# Fortran

## Commentary

I started with a search for “Fortran examples.” At this point, a new problem presented itself; it became apparent that there are enough versions of fortran that my search wasn’t all that enlightening. I then went to the official documentation, which was somewhat helpful. I was able to quickly learn a lot of the syntax and how common things like variables and loops are implemented in Fortran. I noticed I didn’t see a section for “string methods” or “helper functions.” At this point I knew I would be writing those. I started programming a hello world on IDEone.com and got it on my first try but the compiler gave me a warning about using tabs. Did Fortran come before tabs? I wind up googling about how to get rid of it and I’m told to use "-Wtabs" as a flag when compiling. I ignore this because I’m not sure if I can control that on IDEone. My approach for creating the Caesar cipher is to make a function to handle shifting a single letter, and then carry that to the whole word in the encrypt and decrypt functions. I’m happily coding along at this point and while trying to get a single letter to work, I come across a runtime error. I wish I could tell you what it was for, but I’m just passing along the same message Fortran gave me, “Runtime Error.” This was painful to debug, I fiddled around for 15 minutes and deleted everything I changed and re-implemented it, somehow working the second time. Getting the loop to work was a pain. The syntax for accessing the index of a string is a bit strange, getting a slice of one element. After implementing the for loop, my function would successfully compute the encrypted cipher-text but there were random characters like “���▯�� �V@��V” following the result. I was able to fix this after some help from a classmate, and figuring out I can just replace the empty space at the end of a string with space characters.

## Google Searches

- fortran examples
- fortran hello world
- fortran getting started
- fortran string methods
- fortran helper functions
- fortran compiler warning tabs
- ideone compiler settings
- fortran character to int
- fortran strings
- fortran "runtime error"
- fortran index of
- fortran illegible characters end of string

## Caesar Implementation

```fortran
! compile and run with gfortran caesar.f90 && ./a.out
program Caesar
    character(1) :: shift
    character(64) :: encrypt
    character(64) :: decrypt
    print *, encrypt('ATTACK AT ONCE', 4)
    print *, decrypt('EXXEGO EX SRGI', 4)

    call solve("abcdeFGHIJKLmnopqrstuvwxyz ,?;{[()]}", 26)

    stop
end

function encrypt(str, shiftAmt) result(res)
    character(*) :: str
    character(1) :: shift
    integer :: shiftAmt
    character(*) :: res
    integer :: i
    integer :: j

    do i = 1, len_trim(str)
        res(i:i) = shift(str(i:i), shiftAmt)
    end do
    do j = len_trim(str) + 1, 64
       res(j:j) = ' '
    end do


end function

function decrypt(str, shiftAmt) result(res)
    character(*) :: str
    integer :: shiftAmt
    character(64) :: res
    character(64) :: encrypt


    res = encrypt(str, -shiftAmt)

end function

subroutine solve(str, maxShiftValue)
    implicit none
    character(*) :: str
    integer :: maxShiftValue
    character(64) :: decrypt
    integer :: i

    do i = 1, maxShiftValue
        print *, 'Shift: ', i, decrypt(str, i)
    end do

end subroutine

function toUpper(chr) result(upper)
    implicit none
    character :: chr
```

```fortran
    character :: upper
    integer :: charcode

    charcode = iachar(chr)

    if(charcode >= 97 .and. charcode <= 122) then
        ! lowercase
        upper = achar(charcode - 32)
    else
        upper = chr
    end if

end function

function shift(chr, num) result(shifted)
    implicit none
    character :: chr
    integer :: num
    character :: shifted
    integer :: shiftedCode
    character :: toUpper

    shiftedCode = iachar(toUpper(chr))

    if(shiftedCode >= 64 .and. shiftedCode <= 90) then
        shiftedCode = modulo(shiftedCode - 65 + num, 26) + 65
    end if

    shifted = achar(shiftedCode)

    !print *, chr, ' -> ', shifted


end function
```

Output

```
EXXEGO EX SRGI
ATTACK AT ONCE
Shift:            1 ZABCDEFGHIJKLMNOPQRSTUVWXY ,?;{[()]}
Shift:            2 YZABCDEFGHIJKLMNOPQRSTUVWX ,?;{[()]}
Shift:            3 XYZABCDEFGHIJKLMNOPQRSTUVW ,?;{[()]}
Shift:            4 WXYZABCDEFGHIJKLMNOPQRSTUV ,?;{[()]}
Shift:            5 VWXYZABCDEFGHIJKLMNOPQRSTU ,?;{[()]}
Shift:            6 UVWXYZABCDEFGHIJKLMNOPQRST ,?;{[()]}
Shift:            7 TUVWXYZABCDEFGHIJKLMNOPQRS ,?;{[()]}
Shift:            8 STUVWXYZABCDEFGHIJKLMNOPQR ,?;{[()]}
Shift:            9 RSTUVWXYZABCDEFGHIJKLMNOPQ ,?;{[()]}
Shift:           10 QRSTUVWXYZABCDEFGHIJKLMNOP ,?;{[()]}
Shift:           11 PQRSTUVWXYZABCDEFGHIJKLMNO ,?;{[()]}
Shift:           12 OPQRSTUVWXYZABCDEFGHIJKLMN ,?;{[()]}
```

```
Shift:           13 NOPQRSTUVWXYZABCDEFGHIJKLM ,?;{[()]}
Shift:           14 MNOPQRSTUVWXYZABCDEFGHIJKL ,?;{[()]}
Shift:           15 LMNOPQRSTUVWXYZABCDEFGHIJK ,?;{[()]}
Shift:           16 KLMNOPQRSTUVWXYZABCDEFGHIJ ,?;{[()]}
Shift:           17 JKLMNOPQRSTUVWXYZABCDEFGHI ,?;{[()]}
Shift:           18 IJKLMNOPQRSTUVWXYZABCDEFGH ,?;{[()]}
Shift:           19 HIJKLMNOPQRSTUVWXYZABCDEFG ,?;{[()]}
Shift:           20 GHIJKLMNOPQRSTUVWXYZABCDEF ,?;{[()]}
Shift:           21 FGHIJKLMNOPQRSTUVWXYZABCDE ,?;{[()]}
Shift:           22 EFGHIJKLMNOPQRSTUVWXYZABCD ,?;{[()]}
Shift:           23 DEFGHIJKLMNOPQRSTUVWXYZABC ,?;{[()]}
Shift:           24 CDEFGHIJKLMNOPQRSTUVWXYZAB ,?;{[()]}
Shift:           25 BCDEFGHIJKLMNOPQRSTUVWXYZA ,?;{[()]}
Shift:           26 ABCDEFGHIJKLMNOPQRSTUVWXYZ ,?;{[()]}
```

## Log

Prediction: 2 hours

| Date | Hours Spent | Accomplishments |
|---|---|---|
| 1/26 | 2 | Get familiar with Fortran, Create a function to shift a single letter |
| 1/27 | 1 | Figure out how to apply it to a variable length string |
| 1/27 | 1 | Handle non alphabetic characters |
| 1/27 | 0.5 | Create “solve” subroutine |
| 2/18 | 0.5 | Fix the random characters caused by overallocating |
| 2/21 | 0.5 | Test thoroughly |

### Discrepancy of time

I think the discrepancy in time taken to complete this program is just how different this language is from most of what I've done in the past. Also, IDEone definitely added time, my development was much smoother once I removed them from the equation. This was also my first iteration through creating a caesar cipher, at least in a little while, so I had to actually debug my code to make sure my standard algorithm was working. This increasingly led to extra time with the terrible error messages given in the event of a runtime error.

## Overall Review

Fortran was tough to develop mainly because of how different some of the syntax was. The readability and writability are both not that great. One thing in particular that was annoying was having to declare the types of everything multiple times when it should be able to inferred by the context of the program. This may have increased some readability by having the types located close to the functions but I feel the writability was hurt more than that. I'm not a huge fan of the use of end in place of curly braces, they tend to blend into the rest of the program. I think programming languages made the right move going forward with the use of symbols. It's also unclear to me why the for loops don't have parentheses but the if statements do, this seems like something that should be consistent throughout he language.

## Ratings

Readability: 5/10

Writability: 4/10

Ranking: 4/5

# COBOL

## Commentary

For COBOL, I knew it was a much more structured language and that a simple hello world would not be simple, similar to Java, but worse. I started by looking for a youtube video to get a general overview. The only real video for beginners was COBOL in 100 seconds by fireship, a channel I subscribe to and watch regularly. This allowed me to understand the tip of the iceberg, and nothing more, and from there I copied a hello world example into IDEone. Doesn’t compile. After debugging this for way longer than I should have the problem was related to how the code was spaced. Little do I know how much time I would save if I was to just abandon IDEone at this point. I end up counting out the correct number of spaces for each line and eventually get it to compile. *woot!* My next problem was to find any sort of documentation of the keywords. It seems IBM is the only real provider of this. I also hate IBM’s website, I’ve never seen such a bloated website for documentation, only for it to be hard to find any useful info, no code snippets, no getting started tutorials, a long way from geeksforgeeks.org. Maximum line length was a big blocker for a while, I couldn’t figure out how to override it or anything so I eventually came up with breaking it up into smaller statements and hold the intermediate information in temp variables. This problem wasn’t made any easier by the amount of text it takes to call a function. Why should I have to write FUNCTION in front of everything I want to use? Also, I was never able to find a way to use functions, I'm just kinda guessing the way I did it is reasonable. I am basically using a combination of GOTOs and using some global variables. They have to exist... right??

## Google Searches

- COBOL getting started
- Fireship cobol 100 seconds
- GET ASCII CODE FROM CHARACTER IN COBOL
- COBOL TO UPPERCASE
- CREATE A FUNCTION IN COBOL (spent like 30 minutes here just trying to figure out if they exist or not)
- FIX COBOL LINE LENGTH TOO LONG
- COBOL LOOP OVER STRING
- Split cobol code onto multiple lines – couldn’t find an answer to this
- COBOL compiler ubuntu

## Caesar Implementation

NOTE: This language sucks so much it doesn't have syntax highlighting

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CAESAR.
       ENVIRONMENT DIVISION.
       DATA DIVISION.

       WORKING-STORAGE SECTION.

       01 INP PIC X(36) VALUE "ATTACK AT ONCE".
       01 SHIFT PIC 99 VALUE 4.
       01 OUT PIC X(36).
       01 LEN PIC 999.
       01 TMP1 PIC 999.
       01 TMP2 PIC 999.
       01 I PIC 999.
       01 S PIC 999.
       01 TMPSHIFT PIC 99.

       PROCEDURE DIVISION.

           MOVE FUNCTION UPPER-CASE(INP) TO INP.

           PERFORM ENCRYPT.

           DISPLAY FUNCTION TRIM(OUT).

           MOVE OUT TO INP.

           PERFORM DECRYPT.

           DISPLAY FUNCTION TRIM(OUT).

           MOVE 'abcdeFGHIJKLmnopqrstuvwxyz' TO INP.
           MOVE FUNCTION UPPER-CASE(INP) TO INP.

           PERFORM SOLVE.

           STOP RUN.

           ENCRYPT.
           MOVE FUNCTION LENGTH(INP) TO LEN

           PERFORM VARYING I FROM 1 BY 1 UNTIL I > LEN
               MOVE FUNCTION ORD(INP(I:1)) TO TMP1
               IF TMP1 IS NOT EQUAL TO 33
               MOVE FUNCTION MOD(TMP1 - 66 + SHIFT, 26) TO TMP2
               MOVE FUNCTION CHAR(TMP2 + 66) TO OUT(I:1)
           END-PERFORM.

           DECRYPT.
           MOVE FUNCTION LENGTH(INP) TO LEN

           PERFORM VARYING I FROM 1 BY 1 UNTIL I > LEN
               MOVE FUNCTION ORD(INP(I:1)) TO TMP1
               IF TMP1 IS NOT EQUAL TO 33
               MOVE FUNCTION MOD(TMP1 - 66 - SHIFT, 26) TO TMP2
               MOVE FUNCTION CHAR(TMP2 + 66) TO OUT(I:1)
           END-PERFORM.

           SOLVE.
           DISPLAY "SOLVING...".
           MOVE SHIFT TO TMPSHIFT.
           PERFORM VARYING S FROM 1 BY 1 UNTIL S > 26
```

```
                MOVE S TO SHIFT
                PERFORM DECRYPT
                DISPLAY "SHIFT " SHIFT " " FUNCTION TRIM(OUT)
            END-PERFORM.
            MOVE TMPSHIFT TO SHIFT.
```

Output

```
EXXEGO EX SRGI
ATTACK AT ONCE
SOLVING...
SHIFT 01 ZABCDEFGHIJKLMNOPQRSTUVWXY
SHIFT 02 YZABCDEFGHIJKLMNOPQRSTUVWX
SHIFT 03 XYZABCDEFGHIJKLMNOPQRSTUVW
SHIFT 04 WXYZABCDEFGHIJKLMNOPQRSTUV
SHIFT 05 VWXYZABCDEFGHIJKLMNOPQRSTU
SHIFT 06 UVWXYZABCDEFGHIJKLMNOPQRST
SHIFT 07 TUVWXYZABCDEFGHIJKLMNOPQRS
SHIFT 08 STUVWXYZABCDEFGHIJKLMNOPQR
SHIFT 09 RSTUVWXYZABCDEFGHIJKLMNOPQ
SHIFT 10 QRSTUVWXYZABCDEFGHIJKLMNOP
SHIFT 11 PQRSTUVWXYZABCDEFGHIJKLMNO
SHIFT 12 OPQRSTUVWXYZABCDEFGHIJKLMN
SHIFT 13 NOPQRSTUVWXYZABCDEFGHIJKLM
SHIFT 14 MNOPQRSTUVWXYZABCDEFGHIJKL
SHIFT 15 LMNOPQRSTUVWXYZABCDEFGHIJK
SHIFT 16 KLMNOPQRSTUVWXYZABCDEFGHIJ
SHIFT 17 JKLMNOPQRSTUVWXYZABCDEFGHI
SHIFT 18 IJKLMNOPQRSTUVWXYZABCDEFGH
SHIFT 19 HIJKLMNOPQRSTUVWXYZABCDEFG
SHIFT 20 GHIJKLMNOPQRSTUVWXYZABCDEF
SHIFT 21 FGHIJKLMNOPQRSTUVWXYZABCDE
SHIFT 22 EFGHIJKLMNOPQRSTUVWXYZABCD
SHIFT 23 DEFGHIJKLMNOPQRSTUVWXYZABC
SHIFT 24 CDEFGHIJKLMNOPQRSTUVWXYZAB
SHIFT 25 BCDEFGHIJKLMNOPQRSTUVWXYZA
SHIFT 26 ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

## Log

| Date | Hours Spent | Accomplishments |
|---|---|---|
| 1/27 | 1 | RESEARCH COBOL |
| 1/27 | 2 | GET AROUND MAXIMUM LINE LENGTH, I NEVER WOULD’VE EXPECTED THIS PROBLEM |
| 1/28 | 1 | Install cobol compiler and set up local dev environment |

| **Date** | **Hours Spent** | **Accomplishments** |
|---|---|---|
| 1/28 | 1 | Transform encrypt function into decrypt and solve |
| 2/21 | .5 | Test thoroughly |

### Discrepancy of time

The biggest setbacks I had in this were just the very limited amount of help I could get from the internet. I know there is a meme going around on the internet that programmers are just good at google, but I guess COBOL programmers must be a different breed. I couldn't understand any resources that were provided or they were just way too wordy. I was caught in the endless loop of googling, clicking links, having too many words on the page, closing out and doing it again after 5 minutes.

## Overall Review

This took me by far the most time to implement. One of my biggest problems was finding good documentation. It just doesn’t exist, at least for a basic program like caesar cipher. The language was designed for business applications, so one shouldn’t expect COBOL to make sense here, but at the same time, a simple program shouldn’t ever be difficult to implement. By far the least writable of the languages, way too verbose and structured. It somewhat makes up for it with the readability which is very close to english. Overall, a terrible language but I can’t blame the creators of it because it suits the needs of that time period very well. Having specific columns of code for certain things makes no sense nowadays, but at the time, when punch cards were being used, I’m sure it made a lot of sense. Having to correctly space the columns was a huge downside, both for readability and writability.

## Ratings

Readability: 4/10

Writability: 1/10

Ranking: 5/5

# Basic

## Commentary

Wow… I must say this language is the best one I've used so far in this assignment, that doesn't say much with COBOL and Fortran as what I'm comparing with. It really isn't far off from today's modern languages. With the other "journal entries," I had written them as I went, but here, there wasn't enough friction to stop me from just coding all the way through. Half of the time spent getting this program to run was just finding out what version of BASIC I should use and setting up the compiler. I went with FreeBASIC as it seemed to have the best docs and was listed on your website. With the other languages, finding help was very hard compared to BASIC. It probably has something to do with how this was a common language for students to use so there are plentiful beginner documentations. COBOL being business oriented doesn't have an audience that is normally beginners. I was able to find a list of the string methods, and a way to loop over them and everything just kinda made sense based on my existing knowledge of high-level languages.

## Google Searches

- basic compiler
- basic getting started
- basic hello world
- basic ubuntu apt
- basic for loop
- freebasic string addition
- wordle
- freebasic string index
- freebasic modulus
- freebasic functions
- freebasic sub

## Caesar Implementation

```
'compile and run with fbc caesar.bas && ./caesar

declare function encrypt(word as String, shift as Integer) as String
declare function decrypt(word as String, shift as Integer) as String
declare Sub solve(word as String, maxShiftValue as Integer)

print encrypt("ATTACK AT ONCE", 4)
print decrypt("EXXEGO EX SRGI", 4)

solve("abcdeFGHIJKLmnopqrstuvwxyz ,?;{[()]}", 26)

function encrypt(word as String, shift as Integer) as String
    dim i as Integer
    word = ucase(word)

    for i = 0 TO len(word) - 1
```

```
        'print i, word[i], chr(word[i])

        if word[i] >= 64 and word[i] <= 90 then
            word[i] = (word[i] - 65 + shift) mod 26 + 65
        end if
    next

    Return(word)

end function

function decrypt(word as String, shift as Integer) as String
    Return(encrypt(word, 26-shift))
end function

sub solve(word as String, maxShiftValue as Integer)
    dim i as Integer

    for i = 1 to maxShiftValue
        'strings are modified in place so always shift by 1
        print "shift: " & i, decrypt(word, 1)
    next


end sub
```

Output

```
EXXEGO EX SRGI
ATTACK AT ONCE
shift: 1      ZABCDEFGHIJKLMNOPQRSTUVWXY ,?;{[()]}
shift: 2      YZABCDEFGHIJKLMNOPQRSTUVWX ,?;{[()]}
shift: 3      XYZABCDEFGHIJKLMNOPQRSTUVW ,?;{[()]}
shift: 4      WXYZABCDEFGHIJKLMNOPQRSTUV ,?;{[()]}
shift: 5      VWXYZABCDEFGHIJKLMNOPQRSTU ,?;{[()]}
shift: 6      UVWXYZABCDEFGHIJKLMNOPQRST ,?;{[()]}
shift: 7      TUVWXYZABCDEFGHIJKLMNOPQRS ,?;{[()]}
shift: 8      STUVWXYZABCDEFGHIJKLMNOPQR ,?;{[()]}
shift: 9      RSTUVWXYZABCDEFGHIJKLMNOPQ ,?;{[()]}
shift: 10     QRSTUVWXYZABCDEFGHIJKLMNOP ,?;{[()]}
shift: 11     PQRSTUVWXYZABCDEFGHIJKLMNO ,?;{[()]}
shift: 12     OPQRSTUVWXYZABCDEFGHIJKLMN ,?;{[()]}
shift: 13     NOPQRSTUVWXYZABCDEFGHIJKLM ,?;{[()]}
shift: 14     MNOPQRSTUVWXYZABCDEFGHIJKL ,?;{[()]}
shift: 15     LMNOPQRSTUVWXYZABCDEFGHIJK ,?;{[()]}
shift: 16     KLMNOPQRSTUVWXYZABCDEFGHIJ ,?;{[()]}
shift: 17     JKLMNOPQRSTUVWXYZABCDEFGHI ,?;{[()]}
shift: 18     IJKLMNOPQRSTUVWXYZABCDEFGH ,?;{[()]}
shift: 19     HIJKLMNOPQRSTUVWXYZABCDEFG ,?;{[()]}
shift: 20     GHIJKLMNOPQRSTUVWXYZABCDEF ,?;{[()]}
shift: 21     FGHIJKLMNOPQRSTUVWXYZABCDE ,?;{[()]}
```

```
shift: 22     EFGHIJKLMNOPQRSTUVWXYZABCD ,?;{[()]}
shift: 23     DEFGHIJKLMNOPQRSTUVWXYZABC ,?;{[()]}
shift: 24     CDEFGHIJKLMNOPQRSTUVWXYZAB ,?;{[()]}
shift: 25     BCDEFGHIJKLMNOPQRSTUVWXYZA ,?;{[()]}
shift: 26     ABCDEFGHIJKLMNOPQRSTUVWXYZ ,?;{[()]}
```

## Log

Prediction: 2 hours

| Date | Hours Spent | Accomplishments |
|---|---|---|
| 2/3 | 1 | Figure which version of basic and install compiler |
| 2/7 | .5 | Paste hello world example and find helper method docs |
| 2/7 | .5 | Implement encrypt, decrypt, solve |
| 2/21 | .5 | Test thoroughly |

### Discrepancy of time

I still went over in time here, but not in the initial pass through the program. I was pretty close to my estimate. The biggest loss in time was just figuring out which version of basic to use. I kept seeing results of visual basic and the likes. Also, there wasn't a package on apt for basic, I had to manually install a linux binary, which I should've just done and not tried to find a different version on apt.

## Overall Review

I really enjoyed this language, there were a few aspects which confused me. For example, declaring the function signatures at the top of the file and then having to completely re-write them when I want to actually use it. This made the program less writable and I don't see much of a benefit from doing it this way. The way the language requires all variables at the top of the function reminds me of Dr. Schwartz's SD1 class. He would always take off points when they weren't declared at the top and I never understood why.

## Ratings

Readability: 8/10

Writability: 7/10

Ranking: 2/5

# Pascal

## Commentary

After doing a few of these I feel like I have a good grasp of what I need from a language to make a caesar cipher. I start by finding a compiler and getting hello world to run. From there, I look up how to create a function. After that I create essentially an “echo” function, which just returns one of the parameters I give it. So far, Pascal has had very helpful online documentation and it is easy for me to find all of the resources I need. Next I get into the meat of the program, needing helper functions like toUpper, if the language has it, a way to do modulus, a looping structure, and a char to int conversion and vice versa. Once again, this info is readily available. I assemble this into the way that I know how to by now, and voila, a working program. No big problems arose during the development of this program.

## Google Searches

- Pascal getting started
- Freepascal ubuntu
- Pascal touppercase
- Pascal for loop syntax
- Pascal if syntax
- Pascal modulus
- Pascal character conversion
- Pascal check if character is a letter

## Caesar Implementation

```pascal
{compile and run with fpc -v0 caesar.pas && ./caesar}
program Hello;
uses sysutils;
function encrypt(str:string; shift:integer): string;
var i: integer;
begin
    str:=upcase(str);
    for i:=1 to length(str) do
        begin
            {test for if it is a letter, at this point all is uppercase}
            if (str[i] in ['A'..'Z']) then
                begin
                    str[i]:= chr((ord(str[i]) -65 + shift) mod 26 + 65);
                end;
        end;
    encrypt:=str;
end;

function decrypt(str:string; shift:integer): string;
var i: integer;
begin
    decrypt:=encrypt(str, 26-shift);
```

```pascal
end;

procedure solve(str:string; maxShiftValue:integer);
var i: integer;
begin
    str:=upcase(str);
    for i:=1 to 26 do
        begin
            writeln('Shift: ', i, ' Result: ', decrypt(str, i));
        end;
end;

begin
    writeln(encrypt('ATTACK AT ONCE',4));
    writeln(decrypt('EXXEGO EX SRGI',4));
    solve('abcdeFGHIJKLmnopqrstuvwxyz ,?;{[()]}', 26)
end.
```

Output

```
EXXEGO EX SRGI
ATTACK AT ONCE
Shift: 1 Result: ZABCDEFGHIJKLMNOPQRSTUVWXY ,?;{[()]}
Shift: 2 Result: YZABCDEFGHIJKLMNOPQRSTUVWX ,?;{[()]}
Shift: 3 Result: XYZABCDEFGHIJKLMNOPQRSTUVW ,?;{[()]}
Shift: 4 Result: WXYZABCDEFGHIJKLMNOPQRSTUV ,?;{[()]}
Shift: 5 Result: VWXYZABCDEFGHIJKLMNOPQRSTU ,?;{[()]}
Shift: 6 Result: UVWXYZABCDEFGHIJKLMNOPQRST ,?;{[()]}
Shift: 7 Result: TUVWXYZABCDEFGHIJKLMNOPQRS ,?;{[()]}
Shift: 8 Result: STUVWXYZABCDEFGHIJKLMNOPQR ,?;{[()]}
Shift: 9 Result: RSTUVWXYZABCDEFGHIJKLMNOPQ ,?;{[()]}
Shift: 10 Result: QRSTUVWXYZABCDEFGHIJKLMNOP ,?;{[()]}
Shift: 11 Result: PQRSTUVWXYZABCDEFGHIJKLMNO ,?;{[()]}
Shift: 12 Result: OPQRSTUVWXYZABCDEFGHIJKLMN ,?;{[()]}
Shift: 13 Result: NOPQRSTUVWXYZABCDEFGHIJKLM ,?;{[()]}
Shift: 14 Result: MNOPQRSTUVWXYZABCDEFGHIJKL ,?;{[()]}
Shift: 15 Result: LMNOPQRSTUVWXYZABCDEFGHIJK ,?;{[()]}
Shift: 16 Result: KLMNOPQRSTUVWXYZABCDEFGHIJ ,?;{[()]}
Shift: 17 Result: JKLMNOPQRSTUVWXYZABCDEFGHI ,?;{[()]}
Shift: 18 Result: IJKLMNOPQRSTUVWXYZABCDEFGH ,?;{[()]}
Shift: 19 Result: HIJKLMNOPQRSTUVWXYZABCDEFG ,?;{[()]}
Shift: 20 Result: GHIJKLMNOPQRSTUVWXYZABCDEF ,?;{[()]}
Shift: 21 Result: FGHIJKLMNOPQRSTUVWXYZABCDE ,?;{[()]}
Shift: 22 Result: EFGHIJKLMNOPQRSTUVWXYZABCD ,?;{[()]}
Shift: 23 Result: DEFGHIJKLMNOPQRSTUVWXYZABC ,?;{[()]}
Shift: 24 Result: CDEFGHIJKLMNOPQRSTUVWXYZAB ,?;{[()]}
Shift: 25 Result: BCDEFGHIJKLMNOPQRSTUVWXYZA ,?;{[()]}
Shift: 26 Result: ABCDEFGHIJKLMNOPQRSTUVWXYZ ,?;{[()]}
```

## Log

Prediction: 2 hours

| Date | Hours Spent | Accomplishments |
|---|---|---|
| 2/7 | 0.5 | Set up dev env |
| 2/10 | 0.5 | Find toUpper, modulus, for loops, and char int conversion documentation |
| 2/14 | 1 | Implement encrypt, decrypt, solve |
| 2/21 | 0.5 | Test thoroughly |

### Discrepancy of time

This language went by really quick. Largely due to good documentation, which I am learning to be one of the most important aspect of a language. I still went a little bit over budget due to extensive testing, but nothing really arose during the testing period except for a few minor tweaks.

## Overall Review

This language is a pretty good language. It has a lot of strange syntactical choices, I think. For example, the walrus operator for equals is not a common thing in other languages and adds to the cost of using the language. I know you have mentioned that you like this decision, but on my personal machine I use font ligatures which change the == into a different symbol so this was never a problem for me. They do similar odd things like this in a few spots... the use of writeln vs println, comments, semicolons in the function typing parameters instead of commas, thing like that do not some to add anything to the language and are just being done to be different. One thing that was kind of strange to me is the need to declare variables at the function header but the compiler gives a good enough error message so that it is something I can search for, and once again quickly resolve the problem. I found that the toUpper function is located in the sysutils library, so that is something new, I haven’t had to import anything in the other languages. I kind of like that extra functionality can be located in external libraries to reduce the size of the compiled program. This makes the language more orthogonal by default, while still allowing for extra convenience where it makes sense.

## Ratings

Readability: 7/10

Writability: 6/10

Ranking: 3/5

# Basic

## Commentary

Wow… I must say this language is the best one I've used so far in this assignment, that doesn't say much with COBOL and Fortran as what I'm comparing with. It really isn't far off from today's modern languages. With the other "journal entries," I had written them as I went, but here, there wasn't enough friction to stop me from just coding all the way through. Half of the time spent getting this program to run was just finding out what version of BASIC I should use and setting up the compiler. I went with FreeBASIC as it seemed to have the best docs and was listed on your website. With the other languages, finding help was very hard compared to BASIC. It probably has something to do with how this was a common language for students to use so there are plentiful beginner documentations. COBOL being business oriented doesn't have an audience that is normally beginners. I was able to find a list of the string methods, and a way to loop over them and everything just kinda made sense based on my existing knowledge of high-level languages.

## Google Searches

- basic compiler
- basic getting started
- basic hello world
- basic ubuntu apt
- basic for loop
- freebasic string addition
- wordle
- freebasic string index
- freebasic modulus
- freebasic functions
- freebasic sub

## Caesar Implementation

```
'compile and run with fbc caesar.bas && ./caesar

declare function encrypt(word as String, shift as Integer) as String
declare function decrypt(word as String, shift as Integer) as String
declare Sub solve(word as String, maxShiftValue as Integer)

print encrypt("ATTACK AT ONCE", 4)
print decrypt("EXXEGO EX SRGI", 4)

solve("abcdeFGHIJKLmnopqrstuvwxyz ,?;{[()]}", 26)

function encrypt(word as String, shift as Integer) as String
    dim i as Integer
    word = ucase(word)

    for i = 0 TO len(word) - 1
```

```
        'print i, word[i], chr(word[i])

        if word[i] >= 64 and word[i] <= 90 then
            word[i] = (word[i] - 65 + shift) mod 26 + 65
        end if
    next

    Return(word)

end function

function decrypt(word as String, shift as Integer) as String
    Return(encrypt(word, 26-shift))
end function

sub solve(word as String, maxShiftValue as Integer)
    dim i as Integer

    for i = 1 to maxShiftValue
        'strings are modified in place so always shift by 1
        print "shift: " & i, decrypt(word, 1)
    next


end sub
```

Output

```
EXXEGO EX SRGI
ATTACK AT ONCE
shift: 1      ZABCDEFGHIJKLMNOPQRSTUVWXY ,?;{[()]}
shift: 2      YZABCDEFGHIJKLMNOPQRSTUVWX ,?;{[()]}
shift: 3      XYZABCDEFGHIJKLMNOPQRSTUVW ,?;{[()]}
shift: 4      WXYZABCDEFGHIJKLMNOPQRSTUV ,?;{[()]}
shift: 5      VWXYZABCDEFGHIJKLMNOPQRSTU ,?;{[()]}
shift: 6      UVWXYZABCDEFGHIJKLMNOPQRST ,?;{[()]}
shift: 7      TUVWXYZABCDEFGHIJKLMNOPQRS ,?;{[()]}
shift: 8      STUVWXYZABCDEFGHIJKLMNOPQR ,?;{[()]}
shift: 9      RSTUVWXYZABCDEFGHIJKLMNOPQ ,?;{[()]}
shift: 10     QRSTUVWXYZABCDEFGHIJKLMNOP ,?;{[()]}
shift: 11     PQRSTUVWXYZABCDEFGHIJKLMNO ,?;{[()]}
shift: 12     OPQRSTUVWXYZABCDEFGHIJKLMN ,?;{[()]}
shift: 13     NOPQRSTUVWXYZABCDEFGHIJKLM ,?;{[()]}
shift: 14     MNOPQRSTUVWXYZABCDEFGHIJKL ,?;{[()]}
shift: 15     LMNOPQRSTUVWXYZABCDEFGHIJK ,?;{[()]}
shift: 16     KLMNOPQRSTUVWXYZABCDEFGHIJ ,?;{[()]}
shift: 17     JKLMNOPQRSTUVWXYZABCDEFGHI ,?;{[()]}
shift: 18     IJKLMNOPQRSTUVWXYZABCDEFGH ,?;{[()]}
shift: 19     HIJKLMNOPQRSTUVWXYZABCDEFG ,?;{[()]}
shift: 20     GHIJKLMNOPQRSTUVWXYZABCDEF ,?;{[()]}
shift: 21     FGHIJKLMNOPQRSTUVWXYZABCDE ,?;{[()]}
```

```
shift: 22     EFGHIJKLMNOPQRSTUVWXYZABCD ,?;{[()]}
shift: 23     DEFGHIJKLMNOPQRSTUVWXYZABC ,?;{[()]}
shift: 24     CDEFGHIJKLMNOPQRSTUVWXYZAB ,?;{[()]}
shift: 25     BCDEFGHIJKLMNOPQRSTUVWXYZA ,?;{[()]}
shift: 26     ABCDEFGHIJKLMNOPQRSTUVWXYZ ,?;{[()]}
```

## Log

Prediction: 2 hours

| Date | Hours Spent | Accomplishments |
|---|---|---|
| 2/3 | 1 | Figure which version of basic and install compiler |
| 2/7 | .5 | Paste hello world example and find helper method docs |
| 2/7 | .5 | Implement encrypt, decrypt, solve |
| 2/21 | .5 | Test thoroughly |

### Discrepancy of time

I still went over in time here, but not in the initial pass through the program. I was pretty close to my estimate. The biggest loss in time was just figuring out which version of basic to use. I kept seeing results of visual basic and the likes. Also, there wasn't a package on apt for basic, I had to manually install a linux binary, which I should've just done and not tried to find a different version on apt.

## Overall Review

I really enjoyed this language, there were a few aspects which confused me. For example, declaring the function signatures at the top of the file and then having to completely re-write them when I want to actually use it. This made the program less writable and I don't see much of a benefit from doing it this way. The way the language requires all variables at the top of the function reminds me of Dr. Schwartz's SD1 class. He would always take off points when they weren't declared at the top and I never understood why.

## Ratings

Readability: 8/10

Writability: 7/10

Ranking: 2/5

# COBOL

## Commentary

For COBOL, I knew it was a much more structured language and that a simple hello world would not be simple, similar to Java, but worse. I started by looking for a youtube video to get a general overview. The only real video for beginners was COBOL in 100 seconds by fireship, a channel I subscribe to and watch regularly. This allowed me to understand the tip of the iceberg, and nothing more, and from there I copied a hello world example into IDEone. Doesn’t compile. After debugging this for way longer than I should have the problem was related to how the code was spaced. Little do I know how much time I would save if I was to just abandon IDEone at this point. I end up counting out the correct number of spaces for each line and eventually get it to compile. *woot!* My next problem was to find any sort of documentation of the keywords. It seems IBM is the only real provider of this. I also hate IBM’s website, I’ve never seen such a bloated website for documentation, only for it to be hard to find any useful info, no code snippets, no getting started tutorials, a long way from geeksforgeeks.org. Maximum line length was a big blocker for a while, I couldn’t figure out how to override it or anything so I eventually came up with breaking it up into smaller statements and hold the intermediate information in temp variables. This problem wasn’t made any easier by the amount of text it takes to call a function. Why should I have to write FUNCTION in front of everything I want to use? Also, I was never able to find a way to use functions, I'm just kinda guessing the way I did it is reasonable. I am basically using a combination of GOTOs and using some global variables. They have to exist... right??

## Google Searches

- COBOL getting started
- Fireship cobol 100 seconds
- GET ASCII CODE FROM CHARACTER IN COBOL
- COBOL TO UPPERCASE
- CREATE A FUNCTION IN COBOL (spent like 30 minutes here just trying to figure out if they exist or not)
- FIX COBOL LINE LENGTH TOO LONG
- COBOL LOOP OVER STRING
- Split cobol code onto multiple lines – couldn’t find an answer to this
- COBOL compiler ubuntu

## Caesar Implementation

NOTE: This language sucks so much it doesn't have syntax highlighting

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CAESAR.
       ENVIRONMENT DIVISION.
       DATA DIVISION.

       WORKING-STORAGE SECTION.

       01 INP PIC X(36) VALUE "ATTACK AT ONCE".
       01 SHIFT PIC 99 VALUE 4.
       01 OUT PIC X(36).
       01 LEN PIC 999.
       01 TMP1 PIC 999.
       01 TMP2 PIC 999.
       01 I PIC 999.
       01 S PIC 999.
       01 TMPSHIFT PIC 99.

       PROCEDURE DIVISION.

           MOVE FUNCTION UPPER-CASE(INP) TO INP.

           PERFORM ENCRYPT.

           DISPLAY FUNCTION TRIM(OUT).

           MOVE OUT TO INP.

           PERFORM DECRYPT.

           DISPLAY FUNCTION TRIM(OUT).

           MOVE 'abcdeFGHIJKLmnopqrstuvwxyz' TO INP.
           MOVE FUNCTION UPPER-CASE(INP) TO INP.

           PERFORM SOLVE.

           STOP RUN.

           ENCRYPT.
           MOVE FUNCTION LENGTH(INP) TO LEN

           PERFORM VARYING I FROM 1 BY 1 UNTIL I > LEN
               MOVE FUNCTION ORD(INP(I:1)) TO TMP1
               IF TMP1 IS NOT EQUAL TO 33
               MOVE FUNCTION MOD(TMP1 - 66 + SHIFT, 26) TO TMP2
               MOVE FUNCTION CHAR(TMP2 + 66) TO OUT(I:1)
           END-PERFORM.

           DECRYPT.
           MOVE FUNCTION LENGTH(INP) TO LEN

           PERFORM VARYING I FROM 1 BY 1 UNTIL I > LEN
               MOVE FUNCTION ORD(INP(I:1)) TO TMP1
               IF TMP1 IS NOT EQUAL TO 33
               MOVE FUNCTION MOD(TMP1 - 66 - SHIFT, 26) TO TMP2
               MOVE FUNCTION CHAR(TMP2 + 66) TO OUT(I:1)
           END-PERFORM.

           SOLVE.
           DISPLAY "SOLVING...".
           MOVE SHIFT TO TMPSHIFT.
           PERFORM VARYING S FROM 1 BY 1 UNTIL S > 26
```

```
                    MOVE S TO SHIFT
                    PERFORM DECRYPT
                    DISPLAY "SHIFT " SHIFT " " FUNCTION TRIM(OUT)
                END-PERFORM.
                MOVE TMPSHIFT TO SHIFT.
```

Output

```
EXXEGO EX SRGI
ATTACK AT ONCE
SOLVING...
SHIFT 01 ZABCDEFGHIJKLMNOPQRSTUVWXY
SHIFT 02 YZABCDEFGHIJKLMNOPQRSTUVWX
SHIFT 03 XYZABCDEFGHIJKLMNOPQRSTUVW
SHIFT 04 WXYZABCDEFGHIJKLMNOPQRSTUV
SHIFT 05 VWXYZABCDEFGHIJKLMNOPQRSTU
SHIFT 06 UVWXYZABCDEFGHIJKLMNOPQRST
SHIFT 07 TUVWXYZABCDEFGHIJKLMNOPQRS
SHIFT 08 STUVWXYZABCDEFGHIJKLMNOPQR
SHIFT 09 RSTUVWXYZABCDEFGHIJKLMNOPQ
SHIFT 10 QRSTUVWXYZABCDEFGHIJKLMNOP
SHIFT 11 PQRSTUVWXYZABCDEFGHIJKLMNO
SHIFT 12 OPQRSTUVWXYZABCDEFGHIJKLMN
SHIFT 13 NOPQRSTUVWXYZABCDEFGHIJKLM
SHIFT 14 MNOPQRSTUVWXYZABCDEFGHIJKL
SHIFT 15 LMNOPQRSTUVWXYZABCDEFGHIJK
SHIFT 16 KLMNOPQRSTUVWXYZABCDEFGHIJ
SHIFT 17 JKLMNOPQRSTUVWXYZABCDEFGHI
SHIFT 18 IJKLMNOPQRSTUVWXYZABCDEFGH
SHIFT 19 HIJKLMNOPQRSTUVWXYZABCDEFG
SHIFT 20 GHIJKLMNOPQRSTUVWXYZABCDEF
SHIFT 21 FGHIJKLMNOPQRSTUVWXYZABCDE
SHIFT 22 EFGHIJKLMNOPQRSTUVWXYZABCD
SHIFT 23 DEFGHIJKLMNOPQRSTUVWXYZABC
SHIFT 24 CDEFGHIJKLMNOPQRSTUVWXYZAB
SHIFT 25 BCDEFGHIJKLMNOPQRSTUVWXYZA
SHIFT 26 ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

## Log

| Date | Hours Spent | Accomplishments |
|---|---|---|
| 1/27 | 1 | RESEARCH COBOL |
| 1/27 | 2 | GET AROUND MAXIMUM LINE LENGTH, I NEVER WOULD’VE EXPECTED THIS PROBLEM |
| 1/28 | 1 | Install cobol compiler and set up local dev environment |

| Date | Hours Spent | Accomplishments |
|---|---|---|
| 1/28 | 1 | Transform encrypt function into decrypt and solve |
| 2/21 | .5 | Test thoroughly |

### Discrepancy of time

The biggest setbacks I had in this were just the very limited amount of help I could get from the internet. I know there is a meme going around on the internet that programmers are just good at google, but I guess COBOL programmers must be a different breed. I couldn't understand any resources that were provided or they were just way too wordy. I was caught in the endless loop of googling, clicking links, having too many words on the page, closing out and doing it again after 5 minutes.

## Overall Review

This took me by far the most time to implement. One of my biggest problems was finding good documentation. It just doesn’t exist, at least for a basic program like caesar cipher. The language was designed for business applications, so one shouldn’t expect COBOL to make sense here, but at the same time, a simple program shouldn’t ever be difficult to implement. By far the least writable of the languages, way too verbose and structured. It somewhat makes up for it with the readability which is very close to english. Overall, a terrible language but I can’t blame the creators of it because it suits the needs of that time period very well. Having specific columns of code for certain things makes no sense nowadays, but at the time, when punch cards were being used, I’m sure it made a lot of sense. Having to correctly space the columns was a huge downside, both for readability and writability.

## Ratings

Readability: 4/10

Writability: 1/10

Ranking: 5/5

# Fortran

## Commentary

I started with a search for “Fortran examples.” At this point, a new problem presented itself; it became apparent that there are enough versions of fortran that my search wasn’t all that enlightening. I then went to the official documentation, which was somewhat helpful. I was able to quickly learn a lot of the syntax and how common things like variables and loops are implemented in Fortran. I noticed I didn’t see a section for “string methods” or “helper functions.” At this point I knew I would be writing those. I started programming a hello world on IDEone.com and got it on my first try but the compiler gave me a warning about using tabs. Did Fortran come before tabs? I wind up googling about how to get rid of it and I’m told to use "-Wtabs" as a flag when compiling. I ignore this because I’m not sure if I can control that on IDEone. My approach for creating the Caesar cipher is to make a function to handle shifting a single letter, and then carry that to the whole word in the encrypt and decrypt functions. I’m happily coding along at this point and while trying to get a single letter to work, I come across a runtime error. I wish I could tell you what it was for, but I’m just passing along the same message Fortran gave me, “Runtime Error.” This was painful to debug, I fiddled around for 15 minutes and deleted everything I changed and re-implemented it, somehow working the second time. Getting the loop to work was a pain. The syntax for accessing the index of a string is a bit strange, getting a slice of one element. After implementing the for loop, my function would successfully compute the encrypted cipher-text but there were random characters like “���▯�� �V@��V” following the result. I was able to fix this after some help from a classmate, and figuring out I can just replace the empty space at the end of a string with space characters.

## Google Searches

- fortran examples
- fortran hello world
- fortran getting started
- fortran string methods
- fortran helper functions
- fortran compiler warning tabs
- ideone compiler settings
- fortran character to int
- fortran strings
- fortran "runtime error"
- fortran index of
- fortran illegible characters end of string

## Caesar Implementation

```fortran
! compile and run with gfortran caesar.f90 && ./a.out
program Caesar
    character(1) :: shift
    character(64) :: encrypt
    character(64) :: decrypt
    print *, encrypt('ATTACK AT ONCE', 4)
    print *, decrypt('EXXEGO EX SRGI', 4)

    call solve("abcdeFGHIJKLmnopqrstuvwxyz ,?;{[()]}", 26)

    stop
end

function encrypt(str, shiftAmt) result(res)
    character(*) :: str
    character(1) :: shift
    integer :: shiftAmt
    character(*) :: res
    integer :: i
    integer :: j

    do i = 1, len_trim(str)
        res(i:i) = shift(str(i:i), shiftAmt)
    end do
    do j = len_trim(str) + 1, 64
       res(j:j) = ' '
    end do


end function

function decrypt(str, shiftAmt) result(res)
    character(*) :: str
    integer :: shiftAmt
    character(64) :: res
    character(64) :: encrypt


    res = encrypt(str, -shiftAmt)

end function

subroutine solve(str, maxShiftValue)
    implicit none
    character(*) :: str
    integer :: maxShiftValue
    character(64) :: decrypt
    integer :: i

    do i = 1, maxShiftValue
        print *, 'Shift: ', i, decrypt(str, i)
    end do

end subroutine

function toUpper(chr) result(upper)
    implicit none
    character :: chr
    character :: upper
    integer :: charcode

    charcode = iachar(chr)

    if(charcode >= 97 .and. charcode <= 122) then
        ! lowercase
        upper = achar(charcode - 32)
    else
        upper = chr
    end if

end function

function shift(chr, num) result(shifted)
    implicit none
    character :: chr
    integer :: num
    character :: shifted
    integer :: shiftedCode
    character :: toUpper

    shiftedCode = iachar(toUpper(chr))

    if(shiftedCode >= 64 .and. shiftedCode <= 90) then
        shiftedCode = modulo(shiftedCode - 65 + num, 26) + 65
    end if

    shifted = achar(shiftedCode)

    !print *, chr, ' -> ', shifted


end function
```

Output

```
EXXEGO EX SRGI
ATTACK AT ONCE
Shift:            1 ZABCDEFGHIJKLMNOPQRSTUVWXY ,?;{[()]}
Shift:            2 YZABCDEFGHIJKLMNOPQRSTUVWX ,?;{[()]}
Shift:            3 XYZABCDEFGHIJKLMNOPQRSTUVW ,?;{[()]}
Shift:            4 WXYZABCDEFGHIJKLMNOPQRSTUV ,?;{[()]}
Shift:            5 VWXYZABCDEFGHIJKLMNOPQRSTU ,?;{[()]}
Shift:            6 UVWXYZABCDEFGHIJKLMNOPQRST ,?;{[()]}
Shift:            7 TUVWXYZABCDEFGHIJKLMNOPQRS ,?;{[()]}
Shift:            8 STUVWXYZABCDEFGHIJKLMNOPQR ,?;{[()]}
Shift:            9 RSTUVWXYZABCDEFGHIJKLMNOPQ ,?;{[()]}
Shift:           10 QRSTUVWXYZABCDEFGHIJKLMNOP ,?;{[()]}
Shift:           11 PQRSTUVWXYZABCDEFGHIJKLMNO ,?;{[()]}
Shift:           12 OPQRSTUVWXYZABCDEFGHIJKLMN ,?;{[()]}
```

```
Shift:           13 NOPQRSTUVWXYZABCDEFGHIJKLM ,?;{[()]}
Shift:           14 MNOPQRSTUVWXYZABCDEFGHIJKL ,?;{[()]}
Shift:           15 LMNOPQRSTUVWXYZABCDEFGHIJK ,?;{[()]}
Shift:           16 KLMNOPQRSTUVWXYZABCDEFGHIJ ,?;{[()]}
Shift:           17 JKLMNOPQRSTUVWXYZABCDEFGHI ,?;{[()]}
Shift:           18 IJKLMNOPQRSTUVWXYZABCDEFGH ,?;{[()]}
Shift:           19 HIJKLMNOPQRSTUVWXYZABCDEFG ,?;{[()]}
Shift:           20 GHIJKLMNOPQRSTUVWXYZABCDEF ,?;{[()]}
Shift:           21 FGHIJKLMNOPQRSTUVWXYZABCDE ,?;{[()]}
Shift:           22 EFGHIJKLMNOPQRSTUVWXYZABCD ,?;{[()]}
Shift:           23 DEFGHIJKLMNOPQRSTUVWXYZABC ,?;{[()]}
Shift:           24 CDEFGHIJKLMNOPQRSTUVWXYZAB ,?;{[()]}
Shift:           25 BCDEFGHIJKLMNOPQRSTUVWXYZA ,?;{[()]}
Shift:           26 ABCDEFGHIJKLMNOPQRSTUVWXYZ ,?;{[()]}
```

## Log

Prediction: 2 hours

| Date | Hours Spent | Accomplishments |
|---|---|---|
| 1/26 | 2 | Get familiar with Fortran, Create a function to shift a single letter |
| 1/27 | 1 | Figure out how to apply it to a variable length string |
| 1/27 | 1 | Handle non alphabetic characters |
| 1/27 | 0.5 | Create “solve” subroutine |
| 2/18 | 0.5 | Fix the random characters caused by overallocating |
| 2/21 | 0.5 | Test thoroughly |

### Discrepancy of time

I think the discrepancy in time taken to complete this program is just how different this language is from most of what I've done in the past. Also, IDEone definitely added time, my development was much smoother once I removed them from the equation. This was also my first iteration through creating a caesar cipher, at least in a little while, so I had to actually debug my code to make sure my standard algorithm was working. This increasingly led to extra time with the terrible error messages given in the event of a runtime error.

## Overall Review

Fortran was tough to develop mainly because of how different some of the syntax was. The readability and writability are both not that great. One thing in particular that was annoying was having to declare the types of everything multiple times when it should be able to inferred by the context of the program. This may have increased some readability by having the types located close to the functions but I feel the writability was hurt more than that. I'm not a huge fan of the use of end in place of curly braces, they tend to blend into the rest of the program. I think programming languages made the right move going forward with the use of symbols. It's also unclear to me why the for loops don't have parentheses but the if statements do, this seems like something that should be consistent throughout he language.

## Ratings

Readability: 5/10

Writability: 4/10

Ranking: 4/5

# Pascal

## Commentary

After doing a few of these I feel like I have a good grasp of what I need from a language to make a caesar cipher. I start by finding a compiler and getting hello world to run. From there, I look up how to create a function. After that I create essentially an “echo” function, which just returns one of the parameters I give it. So far, Pascal has had very helpful online documentation and it is easy for me to find all of the resources I need. Next I get into the meat of the program, needing helper functions like toUpper, if the language has it, a way to do modulus, a looping structure, and a char to int conversion and vice versa. Once again, this info is readily available. I assemble this into the way that I know how to by now, and voila, a working program. No big problems arose during the development of this program.

## Google Searches

- Pascal getting started
- Freepascal ubuntu
- Pascal touppercase
- Pascal for loop syntax
- Pascal if syntax
- Pascal modulus
- Pascal character conversion
- Pascal check if character is a letter

## Caesar Implementation

```pascal
{compile and run with fpc -v0 caesar.pas && ./caesar}
program Hello;
uses sysutils;
function encrypt(str:string; shift:integer): string;
var i: integer;
begin
    str:=upcase(str);
    for i:=1 to length(str) do
        begin
            {test for if it is a letter, at this point all is uppercase}
            if (str[i] in ['A'..'Z']) then
                begin
                    str[i]:= chr((ord(str[i]) -65 + shift) mod 26 + 65);
                end;
        end;
    encrypt:=str;
end;

function decrypt(str:string; shift:integer): string;
var i: integer;
begin
    decrypt:=encrypt(str, 26-shift);
```

```
end;

procedure solve(str:string; maxShiftValue:integer);
var i: integer;
begin
    str:=upcase(str);
    for i:=1 to 26 do
        begin
            writeln('Shift: ', i, ' Result: ', decrypt(str, i));
        end;
end;

begin
    writeln(encrypt('ATTACK AT ONCE',4));
    writeln(decrypt('EXXEGO EX SRGI',4));
    solve('abcdeFGHIJKLmnopqrstuvwxyz ,?;{[()]}', 26)
end.
```

Output

```
EXXEGO EX SRGI
ATTACK AT ONCE
Shift: 1 Result: ZABCDEFGHIJKLMNOPQRSTUVWXY ,?;{[()]}
Shift: 2 Result: YZABCDEFGHIJKLMNOPQRSTUVWX ,?;{[()]}
Shift: 3 Result: XYZABCDEFGHIJKLMNOPQRSTUVW ,?;{[()]}
Shift: 4 Result: WXYZABCDEFGHIJKLMNOPQRSTUV ,?;{[()]}
Shift: 5 Result: VWXYZABCDEFGHIJKLMNOPQRSTU ,?;{[()]}
Shift: 6 Result: UVWXYZABCDEFGHIJKLMNOPQRST ,?;{[()]}
Shift: 7 Result: TUVWXYZABCDEFGHIJKLMNOPQRS ,?;{[()]}
Shift: 8 Result: STUVWXYZABCDEFGHIJKLMNOPQR ,?;{[()]}
Shift: 9 Result: RSTUVWXYZABCDEFGHIJKLMNOPQ ,?;{[()]}
Shift: 10 Result: QRSTUVWXYZABCDEFGHIJKLMNOP ,?;{[()]}
Shift: 11 Result: PQRSTUVWXYZABCDEFGHIJKLMNO ,?;{[()]}
Shift: 12 Result: OPQRSTUVWXYZABCDEFGHIJKLMN ,?;{[()]}
Shift: 13 Result: NOPQRSTUVWXYZABCDEFGHIJKLM ,?;{[()]}
Shift: 14 Result: MNOPQRSTUVWXYZABCDEFGHIJKL ,?;{[()]}
Shift: 15 Result: LMNOPQRSTUVWXYZABCDEFGHIJK ,?;{[()]}
Shift: 16 Result: KLMNOPQRSTUVWXYZABCDEFGHIJ ,?;{[()]}
Shift: 17 Result: JKLMNOPQRSTUVWXYZABCDEFGHI ,?;{[()]}
Shift: 18 Result: IJKLMNOPQRSTUVWXYZABCDEFGH ,?;{[()]}
Shift: 19 Result: HIJKLMNOPQRSTUVWXYZABCDEFG ,?;{[()]}
Shift: 20 Result: GHIJKLMNOPQRSTUVWXYZABCDEF ,?;{[()]}
Shift: 21 Result: FGHIJKLMNOPQRSTUVWXYZABCDE ,?;{[()]}
Shift: 22 Result: EFGHIJKLMNOPQRSTUVWXYZABCD ,?;{[()]}
Shift: 23 Result: DEFGHIJKLMNOPQRSTUVWXYZABC ,?;{[()]}
Shift: 24 Result: CDEFGHIJKLMNOPQRSTUVWXYZAB ,?;{[()]}
Shift: 25 Result: BCDEFGHIJKLMNOPQRSTUVWXYZA ,?;{[()]}
Shift: 26 Result: ABCDEFGHIJKLMNOPQRSTUVWXYZ ,?;{[()]}
```

## Log

Prediction: 2 hours

| Date | Hours Spent | Accomplishments |
|---|---|---|
| 2/7 | 0.5 | Set up dev env |
| 2/10 | 0.5 | Find toUpper, modulus, for loops, and char int conversion documentation |
| 2/14 | 1 | Implement encrypt, decrypt, solve |
| 2/21 | 0.5 | Test thoroughly |

### Discrepancy of time

This language went by really quick. Largely due to good documentation, which I am learning to be one of the most important aspect of a language. I still went a little bit over budget due to extensive testing, but nothing really arose during the testing period except for a few minor tweaks.

## Overall Review

This language is a pretty good language. It has a lot of strange syntactical choices, I think. For example, the walrus operator for equals is not a common thing in other languages and adds to the cost of using the language. I know you have mentioned that you like this decision, but on my personal machine I use font ligatures which change the == into a different symbol so this was never a problem for me. They do similar odd things like this in a few spots... the use of writeln vs println, comments, semicolons in the function typing parameters instead of commas, thing like that do not some to add anything to the language and are just being done to be different. One thing that was kind of strange to me is the need to declare variables at the function header but the compiler gives a good enough error message so that it is something I can search for, and once again quickly resolve the problem. I found that the toUpper function is located in the sysutils library, so that is something new, I haven't had to import anything in the other languages. I kind of like that extra functionality can be located in external libraries to reduce the size of the compiled program. This makes the language more orthogonal by default, while still allowing for extra convenience where it makes sense.

## Ratings

Readability: 7/10

Writability: 6/10

Ranking: 3/5