

FORTRAN

First popular higher-level programming language

50 years (about) old

Even the first compiler was very efficient

It took 15 years before another compiler beat it

Stood for FORMula TRANslation

Written by one of the 704 designers

The IBM 704 Computer

First "modern" machine - it had

Index registers

Random access (core) memory

Magnetic tape and card input and output

Line printer output

The console had programmable lights and sense switches

There was no operating system

Early importance of speed and efficiency

Early machines had small memories

Memory efficiency very important

Early machines were slow and expensive

Programmers programmed in assembly/machine language

Programmer time was not valued as much as machine time

Early programmers would not even consider a technique that was less efficient than machine language

The FORTRAN language was designed for the IBM 704

Designed for scientific calculation

It is still very good for numerical processing with multidimensional arrays

All input on cards (or tape produced by card to tape machine)

All output on cards or line printer (or tape)

Could use magnetic tape for intermediate storage

But could only read in 72 columns of 80 columns of card (hardware limitation)

Original Language Features

Constants

fixed point (15 - bit)

floating point from 10^{-38} to 10^{38}

Variables

I, J, K, L, M, or N for fixed point

all others for floating point

Could not be more than 4 characters and end in an "F"

Arrays

Variables can represent a 1, 2, or 3 dimensional array

Array indices start with one not zero

Array subscripts must be of one of the forms

V

C

V+C

V-C

C*V

C*V+C'

C*V-C'

Subscripted variables

A(I, 2*J-3)

Functions

function names are 4 to 7 characters with the last character "F"

SINEF(2.34)

(SQRTF(5.0) + 1.0) / 2.0

starts with "X" if returns a fixed point

Allowed (Mixed-Mode) Expressions

FORTRAN after 1958

**Eventually allowed independent compilation of programs,
subroutines, and functions**

linkage of subroutine libraries

Program Input format

All input was on IBM 80-column cards

col 1 "C for comment"

col 1-5 statement label

col 6 continuation

col 7-72 statement text

col 73-80 not read - for identification or sequence numbering

FORTRAN Statements

a=b [assignment statement]

GO TO n

Statement n

GO TO s, (s₁, s₂, s₃, ... , s_m)

Statement number last assigned to s

ASSIGN s TO n

GO TO (s₁, s₂, s₃, ... , s_m), i

Statement s_i

IF (a) s₁, s₂, s₃

Statement s₁, s₂, s₃ as a is less than, =, or greater than 0

PAUSE or **PAUSE n**

STOP or **STOP n**

DO s i = m₁, m₂ or **DO s i = m₁, m₂, m₃**

Iterate through statement s with i starting at m₁ and stepping by m₂ until m₃ but executing the loop at least once

CONTINUE

Used to place a statement label if necessary

DIMENSION A(20), B(10,10), C(5,6,7)

Declares variables to be arrays with the indicated sizes

EQUIVALENCE (A(3), B(2,3)), (X, C(1,1,1))

Allocate memory so that the indicated expressions are at exactly the same memory location

FORTRAN IO Statements

FORMAT (Specification)

Specify the record format for input or output

READ f, List

READ INPUT TAPE i, f, List

PUNCH f, List

PRINT f, List

WRITE OUTPUT TAPE i, f, List

Do IO under control of the format statement f using the values specified by List

READ TAPE i, List

WRITE TAPE i, List

Do binary IO

END FILE i

REWIND i

BACKSPACE i

Obsolete FORTRAN Statements

SENSE LIGHT i

IF (SENSE LIGHT i) n₁, n₂

Statement n₁, n₂ as Sense Light is ON or OFF

IF (SENSE SWITCH i) n₁, n₂

Statement n₁, n₂ as Sense Switch is DOWN or UP

IF ACCUMULATOR OVERFLOW n₁, n₂

Statement n₁, n₂ as Accumulator Overflow trigger is ON or OFF

IF QUOTIENT OVERFLOW n₁, n₂

Statement n₁, n₂ as MQ Overflow trigger is ON or OFF

IF DIVIDE CHECK n₁, n₂

Statement n₁, n₂ as Divide Check trigger is ON or OFF

READ DRUM i, j, List

WRITE DRUM i, j, List

FREQUENCY n(i, j, ...), m(k, l, ...), ...

Used by the compiler to optimize the generated code

IO List Specification

Like mini DO loops

Input/output lists (examples)

These are to be interpreted as DO loops

`A, B (3) , (C (I) , D (I, K) , I=1, 10)`

`((E (I, J) , I=1, 10, 1) , F (J, 3) , J=1, K)`

An array can be written or read without a loop as the program knows the size of the array

The FORMAT Statement

All IO is either binary or formatted

Formatted IO has rigid column positions

FORMAT statement controls formatted IO

Formatted output is fixed format - there are no variable width fields

Formatted IO is designed for "unit record" devices

Each record consists of a fixed number of characters

Records are read or written as a unit

Examples of unit record devices are

IBM card reader/punch - 80 columns

line printer - 120 columns

Fonts/proportional spacing - forget it

Magnetic tape - generally wrote and read fixed-length records - max 120 characters

FORMAT(specification)

```
FORMAT(1H ,14HTHE ANSWER IS ,I3,2E16.8)
```

The **FORMAT** specification describes the record by giving, for each field in the line:

The conversion (E, F, or I)

The width of the field

The number of decimal places to be rounded and printed (for E and F)

The basic numeric formats are

nIw

$nEw.d$

$nFw.d$

where

n is a repetition count for this field specification

w is the width of the field

d is the number of digits after the decimal point

If the field width is bigger than necessary then the result is right-justified in the field and the remainder of the field is filled with blanks

The Hollerith field

$nHx_1x_2x_3\dots x_n$

Characters $x_1x_2x_3\dots x_n$ are written on output and replaced on input

FORMAT (continued)

Additional specifications

The / character ends one record and starts another

If the specification is too short the last parenthesized group is repeated

Continuation cards may be used for long FORMAT statements

Carriage Control

The first character of a line sent to the printer has the following meaning

Blank	Single space before printing
0	Double space before printing
+	No space before printing
1-9	Skip to channels 1-9
J-R	Short skip to channels 1-9

This can be done with a 1Hx as the first format specification

Example of a Fortran program

binomial.f

```
C THIS PROGRAM CALCULATES BINOMIAL COEFFICIENTS
C
      DIMENSION NBINOM(20)
      1 FORMAT(20I4)
      DO 10 K=1,20
10    NBINOM(K) = 0
      NBINOM(1) = 1
      DO 30 K=1,20
      DO 20 J=K,2,-1
20    NBINOM(J) = NBINOM(J) + NBINOM(J-1)
30    PRINT 1, (NBINOM(I), I=1,K)
      END
```

Program Output

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1
1 10 45 120 210 252 210 120 45 10 1
1 11 55 165 330 462 462 330 165 55 11 1
1 12 66 220 495 792 924 792 495 220 66 12 1
1 13 78 286 715 1287 1716 1716 1287 715 286 78 13 1
1 14 91 364 1001 2002 3003 3432 3003 2002 1001 364 91 14 1
1 15 105 455 1365 3003 5005 6435 6435 5005 3003 1365 455 105 15 1
1 16 120 560 1820 4368 8008*****800843681820 560 120 16 1
1 17 136 680 2380 6188*****61882380 680 136 17 1
1 18 153 816 3060 8568*****85683060 816 153 18 1
1 19 171 969 3876*****3876 969 171 19 1
```

Special FORTRAN Program Properties

No memory allocation after program loading (no stack)

Each subroutine has a fixed allocation of memory

Return address for each subroutine is stored in fixed location

No recursion

All control transfers were branches or function calls

No if-then-else constructs or other modern loop constructs

This required a style of programming that provoked Dijkstra's goto letter

Statements to sense console switches, set and sense console lights, pause (halt but hitting start button would continue), and stop statements (halt and refuse to continue)

The ** operator stood for exponentiation ($A^{**}B$)

Function statements

$F(X) = X^{**}2$

FORTRAN IV or Fortran 77 Additional Features

Common areas and named common

Boolean expressions and types

Logical if

```
IF (X.LT.0) N=N+1
```

```
IF (X.LT.0) GOTO 130
```

Text arrays

More FORMAT types

E, I, F, L, A, X, G, H, /, D, ...

Subroutines and Function subprograms

Independent compilation

Entry statement

Return address was stored as data in the subroutine

Arguments passed by reference (address)

Absolute address of argument passed which allowed storing back through a formal parameter of a subroutine or function

subroutine variable value persistence

passing labels as subroutine parameters

SUBROUTINE, FUNCTION and ENTRY Statements

To enable independent compilation the language was extended with SUBROUTINE and FUNCTION compilation units in addition to normal "main" programs

```
SUBROUTINE S (X, Y, Z)  
[type] FUNCTION F (X, Y, Z)  
ENTRY G (X, Y)
```

All three of these statements specify callable entry points

FUNCTION subprograms return a value

SUBROUTINE subprograms do not return a value

ENTRY statements are alternative entry points into the subprogram

Since variables of a subprogram exist

COMMON

Declares a block of memory that is shared between compilation units

Equivalence statements can reference variables in common areas

All compilation units must declare common areas with the same label to be the same size

Blank common can be declared to be of different sizes

It will be the maximum size declared in any compilation unit

```
COMMON/STUFF/A(10), B(10), C, D, E
```

Blank COMMON

```
COMMON//A(10), B(10), C, D, E
```

or

```
COMMON A(10), B(10), C, D, E
```

BLOCK DATA

Used to initialize named common areas

Cannot be used to initialize blank COMMON

```
BLOCK DATA
```

```
COMMON/X/X(10),Y(10)
```

```
DATA X/1,2,3,4,5,6,7,8,9,10/, (Y(I), I=1,2)/10,20/
```

Matrix Multiplication

```
        SUBROUTINE MATMULT(A, B, C, L, M, N)
        DIMENSION A(L,M), B(M,N), C(L,N)
C   Calculates the matrix product  $C = A*B$  where
C   A, B, and C are matrices with the dimensions
C   A(L,M), B(M,N), and C(L,N)
C
        DO 20 I = 1, L
        DO 20 J = 1, N
        SUM = 0.0
        DO 10 K = 1, M
10    SUM = SUM + A(I, K) * B(K, J)
20    C(I, J) = SUM
        RETURN
        END
```

Statement List (1980)

ASSIGN *s* TO *i*

BACKSPACE *u*

BACKSPACE (*alist*)

BLOCK DATA [*sub*]

CALL *sub*([(a [, a]...)])

CHARACTER [**len*[,]] *nam* [, *nam*]...

CLOSE (*cclist*)

COMMON [/[*cb*]/] *nlist* [[,]/[*cb*]/ *nlist*]...

COMPLEX *v* [,*v*]...

CONTINUE

DATA *nlist/cclist*/ [[,]*nlist/cclist*/]...

DIMENSION *a*(*d*) [,*a*(*d*)]...

DO *s* [,] *i* = *e*₁, *e*₂ [,*e*₃]

DOUBLE PRECISION *v* [,*v*]...

ELSE

ELSE IF (*e*) **THEN**

END

END IF

ENDFILE *u*

ENDFILE (*alist*)

ENTRY *en* [(*d* [,*d*]...)]

EQUIVALENCE (*nlist*) [, (*nlist*)]...

EXTERNAL *proc* [, *proc*]...

FORMAT *fs*

fun(*d* [,*d*]...) = *e*

[typ] FUNCTION fun ([d [,d]...])
GO TO i [[,] (s [,s]...)]
GO TO s
GO TO (s [,s]...)[,] i
IF (e) st
IF (e) s₁, s₂, s₃
IF (e) THEN
IMPLICIT typ (a [,a]...) [typ (a [,a]...)]...
INQUIRE (iflist)
INQUIRE (iulist)
INTEGER v [,v]...
INTRINSIC fun [, fun]...
LOGICAL v [,v]...
OPEN (olist)
PARAMETER (p = e [, p = e]...)
PAUSE [n]
PRINT f [, iolist]
PROGRAM pgm
READ (cilst) [iolist]
READ f [,iolist]
REAL v [, v]...
RETURN [e]
REWIND u
REWIND (alist)
SAVE [a [,a]...]
STOP [n]

SUBROUTINE sub [(**[d** [,d]...**]**)]

v = e

WRITE (**cilist**) [**iolist**]

Types

INTEGER

REAL

DOUBLE PRECISION

COMPLEX

LOGICAL

CHARACTER [* n]

Spice circuit analyzer was written in FORTRAN originally

used list structures coded as integer arrays

**lots of labeled commons for communication of program
state**

Finally FORTRAN 90 adds

Structures

Pointers

Multiprocessing

etc.

FORTRAN Summary

FORTRAN has survived 50 years

Still the favorite of some programmers

Large subroutine libraries available and being used

Still the benchmark for numerical codes