Fortran history

Fortran is one of the first high-level programming languages and has been in use for over six decades.

When the first computers appeared, using assembler instructions was the only way to program them. Such instructions are quite simple: moving data from a memory location to a register, or vice versa, doing comparisons between values in registers, and conditionally jump to other instructions, and of course arithmetic operations.

Although assembler is conceptually simple, it is quite hard to formulate high-level algorithms using such low-level operations. Also, assembler instructions as well as the memory layout tended to be different for new types of computers, hence forcing programmers to re-implement their application time and time again. People also realized that this was not an efficient way of working in a scientific context. Translating sophisticated mathematics into assembler was very time consuming, and your typical physicist or chemist had better things to do with his time.

By the way, don't get this wrong: assembler, first developed by Kathleen Booth, was a major step forward, and is still used today for performance critical code.

Meanwhile at IBM, in 1953, John Backus came up with a proposal for a programming language that would make it easy to formulate mathematical problems in. The proposal was approved and his team started working on FORTRAN. The first draft specification for the IBM Mathematical Formula Translating System was ready by November 1954, followed by the first manual almost two years later. Finally, in April 1957 the first compiler was made available.

This compiler would translate high-level code into assembler instructions. It was an optimizing compiler, since based on feedback from its user base, IBM realized that the compiler would not be used unless the performance of the generated assembler code would be comparable to that written by hand. Although many were initially skeptical about the efficiency of compilers, a program written in FORTRAN had 20 times less instructions than an equivalent assembler program, so the new technology gained traction pretty fast.

Backus once mentioned in an interview that he started work on FORTRAN out of laziness. He didn't like writing programs, so he came up with a way to reduce the effort considerably. Needless to say that laziness is a virtue for computer scientists. Later on, Backus would do very useful work on the formal representation of programming languages by defining the Backus-Naur normal form for context-free languages.

The first version of FORTRAN had a few features that are quite interesting. The conditional arithmetic `IF`-statement had three branches based on whether the arithmetic expression resulted in a negative number, zero, or a positive number. This was complemented by a `FREQUENCY` statement that could be used to pass the probability of each branch to the compiler so that it could generate more efficient code.

In those days, hard disks or terminals didn't exist, so a computer program was a deck of punched cards. An IBM punch card as used to store FORTRAN programs has 12 rows and 80 columns, but only the first 72 columns were used for data. Each column represents a single character which could be a digit, an upper case letter, or a symbol, so arithmetic operators and such. A FORTRAN statement would fit on a single punch card. Hence your hand written source code would be transcribed on a deck of punched cards using a keypunch machine that has a keyboard reminiscent of a typewriter. If you were lucky, you wouldn't have to do that yourself. Data was also entered on punched cards. In the early days, punched cards were not numbered automatically, so dropping a deck was a nightmare.

It is interesting that the format of a punch card is reflected in the formatting rules of early, and let's be honest, not so early FORTRAN. The first 5 columns are used for labels, column 6 can hold a continuation symbol, the columns 7 through 72 have the characters of the actual statement. Columns 73 to 80 are ignored. Labels are numeric, and can be used in `GO TO` statements, or to refer to `FORMAT` statements. Remember that Fortran is case-insensitive? Now your realize why that is the case: on a punched card, you could not distinguish between an upper and a lower case letter. To save space and typing, white space in the statement columns 7 to 72 was optional. These restrictions are referred to as fixed format, and it is only since Fortran 90 that you can use free format as we are so used to nowadays.

One of the features that is missing when considering the original FORTRAN specification is the lack of subroutines, and this was already addressed in 1958 by FORTRAN II. Recursive subroutines were not an option since the hardware at the time didn't support a stack for procedure calls. Another innovation was the common block, allowing for global variables. This illustrates how perception of good practices changes over time. In this course we haven't even mentioned common blocks since global variables are evil.

As an aside, a Fortran compiler in those days fit on a deck of 2200 punched cards with assembler instructions.

In 1961 IBM started the development of FORTRAN IV to eliminate machine dependent features in earlier version of the language. A few years later, the compiler was compliant with the FORTRAN specification formulated by the American Standards Association FORTRAN Working Group.

The specification approved in 1966 is the first industry-standard specification of the programming language, known as FORTRAN 66. It was largely based on FORTRAN IV, and introduced features such as a logical `IF` statement and variable names of up to 6 characters. Many intrinsic procedures were added, as well as the `EXTERNAL` keyword for library procedures.

A decade later, the next generation of the standard was approved, FORTRAN 77. This would be the last standard that spelled the name "Fortran" in all uppercase characters. It had many improvements over FORTRAN 66, block `IF`/`END IF` statements, improved `DO` loops, direct I/O, the `CHARACTER` data type, the `PARAMETER` statement to define constants and so on. It also simplified the life of the programmer since intrinsic functions such as `SQRT` would also accept double precision arguments.

Although there were official standards, many compiler developers had their own extensions. For instance, although FORTRAN 77 still didn't provide for recursive procedures, many compilers would in fact support their use.

In 1978, the standard transitioned to an ANSI standard, and some features of that specification made it into Fortran 90 years later, for example `DO WHILE` and `IMPLICIT NONE`.

Fortran 90 introduced a number of very nice features. It is the first version of Fortran to allow free-form source code, lifting the restrictions on the first 6 columns, and allowing for longer lines beyond 72 characters. Identifier such as variable names can be up to 31 characters. More importantly from the point of view of scientific computing, operations such as assignment or arithmetic could be performed on entire arrays, eliminating the need for iteration statements. It was also possible to define your own data types, to overload operators, do dynamic memory allocation and work with pointers. From a software engineering perspective, the introduction of modules was very useful as well.

Although Fortran 90 didn't remove any features, a number of them were tagged as deprecated so that they could be deleted in future specifications. Examples of this are the arithmetic `IF` statement and the computed `GO TO` (you don't even want to know what that does).

Fortran 95 introduced some quite interesting additions to the language. For example the `FORALL` and nested `WHERE` statements that can help the compiler to better vectorize code. You could also declare your procedures to be pure or elemental. The do loop with floating point values got deleted from the language.

Fortran 2003 was again a major leap forward, especially with respect to support for object-oriented programming. The introduction of streaming I/O helped to ensure that files could much more easily be exchanged with applications written in other programming languages. A start was made to standardize the interoperability with C and compliance with the IEEE 754 standard on floating point arithmetic.

Submodules were introduced in Fortran 2008 to reduce build times of large projects. It also defined coarrays for parallel programming in pure Fortran, and the `DO CONCURRENT` statement to help the compiler detect independent iterations.

The most recent specification, Fortran 2018, improves interoperability with C and parallel programming with coarrays.

When you consider the evolution of the Fortran programming language, it is clear that it mirrors best practices in programming. Statements that would easily lead to spaghetti code such as `GO TO`, computed or not, the arithmetic `IF` and the `CONTINUE` statement have been phased out, and replaced by modern language constructs designed to help the compiler generate efficient assembler. Another recurring theme has been present: define a language for scientists and engineers. Make it as easy as possible for them to translate their formulas into computer programs. Fortran is without a doubt one of the programming languages that has succeeded doing just that, formula translation.

References

- Kathleen Booth
- John Backus
- Programming with punched cards, Dale Fisk
- Punch card computer data processing overview IBM 029 Key Punch Machine