Date: 26 June 1998
To: J3
From: Van Snyder
Subject: Discussion paper – Using POINTER(C) for C interoperability
References: 97-108, 97-123, 98-165r1

1 Introduction

Using POINTER(C) for several purposes, for which papers 97-108 and 98-165r1 proposed separate mechanisms, would simplify C interoperability, while simultaneously expanding its expressive power.

POINTER(C) can provide the functionality of and replace the need for the following facilities proposed by paper 98-165r1:

- The type C_PTR
- A VALUE annotation for dummy arguments
- A LOC intrinsic function

Facilities prohibited or not addressed by paper 98-165r1, but that are supported by using POINTER(C), include:

- Pointer(C) allows safe C object dereferencing, using semantics very similar to Fortran semantics.
- It should not be necessary to export a C_NULL object from ISO_C_TYPES.

2 Proposal

Use the POINTER(C) annotation in ways similar to the Fortran POINTER attribute, with modest restrictions, to provide support for several necessary facilities of C interoperability. The POINTER(C) annotation could be used in any situation where a Fortran POINTER annotation is allowed, except for the case of assumed or deferred type parameters (including dimensions), or an ALLOCATE or DEALLOCATE statement. Unlike Fortran pointers, arrays with the POINTER(C) annotation shall *not* be declared as deferred shape arrays, and they can be assumed-size arrays.

The restriction that Fortran pointer arrays shall be declared as deferred shape arrays seems unnecessary. There is no reason that pointer arrays could not have explicit shape. If the shape is explicit, it should be a constant specification expression so as to avoid undesirable side-effects. *J3 note*

3 The type C_PTR

Fortran does not need a distinct type for Fortran pointers. With the POINTER(C) annotation, Fortran would not need a distinct type for C pointers.

C pointer manipulation (except for pointer arithmetic, for which no provision should be made), and interoperability with Fortran pointers, can be achieved by using Fortran pointer assignment or argument association, with appropriate modifications to their definitions.

If a Fortran object is the *target* in a pointer assignment, and a C pointer object is the *pointer-object*, or if a Fortran object is an actual argument and a C pointer is the corresponding dummy argument, the type and type parameters (including dimensions) of the Fortran object shall be the same as the type and type parameters of the C pointer, except that the last dimension cannot be checked if the C pointer object is an assumed size array. If a C pointer is a *pointer-object* in a pointer assignment statement, the *bounds-spec-list* shall not be specified.

The case of a Fortran array pointer with assumed or deferred type parameters (including dimension) getting its association from a C pointer is allowed; other rules require that all type parameters or C pointers must be explicit, except in the case of assumed size arrays. If an assumed size array with the POINTER(C) annotation is the *target* in a pointer assignment statement, and a Fortran pointer is the *pointer-object*, the *bounds-spec-list* shall be specified.

There is at present no prohibition that the *target* in a pointer assignment statement shall not be an assumed-size array, or that if it is, the *bounds-spec-list* shall be specified. Is this as intended? *J3 note*

4 A VALUE annotation for dummy arguments

If POINTER(C) is used to annotate C functions' dummy arguments that are C pointers, and not used to annotate dummy arguments that are not C pointers, the VALUE annotation proposed by paper 98-165r1 is not needed. It isn't necessary, and in fact it is undesirable to require that POINTER(C) shall only be allowed to annotate arguments of procedures that are declared to be BIND(C) procedures.

Arguments that are pointer to pointer to ... can be represented by the usual Fortran subterfuge of using a derived type having a component that is a pointer, etc., provided that one abandons support for the possibility that C pointers to objects of different type are allowed to have different representation. Abandonment of this support is also advocated by paper 98-165r1.

The semantics of the presence or absence of POINTER(C) annotation of a dummy argument of a BIND(C) procedure, and POINTER or POINTER(C) annotation of a dummy argument of a Fortran procedure are not identical. For a Fortran procedure, absence of the POINTER annotation nonetheless allows the argument to be passed by reference, while in the case of a BIND(C) procedure, absence of the POINTER(C) annotation requires that the argument be passed by value.

In the case of a Fortran procedure, POINTER(C) annotation has the same meaning as POINTER annotation, except that type parameters are not passed as hidden parts of the argument.

Fortran at present does not do "automatic targeting" of non-pointer actual arguments associated to pointer dummy arguments. If the POINTER(C) approach is adopted, and it is desirable to allow non-pointer Fortran objects to be argument associated with POINTER(C) dummy arguments, it will be necessary to debate and vote whether there should be no exception for C pointers, an exception for C pointers, or a general change to allow "automatic targeting" of non-pointer actual arguments associated to pointer dummy arguments.

See also section 7 concerning procedure interface.

5 A LOC intrinsic function

A LOC intrinsic function is not required to make a Fortran object a target of a Fortran pointer. By using POINTER(C), it would not be needed in order to make a Fortran object a target of a C pointer. If a Fortran object has the TARGET attribute, it should be allowed to assign it as a target to a C pointer using pointer assignment. This would make the syntax of pointer association uniform, no matter whether the pointer is a C pointer or a Fortran pointer, rather than using => only for Fortran pointer association, and = for C pointer association, as would be the case if using TYPE(C_PTR) and LOC.

A LOC intrinsic function may be *useful* in the absence of "automatic targeting" of non-pointer actual arguments that one wishes to correspond to a pointer dummy argument, but it should not be considered exclusively a feature of C interoperability. "Automatic targeting" may be a better solution.

6 Dereferencing C pointer objects

It should be allowed for scalar or array element POINTER(C) objects to be dereferenced automatically in the same way as are Fortran POINTER objects. POINTER(C) objects cannot be assumed- or deferred-shape arrays, so it may be harmless to allow array sections in the case of explicit-shape POINTER(C) arrays. In the case when an array section of a POINTER(C) object is used as an actual argument, it shall be associated to an assumed-shape dummy argument (or there will be copy-in/copy-out?).

7 Interface specifications

Contrary to the recommendation in paper 98-165r1 that BIND(C) procedures do not have explicit interface, this paper *requires* that they have explicit interface. This allows a BIND(C) procedure to be a defined operator, to have arguments with the ASYNCHRONOUS attribute, to be PURE (this may be undesirable since one can't reason by induction that every procedure it calls is pure, ...), to be referenced by a generic name, to implement a defined operator, or to return a POINTER(C) result.

This paper proposes the following rules in order for a Fortran procedure to interoperate with a C function:

- The interface to the C function shall be specified by a Fortran interface body.
- The procedure declaration shall include the BIND(C) annotation.
- No dummy argument shall have an assumed type parameter (including dimension or length), except that assumed size arrays are permitted.
- No dummy argument shall have the Fortran POINTER attribute.
- No dummy argument shall be ALLOCATABLE.
- Dummy arguments that are declared without the POINTER(C) annotation shall be declared with the INTENT(IN) specification.
- If a dummy argument has the POINTER(C) annotation and an INTENT specification, the intent applies to the object, not the pointer association status. POINTER(C) and

INTENT(IN) correspond to the C **const** annotation. POINTER(C) and any other intent, or no intent specification, correspond to absence of the C **const** annotation.

- The procedure shall not have an asterisk (alternate return) argument.
- The procedure shall not be a defined assignment.
- The procedure shall not be elemental (is this required?).

A Fortran procedure declared in this way interoperates with a C function if:

- The result of a BIND(C) function is of a type that interoperates with Fortran, or the BIND(C) procedure is a subroutine, in which case it interoperates with a C function having result type **void**.
- The number of dummy arguments declared in the Fortran interface body is the same as the number of formal parameters of the C function.
- Dummy arguments with the POINTER(C) annotation, or that are of a derived type for which there is only one component, and that component has the POINTER(C) annotation, correspond to formal parameters that are C pointers, i.e. declared with a "*" prefix. (This definition allows arguments that are arrays of pointers.) Otherwise they correspond to formal parameters that are not C pointers, i.e. declared without a "*" prefix.
- The types of the dummy arguments are the same as the types of the corresponding formal arguments of the C procedure.

Symmetrically, a Fortran procedure defined according to these rules can be referenced from a C procedure (or a Fortran procedure!). There should be no problem allowing a Fortran procedure defined with the BIND(C) annotation to be pure, since pureness can be verified.

8 Miscellaneous rules and observations

An object with the POINTER(C) annotation shall not also have the Fortran POINTER attribute, nor be ALLOCATABLE, nor be used in an ALLOCATE or DEALLOCATE statement.

An object with the POINTER(C) annotation shall not have any assumed or deferred type parameters (including dimensions), except that an assumed-size POINTER(C) array is allowed.

It should not be necessary to export a C_NULL object from ISO_C_TYPES. The NULLIFY statement and the NULL() and ASSOCIATED() intrinsic functions should be usable with C pointers. ASSOCIATED() should work even in the case of inquiring whether an assumed- or deferred-shape Fortran pointer is associated to a C pointer, for the same reasons that pointer assignment works.

It is allowed, but invites portability problems, for objects having types other than those exported from ISO_C_TYPES, or derived types in which all components have types exported from ISO_C_TYPES or are derived types..., to have the POINTER(C) annotation.

If a dummy procedure has the BIND(C) annotation, the corresponding actual argument shall have it also. If a dummy procedure does not have the BIND(C) annotation, the corresponding actual argument shall not have it.