Section 15: Interoperability with C

Fortran provides a means of referencing procedures that are defined by means of the C programming language or procedures that can be described by C prototypes as defined in 6.5.5.3 of the C standard, even if they are not actually defined by means of C. Conversely, there is a means of specifying that a procedure defined by a Fortran subprogram can be referenced from a function defined by means of C. In addition, there is a means of declaring global variables that are linked with C variables that have external linkage as defined in 6.1.2.2 of the C standard.

The ISO_C_BINDING module provides access to named constants that represent kind type parameters of data representations compatible with C types. Fortran also provides facilities for defining derived types (4.5), enumerations (4.7), and type aliases (4.6) that correspond to C types.

15.1 The ISO_C_BINDING intrinsic module

The processor shall provide the intrinsic module ISO_C_BINDING. This module shall make accessible the following entities: C_INT, C_SHORT, C_LONG, C_LONG_LONG, C_SIZE_T, C_INT_LEAST8_T, C_INT_LEAST16_T, C_INT_LEAST32_T, C_INT_LEAST64_T, C_INT_FAST8_T, C_INT_FAST16_T, C_INT_FAST32_T, C_INT_FAST64_T, C_INTMAX_T, C_SIGNED_CHAR, C_FLOAT, C_DOUBLE, C_LONG_DOUBLE, C_COMPLEX, C_DOUBLE_COMPLEX, C_LONG_DOUBLE_COMPLEX, C_CHAR, C_BOOL, C_PTR, C_NULL_CHAR, C_LOC, C_ASSOCIATED, and C_NULL_PTR. The ISO_C_BINDING module shall not make accessible any other entity.

The entities C_INT, C_SHORT, C_LONG, C_LONG_LONG, C_SIGNED_CHAR, C_FLOAT, C_DOUBLE, C_LONG_DOUBLE, C_SIZE_T, C_INT_LEAST8_T, C_INT_LEAST16_T, C_INT_LEAST32_T, C_INT_LEAST64_T, C_INT_FAST8_T, C_INT_FAST16_T, C_INT_FAST32_T, C_INT_FAST64_T, C_INTMAX_T, C_COMPLEX, C_DOUBLE_COMPLEX, C_LONG_DOUBLE_COMPLEX, and C_CHAR shall be named constants of type default integer.

The value of C_INT shall be a valid value for an integer kind parameter on the processor. The values of C_SHORT, C_LONG, C_LONG_LONG, C_SIZE_T, C_INT_LEAST8_T, C_INT_LEAST16_T, C_INT_LEAST32_T, C_INT_LEAST64_T, C_INT_FAST8_T, C_INT_FAST16_T, C_INT_FAST32_T, C_INT_FAST64_T, C_INTMAX_T, and C_SIGNED_CHAR shall each be a valid value for an integer kind type parameter on the processor or shall be -1.

The values of C_FLOAT, C_DOUBLE, and C_LONG_DOUBLE shall each be a valid value for a real kind type parameter on the processor or shall be -1 if the C processor's type does not have a precision equal to the precision of any of the Fortran processor's real kinds, -2 if the C processor's type does not have a range equal to the range of any of the Fortran processor's real kinds, -3 if the C processor's type has neither the precision nor range of any of the Fortran processor's real kinds, and equal to -4 if there is no interoperating Fortran processor kind for other reasons. The values of C_COMPLEX, C_DOUBLE_COMPLEX, and C_LONG_DOUBLE_COMPLEX shall be the same as those of C_FLOAT, C_DOUBLE, and C_LONG_DOUBLE, respectively.

NOTE 15.1

If the C processor supports more than one variety of float, double or long double, the Fortran processor may find it helpful to select from among more than one ISO_C module by a processor dependent means.

The value of C_CHAR shall be a valid value for a character kind type parameter on the processor or shall be -1. The value of C_CHAR is known as the **C character kind**.

The value of C_BOOL shall be a valid value for a logical kind parameter on the processor or shall be -1.

The entity C_NULL_CHAR shall be a named constant of type character with a length parameter of one and a kind parameter equal to the value of C_CHAR, unless the value of C_CHAR is -1, in which case the kind parameter of C_NULL_CHAR is the default character kind. If C_CHAR has a non-negative value, the value of C_NULL_CHAR shall be the same as the null value of the C character type; otherwise, the value of C_NULL_CHAR shall be the first character in the collating sequence for characters of default kind.

The entity C_PTR shall be a derived type or a type alias name. Its use is described in 15.2.2.

The C_LOC function is described in 15.2.3.

The entity C_NULL_PTR shall be a named constant of type C_PTR. The value of C_NULL_PTR shall be the same as the value NULL in C.

15.2 Interoperation between Fortran and C entities

If a variable, structure component, subprogram, or procedure is declared in a way that allows an equivalent declaration of an entity in a C program, the entity in the Fortran program is said to be **interoperable** with such a C entity. The following subclauses define the conditions under which a Fortran entity is interoperable with a C entity.

NOTE 15.2

A Fortran entity can be interoperable with more than one C entity.

15.2.1 Fortran scalar intrinsic entities and C entities

Table 15.1 shows the correspondence between Fortran and C types. The second column of the table refers to the names made accessible by the ISO_C_BINDING intrinsic module. A scalar Fortran entity that has neither the POINTER nor the ALLOCATABLE attribute, and is of the indicated type and kind type parameter, is interoperable with a scalar C entity whose unqualified type is compatible with the indicated C type in the same row of the table. See 6.2.5 and 6.2.7 of the C standard for definitions of unqualified types and compatible types. If the value of any of the kind type parameters in this table is -1, there is no Fortran entity that is interoperable with a C entity of the C types specified in that row of the table. Furthermore, a Fortran entity of type character is not interoperable with any C entity unless the length of the Fortran entity is omitted or is specified by an initialization expression whose value is one.

NOTE 15.3

The C programming language defines null-terminated strings, which are actually arrays of the C type char that have a C null character in them to indicate the last valid element. A Fortran array of type character with a kind type parameter equal to C_CHAR is interoperable with a C string.

Fortran's rules of sequence association (12.4.1.5) permit a character scalar actual argument to be associated with a dummy argument array. This makes it possible to argument associate a Fortran character string with a C string.

Note 15.19 has an example of interoperation between Fortran and C strings.

Table 15.1 Correspondence between Fortran and C types

Fortran type	Kind type parameter	C type
INTEGER	C_INT	int signed int
	C_SHORT	short int signed short int
	C_LONG	long int signed long int
	C_LONG_LONG	long long int signed long long int
	C_SIGNED_CHAR	signed char unsigned char
	C_SIZE_T	size_t
	C_INT_LEAST8_T	int_least8_t
	C_INT_LEAST16_T	int_least16_t
	C_INT_LEAST32_T	int_least32_t
	C_INT_LEAST64_T	int_least64_t
	C_INT_FAST8_T	int_fast8_t
	C_INT_FAST16_T	int_fast16_t
	C_INT_FAST32_T	int_fast32_t
	C_INT_FAST64_T	int_fast64_t
	C_INTMAX_T	c_intmax_t
REAL	C_FLOAT	float float _Imaginary
	C_DOUBLE	double double _Imaginary
	C_LONG_DOUBLE	long double long double _Imaginary
COMPLEX	C_COMPLEX	_Complex
	C_DOUBLE_COMPLEX	double _Complex
	C_LONG_DOUBLE_COMPLEX	long double _Complex
LOGICAL	C_BOOL	_Bool
CHARACTER	C_CHAR	char
The above mentioned C types are defined in the C standard, clause 6.2.5.		

NOTE 15.4

For example, a scalar object of type integer with a kind type parameter of C_SHORT is interoperable with a scalar object of the C type short or of any C type derived (via typedef) from short.

NOTE 15.5

The C standard specifies that the representations for positive signed integers are the same as the corresponding values of unsigned integers. Because Fortran does not provide direct support for unsigned kinds of integers, the ISO_C_BINDING module does not make accessible named constants C_UNSIGNED_INT, C_UNSIGNED_SHORT, C_UNSIGNED_LONG, C_UNSIGNED_LONG_LONG, or C_UNSIGNED_CHAR. Instead a user can use the signed kinds of integers to interoperate with the unsigned types and all their qualified versions as well. This has the potentially surprising side effect that a scalar of the C type unsigned char is interoperable with scalars of type integer with a kind type parameter of C_SIGNED_CHAR.

Values that can be represented in an unsigned type but not in the corresponding signed type are represented by different values.

15.2.2 Interoperation with C pointer types

A scalar Fortran entity of type C_PTR, that has neither the POINTER nor the ALLOCATABLE attribute, is interoperable with a scalar C entity of any C pointer type.

NOTE 15.6

This implies that a C processor is required to have the same representation method for all C pointer types if the C processor is to be the target of interoperability of a Fortran processor. The C standard does not impose this requirement.

NOTE 15.7

No facility for dereferencing of C pointers within Fortran is provided.

15.2.3 Interoperability inquiry functions

Many C interfaces are defined in terms of ‘‘addresses’’. The C_LOC function is provided so that Fortran applications can determine the appropriate value to use with C facilities. The C_ASSOCIATED function is provided so that Fortran programs can compare ‘‘C addresses’’.

C_LOC (X)

Description. Returns the ‘‘C address’’ of the argument.

Class. Inquiry function.

Argument. X shall be either a variable that has the TARGET attribute and is interoperable with some C object or a procedure that has the BIND attribute. It shall not be an array pointer, an assumed shape array, or an array section.

Result Characteristics. Scalar of type C_PTR.

Result Value. 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.

C_ASSOCIATED (C_PTR_1 [, C_PTR_2]))

Description. Indicates the association status of C_PTR_1 or indicates whether C_PTR_1 and C_PTR_2 are associated with the same object.

Class. Inquiry function.

Arguments.

C_PTR_1 shall be a scalar of type C_PTR.

C_PTR_2 (optional) shall be a scalar of type C_PTR.

Result Characteristics. Default logical scalar.

Result Value.

Case (i): If C_PTR_2 is absent, the result is false if C_PTR_1 is a C null pointer and true otherwise.

Case (ii): If C_PTR_2 is present, the result is false if C_PTR_1 is a C null pointer. Otherwise, the result is true if C_PTR_1 compares equal to C_PTR_2 in the sense of 6.3.2.3 of the C standard, and false otherwise.

NOTE 15.8

The following example illustrates the use of C_LOC and C_ASSOCIATED.

```
USE ISO_C_BINDING
INTERFACE
  SUBROUTINE FOO(GAMMA), BIND(C)
    USE ISO_C_BINDING
    TYPE(C_PTR), VALUE :: GAMMA
  END SUBROUTINE FOO
END INTERFACE
REAL(C_FLOAT), TARGET, DIMENSION(100) :: ALPHA
TYPE(C_PTR) :: BETA
...
IF (.NOT. C_ASSOCIATED(BETA)) THEN
  BETA = C_LOC(ALPHA)
ENDIF
CALL FOO(BETA)
```

15.2.4 Interoperation with C struct types

A scalar Fortran entity of derived type, that has neither the POINTER nor the ALLOCATABLE attribute, is interoperable with a scalar C entity of a struct type if the derived type definition of the Fortran type specifies BIND(C) (4.5.1), the Fortran derived type and the C struct type have the same number of components, and the components of the Fortran derived type are interoperable with the corresponding components of the struct type. A component of a Fortran derived type and a component of a C struct type correspond if they are declared in the same relative position in their respective type definitions.

NOTE 15.9

The names of the corresponding components of the derived type and the C struct type need not be the same.

There is no Fortran entity that is interoperable with a C entity of a struct type that contains a bit field or that contains a flexible array member. There is no Fortran entity that is interoperable with a C entity of a union type.

NOTE 15.10

For example, a scalar C object of type myctype, declared below, is interoperable with a scalar Fortran object of type myftype, declared below.

```
typedef struct {
  int m, n;
  float r;
} myctype
```

```
USE ISO_C_BINDING
TYPE, BIND(C) :: MYFTYPE
  INTEGER(C_INT) :: I, J
  REAL(C_FLOAT) :: S
END TYPE MYFTYPE
```

The names of the types and the names of the components are not significant for the purposes of determining whether entities of a derived type are interoperable with entities of a C struct type.

NOTE 15.11

The C standard requires the names and component names to be the same in order for the types to be compatible (C standard, clause 6.2.7). This is similar to Fortran's rule describing when sequence derived types are considered to be the same type. This rule was not extended to determine whether a Fortran entity of derived type is interoperable with a C entity of a struct type because the case of identifiers is significant in C but not in Fortran.

15.2.5 Interoperation with C array types

An explicit-shape or assumed-size array of rank *r*, with a shape of $[e_1 \ldots e_r]$ is interoperable with a C array if either

(1) the array is assumed size, and the C array does not specify a size or specifies a size of *, or

(2) the array is an explicit shape array, and the extent of the last dimension (e_r) is the same as the size of the C array,

and either

(1) *r* is equal to one, and an element of the array is interoperable with an element of the C array, or

(2) *r* is greater than one, and an explicit-shape array with shape of $[e_1 \ldots e_{r-1}]$, with the same type and type parameters as the original array, is interoperable with a C array of a type equal to the element type of the original C array.

NOTE 15.12

An element of a multi-dimensional C array is an array type, so a Fortran array of rank one is not interoperable with a multidimensional C array.

NOTE 15.13

For example, a Fortran array declared as

```
INTEGER :: A(18, 3:7, *)
```

is interoperable with a C array declared as

```
int b[][5][18]
```

15.2.6 Interoperation with C functions

A Fortran procedure interface is interoperable with a C function prototype if

(1) the interface has the BIND attribute;

(2) either

  (a) the interface describes a function whose result variable is interoperable with the result of the prototype or

  (b) the interface describes a subroutine, and the prototype has a result type compatible with the C type void;

(3) the number of dummy arguments of the interface is equal to the number of formal parameters of the prototype;

(4) all the dummy arguments are dummy data objects;

(5) any dummy argument with the VALUE attribute is interoperable with the corresponding formal parameter of the prototype;

(6) any dummy argument without the VALUE attribute corresponds to a formal parameter of the prototype that is of a pointer type, and the type of the dummy argument is interoperable with the referenced type (C standard, 6.2.5) of the formal parameter; and

(7) the prototype does not have variable arguments as denoted by the ellipsis (...).

NOTE 15.14

The **referenced type** of a C pointer type is the C type of the object that the C pointer type points to. For example, the referenced type of the pointer type **`int *`** is **`int`**.

NOTE 15.15

The C language allows specification of a C function that can take a variable number of arguments (C standard, 7.15). This standard does not provide a mechanism for Fortran procedures to interoperate with such C functions.

A formal parameter of a C function prototype corresponds to a dummy argument of a Fortran procedure interface if they are in the same relative positions in the C parameter list and the dummy argument list, respectively.

NOTE 15.16

The VALUE attribute may not be applied to arrays (5.1.2.14). It would have been unhelpful to have permitted it, because arrays in C are always passed by reference.

NOTE 15.17

For example, a Fortran procedure interface described by

```
FUNCTION FUNC(I, J, K, L, M), BIND(C)
  USE ISO_C_BINDING
  INTEGER(C_SHORT) :: FUNC
  INTEGER(C_INT), VALUE :: I
  REAL(C_DOUBLE) :: J
  INTEGER(C_INT) :: K, L(10)
  TYPE(C_PTR), VALUE :: M
END FUNCTION FUNC
```

is interoperable with the C function prototype

```
short func(int i; double *j; int *k; int l[10]; void *m)
```

A C pointer may correspond to a Fortran dummy argument of type C_PTR or to a Fortran scalar that does not have the VALUE attribute. In the above example, the C pointers **`j`** and **`k`** correspond to the Fortran scalars **`J`** and **`K`**, respectively, and the C pointer **`m`** corresponds to the Fortran dummy argument **`M`** of type C_PTR.

A *language-binding-spec* shall not be specified for a procedure that has an asterisk dummy argument. A *language-binding-spec* shall not be specified for a procedure that requires an explicit interface for any reason other than the presence of the *language-binding-spec*.

NOTE 15.18

The requirement that the Fortran procedure not require an explicit interface prohibits its dummy arguments from having the POINTER attribute, having the ALLOCATABLE attribute, or being assumed shape arrays. It also prohibits the Fortran procedure from being elemental or having array results.

NOTE 15.19

The interopability of Fortran procedure interfaces with C function prototypes is only one part of invocation of a C function from Fortran. There are four pieces to consider in such an invocation: the procedure reference, the Fortran procedure interface, the C function prototype, and the C function. Conversely, the invocation of a Fortran procedure from C involves the function reference, the C function prototype, the Fortran procedure interface, and the Fortran procedure. In order to determine whether a reference is allowed, it is necessary to consider all four pieces.

For example, consider a C function that can be described by the C function prototype

```
void copy(char in[], char out[]);
```

Such a function may be invoked from Fortran as follows:

```
  USE ISO_C_BINDING
  INTERFACE
    SUBROUTINE COPY(IN, OUT), BIND(C)
      USE ISO_C_BINDING
      CHARACTER(KIND=C_CHAR), DIMENSION(*) :: IN, OUT
    END SUBROUTINE COPY
  END INTERFACE

  CHARACTER(LEN=10, KIND=C_CHAR) :: &
 &     DIGIT_STRING = C_CHAR_'123456789' // C_NULL_CHAR
  CHARACTER(KIND=C_CHAR) :: DIGIT_ARR(10)

  CALL COPY(DIGIT_STRING, DIGIT_ARR)
  PRINT '(1X, A1)', DIGIT_ARR
  END
```

The procedure reference has a character string actual arguments. These correspond to character array dummy arguments in the procedure interface body as allowed by Fortran's rules of sequence association (12.4.1.5). Those array dummy arguments in the procedure interface are interoperable with the formal parameters of the C function prototype. The C function is not shown here, but is assumed to be compatible with the C function prototype.

15.2.7 Interoperation with C global variables

A C variable with external linkage may interoperate with a variable declared in the scope of a module or with a common block. A variable that is specified to have the BIND attribute is interoperable with a C variable that has external linkage. At most one variable that is associated with a particular C variable with external linkage is permitted to be declared within a program. A variable shall not be initially defined by more than one processor.

There does not have to be an associated C entity for a Fortran entity with the BIND attribute.

If a common block is specified in a BIND statement, it shall be specified in a BIND statement with the same binding label in each scoping unit in which it is declared. A C variable with external linkage is interoperable with a common block that has been specified in a BIND statement,

(1) if the C variable is of a struct type and the variables that are members of the common block are interoperable with corresponding components of the struct type, or

(2) if the common block contains a single variable, and the variable is interoperable with the C variable.

NOTE 15.20

The following are examples of the usage of the BIND attribute for variables and for a common block. The Fortran variables, C_EXTERN and C2, interoperate with the C variables, c_extern and myVariable, respectively. The Fortran common blocks, COM and SINGLE, interoperate with the C variables, com and single, respectively.

```
MODULE LINK_TO_C_VARS
  USE ISO_C_BINDING
  INTEGER(C_INT), BIND(C) :: C_EXTERN
  INTEGER(C_LONG) :: C2
  BIND(C, NAME='myVariable') :: C2

  COMMON /COM/ R, S
  REAL(C_FLOAT) :: R, S, T
  BIND(C) :: /COM/, /SINGLE/
  COMMON /SINGLE/ T
END MODULE LINK_TO_C_VARS
```

```
int c_extern;
long myVariable;
struct {float r, s;} com;
float single
```

15.2.7.1 Binding labels for common blocks and variables

The **binding label** of a variable or common block is a value of type default character that specifies the name by which the variable or common block is known to the companion processor.

If a variable or common block has the BIND attribute specified with a NAME= specifier, the binding label is the value of the expression specified for the NAME= specifier. The case of letters in the binding label is significant, but leading and trailing blanks are ignored. If a variable or common block has the BIND attribute specified without a NAME= specifier, the binding label is the same as the name of the entity using lower case letters.

The binding label of a C variable with extern linkage is the same as the name of the C variable. A Fortran variable or common block with the BIND attribute that has the same binding label as a C variable with extern linkage is associated with that variable.

A BINDNAME= specifier for a variable or common block is processor-dependent specification of a name by which that entity may be known to a companion processor. The valid values for and interpretation of the character expression in a BINDNAME= specifier are processor dependent.