FORTRAN IS ALIVE AND WELL!

M. Metcalf
CERN, Geneva, Switzerland

1. A LITTLE HISTORY

In 1982 it is perhaps better to examine the state of health of FORTRAN, before making the bold assertion which has been assigned to my talk. In the course of such a review, we can try to extract, at the same time, the answers to three questions:
- Why did physicists start to use FORTRAN?
- Why do physicists continue to use FORTRAN?
- Will physicists always use FORTRAN?

FORTRAN was invented by a team led by John Backus, and was introduced by IBM for its 704 computer in 1957 [1]. It was both an innovatory and a revolutionary stride. Innovatory, firstly because it was the first high-level language ever to be devised; secondly, because the use of a high-level language implies the existence of a compiler, and it was, in those days, by no means obvious how to design such a program, as no theories of compiler techniques existed, and they had to be developed by the team itself.

The revolutionary aspects were also twofold: scientific programmers were released from the burden of tedious programming in octal code or assembler language, the first step in a series increasing programmer productivity; but more importantly any scientist was able to approach and use a computer, without needing either a specialist as an intermediary or to learn a difficult low-level language.

The mainstream version at that time was known as FORTRAN II, and by 1964 there were 43 different compilers running on 16 systems. This wide availability of FORTRAN meant that high-energy physicists were also able to switch from low-level languages, and in the early 1960's FORTRAN became the main programming language at CERN, in a dialect known as CERN FORTRAN, a common set among the various compilers available. The diversification of dialects led to ASA (later ANSI, the American National Standards Institute) issuing the first standard in 1966, after four years' work. This standard was the first produced for any programming language, and was at the time the longest standard ever produced by ASA.

It is interesting to consider the advances brought by the introduction of FORTRAN through the eyes of someone writing in 1969 [2]. In her history of programming languages, Jean Sammet listed them as:
- use of available hardware;
- the possibility, via the EQUIVALENCE statement, for a programmer to control storage allocation;
- the non-dependence of blanks in the syntax;
- the ease of learning;
- the stress on optimization.

The first point is evident—the language was designed to run on the IBM 704. The second point is no longer obvious today, in the era of cheap semi-conductor storage, but in the time of small core memories it was indeed a boon for programmers to be able to overlay storage areas directly. The last three points remain equally valid now but, as we shall see, the blank will reassume its significance in a future FORTRAN standard, as a separator between syntactic items.

An interesting *post script* to the early history of FORTRAN has been provided by Backus in his 1977 Turing Award Lecture. In the published paper[3], he describes how he considers that *all* programming languages (with some exception made for APL and LISP) have followed a misguided development path, determined by their close correspondence to the von Neumann computer architecture. These languages are large, clumsy and not capable of proper verification. He proposes a new class of programming systems, functional programs, which are capable of algebraic manipulation and proof, and it will be interesting to see whether his ideas have any impact on future computer and language design.

2. FORTRAN'S PRESENT STATUS

Following the publication of the 1966 standard, a new proliferation of dialects and extensions began to appear. These, and the more obvious flaws in the language, led to a new standardization effort which resulted, in 1978, in the definition of what is now known as FORTRAN 77. This is the only version of FORTRAN available on many computers systems, and has been adopted as the CERN standard for new programs as of this Summer.

In principle, the transition from FORTRAN 66 to FORTRAN 77 is relatively simple, as backwards compatibility for standard conforming programs is guaranteed by the standard, apart from a number of minor details, and two major ones: the elimination of the extended range DO-loop, and the replacement of Hollerith constants and data by the CHARACTER data type. This latter point has led to some significant problems for some types of code used in high-energy physics programs, particularly as the Hollerith data are often used in argument lists, for example as histogram titles, and are also often mixed via an EQUIVALENCE statement with numeric data, for example in I/O buffers.

3. THE MAIN NEW FEATURES OF FORTRAN 77

The new features of FORTRAN 77 are described in many books on the subject, and are also summarized in Ref. 4. Here I list only those points I consider especially useful for scientific applications:

i) the ability to declare arrays with up to seven dimensions, each with an optional lower bound, and the ability to use any integer expression as an array subscript expression;
ii) the introduction of the block-IF construct (IF...THEN...ELSE);
iii) the extension of the DO-construct to accept expressions as control parameters, to allow redefinition within the range of the loop of any variable in a control parameter expression without affecting the control of the loop, and the concept of the zero-trip loop;
iv) the introduction of symbolic constants via the PARAMETER statement;
v) the introduction of implied DO-loops in DATA statements;
vi) the introduction of the alternate RETURN;
vii) the introduction of the CHARACTER data type with its associated operators and functions;
viii) the extensions to the I/O specification (now occupying 45 pages in the standard) to include direct access files, internal files, execution-time format specification, list directed I/O, file control and enquiry, and some new edit descriptors.

4. THE PORTABILITY OF FORTRAN 77

Program portability is vital for all those working in a dispersed scientific computing community, and since the old dialects have now once again been unified, there should be a higher degree of portability with the new standard than with the old, particularly in view of the new CHARACTER data type and the introduction of direct access files. However, already new extensions are creeping into some compilers, and this trend is to be much regretted, and the use of the extensions avoided, as they will not be generally available, nor necessarily included in a future standard.

Detailed guidance on this subject is to be found in Ref. 5, but the basic rule is, as always, to stick to the standard, using compiler diagnostics and manuals, where they are provided, to give the relevant information. Reading the standard[6)] is in itself a worthwhile activity, even if it is somewhat hard going.

5. FUTURE FORTRAN—FORTRAN 8x

The ANSI committee responsible for FORTRAN standardization, X3J3, is currently engaged in a major redesign of the language. Their work and the currrent status of the revision is described in some detail in Ref. 5. Here, once again, only a few of the more significant points are listed:

i) CORE + MODULE design, allowing flexible growth of the language;
ii) introduction of free form source, with longer names and in-line comments;
iii) entity-oriented declarations;

iv) data structures;
v) new style of DO-loops;
vi) basic and advanced array processing facilities, especially useful on vector processors;
vii) precision specification;
viii) enhanced procedure CALL's
ix) BIT data type;
x) environmental enquiry;
xi) recursion;
xii) dynamic storage allocation (of local arrays);
xiii) CASE control construct;
xiv) compile-time facilities.

6. THE PAST, PRESENT AND FUTURE

Since FORTRAN was the first high-level language to appear, it is clear why physicists began to use it. The reasons why they kept to FORTRAN, rather than follow the ALGOL path, have been outlined in Ref. 7. More relevant is the actual use of PASCAL in areas where FORTRAN would previously have been the obvious choice. In 1976 an attempt to encourage the use of PASCAL in high-energy physics was made[8], but has remained unsuccessful. The reasons for this may perhaps be given as:

- the smallness of the language (e.g. no exponentiation);
- its primitive I/O capability;
- its lack of a mechanism to override strong typing;
- its lack of extended precision;
- its lack of a means to initialize variables (cf. BLOCK DATA);
- its lack of complex arithmetic;
- its lack of a means to pass variably dimensioned arrays through argument lists (except as an option);
- its lack of an interface to libraries or other languages (except in extensions);
- its lack of separate compilation (except in extensions);
- the generally poor optimization of its compilers.

We may thus conclude that whilst it is ideal as a teaching language, and for smaller-scale applications, it remains generally unsuited for large-scale scientific programming over many sites and mainframes.

When discussing the future of FORTRAN, we cannot ignore the emergence of a new language, ADA, which has been developed by the American Department of Defense for use in embedded systems, but which has become to be a general purpose language, containing a strong numerical capability. It will not be long before compilers for ADA become available, and we can expect to see strong competition between the two languages at the end of this decade.

The ADA language will be described by another speaker (Marty, these proceedings), but I want to give here a short list of its main features, as listed by Barnes[9].

FORTRAN was the first language to introduce what is known as expression abstraction, namely the ability to write directly a statement such as X = A + C(J) without having to be concerned about the allocation of registers, etc. ALGOL introduced the concept of control abstraction, whereby it is possible to write a statement such as *if* X = Y *then* A: = B *else* P: = Q, without having to worry about GO TO's and statement labels. ADA, drawing on the experience of PASCAL and SIMULA, contains a still higher level of abstraction known as data abstraction, which implies a separation of the representation of the data from the abstract operations performed upon them. This is achieved by using enumeration data types, and by hiding the details of data as 'private data' types inside 'packages'. Other features of ADA are:

- its readability;
- its strong typing;
- its provision of facilities for large-scale programming;
- its exception handling;
- its tasking features (the reason for its design);
- its generic units.

Coming back to the title I have been given, I think that nobody can deny that FORTRAN is alive. If we read the software catalogues of programs for use in scientific and engineering applications issued by some computer vendors, we find that, for instance, the 1980 DEC catalogue[10] contains, out of 331 programs, 245 in FORTRAN (74%), 27 in BASIC, 21 in Assembler, 2 in PASCAL and 34 in other languages. The larger 1982 catalogue contains 13 programs in PASCAL, showing some growth in an area completely dominated by FORTRAN. Looking at the similar IBM catalogue[11], we find in a collection of 175 programs about 150 written in FORTRAN (87%), 8 in Assembler, 5 in COBOL, 12 in other languages and none in PASCAL. We see then that in an established scientific computing environment involving commercially available programs, FORTRAN is very much the principal language.

The statement that FORTRAN is well can be deduced from a comparison with some other modern languages. In a recent comparison of PASCAL and C, Feuer and Gehani[12] list eight attributes which they consider desirable for a language used for numerical computation. To this list I have added ease of learning and efficiency of object code, in the following table. In each box, I give a tick for a feature which is fully defined in the given language, enclosed in parentheses where there is a possible hardware dependence, and a question mark where a feature is not fully defined, is unclear or is not defined but nevertheless possible by a more or less easy mean.

Of course, the weights which should be given to each of the attributes varies in a subjective way according to one's own applications, and the assignment of ticks, question marks and blanks is also an exercise whose solution will vary from programmer to programmer. Nevertheless, it is clear that FORTRAN, both present and future, comes out very well by this comparison, which ignores other features which are less relevant to scientific computing.

Table

	PASCAL	C	F77	F8x	ADA
Extended precision arithmetic		√	√	√	√
Overflow and underflow detection				(√)	(√)
Array operations		?	?	√	?
Complex arithmetic	?	?	√	√	?
Large no. of maths. functions	?	?	√	√	?
Binary I/O	√	√	√	√	?
Routine names as parameters	√	√	√	√	√
Lower bounds for arrays	√		√	√	√
Ease of learning	√	√	√	?	
Efficiency of object code	?	?	√	√	?

7. FINAL STATEMENTS

I conclude these brief remarks by making three statements which summarize my own opinion:

i) In the past, FORTRAN was usually the only high-level language generally available, and was therefore used not only for those applications for which it was intended, but also for many for which it was not wholly appropriate. This has given it a worse reputation than it deserves.

ii) FORTRAN is today the only high-level general purpose language generally available for large-scale scientific and numerical applications.

iii) FORTRAN is likely to remain into the next century as, at the very least, a special purpose scientific and numerical language for large-scale, computing intensive applications, and strengthened especially by its array processing capabilities, will be one of a small range of widely used languages in general use.

REFERENCES

1. J. Backus et al., *in* Programming Systems and Languages (ed. S. Rosen) (McGraw Hill, New York, 1967), pp 29-47.
2. J. Sammet, Programming Languages: History and Fundamentals (Prentice-Hall, Englewood Cliffs, N.J.), 1969.
3. J. Backus, CACM **21**, 8 (1978).
4. M. Metcalf, An introduction to FORTRAN 77, CERN report DD/US/11 (1982).
5. M. Metcalf, FORTRAN Optimization (Academic Press, London and New York), 1982.
6. ANSI, Programming Language FORTRAN, X3.9-1978 (ANSI, New York, 1978).
7. M. Metcalf, Aspects of FORTRAN in Large-Scale Programming, to appear in Proc. CERN School of Computing, Zinal, Switzerland, 1982.
8. I. Willers (Editor), High-Level Languages at CERN, CERN report DD/MCM/76/60 (1976).
9. J. Barnes, Programming in ADA (Addison Wesley, London, 1982).
10. DEC, Engineering Systems and Software Referal Catalog (Digital Equipment Corp. Marlborough, MA, 1982).
11. IBM, Engineering and Scientific Applications Programs available from non-IBM Sources (IBM, White Plains, N.Y., 1981).
12. A. Feuer and N. Gehani, Computing Surveys, **14**, 1 (1982).

* * *

QUESTIONS

MR. R. PIKE - BELL TELEPHONE LAB. MURRAY HILL, N.J.

- Q ---> a) FORTRAN 8x seens to be all features and no design.
  b) It is unclear how subroutines discover the type (e.g. array or scalar) of their arguments.

- A ---> a) This is due to the selection I have made, and the fact that X3J3 has worked until now on the features, but not yet on making them coherent and regular. This work is now beginning.
  b) Where information about arguments is required, an INTERFACE block will be written by the programmer or a dope vector provided by the compiler.

MR. R. CAILLIAU CERN

- Q ---> I want to add some recent information about ADA: Matrix and complex arithmetic is not in the language, but it can be defined in a package. This makes it possible to implement it either as a set of operator routines (in ADA) or to generate the appropriate code for a machine with matrix arithmetic instructions. The user would always see the same interface.
  Also, ADA may have a big definition, but it is not such a big language, FORTRAN 8x is definitely of the same size.
  As to the criteria in your evaluation of languages, I would like to see "ease of mastering" rather than "ease of learning", i.e. how easy is it for me to remember all its details once I am reasonably fluent in a language, irrespective of how easy it was to start.

Lastly, the tasking features of ADA are about the best ones I have seen, and they will influence the way we think about programming very much. Thus I would like to see tasking as one of ADA's most important features.

- A ---> **Your points are taken. About matrix arithmetic: this is true, but the advantage of FORTRAN 8x is that it is defined there in a standard fashion.**

MR. R. BOCK CERN

- Q ---> When saying FORTRAN 77 is easy to learn and ADA is not, do you not refer unconsciously to FORTRAN-experencied users?

- A ---> **I believe ADA, by its size, will be harder to learn, even for a newcomer. There are estimates of six months to re-train a programmer to use ADA.**

MR. M. TURNILL BNOC GLASGOW

- Q ---> The driving force for the introduction of new language standards must be improved productivity. What is the perceived or expected gain for the introduction of FORTRAN 77?

- A ---> **We have too little experience to judge that yet.**

MR. W. MITAROFF INST. F. HOCHENERGIEPHYSIK VIENNA

- COMMENT ---> My experience is 30%. Programs look more readable as it is easier to express problems.

- REMARK ---> I want to stress the importance of producing efficient object code. We are told that, as hardware becomes cheaper, this is no longer important - but bitter experience has shown me that the mainframe hardware to which I have access tends to be always a speed factor of 2-3 behind what I would need. If, by using another language, the execution time would increase by another factor of (say) ≈3, this would be catastrophic for the performance of our experimental analysis programs. Only FORTRAN has, as one of its primary design goals, to produce efficient object code!

- Q ---> a) what about "freezing" preliminary "desired" features of 8x, to be implemented as (optional) extensions to 77? (I am thinking of better DO-loop structures, and array operations).

  b) will a "Dynamic Memory Menagement" (like "GEM") become part of the 8x standard, or at least become an "Application Module"?

- A ---> **It is not possible under ANSI procedures for individual items to be standardized before others.**
  There is no plan to introduce dynamic memory management into the standard.

MR. R. WORDEN LOGICA LTD LONDON

- Q ---> You say that the array handling features of FORTRAN 8x are now well defined, and that manufactures of vector processors are likely to implement them in advance of the full standard. As they are useful not only for efficiency of execution but also as a programmer convenience, would it be possible to provide a preprocessor to FORTRAN 77 so that people could start using them now?

- **A ---> It could be done, but would be difficult as a full syntax analysis would be required.**

MR. S. O'NEALE CERN

- Q ---> (privately) A large number of physicists write fairly small analysis routines (which are often interfaced to a large package) which are in the throw-away category. Many of these physicists are interested in improved debugging aids in batch and interactive running. Would you comment on the thoughts of manufactures and committees on providing or extending such facilities in FORTRAN 77, 8x, etc.

- **A ---> The standards committee is not considering this issue at all Better facilities will come from manufactures in response to competition and to user pressure. It is up to users to use both these levers to improve their working FORTRAN environment. The excellent debugging facilities of the Siemens/Fujutsu compiler are to be compared with those of IBM's VS FORTRAN compiler and, of course, the VAX interactive debugger is a delight to use. Wield your purchasing power!**