K.H. Rotthäuser 1987-07-27

German Position on Fortran 8x/S8.104

We would like to make some comments about the proposed features for Fortran 8x.

+ We believe that the BIT DATA TYPE which has been removed from the draft proposal should be incorporated directly into the standard. In addition, we feel that the related BIT Functions should be reinstated in the document. Real time applications; in particular» require these features.

+ The statement FORALL and "vector valued subscripts“ should be reinstated in the document. These features have been implemented in numerous Fortran Compilers and should therefore be standardized.

+ The DIN/WG Fortran believes that significant blanks are logically associated with free source forms and should be introduced at the same time as the free source form is introduced. The presence of significant blanks in Fortran 8x will provide greater flexibility and safety for the future development of the language.

+ We feel that pointers are important: and we urge that they be incorporated. The combination of pointers, derived data types and dynamic storage allocation would permit the implementation of graph- and list-processing algorithms in Fortran. Many other programming languages» already have pointers.

+ We are very disappointed that Fortran 8x does not include a stream-oriented I/0 method. The problem is to provide a user friendly, portable way for a Fortran program to read and write a stream of characters on a terminal, screen etc..

- Variant Structures should be removed as intended by X3J3 because their application is limited.

- The features IDENTIFY and RANGE should be unified because these two facilities can be used to solve some of the same problems.

- Remove "passed-on" precision REAL(∗, ∗)

∗ Current draft

The current draft has one REAL type and one COMPLEX type: which correspond to the mathematical real and complex numbers. Because entities of these types can only be approximated in a computer» the minimum approximation requirements for the representation of real entities are specified (parameter PRECISION and EXPONENT_RANGE). These are used to select a native floating point representation. Complications occur with argument passing: The rule is that if type parameters are declared explicitly for the dummy argument then only actual arguments with identical explicit parameters will match.
If a procedure has dummy arguments which are declared with asterisks for PRECISION and EXPONENT_RANGE the processor can implement REAL(∗,∗) by generating overloaded versions of the procedure, one for each native floating point data representation supported.
The generic feature for intrinsics is also based on the native floating point representation.

* Disadvantages of REAL(*,*)

REAL(*,*) can indeed be implemented very cheaply: if REAL(*,*) is interpreted simply as REAL with the highest effective precision. This implementation, however, will cause an extreme loss of performance (for Siemens and IBM type computers, e.g. the instructions for 16 Byte-REAL's may be 4 times slower than corresponding instructions for 4-Byte REAL's).
Therefore users will tend to avoid the REAL(*,*) feature altogether.
Furthermore, we don't really believe that the feature REAL(*,*) is actually sufficient to cover customers' needs: the intrinsic functions of the FORTRAN library, for example, have an entirely different implementation for the different precisions: different degrees and coefficients of the polynomials, different size of the intervals used for approximation, special tests to ensure accuracy.
Hence, we assume that users may also wish to choose different algorithms for different precisions.

* Proposed changes for user-defined overloading to replace REAL(*,*)

Change the rule for association of actual and dummy REAL arguments of a procedure as follows: If type parameters are declared explicitly for the dummy argument then all actual arguments with identical effective parameters match this dummy argument. The effective parameters depend on the floating point representation.
If this is implemented then we think that user-defined overloading would be an efficient and perfect way to satisfy all user needs: the user could decide himself for which (explicit) precision he wants to offer different algorithms, and the compiler would select the suitable one.
No conflicts would arise with the previous types as DOUBLE PRECISION, which can be mixed.
This solution would also conform to the generic feature which is based on the native floating point representation of data.