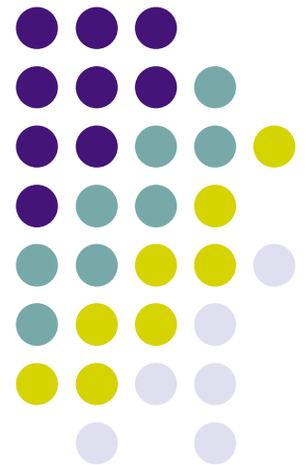


Interoperability with C in Fortran 2003

Megan Damon - SIVO/NGIT
SIVO Fortran Series
February 12th 2008





Logistics

- Materials for this series can be found at <http://modelingguru.nasa.gov/clearspace/docs/DOC-1375>
 - Contains slides and source code examples.
 - Latest materials may only be ready at-the-last-minute.
- Please be courteous:
 - Remote attendees should use “*6” to toggle the mute. This will minimize background noise for other attendees.
- Webex - under investigation



Outline

- Introduction
- ISO_C_BINDING intrinsic module
- Intrinsic types
- Interoperable procedures
- Interoperable data
- Best Practices & Limitations
- Resources



Introduction

- Fortran 2003 provides a *standard (and ultimately portable)* means for interoperating with C
 - Provides access to libraries and procedures developed in C
 - Conversely, provides access to Fortran libraries and procedures from C
- Interoperability enforced by requirements on Fortran syntax; compiler knows at compile time
 - Fortran compiler may support multiple C compilers
 - Selection of C compiler is vendor dependent
- Use of these features requires some familiarity with both C and Fortran



Supported Features

Features	ifort	NAG	xlf	g95	gfortran
Interoperability with C	10.0	5.1	10.0	0.90	20070810

Feel free to contribute if you have access to other compilers not mentioned!



ISO_C_BINDING

- A vendor provided *intrinsic* module
 - Provides named constants for declaring Fortran data which interoperates with C data
 - Small number of procedures for managing pointers and addresses
- Vendors may provide a means to select different ISO_C_BINDING modules among varying C compilers

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

Intrinsic Data Types

Interoperable entities



- 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
- Example of Fortran declaration interoperable with C double:

```
real(KIND=C_DOUBLE) :: temperature
```



Caveats

- Vendor is not required to support all cases
 - Should not be an issue on IEEE hardware (?)
 - Integer kinds:
 - -1 indicates no corresponding Fortran kind
 - -2 indicates no corresponding C data type
 - Floating point kinds:
 - -1 indicates no exact correspondence in *precision*
 - -2 indicates no exact correspondence in *range*
 - -3 neither
 - -4 any other reason
 - Same idea for invalid values of boolean and characters kinds
- Fortran does not provide support for unsigned kinds of integers



Intrinsic Types

Fortran type	Named constant from 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

Intrinsic Types -

C characters with special semantics



Name	C definition	Value (C_CHAR= -1)	Value (C_CHAR≠ -1)
C_NULL_CHAR	null character	CHAR(0)	'\0'
C_ALERT	alert	ACHAR(7)	'\a'
C_BACKSPACE	backspace	ACHAR(8)	'\b'
C_FORM_FEED	form feed	ACHAR(12)	'\f'
C_NEW_LINE	new line	ACHAR(10)	'\n'
C_CARRIAGE_RETURN	carriage return	ACHAR(13)	'\r'
C_HORIZONTAL_TAB	horizontal tab	ACHAR(9)	'\t'
C_VERTICAL_TAB	vertical tab	ACHAR(11)	'\v'



Intrinsic Procedures

- **C_LOC** (var)
 - Returns C address (type C_PTR) of var
 - Some restrictions may apply
- **C_FUNLOC** (proc)
 - Returns C address (type C_FUNPTR) of procedure
- **C_ASSOCIATED** (cPtr1 [, cPtr2])
 - Inquiry function for object and function pointers
 - Returns false if cPtr1 is a null C pointer or if cPtr2 is present with a different value
- **C_F_POINTER** (cPtr1, fPtr1 [, shape])
 - Associates Fortran pointer, fPtr1, with address cPtr1 (type C_PTR)
 - Shape is required when fPtr1 is an array pointer
- **C_F_PROCPOINTER** (cPtr1, fPtr1)
 - 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
 - and all the 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)
```

```
    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
```

```
...
```

```
short 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
- **Examples of binding labels for Fortran procedures**
 - Function with assumed binding label of **func**
FUNCTION FUNC (i, j, k, l, m), BIND (C)
 - Function with explicit binding label of **C_Func**
FUNCTION FUNC (i, j, k, l, m), BIND (C, 'C_Func')



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 and
 - they are not declared as pointers nor have the allocatable attribute
- Fortran arrays
 - are interoperable if
 - the type and type parameters are interoperable
 - and are of explicit shape or assumed size
 - e.g. `real :: A(3,4)`
 - e.g. `real :: A(3,*)`
 - Not allowed `real :: A(:,:)`
 - interoperate with C arrays of the same type, type parameters and shape, but with reversed subscripts
 - Example of an interoperable Fortran and C array
`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
- Caveats
 - C struct types with bit fields or flexible array members are not interoperable with Fortran types
 - Fortran types are not interoperable with a C union type



Derived Type Source Example

```
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 module 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



Global Data Example

```
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

Examples of Interoperable Dynamic Arrays



```
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];  
}
```

1. C initializes the arrays to be passed to Fortran

```
/* Calling Fortran routine "simulation" */  
simulation(arrays);
```

```
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
```

2. Fortran associates cArray with array initialized in C program and prints the values

operability with C



Examples of Interoperable Dynamic Arrays

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

int main () {
  ...
  pass *arrays=(pass *)malloc(sizeof(pass));

  /* Calling Fortran routine "simulation" */
  simulation(arrays);

  for(i=0;i<(*arrays).lenf;i++) {
    printf("%f\n",*(arrays->f+i));
  }
  ...
}
```

1. C program allocates arrays of type pass



3. C prints the modified values of arrays->f

```
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),ALLOCATABLE,TARGET,SAVE :: eta(:)
```

```
arrays%lenf = 3
ALLOCATE (eta(arrays%lenf))
do i = 1,arrays%lenf
  eta(i) = 10.*i
enddo
arrays%f = C_LOC(eta)
```

2. Fortran allocates an array and makes it available in C





Best Practices & Limitations

- Best practices:
 - Use explicit “ONLY” clause for use of ISO_C_BINDING
 - Use “name=” specifier for external names
 - Use all caps for named constants
 - Use ISO_C_BINDING for portability in-and-of-itself?
- Limitations
 - Vendor need not support all available C compilers
 - This is C not C++
 - Limited support for advanced Fortran features
 - No optional arguments
 - No array return values
 - No assumed-shape arrays (arr(:,:)) nor pointer targets
 - Etc.



Resources

- **This talk:**
<https://modelingguru.nasa.gov/clearspace/docs/DOC-1390>
- **Questions to Modeling Guru:** <https://modelingguru.nasa.gov>
- **SIVO code examples on Modeling Guru**
- **Fortran 2003 standard:**
<http://www.open-std.org/jtc1/sc22/open/n3661.pdf>
- ***John Reid summary:***
 - <ftp://ftp.nag.co.uk/sc22wg5/N1551-N1600/N1579.pdf>
 - <ftp://ftp.nag.co.uk/sc22wg5/N1551-N1600/N1579.ps.gz>
- ***Real world examples***
 - ***Fortran 2003 Interface to OpenGL:***
<http://www-stone.ch.cam.ac.uk/pub/f03gl/>
 - ***Fotran 2003 version of NETCDF:***
<ftp://ftp.unidata.ucar.edu/pub/netcdf/contrib/netcdf-3.6.1-f03-2.tgz>
 - **FGSL: A Fortran interface to the GNU Scientific Library**
<http://www.lrz-muenchen.de/services/software/mathematik/gsl/fortran/index.html>



Next Fortran 2003 Session

- Extensions to Allocatables and Pointers
- Tom Clune will present
- Tuesday, February 26 2008
- B28-E210