FORTRAN: A simple procedural language

FORTRAN = FORmula TRANslator (1957)

- The first high-level programming language to become widely used.
- Developed in the 1950s by an IBM team led by John Backus.
- At the time the utility of any high-level language was open to question!

**Standards:**

John Backus

As far as we were aware, we simply made up the language as we went along. We did not regard language design as a difficult problem, merely a simple prelude to the real problem: designing a compiler which could produce efficient programs.\(^a\)


**Overview**

**Execution model**

- FORTRAN program = main program + subprograms
  - Each is *compiled separate* from all others.
  - Translated programs are linked into final executable form during loading.
- All *storage is allocated statically* before program execution begins; no run-time storage management is provided.
- Flat register machine. *No stacks, no recursion*. Memory arranged as linear array.

References:

- Chapter 10(§1) of *Programming Languages: Design and implementation* (3RD EDITION) by T. W. Pratt and M. V. Zelkowitz. Prentice Hall, 1999.
Overview

Data types

- Numeric data: Integer, real, complex, double-precision real.
- Boolean data. called logical
- Arrays. of fixed declared length
- Character strings. of fixed declared length
- Files.

Overview

Control structures

- FORTRAN 66
  Relied heavily on statement labels and \texttt{GOTO} statements.
- FORTRAN 77
  Added some modern control structures (\textit{e.g.}, conditionals).

Example

PROGRAM MAIN
  PARAMETER (MaxSiz=99)
  REAL A(MaxSiz)
  10 READ (5,100,END=999) K
  100 FORMAT(I5)
  IF (K.LE.0 .OR. K.GT.MAXSIZ) STOP
  READ *,(A(I),I=1,K)
  PRINT *,(A(I),I=1,K)
  PRINT *,'SUM=',SUM(A,K)
  GO TO 10
  999 PRINT *, "All Done"
STOP
END

C SUMMATION SUBPROGRAM
  FUNCTION SUM(V,N)
    REAL V(N)
    SUM = 0.0
    DO 20 I = 1,N
      SUM = SUM + V(I)
    20 CONTINUE
    RETURN
  END
Example
Commentary

_columns and lines are relevant.

Blanks are ignored (by early FORTRANs).

Variable names are from 1 to 6 characters long, begin with a letter, and contain letters and digits.

Programmer-defined constants.

Arrays: when sizes are given, lower bounds are assumed to be 1; otherwise subscript ranges must be explicitly declared.

Variable types may not be declared: implicit naming convention.

Data formats.

FORTRAN 77 has no `while` statement.

Functions are compiled separately from the main program. Information from the main program is not used to pass information to the compiler. Failure may arise when the loader tries to merge subprograms with main program.

Function parameters are uniformly transmitted by reference (or value-result).
Recall that allocation is done statically.

`DO` loops by increment.

A value is returned in a FORTRAN function by assigning a value to the name of a function.

On syntax

A misspelling bug ...

```fortran
  do 10 i = 1,100    vs.    do 10 i = 1.100
```

... that is reported to have caused a rocket to explode upon launch into space!

Types

FORTRAN has no mechanism for creating user types.

`Static type checking` is used in FORTRAN, but the checking is incomplete.

Many language features, including arguments in subprogram calls and the use of `COMMON` blocks, cannot be statically checked (in part because subprograms are compiled independently).

Constructs that cannot be statically checked are ordinarily left unchecked at run time in FORTRAN implementations.
Storage Representation and Management

Storage representation in FORTRAN is **sequential**.

Only two levels of referencing environment are provided, **global** and **local**.

The global environment may be partitioned into separate **common** environments that are shared amongst sets of subprograms, but only data objects may be shared in this way.

The sequential storage representation is critical in the definition of the **EQUIVALENCE** and **COMMON** declarations.

**EQUIVALENCE**

This declaration allows more than one simple or subscripted variable to refer to the same storage location.

Is this a good idea?

Consider the following:

```fortran
REAL X
INTEGER Y
EQUIVALENCE (X,Y)
```

**COMMON**

The global environment is set up in terms of **sets of variables and arrays**, which are termed **COMMON blocks**.

A **COMMON** block is a named block of storage and may contain the values of any number of simple variables and arrays.

**COMMON** blocks may be used to isolate global data to only a few subprograms needing that data.

Is the **COMMON** block a good idea?

Consider the following:

```fortran
COMMON/BLK/X,Y,K(25) in MAIN
COMMON/BLK/U,V,I(5),M(4,5) in SUB
```

**Aliasing**

Aliasing occurs when two names or expressions refer to the same object or location.

Aliasing raises serious problems for both the user and implementor of a language.

Because of the problems caused by aliasing, new language designs sometimes attempt to restrict or eliminate altogether features that allow aliases to be constructed.\(^a\)

\(^a\)In what regards to the problem of checking for aliasing, interested students may wish to investigate the work on the programming language *Euclid* and/or the work of John Reynolds and followers on *Syntactic Control of Interference*.
Parameters

There are two concepts that must be clearly distinguished.

- The parameter names used in a function declaration are called **formal parameters**.
- When a function is called, expressions called **actual parameters** are used to compute the parameter values for that call.

- The language specifies that if a formal parameter is assigned to, the actual parameter must be a variable, but because of independent compilation this rule cannot be checked by the compiler.

**Example:**

```
SUBROUTINE SUB(X,Y)
   X = Y
END

CALL SUB(-1.0,1.0)
```

- Parameter passing is uniformly by **reference**.

**FORTRAN subroutines and functions**

- **Actual parameters** may be simple variables, literals, array names, subscripted variables, subprogram names, or arithmetic or logical expressions.
  The interpretation of a **formal parameter** as an array is done by the called subroutine.
- Each subroutine is compiled independently and no checking is done for compatibility between the subroutine declaration and its call.