****

J3/02-313r2

Date: October 2002
To: J3
From: Aleksandar Donev
Subject: Corrections to C_LOC and C_F_POINTER
Reference: J3-007R3, J3/02-230r2

****

Summary

Richard had some objections to the edits made in 02-230r2 in the section on the C_LOC and C_F_POINTER intrinsics, which I address in this paper, and propose some edits. Also some questions which need to be addressed by all of J3 are given.

Nontrivial issues

Some of the comments are not directly resolved or addressed:

The current wording of the workings on C_LOC and C_F_POINTER is convoluted and unclear to the outsider. Therefore more work is needed to clarify the formulation. Some edits to correct this and some plain omissions are given below.

384: 17-19
Richard thinks we need to determine whether we really mean to impose the requirement that the target of PX "shall not have been deallocated". With ordinary pointer to pointer assignment PX=>PY, PY can be undefined and the assignment is legal but the results are not. We could do the same here, since we are now pointing a Fortran pointer to a Fortran pointer, FTPR=>PX, indirectly (i.e., via a C pointer CPTR).

However, since we have a very strong requirement on CPTR in Case (i), i.e., we do not allow the C pointer to be invalid or null or anything like that, I think we ought to keep the same strong requirement here. Of course, no compiler will ever be able to check any of this anyway.

Edits

NOTE: The functionality remains within the agreed specification from meeting 162.

382:17-385:1-
Alphabetize the descriptions of the three procedures and give them subclause numbers 15.1.2.1-3.

383:2 Delete the redundant ‘‘scalar’’.

383: 3 Add
Result Characteristics. Scalar of type C_PTR. Let the dummy name of this result be CPTR.
and replace
"Result"
with
"Result Value".

383:3+ Add the inadvertently omitted procedure pointers to the description, and clarify the meaning of PX:

The result is determined as if C_PTR were a derived type containing a private pointer component PX with the appropriate type, type parameters, and rank, and the pointer assignment CPTR%PX => X were executed.

If X is interoperable, has interoperable type and type parameters, or is a procedure pointer, then the result is the value that the C processor returns as the result of applying the unary "&" operator (as defined in the C standard, 6.5.3.2.) to the target of PX.

If X is scalar, the result is a value that can be used as an actual CPTR argument in a call to C_F_POINTER where FPTR is scalar and has the same type and type parameters as X. Such a call to C_F_POINTER shall have the effect of the pointer assignment FPTR => PX.

383: Add Note 15.3-:
The effect of the private component PX and the model pointer assignment CPTR%PX => X is to emphasize the similarities in behaviour between Fortran and C pointers. In particular, 12.4.1.2 applies to the definition status of the result CPTR when X is a subobject of a dummy argument.

383: Note 15.3
Delete ‘‘base’’

384: 5-20 Reorganize, say what happens for procedure pointers, which seem to have been omitted, and prohibit polymorphic FPTR:

CPTR shall be a scalar of type C_PTR. It is an INTENT(IN) argument. It’s value shall be:
(1) The C address of a procedure whose C prototype is interoperable,
(2) The C address of an interoperable data entity, or
(3) The result of a reference to C_LOC with a noninteroperable argument.
The value of C_PTR shall not be the C address of a Fortran variable that does not have the TARGET attribute.

FPTR shall be a pointer. It is an INTENT(OUT) argument.

Case (i): If the value of CPTR is the C address of an interoperable procedure, then FPTR shall be a procedure pointer whose interface is interoperable with the prototype that describes that procedure. In this case, F_PTR becomes pointer associated with the target of CPTR,

Case (ii) If the value of CPTR is the C address of an interoperable data entity, then FPTR shall be a data pointer with type and type parameters interoperable with the type of the entity. In this case, FPTR becomes pointer associated with the target of CPTR. If it is an array, its shape is specified by SHAPE, and each lower bound is 1.

Case (iii): If the value of CPTR is the result of a reference to C_LOC with a noninteroperable argument, then FPTR shall be a nonpolymorphic scalar pointer with the same type and type parameters as that target. In this case, that reference to C_LOC behaves as if a pointer assignment CPTR%PX => X were made, as described in (REFENCE to C_LOC section), and the association status of PX has not changed since the reference to C_LOC. The target of PX shall not have been deallocated or have become undefined due to execution of a RETURN or END statement since the reference to C_LOC. FPTR becomes pointer associated with the target of PX.

384: Note 15.5
Delete this old and incorrect note.

474:2
Replace
"real numbers, not supported"
with
"real numbers, which may not be supported"

474: 31 Replace with new syntax for deferred bindings:
PROCEDURE, DEFERRED, PASS(STREAM) :: NEXT=>RANDOM_UNIFORM

475:12 Replace
"URNG"
with
"uniform random number generator"