Elf90 - A First Programming Language

Thomas M. Lahey , Thomas D. L. Walker
Lahey Computer Systems, Inc./Virginia Polytechnic Institute & State University

Abstract

Essential Lahey FORTRAN 90 (Elf90) is a FORTRAN 90 compiler specifically designed from a pedagogical viewpoint to provide a FORTRAN teaching/learning environment that is user-friendly without damaging the power of the language. This paper introduces the language and its design criteria. It also explores the question of what language to teach as a first programming language.

Creating a First Programming Language

Any programming instructor with teaching experience in multiple languages has probably arrived at the conclusion that:

- some programming languages are easier to teach/learn than others;
- programs written in those languages are generally easier to understand; and
- programming languages that are comparatively easy to learn and understand are less powerful

During the past thirty years, some programming languages were written almost completely from a pedagogical standpoint. Basic (and its variants) and Pascal are the two that come immediately to mind. While some advanced forms of these languages exist today, they are not considered “main stream” languages in the same genre as FORTRAN, C, and C++, at least in the field of engineering. The failure of these languages to rise to prominence is probably due to many factors but the following would be among them:

- they were not designed and used by practicing engineers
- FORTRAN serves engineers and scientists quite well

What if engineering instructors with multiple language experience got together and designed an ideal language, from both a practical and pedagogical viewpoint? How would they do it? Probably they would discuss the idea with colleagues and write a list of requirements. The authors did our version of that and here is our list, prioritized from the top down (just as in good programming style).

- The language must be modern, i.e., modern programming concepts are available, e.g., objects, encapsulation, structures, and pointers to name a few.

- There must be reasonable expectation that the language will continue to evolve.

- The language must have features that make it specifically useful to the engineering and scientific communities.

- Programmers knowing the language must be attractive candidates for jobs in industry.

- The programming environment must include an integrated editor, compiler, linker, and debugger plus a graphics package. The manual is readable.

- The language system enforces those programming disciplines commonly accepted as good programming practices, e.g., strong typing and interface checking among others.

- A text book is available.

- A minimum number of redundancies, i.e. alternate ways of expressing or accomplishing the same thing, is preferred. This would reduce class time, student confusion, text book length, etc.

- The language should be readable, i.e., a person who understands the problem being solved is able to read the program without knowing the language.

- Economical but robust language systems must be available for student use

With two exceptions, Elf90 (Essential Lahey FORTRAN 90) is a language that meets the above requirements. An Elf90-specific textbook is not in print (more than one textbook is in the works at this time), and the debugger is not integrated into the editing/compiling environment (this will be changed in version 2.00 of the compiler which will be released in time for the Fall '96 semester).

Why did we start with FORTRAN?

The computer environment for freshman in the College of Engineering at Virginia Tech is somewhat unique in that all freshman are required to purchase a PC compatible computer equivalent to a minimum specified standard and a standard set of software that is used in the College. The students enter the College via the Division of Engineering Fundamentals, the division responsible for advising the students and teaching two required courses, EF 1005 the first semester and EF 1006 the second semester. Approximately 1400 freshman entered the College last year, however some of those did not formally start the EF 1005 course until this Spring. Almost 70% of the freshman who start EF 1005 each Fall, graduate with engineering degrees. With this retention rate, the Fundamentals Division does not make impulsive changes in how or what is taught. Since 1989, the College has surveyed its faculty twice, most recently in 1994. In the recent survey the faculty chose FORTRAN over any other language or the teaching of application tools such as MATLAB, or spreadsheets by an overwhelming margin.

Although FORTRAN's popularity has dropped since the introduction of Pascal, C, and C++, it is still the principal language for extensive "number crunching" applications. While millions of lines of legacy code must be maintained and updated, new code is still written every day. Why? According to one industry leader "FORTRAN was designed with the end user in mind,...[The user] is only interested in the logic being correct. In C, the assumption is that the user wants to have control of every single aspect

of his computer. As a result, the C programmer runs into problems that the FORTRAN programmer will never see."[1] Lest we forget, FORTRAN was designed for scientists and engineers, and it was very well designed. Additionally, the regular updates to the language via the ANSI standards committees keep the language responsive to the needs of the user community and the capabilities of the hardware. Instructors with experience teaching simultaneous courses in C and FORTRAN tend to agree that FORTRAN is less difficult to learn and more forgiving, making it definitely a better first language for the average engineering student. As one anonymous instructor put it, "FORTRAN rarely allows the programmer to shoot themselves in the foot, C allows the programmer to place one foot on top of the other and get them both with one shot." One industry programmer who works in both languages stated "Programming in C is kind of quirky because its strict syntax is counterintuitive. FORTRAN, on the other hand, is quite easy for people to understand by looking at it."[2]

Admittedly, FORTRAN has lost some of its popularity with the students and some parents. During Summer Orientation, we are often asked why we are still teaching FORTRAN. The rise of the "home computer" has increased the number of parents and students who can converse easily in "computerese", but unfortunately, popular PC magazines do not publish many articles on FORTRAN. However, the release of the FORTRAN 90 standard and several compilers built to the standard may be changing that, at least in the community for which the language was designed. A good compilation of technical reasons for staying with FORTRAN 90 is contained in the following exerpt from a letter written by a research scientist to the Chairman of the Physics Department at a University considering dropping FORTRAN and moving to C++.

> "One of our largest computational physics projects is the development of advanced methods for modeling solid dynamics based on first-principle physics. This is a multi-year, multi-million dollar project....For all of these projects, we employ FORTRAN 90 as our main language. We do some development work in FORTRAN 77, C, and C++, but we are moving away from those languages as quickly as possible. There are many reasons for our choice of FORTRAN 90, but first let me say a bit about why we are not enthusiastic about C++. The biggest strength of C++ is probably the availability of relatively inexpensive and high quality C++ compilers for PCs. But that is a pretty minor consideration in our business and it is outweighed by the enormous liabilities we have observed with C++. First, we regard C++ as the weakest of the object-oriented languages. Objective C is a far more solid and well designed OOPS language, C++ is really some OOPS capability slapped on top of C. C++ is consequently extremely inefficient, inconsistent, overly large, and enormously difficult to program in. The experience of our clients mirrors our own, and in fact many DoE and DoD laboratories are finding that their headlong rush to C++ has been a hideously expensive mistake. I know of several C++ scientific coding projects there that consumed millions of dollars and tens of man-years, only to be abandoned because the resulting code was enormously inefficient on both traditional serial computers and on their large parallel super computers. Similar horror stories abound throughout the programming community at this point. Bill Gates claimed that his biggest mistake in designing their new NT operating system was adopting C++ for the graphics coding, the resulting code took years longer to write than it should have and ran terribly slow. While OOPS is a solid development in the computer-science community, I think it is fair to say that C++ is destined to be a passing fad, much like Pascal and Ada before it. The main reason C++ has attracted the attention it has in the scientific community is because FORTRAN 77 was a terribly outdated language. The many weaknesses of FORTRAN 77 were solved with FORTRAN 90 however. FORTRAN 90 has every feature in C that is important to scientific programming and most of the features of an object-oriented language (it lacks only inheritance and that is likely going to be added in FORTRAN 2000). However, unlike C and C++, FORTRAN 90 is designed to generate executable codes that are highly optimized and thus run extremely fast. An example is pointers. Pointers are integral to C and C++ programming and because the compiler cannot determine whether a pointer is aliased, it is impossible for it to determine inter-procedural dependencies. The result is a significant degradation in optimization and extremely slow execution speeds (for most scientific codes, C and C++ generally produce code which is commonly an order of magnitude slower than FORTRAN 90 codes, based on the benchmarks we and others have done). FORTRAN 90 pointers are designed to give the functionality of pointers, but with restrictions that

> eliminate issues such as aliasing. From a programming perspective however, an even more important point is that FORTRAN 90 has more natural ways of expressing the functionality that C and C++ require pointers to express. Because of this, FORTRAN 90 is a more natural language to program in and the time required for debugging codes is a fraction of that required by C and C++....Another important point is that the time required to learn FORTRAN 90 is much less than the time to learn either C or C++....[The] issue is what freshman should learn as their first language and for that I would recommend FORTRAN 90 hands down.....It is also much more likely to be the language students will be employing in their jobs upon graduation and it is the most promising route currently developing for the programming of parallel computers."[3]

FORTRAN 90

FORTRAN 90 is not FORTRAN 77 plus some new features. Rather, FORTRAN 90 is a **new** scientific programming language **plus** FORTRAN 77.

Features:[4]

- Derived data types ("structures") - with components that can be of intrinsic or another derived type
- Modules - collections of type, interface, and procedure definitions and data. Has facility to declare an item PRIVATE, thus restricting its availability to within the module
- Procedures - may be called recursively, dummy arguments may be declared with INTENT IN, OUT, or IN OUT
- Kind parameters - the five intrinsic types have a "kind" parameter to obtain required precision
- Pointers - all variables, including arrays, may have the pointer attribute with automatic dereferencing, the pointers feature "strong typing" in that they include both a type and a rank (number of dimensions) thus increasing safety over the "C" implementation
- Variable names may have up to 31 characters including "_"
- The logical operators .LT., .GT., .LE., .GE., .EQ., and .NE. have the alternative spellings <,>,<=,>=,==, and /=
- A new CASE construct and a non-indexed DO loop construct
- Intrinsic functions - all appropriate FORTRAN 77 intrinsic functions have been extended to be applicable to arrays also, many new functions including 25 specifically for arrays such as matrix multiplication, transpose, dot product, and others
- Intrinsic subroutines including a random number generator
- Dynamic memory allocation - allocatable array size and lifetime under programmer control
- Whole array expressions and assignments are allowed, i.e COS(A) where A is an array
- Array sections are allowed
- WHERE statement applies a conforming logical array as a mask on individual operations in an array assignment, i.e. WHERE (A > 0) B = LOG(A)
- Intrinsic functions may be array-valued
- Assumed shape arguments, i.e., dummy arguments assume the length/dimensions of the actual arguments

Elf90

Elf90 is the acronym for Essential Lahey FORTRAN 90. Lahey began creating Elf90 early in 1995 primarily as an "educational version" of their ANSI standard FORTRAN 90 compiler. However, unlike most educational versions, this was not to be a less powerful language but rather a language that was tuned to enhance its teaching/learning.

- Obsolete features were removed (those features identified by the FORTRAN 90 and pending 95 standards but still included in the language to provide FORTRAN 66 and 77 backwards compatibility in FORTRAN 90).

- Redundant features of the language were removed with a view to move towards requiring a more structured approach to programming while minimizing the number of statements that students are required to learn.

- Finally, specific features of FORTRAN 90 were changed from optional to required when they increased learnability, teachability, and/or proper programming techniques.

The compiler includes an integrated editing, compiling, and linking environment.

The Elf90 Experience at Virginia Tech

Virginia Tech began using the Elf90 compiler in the Fall of 1995. It was used to teach approximately 1150 students in 38 different sections of EF 1005, "Introduction to Engineering." Approximately half of this semester course is devoted to FORTRAN. In prior years, WATCOM Incorporated WATFOR-77 was employed. The WATFOR-77 compiler is an excellent FORTRAN 77 learning environment and includes an integrated editor, compiler, and debugger. It requires minimal machine resources and can even be run from a single 3.5" floppy disk. However, as an educational version, it is source-code-size limited and it does not include the advance features available in FORTRAN 90. While the editor in the initial version of Elf90 is not as "user friendly" as that experienced with WATFOR-77, it is an integrated editing, compiling, and linking environment. It is not limited by source-code-size, and any program written in Elf90 will run under ANSI FORTRAN 90; while the opposite is not necessarily true. Additionally, because Elf90 is a full 32-bit FORTRAN compiler, it requires a more capable machine. It must be installed on the hard drive and requires eight megabytes of RAM to operate realistically. None of these negatively impacted the FORTRAN instruction in the Fall because the minimum freshman machine requirement was a PC compatible with Intel 486DX2-66 processor, 12 megabytes of RAM, and a 540 megabyte fixed disk.

Purely from a FORTRAN pedagogical environment the Elf90 compiler has had a postive impact. After the faculty became accustomed to the coding requirements of Elf90 as opposed to those in FORTRAN 77, classroom lectures, homework, and tests were approached in a similar fashion to that when using FORTRAN 77. Although seemingly trivial, the requirement to use the IMPLICIT NONE statement probably saved many manhours. Additionally, because Elf90 (as opposed to FORTRAN 90) is not backwards compatible with FORTRAN 77, the faculty were forced to get up-to-speed on the capabilities of FORTRAN 90 immediately and use some of the power of the new language. Finally, the diagnostic messages given by the compiler are much clearer than those of WATFOR 77 and the integrated environment is better designed for troubleshooting. Unfortunately, because only half of the course is devoted to FORTRAN, many of the capabilities of FORTRAN 90 could not be addressed. As faculty become more familiar and comfortable with the language, many of those features will be covered.

Elf90 Version 2.00

The second version of the Elf90 compiler is scheduled for an August '96 release. This major update will include:

- **ED** - a state of the art Windows editor with FORTRAN-language sensitivity. Within ED, the programmer will be able to invoke either the compiler or the source debugger. When ED invokes the compiler, it will capture the compiler diagnostics and use that information to navigate through the source code file to where the compiler has diagnosed errors.

- **SOLD** - Source On-Line Debugger. This interactive debugger facilitates the user almost being able to treat the program as if the language system was an interpreter.

- **Diagnostics** - FORTRAN 90 plus Elf90-specific diagnostics that clearly inform the user of the error that has been detected.

- **Windows Environment** - All of the elements of the compiler will be integrated into a Windows 3.1/95 environment.

Conclusions

With the release of compilers built to the FORTRAN 90 standard the FORTRAN language will continue to be the primary programming language of choice for the engineering and scientific communities as long as we do objective analysis of the available languages and avoid the marketing trap. To accomplish the move to FORTRAN 90, Elf90 offers the best combination of the power of FORTRAN 90, a pedagogically "tuned" language, integrated programming environment, and low cost.

---

[1] "Tenacious FORTRAN finds a way in the '90s", PC WEEK, Sept. 27, 1993, pg 123
[2] Ibid, pg 126
[3] Dr. John K. Prentice, Quetzal Computational Associates, 3701 San Mateo N.E., Suite I, Albuquerque, NM 87110-1249 as reported in *FORTRAN Forum*, March 1995, pgs 34-35
[4] "Why FORTRAN 77 programmers should switch to FORTRAN 90", John Reid, Atlas Centre, Rutherford Appleton Laboratory, Didcot, Oxon, OX11 0Qx, UK

THOMAS M. LAHEY, CEO, Lahey Computer Systems, Inc. Mr. Lahey earned his B.S. in Engineering Physics and M.S. in Mathematics from the University of Illinois. Prior to founding Lahey Computer Systems, he held programming positions at General Electric, Control Data Corporation, and Digitek. Mr. Lahey developed and licensed the GE and DTSS time sharing FORTRAN systems and is an active member of the FORTRAN 90 standards committee.

THOMAS D. L. WALKER, Associate Professor of Engineering Fundamentals at VPI & SU. He earned his BSEE degree from Purdue University and his MSME from the Naval Postgraduate School. He is interested in the restructuring of the education industry and the use of current technology to empower the student to learn outside the formal classroom environment.