Date: 1998/02/18
To: J3
From: interop
Subject: Interoperability with C: Specifications

After some consideration, the interop subgroup feels that, in order to be effective, an interoperability feature between Fortran and C must meet the following criteria.

- The binding must be to the new C standard (C9x) upon which J11 and WG14 are currently working. This standard is scheduled for adoption prior to the completion of the current revision of the Fortran standard, and it is very likely that there will be new implementations and C programs that will adhere to C9x before the Fortran standard is published.

- It should allow Fortran programmers to write interface bodies for arbitrary C functions in Fortran, without requiring recourse to "wrapper" routines.

- A method of specifying the *bind-time* name of a procedure defined in C is required, since, in Fortran, names are case insensitive.

- Some method of describing the following C data types for the interface to a procedure written in C and for globally accessible C data must be supported: **int**, **short**, **long**, **long long**, **signed char**, **unsigned int**, **unsigned short**, **unsigned long**, **unsigned long long**, **unsigned char**, **float**, **double**, **long double**, **complex**, **double complex**, **long double complex**, **char**.

- Some method of describing C **struct** data types must be provided for both the interface to a procedure written in C and for globally accessible C data.

- Some method of describing C **array** data types must be provided for both the interface to a procedure written in C and for globally accessible C data.

  Note that no method of transforming an array from row-major to column-major, and vice versa, need be provided. If a user finds the differences in the array notations in the two languages sufficiently confusing that such a transformation is desired, Fortran already provides sufficient facilities to make the transformation "manually".

- Some method of describing C **pointer** types to all of the above data types must be provided - not just as dummy arguments to procedures defined in C, but also as members in **struct**s and the types of **array** elements.

  It may be sufficient to assume that all **pointer** types have the same representation (which is not guaranteed by either the current C standard or C9x). There is some risk that this might prevent some processor from being standard-conforming, particularly, processors for embedded systems. However, it would avoid some of the problems encountered in the Interoperability PDTR with supporting the rich (and infinite) variety of pointer representations available to a standard-conforming C processor.

- Associated with the preceding requirement is the requirement that some method of enquiring about the address of a Fortran object (with some restrictions) be provided. Many C functions have parameters of type **void ***, that are provided to allow pointers to arbitrary types of objects to be passed. The Message Passing Interface (MPI) is an example of a facility that uses this, which makes it impossible to specify a standard-conforming interface to most MPI procedures. Being able to specify such interfaces should be a goal of this feature.

- Support for function pointers. Many C functions accept parameters that act as *call back* functions. That is, the caller of the function must provide a function that is used to perform some sort of initialization action, or perhaps a per-element computation action.

  (Aside: A straw poll was held on whether this was required function, and the result was 17-0-0.)

- Direct support for C functions that have ellipsis parameters must be provided.

- C programmers very often use **typedef**s to hide the underlying definitions of objects. For example, on one system the underlying data type used as a function parameter might be **int** while on another it might be a **struct**.

  In order to provide a straightforward, portable way of being able to handle both cases in the Fortran interface to such a

procedure, a similarly functional type definition system would be required. Such a facility would benefit Fortran, in general, in the same way that C benefits.

- Even though direct support of the **setjmp** and **longjmp** facilities is not required, there may be implications for a Fortran program that uses a C function that uses these functions.

The following additional sub-features may be included, if the development body finds that time permits their inclusion.

- A method of directly de-referencing or manipulating (via pointer arithmetic) C **pointer**s in a Fortran program could be provided. Without such a facility, it may prove difficult to share abstract data structures between the two languages; a user would have to write C routines to be able to de-reference such an object or to store the address of a Fortran object, although these would be likely to be very short C routines. On the other hand, this may have a significant impact on the Fortran standard.

  (Aside: A straw poll was held on whether this should be a required feature, and the result was 5-0-13. For this reason, work on this feature will be deferred.)

- Support for **union**s. The C standard places certain requirements on the layout of storage in **union**s that makes it extremely difficult to simulate this feature without direct support in the Fortran standard. Many languages provide such a feature, and it is likely that such a feature would prove useful to many Fortran users. However, this may be a somewhat large feature to integrate into Fortran.

- Support for **bit fields** in **struct**s. There has been a long-standing requirement for support for a bit data type in Fortran. However, there has been considerable difference of opinion about the precise semantics required. Including such a feature may have a significant impact on the standard, and may not fully satisfy the requirement for a bit data type.

- Support for **enum**s. This could be limited to providing a way of declaring an integer type that is compatible with the particular **enum** type, or it could be full-blown support in Fortran for **enum**s.

Features that will not be supported.

- Support for **pragma**s. The Interoperability PDTR made an attempt to provide support for **pragma**s that might affect things such as storage layouts. This was done by associating a *scalar-char-init-expr* with interface bodies for procedures defined in C, with dummy arguments for the same, and with variables associated with globally accessible C data.