Fortran

Fortran (/ˈfɔːrtræn/; formerly **FORTRAN**, derived from *Formula Translation*[2]) is a general-purpose, compiled imperative programming language that is especially suited to numeric computation and scientific computing.

Originally developed by IBM[3] in the 1950s for scientific and engineering applications, FORTRAN came to subsequently dominate scientific computing. It has been in use for over six decades in computationally intensive areas such as numerical weather prediction, finite element analysis, computational fluid dynamics, geophysics, computational physics, crystallography and computational chemistry. It is a popular language for high-performance computing[4] and is used for programs that benchmark and rank the world's fastest supercomputers.[5][6]

Fortran has had multiple versions, each adding extensions while largely retaining compatibility with prior 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), concurrent programming (Fortran 2008), and native parallel computing capabilities (Coarray Fortran 2008/2018).

Fortran's design was the basis for many other programming languages. Among the better-known is BASIC, which is based on FORTRAN II with a number of syntax cleanups, notably better logical structures,[7] and other changes to work more easily in an interactive environment.[8]

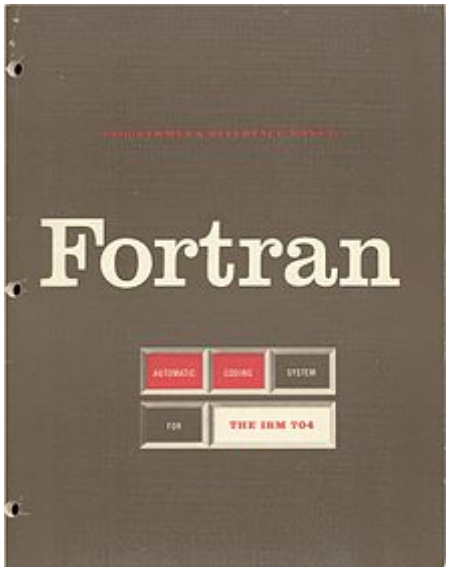
As of April 2021, Fortran was ranked 20th in the TIOBE index, a measure of the popularity of programming languages, climbing 14 positions from its ranking of 34th in January 2020.[9]

Contents

Fortran

Paradigm	Multi-paradigm: structured, imperative (procedural, object-oriented), generic, array
Designed by	John Backus
Developer	John Backus and IBM
First appeared	1957
Stable release	Fortran 2018 (ISO/IEC 1539-1:2018) / 28 November 2018
Typing discipline	strong, static, manifest
Filename extensions	`.f`, `.for`, `.f90`
Website	fortran-lang.org (https://fortran-lang.org)
Major implementations	
Absoft, Cray, GFortran, G95, IBM XL Fortran, Intel, Hitachi, Lahey/Fujitsu, Numerical Algorithms Group, Open Watcom, PathScale, PGI, Silverfrost, Oracle Solaris Studio, Visual Fortran, others	
Influenced by	
Speedcoding	

Influenced
ALGOL 58, BASIC, C, Chapel,[1] CMS-2, DOPE, Fortress, PL/I, PACT I, MUMPS, IDL, Ratfor



The Fortran Automatic Coding System for the IBM 704 (15 October 1956), the first programmer's reference manual for Fortran

Naming

The names of earlier versions of the language through FORTRAN 77 were conventionally spelled in all-uppercase[10] (FORTRAN 77 was the last version in which the Fortran character set included only uppercase letters[11]). The official language standards for Fortran have referred to the language as “Fortran” with initial caps (rather than “FORTRAN” in all-uppercase) since Fortran 90.

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.[12]:69 Backus' historic FORTRAN team consisted of programmers Richard Goldberg, Sheldon F. Best, Harlan Herrick, Peter

Sheridan, Roy Nutt, Robert Nelson, Irving Ziller, Harold Stern, Lois Haibt, and David Sayre.[13] Its concepts included easier entry of equations into a computer, an idea developed by J. Halcombe Laning and demonstrated in the Laning and Zierler system of 1952.[14]


An IBM 704 mainframe computer

A draft specification for *The IBM Mathematical Formula Translating System* was completed by November 1954.[12]:71 The first manual for FORTRAN appeared in October 1956,[12]:72 with the first FORTRAN compiler delivered in April 1957.[12]:75 This was the first optimizing compiler, because customers were reluctant to use a high-level programming language unless its compiler could generate code with performance approaching that of hand-coded assembly language.[15]

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."[16]

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.[17]

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 cross-platform programming language.

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 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
- `GO TO`, computed `GO TO`, `ASSIGN`, and assigned `GO TO`
- `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 reminiscent of (but not readily implementable by) a three-way comparison instruction (CAS—Compare Accumulator with Storage) available on the 704. The statement provided the only way to compare numbers—by testing their difference, with an attendant risk of overflow. This deficiency was later overcome by "logical" facilities introduced in FORTRAN IV.

The `FREQUENCY` statement 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 one entry point and one 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 running the program once 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.[13]

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, e.g. the `register` keyword in C.

The first FORTRAN compiler reported diagnostic information by halting the program when an error was found and outputting an error code on its console. That code could be looked up by the programmer in an error messages table in the operator's manual, providing them with a brief description of the problem.[18][19] Later, an error-handling subroutine to handle user errors such as division by zero, developed by NASA,[20] was incorporated, informing users of which line of code contained the error.

Fixed layout and punched cards

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. Punched card codes included no lower-case letters or many special characters, and special versions of the IBM 026 keypunch were offered that would correctly print the re-purposed special characters used in FORTRAN.

Reflecting punched card input practice, Fortran programs were originally written in a fixed-column format, with the first 72 columns read into twelve 36-bit words.

A letter "C" in column 1 caused the entire card to be treated as a comment and ignored by the compiler. Otherwise, the columns of the card were divided into four fields:

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

- 1 to 5 were the label field: a sequence of digits here was taken as a label for use in DO or control statements such as GO TO and IF, or to identify a FORMAT statement referred to in a WRITE or READ statement. Leading zeros are ignored and 0 is not a valid label number.
- 6 was a continuation field: a character other than a blank or a zero here caused the card to be taken as a continuation of the statement on the prior card. The continuation cards were usually numbered 1, 2, *etc.* and the starting card might therefore have zero in its continuation column—which is not a continuation of its preceding card.
- 7 to 72 served as the statement field.
- 73 to 80 were ignored (the IBM 704's card reader only used 72 columns).[21]

Columns 73 to 80 could therefore be used for identification information, such as punching a sequence number or text, 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 due to limitations on their card readers.[22] Keypunches could be programmed to tab to column 7 and skip out after column 72. Later compilers relaxed most fixed-format restrictions, and the requirement was eliminated in the Fortran 90 standard.

Within the statement field, whitespace characters (blanks) were ignored outside a text literal. This allowed omitting spaces between tokens for brevity or including spaces within identifiers for clarity. For example, `AVG OF X` was a valid identifier, equivalent to `AVGOFX`, and `101010DO101I=1,101` was a valid statement, equivalent to `10101 DO 101 I = 1, 101` because the zero in column 6 is treated as if it were a space (!), while `101010DO101I=1.101` was instead `10101 DO101I = 1.101`, the assignment of 1.101 to a variable called `DO101I`. Note the slight visual difference between a comma and a period.

Hollerith strings, originally allowed only in FORMAT and DATA statements, were prefixed by a character count and the letter H (e.g., `26HTHIS IS ALPHANUMERIC DATA.`), allowing blanks to be retained within the character string. Miscounts were a problem.

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:[23]

- `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 supported no recursion in subroutines. Early computer architectures supported no concept of a stack, and when they did directly support subroutine calls, the return location was often stored in one fixed location adjacent to the subroutine code (e.g. the IBM 1130) or a specific machine register (IBM 360 *et seq*), which only allows recursion if a stack is maintained by software and the return address is stored on the stack before the call is made and restored after the call returns. Although not specified in FORTRAN 77, many F77 compilers supported recursion as an option, and the Burroughs mainframes, designed with recursion built-in, did so by default. It became a standard in Fortran 90 via the new keyword RECURSIVE.[24]

Simple FORTRAN II program

This program, for Heron's formula, reads data on a tape reel containing three 5-digit integers A, B, and C as input. There are no "type" declarations available: variables whose name starts with I, J, K, L, M, or N are "fixed-point" (i.e. integers), otherwise floating-point. Since integers are to be processed in this example, the names of the variables start with the letter "I". The name of a variable must start with a letter and can continue with both letters and digits, up to a limit of six characters in FORTRAN II. 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 occupying ten spaces along the line of output and showing 2 digits after the decimal point, the .2 in F10.2 of the FORMAT statement with label 601.

```
C AREA OF A TRIANGLE WITH A STANDARD SQUARE ROOT FUNCTION
C INPUT - TAPE 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 OR ZERO
C FURTHERMORE, THE SUM OF TWO SIDES OF A TRIANGLE
C MUST BE 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 = SQRTF( 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
```

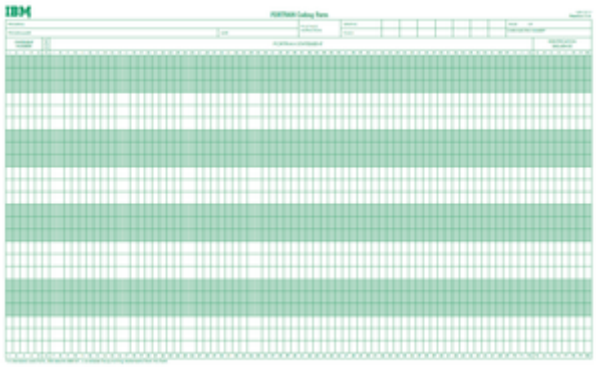
FORTRAN III

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.[12]:76 Early versions of FORTRAN provided by other vendors suffered from the same disadvantage.

IBM 1401 FORTRAN

FORTRAN was provided for the IBM 1401 computer by an innovative 63-phase compiler that ran entirely in its core memory of only 8000 (six-bit) characters. The compiler could be run from tape, or from a 2200-card deck; it used no further tape or disk storage. It kept the program in memory and loaded overlays that gradually transformed it, in place, into executable form, as described by Haines.[25] This article was reprinted, edited, in both editions of *Anatomy of a Compiler* [26] and in the IBM manual "Fortran Specifications and Operating Procedures, IBM 1401".[27] The executable form was not entirely machine language; rather, floating-point arithmetic, sub-scripting, input/output, and function references were interpreted, preceding UCSD Pascal P-code by two decades.

IBM later provided a FORTRAN IV compiler for the 1400 series of computers.[28]



A reproduction of 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 IV

IBM began development of FORTRAN IV starting in 1961, as a result of customer demands. *FORTRAN IV* 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, IBM 7094, and later for the IBM 1401 in 1966.

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.[29]

Between 1966 and 1968, IBM offered several FORTRAN IV compilers for its System/360, each named by letters that indicated the minimum amount of memory the compiler needed to run. [30] The letters (F, G, H) matched the codes used with System/360 model numbers to indicate memory size, each letter increment being a factor of two larger:[31]:p. 5

- 1966 : FORTRAN IV F for DOS/360 (64K bytes)
- 1966 : FORTRAN IV G for OS/360 (128K bytes)
- 1968 : FORTRAN IV H for OS/360 (256K bytes)

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

FORTRAN 66

Perhaps the most significant development in the early history of FORTRAN was the decision by the *American Standards Association* (now American National Standards Institute (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

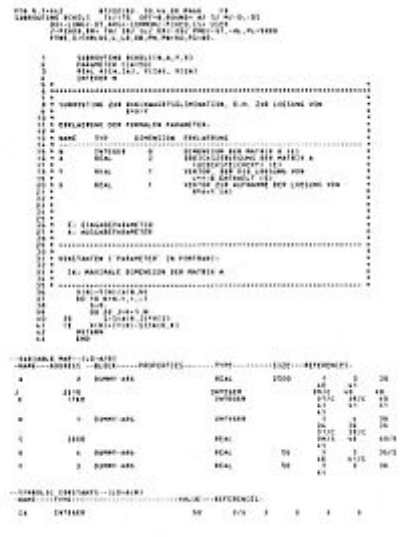
IV, which had served as a de facto standard), and *Basic FORTRAN* (based on FORTRAN II, but stripped of its machine-dependent features). The FORTRAN defined by the first standard, officially denoted X3.9-1966, became known as *FORTRAN 66* (although many continued to term it FORTRAN IV, the language on 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
- `GO TO`, computed `GO TO`, assigned `GO TO`, and `ASSIGN` statements
- Logical `IF` and arithmetic (three-way) `IF` statements
- `DO` loop statement
- `READ`, `WRITE`, `BACKSPACE`, `REWIND`, and `ENDFILE` statements for sequential I/O
- `FORMAT` statement and assigned format
- `CALL`, `RETURN`, `PAUSE`, and `STOP` statements
- Hollerith constants in `DATA` and `FORMAT` statements, and as arguments to procedures
- Identifiers of up to six characters in length
- Comment lines
- `END` line

FORTRAN 77

After the release of the FORTRAN 66 standard, compiler vendors introduced several 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, called *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, to override implicit conventions that undeclared variables are INTEGER if their name begins with I, J, K, L, M, or N (and REAL otherwise)



FORTRAN-77 program with compiler output, written on a CDC 175 at RWTH Aachen University, Germany, in 1987

- `CHARACTER` data type, replacing Hollerith strings 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 (e.g. `SQRT` also accepts arguments of other types, such as `COMPLEX` or `REAL*16`).
- 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.)

4.3 BSD for the Digital Equipment Corporation (DEC) VAX, displaying the manual for FORTRAN 77 (f77) compiler

In this revision of the standard, a number of features were removed or altered in a manner that might invalidate formerly 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 prior 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 COMMON and DATA statements, optimization code levels for compiling, detailed error listings, extensive warning messages, and debugging features.[32] MNF was developed by people (Liddiard & Mundstock) at the University of Minnesota.[33] MNF was available basically for free.

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.[34] 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`, `EXIT`, `CYCLE`, 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.[35] 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 (In the previous standard, it was only six characters).
- 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 prior revision, Fortran 90 removed no features.[36] 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. All of the functionalities of these early version features are performed by new Fortran 95 features. Some are kept to simplify porting of old programs but may eventually be deleted.

Obsolescent feature	Example	Status/fate in Fortran 95
Arithmetic IF-statement	`IF (X) 10, 20, 30`	Deprecated
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`
		Deprecated

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 )`	Deprecated
PAUSE statement	`PAUSE 600`	Deleted

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

Assigned statement numbers and FORMAT specifiers	`ASSIGN 606 TO F ... WRITE ( 6, F )...`	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)[37]
Fixed form source code	Column 1 contains C or * or ! for comments. Columns 1 through 5 for statement numbers Any character in column 6 for continuation. Columns 73 and up ignored	(obsolete)

"Hello, World!" example

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

Fortran 95

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
- Initialization of pointers to `NULL()`
- 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` index variables
- Branching to an `END IF` statement from outside its block
- `PAUSE` statement
- `ASSIGN` and assigned `GO TO` statement, and assigned format specifiers
- `H` Hollerith 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, elements of allocatable arrays are contiguous, and aliasing is not an issue for optimization of array references, allowing compilers to generate faster code than in the case of pointers.[38])

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 length 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.[39] 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.[40]

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 *sub-modules*, which make Fortran modules more similar to Modula-2 modules. They are similar to Ada private child sub-units. 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

ISO/IEC 1539-1:2010, informally known as Fortran 2008, was approved in September 2010.[41][42] As with Fortran 95, this is a minor upgrade, incorporating clarifications and corrections to Fortran 2003, as well as introducing some new capabilities. The new capabilities include:

- Sub-modules—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.[43]

A supplement to Fortran 2008 is the International Organization for Standardization (ISO) Technical Specification (TS) 29113 on *Further Interoperability of Fortran with C*,[44][45] which has been submitted to ISO in May 2012 for approval. The specification adds support for accessing the array descriptor from C and allows ignoring the type and rank of arguments.

Fortran 2018

The latest revision of the language (Fortran 2018) was earlier referred to as Fortran 2015.[46] It is a significant revision and was released on 28 November 2018.[47]

Fortran 2018 incorporates two previously published Technical Specifications:

- ISO/IEC TS 29113:2012 Further Interoperability with C[48]
- ISO/IEC TS 18508:2015 Additional Parallel Features in Fortran[49]

Additional changes and new features include support for ISO/IEC/IEEE 60559:2011 (the version of the IEEE floating-point standard before the latest minor revision IEEE 754-2019), hexadecimal input/output, IMPLICIT NONE enhancements and other changes.[50][51][52][53]

Language features

A full description of Fortran language features since Fortran 95 is covered in the related article, *Fortran 95 language features*.

Science and engineering

Although a 1968 journal article by the authors of BASIC already described FORTRAN as "old-fashioned",[54] Fortran has now been in use for several decades and there is a vast body of Fortran software in daily use throughout the scientific and engineering communities.[55] Jay Pasachoff wrote in 1984 that "physics and astronomy students simply have to learn FORTRAN. So much exists in FORTRAN that it seems unlikely that scientists will change to Pascal, Modula-2, or whatever."[56] In 1993, Cecil E. Leith called FORTRAN the "mother tongue of scientific computing", adding that its replacement by any other possible language "may remain a forlorn hope".[57]

It is the primary language for some of the most intensive super-computing tasks, such as in astronomy, climate modeling, computational chemistry, computational economics, computational fluid dynamics, computational physics, data analysis, hydrological modeling, numerical linear algebra and numerical libraries (LAPACK, IMSL and NAG), optimization, satellite simulation, structural engineering, and weather prediction. Many of the floating-point benchmarks to gauge the performance of new computer processors, such as the floating-point components of the SPEC benchmarks (e.g., CFP2006 (http://www.spec.org/cpu2006/CFP2006/), CFP2017 (http://www.spec.org/cpu2017/Docs/overview.html#benchmarks)) are written in Fortran. Math algorithms are well documented in Numerical Recipes.

Apart from this, more modern codes in computational science generally use large program libraries, such as METIS for graph partitioning, PETSc or Trilinos for linear algebra capabilities, DUNE or FEniCS for mesh and finite element support, and other generic libraries. Since the early 2000s, many of the widely used support libraries have also been implemented in C and more recently, in C++. On the other hand, high-level languages such as MATLAB, Python, and R have become popular in particular areas of computational science. Consequently, a growing fraction of scientific programs is also written in such higher-level scripting languages. For this reason, facilities for inter-operation with C were added to Fortran 2003 and enhanced by the ISO/IEC technical specification 29113, which was incorporated into Fortran 2018 to allow more flexible interoperation with other programming languages.

Software for NASA probes Voyager 1 and Voyager 2 was originally written in FORTRAN 5, and later ported to FORTRAN 77. As of 25 September 2013, some of the software is still written in Fortran and some has been ported to C.[58]

Portability

Portability was a problem in the early days because there was no agreed upon 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*,[59][60] 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.[61][62]

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.[60] 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

Until the Fortran 66 standard was developed, each compiler supported its own variant of Fortran. Some were more divergent from the mainstream than others.

The first Fortran compiler set a high standard of efficiency for compiled code. This goal made it difficult to create a compiler so it was usually done by the computer manufacturers to support hardware sales. This left an important niche: compilers that were fast and provided good diagnostics for the programmer (often a student). Examples include Watfor, Watfiv, PUFFT, and on a smaller scale, FORGO, Wits Fortran, and Kingston Fortran 2.

Fortran 5

Fortran 5 was 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.

FORTRAN V

FORTRAN V was distributed by Control Data Corporation in 1968 for the CDC 6600 series. The language was based upon FORTRAN IV.[63]

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.[64]

Specific variants

Vendors of high-performance scientific computers (*e.g.*, Burroughs, Control Data Corporation (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 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 in the late 1950s.[65] 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 (n1, n2, ..., nm), i`
- `IF (a) n1, n2, n3`
- `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 through 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 preprocessors 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.

One of the earliest versions of FORTRAN, introduced in the '60s, was popularly used in colleges and universities. Developed, supported, and distributed by the University of Waterloo, WATFOR was based largely on FORTRAN IV. A student using WATFOR could submit their batch FORTRAN job and, if there were no syntax errors, the program would move straight to execution. This simplification allowed students to concentrate on their program's syntax and semantics, or execution logic flow, rather than dealing with submission Job Control Language (JCL), the compile/link-edit/execution successive process(es), or other complexities of the mainframe/minicomputer environment. A down side to this simplified environment was that WATFOR was not a good choice for programmers needing the expanded abilities of their host processor(s), e.g., WATFOR typically had very limited access to I/O devices. WATFOR was succeeded by WATFIV and its later versions.

```
 program; s=0 i=1,n; s=s+1; stop i; s='s'  Stop
```

(line programming)

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".[66]

Lahey and Fujitsu teamed up to create Fortran for the Microsoft .NET Framework.[67] Silverfrost FTN95 is also capable of creating .NET code.[68]

Code 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=0
   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".[69][70]

When X3J3 debated whether the minimum trip count for a DO loop should be zero or one in Fortran 77, Loren Meissner suggested a minimum trip count of two—reasoning *(tongue-in-cheek)* that if it was less than two then there would be no reason for a loop!

When assumed-length arrays were being added, there was a dispute as to the appropriate character to separate upper and lower bounds. In a comment examining these arguments, Dr. Walt Brainerd penned an article entitled "Astronomy vs. Gastroenterology" because some proponents had suggested using the star or asterisk ("*"), while others favored the colon (":").

In FORTRAN 77 (and most earlier versions), variable names beginning with the letters I–N had a default type of integer, while variables starting with any other letters defaulted to real, although programmers could override the defaults with an explicit declaration.[71] This led to the joke: "In Fortran, GOD is REAL (unless declared INTEGER)."

See also

- f2c
- FORMAC
- List of Fortran compilers
- List of Fortran numerical libraries
- List of programming languages
- Matrix representation
- Row-major order
- Spaghetti code

References

1. "Chapel spec (Acknowledgements)" (http://chapel.cray.com/spec/spec-0.98.pdf) (PDF). Cray Inc. 1 October 2015. Retrieved 14 January 2016.
2. "FORTRAN" (http://www.thefreedictionary.com/FORTRAN). *American Heritage Dictionary of the English Language* (5 ed.). The Free Dictionary. 2011. Retrieved 8 February 2016.
3. John Backus. "The history of FORTRAN I, II and III" (http://www.softwarepreservation.org/projects/FORTRAN/paper/p25-backus.pdf) (PDF). Softwarepreservation.org. Retrieved 19 November 2014.
4. Eugene Loh (18 June 2010). "The Ideal HPC Programming Language" (http://queue.acm.org/detail.cfm?id=1820518). *Queue*. **8** (6).
5. "HPL – A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers" (http://www.netlib.org/benchmark/hpl). Retrieved 21 February 2015.
6. "Q13. What are the benchmarks?" (http://www.spec.org/cpu2017/Docs/overview.html#benchmarks). *Overview - CPU 2017*. SPEC. Retrieved 13 November 2019.
7. "Fifty Years of BASIC" (http://time.com/69316/basic/). *Time*. 29 April 2014.
8. Szczepaniak, John (1 May 2014). "A basic history of BASIC on its 50th birthday" (https://www.gamasutra.com/view/news/216469/A_basic_history_of_BASIC_on_its_50th_birthday.php). *Gamasutra*.

9. TIOBE Software BV (April 2021). "TIOBE Index for April 2021" (https://www.tiobe.com/tiobe-index/). *TIOBE.com*. TIOBE. Retrieved 11 April 2021.
10. Chapman, Stephen J. (2018). *Fortran for Scientists and Engineers* (https://www.mheducation.com/highered/product/fortran-scientists-engineers-chapman/M9780073385891.html) (Fourth ed.). New York: McGraw-Hill Education. p. 13. ISBN 978-0-07-338589-1.
11. The “Fortran character set” defined by the FORTRAN 77 standard was the minimal character set that standard-compliant compilers were required to support; in practice, many FORTRAN 77 compilers supported the full ASCII character set.
12. Backus, John (October–December 1998). "The History of Fortran I, II, and III" (http://www.softwarepreservation.org/projects/FORTRAN/paper/p165-backus.pdf) (PDF). *IEEE Annals of the History of Computing*. **20** (4): 68–78. doi:10.1109/85.728232 (https://doi.org/10.1109%2F85.728232). Archived (https://web.archive.org/web/20160303230833/http://www.softwarepreservation.org/projects/FORTRAN/paper/p165-backus.pdf) (PDF) from the original on 3 March 2016. Retrieved 17 June 2020. [1] (https://archive.org/details/history-of-fortran)[2] (https://ia800807.us.archive.org/0/items/history-of-fortran/Image072217151026_text.pdf)
13. J. W. Backus; R. J. Beeber; S. Best; R. Goldberg; L. M. Haibt; H. L. Herrick; R. A. Nelson; D. Sayre; P. B. Sheridan; H. Stern; L. Ziller; R. A. Hughes; R. Nutt (February 1957). *The FORTRAN Automatic Coding System* (http://www.softwarepreservation.org/projects/FORTRAN/paper/BackusEtAl-FortranAutomaticCodingSystem-1957.pdf) (PDF). Western Joint Computer Conference. pp. 188–198. doi:10.1145/1455567.1455599 (https://doi.org/10.1145%2F1455567.1455599).
14. Mindell, David, Digital Apollo, MIT Press, Cambridge MA, 2008, p.99
15. Padua, David (January–February 2000). "The Fortran I Compiler" (https://web.archive.org/web/20200617113640/http://polaris.cs.uiuc.edu/publications/c1070.pdf) (PDF). *Computing in Science and Engineering (CiSE)*. the Top Algorithms. University of Illinois: IEEE: 70–75. Archived from the original (http://polaris.cs.uiuc.edu/publications/c1070.pdf) (PDF) on 17 June 2020. "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."
16. Brian Bergstein (20 May 2007). "Fortran creator John Backus dies" (http://www.nbcnews.com/id/17704662/ns/technology_and_science-tech_and_gadgets/t/fortran-creator-john-backus-dies). MSNBC. Retrieved 29 October 2018.
17. http://scihi.org/fortran-programming/
18. Applied Science Division and Programming Research Department, International Business Machines Corporation (15 October 1956). *The FORTRAN Automatic Coding System for the IBM 704 EDPM : Programmer's Reference Manual* (http://archive.computerhistory.org/resources/text/Fortran/102649787.05.01.acc.pdf) (PDF). pp. 19–20.
19. Programming Research Department, International Business Machines Corporation (8 April 1957). *The FORTRAN Automatic Coding System for the IBM 704 EDPM : Preliminary Operator's Manual* (http://www.softwarepreservation.org/projects/FORTRAN/manual/Prelim_Oper_Man-1957_04_07.pdf) (PDF). pp. 6–37.
20. Betty Jo Armstead (21 January 2015). "My Years at NASA" (https://web.archive.org/web/20191224083647/https://spaceodyssey.dmns.org/media/62497/myyearsatnasa-_bettyjoarmstead.pdf) (PDF). *Denver Museum of Nature & Science*. Archived from the original (https://spaceodyssey.dmns.org/media/62497/myyearsatnasa-_bettyjoarmstead.pdf) (PDF) on 24 December 2019. Retrieved 15 June 2019.
21. Reference Manual, IBM 7090 Data Processing System (http://www.mirrorservice.org/sites/www.bitsavers.org/pdf/ibm/7090/22-6528-4_7090Manual.pdf), 1961, IBM A22-6528-3.
22. "Fortran II User Manual" (http://www.bitsavers.org/pdf/ibm/fortran/F28-8074-3_FORTRANII_GenInf.pdf) (PDF). Bitsavers.org. Retrieved 19 November 2014.

23. *Reference Manual, FORTRAN II for the IBM 704 Data Processing System* (http://bitsavers.org/pdf/ibm/704/C28-6000-2_704_FORTRANII.pdf) (PDF). 1958. C28-6000-2.
24. "Ibibilio.org" (http://www.ibiblio.org/pub/languages/fortran/ch1-12.html). Ibiblio.org. Retrieved 15 September 2014.
25. 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 (https://doi.org/10.1147%2Fsj.41.0073).
26. Lee, John A. N. (1967). *Anatomy of a Compiler*. Van Nostrand Reinhold.
27. *Fortran Specifications and Operating Procedures, IBM 1401* (http://bitsavers.org/pdf/ibm/1401/C24-1455-2_Fortran_Specifications_and_Operating_Procedures_Apr65.pdf) (PDF). IBM. C24-1455-2.
28. *Fortran IV Language Specifications, Program Specifications, and Operating Procedures, IBM 1401, 1440, and 1460* (http://bitsavers.org/pdf/ibm/1401/C24-3322-2_Fortran_IV_Language_Specifications_IBM_1401_1440_1460_Apr66.pdf) (PDF). IBM. April 1966. C24-3322-2.
29. McCracken, Daniel D. (1965). "Preface" (https://archive.org/details/guidetofortraniv00mccr). *A Guide to FORTRAN IV Programming*. New York: Wiley. p. v. ISBN 978-0-471-58281-6.
30. "List of FORTRAN Implementations 1957 - 1967" (http://www.fortran.bcs.org/2007/jubilee/implementations.php). IEEE Annals. 2017. Retrieved 17 October 2017.
31. *IBM System/360 Model 50 Functional Characteristics* (http://bitsavers.org/pdf/ibm/360/funcChar/A22-6898-1_360-50_funcChar_1967.pdf) (PDF). IBM. 1967. A22-6898-1.
32. "FORTRAN Compilers and Loaders" (http://www.chilton-computing.org.uk/acd/literature/reports/p008.htm). Chilton-programming.org.uk. Retrieved 19 November 2014.
33. Frisch, Michael (December 1972). "Remarks on Algorithms". *Communications of the ACM*. **15** (12): 1074. doi:10.1145/361598.361914 (https://doi.org/10.1145%2F361598.361914). S2CID 6571977 (https://api.semanticscholar.org/CorpusID:6571977).
34. Mil-std-1753. *DoD Supplement to X3.9-1978* (https://web.archive.org/web/20071109170658/http://www.fortran.com/fortran/mil_std_1753.html). United States Government Printing Office. Archived from the original (http://www.fortran.com/fortran/mil_std_1753.html) on 9 November 2007. Retrieved 13 December 2007.
35. *IEEE 1003.9-1992 - IEEE Standard for InformationTechnology - POSIX(R) FORTRAN 77 Language Interfaces - Part 1: Binding for System Application Program Interface (API)* (https://standards.ieee.org/standard/1003_9-1992.html). IEEE. Retrieved 24 November 2018.
36. Appendix B.1
37. "Declaration Statements for Character Types" (https://web.archive.org/web/20180918054356/http://h30266.www3.hpe.com/odl/unix/progtool/cf95au56/lrm0085.htm). *Compaq Fortran Language Reference Manual*. Texas, Huston, US: Compaq Computer Corporation. 1999. Archived from the original (http://h30266.www3.hpe.com/odl/unix/progtool/cf95au56/lrm0085.htm) on 18 September 2018. Retrieved 17 September 2018. "The form CHARACTER*(*) is an obsolescent feature in Fortran 95."
38. "Fortran 95 Reference" (https://gcc.gnu.org/onlinedocs/gcc-4.1.0/gfortran/). Gnu.Org. Retrieved 10 May 2014.
39. "Fortran 2003– Last Working Draft" (http://www.j3-fortran.org/doc/year/04/04-007.txt). Gnu.Org. Retrieved 10 May 2014.
40. Fortran Working Group (WG5) (http://www.nag.co.uk/sc22wg5/). It may also be downloaded as a PDF file (https://wg5-fortran.org/N1551-N1600/N1579.pdf), FTP.nag.co.uk
41. "N1836, Summary of Voting/Table of Replies on ISO/IEC FDIS 1539-1, Information technology – Programming languages – Fortran – Part 1: Base language" (https://wg5-fortran.org/N1801-N1850/N1836.pdf) (PDF).
42. "Fortran 2008 – Last Working Draft" (http://www.j3-fortran.org/doc/year/10/10-007.pdf) (PDF). Gnu.Org. Retrieved 10 May 2014.

43. N1830, Information technology – Programming languages – Fortran – Part 1: Base language [3] (ftp://ftp.nag.co.uk/sc22wg5/N1801-N1850/N1830.pdf)
44. 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)
45. "Draft of the Technical Specification (TS) 29113" (https://wg5-fortran.org/N1901-N1950/N1917.pdf) (PDF). *wg5-fortran.org*.
46. "Doctor Fortran in "Eighteen is the new Fifteen" " (https://software.intel.com/en-us/blogs/2017/11/20/doctor-fortran-in-eighteen-is-the-new-fifteen). Software.intel.com. Retrieved 20 November 2017.
47. "Fortran 2018" (https://wg5-fortran.org/f2018.html). ISO. Retrieved 30 November 2018.
48. "Further Interoperability with C" (https://wg5-fortran.org/N1901-N1950/N1942.pdf) (PDF). ISO. Retrieved 20 November 2017.
49. "Additional Parallel Features in Fortran" (http://isotc.iso.org/livelink/livelink?func=ll&objId=17288706&objAction=Open). ISO. Retrieved 20 November 2017.
50. "The New Features of Fortran 2015" (http://isotc.iso.org/livelink/livelink?func=ll&objId=19044944&objAction=Open). ISO. Retrieved 23 June 2017.
51. "Doctor Fortran in "One Door Closes" " (https://software.intel.com/en-us/blogs/2015/09/04/doctor-fortran-in-one-door-closes). Software.intel.com. Retrieved 21 September 2015.
52. "Doctor Fortran Goes Dutch: Fortran 2015" (http://software.intel.com/en-us/blogs/2013/08/08/doctor-fortran-goes-dutch-fortran-2015). Software.intel.com. Retrieved 19 November 2014.
53. Fortran 2018 Interpretation Document (http://j3-fortran.org/doc/year/18/18-007r1.pdf), 9 October 2018
54. Kemeny, John G.; Kurtz, Thomas E. (11 October 1968). "Dartmouth Time-Sharing" (http://dtss.dartmouth.edu/sciencearticle/index.html). *Science*. **162** (3850): 223–228. Bibcode:1968Sci...162..223K (https://ui.adsabs.harvard.edu/abs/1968Sci...162..223K). doi:10.1126/science.162.3850.223 (https://doi.org/10.1126%2Fscience.162.3850.223). PMID 5675464 (https://pubmed.ncbi.nlm.nih.gov/5675464).
55. Phillips, Lee. "Scientific computing's future: Can any coding language top a 1950s behemoth?" (https://arstechnica.com/science/2014/05/scientific-computings-future-can-any-coding-language-top-a-1950s-behemoth/). *Ars Technica*. Retrieved 8 May 2014.
56. Pasachoff, Jay M. (April 1984). "Scientists: FORTRAN vs. Modula-2" (https://archive.org/stream/byte-magazine-1984-04/1984_04_BYTE_09-04_Real-World_Interfacing#page/n403/mode/2up). *BYTE* (letter). p. 404. Retrieved 6 February 2015.
57. Galperin, Boris (1993). "26". *Large Eddy Simulation of Complex Engineering and Geophysical Flows*. London: Cambridgey. p. 573. ISBN 978-0-521-43009-8.
58. "Interstellar 8-Track: How Voyager's Vintage Tech Keeps Running" (https://www.wired.com/2013/09/vintage-voyager-probes/). *WIRED*. Retrieved 23 December 2017.
59. "Methods to ensure the standardization of FORTRAN software". OSTI 5361454 (https://www.osti.gov/biblio/5361454). "PFORT ... Library ..."
60. P. A. Fox (1977). "Port — A portable mathematical subroutine library". *A portable mathematical subroutine library*. Lecture Notes in Computer Science. **57**. pp. 165–177. doi:10.1007/3-540-08446-0_42 (https://doi.org/10.1007%2F3-540-08446-0_42). ISBN 978-3-540-08446-4. "PORT ... written in (PFORT) .. ANS Fortran"
61. D. E. Whitten (1975). "A machine and configuration independent Fortran: Portable Fortran". doi:10.1109/TSE.1975.6312825 (https://doi.org/10.1109%2FTSE.1975.6312825). S2CID 16485156 (https://api.semanticscholar.org/CorpusID:16485156).
62. "Portability Issues" (https://www.gnu.org/software/sather/docs-1.2/tutorial/fortran-portability.html). ".. discusses .. portability of .. Fortran"

63. Healy, MJR (1968). "Towards FORTRAN VI" (https://web.archive.org/web/20090705035806/http://hopl.murdoch.edu.au/showlanguage.prx?exp=1092&language=CDC%20Fortran). *Advanced scientific Fortran by CDC*. CDC. pp. 169–172. Archived from the original (http://hopl.murdoch.edu.au/showlanguage.prx?exp=1092&language=CDC%20Fortran) on 5 July 2009. Retrieved 10 April 2009.
64. "third party release notes for Fortran v6.1" (http://www.cs-software.com/software/fortran/compaq/cvf_relnotes.html#61ver_news). Cs-software.com. 15 March 2011. Retrieved 19 November 2014.
65. "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
66. "F Programming Language Homepage" (https://web.archive.org/web/20150109130310/http://www.fortran.com/F/index.html). Fortran.com. Archived from the original (http://www.fortran.com/F/index.html) on 9 January 2015. Retrieved 19 November 2014.
67. "Fortran for .NET Language System" (https://web.archive.org/web/20141018201259/http://www.lahey.com/lf71/lfnet.htm). Archived from the original (http://www.lahey.com/lf71/lfnet.htm) on 18 October 2014.
68. "FTN95: Fortran 95 for Windows" (http://www.silverfrost.com/11/ftn95_overview.aspx). Silverfrost.com. Retrieved 19 November 2014.
69. X3J3 post-meeting distribution for meeting held at Brookhaven National Laboratory in November 1976.
70. "The obliteration of O", Computer Weekly, 3 March 1977.
71. "Rules for Data Typing (FORTRAN 77 Language Reference)" (http://docs.oracle.com/cd/E19957-01/805-4939/z40007365fbc/index.html). *docs.oracle.com*. Retrieved 29 September 2016.

Further reading

Language standards

- Ansi x3.9-1966. *USA Standard FORTRAN* (https://web.archive.org/web/20110515143149/http://www.fh-jena.de/~kleine/history/languages/ansi-x3dot9-1966-Fortran66.pdf) (PDF). American National Standards Institute. Archived from the original (http://www.fh-jena.de/~kleine/history/languages/ansi-x3dot9-1966-Fortran66.pdf) (PDF) on 15 May 2011. Retrieved 5 May 2010. Informally known as FORTRAN 66.
- Ansi x3.9-1978. *American National Standard – Programming Language FORTRAN* (https://web.archive.org/web/20131029134137/http://www.fortran.com/fortran/F77_std/rjcnf.html). American National Standards Institute. Archived from the original (http://www.fortran.com/fortran/F77_std/rjcnf.html) on 29 October 2013. Retrieved 11 December 2007. 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* (https://web.archive.org/web/20020501111055/http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=17366). American National Standards Institute / ISO/IEC. Archived from the original (http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=17366) on 1 May 2002. Informally known as Fortran 90.
- ISO/IEC 1539-1:1997. *Information technology – Programming languages – Fortran – Part 1: Base language* (https://web.archive.org/web/20110818190618/http://j3-fortran.org/doc/standing/archive/007/97-007r2/pdf/97-007r2.pdf) (PDF). Archived from the original (http://j3-fortran.org/doc/standing/archive/007/97-007r2/pdf/97-007r2.pdf) (PDF) on 18 August 2011. Retrieved 13 December 2007. 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) (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) (PDF). Informally known as Fortran 2008.

Related standards

- Kneis, Wilfried (October 1981). "Draft standard Industrial Real-Time FORTRAN". *ACM SIGPLAN Notices*. **16** (7): 45–60. doi:10.1145/947864.947868 (https://doi.org/10.1145%2F947864.947868). ISSN 0362-1340 (https://www.worldcat.org/issn/0362-1340). S2CID 8662381 (https://api.semanticscholar.org/CorpusID:8662381).
- *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.

Other reference material

- *ECMA Standard on FORTRAN* (http://www.ecma-international.org/publications/files/ECMA-ST-WITHDRAWN/ECMA-9,%201st%20Edition,%20April%201965.pdf) (PDF). European Computer Manufacturers Association. April 1965. Retrieved 17 November 2014.
- *FORTRAN 77 4.0 Reference Manual* (https://web.archive.org/web/20120105170048/http://wwwcdf.pd.infn.it/localdoc/f77_sun.pdf) (PDF). Sun Microsystems, Inc. 1995. Archived from the original (http://wwwcdf.pd.infn.it/localdoc/f77_sun.pdf) (PDF) on 5 January 2012. Retrieved 17 November 2014.
- "FORTRAN Coding Form" (https://web.archive.org/web/20150608095341/http://www.atkielski.com/PDF/data/fortran.pdf) (PDF). IBM. Archived from the original (http://www.atkielski.com/PDF/data/fortran.pdf) (PDF) on 8 June 2015. Retrieved 17 November 2014.
- *IBM System/360 and System/370 Fortran IV Language* (https://web.archive.org/web/20110406115810/http://www.fh-jena.de/~kleine/history/languages/GC28-6515-10-FORTRAN-IV-Language.pdf) (PDF). International Business Machines. May 1974. Archived from the original (http://www.fh-jena.de/~kleine/history/languages/GC28-6515-10-FORTRAN-IV-Language.pdf) (PDF) on 6 April 2011. Retrieved 17 November 2014.
- Goerz, Michael (2014). "Modern Fortran Reference Card" (http://michaelgoerz.net/refcards/fortran_refcard_a4.pdf) (PDF). Retrieved 14 December 2014.

Books

- 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.
- Ruetsch, Gregory; Fatica, Massimiliano (2013). *CUDA Fortran for Scientists and Engineers* (1st ed.). Elsevier. p. 338. ISBN 9780124169708.
- Akin, Ed: "Object-Oriented Programming via Fortran 90/95", Cambridge Univ Press, ISBN 978-0521524087, (Feb. 2003).
- Brainerd, Walter S., Goldberg, Charles H., Adams, Jeanne C.: "Programmer's guide to Fortran 90"(3rd Ed.), Springer, (1996).
- Brainerd, Walter S.: "Guide to Fortran 2008 Programming"(2nd Ed.), Springer, ISBN 978-1447167587, (Sep. 2015).
- Chapman, Stephen J. (2018). *Fortran for Scientists and Engineers* (https://www.mheducation.com/highered/product/fortran-scientists-engineers-chapman/M9780073385891.html) (Fourth ed.). New York: McGraw-Hill Education. pp. xxiv + 1024. ISBN 978-0-07-338589-1.

- Chivers, Ian; Sleightholme, Jane (2018). *Introduction to Programming with Fortran* (4th ed.). Springer. ISBN 978-3-319-75501-4.
- Clerman, Norman S., Spector, Walter: "Modern Fortran: Style and Usage", Cambridge University Press, ISBN 978-0521514538, (Feb. 2012).
- Curcic, Milan : "Modern Fortran: Building efficient parallel applications", Manning Publications, ISBN 978-1617295287 (Nov. 2020).
- Ellis, T. M. R.; Phillips, Ivor R.; Lahey, Thomas M. (1994). *Fortran 90 Programming* (1st ed.). Addison Wesley. ISBN 978-0-201-54446-6.
- Etter, D. M. (1990). *Structured FORTRAN 77 for Engineers and Scientists* (https://archive.org/details/structuredfortra00ette_0) (3rd ed.). The Benjamin/Cummings Publishing Company, Inc. ISBN 978-0-8053-0051-2.
- Kerrigan, J. F. (1993). *Migrating to Fortran 90* (1st ed.). O'Reilly & Associates, Inc. ISBN 1-56592-049-X.
- Kupferschmid, Michael (2002). *Classical Fortran: Programming for Engineering and Scientific Applications*. Marcel Dekker (CRC Press). ISBN 978-0-8247-0802-3.
- Lorenzo, Mark Jones: "Abstracting Away the Machine: The History of the FORTRAN Programming Language (FORmula TRANslation)", Independently published, ISBN 978-1082395949, (Aug. 2019).
- Loukides, Mike (1990). *Unix for FORTRAN Programmers*. Sebastopol, CA 95472: O'Reilly & Associates, Inc. ISBN 0-937175-51-X.
- McCracken, Daniel D. (1961). *A Guide to FORTRAN Programming* (https://archive.org/details/guidetofortranpr00mccr). New York: Wiley. LCCN 61016618 (https://lccn.loc.gov/61016618).
- Metcalf, Michael; Reid, John; Cohen, Malcolm: "Modern Fortran Explained: Incorporating Fortran 2018" (5th Ed.), Oxford Univ. Press, ISBN 978-0198811886, (Nov. 2018).
- Nyhoff, Larry; Sanford Leestma (1995). *FORTRAN 77 for Engineers and Scientists with an Introduction to Fortran 90* (4th ed.). Prentice Hall. ISBN 978-0-13-363003-9.
- 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 978-0-273-02856-7. 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 978-0-521-57439-6.
- Sleighthome, Jane; Chivers, Ian David (1990). *Interactive Fortran 77: A Hands-On Approach* (https://web.archive.org/web/20140312213359/http://www.fortranplus.co.uk/fortran_books.html). Computers and their applications (2nd ed.). Chichester: E. Horwood. ISBN 978-0-13-466764-5. Archived from the original (http://www.fortranplus.co.uk/fortran_books.html) on 12 March 2014. Retrieved 12 March 2014.

Articles

- Allen, F.E. (September 1981). "A History of Language Processor Technology in IBM". *IBM Journal of Research and Development*. **25** (5): 535–548. doi:10.1147/rd.255.0535 (https://doi.org/10.1147%2Frd.255.0535). S2CID 14149353 (https://api.semanticscholar.org/CorpusID:14149353).
- J. W. Backus; R. J. Beeber; S. Best; R. Goldberg; L. M. Haibt; H. L. Herrick; R. A. Nelson; D. Sayre; P. B. Sheridan; H. Stern; L. Ziller; R. A. Hughes; R. Nutt (February 1957). *The FORTRAN Automatic Coding System* (http://www.softwarepreservation.org/projects/FORTRAN/paper/BackusEtAl-FortranAutomaticCodingSystem-1957.pdf) (PDF). Western Joint Computer Conference. pp. 188–198. doi:10.1145/1455567.1455599 (https://doi.org/10.1145%2F1455567.1455599).
- Chivers, Ian D.; Sleightholme, Jane (2013). "Compiler support for the Fortran 2003 & 2008 standards" (https://web.archive.org/web/20080516202558/http://www.fortranplus.co.uk/fortran_i

nfo.html). *ACM SIGPLAN Fortran Forum*. **28** (1): 26–28. doi:10.1145/1520752.1520755 (https://doi.org/10.1145%2F1520752.1520755). ISSN 1061-7264 (https://www.worldcat.org/issn/1061-7264). S2CID 26200779 (https://api.semanticscholar.org/CorpusID:26200779). Archived from the original (http://www.fortranplus.co.uk/fortran_info.html) on 16 May 2008.

- Metcalf, Michael (2011). "The Seven Ages of Fortran" (https://journal.info.unlp.edu.ar/JCST/article/view/681/210). *Journal of Computer Science & Technology*. **11** (1): 1–8.
- Pigott, Diarmuid (2006). "FORTRAN – Backus et al high-level compiler (Computer Language)" (https://web.archive.org/web/20091008230959/http://hopl.murdoch.edu.au/showlanguage.prx?exp=8&language=FORTRAN). *The Encyclopedia of Computer Languages*. Murdoch University. Archived from the original (http://hopl.murdoch.edu.au/showlanguage.prx?exp=8&language=FORTRAN) on 8 October 2009. 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) (PDF). *IBM Systems Journal*. **24** (1): 49–60. doi:10.1147/sj.241.0049 (https://doi.org/10.1147%2Fsj.241.0049).

External links

- ISO/IEC JTC1/SC22/WG5 (https://wg5-fortran.org/)—the official home of Fortran standards
- Fortran Standards Documents (https://gcc.gnu.org/wiki/GFortranStandards)—GFortran standards
- fortran-lang.org (https://fortran-lang.org/)—the new home of Fortran on the internet (2020).
- History of FORTRAN and Fortran II (http://www.softwarepreservation.org/projects/FORTRAN/)—Computer History Museum
- Valmer Norrod, et al.: *A self-study course in FORTRAN programing—Volume I—textbook* (https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19700015982.pdf), Computer Science Corporation El Segundo, California (April 1970). NASA (N70-25287).
- Valmer Norrod, Sheldom Blecher, and Martha Horton: *A self-study course in FORTRAN programing—Volume II—workbook* (https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19700015983.pdf), NASA CR-1478 (April 1970), NASA (N70-25288).

Retrieved from "https://en.wikipedia.org/w/index.php?title=Fortran&oldid=1026125155"

This page was last edited on 31 May 2021, at 14:43 (UTC).

Text is available under the Creative Commons Attribution-ShareAlike License; additional terms may apply. By using this site, you agree to the Terms of Use and Privacy Policy. Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a non-profit organization.