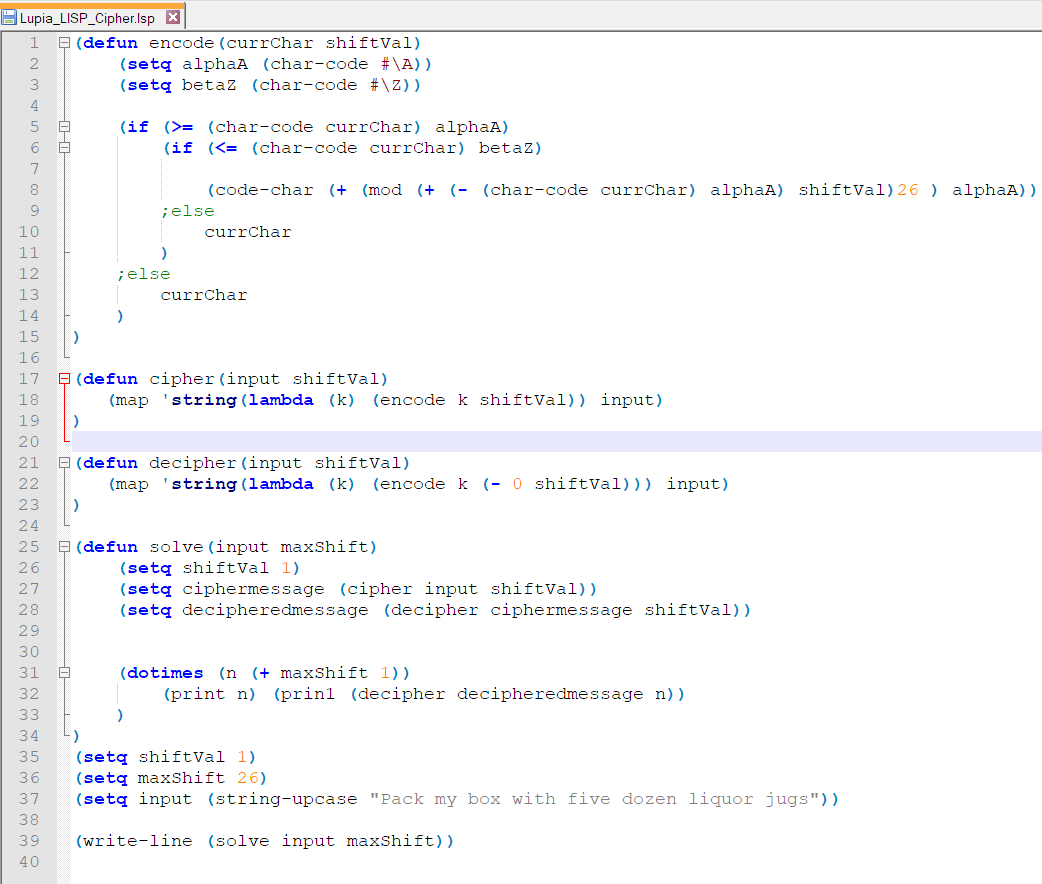


|  |  |  |
| --- | --- | --- |
| **LISP Programming Language:** | | |
| **Predicted Time:** 4 Hours | **Actual Time:**  7 Hours | |
| **Date** | **Hrs** | **Comments/ Summary** |
| 11/25 | 1.5 | Began programming in LISP- downloaded IDE, But it said It was missing necessary referential files. Retried download, but to no avail. After about 30 minutes, decided to work in online compiler- Tutorialspoint.com  Followed online reading for basic String output program “Hello world”  Unknown error was giving me difficulties moving past the tutorial program |
| 11/25 | 2 | Returned to working on LISP – debugging output program. As it turned out, I was wrapping the whole program in paren’s, which was preventing it from compiling properly- misunderstanding with the concept of functional languages. Did some more reading to realize my mistake. At this point, I began working on the encrypt function  I was having trouble dealing with specific chars all in one func, so I created an ENCODE function, which I would use to deal with chars. Worked with *char-code* & *code-char*  functions, I got into trouble with the use of prefix notation |
| 11/26 | 3.5 | Figured out that the lack of using Prefix properly was the problem I was running into  Got char to change into a new char!  Oh boy, I doesn’t account for lowercase of going past z  I just found that It also doesn’t account for spaces  Okay, I used the *toupper function* while passing the string so I don’t have to worry about that. And made an *if statement* to see if it’s a alphabetical character or not. Also-*Mod function* for passing from z to a  The function can now change the string  I finally figured out the solve function, though the *dotimes loop* took forever to figure out. It’s rather confusing at first. |

****CODE:

Output:

0 "PACK MY BOX WITH FIVE DOZEN LIQUOR JUGS"

1 "OZBJ LX ANW VHSG EHUD CNYDM KHPTNQ ITFR"

2 "NYAI KW ZMV UGRF DGTC BMXCL JGOSMP HSEQ"

3 "MXZH JV YLU TFQE CFSB ALWBK IFNRLO GRDP"

4 "LWYG IU XKT SEPD BERA ZKVAJ HEMQKN FQCO"

5 "KVXF HT WJS RDOC ADQZ YJUZI GDLPJM EPBN"

6 "JUWE GS VIR QCNB ZCPY XITYH FCKOIL DOAM"

7 "ITVD FR UHQ PBMA YBOX WHSXG EBJNHK CNZL"

8 "HSUC EQ TGP OALZ XANW VGRWF DAIMGJ BMYK"

9 "GRTB DP SFO NZKY WZMV UFQVE CZHLFI ALXJ"

10 "FQSA CO REN MYJX VYLU TEPUD BYGKEH ZKWI"

11 "EPRZ BN QDM LXIW UXKT SDOTC AXFJDG YJVH"

12 "DOQY AM PCL KWHV TWJS RCNSB ZWEICF XIUG"

13 "CNPX ZL OBK JVGU SVIR QBMRA YVDHBE WHTF"

14 "BMOW YK NAJ IUFT RUHQ PALQZ XUCGAD VGSE"

15 "ALNV XJ MZI HTES QTGP OZKPY WTBFZC UFRD"

16 "ZKMU WI LYH GSDR PSFO NYJOX VSAEYB TEQC"

17 "YJLT VH KXG FRCQ OREN MXINW URZDXA SDPB"

18 "XIKS UG JWF EQBP NQDM LWHMV TQYCWZ RCOA"

19 "WHJR TF IVE DPAO MPCL KVGLU SPXBVY QBNZ"

20 "VGIQ SE HUD COZN LOBK JUFKT ROWAUX PAMY"

21 "UFHP RD GTC BNYM KNAJ ITEJS QNVZTW OZLX"

22 "TEGO QC FSB AMXL JMZI HSDIR PMUYSV NYKW"

23 "SDFN PB ERA ZLWK ILYH GRCHQ OLTXRU MXJV"

24 "RCEM OA DQZ YKVJ HKXG FQBGP NKSWQT LWIU"

25 "QBDL NZ CPY XJUI GJWF EPAFO MJRVPS KVHT"

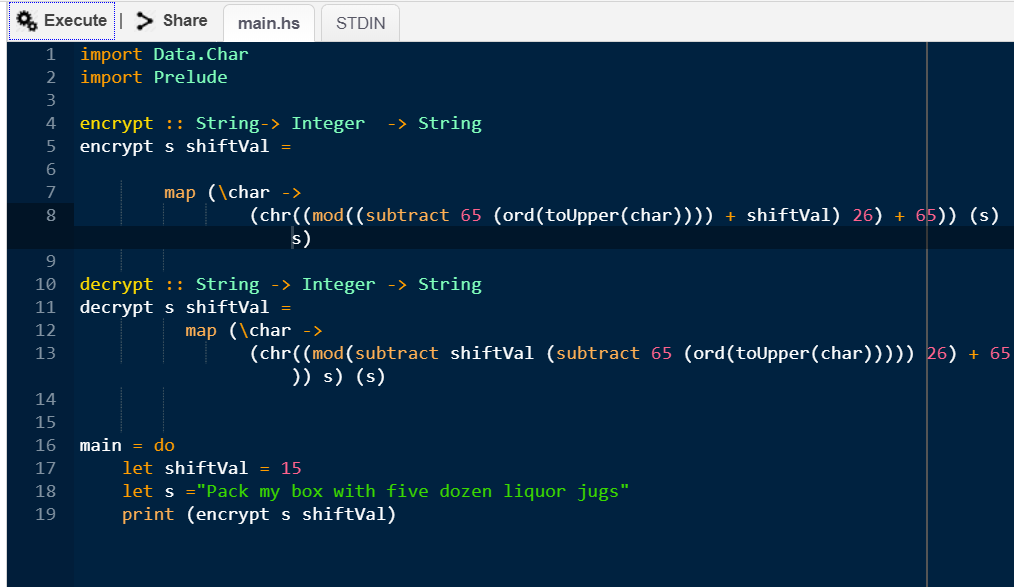
26 "PACK MY BOX WITH FIVE DOZEN LIQUOR JUGS"

Commentary:

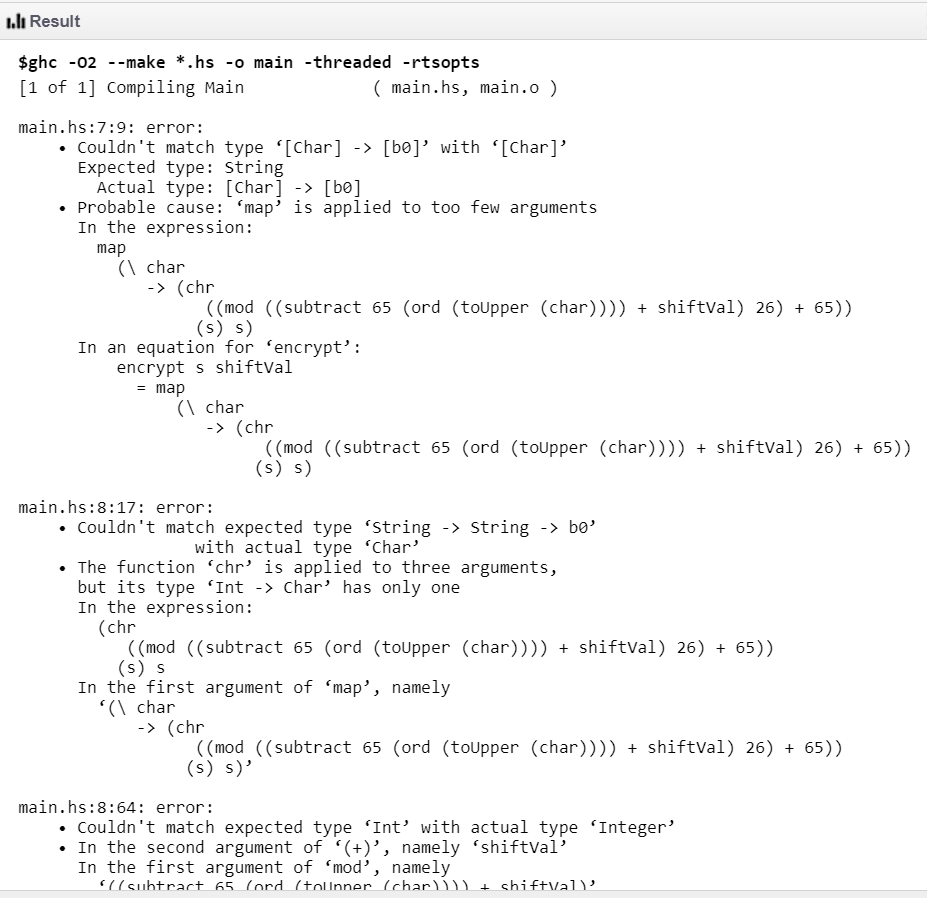
LISP was the language I decided to program in first. My understanding on the difference between functional languages and traditional programming languages was limited going into this assignment. The best practice to start, would likely have been to read more deeply into the theory behind functional languages, so I would be able to better understand them in the context of what I had already learned. Initially, however, I had decided to delve right into readings on the specific syntax of Lisp. I first visited tutorialsPoint.com to read up on some of the examples of LISP programs. They had suggested initially, the downloading of an IDE, after which the program could be executed through the command prompt. After spending almost 30 minutes, in an attempt to download and properly run the IDE, I decided that My time would be better spent working through the code. Initially, I looked into how I would execute a simple “hello World” program in LISP Language. My program ended up printing ”It WORKS! It’s Actually Functioning Properly!!” this took more time than I had expected it too, as I was making a simple mistake because I did not quite understand the theory behind the languages. Ultimately one of the biggest hurdles in understanding lisp, was the realization that it is ultimately one large expression, surrounded in parentheses being evaluated in a mathematical fashion according to those parentheses. It took me over an hour of debugging, looking up documentation, and online searching to realize that this was where I was having a lot of my difficulties. Once this understood, reading the documentation and examples of programs in LISP was much easier to understand. Ultimately my lisp Program was able to break the Caesar cipher. I was honestly so excited when I got the output correct after about an hour of debugging, I literally ran into G’s room excitedly to tell him.

|  |  |  |
| --- | --- | --- |
| **Haskell Programming Language:** | | |
| **Predicted Time:** 4 Hours | **Actual Time:**  5 hours | |
| **Date** | **Hrs** | **Comments/ Summary** |
| 11/26 | 1 | The option to import packages known as “modules” which contain data needed to perform certain functions such as map, and dealing with character types even. This was almost reminiscent of Java, when importing needed packages like math at the beginning of the code. |
| 11/26 | 3 | The syntax for this is very strange. It’s really type dependent.  I also find it odd that the types are declared before the actual functions. Normally I would think of the types as an attribute given to a variable, but in this example, the type declarations acted almost like constructors, with the variable name being assigned to the respective, predefined type as it is read by the program. |
| 11/27 | 1 | I built a function which, according to my reading into the documentation for Haskell, I believe should work properly to encrypt a string. After spending an hour trying to debug type-mismatch errors, which I couldn’t get to work for an unknown reason, I decided that my time would be better spent working on other languages, and hopefully I could come back to this. |

CODE:



OUTPUT:



COMMENTARY:

My first observation when using Haskell was that the language was the specificity of the type declarations. Upon declaring a function, the type of the parameters as well as output were defined before the function was actually written. They were assigned using an “->” symbol to follow the type progression which this language most closely sticks to. If there was one thing to point to which gave me the most difficulty while programming in Haskell, it would have to be dealing with the debugging of type mismatch errors. Honestly, one of the main reasons I decided to move on from Haskell and spend my time working on other languages, was because of the difficulty and tediousness of the type mismatches, and declarations. At many times, when I was working in Haskell, it almost felt like I was programming backwards in a way; this is in regards to the mathematical-kind of logic, the language follows as it evaluates the expressions.

|  |  |  |
| --- | --- | --- |
| **(Functional) SCALA Programming Language:** | | |
| **Predicted Time: 2** Hours | **Actual Time:**  1.5 hours | |
| **Date** | **Hrs** | **Comments/ Summary** |
| 11/27 | 1.5 | Begin by looking up the diff between functional and non functional (variables cant be reassigned in functional language)  Started with my previous code as a jumping-off point Implementation of a map function  This let me not have to reassign variables  Basic debugging to make the map function work  Surprisingly simple changes |

CODE:

objectCaesarCipher**{**

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

defmain**(**args**:** Array**[**String**]) {**

varinput**:** String="Pack my box with five dozen liquor jugs"**;**

varencryptedStr **:** String=encrypt**(**input**,** 7**);**

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

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

println**(**" "**);**

solve**(**input**,** 26**);**

**}**

defencrypt**(**input**:**String**,** shiftFactor**:**Int**) :** String= **{**

varencrypted**:** String=input**;**

encrypted**.**map**(**

c=> **{**

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

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

**}**

elseif**(**c>='a'&&c<='z'**){**

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

**}**

else**{**

c**;**

**}**

**});**

**}**

defdecrypt**(**input**:**String**,** shiftFactor**:**Int**) :** String= **{**

vardecrypted**:** String=input**;**

decrypted**.**map**(**

c=> **{**

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

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

**}**

elseif**(**c>='a'&&c<='z'**){**

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

**}**

else**{**

c**;**

**}**

**});**

**}**

defsolve **(**input**:**String**,** maxShiftValue**:**Int**):** Unit= **{**

varshiftValue **:** Integer=0**;**

for**(**shiftValue<-0tomaxShiftValue**){**

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

**}**

**}**

**}**

OUTPUT:Console

1. Encrypted: Whjr tf ive dpao mpcl kvglu spxbvy qbnz
2. Decrypted: Pack my box with five dozen liquor jugs
4. Caesar 26: Pack my box with five dozen liquor jugs
5. Caesar 25: Qbdl nz cpy xjui gjwf epafo mjrvps kvht
6. Caesar 24: Rcem oa dqz ykvj hkxg fqbgp nkswqt lwiu
7. Caesar 23: Sdfn pb era zlwk ilyh grchq oltxru mxjv
8. Caesar 22: Tego qc fsb amxl jmzi hsdir pmuysv nykw
9. Caesar 21: Ufhp rd gtc bnym knaj itejs qnvztw ozlx
10. Caesar 20: Vgiq se hud cozn lobk jufkt rowaux pamy
11. Caesar 19: Whjr tf ive dpao mpcl kvglu spxbvy qbnz
12. Caesar 18: Xiks ug jwf eqbp nqdm lwhmv tqycwz rcoa
13. Caesar 17: Yjlt vh kxg frcq oren mxinw urzdxa sdpb
14. Caesar 16: Zkmu wi lyh gsdr psfo nyjox vsaeyb teqc
15. Caesar 15: Alnv xj mzi htes qtgp ozkpy wtbfzc ufrd
16. Caesar 14: Bmow yk naj iuft ruhq palqz xucgad vgse
17. Caesar 13: Cnpx zl obk jvgu svir qbmra yvdhbe whtf
18. Caesar 12: Doqy am pcl kwhv twjs rcnsb zweicf xiug
19. Caesar 11: Eprz bn qdm lxiw uxkt sdotc axfjdg yjvh
20. Caesar 10: Fqsa co ren myjx vylu tepud bygkeh zkwi
21. Caesar 9: Grtb dp sfo nzky wzmv ufqve czhlfi alxj
22. Caesar 8: Hsuc eq tgp oalz xanw vgrwf daimgj bmyk
23. Caesar 7: Itvd fr uhq pbma ybox whsxg ebjnhk cnzl
24. Caesar 6: Juwe gs vir qcnb zcpy xityh fckoil doam
25. Caesar 5: Kvxf ht wjs rdoc adqz yjuzi gdlpjm epbn
26. Caesar 4: Lwyg iu xkt sepd bera zkvaj hemqkn fqco
27. Caesar 3: Mxzh jv ylu tfqe cfsb alwbk ifnrlo grdp
28. Caesar 2: Nyai kw zmv ugrf dgtc bmxcl jgosmp hseq
29. Caesar 1: Ozbj lx anw vhsg ehud cnydm khptnq itfr
30. Caesar 0: Pack my box with five dozen liquor jugs

COMMENTARY:

I was surprised to discover that the implementation of the Scala programming language was surprisingly similar to the non-functional implementation. After doing some research on the differences between functional and non-functional SCALA implementations, I found that starting with my previous code as a jumping-off point was tremendously helpful. The real challenge was the implementation of a map function. This allowed me to bypass the reassignment of variables(which would not be present in a functional implementation of SCALA) there was surprisingly little debugging needed to make it function after this.

|  |  |  |
| --- | --- | --- |
| **ML Programming Language:** | | |
| **Predicted Time: 3** Hours | **Actual Time:**  4.5 hours | |
| **Date** | **Hrs** | **Comments/ Summary** |
| 11/29 | 4.5 | Could not find online compiler for ML. spent a while looking until I discovered that the best method is to use the SMLNJ implementation of ML  SMLNJ, had documentation much easier to find and more succinct that traditional ML  The declaration of variables, along with their types threw me off  A big difficulty I had was with declaring the variables within the correct scope, as these values could not be reassigned  spent several hours debugging the errors it was outputting-keeps returning new errors…  Ultimately was unable to get the encrypt or decrypt function to execute properly…. |

CODE:

let

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

val aMinor := ord #"a";

(\* 97\*)

val zMinor := ord #"z";

val AMajor := ord #"A";

(\*65\*)

val ZMajor := ord #"Z";

val count: int := 0

val shiftFactor: int := 15

val charTemp: char := sub(input, count)

val ordValTemp: int := ord charTemp

val lowercase: int := (((ordValTemp - aMinor + shiftFactor)mod(26)) + AMajor)

val uppercase: int := (((ordValTemp - AMajor + shiftFactor)mod(26)) + AMajor)

val result1: char := chr lowercase;

val result2: char := chr upperrcase;

val resultList: char list;

fun encrypt (val s: string, val shft: int)

(shft:= shiftFactor)

(s := input

val charList := explode(s)

while

(count < (length charList))

do

if (ordValTemp >= aMinor)

then

(ordValTemp := lowercase)

(result := chr ordValTemp)

else

ordValTemp := uppercase;

nth(charList, count)

count := (count+1);

end;

OUTPUT:



COMMENTARY:

This was a tough language to work in for me. I initially had trouble finding proper documentation on the language, partially because of the obscurity of the two-letter name in the context of a google search. After being informed that the best method of finding documentation for SMLNJ, it was much easier to find non-conflicting documentation. Despite this, the examples I was able to find of the proper execution of ML seemed to be very much simplified, and beautified for efficiency, however, this made many of the examples almost unfollowable because of their decreased readability. Ultimately my attempted implementation of the ML Caesar cipher was greeted with nothing but type errors and general syntax errors, but after spending several hours debugging the errors it was giving to me to no avail, or returning new errors, I had to give up because It was sunrise, and I was no longer making notable progress