Programming in the Past

Aidan Carr

Aidan.Carr1@marist.edu

February 21, 2025

1 Introduction

1.1 Directions

Develop a set of functions that will allow you to **encrypt** a string using a Caesar cipher. Develop a set of functions that will allow you to **decrypt** a string using a Caesar cipher. Develop a set of functions that will help you to **solve** (break) a Caesar cipher.

Implement the above functions for five programming languages.

1.2 Languages and Time Predictions

Before the assignment, I predicted how long it would take to write the Caesar Cipher programs in each language. *Tadaa*:

1. *Java, predicted 0:30*
2. Fortran, predicted 2:00
3. BASIC, predicted 1:30
4. Pascal, predicted 1:00
5. COBOL, predicted 1:30
6. Scala, predicted 2:00

Note: This is the order I wrote my programs in. Also I wrote the Java code in 1:00

2 Fortran

2.1 Fortran Code

Check out this awesome code! On the next page!!!

```
program CAESARCIPHER

  ! Variable Declaration
  IMPLICIT NONE
  INTEGER :: caesarValue
  CHARACTER (len=14) :: plainText
  CHARACTER (len=14) :: a
  CHARACTER (len=14) :: b
  INTEGER :: index

  ! Functions
  CHARACTER(len=14) :: ENCRYPT
  CHARACTER(len=14) :: DECRYPT

  ! Assign values
  caesarValue = 30
  plainText = "FORTRAN IS FUN"
  PRINT *, "Plain text: ", plainText, " + ", caesarValue

  ! Encrypt the text
  a = ENCRYPT(plainText, caesarValue)
  PRINT *, "Encrypted text: ", a

  ! Decrypt the text
  b = DECRYPT(a, caesarValue)
  PRINT *, "Decrypted text: ", b

  ! Solve the cipher
  PRINT *, "Solve: V*SA JXYI IX!J"
  call SOLVE("V*SA JXYI IX!J", 60)

end program CAESARCIPHER


! Encryption function
function ENCRYPT(message, value) result(encryptedText)
  ! Variable Declaration
  IMPLICIT NONE
  INTEGER :: index
  INTEGER :: value
  CHARACTER(len=14) :: message
  CHARACTER(len=14) :: encryptedText
  CHARACTER :: currentChar
  INTEGER :: shiftedASCII

  ! Modify value
  value = MOD(value, 26)
  ! Start with original message
  encryptedText = message

  ! Encrypt each character
  DO index = 1, LEN_TRIM(message)
    currentChar = message(index:index)

    ! Letter
    IF (currentChar >= 'A' .AND. currentChar <= 'Z') THEN
      shiftedASCII = iachar(currentChar) + value

      ! Wrap around if past 'Z'
      IF (shiftedASCII > iachar('Z')) THEN
        shiftedASCII = shiftedASCII - 26
      END IF

      ! Encrypted a character
      encryptedText(index:index) = achar(shiftedASCII)
    END IF
  END DO
end function ENCRYPT
```

```
71 ! Decryption function
72 function DECRYPT(message, value) result(decryptedText)
73   ! Variable Declaration
74   IMPLICIT NONE
75   INTEGER :: value
76   CHARACTER(len=14) :: message
77   CHARACTER(len=14) :: decryptedText
78   CHARACTER(len=14) :: ENCRYPT
79
80   ! Modify value
81   value = MOD(value, 26)
82   ! Encrypt it but backwards
83   decryptedText = ENCRYPT(message, 26-value)
84
85 end function DECRYPT
86
87
88 ! Solve subroutine
89 subroutine SOLVE(message, maxShiftValue)
90   ! Variable Declaration
91   IMPLICIT NONE
92   INTEGER :: maxShiftValue
93   CHARACTER(len=14) :: message
94   INTEGER :: index
95   CHARACTER(len=14) :: ENCRYPT
96
97   ! Loop desired encryption
98   DO index = 0, maxShiftValue
99       PRINT *, "Caesar ", maxShiftValue-index, ": ", ENCRYPT(message, maxShiftValue-index)
100  END DO
101
102 end subroutine SOLVE
```

2.2 Fortran Test Output

```
 Plain text: FORTRAN IS FUN +          30
 Encrypted text: JSVXVER MW JYR
 Decrypted text: FORTRAN IS FUN
 Solve: V*SA JXYI IX!J
 Caesar            60 : D*AI RFGQ QF!R
 Caesar            59 : C*ZH QEFP PE!Q
 Caesar            58 : B*YG PDEO OD!P
 Caesar            57 : A*XF OCDN NC!O
 Caesar            56 : Z*WE NBCM MB!N
 Caesar            55 : Y*VD MABL LA!M
 Caesar            54 : X*UC LZAK KZ!L
 Caesar            53 : W*TB KYZJ JY!K
 Caesar            52 : V*SA JXYI IX!J
 Caesar            51 : U*RZ IWXH HW!I
 Caesar            50 : T*QY HVWG GV!H
 Caesar            49 : S*PX GUVF FU!G
 Caesar            48 : R*OW FTUE ET!F
 Caesar            47 : Q*NV ESTD DS!E
 Caesar            46 : P*MU DRSC CR!D
 Caesar            45 : O*LT CQRB BQ!C
```

...

2.3 Fortran Thoughts

I was not a fan of Fortran. This was the first language I programmed in for this project. My method of learning the language began on Google. Questions I Googled included:

- how to declare variables fortran
- implict none fortran
- string in fortran
- fortran print statement
- function with return type fortran
- for loop fortran
- how to fix non conforming tab error fortran

I was incredibly frustrated with this language many times because I kept receiving two errors. The first error was `Warning: Nonconforming tab character at (1) [-Wtabs] prog.f95:4:1:`. This actually pissed me off because I didn't know what I was doing wrong. It turns out it was tabbing and spacing throwing it off? Not entirely sure but I got through it.

The nex major issue I faced was `Error: Unexpected data declaration statement at (1)`. This type conversion or mismatch error came up so often. Declaring variables was already a struggle in this language, then I had to learn to declare functions with return types and re-declare variables. It was messy. For a while, I used the REAL data type to avoid these errors, then eventually learned how to fix it using Chat GPT.

Chat GPT became a HUGE help to me during this project... but *after* suffering through Fortran. To me, the readability and writeability of the language are incredibly low. Not only that, but the online community is much smaller than some of the other languages. I don't like the variable declaration aspect, Strings were a bit of a mess at first, variable re-declaration was stupid for inside functions and subroutine, and I NEVER learned what `IMPLICIT NONE` meant, and I'm not gonna look it up. Next language.

3 BASIC

3.1 BASIC Code

Check out this awesome BASIC code! I'm so awesome!!!

```
' ENCRYPT FUNCTION
FUNCTION ENCRYPT (message AS STRING, value AS INTEGER) AS STRING

  ' Modify value
  value = value MOD 26
  ' Start with original message
  DIM encryptedText AS STRING = message

  FOR i AS INTEGER = 1 to LEN(message)
    DIM currentChar AS STRING = MID(message, i, 1)
    DIM shiftedASCII AS INTEGER = 0

    ' LETTER
    IF ASC(currentChar) >= ASC("A") AND ASC(currentChar) <= ASC("Z") THEN
      shiftedASCII = ASC(currentChar) + value

      ' WRAP AROUND
      IF (shiftedASCII > ASC("Z")) THEN
        shiftedASCII = shiftedASCII - 26
      END IF

      ' Encrypted a character
      MID(encryptedText,i,1) = CHR(shiftedASCII)

    ' NOT A LETTER
    ELSE
      ' No change to original
    END IF
  NEXT i

  RETURN encryptedText

END FUNCTION


' DECRYPT FUNCTION
FUNCTION DECRYPT (message AS STRING, value AS INTEGER) AS STRING
  ' Modify value
  value = value MOD 26
  ' Just decrypt it backwards
  RETURN ENCRYPT(message, 26-value)
END FUNCTION


' SOLVE SUBROUTINE/FUNCTION
SUB SOLVE (message AS STRING, maxCaesarValue AS INTEGER)
  FOR i AS INTEGER = 0 to maxCaesarValue
    PRINT "Caesar"; maxCaesarValue-i; ": "; ENCRYPT(message, maxCaesarValue-i)
  NEXT i
END SUB
```

```
' MAIN FUNCTION
' Assign values
DIM caesarValue AS INTEGER = 13
DIM plainText AS STRING = "JUST BASIC BEING BASIC"
PRINT plainText; " +";caesarValue


' Encrypt the text
DIM encryptedText AS STRING = ENCRYPT(plainText, caesarValue)
PRINT "Encrypted text: "; encryptedText

' Decrypt the text
DIM decryptedText AS STRING = DECRYPT(encryptedText, caesarValue)
PRINT "Decrypted text: "; decryptedText

' Solve the cipher to JPU
PRINT "Solve: JPU"
SOLVE("JPU", 25)
```

3.2 BASIC Test Output

```
JUST BASIC BEING BASIC + 13
Encrypted text: WHFG ONFVP ORVAT ONFVP
Decrypted text: JUST BASIC BEING BASIC
Solve: JPU
Caesar 25: IOT
Caesar 24: HNS
Caesar 23: GMR
Caesar 22: FLQ
Caesar 21: EKP
Caesar 20: DJO

...
```

3.3 BASIC Thoughts

Following Fortran, I was much happier to work with BASIC. I started off by Googling some interesting things:

- basic variable declaration
- printing basic programming
- basic langauge

This slowly became annoying because I would get results for introductory programming or easy languages to learn. So I switched over to Chat GPT and got much much better results. Here's what I asked:

- give me a crash course on coding in BASIC
- i want int and string variables, functions, and for loops
- how can i write a function
- how to write if statement, give one example
- whats the AND operator look like

You'll see similar questions in the future, but these and future sentences are all ACTUAL questions I asked. Each answer gave code snippets that were almost always correct. This method of learning a language proved to be incredibly efficient. I am not one to usually use AI, but I think this is a good way to do it, while still writing 100% of your own code.

So what did I think about the language? I really liked it. I really take variable declaration for granted, writing it whenever was a nice change of pace. This syntax make sense: `DIM shiftedASCII AS INTEGER = 0`. It is readable for sure. Functions look more modern here with better return types and much better return statements. In terms of writeability, there was a lot of consistency throughout similar things like ending a code block with `END`.

4 Pascal

4.1 Pascal Code

```
program CasearCipher;

{Encryption Function}
FUNCTION ENCRYPT(message: String; value: Integer): String;
var
  i: Integer;
  encryptedText: String;
  currentChar: Char;
  shiftedASCII: Integer;

begin
  {Modify value}
  value := value MOD 26;
  {Start with original message}
  encryptedText := message;

  FOR i := 1 to Length(message) DO begin
    currentChar := Copy(message,i,1)[1]; {convert string to char}

    {Letter}
    IF (Ord(currentChar) >= Ord('A')) AND (Ord(currentChar) <= Ord('Z')) THEN begin
      shiftedASCII := Ord(currentChar) + value;

      {Wrap Around}
      IF (shiftedASCII > Ord('Z')) THEN begin
        shiftedASCII := shiftedASCII - 26;
      end;

      {Encrypted a Character}
      encryptedText[i] := Chr(shiftedASCII);

    end
    ELSE begin
      {Not a letter, no change}
    end;
  end;

  ENCRYPT := encryptedText;
end;


{Decryption Function}
FUNCTION DECRYPT(message: String; value: Integer): String;
begin
  {Modify value}
  value := value MOD 26;
  DECRYPT := ENCRYPT(message, 26-value);
end;


{Solve Function/Procedure}
PROCEDURE SOLVE (message: String; maxCaesarValue: Integer);
var
  i: Integer;

begin
  {Loop desired encryption}
  FOR i := 0 to maxCaesarValue DO begin
    writeln ('Caesar ', maxCaesarValue-i, ': ', ENCRYPT(message, maxCaesarValue-i));
  end;
end;
```

```
{MAIN PROGRAM}
var
  caesarValue: Integer;
  plainText: String;
  encryptedText: String;
  decryptedText: String;
  i: Integer;

begin
  {Assign Values}
  caesarValue := 125;
  plainText := 'MR PEDRO PASCAL';
  writeln (plainText, ' + ', caesarValue);

  {Encrypt the text}
  encryptedText := ENCRYPT(plainText, caesarValue);
  writeln ('Encrypted text: ', encryptedText);

  {Decrypt the text}
  decryptedText := DECRYPT(encryptedText, caesarValue);
  writeln ('Decrypted text: ', decryptedText);

  {Solve the Cipher}
  writeln ('Solve: HAL');
  SOLVE('HAL', 30);

end.
```

4.2 Pascal Test Output

```
MR PEDRO PASCAL + 125
Encrypted text: HM KZYMJ KVNXVG
Decrypted text: MR PEDRO PASCAL
Solve: HAL
Caesar 30: LEP
Caesar 29: KDO
Caesar 28: JCN
Caesar 27: IBM
Caesar 26: HAL
Caesar 25: GZK
```

...

4.3 Pascal Thoughts

I enjoyed Pascal. At this point, it was my favorite language. I enjoyed the walrus assignment operator, the simpler separation of variables, the simple string substring functions and the seemless return types of functions. When looking for help I once again asked ChatGPT a few things:

- i am learning Pascal coding. give me the basics
- how to write a function and show how variable declarations work in that and the main program
- show me how to get substring. show me how to convert char into ascii. show me how to convert ascii to char.

The semi colons in this language remind me of the C family and many more modern languages that derive from similar languages. Messed up a few time writing double quotes inside of single quotes, but easy fixable error. At this point the process of learning an rewriting a Caesar Cipher program was getting easier, so there were less hiccups. We are still ending blocks with `END`, but braces have arrived, even though they are for comments.

This language is incredibly readable. There are no big confusing of foreign words in the syntax I used. Everything makes sense and is similar enough to pseudo-code. More symbols are utilized rather than key words, which can affect writeability and readability in both positive and negative way, but overall positive in my opinion. Chat GPT had no issue helping me out, so learning cost was great. The occasional Google search was no hassle either.

5 COBOL

5.1 COBOL Code

Oh god COBOL? Isn't this supposed to be the worst language ever in the whole entire universe?

```
IDENTIFICATION DIVISION.
  PROGRAM-ID. CAESAR-CIPHER.

DATA DIVISION.
  WORKING-STORAGE SECTION.
    01 plainText PIC X(17).
    01 caesarValue PIC 9(2).

    01 encryptedText PIC X(17).
    01 decryptedText PIC X(17).
    01 extraValue PIC 9(2).

    01 i PIC 9(2).
    01 shiftedASCII PIC 9(3).
    01 currentChar PIC X(1).

    01 j PIC 9(2).
    01 maxShiftValue PIC 9(2).

PROCEDURE DIVISION.
  MAIN_PROCEDURE.

    *> Assign values <
    MOVE "MRS HARMONY COBOL" TO plainText.
    MOVE 20 TO caesarValue.
    DISPLAY plainText " + " caesarValue.

    *> Encrypt the text <
    PERFORM ENCRYPT.
    DISPLAY "Encrypted text: " encryptedText.

    *> Decrypt the text <
    PERFORM DECRYPT.
    DISPLAY "Decrypted text: " decryptedText.

    *> Solve the cipher for GLAK JWVVAFY <
    MOVE "HAL" TO plainText
    MOVE 10 TO maxShiftValue.
    DISPLAY "Solve: " plainText.
    PERFORM SOLVE.

  STOP RUN.


*> Encrypt function <
ENCRYPT.

  *> Modify value <
    DIVIDE caesarValue BY 26 GIVING extraValue
      REMAINDER caesarValue.
  *> Start with original message <
  MOVE plainText TO encryptedText.

  *> Encrypt each character <
  PERFORM VARYING i FROM 1 BY 1 UNTIL i > FUNCTION LENGTH(plainText)
    MOVE plainText(i:1) TO currentChar

    *> Letter <
    IF FUNCTION ORD(currentChar) >= FUNCTION ORD("A") AND
      FUNCTION ORD(currentChar) <= FUNCTION ORD("Z") THEN

      MOVE FUNCTION ORD(currentChar) TO shiftedASCII
      ADD caesarValue TO shiftedASCII

      *> Wrap around <
      IF shiftedASCII > FUNCTION ORD("Z") THEN
        SUBTRACT 26 FROM shiftedASCII
      END-IF

      *> Encrypt a character <
      MOVE FUNCTION CHAR(shiftedASCII) TO encryptedText(i:1)

    END-IF
  END-PERFORM.
```

```
*> Decrypt function <
DECRYPT.

  MOVE encryptedText TO plainText.
  SUBTRACT caesarValue FROM 26 GIVING caesarValue.

  PERFORM ENCRYPT.

  MOVE encryptedText TO decryptedText.


*> Solve Function <
SOLVE.

  *> Loop desired encryption <
  PERFORM VARYING j FROM 0 BY 1 UNTIL j > maxShiftValue
    SUBTRACT j FROM maxShiftValue GIVING caesarValue
    PERFORM ENCRYPT
    DISPLAY "Caesar " caesarValue ": " encryptedText
  END-PERFORM.
```

5.2 COBOL Test Output

```
MRS HARMONY COBOL + 20
Encrypted text: GLM BULGIHS WIVIF
Decrypted text: MRS HARMONY COBOL
Solve: HAL
Caesar 10: RKV
Caesar 09: QJU
Caesar 08: PIT
Caesar 07: OHS
Caesar 06: NGR
Caesar 05: MFQ
Caesar 04: LEP
Caesar 03: KDO
Caesar 02: JCN
Caesar 01: IBM
Caesar 00: HAL
```

5.3 COBOL Thoughts

Yea wasn't a huge fan of this language but my learning method has become so perfected that I didn't struggle too hard. Not the worst thing ever I'll say. My Immediate COBOL thoughts were that its very wordy. Also, my first snag came trying to calculate higher caesar values:

```
MRS HARMONY COBOL + 19
Encrypted text: F[?][?] AT[?]FHG[?] VHUHE
Plain: F[?][?] AT[?]FHG[?] VHUHE
```

I solved this quickly after 5 or 10 minutes. The issue was that I declared an integer to be only 2 digits in length, so the ASCII value of the shifted letter would go past 99, wrapping back to 0 and printing very low ASCII values. Silly COBOL!

Also someone in the library smelled like my old neighbor and reminded me how interesting scent memory is.

One quirk of this language is the lack of functions (at least from what I could find). Using other sub procedures worked well, but the lack of parameters or variable scope added a few extra lines to switch variables in an out. I reused `ENCRYPT` inside `DECRYPT` so thing variables had to be reassigned.

I did not like the lack of symbols. `ADD` and `DIVIDE` along with others are better as symbols. The way I wrote data types are incredibly unreadable: strings are written as `PIC X(length)` and numbers are `PIC 9(digits)`. I did not attempt looking much COBOL up accept for the `MOD` function on a page that's older than a few of my cousins (ibmmainframes.com/about36419.html). Oh and those periods show up most of the time but then when away inside a block. But most of this and all programs are IF or PERFORM blocks.

6 Scala

6.1 Scala Code

Scala Scala Scala...

```
object CaesarCipher {
  def main(args: Array[String]): Unit = {

    //Encrypt Function
    def encrypt(message: String, caesarValue: Int): String = {

      //Modify the value
      var value = (caesarValue % 26)
      //Start with nothing because string are immutable
      var encryptedText: String = ""

      for (i <- 0 to message.length-1 by 1) {
        var currentChar: Char = message.substring(i,i+1).charAt(0)
        var shiftedASCII: Int = 0;

        //Letter
        if (currentChar.toInt >= 'A'.toInt && currentChar.toInt <= 'Z'.toInt) {
          shiftedASCII = currentChar.toInt + value

          //Wrap around
          if (shiftedASCII > 'Z'.toInt) {
            shiftedASCII = shiftedASCII - 26
          }

          //Encrypted a character
          encryptedText = encryptedText + shiftedASCII.toChar
        }

        //Not a letter
        else {
          encryptedText = encryptedText + currentChar
        }
      }
      // DONE!
      s"$encryptedText"
    }


    //Decrypt Function
    def decrypt(message: String, caesarValue: Int): String = {
      //Modify the value
      var value = (caesarValue % 26)
      //Just encrypt it but the compliment
      var decryptedText = encrypt(message, 26-value)
      s"$decryptedText"
    }


    //Solve Function
    def solve(message: String, maxCaesarValue: Int): Unit = {
      //Loop desired encryptions
      for (i <- maxCaesarValue to 0 by -1) {
        println("Caesar " + i + ": "+ encrypt(message, i))
      }
    }
```

```
58      //Main Program
59      //Assign Values
60      var caesarValue = 22
61      var plainText = "SCALAS AND ALL"
62      println(plainText + " + " + caesarValue)
63
64      //Encrypt the text
65      var encryptedText = encrypt(plainText, caesarValue)
66      println("Encrypted text: " + encryptedText)
67
68      //Decrypt the text
69      var decryptedText = decrypt(encryptedText, caesarValue)
70      println("Decrypted text: " + decryptedText)
71
72      //Solve the cipher
73      println("Solve: URAEL SBAQN")
74      solve("URAEL SBAQN", 19)
75
76    }
77  }
```

6.2 Scala Test Output

```
SCALAS AND ALL + 22
Encrypted text: OYWHWO WJZ WHH
Decrypted text: SCALAS AND ALL
Solve: URAEL SBAQN
Caesar 19: NKTXE LUTJG
Caesar 18: MJSWD KTSIF
Caesar 17: LIRVC JSRHE
Caesar 16: KHQUB IRQGD
Caesar 15: JGPTA HQPFC
Caesar 14: IFOSZ GPOEB
Caesar 13: HENRY FONDA
Caesar 12: GDMQX ENMCZ
Caesar 11: FCLPW DMLBY
Caesar 10: EBKOV CLKAX
Caesar 9: DAJNU BKJZW
Caesar 8: CZIMT AJIYV
Caesar 7: BYHLS ZIHXU
Caesar 6: AXGKR YHGWT
Caesar 5: ZWFJQ XGFVS
Caesar 4: YVEIP WFEUR
Caesar 3: XUDHO VEDTQ
Caesar 2: WTCGN UDCSP
Caesar 1: VSBFM TCBRO
Caesar 0: URAEL SBAQN
```

6.3 Thoughts

Was Scala my favorite program? Yes, spoiler. I enjoyed this one a lot, because of its similarity to Java languages. Even `Main string(args[])` decided to show up! The semi colon and braces are prominent in this language and keep the readability high. Variables can be declared like in JavaScript. But also like in TypeScript, which I can absolutely be behind. What else can I say, I'm getting really bored talking about this. For loops looked a little funny, the arrow is a little random but the rest of the code inside there is super readable. Like come on dude: `for (i <- 0 to 10 by 1)`.

Here's some Chat GPT messages, a bit repetitive:

- how do you create a function with string and int parameters, returning a string
- how to get length of a string and substring of a string. are strings mutable?
- function without return type

This was the one language where strings were not mutable, so I had to go back to how we originally learned to program, not too much of a drawback, just a little algorithm edit. Also, capitalization mattered here, forced uniformity which I like.

7 Conclusion

7.1 Overall Thoughts

One observation I made during this project was the importance of the text editor used. I ended up using the site One Compiler for all five languages. Writeability was definitely improved when seeing all the different colors.

When I completed COBOL, my fourth language, I had to go back and fix all my programs. I had to make sure `Decrypt` and `Solve` were their own functions. In doing this seemingly simple task in modern languages, I realized which languages were a pain in making these small adjustments. A few were only a bit more lines of code, but others like Fortran were a real pain in the ass, re-declaring variables, even though my `decrypt` function is literally just a redirect to `encrypt(message, 26-value)`.

7.2 Rankings

My overall rankings in terms of fun, readability, writeability, and vibes:

1. Scala
    a) Fun language, high readability and writeability. Good use of symbols.
    b) Time taken: **1:18** (Predicted 2:00)
2. Pascal
    a) Good mix of symbols and words. Variables had to be declared prior but it was simple.
    b) Time taken: **1:43** (Predicted 1:00)
3. BASIC
    a) Reaching over into more words than symbols over here. Didn't hate it.
    b) Time taken: **1:55** (Predicted 1:30)
4. COBOL
    a) Very verbose, difficult to read many lines and writing without symbols. But many blocks of code are extremely readable.
    b) Time taken: **2:30** (Predicted 1:30)
5. Fortran
    a) This one screwed me over and I took it personally. If it wasn't first maybe it would have a higher ranking.
    b) Time taken: **4:18** (Predicted 2:00)

And look, it just so happens that my favorite languages were quicker to write in.