Functional Programming

Michael Durso
michael.durso1@marist.edu

November 22, 2019

Contents

1 LISP

1.1 The Code

```lisp
;  Michael Durso                                                 ;
;  Professor Labouseur                                           ;
;  CMPT331 - Theory of Programming Languages                     ;
;  Caesar Cipher: The LISP Legend                                ;
;----------------------------------------------------------------;

; Grabs the char-code for lowercase 'a' and 'z' and stores them
; Conditional for shifting letters
(defun shift(caesar shiftAmount)
    (let* ((shifter (char-code caesar))
        (aLow (char-code #\a))(zLow (char-code #\z))
        (base (cond ((and (>= shifter aLow) (<= shifter zLow)) aLow)(t nil))))
        (if base (code-char (+ (mod (+ (- shifter base) shiftAmount) 26) base)) caesar)))

; Encrypt
(defun doEncrypt(str shiftAmount)
    (map 'string #'(lambda (x) (shift x shiftAmount)) str))

; Decrypt
; All this does is doEncrypt with a negative shiftAmount val
(defun doDecrypt(str shiftAmount)
    (doEncrypt str (- shiftAmount)))

; Solve
; Loops the Decrypt function and formats print statements
(defun solve (str maxShift)
    (let ((count 0))
        (loop while (>= maxShift 0) do
            (format t "Caesar ~d: ~a ~%" maxShift (doDecrypt str count))
            (setq maxShift (- maxShift 1))
            (setq count (+ count 1)))))

; main
; Make the things go
(let* ((str "the cake is a lie")(shiftAmount 8)(maxShift 26)(strSolve "hal")
  (encrypt (doEncrypt str shiftAmount))
  (decrypt (doDecrypt encrypt shiftAmount)))
(format t "Original Message: ~a ~%" str)
(format t "Encrypted String: ~a ~%" encrypt)
(format t "Decrypted String: ~a ~%" decrypt)
(solve strSolve maxShift))
```

1.2 The Output

1.2.1 Original Messages

```
Original Message: hal
Original Message: this is a test string from alan
Original Message: the cake is a lie
```

1.2.2 Encrypted Strings

```
Encrypted String: pit
Encrypted String: bpqa qa i bmab abzqvo nzwu itiv
Encrypted String: bpm kism qa i tqm
```

1.2.3 Decrypted Strings

```
Decrypted String: hal
Decrypted String: this is a test string from alan
Decrypted String: the cake is a lie
```

1.2.4 Solve Character Shifts

```
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
```

2 ML

2.1 The Code

```
(*  Michael Durso                                                 *)
(*  Professor Labouseur                                           *)
(*  CMPT331 - Theory of Programming Languages                     *)
(*  Caesar Cipher: The ML Myth                                    *)
(*--------------------------------------------------------------*)


(* shift characters *)
fun shift(letter: char, shiftAmount : int) : char =
    let
           val charCode = Char.ord letter;
        in
           (* Shift if it is any char (not spaces) *)
           if charCode = 32 then Char.chr charCode
           else chr (((charCode - 65 + shiftAmount) mod 26) + 65)
        end;

(* Encrypt *)
(* map each character to a new character and put the string back together *)
fun encrypt (str : string, shiftAmount : int) : string =
    String.implode(map (fn x => shift ((Char.toUpper x), shiftAmount)) (String.explode(str)));

(* Decrypt *)
(* Do Encrypt with a negative shiftAmount*)
fun decrypt(str : string, shiftAmount : int) : string = encrypt(str, (shiftAmount * ~1));

(* Encrypt a string from maxShift to 0 using recursion*)
fun solve(str, maxShift)  = (
    print(("Caesar " ^ Int.toString maxShift)^": " ^ encrypt(str, maxShift)^"\n");
    if maxShift > 0 then solve(str, maxShift-1) else ())


(* The "Make Things Happen" Zone *)
val str = "This is a test string from Alan";
val encryptedString = "BPQA QA I BMAB ABZQVO NZWU ITIV";
print("Original Message: " ^ str ^ "\n");
print("Encrypted String: " ^ encrypt(str, 8) ^ "\n");
print("Decrypted String: " ^ decrypt(encryptedString, 8) ^ "\n");
solve("hal", 26);
```

2.2 The Output

2.2.1 Original Messages

```
Original Message: hal
Original Message: This is a test string from Alan
Original Message: The cake is a lie
```

2.2.2 Encrypted Strings

```
Encrypted String: PIT
Encrypted String: BPQA QA I BMAB ABZQVO NZWU ITIV
Encrypted String: BPM KISM QA I TQM
```

2.2.3 Decrypted Strings

```
Decrypted String: HAL
Decrypted String: THIS IS A TEST STRING FROM ALAN
Decrypted String: THE CAKE IS A LIE
```

2.2.4 Solve Character Shifts

```
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
```

3 Haskell

3.1 The Code

```haskell
--  Michael Durso                                                --
--  Professor Labouseur                                          --
--  CMPT331 - Theory of Programming Languages                    --
--  Caesar Cipher: The Haskell Hoax                              --
--_____________________________________________________________--


import Data.Char (ord, chr)
-- Convert to ASCII then shift and convert back
shifty :: Char -> Int -> Char -> Char
shifty base offset char = chr $ ord base + (ord char - ord base + offset) `mod` 26

-- Encrypt
encrypt :: Int -> String -> String
encrypt shiftAmount = map str
        where str char
                | char >= 'a' && char <= 'z' = shifty 'a' shiftAmount char
                | char >= 'A' && char <= 'Z' = shifty 'A' shiftAmount char
                | otherwise = char

-- Decrypt - Encrypt but with a negative shift.
decrypt :: Int -> String -> String
decrypt shiftAmount = encrypt (-shiftAmount)

-- Solve - recursion
solve :: Int -> String -> IO ()
solve maxShift str
        | maxShift > 0 = do
                putStrLn ("Caesar " ++ show maxShift ++ ": " ++ encrypt maxShift str)
                solve (maxShift-1) str
        | otherwise = putStrLn ("Caesar " ++ show maxShift ++ ": " ++ str)

-- Main - Do It
main :: IO ()
main = do
        putStrLn ("Original Message: Hal")
        putStrLn ("Encrypted String: " ++ encrypt 8 "Hal")
        putStrLn ("Decrypted String: " ++ decrypt 8 (encrypt 8 "Hal"))
        solve 26 "HAL"
```

3.2 The Output

3.2.1 Original Messages

```
Original Message: HAL
Original Message: This is a test string from Alan
Original Message: The cake is a lie
```

3.2.2 Encrypted Strings

```
Encrypted String: PIT
Encrypted String: Bpqa qa i bmab abzqvo nzwu Itiv
Encrypted String: Bpm kism qa i tqm
```

3.2.3 Decrypted Strings

```
Decrypted String: HAL
Decrypted String: This is a test string from Alan
Decrypted String: The cake is a lie
```

3.2.4 Solve Character Shifts

```
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
```

4 Scala (The Functional One)

4.1 The Code

```
// Michael Durso                                              \\
// Professor Labouseur                                        \\
// CMPT331 - Theory of Programming Languages                  \\
// Caesar Cipher: The Scala Sequel                            \\
//____________________________________________________________\\

// convert to ASCII
object caesar {
    private def shifty(base: Char, char: Char, offset: Int) =
        (base.toInt + ((char.toInt - base.toInt + offset) %26)).toChar

    // map values and shift
    def encrypt(str: String, shiftAmount: Int) = str.map {
        case x if x isLower => shifty('a', x, shiftAmount)
        case x => x
    }

    // Use Encrypt with -shiftAmount
    def decrypt(str: String, shiftAmount: Int) = encrypt(str, -shiftAmount)

    // Loop Encrypt
    def solve(str: String, maxShift:Int): Unit = {
        if(maxShift > -1){
            println("Caesar " + maxShift + ": " + encrypt(str, maxShift))
            solve(str, (maxShift-1))
        }
    }
}

// Get the job done
object Main extends App {
    val str = "hal"
    val shiftAmount = 8
    val maxShift = 26
    println("Original Message: " + str)
    val encrypted = caesar.encrypt(str, shiftAmount)
    println("Encrypted String: " + encrypted)
    val decrypted = caesar.decrypt(encrypted, shiftAmount)
    println("Decrypted String: " + decrypted)
    caesar.solve("hal", maxShift)
}
```

4.2 The Output

4.2.1 Original Messages

```
Original Message: hal
Original Message: this is a test string from alan
Original Message: the cake is a lie
```

4.2.2 Encrypted Strings

```
Encrypted String: pit
Encrypted String: bpqa qa i bmab abzqvo nzwu itiv
Encrypted String: bpm kism qa i tqm
```

4.2.3 Decrypted Strings

```
Decrypted String: hal
Decrypted String: this is a test string from alan
Decrypted String: the cake is a lie
```

4.2.4 Solve Character Shifts

```
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
```

5 Erlang

5.1 The Code

```
% Michael Durso                                                  %
% Professor Labouseur                                            %
% CMPT331 - Theory of Programming Languages                      %
% Caesar Cipher: The Erlang Epilogue                             %
%________________________________________________________________%

% Codingground calls the file "helloworld.erl"
-module(helloworld).
-export([encrypt/2, decrypt/2, solve/2, main/0]).

% Shift characters
shifty(Char, ShiftAmount) ->
        ((((hd(Char)) - 65 + ShiftAmount) rem (26) + 26) rem 26) + 65.

%  Encrypt
encrypt(Str, ShiftAmount) ->
        lists:map(fun(Char) -> shifty([(Char)], ShiftAmount) end, Str).

% Decrypt
decrypt(Str, ShiftAmount) ->
        encrypt(Str, ShiftAmount * -1).

% Solve
solve(Str, MaxShift) ->
        NumList = lists:seq(1, MaxShift),
        lists:foreach(fun(ShiftAmount) ->
                io:format("Caesar ~p: ", [ShiftAmount]),
                io:format("~s~n ",
                [decrypt(Str, ShiftAmount)]) end, NumList).

% Fin.
main() ->
    io:format("Original Message: ~s~n", [Str]),
        io:format("Encrypted String: ~s~n", [encrypt("hal", 8)]),
        io:format("Decrypted String: ~s~n", [decrypt("pit", 8)]),
        solve("hal", 26).
```

5.2 The Output

```
Original Message: hal
Original Message: this is a test string from alan
Original Message: the cake is a lie
```

5.2.1 Encrypted Strings

```
Encrypted String: pit
Encrypted String: bpqa qa i bmab abzqvo nzwu itiv
Encrypted String: bpm kism qa i tqm
```

5.2.2 Decrypted Strings

```
Decrypted String: hal
Decrypted String: this is a test string from alan
Decrypted String: the cake is a lie
```

5.2.3 Solve Character Shifts

```
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
```

6 The Log, A Memoir, A True Fictional Epic, A Journey of Fates, The Cathartic Conclusion

6.1 Arrival

So of course I had the lovely call to action to start this assingment just after Fun With Lambda Calculus was due. Boy, that was ROUGH for me, I do not know what happened there. Anyway, As usual I begin to think I have plenty of time I don't need to start yet. Well Beer O'clock came and went at least seven times before I got around to this assignment. Sometimes priorities are just priorities. Also when it wasn't Beer O'clock I had to work on my website for my Internship and the site finally went public. good news is it is getting a lot of use and a lot of good reviews.

6.2 The Storm

Now, It is now much later after this project has been assigned and again no real programming has been attempted yet. This was Monday November 10th. I gave myself a good four days to do this project so that is a plus. I started Monday with LISP and ended it with ML so it was pretty productive. I enjoyed them both and I felt like I had a decent understanding going into them since we went over them in class with a lot of examples.

6.3 floodgate

So Haskell. Once you learn to love it you just still really don't love it. The code would not run because one of the lines registered a tab instead of spaces and it would not tell me that was the reason. It is so hard to find an error that is actually invisible. I was actually expecting a lot worse from many of the cries that echoed in the class but it was not as horrible as I expected. Just very unpleasant is all.

Scala was definitely a step up from last time. there were more references that help with this program regarding functional programming rather than procedural. Also now that I had already dealt with learning the syntax once, it was easier to come back to it. I worked on Scala right after our class on Wednesday until around 2 when I finished. I gave myself a break then I tried to tackle Erlang.

6.4 The Covenant

So I started Erlang Wednesday and I basically had the whole program written except nothing would run. I tried some other test programs in the compiler I was using and they all worked fine and at one point I even rewrote the whole thing from scratch because I was at a loss. After I rewrote it and it still wasn't working I gave up for Wednesday. Then Thursday I just spent around 2 hours trying to figure out what was wrong, I tried different compilers and spent a lot of time on Stack Overflow but nothing really jumped out at me. I rewrote the whole thing again except this time I had two windows side by side and I just typed it out exactly. Then for some reason it worked. So. I mean I'm not going to be the guy that says the compiler is broken but what else am I gonna do? Admit I messed up somewhere even though I don't know where and still am unsure about what was wrong? absolutely not! I of course did everything correctly and "supposedly" the compiler did everything right too so we can just call it even.

6.5 Cortana

This is another one of those things that you enjoy while I get to reflect on the things that made my head go through the wall and caused my will to exist to decompose. Here you can find some interesting facts about how long I thought it would take me to find salvation compared to how long it took me to reach my empty hollow shell of a programmer. But isn't that what we all do it for? That brief moment of "Hey this Works!"

that you can enjoy until you have to move on to the next thing that doesn't work. It probably sounds like I hate this... but I live for those few seconds of achievement I get after figuring out a program or debugging errors.

I am going to mention the Google searches here since they basically all overlap again. First I was looking for online compilers for each language and I was able to use Codingground for almost all of them again. I also looked up example programs, cheat sheets, style and syntax guides among some other general concepts that would go with any language. Compared to the last assignment, all of these languages felt much more similar than those in the last assignment. If you put all 5 sections of the code next to each other they follow a very similar layout. Each encrypt function is usually 2-3 lines and the decrypt function is just calling encrypt with a negative shift. then the solve function is typically a few more lines so that it can loop/ recurse-ion-ing(some form of the word recursion goes there) and print the output. Then at the bottom is the main which is usually 4-8 lines.

6.5.1 LISP

Readability- I think this is one of the more readable languages in this assignment being that most of the functions appear simple. The shift function is the only one that I would disagree with this on because there is ton of parentheses and the cond statement is not very readable.

Writability- A little hard keeping track of the parentheses but all of the other functions after shift were very easy to write.

Language loves and hates- It was easy to get used to it since we went over examples in class and I felt like I had a good basis on how the simple things worked. Also I got to use lambda!

Hours expected- 3

Hours spent- 3

6.5.2 ML

Readability- Not the most readable language but part of that comes from the fact that it might use a lot of methods in one line that will make it look a lot longer and more complicated than it really is.

Writability- Pretty good to write in because there are a lot of helpful features like imploding and exploding. That gets a lot of the dirty work done for you especially if you sprinkle a little bit of toUpper in there.

Language love and hates- I loved using ML because its from New Jersey and so am I. Crazy. Also since we went over it in class and you provided a lot of test examples it made it much easier to complete.

Hours expected- 3

Hours spent- 2.5

6.5.3 Haskell

Readability- Because of the forced indentation it makes the code more readable and organized.

Writability- because indentation has to be exact it makes it harder to write. It forces good practice but can cause unnecessary issues in the code.

Language love and hates- The indentation thing kinda threw me off. I originally wrote it on one compiler and when I copied it into another the lining up did not look right. So when I tried fixing the indentation it would cause errors. Mostly was an issue because I switched the compiler I was using and it appeared different after a copy and paste.

Hours expected- 5

Hours spent- 6

6.5.4 Scala

Readability- Not the most readable but it is definitely less intimidating to look at than the procedural Scala.

Writability- I enjoyed writing in this. It felt fairly straight forward and I was used to some of my issues from the last assignment.

Language love and hates- Same as the last project it was just an ok time for me. A little better experience this time than last.

Hours expected- 4

Hours spent- 4

6.5.5 Erlang

Readability- Really not horrible to read. My parentheses got a little out of hand in my character shift section but the rest is fine. Each function is pretty much 2 lines except solve but that is just because it handled the output formatting.

Writability- Writing it was just smack in the middle OK for me. I would constantly miss having a variable start with an uppercase letter since I am just not used to writing like that. I also could not get my code to run forever but eventually it worked and I am still not sure what fixed it but I am not complaining.

Language love and hates- I really just did not like the capitalization requirement. I feel like it lost some readability there as well. I do like the use of "->" though...It's fun.

hours expected- 5

Hours spent- 7

6.6 Halo

The LaTeX. It's ability to create perfect documents for people with the most extreme OCD got to me again but luckily I was able to use the same template as last time and it was much quicker. Since the code was smaller it always fit on one page which was much nicer for organization. I love the way these documents come out because they look so professional and then you look at the contents and see references to Halo and Beer O'clock. Alas, if you look at the time now of me completing this assignment you will see that it is none other than the ever coveted: Whisky Hour. It calls and I must answer.