Lab One

Michael Durso
michael.durso1@marist.edu

October 4, 2019

Contents

1 Fortran

1.1 The Code

```
! Michael Durso                                                    !
! Professor Labouseur                                             !
! CMPT331 - Theory of Programming Languages                       !
! Caesar Cipher: Fortran Fairy Tale                               !
!-----------------------------------------------------------------!

program Caesar_Cipher

! Set the shiftAmount to the amount of letters you would like to shift the text
  integer, parameter :: shiftAmount = 8

! Test strings for the cipher
    CHARACTER(40) :: testStr1="hal"
    CHARACTER(40) :: testStr2="This is a test string from Alan"
    CHARACTER(40) :: testStr3="The cake is a lie"


!!!!!  This is where stuff goes down  !!!!!

! Print the original message
    write(*,"(2a)") "Original Message: ", testStr1
    write(*,"(2a)") "Original Message: ", testStr2
    write(*,"(2a)") "Original Message: ", testStr3
    print*, ""

! Convert to uppercase
    CALL upperCase(testStr1)
    CALL upperCase(testStr2)
    CALL upperCase(testStr3)

! Print the encrypted string
    call encrypt(testStr1)
    call encrypt(testStr2)
    call encrypt(testStr3)
    write(*, "(2a)") "Encrypted String: ", testStr1
    write(*, "(2a)") "Encrypted String: ", testStr2
    write(*, "(2a)") "Encrypted String: ", testStr3
    print*, ""

! Print the decrypted string
    call decrypt(testStr1)
    call decrypt(testStr2)
    call decrypt(testStr3)
    write(*, "(2a)") "Decrypted String: ", testStr1
    write(*, "(2a)") "Decrypted String: ", testStr2
    write(*, "(2a)") "Decrypted String: ", testStr3
    print*, ""

! Show all shift amounts to for the enctrypted string
    call solve(testStr1,26)


contains


! Subroutine to convert to uppercase

subroutine upperCase(letter)
   ! Variable fun-time
       CHARACTER(*) letter
```

```
       CHARACTER(1) c
       INTEGER charInt

   ! Loop through all letters
       do i=1, len(letter), 1

        ! Get the current letter
          c = letter(i:i+1)
          charInt = iachar(c)

                ! Convert character to uppercase
            IF (charInt > 90) THEN
              charInt = charInt - 32
            END IF
          c = achar(charInt)

                ! Put the character back
            letter(i:i) = c
       end do
end subroutine

!---------------------------------------------------------------!
! Below are the Encrypt and Decrypt subroutines                 !
! They are exactly the same except for - or + the shiftAmount  !
! The 65 is for ASCII character 'A'                             !
!---------------------------------------------------------------!

! Subroutine to ENCRYPT the string

subroutine encrypt(string)
   ! Variable fun-time
   CHARACTER(*), intent(inout) :: string
   INTEGER :: i

   ! Loop to encrypt the string
   do i = 1, len(string)
      select case(string(i:i))
         case('A':'Z')
            string(i:i) = achar(modulo(iachar(string(i:i)) - 65 + shiftAmount, 26) + 65)
      end select
   end do
end subroutine


! Subroutine to DECRYPT the string

subroutine decrypt(string)
   ! Variable fun-time
   CHARACTER(*), intent(inout) :: string
   INTEGER :: i

   ! Loop to decrypt the string
   do i = 1, len(string)
      select case(string(i:i))
         case('A':'Z')
            string(i:i) = achar(modulo(iachar(string(i:i)) - 65 - shiftAmount, 26) + 65)
      end select
   end do
end subroutine


! Subroutine to SOLVE by printing all shifts of the string

subroutine solve(str, maxShift)
   ! Variable fun-time
```

```
126     CHARACTER(*) str
127     CHARACTER(len=len(str)) strHolder
128     CHARACTER(1) c
129     INTEGER charInt
130     INTEGER maxShift
131     strHolder = str
132 
133     ! Outer loop
134     do n=maxShift, 0, -1
135         ! Inner loop to go through letters
136      do i=1, len(str), 1
137           c = str(i:i+1)
138           charInt = iachar(c)
139         IF (charInt == 32) THEN
140           newInt = charInt
141         ELSE
142           newInt = charInt + n
143         IF (newInt > 90) THEN
144           newInt = 64 + (newInt - 90)
145         END IF
146       END IF
147       c = achar(newInt)
148       ! Put characters back
149       strHolder(i:i) = c
150      end do
151     print *," Caesar", n, ": ", strHolder
152    end do
153 end subroutine
154 
155 end program Caesar_Cipher
```

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 COBOL

2.1 The Code

```
*> Michael Durso                                                    <*
*> Professor Labouseur                                             <*
*> CMPT331 - Theory of Programming Languages                       <*
*> Caesar Cipher: COBOL Chronicle                                  <*
*>_________________________________________________________________<*
IDENTIFICATION DIVISION.
       PROGRAM-ID. Caesar_Cipher.
       *> Variable Fun-Time
       DATA DIVISION.
                WORKING-STORAGE SECTION.
       1 str PIC x(50)
           VALUE "hal".
       1 offset binary PIC 9(4) VALUE 8.
       1 from-chars PIC x(26).
       1 to-chars PIC x(26).
       *> Values are needed twice for upper and lowercase conversion
       1 tabl.
        2 PIC x(26) VALUE "ABCDEFGHIJKLMNOPQRSTUVWXYZ".
        2 PIC x(26) VALUE "ABCDEFGHIJKLMNOPQRSTUVWXYZ".
       1 i PIC 9(2) VALUE 1.

       *> Make the stuff do the things
       PROCEDURE DIVISION.
       biggerLetterMaker.
           DISPLAY "Original Message: " str
           Move Function Upper-case(str) to str
           .
       doTheThings.
           PERFORM encrypt
           DISPLAY "Encrypted String: " str
           PERFORM decrypt
           DISPLAY "Decrypted String: " str
           PERFORM solve VARYING i FROM 0 BY 1 UNTIL i = 27.
           STOP RUN
           .
       *> Encrypt, Decrypt, Solve
       encrypt.
           MOVE tabl (1:26) TO from-chars
           MOVE tabl (1 + offset:26) TO to-chars
           INSPECT str CONVERTING from-chars
               TO to-chars
           .
       decrypt.
           MOVE tabl (1 + offset:26) TO from-chars
           MOVE tabl (1:26) TO to-chars
           INSPECT str CONVERTING from-chars
               TO to-chars
           .
        solve.
            MOVE tabl(2:26) TO from-chars.
            MOVE tabl(1:26) TO to-chars.
            INSPECT str CONVERTING from-chars
              TO to-chars.
                  DISPLAY "Caeser " i ": " str
           .
       *> Things done successfully via the stuff
       END PROGRAM Caesar_Cipher.
```

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

Caeser 00: GZK
Caeser 01: FYJ
Caeser 02: EXI
Caeser 03: DWH
Caeser 04: CVG
Caeser 05: BUF
Caeser 06: ATE
Caeser 07: ZSD
Caeser 08: YRC
Caeser 09: XQB
Caeser 10: WPA
Caeser 11: VOZ
Caeser 12: UNY
Caeser 13: TMX
Caeser 14: SLW
Caeser 15: RKV
Caeser 16: QJU
Caeser 17: PIT
Caeser 18: OHS
Caeser 19: NGR
Caeser 20: MFQ
Caeser 21: LEP
Caeser 22: KDO
Caeser 23: JCN
Caeser 24: IBM
Caeser 25: HAL
Caeser 26: GZK

3 BASIC

3.1 The Code

```
' Michael Durso                                                    '
' Professor Labouseur                                             '
' CMPT331 - Theory of Programming Languages                       '
' Caesar Cipher: BASIC Book (Get it... it gets a BASIC name)      '
'-----------------------------------------------------------------'


' ___________Some Notes___________
' strang is a string but cooler
' Encrypt and Decrypt are the same except + or - the 26
' UCase to capitalize letters

' Encrypt
Function encrypt(strang As String, shiftAmount As Integer) As String
    dim i As Integer
    Const minChar As Integer = 65, maxChar As Integer = 90
    strang = UCase(strang)

    for i = 0 to len(strang)
        if strang[i] >= minChar And strang[i] <= maxChar then
            strang[i] = strang[i] + shiftAmount
            if strang[i] > maxChar then
                strang[i] = strang[i] - 26
            end if
        end if
    next
    Return strang
End Function

' Decrypt
Function decrypt(strang As String, shiftAmount As Integer) As String
    dim i As Integer
    Const minChar As Integer = 65, maxChar As Integer = 90
    strang = UCase(strang)

    for i = 0 to len(strang)
        if strang[i] >= minChar And strang[i] <= maxChar then
            strang[i] = strang[i] - shiftAmount
            if strang[i] < minChar then
                strang[i] = strang[i] + 26
            end if
        end if
    next
    Return strang
End Function

' Solve
Sub solve(strang As String, currentShiftAmt As Integer)
    dim i As Integer
    Const minChar As Integer = 65, maxChar As Integer = 90
    strang = UCase(strang)

    Do
        Print "Caesar " + str$(currentShiftAmt) + ": " + strang
        for i = 0 to len(strang)
            if strang[i] >= minChar And strang[i] <= maxChar then
                strang[i] = strang[i] - 1
                if strang[i] < minChar then
                    strang[i] = strang[i] + 26
                end if
```

```
            end if
        next
        currentShiftAmt = currentShiftAmt - 1
    Loop While currentShiftAmt > -1
End Sub

dim strang As String = "Hal"
Print "Original Message: "; strang
Print "Encrypted String: "; encrypt(strang, 8)
Print "Decrypted String: "; decrypt(strang, 8)
solve(strang, 26)
Sleep
```

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 PASCAL

4.1 The Code

```
{ Michael Durso                                                 }
{ Professor Labouseur                                           }
{ CMPT331 - Theory of Programming Languages                     }
{ Caesar Cipher: Pascal Parable                                 }
{_______________________________________________________________}

Program CaesarCipher(output);
{ All the procedures are below, they all work very similarly    }
{ Encrypt adds the "shiftAmount"                                }
{ Decrypt subtracts the "shiftAmount"                           }
{ Solve adds 1 so that it it displays each shift amount         }
{_______________________________________________________________}


{ Encrypt }
procedure encrypt(var str: string; shiftAmount: integer);
  var
    i: integer;
  begin
    for i := 1 to length(str) do
     case str[i] of
      'A'..'Z': str[i] := chr(ord('A') + (ord(str[i]) - ord('A') + shiftAmount) mod 26);
      'a'..'z': str[i] := chr(ord('a') + (ord(str[i]) - ord('a') + shiftAmount) mod 26);
     end;
  end;

{ Decrypt }
procedure decrypt(var str: string; shiftAmount: integer);
  var
    i: integer;
  begin
    for i := 1 to length(str) do
     case str[i] of
      'A'..'Z': str[i] := chr(ord('A') + (ord(str[i]) - ord('A') - shiftAmount + 26) mod 26);
      'a'..'z': str[i] := chr(ord('a') + (ord(str[i]) - ord('a') - shiftAmount + 26) mod 26);
     end;
  end;


{ Solve }
procedure solve(var str : string; maxShift : integer);
  var
    j : integer;
    i : integer;
  begin
    j := 0;
    while j <= maxShift do
      begin
       for i := 1 to length(str) do
        case str[i] of
         'A'..'Z': str[i] := chr(ord('A') + (ord(str[i]) - ord('A') + 1) mod 26);
         'a'..'z': str[i] := chr(ord('a') + (ord(str[i]) - ord('a') + 1) mod 26);
        end;
        { prints the cases each iteration }
        writeln('Caesar ', j, ': ', str);
        j := j + 1;
    end;
end;


```

```
{ Everyones favorite time, Variable Fun-Time}
var
  shiftAmount: integer;
  str: string;
  maxShift: integer;

{ The things will happen here }
begin
  shiftAmount := 8;
  maxShift := 26;
  str := 'Hal';
  writeln ('Original Message: ', str);
  encrypt(str, shiftAmount);
  writeln ('Encrypted String: ', str);
  decrypt(str, shiftAmount);
  writeln ('Decrypted String: ', str);
  solve(str, maxShift);
end.
```

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 0: Ibm
Caesar 1: Jcn
Caesar 2: Kdo
Caesar 3: Lep
Caesar 4: Mfq
Caesar 5: Ngr
Caesar 6: Ohs
Caesar 7: Pit
Caesar 8: Qju
Caesar 9: Rkv
Caesar 10: Slw
Caesar 11: Tmx
Caesar 12: Uny
Caesar 13: Voz
Caesar 14: Wpa
Caesar 15: Xqb
Caesar 16: Yrc
Caesar 17: Zsd
Caesar 18: Ate
Caesar 19: Buf
Caesar 20: Cvg
Caesar 21: Dwh
Caesar 22: Exi
Caesar 23: Fyj
Caesar 24: Gzk
Caesar 25: Hal
Caesar 26: Ibm

5 Scala (The Procedural One)

5.1 The Code

```
1
2  // Michael Durso                                                \\
3  // Professor Labouseur                                          \\
4  // CMPT331 - Theory of Programming Languages                    \\
5  // Caesar Cipher: Scala Saga                                    \\
6  //____________________________________________________________\\
7
8  object Main extends App{
9  // Variable Fun-Time
10     var strang = "hal"
11     var minChar = 65
12     var maxChar = 90
13
14 // Set shiftAmount before calling each function
15 // Call Encrypt, Decrypt, and Solve functions
16   var shiftAmount = 1
17   encrypt(strang, shiftAmount)
18
19   shiftAmount = 1
20   decrypt(strang, shiftAmount)
21
22   shiftAmount = 26
23   solve(strang, shiftAmount)
24
25 // Function definitions for Encrypt, Decrypt, and Solve
26 // toUpperCase for everything before the loop shift stuff
27
28   def encrypt (strang:String, shiftAmount:Int) : Unit = {
29     // Small variable fun time
30     var ltrs = strang.toUpperCase.toCharArray
31     print("Original Message: ")
32     println(ltrs.mkString)
33     var i = 0
34     // Loop and shift
35     while (i < ltrs.length()) {
36       var j = ltrs(i).toInt
37       if (j >= minChar && j <= maxChar) {
38         j = if (j + shiftAmount > maxChar) j + shiftAmount - 26 else j + shiftAmount
39       }
40       ltrs(i) = j.toChar
41       i = i + 1
42     }
43     print("Encrypted String: ")
44     println(ltrs.mkString)
45   }
46
47   def decrypt (strang:String, shiftAmount:Int) : Unit = {
48     // Small variable fun time
49     var ltrs = strang.toUpperCase.toCharArray
50     var i = 0
51     // Loop and shift
52     while (i < ltrs.length()) {
53       var j = ltrs(i).toInt
54       if (j >= minChar && j <= maxChar) {
55         j = if (j - shiftAmount < minChar) j - shiftAmount + 26 else j - shiftAmount
56       }
57       ltrs(i) = j.toChar
58       i = i + 1
59     }
60     print("Decrypted String: ")
```

```
    println(ltrs.mkString)
  }

  def solve (strang:String, maxShiftValue:Int) : Unit = {
    // Small variable fun time
    var shiftValue = maxShiftValue
    val ltrs = strang.toUpperCase.toCharArray
    var i = 0
    // Loop and shift
    while (shiftValue > -1) {
      println("Caesar " + shiftValue + ": " + ltrs.mkString)
      while (i < ltrs.length()) {
        var j = ltrs(i).toInt
        if (j >= minChar && j <= maxChar) {
          j = j - 1
          if (j < minChar) {
            j = j + 26
          }
        }
        ltrs(i) = j.toChar
        i = i + 1
      }
      i = 0
      shiftValue = shiftValue - 1
    }
  }
}
```

5.2 The Output

5.2.1 Original Messages

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

5.2.2 Encrypted Strings

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

5.2.3 Decrypted Strings

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

5.2.4 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

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

6.1 The Courtesy Call

It all started one cold rainy day at Marist in a Theory of Programming Languages Course. I actually don't remember what day this was officially assigned nor do I remember the weather but that's fine. Every couple days here and there I would search random things on my phone about the languages and read something quickly but it was nothing too intensive until I actually started the programming. For the five languages I was dealing with I would search for things like "Insert Language Here cheat sheet" or "Insert Language Here program structure and "Insert Language Here example programs. These sort of general searches gave me some insight into what I was going to be dealing with in terms of formatting things and how complicated it would be to perform the task in each different language.

6.2 The Cold Boot

It is now much later after this project has been assigned and no real programming has been attempted yet. This day sort of **kicked** (kicked like as in "The Cold Boot") me hard and made me realize I should probably start doing work. We will call this day of realization: September 25th.

So I break out the project sheet to make sure I know every thing that I need to include in what I do. To start I will predict roughly 6 hours it will take me to complete the Fortran Caesar Cipher. I chose 6 because many students in class would say there estimations that would be higher or lower than that and 6 sounded like a good guess. Right now I am at the first language still and I plan to use comments well and effectively as progress. It will be interesting to see how after each language I will probably start getting lazier with this.

At this point I have grown tired of staring at my screen whilst being confused so I will come back to this later. I have successfully made the Encrypt and Decrypt subroutines work in roughly 4 hours.

6.3 The Return

Back to the slow pain that Fortran is causing me. The Solve function was so close to being right for a while and I just couldn't get it and I didn't know why. Eventually it clicked and just kind of made sense to me. My process is a bit long to do the Solve function and I am convinced that there is a shorter and more efficient way to do it but as of now i am happy that it works. Now that I finished Fortran I feel like I know the kind of stuff that I need to search for to do the next languages... or at least I hope.

6.4 The Surprise

It is now September 26th and I finished my COBOL Caesar Cipher. When I first started looking at sample programs and just general rules I partially liked it a lot and also partially hated it. I thought the idea of having to have certain parts of code in different columns was annoying but in practice I did not run into many issues. I imagine with larger programs it would have been more annoying. One thing that grew on me was the different divisions in the code. I especially liked how the procedure division seemed like it could stay organized all in the one location. I predicted that this was going to take me around 4 hours before I started it and I almost finished it in about 3 hours. Then right when I thought I was done and getting ready to put my code in this LaTeX document, I did some extra testing and some outcomes were slightly wrong while others were right. It was a quite annoying **surprise** but it only took me 30 minutes to figure out the issue and then fix it to make everything work nicely.

6.5 The Escape

It is September 27th and I just finished the BASIC version of the Caesar Cipher. BASIC is well pretty basic. The format and program structure was easy to learn and follow and the Encrypt and Decrypt functions seem to be getting easier each time since I know the numbers to use and how to use them. Since I am getting better at writing code for a Caesar Cipher in general it is making it easier to write the program for any language. The first time in Fortran compared to now was a big difference because the last time that I had to focus on how the program works aside from just the way to write in that specific language. Now I can basically just focus on the syntax of how to write what exactly I want to do.

6.6 The Fall

September 28th: My laptop broke. Luckily it is just the left hinge and it happened while I was opening it. I say luckily because at least I can still use it I just can't close it. All in all this lead to a fall in my morale so I took a day off so I wouldn't have to look and my mangled machine.

6.7 The Reunion

Did I say I took a day off? I meant the entire weekend because my parents came to visit and all that good stuff. My Dad thought it was funny that I was using Fortran since he used that for a class in college. Anyways this isn't a Family Reunion but actually me reuniting with my laptop that I refused to believe broke. Good news is it broke in the same spot last year and the warranty is still good so that is very cool. Now back to the fact that I still have 2 languages left to program in at this point. Well actually only 1 because I finished Pascal in class because it was an easy one. "Well Mike shouldn't you be paying attention in class?" The answer to that is yes but I finished a lab prior to class so I basically had a free hour to complete Pascal which I did. it took another little bit to clean up the code and make the comments nicer but we won't count that.

So Scala bothered me before I even started because I used the coding ground online compilers for every other language and then for some reason Scala just would not work for me. for a while things wouldn't compile at all. Eventually when it would compile nothing displayed. this took a chunk of time because the website worked great for all of the other languages. So I took my code and went to some other online Scala compiler that I don't remember the name of and it sort of worked there. I had to change some things around because I was basically working in the regular notepad since the coding ground compiler was no help. Scala was also fairly easy to write, it looks a bit more intimidating when reading it but when I wrote it it was a lot of short lines of code that all did something really simple. Since Scala is object oriented and this was the procedural version it worked sort of like Java when I was writing it. Man, remember Java? Life used to be so simple.

6.8 The Itch

So this is a nice little section about the things that would really just get to me during the project (aside from my laptop falling apart) as well as some things I noticed about reading and writing the programs. Also I figured I could summarize some of the stuff about each language that I did here. As for some google searches the things that every language had in common were: cheat sheets, program layout, syntax, is there a function to convert to uppercase.

6.8.1 Fortran

Readability- I think that for the most part it is a fairly readable language. Nothing strikes me as a "what am I looking at" kind of thing which is always good.

Writability- strings are stupid and they don't actually work like strings it is basically just an array of integers that likes to annoy me. I also didn't like all the annoyances when it came to the variable declarations. I get that it forces you to write with better practice but I would like the option to be lazy in certain places.

Language loves and hates- I was kind of indifferent to Fortran. I would say I was definitely more annoyed with it than not annoyed with it. Mostly just the variable stuff that bothered me.

Similarities and differences- The Solve function seemed to work kind of the same way as it did in Scala. That or it could just be in my head. When I go back and read it it seems to be the most similar feature.

Hours expected- 6

Hours spent- 7

6.8.2 COBOL

Readability- I thought COBOL was a very readable language. Everything seemed very simple and made a lot of sense as well. The only part that was a little strange was the Identification Division.

Writability- I also enjoyed writing in COBOL as it also was very simple to do. There were a couple of spots where I was messed up but I think overall it was one of the easiest to do.

Language love and hates- I did like how easy it was for me to both read and write in COBOL but I disliked the rules for the different columns in the program for what can go where.

Similarities and differences- Definitely the Most straightforward reading for me which is what made it different from the others. It was more like reading in English than a programming language.

Hours expected- 4

Hours spent- 3.5

6.8.3 BASIC

Readability- Fairly simple when it comes to reading it, reads similarly to some of the other languages.

Writability- Straight forward and very easy to write in.

Language love and hates- I enjoyed how easy it was to write stuff in BASIC because it did not feel as if it were a foreign language.

Similarities and differences- It used a for loop in the same way as other languages used loops like for and do.

Hours expected- 3

Hours spent- 3

6.8.4 Pascal

Readability- I felt that this was pretty readable for a programming language. it is not as readable as COBOL but it does structure very nicely.

Writability- I also enjoyed writing in Pascal a lot as I said I felt the format was nice. It didn't feel complicated while I wrote it and I was able to keep the program organized into nice sections.

Language love and hates- I loved how easy it was for me to keep the program organized because I tend to get disorganized while I program and this made it fairly simple.

Similarities and differences- Same as some of the other languages where it is broken up into three main parts: the variables, the functions, and the make-stuff-happen section.

Hours expected- 2

Hours spent- 1.5

6.8.5 Scala

Readability- It seems a bit more intimidating to read than it is to write it just because of the size of the functions. They are actually fairly simple when you break them down line by line but all together it looks like there is a lot going on.

Writability- I thought it was easier to write than to read. This is because as I said before that each line of the function was fairly simple and it was not hard to make sense of it as i was writing it out to make everything work.

Language love and hates- Not really Scala's fault but I hated my experience with the online compiler. It just gave me issues that took an unnecessary amount of time to solve. As for the language itself it did not really sway me either way.

Similarities and differences- Worked kind of like Fortran when it came to the functions.

hours expected- 4

Hours spent- 4.5

6.9 The Part Where he Kills You

The LaTeX. Well I never knew that I could actually enjoy word documents this much but it is actually really cool to use LaTeX. The issue is there are so many things I want to do now to make things look perfect and it takes up even more time to try and figure out how to do them. I felt like there were a lot of areas where I could have formatted this document better but I wasn't able to get it exactly how I wanted it. I am going to be spending way to much time in LaTeX.