Interoperability with C

Jules Kouatchou

NASA GSFC Code 606 (ASTG)
Greenbelt, Maryland 20771

`Jules.Kouatchou@nasa.gov`

October 25, 2018

Agenda

Interoperability with C

- ISO_C_BINDING
- Intrinsic Types
- Interoperable procedures
- Interoperable data

Introduction

- Fotran provides standards for accessing libraries and procedures developed in C
- Fotran also provides standards for accessing Fortran libraries and procedures developed from C
- Interoperability enforced by requirements on Fortran syntax
- Use of these features requires some familiarity with both C and Fortran

ISO_C_BINDING

Intrinsic module that provides:

- Named constants for declaring Fortran data which interoperates with C data
- Small number of procedures for managing pointers and addresses

To avoid naming conflicts it is recommended that the ONLY options is used in the USE statement.

Intrinsic Data Types

For each C data type provided by the vendor there is an equivalent named constant in ISO_C_BINDING

- Value of the named constant specifies the **KIND** for the corresponding Fortran data type
- Support for: INTEGER, REAL, COMPLEX, LOGICAL, and CHARACTER types

Intrinsic Types

Fortran Type	**Named constant from C type ISO_C_BINDING**	**C Type**
INTEGER	C_INT	int
INTEGER	C_LONG	short int
INTEGER	C_INT32	int32_t
INTEGER	C_INT64	int64_t
REAL	C_FLOAT	float
REAL	C_DOUBLE	double
COMPLEX	C_FLOAT_COMPLEX	float_Complex
LOGICAL	C_BOOL	_Bool
CHARACTER	C_CHAR & char	

Example of Declarations

Example of Fortran declaration interoperable with C double

```
    use, intrinsic :: ISO_C_Binding, only: C_int, C_char, C_
    IMPLICIT NONE

    integer, parameter :: maxLengthChar= 100

    character(kind=C_CHAR), DIMENSION(*) :: stationName
    real(KIND=C_FLOAT) :: latitude
    real(KIND=C_FLOAT) :: longitude
```

Intrinsic Procedures

- **C_LOC** (var) → Returns C address (type C_PTR) of var
- **C_FUNLOC** (proc) → Returns C address (type C_FUNPTR) of procedure
- **C_ASSOCIATED** (cPtr1 [, cPtr2]) → Returns false if cPtr1 is a null C pointer or if cPtr2 is present with a different value
- **C_F_POINTER** (cPtr1, fPtr [, shape]) → Associates Fortran pointer, fPtr1, with address cPtr1 (type C_PTR). Shape is required when fPtr1 is an array pointer.
- **C_F_PROCPOINTER** (cPtr1, fPtr) → Associates Fortran procedure pointer, fPtr1 with the address of interoperable C procedure cPtr1 (type C_FUNPTR)

Interoperable Procedures

- A Fortran procedure is interoperable if
  - It has an explicit interface
  - It has been declared with the BIND attribute
  - The number of dummy arguments is equal to the number of formal parameters in the prototype and are in the same relative positions as the C parameter list
  - All dummy arguments are interoperable
- Return values
  - An interoperable Fortran function must have a result that is scalar and interoperable
  - For a subroutine, the C prototype must have a void result
- Caveats
  - Interoperable functions cannot return array values
  - Fortran procedures cannot interoperate with C functions that take a variable number of arguments (the C language specification allows this)

Example of Interoperable Fortran Procedure Interface

```
INTERFACE
   FUNCTION func (i, j, k, l, m), BIND (C, name=''C_Func'')
       USE, INTRINSIC :: 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
END INTERFACE

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

Binding Labels for Procedures

A **binding label** is a value that specifies the name by which a procedure with the BIND attribute is known

- Has global scope
- By default, it is the lower-case version of the Fortran name

Interoperable Data

Fortran data is interoperable if an equivalent data declaration can be made in C and the data is said to be interoperable

- Scalar and array variables are interoperable
- Dynamic arrays can be passed between the two languages
- The BIND attribute is required for a Fortran derived type to be interoperable
- C variables with external linkage can interoperate with Fortran common blocks or module variables that have the BIND attribute

Interoperability of Variables

- Fortran scalars are interoperable if
  - the type and type parameters are interoperable with a scalar C variable
  - they are not declared as pointers nor have the allocatable attribute
- Fortran arrays are interoperable if
  - The type and type parameters are interoperable
  - they are of explicit shape or assumed size
- Fortran arrays interoperate with C arrays of the same type, type parameters and shape, but with reversed subscripts

Example of Interoperability of Variables

```
real :: A(3,4)
real :: A(3,*)
real :: A(:,:) ! not allowed

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

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

Derived Types

- Interoperable Fortran derived types must
  - Specify the BIND (C) attribute
  - Have the same number of components as the C struct type
  - Have components with type and type parameters that are interoperable with the types of the corresponding components of the C struct type
- Components of the Fortran derived type
  - Correspond to the C struct type components declared in the same relative position
  - Corresponding components do not need to have the same name

Example of Derived Type

```
TYPE, BIND (C) :: fType
    INTEGER (C_INT) :: i, j
    REAL (C_FLOAT) :: s
END TYPE fType

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

Global Data

- A C variable with external linkage can interoperate with a Fortran common block or variable that has the BIND attribute
- C variable with external linkage interoperates with a common block specified in a BIND statement in one of two ways:
  - The C variable is a struct type and the elements are interoperable with the members of the common block
  - Or the common block contains only one interoperable variable
- Only one variable can be associated with a C variable with external linkage

Example of Global Data

```
use ISO_C_BINDING
COMMON /COM/ r, s
REAL(C_FLOAT) :: r, s
BIND(C) :: /COM/

struct {
   float r, s;
} com; /* external */

void setter() {
   com.r = 3;
   com.s = 4;
}
```

Array Variables

- A Fortran array of rank one is not interoperable with a multidimensional C array
- Polymorphic, allocatable, and pointer arrays are never interoperable
- A Fortran array of type character with a kind type of C_CHAR is interoperable with a C string (C null character as last element of the array)
    - ISO_C_BINDING provides the constant C_NULL_CHAR

Dynamic Arrays

- C pointers are the mechanism for passing dynamic arrays between the two languages
  - an allocated allocatable Fortran array can be passed to C
  - an array allocated in C can be passed to a Fortran pointer
  - a Fortran pointer target or assumed-shape array (no bounds specified) cannot be passed to C
- ISO_C_BINDING provides
  - C_PTR is the derived type for interoperating with any C object pointer type
  - C_NULL_PTR is the named constant of type C_PTR with the value NULL in C

Example of Interoperable Dynamic Arrays - C

```c
typedef struct {
   int lenc, lenf;
   float *c, *f;
} pass;

int main () {

pass *arrays=(pass*)malloc(sizeof(pass));
(*arrays).lenc = 2;
arrays->c =malloc((*arrays).lenc*sizeof(float));
a[0] = 10.0;
a[1] = 20.0;

for(i=0;i<(*arrays).lenc;i++) {
   *(arrays->c+i)=a[i];
}
/* Calling Fortran routine "simulation" */
simulation(arrays);
```

Example of Interoperable Dynamic Arrays - Fortran

```fortran
SUBROUTINE simulation(arrays) bind(c)


TYPE, BIND(c) :: pass
   integer (C_INT) :: lenc, lenf
   TYPE (C_PTR) :: c, f
END TYPE pass

TYPE (pass), INTENT(INOUT) :: arrays
REAL (C_FLOAT), POINTER : cArray (:)
CALL C_F_POINTER(arrays%c,cArray, (/arrays%lenc/))
print*, cArray
```

Exercise

- Write a C subroutine that computes the averrage and standard deviation of an array with n entries.
- Call the subroutine within Fortran.