Nyai kw zmv ugrf dgtc bmxcl jgosmp hseq

Caesar 1: Ozbj lx anw vhsg ehud cnydm khptnq itfr

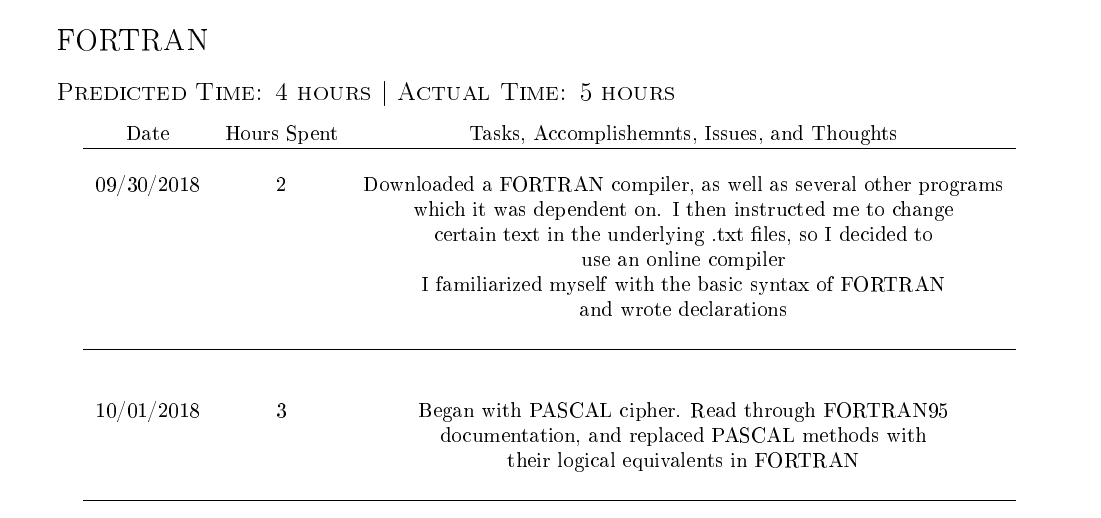
Caesar 0: Pack my box with five dozen liquor jugs

Commentary:

Pascal was the language I decided to program in first. From my understanding, Pascal was the language most closely related to languages I had worked in before. While Pascal was certainly the language I spent the most time working in, this was, in a large part, due to the initial learning curve, including determining the proper way to implement a Caesar Cipher, then translating the modern syntax to that of Pascal. Pascal was also the only language I tested using an installed compiler along with the command line. While this method allows your work to be self-contained on your machine, for the purposes of this assignment, an online compiler is much more convenient, and useful, giving instant, readable feedback, which speeds up the process of debugging greatly. I discovered this upon starting programming in Fortran, and definitely regret not using the online Pascal compiler initially. TutorialsPoint offered many of these online compilers as a free service to visitors of the site.

What first struck me about the syntax of Pascal was the surprising lack of boundaries separating section in the code. Modern programming techniques seem to use brackets as separators much more than the languages worked with in this assignment. The ‘BEGINNING’ and ‘END’ keywords performed a similar function to brackets, however did not seem as necessary for proper functionality, as it might be in Java, for example. In order to take care of the encrypt function, I read in the documentation that two intrinsic functions exist in pascal for the manipulation of characters in a string. The first of these functions (Alphabetically) is the *chr()* function, which returns the ASCII character value of a given integer. The *ord()* function is the inverse of the *chr()* function, returning the assigned ASCII integer value for a given character value. The math and setting up the equation was fairly simple, aside from some errors in counting parenthesis. The decrypt function was easily implemented after this, with the exception of a problem with characters which round the alphabet from ‘Z’ to ‘A’ in their shift. I ultimately figured out the solution when working in Scala and implemented it here. I had trouble with processing the inputs of both upper and lower case characters. To fix this, I looked into the use of case statements, and created a case for both upper and lower-case characters to be processed properly. The solve function was relatively easy aside from an issue I had passing the wrong variable into the parameters. This definitely took me the longest, but what I figured out by finishing this cipher made the others much easier to implement

**FORTRAN Programming language:**

****

CODE:

///////////////////////////////////////////////////////////////////

// Jeffrey Lupia

// Caesar Cipher

// Theory Of Programming languages

// Programming in the past

//////////////////// FORTRAN \\\\\\\\\\\\\\\\\\\\\\\\\

program CaesarCipher

implicit none

Character(len = 39) :: encryptedVal, encrypt, decrypt, decryptedVal, input

input = 'Pack my box with five dozen liquor jugs'

encryptedVal = encrypt(input, 1)

decryptedVal = decrypt(encryptedVal, 1)

print \*, 'The message is: ', input

print \*, 'The encrypted message is: ', encryptedVal

print \*, 'The decrypted message is: ', decryptedVal

call solve(input, 26)

end program CaesarCipher

//////////////////////////////////////////////////////////////

function encrypt(input, shiftValue)

implicit none

Character(len = 39) :: encrypt

Character(len = \*), intent(in) :: input

integer :: shiftValue, cnt

encrypt = input

cnt = 1

do while(cnt <= len(encrypt))

if(encrypt(cnt:cnt) >= 'A' .and. encrypt(cnt:cnt) <= 'Z') then

encrypt(cnt:cnt) = achar(iachar('A') + modulo((iachar(encrypt(cnt:cnt)) - iachar('A')+ shiftValue), 26))

else if(encrypt(cnt:cnt) >= 'a' .and. encrypt(cnt:cnt) <= 'z') then

encrypt(cnt:cnt) = achar(iachar('a') + modulo((iachar(encrypt(cnt:cnt)) - iachar('a')+ shiftValue), 26))

end if

cnt = cnt + 1

end do

end function encrypt

////////////////////////////////////////////////////////

function decrypt(input, shiftValue)

implicit none

Character(len = 39) :: decrypt

Character(len = \*), intent(in) :: input

integer :: shiftValue, cnt

decrypt = input

cnt = 1

do while(cnt <= len(decrypt))

if(decrypt(cnt:cnt) >= 'A' .and. decrypt(cnt:cnt) <= 'Z') then

decrypt(cnt:cnt) = achar(iachar('A') + modulo((iachar(decrypt(cnt:cnt)) - iachar('A')- shiftValue), 26))

else if(decrypt(cnt:cnt) >= 'a' .and. decrypt(cnt:cnt) <= 'z') then

decrypt(cnt:cnt) = achar(iachar('a') + modulo((iachar(decrypt(cnt:cnt)) - iachar('a')- shiftValue), 26))

end if

cnt = cnt + 1

end do

end function decrypt

/////////////////////////////////////////////////////////

subroutine solve(input, maxShiftVal)

implicit none

Character(len = 39) :: encrypt

Character(len = \*), intent(in) :: input

integer :: maxShiftVal, cnt

cnt = 0

do while(cnt <= maxShiftVal)

print \*, 'Caesar ', (maxShiftVal - cnt), ': ', encrypt(input, cnt)

cnt = cnt + 1

end do

end subroutine solve

OUTPUT:

**$gfortran -std=f95 \*.f95 -o main**  
**$main**  
 The message is: Pack my box with five dozen liquor jugs

The encrypted message is: Qbdl nz cpy xjui gjwf epafo mjrvps kvht

The decrypted message is: Pack my box with five dozen liquor jugs

Caesar 26 : Pack my box with five dozen liquor jugs

Caesar 25 : Qbdl nz cpy xjui gjwf epafo mjrvps kvht

Caesar 24 : Rcem oa dqz ykvj hkxg fqbgp nkswqt lwiu

Caesar 23 : Sdfn pb era zlwk ilyh grchq oltxru mxjv

Caesar 22 : Tego qc fsb amxl jmzi hsdir pmuysv nykw

Caesar 21 : Ufhp rd gtc bnym knaj itejs qnvztw ozlx

Caesar 20 : Vgiq se hud cozn lobk jufkt rowaux pamy

Caesar 19 : Whjr tf ive dpao mpcl kvglu spxbvy qbnz

Caesar 18 : Xiks ug jwf eqbp nqdm lwhmv tqycwz rcoa

Caesar 17 : Yjlt vh kxg frcq oren mxinw urzdxa sdpb

Caesar 16 : Zkmu wi lyh gsdr psfo nyjox vsaeyb teqc

Caesar 15 : Alnv xj mzi htes qtgp ozkpy wtbfzc ufrd

Caesar 14 : Bmow yk naj iuft ruhq palqz xucgad vgse

Caesar 13 : Cnpx zl obk jvgu svir qbmra yvdhbe whtf

Caesar 12 : Doqy am pcl kwhv twjs rcnsb zweicf xiug

Caesar 11 : Eprz bn qdm lxiw uxkt sdotc axfjdg yjvh

Caesar 10 : Fqsa co ren myjx vylu tepud bygkeh zkwi

Caesar 9 : Grtb dp sfo nzky wzmv ufqve czhlfi alxj

Caesar 8 : Hsuc eq tgp oalz xanw vgrwf daimgj bmyk

Caesar 7 : Itvd fr uhq pbma ybox whsxg ebjnhk cnzl

Caesar 6 : Juwe gs vir qcnb zcpy xityh fckoil doam

Caesar 5 : Kvxf ht wjs rdoc adqz yjuzi gdlpjm epbn

Caesar 4 : Lwyg iu xkt sepd bera zkvaj hemqkn fqco

Caesar 3 : Mxzh jv ylu tfqe cfsb alwbk ifnrlo grdp

Caesar 2 : Nyai kw zmv ugrf dgtc bmxcl jgosmp hseq

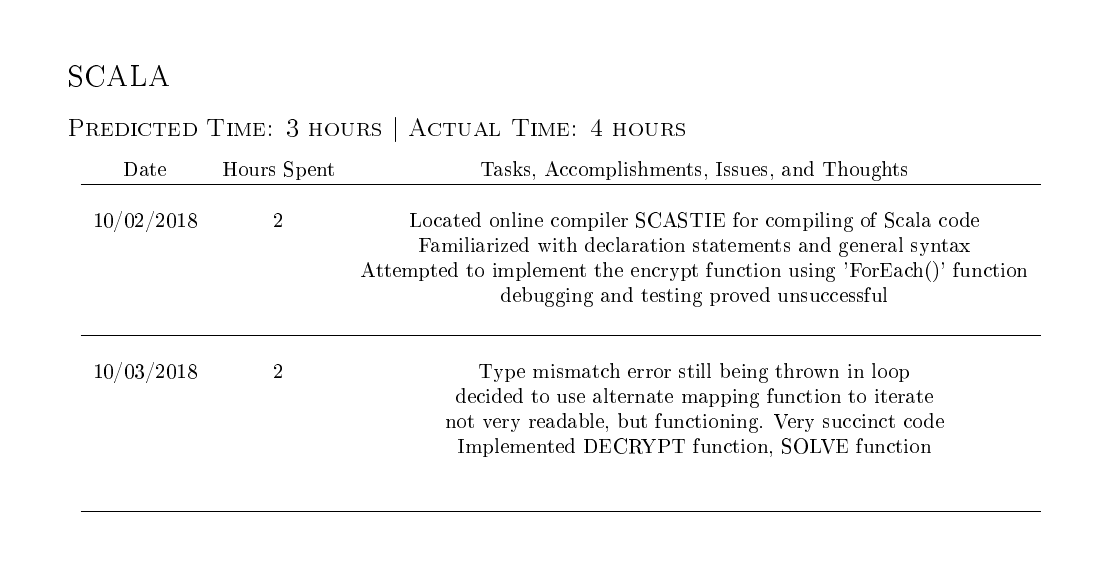
Caesar 1 : Ozbj lx anw vhsg ehud cnydm khptnq itfr

Caesar 0 : Pack my box with five dozen liquor jugs

COMMENTARY:

Using Fortran kind of sucked… having finished Pascale before it, the Syntax differences were a big adjustment, but I have been becoming better at implementing the new languages as I’ve worked on this project. After you do several Caesar Ciphers, they get easier. The most irritating part of Fortran, in my opinion, is the constant necessity to explicitly declare each variable as well as its data type with each instance of use. Additionally one thing that tripped me up about Fortran was the difference between subroutines and functions. I initially wrote my ENCRYPT() and DECRYPT() functions as subroutines before I realized they needed to be switched.

**SCALA Programming language:**

****

CODE:

///////////////////////////////////////////////////////////////////

// Jeffrey Lupia

// Caesar Cipher

// Theory Of Programming languages

// Programming in the past

/////////////////// SCALA \\\\\\\\\\\\\\\\\\\\\\\\\

object CaesarCipher{

/\* Scala Caesar Cipher by Jeff Lupia \*/

def main(args: Array[String]) {

var input: String = "Pack my box with five dozen liquor jugs";

var encryptedStr : String = encrypt(input, 7);

println("Encrypted: " + encrypt(input, 7));

println("Decrypted: " + decrypt(encryptedStr, 7));

println(" ");

solve(input, 26);

}

def encrypt(input:String, shiftFactor:Int) : String = {

var encrypted: String = input;

encrypted = encrypted.map(

c => {

if(c >= 'A' && c <= 'Z'){

('A'.toInt + (c.toInt - ('A'.toInt)+ shiftFactor)%26).toChar;

}

else if(c >= 'a' && c <= 'z'){

('a'.toInt + (c.toInt - ('a'.toInt)+ shiftFactor)%26).toChar;

}

else{

c;

}

});

return encrypted;

}

def decrypt(input:String, shiftFactor:Int) : String = {

var decrypted: String = input;

decrypted = decrypted.map(

c => {

if(c >= 'A' && c <= 'Z'){

('A'.toInt + (c.toInt - ('A'.toInt)- shiftFactor + 26)%26).toChar;

}

else if(c >= 'a' && c <= 'z'){

('a'.toInt + (c.toInt - ('a'.toInt)- shiftFactor + 26)%26).toChar;

}

else{

c;

}

});

return decrypted;

}

def solve (input:String, maxShiftValue:Int): Unit = {

var shiftValue : Integer = 0;

for(shiftValue <- 0 to maxShiftValue){

println("Caesar " + (maxShiftValue - shiftValue) +": "+encrypt(input, shiftValue));

}

}

}

OUTPUT:

Encrypted: Whjr tf ive dpao mpcl kvglu spxbvy qbnz

Decrypted: Pack my box with five dozen liquor jugs

Caesar 26: Pack my box with five dozen liquor jugs

Caesar 25: Qbdl nz cpy xjui gjwf epafo mjrvps kvht

Caesar 24: Rcem oa dqz ykvj hkxg fqbgp nkswqt lwiu

Caesar 23: Sdfn pb era zlwk ilyh grchq oltxru mxjv

Caesar 22: Tego qc fsb amxl jmzi hsdir pmuysv nykw

Caesar 21: Ufhp rd gtc bnym knaj itejs qnvztw ozlx

Caesar 20: Vgiq se hud cozn lobk jufkt rowaux pamy

Caesar 19: Whjr tf ive dpao mpcl kvglu spxbvy qbnz

Caesar 18: Xiks ug jwf eqbp nqdm lwhmv tqycwz rcoa

Caesar 17: Yjlt vh kxg frcq oren mxinw urzdxa sdpb

Caesar 16: Zkmu wi lyh gsdr psfo nyjox vsaeyb teqc

Caesar 15: Alnv xj mzi htes qtgp ozkpy wtbfzc ufrd

Caesar 14: Bmow yk naj iuft ruhq palqz xucgad vgse

Caesar 13: Cnpx zl obk jvgu svir qbmra yvdhbe whtf

Caesar 12: Doqy am pcl kwhv twjs rcnsb zweicf xiug

Caesar 11: Eprz bn qdm lxiw uxkt sdotc axfjdg yjvh

Caesar 10: Fqsa co ren myjx vylu tepud bygkeh zkwi

Caesar 9: Grtb dp sfo nzky wzmv ufqve czhlfi alxj

Caesar 8: Hsuc eq tgp oalz xanw vgrwf daimgj bmyk

Caesar 7: Itvd fr uhq pbma ybox whsxg ebjnhk cnzl

Caesar 6: Juwe gs vir qcnb zcpy xityh fckoil doam

Caesar 5: Kvxf ht wjs rdoc adqz yjuzi gdlpjm epbn

Caesar 4: Lwyg iu xkt sepd bera zkvaj hemqkn fqco

Caesar 3: Mxzh jv ylu tfqe cfsb alwbk ifnrlo grdp

Caesar 2: Nyai kw zmv ugrf dgtc bmxcl jgosmp hseq

Caesar 1: Ozbj lx anw vhsg ehud cnydm khptnq itfr

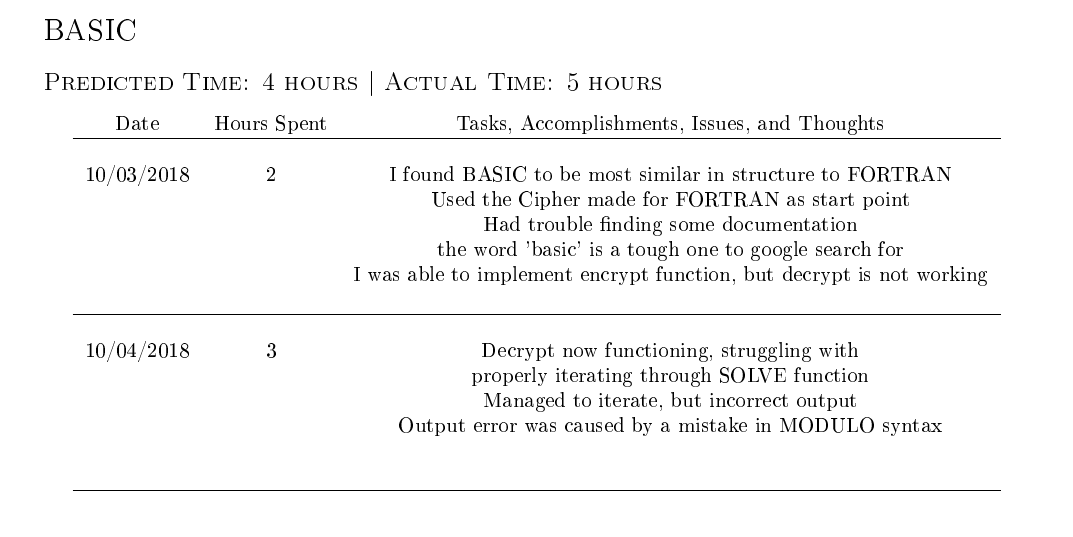
Caesar 0: Pack my box with five dozen liquor jugs

COMMENTARY:

The first problem I ran into when using SCALA was struggling with the best way to mimic a modern for-loop. At first I tried using the ‘foreach()’ function for my ENCRYPT() and DECRYPT() functions in order to iterate through the string. I was continually getting type mismatch error's and, though it seem to be outputting the result of the function correctly as a string I cannot get the error to go away and so I am unable to get the compiler to proceed. I decided to look into alternate substitutes for a basic for-loop in SCALA and stumbled upon the map function. While using the map function to iterate through certainly improves the readability of the program, the way in which the map function is implemented significantly decreases the readability and makes it difficult to follow the flow of data. I had to look over it several times before I understood its implementation enough to finish the DECRYPT() function.

I found SCALA to be the language in this assignment that was most similar to the modern languages we've already learned. Scala, being an object oriented language, is very similar to the syntax that JAVA uses. Certain differences such as the declaration of the IF statements and parameters were much easier to declare then some of the other lower level languages from this assignment.

**BASIC Programming language:**

****

CODE:

///////////////////////////////////////////////////////////////////

// Jeffrey Lupia

// Caesar Cipher

// Theory Of Programming languages

// Programming in the past

/////////////////// BASIC \\\\\\\\\\\\\\\\\\\\\\\\\

input$ = "Pack my box with five dozen liquor jugs"

print "Input: ", input$

print "Encrypted String: ", encrypt$(input$, 1)

print "Decrypted String: ", decrypt$(encrypt$(input$, 1), 1)

print " "

solve(input$, 26)

sub encrypt$(input$, shiftFactor)

encrypt$ = input$

index = 0

repeat

if(mid$(encrypt$, index, 1) >= "A" and mid$(encrypt$, index, 1) <= "Z") then

mid$(encrypt$, index, 1) = chr$(asc("A") + mod((asc(mid$(encrypt$, index, 1)) - asc("A")+ shiftFactor), 26))

else if(mid$(encrypt$, index, 1) >= "a" and mid$(encrypt$, index, 1) <= "z")

mid$(encrypt$, index, 1) = chr$(asc("a") + mod((asc(mid$(encrypt$, index, 1)) - asc("a")+ shiftFactor), 26))

end if

index = index + 1

until(index = len(encrypt$))

return encrypt$

end sub

sub decrypt$(input$, shiftFactor)

decrypt$ = input$

index = 0

repeat

if(mid$(decrypt$, index, 1) >= "A" and mid$(decrypt$, index, 1) <= "Z") then

mid$(decrypt$, index, 1) = chr$(asc("A") + mod((asc(mid$(decrypt$, index, 1)) - asc("A")- shiftFactor +26), 26))

else if(mid$(decrypt$, index, 1) >= "a" and mid$(decrypt$, index, 1) <= "z")

mid$(decrypt$, index, 1) = chr$(asc("a") + mod((asc(mid$(decrypt$, index, 1)) - asc("a")- shiftFactor +26), 26))

end if

index = index + 1

until(index = len(decrypt$))

return decrypt$

end sub

sub solve(input$, maxShiftVal)

loopCt = 0

repeat

print "Caesar ", (maxShiftVal - loopCt), ": ", encrypt$(input$, loopCt)

loopCt = loopCt + 1

until(loopCt > maxShiftVal)

end sub

OUTPUT:

**$yabasic main.bas**  
Input: Pack my box with five dozen liquor jugs

Encrypted String: Qbdl nz cpy xjui gjwf epafo mjrvps kvhs

Decrypted String: Pack my box with five dozen liquor jugs

Caesar 26: Pack my box with five dozen liquor jugs

Caesar 25: Qbdl nz cpy xjui gjwf epafo mjrvps kvhs

Caesar 24: Rcem oa dqz ykvj hkxg fqbgp nkswqt lwis

Caesar 23: Sdfn pb era zlwk ilyh grchq oltxru mxjs

Caesar 22: Tego qc fsb amxl jmzi hsdir pmuysv nyks

Caesar 21: Ufhp rd gtc bnym knaj itejs qnvztw ozls

Caesar 20: Vgiq se hud cozn lobk jufkt rowaux pams

Caesar 19: Whjr tf ive dpao mpcl kvglu spxbvy qbns

Caesar 18: Xiks ug jwf eqbp nqdm lwhmv tqycwz rcos

Caesar 17: Yjlt vh kxg frcq oren mxinw urzdxa sdps

Caesar 16: Zkmu wi lyh gsdr psfo nyjox vsaeyb teqs

Caesar 15: Alnv xj mzi htes qtgp ozkpy wtbfzc ufrs

Caesar 14: Bmow yk naj iuft ruhq palqz xucgad vgss

Caesar 13: Cnpx zl obk jvgu svir qbmra yvdhbe whts

Caesar 12: Doqy am pcl kwhv twjs rcnsb zweicf xius

Caesar 11: Eprz bn qdm lxiw uxkt sdotc axfjdg yjvs

Caesar 10: Fqsa co ren myjx vylu tepud bygkeh zkws

Caesar 9: Grtb dp sfo nzky wzmv ufqve czhlfi alxs

Caesar 8: Hsuc eq tgp oalz xanw vgrwf daimgj bmys

Caesar 7: Itvd fr uhq pbma ybox whsxg ebjnhk cnzs

Caesar 6: Juwe gs vir qcnb zcpy xityh fckoil doas

Caesar 5: Kvxf ht wjs rdoc adqz yjuzi gdlpjm epbs

Caesar 4: Lwyg iu xkt sepd bera zkvaj hemqkn fqcs

Caesar 3: Mxzh jv ylu tfqe cfsb alwbk ifnrlo grds

Caesar 2: Nyai kw zmv ugrf dgtc bmxcl jgosmp hses

Caesar 1: Ozbj lx anw vhsg ehud cnydm khptnq itfs

Caesar 0: Pack my box with five dozen liquor jugs

COMMENTARY:

I feel like BASIC and I have, really come a long way since we first met. Much like Marcus Zimmermann when we were 15 taking in the foothills of Germany, when I first met BASIC I honestly kind of hated it. This was partly because of the heretical lack of any kind of boundaries between different sections of the code. It was unsettling to me, but once I begin looking into the declarations of functions and variables, I started to really enjoying the syntax of BASIC. The declaration of functions and variables using the ‘$’ to indicate a string return or just a standard variable name to indicate an integer return. Additionally the lack of strictly defined types for each variable made the debugging of the code much easier. I didn't have to spend all my time making sure I didn't get mismatched type errors. I much prefer programming in less rigid languages like this as opposed to some of the more modern structured languages.