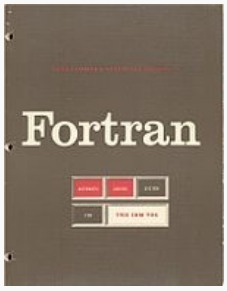
Fortran

From Wikipedia, the free encyclopedia

Fortran (previously **FORTRAN**) is a general-purpose, imperative programming language that is especially suited to numeric computation and scientific computing. Originally developed by IBM at their campus in south San Jose, California[1] in the 1950s for scientific and engineering applications, Fortran came to dominate this area of programming early on and has been in continual use for over half a century in computationally intensive areas such as numerical weather prediction, finite element analysis, computational fluid dynamics, computational physics and computational chemistry. It is one of the most popular languages in the area of high-performance computing[2] and is the language used for programs that benchmark and rank the world's fastest supercomputers.

Fortran (the name is a syllabic abbreviation derived from *The IBM Mathematical **For**mula **Tran**slating System*) encompasses a lineage of versions, each of which evolved to add extensions to the language while usually retaining compatibility with previous versions. Successive versions have added support for structured programming and processing of character-based data (FORTRAN 77), array programming, modular programming and generic programming (Fortran 90), high performance Fortran (Fortran 95), object-oriented programming (Fortran 2003) and concurrent programming (Fortran 2008).

Fortran



Paradigm(s)	multi-paradigm: structured, imperative (procedural, object-oriented), generic
Appeared in	1957
Designed by	John Backus
Developer	John Backus & IBM
Stable release	Fortran 2008 (ISO/IEC 1539-1:2010) (2010)
Typing discipline	strong, static, manifest
Major implementations	Absoft, Cray, GFortran, G95, IBM, Intel, Lahey/Fujitsu, Open Watcom, PathScale, PGI, Silverfrost, Oracle, XL Fortran, Visual Fortran, others
Influenced by	Speedcoding
Influenced	ALGOL 58, BASIC, C, PL/I, PACT I, MUMPS, Ratfor
Usual filename extensions	`.f`, `.for`, `.f90`, `.f95`

Contents

- 1 Capitalization
- 2 History
  - 2.1 FORTRAN
    - 2.1.1 Fixed layout
  - 2.2 FORTRAN II
    - 2.2.1 Simple FORTRAN II program
  - 2.3 FORTRAN III
  - 2.4 IBM 1401 FORTRAN
  - 2.5 FORTRAN IV
  - 2.6 FORTRAN 66
  - 2.7 FORTRAN 77
    - 2.7.1 Variants: Minnesota FORTRAN
  - 2.8 Transition to ANSI Standard Fortran
  - 2.9 Fortran 90
    - 2.9.1 Obsolescence and deletions
    - 2.9.2 "Hello world" example
  - 2.10 Fortran 95
    - 2.10.1 Conditional compilation and varying length strings
  - 2.11 Fortran 2003
  - 2.12 Fortran 2008
  - 2.13 Fortran 2015
- 3 Fortran and supercomputers
- 4 Language features
- 5 Portability
- 6 Variants
  - 6.1 Fortran 5
  - 6.2 Fortran V
  - 6.3 Fortran 6
  - 6.4 Specific variants
    - 6.4.1 FOR TRANSIT for the IBM 650
  - 6.5 Fortran-based languages
- 7 Code examples
- 8 Humor
- 9 See also
- 10 References
- 11 Further reading
- 12 External links

Capitalization

The names of earlier versions of the language through FORTRAN 77 were conventionally spelled in all-caps (FORTRAN 77 was the version in which the use of lowercase letters in keywords was strictly nonstandard). The capitalization has been dropped in referring to newer versions beginning with Fortran 90. The official language standards now refer to the language as "Fortran". Because the capitalization was never completely consistent in actual usage, this article adopts the convention of using the all-caps *FORTRAN* in referring to versions of the language up to FORTRAN 77 and the title-caps *Fortran* in referring to versions of the language from Fortran 90 onward. This convention is reflected in the capitalization of *FORTRAN* in the ANSI X3.9-1966 (FORTRAN 66) and ANSI X3.9-1978 (FORTRAN 77) standards and the title caps *Fortran* in the ANSI X3.198-1992 (Fortran 90), ISO/IEC 1539-1:1997 (Fortran 95) and ISO/IEC 1539-1:2004 (Fortran 2003) standards.

History

In late 1953, John W. Backus submitted a proposal to his superiors at IBM to develop a more practical alternative to assembly language for programming their IBM 704 mainframe computer. Backus' historic FORTRAN team consisted of programmers Richard Goldberg, Sheldon F. Best, Harlan Herrick, Peter Sheridan, Roy Nutt, Robert Nelson, Irving Ziller, Lois Haibt, and David Sayre.[3] Its concepts included easier entry of equations into a computer, an idea developed by J. Halcombe Laning and demonstrated in his GEORGE compiler of 1952.[4]


An IBM 704 mainframe

A draft specification for *The IBM Mathematical Formula Translating System* was completed by mid-1954. The first manual for FORTRAN appeared in October 1956, with the first FORTRAN compiler delivered in April 1957. This was the first optimizing compiler, because customers were reluctant to use a high-level programming language unless its compiler could generate code whose performance was comparable to that of hand-coded assembly language.[5]

While the community was skeptical that this new method could possibly outperform hand-coding, it reduced the number of programming statements necessary to operate a machine by a factor of 20, and quickly gained acceptance. John Backus said during a 1979 interview with *Think*, the IBM employee magazine, "Much of my work has come from being lazy. I didn't like writing programs, and so, when I was working on the IBM 701, writing programs for computing missile trajectories, I started work on a programming system to make it easier to write programs."[6]

The language was widely adopted by scientists for writing numerically intensive programs, which encouraged compiler writers to produce compilers that could generate faster and more efficient code. The inclusion of a complex number data type in the language made Fortran especially suited to technical applications such as electrical engineering.

By 1960, versions of FORTRAN were available for the IBM 709, 650, 1620, and 7090 computers. Significantly, the increasing popularity of FORTRAN spurred competing computer manufacturers to provide FORTRAN compilers for their machines, so that by 1963 over 40 FORTRAN compilers existed. For these reasons, FORTRAN is considered to be the first widely used programming language supported across a variety of computer architectures.

The development of FORTRAN paralleled the early evolution of compiler technology, and many advances in the theory and design of compilers were specifically motivated by the need to generate efficient code for FORTRAN programs.

FORTRAN

The initial release of FORTRAN for the IBM 704 contained 32 statements, including:

- `DIMENSION` and `EQUIVALENCE` statements
- Assignment statements
- Three-way *arithmetic* `IF` statement, which passed control to one of three locations in the program depending on whether or not the result of the arithmetic statement was negative, zero, or positive
- `IF` statements for checking exceptions (`ACCUMULATOR OVERFLOW`, `QUOTIENT OVERFLOW`, and `DIVIDE CHECK`); and `IF` statements for manipulating sense switches and sense lights
- `GOTO`, computed `GOTO`, `ASSIGN`, and assigned `GOTO`
- `DO` loops
- Formatted I/O: `FORMAT`, `READ`, `READ INPUT TAPE`, `WRITE`, `WRITE OUTPUT TAPE`, `PRINT`, and `PUNCH`
- Unformatted I/O: `READ TAPE`, `READ DRUM`, `WRITE TAPE`, and `WRITE DRUM`
- Other I/O: `END FILE`, `REWIND`, and `BACKSPACE`
- `PAUSE`, `STOP`, and `CONTINUE`
- `FREQUENCY` statement (for providing optimization hints to the compiler).

The arithmetic `IF` statement was similar to a three-way branch instruction on the IBM 704. However, the 704 branch instructions all contained only one destination address (e.g., TZE — Transfer AC Zero, TNZ — Transfer AC Not Zero, TPL — Transfer AC Plus, TMI — Transfer AC Minus). The machine (and its successors in the 700/7000 series) did have a three-way *skip* instruction (CAS — Compare AC with Storage), but using this instruction to implement the `IF` would consume 4 instruction words, require the constant Zero in a word of storage, and take 3 machine cycles to execute; using the Transfer instructions to implement the `IF` could be done in 1 to 3 instruction words, required no constants in storage, and take 1 to 3 machine cycles to execute. An optimizing compiler like FORTRAN would most likely select the more compact and usually faster Transfers instead of the Compare (use of Transfers also allowed the `FREQUENCY` statement to optimize `IF`s, which could not be done using the Compare). Also the Compare considered −0 and +0 to be different values while the Transfer Zero and Transfer Not Zero considered them to be the same.

The `FREQUENCY` statement in FORTRAN was used originally (and optionally) to give branch probabilities for the three branch cases of the arithmetic IF statement. The first FORTRAN compiler used this weighting to perform *at compile time* a Monte Carlo simulation of the generated code, the results of which were used to optimize the placement of basic blocks in memory — a very sophisticated optimization for its time. The Monte Carlo technique is documented in Backus et al.'s paper on this original implementation, "The FORTRAN Automatic Coding System":

> The fundamental unit of program is the basic block; a basic block is a stretch of program which has a single entry point and a single exit point. The purpose of section 4 is to prepare for section 5 a table of predecessors (PRED table) which enumerates the basic blocks and lists for every basic block each of the basic blocks which can be its immediate predecessor in flow, together with the absolute frequency of each such basic block link. This table is obtained by an actual "execution" of the program in Monte-Carlo fashion, in which the outcome of conditional transfers arising out of IF-type statements and computed GO TO'S is determined by a random number generator suitably weighted according to whatever FREQUENCY statements have been provided.[7]

Many years later, the `FREQUENCY` statement had no effect on the code, and was treated as a comment statement, since the compilers no longer did this kind of compile-time simulation. A similar fate has befallen "compiler hints" in several other programming languages; for example C's `register` keyword.[*citation needed*]

Fixed layout

Before the development of disk files, text editors and terminals, programs were most often entered on a keypunch keyboard onto 80 column punched cards, one line to a card. The resulting deck of cards would be fed into a card reader to be compiled. *See Computer programming in the punched card era.*

FORTRAN code on a punched card, showing the specialized uses of columns 1-5, 6 and 73-80.

Originally Fortran programs were written in a fixed column format. A letter "C" in column 1 caused the entire card to be treated as a comment and ignored by the compiler. Otherwise, the card was divided into four fields. Columns 1 to 5 were the label field: a sequence of digits here was taken as a label for the purpose of a GOTO or a FORMAT reference in a WRITE or READ statement. Column 6 was a continuation field: a non-blank character here caused the card to be taken as a continuation of the statement on the previous card. Columns 7 to 72 served as the statement field. Columns 73 to 80 were ignored, so they could be used for identification information. One such use was punching a sequence number which could be used to re-order cards if a stack of cards was dropped, though in practice this was reserved for stable, production programs. An IBM 519 could be used to copy a program deck and add sequence numbers. Some early compilers, e.g. the IBM 650's, had additional restrictions.[8] Later compilers relaxed most fixed format restrictions.

Within the statement field, blanks were generally ignored, allowing the programmer to omit space between tokens for brevity, or include spaces within identifiers for clarity (for example, `AVG OF X` was a valid identifier, and equivalent to `AVGOFX`).

FORTRAN II

IBM's *FORTRAN II* appeared in 1958. The main enhancement was to support procedural programming by allowing user-written subroutines and functions which returned values, with parameters passed by reference. The COMMON statement provided a way for subroutines to access common (or global) variables. Six new statements were introduced:

- `SUBROUTINE`, `FUNCTION`, and `END`
- `CALL` and `RETURN`
- `COMMON`

Over the next few years, FORTRAN II would also add support for the `DOUBLE PRECISION` and `COMPLEX` data types.

Early FORTRAN compilers did not support recursion in subroutines. Early computer architectures did not support the concept of a stack, and when they did directly support subroutine calls, the return location was often stored in a single fixed location adjacent to the subroutine code, which does not permit a subroutine to be called again before a previous call of the subroutine has returned. Although not specified in Fortran 77, many F77 compilers supported recursion as an option, while it became a standard in Fortran 90.[9]

Simple FORTRAN II program

This program, for Heron's formula, reads one data card containing three 5-digit integers A, B, and C as input. If A, B, and C cannot represent the sides of a triangle in plane geometry, then the program's execution will end with an error code of "STOP 1". Otherwise, an output line will be printed showing the input values for A, B, and C, followed by the computed AREA of the triangle as a floating-point number with 2 digits after the decimal point.

```
C AREA OF A TRIANGLE WITH A STANDARD SQUARE ROOT FUNCTION
C INPUT - CARD READER UNIT 5, INTEGER INPUT
C OUTPUT - LINE PRINTER UNIT 6, REAL OUTPUT
C INPUT ERROR DISPLAY ERROR OUTPUT CODE 1 IN JOB CONTROL LISTING
      READ INPUT TAPE 5, 501, IA, IB, IC
  501 FORMAT (3I5)
C IA, IB, AND IC MAY NOT BE NEGATIVE
C FURTHERMORE, THE SUM OF TWO SIDES OF A TRIANGLE
C IS GREATER THAN THE THIRD SIDE, SO WE CHECK FOR THAT, TOO
      IF (IA) 777, 777, 701
  701 IF (IB) 777, 777, 702
  702 IF (IC) 777, 777, 703
  703 IF (IA+IB-IC) 777,777,704
  704 IF (IA+IC-IB) 777,777,705
  705 IF (IB+IC-IA) 777,777,799
  777 STOP 1
C USING HERON'S FORMULA WE CALCULATE THE
C AREA OF THE TRIANGLE
  799 S = FLOATF (IA + IB + IC) / 2.0
      AREA = SQRT( S * (S - FLOATF(IA)) * (S - FLOATF(IB)) *
     +     (S - FLOATF(IC)))
      WRITE OUTPUT TAPE 6, 601, IA, IB, IC, AREA
  601 FORMAT (4H A= ,I5,5H  B= ,I5,5H  C= ,I5,8H  AREA= ,F10.2,
     +        13H SQUARE UNITS)
      STOP
      END
```

FORTRAN III

IBM also developed a *FORTRAN III* in 1958 that allowed for inline assembly code among other features; however, this version was never released as a product. Like the 704 FORTRAN and FORTRAN II, FORTRAN III included machine-dependent features that made code written in it unportable from machine to machine. Early versions of FORTRAN provided by other vendors suffered from the same disadvantage.

IBM 1401 FORTRAN

FORTRAN was provided for the IBM 1401 by an innovative 63-pass compiler that ran in only 8k of core. It kept the program in memory and loaded overlays that gradually transformed it, in place, into executable form, as described by Haines et al.[10] The executable form was not machine language; rather it was interpreted, anticipating UCSD Pascal P-code by two decades.

FORTRAN IV

Starting in 1961, as a result of customer demands, IBM began development of a *FORTRAN IV* that removed the machine-dependent features of FORTRAN II (such as `READ INPUT TAPE`), while adding new features such as a `LOGICAL` data type, logical Boolean expressions and the *logical IF*

statement as an alternative to the *arithmetic IF statement.* FORTRAN IV was eventually released in 1962, first for the IBM 7030 ("Stretch") computer, followed by versions for the IBM 7090 and IBM 7094.

By 1965, FORTRAN IV was supposed to be compliant with the "standard" being developed by the American Standards Association X3.4.3 FORTRAN Working Group.[11]

At about this time FORTRAN IV had started to become an important educational tool and implementations such as Waterloo University's WATFOR and WATFIV were created to simplify the complex compile and link processes of earlier compilers.

A FORTRAN coding form, printed on paper and intended to be used by programmers to prepare programs for punching onto cards by keypunch operators. Now obsolete.

FORTRAN 66

Perhaps the most significant development in the early history of FORTRAN was the decision by the *American Standards Association* (now ANSI) to form a committee sponsored by BEMA, the Business Equipment Manufacturers Association, to develop an "American Standard Fortran." The resulting two standards, approved in March 1966, defined two languages, *FORTRAN* (based on FORTRAN IV, which had served as a *de facto* standard), and *Basic FORTRAN* (based on FORTRAN II, but stripped of its machine-dependent features). The FORTRAN defined by the first standard, officially denoted X3.9-1966, became known as *FORTRAN 66* (although many continued to refer to it as FORTRAN IV, the language upon which the standard was largely based). FORTRAN 66 effectively became the first "industry-standard" version of FORTRAN. FORTRAN 66 included:

- Main program, `SUBROUTINE`, `FUNCTION`, and `BLOCK DATA` program units
- `INTEGER`, `REAL`, `DOUBLE PRECISION`, `COMPLEX`, and `LOGICAL` data types
- `COMMON`, `DIMENSION`, and `EQUIVALENCE` statements
- `DATA` statement for specifying initial values
- Intrinsic and `EXTERNAL` (*e.g.,* library) functions
- Assignment statement
- `GOTO`, assigned `GOTO`, and computed `GOTO` statements
- Logical `IF` and arithmetic (three-way) `IF` statements
- `DO` loops
- `READ`, `WRITE`, `BACKSPACE`, `REWIND`, and `ENDFILE` statements for sequential I/O
- `FORMAT` statement
- `CALL`, `RETURN`, `PAUSE`, and `STOP` statements
- Hollerith constants in `DATA` and `FORMAT` statements, and as actual arguments to procedures
- Identifiers of up to six characters in length
- Comment lines

FORTRAN 77

After the release of the FORTRAN 66 standard, compiler vendors introduced a number of extensions to "Standard Fortran", prompting ANSI committee X3J3 in 1969 to begin work on revising the 1966 standard, under sponsorship of CBEMA, the Computer Business Equipment Manufacturers Association (formerly BEMA). Final drafts of this revised standard circulated in 1977, leading to formal approval of the new FORTRAN standard in April 1978. The new standard, known as *FORTRAN 77* and officially denoted X3.9-1978, added a number of significant features to address many of the shortcomings of FORTRAN 66:

- Block `IF` and `END IF` statements, with optional `ELSE` and `ELSE IF` clauses, to provide improved language support for structured programming
- DO loop extensions, including parameter expressions, negative increments, and zero trip counts
- `OPEN`, `CLOSE`, and `INQUIRE` statements for improved I/O capability
- Direct-access file I/O
- `IMPLICIT` statement
- `CHARACTER` data type, with vastly expanded facilities for character input and output and processing of character-based data
- `PARAMETER` statement for specifying constants
- `SAVE` statement for persistent local variables
- Generic names for intrinsic functions
- A set of intrinsics (`LGE, LGT, LLE, LLT`) for lexical comparison of strings, based upon the ASCII collating sequence. (These ASCII functions were demanded by the U.S. Department of Defense, in their conditional approval vote.[*citation needed*])

In this revision of the standard, a number of features were removed or altered in a manner that might invalidate previously standard-conforming programs. *(Removal was the only allowable alternative to X3J3 at that time, since the concept of "deprecation" was not yet available for ANSI standards.)* While most of the 24 items in the conflict list (see Appendix A2 of X3.9-1978) addressed loopholes or pathological cases permitted by the previous standard but rarely used, a small number of specific capabilities were deliberately removed, such as:

- Hollerith constants and Hollerith data, such as:

```
GREET = 12HHELLO THERE!
```

- Reading into an H edit (Hollerith field) descriptor in a FORMAT specification.
- Overindexing of array bounds by subscripts.

```
DIMENSION A(10,5)
Y= A(11,1)
```

- Transfer of control out of and back into the range of a DO loop (also known as "Extended Range").

Variants: Minnesota FORTRAN

Control Data Corporation computers had another version of FORTRAN 77, called Minnesota FORTRAN (MNF), designed especially for student use, with variations in output constructs, special uses of COMMONs and DATA statements, optimizations code levels for compiling, and detailed error listings, extensive warning messages, and debugs.[12]

Transition to ANSI Standard Fortran

The development of a revised standard to succeed FORTRAN 77 would be repeatedly delayed as the standardization process struggled to keep up with rapid changes in computing and programming practice. In the meantime, as the "Standard FORTRAN" for nearly fifteen years, FORTRAN 77 would become the historically most important dialect.

An important practical extension to FORTRAN 77 was the release of MIL-STD-1753 in 1978.[13] This specification, developed by the U.S. Department of Defense, standardized a number of features implemented by most FORTRAN 77 compilers but not included in the ANSI FORTRAN 77 standard. These features would eventually be incorporated into the Fortran 90 standard.

- `DO WHILE` and `END DO` statements
- `INCLUDE` statement
- `IMPLICIT NONE` variant of the `IMPLICIT` statement
- Bit manipulation intrinsic functions, based on similar functions included in Industrial Real-Time Fortran (ANSI/ISA S61.1 (1976))

The IEEE 1003.9 POSIX Standard, released in 1991, provided a simple means for FORTRAN 77 programmers to issue POSIX system calls.[14] Over 100 calls were defined in the document — allowing access to POSIX-compatible process control, signal handling, file system control, device control, procedure pointing, and stream I/O in a portable manner.

Fortran 90

The much delayed successor to FORTRAN 77, informally known as *Fortran 90* (and prior to that, *Fortran 8X*), was finally released as ISO/IEC standard 1539:1991 in 1991 and an ANSI Standard in 1992. In addition to changing the official spelling from FORTRAN to Fortran, this major revision added many new features to reflect the significant changes in programming practice that had evolved since the 1978 standard:

- Free-form source input, also with lowercase Fortran keywords
- Identifiers up to 31 characters in length
- Inline comments
- Ability to operate on arrays (or array sections) as a whole, thus greatly simplifying math and engineering computations.
  - whole, partial and masked array assignment statements and array expressions, such as `X(1:N)=R(1:N)*COS(A(1:N))`
  - `WHERE` statement for selective array assignment
  - array-valued constants and expressions,
  - user-defined array-valued functions and array constructors.
- `RECURSIVE` procedures
- Modules, to group related procedures and data together, and make them available to other program units, including the capability to limit the accessibility to only specific parts of the module.
- A vastly improved argument-passing mechanism, allowing interfaces to be checked at compile time
- User-written interfaces for generic procedures
- Operator overloading
- Derived (structured) data types
- New data type declaration syntax, to specify the data type and other attributes of variables
- Dynamic memory allocation by means of the `ALLOCATABLE` attribute and the `ALLOCATE` and `DEALLOCATE` statements
- `POINTER` attribute, pointer assignment, and `NULLIFY` statement to facilitate the creation and manipulation of dynamic data structures
- Structured looping constructs, with an `END DO` statement for loop termination, and `EXIT` and `CYCLE` statements for terminating normal `DO` loop iterations in an orderly way
- `SELECT . . . CASE` construct for multi-way selection
- Portable specification of numerical precision under the user's control
- New and enhanced intrinsic procedures.

Obsolescence and deletions

Unlike the previous revision, Fortran 90 did not delete any features. *(Appendix B.1 says, "The list of deleted features in this standard is empty.")* Any standard-conforming FORTRAN 77 program is also standard-conforming under Fortran 90, and either standard should be usable to define its behavior.

A small set of features were identified as "obsolescent" and expected to be removed in a future standard.

Obsolescent feature	Example	Status / Fate in Fortran 95
Arithmetic IF-statement	`IF (X) 10, 20, 30`	
Non-integer DO parameters or control variables	`DO 9 X= 1.7, 1.6, -0.1`	Deleted

Shared DO-loop termination or termination with a statement other than END DO or CONTINUE	`DO 9 J= 1, 10`
	`DO 9 K= 1, 10`
	`9 L= J + K`	

Branching to END IF from outside a block	`66 GO TO 77 ; . . .`
	`IF (E) THEN ; . . .`
	`77 END IF`
		Deleted

Alternate return	`CALL SUBR( X, Y *100, *200 )`	
PAUSE statement	`PAUSE 600`	Deleted

ASSIGN statement and assigned GO TO statement	`100 . . .`
	`ASSIGN 100 TO H`
	`. . .`
	`GO TO H . . .`
		Deleted

Assigned FORMAT specifiers	`ASSIGN F TO 606`	Deleted
H edit descriptors	`606 FORMAT ( 9H1GOODBYE. )`	Deleted
Computed GO TO statement	`GO TO (10, 20, 30, 40), index`	(obsolete)
Statement functions	`FOIL( X, Y )= X**2 + 2*X*Y + Y**2`	(obsolete)
DATA statements among executable statements	`X= 27.3` `DATA A, B, C / 5.0, 12.0. 13.0 / . . .`	(obsolete)
CHARACTER* form of CHARACTER declaration	`CHARACTER*8 STRING ! Use CHARACTER(8)`	(obsolete)
Assumed character length functions	`CHARACTER*(*) STRING`	(obsolete)[15]
Fixed form source code	Column 1 contains *, ! or C for comments. Column 6 for continuation.	

"Hello world" example

```
program helloworld
    print *, "Hello, world!"
end program helloworld
```

Fortran 95

Main article: Fortran 95 language features

Fortran 95, published officially as ISO/IEC 1539-1:1997, was a minor revision, mostly to resolve some outstanding issues from the Fortran 90 standard. Nevertheless, Fortran 95 also added a number of extensions, notably from the High Performance Fortran specification:

- `FORALL` and nested `WHERE` constructs to aid vectorization
- User-defined `PURE` and `ELEMENTAL` procedures
- Default initialization of derived type components, including pointer initialization
- Expanded the ability to use initialization expressions for data objects
- Clearly defined that `ALLOCATABLE` arrays are automatically deallocated when they go out of scope.

A number of intrinsic functions were extended (for example a `dim` argument was added to the `maxloc` intrinsic).

Several features noted in Fortran 90 to be "obsolescent" were removed from Fortran 95:

- `DO` statements using `REAL` and `DOUBLE PRECISION` variables
- Branching to an `END IF` statement from outside its block
- `PAUSE` statement
- `ASSIGN` and assigned `GOTO` statement, and assigned format specifiers
- `H` edit descriptor.

An important supplement to Fortran 95 was the ISO technical report *TR-15581: Enhanced Data Type Facilities*, informally known as the *Allocatable TR*. This specification defined enhanced use of `ALLOCATABLE` arrays, prior to the availability of fully Fortran 2003-compliant Fortran compilers. Such uses include `ALLOCATABLE` arrays as derived type components, in procedure dummy argument lists, and as function return values. (`ALLOCATABLE` arrays are preferable to `POINTER`-based arrays because `ALLOCATABLE` arrays are guaranteed by Fortran 95 to be deallocated automatically when they go out of scope, eliminating the possibility of memory leakage. In addition, aliasing is not an issue for optimization of array references, allowing compilers to generate faster

code than in the case of pointers.)

Another important supplement to Fortran 95 was the ISO technical report *TR-15580: Floating-point exception handling*, informally known as the *IEEE TR*. This specification defined support for IEEE floating-point arithmetic and floating point exception handling.

Conditional compilation and varying length strings

In addition to the mandatory "Base language" (defined in ISO/IEC 1539-1 : 1997), the Fortran 95 language also includes two optional modules:

- Varying character strings (ISO/IEC 1539-2 : 2000)
- Conditional compilation (ISO/IEC 1539-3 : 1998)

which, together, compose the multi-part International Standard (ISO/IEC 1539).

According to the standards developers, "the optional parts describe self-contained features which have been requested by a substantial body of users and/or implementors, but which are not deemed to be of sufficient generality for them to be required in all standard-conforming Fortran compilers." Nevertheless, if a standard-conforming Fortran does provide such options, then they "must be provided in accordance with the description of those facilities in the appropriate Part of the Standard."

Fortran 2003

Fortran 2003, officially published as ISO/IEC 1539-1:2004, is a major revision introducing many new features. A comprehensive summary of the new features of Fortran 2003 is available at the Fortran Working Group (ISO/IEC JTC1/SC22/WG5) official Web site.[16]

From that article, the major enhancements for this revision include:

- Derived type enhancements: parameterized derived types, improved control of accessibility, improved structure constructors, and finalizers.
- Object-oriented programming support: type extension and inheritance, polymorphism, dynamic type allocation, and type-bound procedures, providing complete support for abstract data types.
- Data manipulation enhancements: allocatable components (incorporating TR 15581), deferred type parameters, `VOLATILE` attribute, explicit type specification in array constructors and allocate statements, pointer enhancements, extended initialization expressions, and enhanced intrinsic procedures.
- Input/output enhancements: asynchronous transfer, stream access, user specified transfer operations for derived types, user specified control of rounding during format conversions, named constants for preconnected units, the `FLUSH` statement, regularization of keywords, and access to error messages.
- Procedure pointers.
- Support for IEEE floating-point arithmetic and floating point exception handling (incorporating TR 15580).
- Interoperability with the C programming language.
- Support for international usage: access to ISO 10646 4-byte characters and choice of decimal or comma in numeric formatted input/output.
- Enhanced integration with the host operating system: access to command line arguments, environment variables, and processor error messages.

An important supplement to Fortran 2003 was the ISO technical report *TR-19767: Enhanced module facilities in Fortran.* This report provided *submodules,* which make Fortran modules more similar to Modula-2 modules. They are similar to Ada private child subunits. This allows the specification and implementation of a module to be expressed in separate program units, which improves packaging of large libraries, allows preservation of trade secrets while publishing definitive interfaces, and prevents compilation cascades.

Fortran 2008

The most recent standard, ISO/IEC 1539-1:2010, informally known as Fortran 2008, was approved in September 2010.[17] As with Fortran 95, this is a minor upgrade, incorporating clarifications and corrections to Fortran 2003, as well as introducing a select few new capabilities. The new capabilities include:

- Submodules – Additional structuring facilities for modules; supersedes ISO/IEC TR 19767:2005
- Coarray Fortran – a parallel execution model
- The DO CONCURRENT construct – for loop iterations with no interdependencies
- The CONTIGUOUS attribute – to specify storage layout restrictions
- The BLOCK construct – can contain declarations of objects with construct scope
- Recursive allocatable components – as an alternative to recursive pointers in derived types

The Final Draft international Standard (FDIS) is available as document N1830.[18]

An important supplement to Fortran 2008 is the ISO Technical Specification (TS) 29113 on *Further Interoperability of Fortran with C*,[19][20] which has been submitted to ISO in May 2012 for approval. The specification adds support for accessing the array descriptor from C and allows to ignore the type and rank of arguments.

Fortran 2015

The next revision of the language (Fortran 2015) is intended to be a minor revision. It is currently planned to include further interoperability between Fortran and C, additional parallel features, and "the removal of simple deficiencies in and discrepancies between existing facilities."[21]

Fortran and supercomputers

Since Fortran has been in use for more than fifty years, there is a vast body of Fortran in daily use throughout the scientific and engineering communities.[*citation needed*] It is the primary language for some of the most intensive supercomputing tasks, such as astronomy, weather and climate modeling, numerical linear algebra LAPACK, numerical libraries IMSL, structural engineering, hydrological modeling, optimization, satellite simulators,

computational fluid dynamics, computational chemistry, computational economics, animal breeding, plant breeding and computational physics. Even today, half a century later, many of the floating-point benchmarks to gauge the performance of new computer processors are still written in Fortran (*e.g.,* CFP2006 (http://www.spec.org/cpu2006/CFP2006/), the floating-point component of the SPEC CPU2006 (http://www.spec.org/cpu2006/) benchmarks).

Language features

Main article: Fortran language features

Portability

Portability was a problem in the early days because there was no agreed standard—not even IBM's reference manual—and computer companies vied to differentiate their offerings from others by providing incompatible features. Standards have improved portability. The 1966 standard provided a reference syntax and semantics, but vendors continued to provide incompatible extensions. Although careful programmers were coming to realize that use of incompatible extensions caused expensive portability problems, and were therefore using programs such as *The PFORT Verifier,* it was not until after the 1977 standard, when the National Bureau of Standards (now NIST) published *FIPS PUB 69*, that processors purchased by the U.S. Government were required to diagnose extensions of the standard. Rather than offer two processors, essentially every compiler eventually had at least an option to diagnose extensions.

Incompatible extensions were not the only portability problem. For numerical calculations, it is important to take account of the characteristics of the arithmetic. This was addressed by Fox et al. in the context of the 1966 standard by the *PORT* library. The ideas therein became widely used, and were eventually incorporated into the 1990 standard by way of intrinsic inquiry functions. The widespread (now almost universal) adoption of the IEEE 754 standard for binary floating-point arithmetic has essentially removed this problem.

Access to the computing environment (e.g. the program's command line, environment variables, textual explanation of error conditions) remained a problem until it was addressed by the 2003 standard.

Large collections of "library" software that could be described as being loosely related to engineering and scientific calculations, such as graphics libraries, have been written in C, and therefore access to them presented a portability problem. This has been addressed by incorporation of C interoperability into the 2003 standard.

It is now possible (and relatively easy) to write an entirely portable program in Fortran, even without recourse to a preprocessor.

Variants

Fortran 5

Fortran 5 was a programming language marketed by Data General Corp in the late 1970s and early 1980s, for the Nova, Eclipse, and MV line of computers. It had an optimizing compiler that was quite good for minicomputers of its time. The language most closely resembles Fortran 66. The name is a pun on the earlier Fortran IV.

Fortran V

Fortran V was a programming language distributed by Control Data Corporation in 1968 for the CDC 6600 series. The language was based upon Fortran IV.[22]

Univac also offered a compiler for the 1100 series known as Fortran V. A spinoff of Univac Fortran V was Athena Fortran.

Fortran 6

Fortran 6 or Visual Fortran 2001 was licensed to Compaq by Microsoft. They have licensed Compaq Visual Fortran and have provided the Visual Studio 5 environment interface for Compaq v6 up to v6.1.[23]

Specific variants

Vendors of high-performance scientific computers (*e.g.,* Burroughs, CDC, Cray, Honeywell, IBM, Texas Instruments, and UNIVAC) added extensions to Fortran to take advantage of special hardware features such as instruction cache, CPU pipelines, and vector arrays. For example, one of IBM's FORTRAN compilers (*H Extended IUP*) had a level of optimization which reordered the machine code instructions to keep multiple internal arithmetic units busy simultaneously. Another example is *CFD*, a special variant of Fortran designed specifically for the ILLIAC IV supercomputer, running at NASA's Ames Research Center. IBM Research Labs also developed an extended FORTRAN-based language called "VECTRAN" for processing of vectors and matrices.

Object-Oriented Fortran was an object-oriented extension of Fortran, in which data items can be grouped into objects, which can be instantiated and executed in parallel. It was available for Sun, Iris, iPSC, and nCUBE, but is no longer supported.

Such machine-specific extensions have either disappeared over time or have had elements incorporated into the main standards; the major remaining extension is OpenMP, which is a cross-platform extension for shared memory programming. One new extension, Coarray Fortran, is intended to support parallel programming.

FOR TRANSIT for the IBM 650

"FOR TRANSIT" was the name of a reduced version of the IBM 704 FORTRAN language, which was implemented for the IBM 650, using a translator program developed at Carnegie [24] in the late 1950s. The following comment appears in the IBM Reference Manual ("FOR TRANSIT Automatic Coding System" C28-4038, Copyright 1957, 1959 by IBM):

> The FORTRAN system was designed for a more complex machine than the 650, and consequently some of the 32 statements found in the FORTRAN Programmer's Reference Manual are not acceptable to the FOR TRANSIT system. In addition, certain restrictions to the FORTRAN language have been added. However, none of these restrictions make a source program written for FOR TRANSIT incompatible with the FORTRAN system for the 704.

The permissible statements were:

> Arithmetic assignment statements, e.g. a = b
> GO to n
> GO TO (n_1, n_2, ..., n_m), i
> IF (a) n_1, n_2, n_3
> PAUSE
> STOP
> DO n i = m1, m2
> CONTINUE
> END
> READ n, list
> PUNCH n, list
> DIMENSION V, V, V, ...
> EQUIVALENCE (a,b,c), (d,c), ...

Up to ten subroutines could be used in one program.

FOR TRANSIT statements were limited to columns 7 thru 56, only. Punched cards were used for input and output on the IBM 650. Three passes were required to translate source code to the "IT" language, then to compile the IT statements into SOAP assembly language, and finally to produce the object program, which could then be loaded into the machine to run the program (using punched cards for data input, and outputting results onto punched cards).

Two versions existed for the 650s with a 2000 word memory drum: FOR TRANSIT I (S) and FOR TRANSIT II, the latter for machines equipped with indexing registers and automatic floating point decimal (bi-quinary) arithmetic. Appendix A of the manual included wiring diagrams for the IBM 533 card reader/punch control panel.

Fortran-based languages

Prior to FORTRAN 77, a number of preprocessors were commonly used to provide a friendlier language, with the advantage that the preprocessed code could be compiled on any machine with a standard FORTRAN compiler. These preprocesors would typically support structured programming, variable names longer than six characters, additional data types, conditional compilation, and even macro capabilities. Popular preprocessors included FLECS, iftran, MORTRAN, SFtran, S-Fortran, Ratfor, and Ratfiv. Ratfor and Ratfiv, for example, implemented a C-like language, outputting preprocessed code in standard FORTRAN 66. Despite advances in the Fortran language, preprocessors continue to be used for conditional compilation and macro substitution.

LRLTRAN was developed at the Lawrence Radiation Laboratory to provide support for vector arithmetic and dynamic storage, among other extensions to support systems programming. The distribution included the LTSS operating system.

The Fortran-95 Standard includes an optional *Part 3* which defines an optional conditional compilation capability. This capability is often referred to as "CoCo".

Many Fortran compilers have integrated subsets of the C preprocessor into their systems.

SIMSCRIPT is an application specific Fortran preprocessor for modeling and simulating large discrete systems.

The F programming language was designed to be a clean subset of Fortran 95 that attempted to remove the redundant, unstructured, and deprecated features of Fortran, such as the `EQUIVALENCE` statement. F retains the array features added in Fortran 90, and removes control statements that were made obsolete by structured programming constructs added to both Fortran 77 and Fortran 90. F is described by its creators as "a compiled, structured, array programming language especially well suited to education and scientific computing."[25]

Lahey and Fujitsu teamed up to create Fortran for the Microsoft .NET platform.[26] Silverfrost FTN95 is also capable of creating .NET code.[27]

Code examples

> *For more details on this topic, see Wikibooks:Fortran/Fortran examples.*

The following program illustrates dynamic memory allocation and array-based operations, two features introduced with Fortran 90. Particularly noteworthy is the absence of `DO` loops and `IF`/`THEN` statements in manipulating the array; mathematical operations are applied to the array as a whole. Also apparent is the use of descriptive variable names and general code formatting that conform with contemporary programming style. This example computes an average over data entered interactively.

```
program average

! Read in some numbers and take the average
! As written, if there are no data points, an average of zero is returned
! While this may not be desired behavior, it keeps this example simple

implicit none

real, dimension(:), allocatable :: points
integer                         :: number_of_points
real                            :: average_points=0., positive_average=0., negative_average=0.

write (*,*) "Input number of points to average:"
read  (*,*) number_of_points

allocate (points(number_of_points))
```

```
write (*,*) "Enter the points to average:"
read  (*,*) points

! Take the average by summing points and dividing by number_of_points
if (number_of_points > 0) average_points = sum(points) / number_of_points

! Now form average over positive and negative points only
if (count(points > 0.) > 0) then
   positive_average = sum(points, points > 0.) / count(points > 0.)
end if

if (count(points < 0.) > 0) then
   negative_average = sum(points, points < 0.) / count(points < 0.)
end if

deallocate (points)

! Print result to terminal
write (*,'(a,g12.4)') 'Average = ', average_points
write (*,'(a,g12.4)') 'Average of positive points = ', positive_average
write (*,'(a,g12.4)') 'Average of negative points = ', negative_average

end program average
```

Humor

During the same Fortran Standards Committee meeting at which the name "FORTRAN 77" was chosen, a satirical technical proposal was incorporated into the official distribution bearing the title, "Letter O considered harmful". This proposal purported to address the confusion that sometimes arises between the letter "O" and the numeral zero, by eliminating the letter from allowable variable names. However, the method proposed was to eliminate the letter from the character set entirely (thereby retaining 48 as the number of lexical characters, which the colon had increased to 49). This was considered beneficial in that it would promote structured programming, by making it impossible to use the notorious `GO TO` statement as before. (Troublesome `FORMAT` statements would also be eliminated.) It was noted that this "might invalidate some existing programs" but that most of these "probably were non-conforming, anyway".[28][29]

During the standards committee battle over whether the "minimum trip count" for the FORTRAN 77 `DO` statement should be zero (allowing no execution of the block) or one (the "plunge-ahead" `DO`), another facetious alternative was proposed (by Loren Meissner) to have the minimum trip be two—since there is no need for a loop if it is only executed once.

See also

- Address programming language
- f2c
- List of Fortran compilers
- List of Fortran numerical libraries
- List of programming languages
- Matrix representation
- Row-major order

References

1. ^ "Math 169 Notes - Santa Clara University" (http://math.scu.edu/~dsmolars/ma169/notesfortran.html).
2. ^ Eugene Loh (18 June 2010). "The Ideal HPC Programming Language" (http://queue.acm.org/detail.cfm?id=1820518). *Queue* (Association of Computing Machines) **8** (6).
3. ^ Softwarepreservation.org (http://www.softwarepreservation.org/projects/FORTRAN/index.html#By_FORTRAN_project_members)
4. ^ Mindell, David, DIGITAL APOLLO, MIT Press, Cambridge MA, 2008, p.99
5. ^ The Fortran I Compiler (http://polaris.cs.uiuc.edu/publications/c1070.pdf) "The Fortran I compiler was the first major project in code optimization. It tackled problems of crucial importance whose general solution was an important research focus in compiler technology for several decades. Many classical techniques for compiler analysis and optimization can trace their origins and inspiration to the Fortran I compiler."
6. ^ Fortran creator John Backus dies - Gadgets - MSNBC.com (http://www.msnbc.msn.com/id/17704662/), MSN.com
7. ^ Backus, J. W.; H. Stern, I. Ziller, R. A. Hughes, R. Nutt, R. J. Beeber, S. Best, R. Goldberg, L. M. Haibt, H. L. Herrick, R. A. Nelson, D. Sayre, P. B. Sheridan (1957). "The FORTRAN Automatic Coding System". *Western joint computer conference: Techniques for reliability* (Los Angeles, California: Institute of Radio Engineers, American Institute of Electrical Engineers, ACM): 188–198. doi:10.1145/1455567.1455599 (http://dx.doi.org/10.1145%2F1455567.1455599). Online at [1] (http://www.softwarepreservation.org/projects/FORTRAN/paper/BackusEtAl-FortranAutomaticCodingSystem-1957.pdf), [2] (http://archive.computerhistory.org/resources/text/Fortran/102663113.05.01.acc.pdf)
8. ^ Bitsavers.org (http://www.bitsavers.org/pdf/ibm/fortran/F28-8074-3_FORTRANII_GenInf.pdf)
9. ^ Ibibilio.org (http://www.ibiblio.org/pub/languages/fortran/ch1-12.html)
10. ^ Haines, L. H. (1965). "Serial compilation and the 1401 FORTRAN compiler" (http://domino.research.IBM.com/tchjr/journalindex.nsf/495f80c9d0f539778525681e00724804/cde711e5ad6786e485256bfa00685a03?OpenDocument). *IBM Systems Journal* **4** (1): 73–80. doi:10.1147/sj.41.0073 (http://dx.doi.org/10.1147%2Fsj.41.0073). This article was reprinted, edited, in both editions of Lee, John A. N. (1967(1st), 1974(2nd)). *Anatomy of a Compiler*. Van Nostrand Reinhold.
11. ^ McCracken, Daniel D. (1965). "Preface". *A Guide to FORTRAN IV Programming*. New York: Wiley. p. v. ISBN 0-471-58281-6.
12. ^ Chilton Computing with FORTRAN (http://www.chilton-computing.org.uk/acd/literature/reports/p008.htm), Chilton-computing.org.uk
13. ^ Mil-std-1753. *DoD Supplement to X3.9-1978* (http://www.fortran.com/fortran/mil_std_1753.html). United States Government Printing Office.
14. ^ Posix 1003.9-1992. *POSIX FORTRAN 77 Language Interface – Part 1: Binding for System Application Program Interface* ***API*** (http://standards.ieee.org/reading/ieee/std_public/description/posix/1003.9-1992_desc.html). IEEE.
15. ^ "Fortran Variable Declarations" (http://h21007.www2.hp.com/portal/download/files/unprot/fortran/docs/lrm/lrm0085.htm). *Compaq Fortran*. Texas, Huston, US: Compaq Computer Corporation. 1999. Retrieved 14 ൈസപ്ടംബര് 2012. "The form CHARACTER*(*) is an obsolescent feature in Fortran 95."
16. ^ Fortran Working Group (WG5) (http://www.nag.co.uk/sc22wg5/). It may also be downloaded as a PDF file (ftp://ftp.nag.co.uk/sc22wg5/N1551-N1600/N1579.pdf) or `gzip`ped PostScript file (ftp://ftp.nag.co.uk/sc22wg5/N1551-N1600/N1579.ps.gz), FTP.nag.co.uk
17. ^ N1836, Summary of Voting/Table of Replies on ISO/IEC FDIS 1539-1, Information technology - Programming languages - Fortran - Part 1: Base language ftp://ftp.nag.co.uk/sc22wg5/N1801-N1850/N1836.pdf PDF (101 KiB)
18. ^ N1830, Information technology — Programming languages — Fortran — Part 1: Base language ftp://ftp.nag.co.uk/sc22wg5/N1801-N1850/N1830.pdf PDF (7.9 MiB)

19. ^ ISO page to ISO/IEC DTS 29113, Further Interoperability of Fortran with C (http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=45136)
20. ^ Draft of the Technical Specification (TS) 29113 ftp://ftp.nag.co.uk/sc22wg5/N1901-N1950/N1917.pdf PDF (312 kiB)
21. ^ "Doctor Fortran Goes Dutch: Fortran 2015" (http://software.intel.com/en-us/blogs/2013/08/08/doctor-fortran-goes-dutch-fortran-2015).
22. ^ Healy, MJR (1968). "Towards FORTRAN VI" (http://hopl.murdoch.edu.au/showlanguage.prx?exp=1092&language=CDC%20Fortran). *Advanced scientific Fortran by CDC*. CDC. pp. 169–172. Retrieved 10 April 2009.
23. ^ "third party release notes for Fortran v6.1" (http://www.cs-software.com/software/fortran/compaq/cvf_relnotes.html#61ver_news). 15 March 2011.
24. ^ "Internal Translator (IT) A Compiler for the IBM 650", by A. J. Perlis, J. W. Smith, and H. R. Van Zoeren, Computation Center, Carnegie Institute of Technology
25. ^ "F Programming Language Homepage" (http://www.fortran.com/F/index.html).
26. ^ "Fortran for .NET Language System" (http://www.lahey.com/lf71/lfnet.htm).
27. ^ "FTN95: Fortran 95 for Windows" (http://www.silverfrost.com/11/ftn95_overview.aspx).
28. ^ X3J3 post-meeting distribution for meeting held at Brookhaven National Laboratory in November 1976.
29. ^ "The obliteration of O", Computer Weekly, 3 March 1977

Further reading

Articles

- Allen, F.E. (September 1981). "A History of Language Processor Technology in IBM" (http://www.research.ibm.com/journal/rd/255/ibmrd2505Q.pdf). *IBM Journal of Research and Development* (IBM) **25** (5).
- Backus, J. W.; H. Stern, I. Ziller, R. A. Hughes, R. Nutt, R. J. Beeber, S. Best, R. Goldberg, L. M. Haibt, H. L. Herrick, R. A. Nelson, D. Sayre, P. B. Sheridan (1957). "The FORTRAN Automatic Coding System". *Western joint computer conference: Techniques for reliability* (Los Angeles, California: Institute of Radio Engineers, American Institute of Electrical Engineers, ACM): 188–198. doi:10.1145/1455567.1455599 (http://dx.doi.org/10.1145%2F1455567.1455599).
- Chivers, Ian D.; Sleightholme, Jane (2013). "Compiler support for the Fortran 2003 & 2008 standards" (http://www.fortranplus.co.uk/fortran_info.html). *ACM SIGPLAN Fortran Forum* (ACM) **28** (1): 26–28. doi:10.1145/1520752.1520755 (http://dx.doi.org/10.1145%2F1520752.1520755). ISSN 1061-7264 (//www.worldcat.org/issn/1061-7264).
- Pigott, Diarmuid (2006). "FORTRAN - Backus et al high-level compiler (Computer Language)" (http://hopl.murdoch.edu.au/showlanguage.prx?exp=8&language=FORTRAN). *The Encyclopedia of Computer Languages*. Murdoch University. Retrieved 5 May 2010.
- Roberts, Mark L.; Griffiths, Peter D. (1985). "Design Considerations for IBM Personal Computer Professional FORTRAN, an Optimizing Compiler" (http://www.research.ibm.com/journal/sj/241/ibmsj2401G.pdf). *IBM Systems Journal* (IBM) **24** (1): 49–60. doi:10.1147/sj.241.0049 (http://dx.doi.org/10.1147%2Fsj.241.0049).

"Core" language standards

- Ansi x3.9-1966. *USA Standard FORTRAN* (http://www.fh-jena.de/~kleine/history/languages/ansi-x3dot9-1966-Fortran66.pdf). American National Standards Institute. Informally known as FORTRAN 66.
- Ansi x3.9-1978. *American National Standard – Programming Language FORTRAN* (http://www.fortran.com/fortran/F77_std/rjcnf.html). American National Standards Institute. Also known as ISO 1539-1980, informally known as FORTRAN 77.
- ANSI X3.198-1992 (R1997) / ISO/IEC 1539:1991. *American National Standard – Programming Language Fortran Extended* (http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=17366). American National Standards Institute / ISO/IEC. Informally known as Fortran 90.
- ISO/IEC 1539-1:1997. *Information technology – Programming languages – Fortran – Part 1: Base language* (http://j3-fortran.org/doc/standing/archive/007/97-007r2/pdf/97-007r2.pdf). Informally known as Fortran 95. There are a further two parts to this standard. Part 1 has been formally adopted by ANSI.
- ISO/IEC 1539-1:2004. *Information technology – Programming languages – Fortran – Part 1: Base language* (http://www.dkuug.dk/jtc1/sc22/open/n3661.pdf). Informally known as Fortran 2003.
- ISO/IEC 1539-1:2010 (Final Draft International Standard). *Information technology – Programming languages – Fortran – Part 1: Base language* (ftp://ftp.nag.co.uk/sc22wg5/N1801-N1850/N1830.pdf). Informally known as Fortran 2008.

Related standards

- Kneis, Wilfried (October 1981). "Draft standard Industrial Real-Time FORTRAN". *ACM SIGPLAN Notices* (ACM Press) **16** (7): 45–60. doi:10.1145/947864.947868 (http://dx.doi.org/10.1145%2F947864.947868). ISSN 0362-1340 (//www.worldcat.org/issn/0362-1340).
- *ISO 8651-1:1988 Information processing systems—Computer graphics—Graphical Kernel System (GKS) language bindings—Part 1: FORTRAN* (http://www.iso.org/iso/catalogue_detail?csnumber=16024). Geneva, Switzerland: ISO. 1988.

Textbooks

- Adams, Jeanne C.; Brainerd, Walter S.; Hendrickson, Richard A.; Maine, Richard E.; Martin, Jeanne T.; Smith, Brian T. (2009). *The Fortran 2003 Handbook* (1st ed.). Springer. ISBN 978-1-84628-378-9.
- Akin, Ed (2003). *Object Oriented Programming via Fortran 90/95* (1st ed.). Cambridge University Press. ISBN 0-521-52408-3.
- Chapman, Stephen J. (2007). *Fortran 95/2003 for Scientists and Engineers* (3rd ed.). McGraw-Hill. ISBN 978-0-07-319157-7.
- Chivers, Ian; Sleightholme, Jane (2012). *Introduction to Programming with Fortran* (2nd ed.). Springer. ISBN 978-0-85729-232-2.
- Etter, D. M. (1990). *Structured FORTRAN 77 for Engineers and Scientists* (3rd ed.). The Benjamin/Cummings Publishing Company, Inc. ISBN 0-8053-0051-1.
- Ellis, T. M. R.; Phillips, Ivor R.; Lahey, Thomas M. (1994). *Fortran 90 Programming* (1st ed.). Addison Wesley. ISBN 0-201-54446-6.
- Kupferschmid, Michael (2002). *Classical Fortran: Programming for Engineering and Scientific Applications*. Marcel Dekker (CRC Press). ISBN 0-8247-0802-4.
- McCracken, Daniel D. (1961). *A Guide to FORTRAN Programming*. New York: Wiley. LCCN 61016618 (http://lccn.loc.gov/61016618).
- Metcalf, Michael; John Reid, Malcolm Cohen (2011). *Modern Fortran Explained*. Oxford University Press. ISBN 0-19-960142-9.
- Nyhoff, Larry; Sanford Leestma (1995). *FORTRAN 77 for Engineers and Scientists with an Introduction to Fortran 90* (4th ed.). Prentice Hall. ISBN 0-13-363003-X.
- Page, Clive G. (1988). *Professional Programmer's Guide to Fortran77* (http://www.star.le.ac.uk/~cgp/prof77.html) (7 June 2005 ed.). London: Pitman. ISBN 0-273-02856-1. Retrieved 4 May 2010.
- Press, William H. (1996). *Numerical Recipes in Fortran 90: The Art of Parallel Scientific Computing* (http://www.nrbook.com/a/bookf90pdf.php).

Cambridge, UK: Cambridge University Press. ISBN 0-521-57439-0.
- Sleighthome, Jane; Chivers, Ian David (1990). *Interactive Fortran 77: A Hands-On Approach* (http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.95.9503). Computers and their applications (2nd ed.). Chichester: E. Horwood. ISBN 0-13-466764-6.

External links

- JTC1/SC22/WG5 (http://www.nag.co.uk/sc22wg5/) — The ISO/IEC Fortran Working Group
- History of Fortran (http://www.softwarepreservation.org/projects/FORTRAN/) at the Computer History Museum's Software Preservation Group
- Bemer, Bob, "Who Was Who in IBM's Programming Research? -- Early FORTRAN Days" (http://www.trailing-edge.com/~bobbemer/PRORES.HTM), January 1957, Computer History Vignettes
- Comprehensive Fortran Standards Documents (http://gcc.gnu.org/wiki/GFortranStandards) from the gfortran project
- Fortran IV - IBM System/360 and System/370 Fortran IV Language - GC28-6515-10 (http://www.fh-jena.de/~kleine/history/languages/GC28-6515-10-FORTRAN-IV-Language.pdf)
- Fortran 77, 90, 95, 2003, 2008 Information & Resources (http://www.fortranplus.co.uk/fortran_info.html)
- Fortran 77 4.0 Reference Manual (http://wwwcdf.pd.infn.it/localdoc/f77_sun.pdf)
- Fortran 90 Reference Card (http://www.pa.msu.edu/~duxbury/courses/phy480/fortran90_refcard.pdf)
- ECMA-9 Fortran Standard (1965) (http://www.ecma-international.org/publications/files/ECMA-ST-WITHDRAWN/ECMA-9,%201st%20Edition,%20April%201965.pdf)
- Online FORTRAN F compiler for small experiments (http://www.vintagebigblue.org/Compilerator/FORTRAN/dosvsFORTFCompile.php)
- Fortran coding form (http://www.atkielski.com/PDF/data/fortran.pdf)