****
J3/02-229

Date: July 15 2002
To: J3
From: Aleksandar Donev
Subject: Enhanced C_LOC: Pointer Arguments
Reference: Paper J3/02-230
****

Summary

I propose a modification of the specification of C_LOC from ISO_C_BINDING to allow associated scalar pointers of interoperable type and type parameters as an argument. This corrects for the (unjustifiable) lack of this functionality in the current draft.

I hope the modification will be accepted at meeting 162, which I will attend.

I would like to thank and acknowledge the help of Richard Maine and John Reid.

Motivation

Fortran pointers are not interoperable with C and thus not allowed as arguments to C_LOC. However, allocated allocatables are, in which case C_LOC returns the C address of the storage associated with the allocatable array. It only seems reasonable to allow the same for associated scalar pointers, i.e. C_LOC would return the C address of the target of the pointer. Array pointers should be excluded because the variables they are associated to might not be interoperable with C (for example, be noncontiguous array sections).

Possible alternatives

Note that it is possible for a user to make a wrapper around C_LOC to do what I propose in a perfectly conforming way for scalar pointers (but not for array pointers since assumed-shape arrays are not interoperable and cannot be arguments to C_LOC):

```
TYPE(C_PTR) :: c_address
INTEGER, POINTER :: variable

ALLOCATE(variable)
c_address=C_LOC_Integer(variable)
   ! We cannot do c_address=C_LOC(variable) directly

CONTAINS

   FUNCTION C_LOC_Integer(variable) RESULT(c_address)
      USE ISO_C_BINDING
      INTEGER, INTENT(IN), TARGET :: variable
         ! Must be TARGET
      TYPE(C_PTR) :: c_address ! C_LOC(variable)

      c_address=C_LOC(variable)
   END FUNCTION C_LOC_Integer
```

The big disadvantage of this is that we are forcing the user to write a separate wrapper for each TK combination he needs for no reason. This would become particularly painful when interoperable derived types are used, since one cannot make a generic wrapper of C_LOC for all interoperable derived types one might use.

Edits:

These are just excerpts from the accompanying paper number 02-229 which gives more elaborate edits with explanations in the form of self-notes. Please understand that this is my first-ever attempt at edits! I do not see any other changes other then in the description of C_LOC as the proposed modification is really simple.

382: 20-22 Replace with:
Argument. X shall be a procedure that is interoperable, a procedure pointer associated with an interoperable procedure, a variable that has the TARGET attribute and is interoperable, an associated scalar pointer that has interoperable type and type parameters, or an allocated allocatable variable that has the TARGET attribute and has interoperable type and type parameters.

382: 24-25 Replace with:
Result Value:
If the argument X is an interoperable procedure or an interoperable variable, the result is the value that the target C processor returns as the result of applying the unary "&" operator to X, as defined in the C standard, 6.5.3.2.
If the argument X is a procedure pointer or a scalar pointer, the result is as if the target of the pointer had been passed instead of X.
[See self-note 6 in J3/02-230]
If the argument is an allocated allocatable array, the result is the the value that the target C processor returns as the result of applying the unary "&" operator to the first element of the array in array element order.
