The role of the language standards committee.

Neil W. Rickert,
Department of Computer Science,
Northern Illinois University,
DeKalb, IL 6∅115
Bitnet: T9∅NWR1@NIU

Background.

A few years ago the COBOL standards committee released a proposed new standard (the COBOL 85 standard). This proposal generated intense reaction from the COBOL programming community, where MIS managers protested that a tremendous investment in existing software was threatened. Eventually there was a meeting of minds and a standard was adopted which most of the MIS community found acceptable.

We are now in a similar position with the FORTRAN language. The FORTRAN standards committee has released its report on the proposed Fortran 8x. A final standard has yet to be announced, and we are currently in a period of public comment on the proposed standard. But already a number of vendors (DEC, IBM, DG, UNISYS) have opposed the new standard [1], and may refuse to implement it, because they believe it to be unacceptable to their user communities.

When the C language was under review, at least one user protested the very idea that the language be subject to standards review [2], expressing the concern that the standards committee might destroy a fine language.

What is it about the language standards review procedures which raises so much heat?

A STANDARDS standard.

In this article I propose a set of guidelines for the various STANDARDS committees. If you will, a standard for STANDARDS. My guidelines are by no means intended to be definitive, but are presented so as to stimulate further discussion. Like any guidelines they are not completely precise. Interpreting them will require human judgement. There will be times when honest minds will disagree as to whether the guidelines have been broken. There is also some overlap in the various guidelines, but while this is inevitable it is also useful, for each guideline can help us develop an interpretation of the other guidelines.

1. *Changes proposed by a standards committee should be evolutionary, not revolutionary.*

When the changes proposed to a language are so substantial as to effectively create a new language, then it should be released as a new language, and not hidden under the pretense of being a new standard. For example, the PL/I language, although based partially on FORTRAN, did not pretend to be just a new version of FORTRAN. Instead it was presented as a new language which might replace FORTRAN. Likewise ADA was not presented as a new version of PASCAL, but was described as a completely new language.

When a major modification of a language is presented as a new language, rather than as an upgrade, the users have the opportunity to decide for themselves whether the new language is better suited to their purposes. In the case of PL/I the scientific programmers have clearly decided against acceptance. For ADA it is as yet too early to know.

Much of the brouhaha about the proposed Fortran 8x standard is due to the perception of FORTRAN programmers that the changes are revolutionary rather than evolutionary; that the proposed changes significantly alter the nature of the language.

2. *The new standard must respect the philosophy and tradition of the language.*

Each language develops its own philosophy. To some extent this philosophy is determined by the language standard. However some of its philosophy develops as a tradition amongst users of the language. A new standard should not attempt to overturn this philosophy; for by doing so it would surely be providing a revolutionary change rather than an evolutionary one.

Thus part of the philosophy of PASCAL is that the compiler should strictly enforce array bounds, data types, etc. The C language, on the other hand, provides the mechanism of casting to override data types, and a tradition has developed to use this ability freely when it would be a useful programming technique. Most PASCAL programmers would be quite unhappy with a new standard which provided all of the casting facilities of C. Almost all C-programmers would be angry at an attempt to remove their freedom and replace it with the error checking of PASCAL. Similarly, although ADA supports the same strong typing as PASCAL, any attempt to replace PASCAL by ADA in a new standard would be vigorously resisted; for PASCAL is philosophically a simple language, while ADA is a highly complex one.

Programmers and computer scientists often seem to support the philosophy of their favorite language with an almost religious zeal. However a standards committee must take a more ecumenical approach. While it may be unnecessary for a member of a standards committee to zealously support the philosophy of the language, he must at least accept and respect that philosophy if the committee is to develop sound standards.

Here we see some of the problems which have lead to the extensive criticism of Fortran 8x. For while FORTRAN is philosophically a simple language, Fortran 8x is a complex one. FORTRAN is a language which gives relatively low level access to the host machine; Fortran 8x is a language emphasizing higher level constructs which serve to isolate the programmer from the underlying hardware. The DO LOOP is central to the FORTRAN tradition; yet Fortran 8x attempts to replace it with a somewhat different construction, marking the old format as deprecated (i.e. obsolete).

3. *Changes in a language should reflect the needs of the users, rather than be changes imposed by the standards committee.*

This is the guideline most easily subject to misinterpretation. After all given almost any change to a language we can probably find a user who proposes it. This obviously is not the intended meaning.

The role of the language standards committee.

The purpose of a standards committee is primarily to develop a standard. That is, a definition of the language which everyone can agree to. From time to time deficiencies become apparent in a language, and different compilers attempt to correct the deficiency by providing language extensions. The occurrence of language extensions thus provides a clear (but not necessarily decisive) indication of a user identified need. If the need is identified as being of importance to the language, the standards committee must step in and develop a standard solution, so that the language does not fragment into many incompatible versions.

Sometimes needs become apparent from comments in articles, letters to the editor, viewpoints, etc, expressed in the computer science literature. These comments may merely point out a language deficiency, or they may suggest possible solutions. Sometimes these comments are not intended to be taken seriously. It is up to the standards committee to sort through such comments and attempt to discover which of them reveal serious problems which must be addressed.

It is essential that proposed changes to a language be discussed in detail with users of the language well before the standards committee makes any firm decisions. It is not enough to merely ask "Would you like it if such and such a feature is added?" For by nature few users will oppose free extras thrown in. It is important that users also be informed of how the proposal would affect the structure of the language, the compilation speed, the run time speed, the compatibility with existing code, etc.

4. *Standards must attempt to preserve the investment in existing software written in the language.*

The importance of this guideline should be obvious. It is best interpreted in connection with the next guideline. There is an obvious corollary; it is much easier to change a relatively new language with little existing software than it is to change a mature language for which a great deal of software is in use.

5. *New standards should maintain as high a degree of compatibility as possible with previous standards.*

It is possible to define a number of different levels of compatibility.

(a) Object code compatibility. Object code produced by earlier compilers can be compatibly link-edited with new programs. Typically this implies that the new standards can be implemented without substantive changes in the subprogram linkage and parameter passing conventions, or at worst that any changes in these conventions will be invisible to preexisting subprograms.

(b) Source code compatibility. This means that source programs produced for prior standards will correctly compile under the new standard, and the newly compiled program will preserve the functionality of the original version.

(c) Automatic convertibility. A program written for a prior standard can be automatically converted to the new standard by processing it with a software package made available with the new standard.

(d) Semi-automatic convertibility (convertibility with a minimum of human interaction). A software package is made available which will do most of the conversion to the new standard. However it may occasionally prompt the user for further information before making the conversion. The information required should be readily available, and not require complete understanding by the user of a complex program being converted.

The appropriate degree of compatibility depends greatly on how the language is used. For example the COBOL language is primarily used for writing complete programs, while FORTRAN has developed large libraries of scientific subroutines. The degree of compatibility required for these two languages is thus very different.

For COBOL, source code compatibility is more than adequate. Indeed automatic convertibility is sufficient. For since primarily only complete programs are involved in a standards change, existing programs will continue to run unchanged. Only if there is a need to modify the program does it become necessary to recompile the program in accordance with the new standard. The cost of running an automatic conversion program only on software which is about to be modified is a relatively modest cost. Indeed if the compiler for the new standard has an option which allows it to compile to the previous standard, then even this cost can be avoided if only minor code changes are needed. In such a case semi-automatic convertibility may even be acceptable.

FORTRAN however, is in an entirely different category. Here much of the investment in existing software lies in the scientific subroutine libraries. Thus object code compatibility is highly desirable, for it means that new software compiled under the new standard can continue to use existing libraries. If only source level compatibility, or automatic conversion compatibility are available, a FORTRAN shop will have to bear a heavy burden of locating all of the source code for library routines, converting to the new standards if needed, recompiling, and then rebuilding the libraries. Worse still, if the libraries use a few subroutines written in Assembler Language (as is commonly the case where high performance is desired), the Assembler code will need to be changed to meet any new linkage conventions which may be required for compatibility.

By way of example, almost all of the changes from FORTRAN 66 to FORTRAN 77 could be implemented preserving both object code and source code compatibility. The one possible exception was the handling of character string data type. Most of the changes were internal to a procedure, such as those related to improved IF statements. Since the old standard was a subset, source code compatibility was automatic. With object code the one possible problem was with character string data. Prior to FORTRAN 77, character data was handled by reading it into integer arrays. The 77 version provided a character data type. The compatibility difficulty arose when a program compiled for FORTRAN 77 wished to call a subroutine compiled under the earlier standard, and to pass a character string constant as an argument. Had FORTRAN 77 implemented character strings as arrays of one-byte integers (as in the language C) there would have been no problems here. However a character data type was implemented in such a way that the character string length needed to be passed as an additional implied argument to the subprogram. I know that at least one vendor went to great lengths, involving several upgrades to its FORTRAN 77 compiler, in order to reach an implementation which maintained object code compatibility - evidently this vendor believed such compatibility was highly important.

If we assume that the deprecated features are to be eventually removed, Fortran 8x makes many changes to FORTRAN 77 which fail all of our tests of compatibility. As an example it marks the EQUIVALENCE statement as a deprecated feature, implying that it may be eliminated in future versions. Since the EQUIVALENCE statement is used to serve so many different functions it would be quite impossible to write automatic or semi-automatic software to convert programs.

Conclusions:

Some guidelines are needed for the language standards review procedures. We have proposed a possible set of guidelines. Much of the controversy surrounding the adoption of the COBOL 85 standard was due to the standards committee stretching some of these guidelines to their limits. The current controversy over the proposed Fortran 8x standard is related to the extent to which the proposal violates all of these guidelines.

References:

1. News item, "Fortran 8X Revision Raises Questions Over Scientific Upgrades", Computers in Physics, Nov/Dec 1987, pp. 12-14.

2. C. Finseth, "ANSC X3 C Language Standards Project", SIGPLAN Notices, November 1983, p. 18.