Functional Programming

Colin MacDonald

May 14, 2021

1 Caesar Cipher

1.1 LISP

1.1.1 Code

```
;Colin MacDonald
;Run this online: https://www.jdoodle.com/ia/dAF
(defun encryptMath (chr )
    (setq asciCode (char-code chr))
    (cond ((and (>= asciCode 65)
           (<= asciCode 91))
    (code-char(+(mod(+(- asciCode 65)shiftAmount)26)65)))
    ((and (>= asciCode 97)
           (<= asciCode 122))
    (code-char(+(mod(+(- asciCode 97)shiftAmount)26)97)))
    (t chr))
)

(defun encrypt (string) (map 'string #'encryptMath string ))

(defun decrypt (string)
    (setq shiftAmount(- 26 shiftAmount))
    (encrypt string)
)

(defun solve(str num)
    (cond((>= num 0)
        (setq shiftAmount num)
        (print(concatenate 'string "Caesar "(princ-to-string num)": "(encrypt str)))
        (solve str (- num 1)))
    )
)

(setq shiftAmount 8)
(setq x(encrypt "This is a test string from Alan"))
(print x)
```

```
(setq x(decrypt x))
(print x)

(setq shiftAmount 15)
(setq x(encrypt "The password is passw0rd!"))
(print x)
(setq x(decrypt x))
(print x)

(solve "HAL" 26)
(solve "0cdn kmjbmvhhdib npmz dn api. Jm dn do..." 10)
```

1.1.2 Output

```
"Bpqa qa i bmab abzqvo nzwu Itiv"
"This is a test string from Alan"
"Iwt ephhldgs xh ephhl0gs!"
"The password is passw0rd!"
"Caesar 26: HAL"
"Caesar 25: GZK"
"Caesar 24: FYJ"
"Caesar 23: EXI"
"Caesar 22: DWH"
"Caesar 21: CVG"
"Caesar 20: BUF"
"Caesar 19: ATE"
"Caesar 18: ZSD"
"Caesar 17: YRC"
"Caesar 16: XQB"
"Caesar 15: WPA"
"Caesar 14: VOZ"
"Caesar 13: UNY"
"Caesar 12: TMX"
"Caesar 11: SLW"
"Caesar 10: RKV"
"Caesar 9: QJU"
"Caesar 8: PIT"
"Caesar 7: OHS"
"Caesar 6: NGR"
"Caesar 5: MFQ"
"Caesar 4: LEP"
"Caesar 3: KDO"
"Caesar 2: JCN"
"Caesar 1: IBM"
"Caesar 0: HAL"
"Caesar 10: Ymnx uwtlwfrrnsl xzwj nx kzs. Tw nx ny..."
"Caesar 9: Xlmw tvskveqqmrk wyvi mw jyr. Sv mw mx..."
"Caesar 8: Wklv surjudpplqj vxuh lv ixq. Ru lv lw..."
"Caesar 7: Vjku rtqitcookpi uwtg ku hwp. Qt ku kv..."
"Caesar 6: Uijt qsphsbnnjoh tvsf jt gvo. Ps jt ju..."
"Caesar 5: This programming sure is fun. Or is it..."
"Caesar 4: Sghr oqnfqzllhmf rtqd hr etm. Nq hr hs..."
"Caesar 3: Rfgq npmepykkgle qspc gq dsl. Mp gq gr..."
"Caesar 2: Qefp moldoxjjfkd prob fp crk. Lo fp fq..."
```

```
"Caesar 1: Pdeo lnkcnwiiejc oqna eo bqj. Kn eo ep..."
"Caesar 0: Ocdn kmjbmvhhdib npmz dn api. Jm dn do..."
```

1.2 ML

1.2.1 Code

```
(*Colin MacDonald*)
(*Run this online:
    sosml.org/share/efbc4d518c140af0d870007e7c1fa6fcc39aa4e9cae833e653b9ab28c7272977*)
fun charListToIntList(L: char list) : int list =
  if L = nil then
    nil
  else
    [ord(hd(L))] @ charListToIntList(tl(L))
;

fun strToIntList (str)=
  charListToIntList(explode(str))
;

fun shiftBy(value, shiftAmount)=
  if value >= 65 andalso value <=91 then
    (value-65+shiftAmount) mod 26 +65
  else if value >= 97 andalso value <=122 then
    (value-97+shiftAmount) mod 26 +97
  else value
;

fun encodeHelper(list:int list, shiftAmount:int, newList:int list)=
  if length(list)>0 then
     encodeHelper(tl(list), shiftAmount, newList@[shiftBy(hd(list), shiftAmount)])
  else newList
;

fun encrypt(str:string, shiftAmount:int)=
  implode(map chr (encodeHelper(strToIntList(str), shiftAmount, [])));
;

fun decrypt(str:string, shiftAmount:int)=
  encrypt(str, 26-shiftAmount)
;

fun solveRec(str: string, num: int, list: string list)=
  if num>0 then
    solveRec(str, num-1, list@[encrypt(str, num)])
  else list
;

fun solve(str:string, num:int)=
  print( solveRec(str, num, []))
;

(*--------------------------*)
```

```
val x = encrypt("This is a test string from Alan", 8);
print(x);
val x = decrypt(x, 8);
print(x);


val x = encrypt("The password is passw0rd!", 15);
print(x);
val x = decrypt(x, 15);
print(x);

solve("HAL", 26);
solve("Ocdn kmjbmvhhdib npmz dn api. Jm dn do...", 10);
```

1.2.2 Output

```
(Note: excess output from online IDE trimmed from output)
Printed: Bpqa qa i bmab abzqvo nzwu Itiv
Printed: This is a test string from Alan
Printed: Iwt ephhldgs xh ephhl0gs!
Printed: The password is passw0rd!
Printed: ["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"]
Printed: ["Ymnx uwtlwfrrnsl xzwj nx kzs. Tw nx ny...", "Xlmw tvskveqqmrk wyvi mw jyr. Sv mw mx...",
    "Wklv surjudpplqj vxuh lv ixq. Ru lv lw...", "Vjku rtqitcookpi uwtg ku hwp. Qt ku kv...",
    "Uijt qsphsbnnjoh tvsf jt gvo. Ps jt ju...", "This programming sure is fun. Or is it...",
    "Sghr oqnfqzllhmf rtqd hr etm. Nq hr hs...", "Rfgq npmepykkgle qspc gq dsl. Mp gq gr...",
    "Qefp moldoxjjfkd prob fp crk. Lo fp fq...", "Pdeo lnkcnwiiejc oqna eo bqj. Kn eo ep..."]
```

1.3 Erlang

1.3.1 Code

```
%Colin MacDonald
%Run this online: http://tpcg.io/ogf4zwXQ
-module(helloworld).
-export([start/0]).

shift(Value, ShiftAmount) ->
    if
    Value >= 65 andalso Value =< 91 ->
        (Value-65+ShiftAmount) rem 26 +65;
    Value >= 97 andalso Value =< 122 ->
        (Value-97+ShiftAmount) rem 26 +97;
    true ->
        Value
    end.

encrypt(String, ShiftAmount) ->
   X=lists:map(fun(Val) -> shift(Val, ShiftAmount) end, String),
   io:format("~s~n",[X]).
```

```
decrypt(String, ShiftAmount) ->
    encrypt(String, 26-ShiftAmount).

solve(String, Num) ->
    if Num >=0 ->
        io:fwrite(string:concat(string:concat("Ceaser ", integer_to_list(Num)), ": ")),
        encrypt(String, Num),
        solve(String, Num-1);
    true-> 1+1
    end.

%------------

start() ->
   encrypt("This is a test string from Alan", 8),
   decrypt("Bpqa qa i bmab abzqvo nzwu Itiv", 8),
   encrypt("The password is passw0rd!", 15),
   decrypt("Iwt ephhldgs xh ephhl0gs!", 15),
   solve("HAL", 26),
   solve ("Ocdn kmjbmvhhdib npmz dn api. Jm dn do...", 10).
```

1.3.2 Output

```
Bpqa qa i bmab abzqvo nzwu Itiv
This is a test string from Alan
Iwt ephhldgs xh ephhl0gs!
The password is passw0rd!
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
Ceaser 0: HAL
Ceaser 10: Ymnx uwtlwfrrnsl xzwj nx kzs. Tw nx ny...
Ceaser 9: Xlmw tvskveqqmrk wyvi mw jyr. Sv mw mx...
Ceaser 8: Wklv surjudpplqj vxuh lv ixq. Ru lv lw...
Ceaser 7: Vjku rtqitcookpi uwtg ku hwp. Qt ku kv...
Ceaser 6: Uijt qsphsbnnjoh tvsf jt gvo. Ps jt ju...
Ceaser 5: This programming sure is fun. Or is it...
Ceaser 4: Sghr oqnfqzllhmf rtqd hr etm. Nq hr hs...
Ceaser 3: Rfgq npmepykkgle qspc gq dsl. Mp gq gr...
Ceaser 2: Qefp moldoxjjfkd prob fp crk. Lo fp fq...
Ceaser 1: Pdeo lnkcnwiiejc oqna eo bqj. Kn eo ep...
Ceaser 0: Ocdn kmjbmvhhdib npmz dn api. Jm dn do...
```

1.4 Haskell

1.4.1 Code

```
--Colin MacDonald
--Run this online: https://www.jdoodle.com/ia/dDR
import Data.Char (ord, chr)

encryptMath value shiftAmount =
    if value >= 65 && value <=91 then
        rem (value-65+shiftAmount) 26 + 65
    else if value >= 97 && value <=122 then
        rem (value-97+shiftAmount) 26 + 97
    else value

encrypt string shiftAmount = map (\char -> chr(encryptMath (ord char) shiftAmount)) string

decrypt string shiftAmount = encrypt string (26-shiftAmount)

solve string 0 = putStr("Caesar 0: ") >> print(encrypt string 0)
solve string shiftAmount = putStr(concat[ "Caesar ", (show shiftAmount), ": "]) >> print(encrypt
    string shiftAmount) >> solve string (shiftAmount-1)

main :: IO ()
main = do
let x=encrypt "This is a test string from Alan" 8
print x
let xx= decrypt x 8
print xx
let y=encrypt "The password is passw0rd!" 15
print y
let yy = decrypt y 15
print yy
solve "HAL" 26
solve "Ocdn kmjbmvhhdib npmz dn api. Jm dn do..." 10
```

1.4.2 Output

```
"Bpqa qa i bmab abzqvo nzwu Itiv"
```

```
"This is a test string from Alan"
"Iwt ephhldgs xh ephhl0gs!"
"The password is passw0rd!"
Caesar 26: "HAL"
Caesar 25: "GZK"
Caesar 24: "FYJ"
Caesar 23: "EXI"
Caesar 22: "DWH"
Caesar 21: "CVG"
Caesar 20: "BUF"
Caesar 19: "ATE"
Caesar 18: "ZSD"
Caesar 17: "YRC"
Caesar 16: "XQB"
Caesar 15: "WPA"
Caesar 14: "VOZ"
Caesar 13: "UNY"
Caesar 12: "TMX"
Caesar 11: "SLW"
Caesar 10: "RKV"
Caesar 9: "QJU"
Caesar 8: "PIT"
Caesar 7: "OHS"
Caesar 6: "NGR"
Caesar 5: "MFQ"
Caesar 4: "LEP"
Caesar 3: "KDO"
Caesar 2: "JCN"
Caesar 1: "IBM"
Caesar 0: "HAL"
Caesar 10: "Ymnx uwtlwfrrnsl xzwj nx kzs. Tw nx ny..."
Caesar 9: "Xlmw tvskveqqmrk wyvi mw jyr. Sv mw mx..."
Caesar 8: "Wklv surjudpplqj vxuh lv ixq. Ru lv lw..."
Caesar 7: "Vjku rtqitcookpi uwtg ku hwp. Qt ku kv..."
Caesar 6: "Uijt qsphsbnnjoh tvsf jt gvo. Ps jt ju..."
Caesar 5: "This programming sure is fun. Or is it..."
Caesar 4: "Sghr oqnfqzllhmf rtqd hr etm. Nq hr hs..."
Caesar 3: "Rfgq npmepykkgle qspc gq dsl. Mp gq gr..."
Caesar 2: "Qefp moldoxjjfkd prob fp crk. Lo fp fq..."
Caesar 1: "Pdeo lnkcnwiiejc oqna eo bqj. Kn eo ep..."
Caesar 0: "Ocdn kmjbmvhhdib npmz dn api. Jm dn do..."
```

1.5 Scala

1.5.1 Code

```
//Colin MacDonald
//Run this online: https://www.jdoodle.com/ia/dDT
var x = encrypt("This is a test string from Alan", 8);
println(x);
x = decrypt(x, 8);
println(x);
x=encrypt ("The password is passw0rd!", 15);
println(x);
```

```
x = decrypt(x, 15);
println(x);
solve("HAL", 26);
solve ("Ocdn kmjbmvhhdib npmz dn api. Jm dn do...", 10);

def encrypt(string: String, shiftAmount: Int): String = {
    return string.map(x=>(encryptMath(x.toInt, shiftAmount)).toChar);
}

def encryptMath (value:Int, shiftAmount:Int):Int= {
    if (value >= 65 && value <=91){
         return (value-65+shiftAmount) % 26 + 65; }
    else if (value >= 97 && value <=122){
         return (value-97+shiftAmount) % 26 + 97;}
    else {return value;}
}

def decrypt(string: String, shiftAmount: Int): String = {
    return encrypt(string, 26-shiftAmount);
}

def solve(string: String, shiftAmount: Int):Unit = {
    if(shiftAmount>=0){
        println("Caesar " + shiftAmount + ": " + encrypt(string, shiftAmount));
        solve(string, shiftAmount-1);
    }
}
```

1.5.2 Output

```
Bpqa qa i bmab abzqvo nzwu Itiv
This is a test string from Alan
Iwt ephhldgs xh ephhl0gs!
The password is passw0rd!
Caesar 26: HAL
Caesar 25: GZK
Caesar 24: FYJ
Caesar 23: EXI
Caesar 22: DWH
Caesar 21: CVG
Caesar 20: BUF
Caesar 19: ATE
Caesar 18: ZSD
Caesar 17: YRC
Caesar 16: XQB
Caesar 15: WPA
Caesar 14: VOZ
Caesar 13: UNY
Caesar 12: TMX
Caesar 11: SLW
Caesar 10: RKV
Caesar 9: QJU
Caesar 8: PIT
Caesar 7: OHS
```

```
Caesar 6: NGR
Caesar 5: MFQ
Caesar 4: LEP
Caesar 3: KDO
Caesar 2: JCN
Caesar 1: IBM
Caesar 0: HAL
Caesar 10: Ymnx uwtlwfrrnsl xzwj nx kzs. Tw nx ny...
Caesar 9: Xlmw tvskveqqmrk wyvi mw jyr. Sv mw mx...
Caesar 8: Wklv surjudpplqj vxuh lv ixq. Ru lv lw...
Caesar 7: Vjku rtqitcookpi uwtg ku hwp. Qt ku kv...
Caesar 6: Uijt qsphsbnnjoh tvsf jt gvo. Ps jt ju...
Caesar 5: This programming sure is fun. Or is it...
Caesar 4: Sghr oqnfqzllhmf rtqd hr etm. Nq hr hs...
Caesar 3: Rfgq npmepykkgle qspc gq dsl. Mp gq gr...
Caesar 2: Qefp moldoxjjfkd prob fp crk. Lo fp fq...
Caesar 1: Pdeo lnkcnwiiejc oqna eo bqj. Kn eo ep...
Caesar 0: Ocdn kmjbmvhhdib npmz dn api. Jm dn do...
```

---

2 Log and Commentary

2.0.1 Log

Date	Hours Spent	Tasks / Accomplishments / Issues / Thoughts
5-12-21	1hr 17min	Started lab, began work on learning and coding LISP.
5-13-21	3hr 27min	Finished with LISP. Completed encrypt and decrypt in ML.
5-14-21	4hr 15min	Finished with ML. Did Erlang, Haskell, and Scala.

2.0.2 Commentary

5-12-21

Dear diary, hello again. As you can tell by the date, things have not gone smoothly. In any case, all I can do is try to get everything done. Compared to last time, I am predicting it will take a similar amount of time, putting me at a predicted 12hr 30min of work, an average of 2.5 hours per language. Ignoring the 6 hours COBOL took, the other 4 languages took an average of 1hr 40min each, times 5 is 8hr 10min. I will probably fall into the higher end of that range or higher because I haven't really ever worked with functional languages before, and hopefully nothing is as hard as COBOL was. I'm just going to work on the languages as listed in order.

I'm starting with LISP, specifically CLISP because it appears to be nearly the same thing, and is available on jdoodle, which has become my favorite online IDE.

After an hour of work, I can encrypt a single character using the basic algorithm I learned from the previous project. It is a little tedious to keep track of all the parenthesis, but as long as I am careful, it isn't too bad. Another 10 minutes and I have this moved to a function. I now need to map it to every character or recursively call it on each character in a string. At 1hr 17min I'm stopping and will pick this up in the morning.

5-13-21

I reset the clock, and at 38 minutes, I have a working encrypt function. It maps each individual character of a given string to the encryptMath function. I can't figure out how to also pass in the shiftAmount, so for now it will just be a global variable. At 45 minutes I have the decrypt working by just doing a call to encrypt with an inverted shift amount. And at 1hr 12min, I have completed the solve function. It works by calling encrypt with the provided string and number, then making a recursive call to itself and passing the number minus 1 and original string. This brings my total time on LISP to 2hr 30min, which is exactly the same as the average time I took on each language in the previous project.

It was a little hard to get started with LISP, but overall it wasn't bad. I had some trouble with passing variables and values around, and I wasn't able to figure out passing more than one variable in with the map function. Otherwise, besides learning the new syntax, LISP was alright.

Now onto ML. (Reset the clock.) It took 2 hours, but I have encode and decode working. It took so long because I spent a lot of time trying to get the map function working so it could help with calling encode on each character. Once again I hit the issue where you can't pass in both a character and a shift amount. Finally, I resorted to recursion. The result looks really ugly and it is difficult to tell how it works exactly, but it works perfectly. I started on solve, but at 2hr 15min I stopped for the night.

5-14-21 It is the final countdown and I'm way behind. I will do what I can, but I may just have to turn in an incomplete project. I'm resetting the clock again. After 30 minutes, I was finally able to get solve mostly working. I say mostly because I was having too much difficulty trying to concatenate strings, a number, and a value from the list, so I just decided to print the list itself. Even though the output looks bad, it is correct.

Overall, I hated the fact that functions could only contain a single "line" or statement of code. I really had to get creative with recursion and calling one function when it was passed in as a parameter to another. Perhaps there was another way around that restriction, but it isn't like I will go back and change everything. Otherwise, ML was alright, but not good. It was too restrictive in places, such as type conversion restrictions. I really spent a lot of time just researching how to do things in ML, and it didn't help that documentation wasn't as easily accessed as more popular languages. ML took 2hr 45min to complete.

Onto Erlang now, resetting the clock. At 1hr 40min I have encrypt and decrypt working. I discovered that I could map every character to my shift function by using an anonymous function, and this actually allows me to pass in the shift amount. Strangely enough it takes care of converting to ASCII by itself. However, a list of integers is returned, but I can format it as a string which prints it out right away. At 2 hours I have solve completed, once again with recursion, and also an ugly line which concatenates the necessary info to format the output well.

Erlang was the fastest so far because I understand the general strategy and have figured out when to map and when to use recursion. The language itself was a bit weird at times, such as some of the conversions that just worked, and then other places I have to be really specific about things. It was difficult to understand how to use the anonymous function, but it makes sense now and really shortened the code length. Overall it is ok but not great.

Haskell is next, clock is reset. I did encrypt and decrypt in 55 minutes. Thanks to Erlang, I used the anonymous function strategy with much more ease, which really kept the code short and condensed. In the same area I also managed to bake in conversion to ASCII and back. This was super satisfying to see come together. At 1hr 15min, solve is complete and Haskell is done. Like ML, Haskell only wants one statement per function, but thankfully I discovered that '»' could "cheat" that requirement, which allowed me to combine the output formatting, the encrypt, and a recursive call back to itself.

So far, Haskell is the shortest both in lines of code and time to completion. I actually quite liked programming this one. My experience from the previous three really helped me go down the right path, and quickly. The syntax for declaring parameter types initially looked hard, but once I realise that was optional, it became a lot easier. I really liked the flexibility of not needing to declare types for everything, and the type checking made sure you couldn't get away with crazy things like JavaScript lets you do.

Finally, it is Scala time, and the last clock reset. And...I did the whole thing in 30 minutes. I used the exact same strategy as I did with Haskell. The mapping an anonymous function thing is really nice. My familiarity with Scala and how it looks sort of like Java also helped because I didn't have to learn a whole new syntax. This was by far the shortest by time, though Haskell is still a few lines shorter because the curly braces of Scala made it longer. Not much else I can say because this was fairly easy and super similar to Haskell.

Final Thoughts

The whole coding part of this project took 9 hours. I was more strict this time about only tracking my coding hours and pausing my timer when I worked on this document, but I also spent less time with LaTeX because I already had the formatting figured out. Dividing by 5, each language took an average of 1hr 50min. Ignoring Scala, the other 4 languages took an average of 2hr 10min.

Just from using the averages of last time, my predictions were fairly accurate. Before starting the project, I had a feeling that each language would take longer overall, but I decided to base everything off my previous data. I really didn't expect how after ML, each language took much less time, or that Scala would be done in only 30 minutes.

I think there is a correlation between how much time I took and the level of readability and writability. For example, LISP was ok to write and read, but both were impacted by keeping track of all the parenthesis, so it took a decent amount of time. ML was tough in both aspects, and it took the longest. Erlang was average. Haskell and Scala were easy to read and write, and they took the shortest time.

I used https://www.tutorialspoint.com/ as my main reference for all languages except ML. I used it as a substitute for basic internet searches such as syntax of functions and if statements. It also provides a good quick guide which helped me get introduced to the new languages. I searched the internet for everything else I couldn't find quickly, such as how to convert to ASCII, how to do modulus, how to use the map function, and how to use anonymous functions.

Ranking the languages...well Scala and Haskell go at the top, ML is at the bottom, and LISP and Erlang are in the middle. If I had to really pick an ordering, Haskell just barely goes ahead of Scala because it was new and fun, and LISP just slightly goes ahead of Erlang because I still feel slightly confused by Erlang. That makes the final ranking be Haskell, Scala, LISP, Erlang, ML. At least ML wasn't even close to being as bad as COBOL.

Well, that's a wrap. I really procrastinated on starting this, and stayed up really late to finish, but I can partially blame a heavy workload this time. I do agree that I spent more time thinking and less time writing code. It was fun to see everything come together so well, and in much fewer lines of code.