Fortran 2003

Jim Xia
IBM Toronto Lab
jimxia@ca.ibm.com

Agenda

- Background
- What's new in Fortran 2003
- IBM XL Fortran
- Q&A

Background

- 50 year anniversary in 2007
- Fortran 2003 published in Nov, 2004
  - Fourth edition of Fortran standard
  - number of pages increased from 356 (Fortran 95) to 569
  - many new features added
  - include many interpretations to Fortran 95
  - include two Technical Reports
    - allocatable dummy arguments and allocatable components
    - support for IEEE Floating Point Standard (IEEE 1989) (IEEE intrinsic modules)
- Next revision (Fortran 2008) is near FCD phase
  - Publication date is set in 2010
  - Major features added: coarrays, submodules

What's new in Fortran 2003

- Object-oriented programming support
- I/O Enhancements
- Scoping and data manipulation enhancements
- C interoperability
- Procedure enhancements
- Parameterized derived type

Object-oriented programming

- type extension (inheritance)

```
type fluid
  real :: viscosity
  real, allocatable :: velocity(:,:,:)
end type

type, extends(fluid) :: magnetofluid
  real, allocatable :: magnetic_field(:,:,:)
end type
```

  - **type magnetofluid inherited ALL of properties of fluid: viscosity and velocity**
  - **Only support single-rooted inheritance hierarchy**

Object-oriented programming (cont'd)

- type-bound procedures

```
type point
  real x, y
  contains
    procedure :: length => lenBetween2Points
end type
...!definition of lenBetween2Points
real function lenBetween2Points(this, p)
    class(point), intent(in) :: this, p
    ... ! compute the length
end function
...! in main program
type(point) :: pa, pb
...
distance = pa%length(pb)
```

Object-oriented programming (cont'd)

- Declaration of polymorphic data
  - CLASS keyword

```
class(fluid) A !could be fluid or magnetofluid
```

- SELECT TYPE construct
  - Allows execution flow controled by dynamic type of the selector

```
select type(A)
  type is (fluid)
    ... !block for fluid
  class is (magnetofluid)
    ... !block for magnetofluid
  class default
    ...
end select
```

Object-oriented programming (cont'd)

- abstract types and deferred binding
  - No concrete objects can be declared to have an abstract type
  - Deferred binding: defer implementation for type-bound procedures to extending types, only providing well-defined interfaces

```
type, abstract :: shape
  contains
    procedure(shapeArea), deferred :: area
end type


abstract interface
    real function shapeArea (this)
        import shape
        class(shape), intent(in) :: this
    end function
end interface
```

I/O enhancements

- user-defined derived-type I/O
  - allows user to provide procedures (subroutines) to be used for reading or writing a derived type
  - user defined procedures are invoked by Fortran's READ, WRITE or PRINT statement as if an intrinsic IO
  - useful for derived types with POINTER/ALLOCATABLE components
- stream access I/O
- asynchronous I/O
- Infinities and NaNs in formatted READ and WRITE
  - **`[+/-]INF`** or **`[+/-]INFINITY`** for infinities
  - **`NaN[(...)]`**, for NaNs. e.g. **`NaN(Q)`**

I/O enhancements (cont'd)

- specifier enhancements: allow more control at data transfer
  - BLANK= specifier
  - PAD= specifier
  - DELIM= specifier
- New specifiers for more control
  - SIGN= specifier
    - controls whether the plus sign is displayed for positive numbers in formatted I/O
  - DECIMAL= specifier
    - specifies the decimal separator for floating-point number
  - ROUND= specifier
    - controls the rounding mode for formatted I/O

Data manipulation and scoping enhancements

- allocatable components for derived type
- fine-grained data protection
  - private derived type components
- deferred length type parameters:
  - character(:), allocatable :: str ! Length of str can be changed at run-time
- array constructors enhancement
  - square brackets in array constructors
    `[ 1, 2, 3 ]` is equivalent to `(/ 1, 2, 3 /)`
  - allow type specification
    ```
    [ real :: 1, 2, 3, 4, 5 ]
    ```
- pointer assignment enhancement
  - allow lower bounds to be specified for pointer objects
    ```
    p(0:,0:) => a !p's lower bounds are [0,0]
    ```
  - remapping of the elements of a rank-one array
    ```
    p(1:m,1:2*m) => b !b is a rank-one array
    ```

Data manipulation and scoping enhancements (cont'd)

- allow character dummy arguments for MIN, MAX, MINLOC, MAXLOC, MINVAL and MAXVAL
- access host entities via IMPORT statement in interface block

```
integer, parameter :: dp = selected_real_kind(15)
!in the same scope of definition of dp
interface
   function distribution_function (x) result (res)
       IMPORT dp
       real(kind = dp), intent(in) :: x
       real(kind=dp) res
   end function
end interface
```

Data manipulation and scoping enhancements (cont'd)

- allocate statement with source-expr

```
integer, dimension(4,20) :: arr0
...
allocate(arr1(4,20), source=arr0)
```

- allocatable array automatic reallocation on assignment

```
real, dimension(:,:), allocatable :: A
real :: B(2, 10), C (5, 5), D(5,5)
A = B ! A is automatically allocated as 2 x 10
...
A = C ! A is automatically reallocated as 5 x 5
      !reallocation == deallocate -> allocate
A(:,:) = D !NO reallocation here
```

Data manipulation and scoping enhancements (cont'd)

- rename operators on USE statement

```
use a_mod, operator (.plus.) => operator (.add.)
```

- PROTECTED module data

```
module temperature_mod
    real, protected :: temperature
    contains
    subroutine set_temperature (temp)
        real, intent(in) :: temp
        temperature = temp
    end subroutine
end module
```

C interoperability

- provide a standardized mechanism for interoperating with C
- intrinsic module ISO_C_BINDING – contains named constant holding kind type parameter values for intrinsic types

type	named constant	C type or types
integer	C_INT	int
...		
real	C_FLOAT	float
	C_DOUBLE	double

- provide facilities of interoperability with C data, pointers, struct, and procedures
  - Interop with C global data variables

```
real(C_FLOAT),dimension(100),bind(C,name='Alpha')::alpha
```

bound to

```
float  Alpha[100];
```

C interoperability (cont'd)

- Interop with C struct

```
type, BIND(C) :: point
   real(C_FLOAT) :: x, y
end type
```

interoperable with

```
typedef struct{
   float x1, x2;
} point_t;
```

C interoperability (cont'd)

- Interop with C procedure and procedure interfaces

```
interface
   subroutine sub (i, r) bind(C, name='proc')
        integer(C_INT), VALUE :: i
        real(C_DOUBLE) r
   end subroutine
end interface
```

  - Fortran interface interoperable with C prototype

```
void proc (int, double *);
```

  - **Call sub in Fortran resolves to an invocation on a procedure named “proc”**

More C interoperability

- enumerations

```
enum, bind(C)
  enumerator :: two=2, five=5
  enumerator :: six
end enum
```

declares an enumerator with constants 2, 5 and 6

Procedure enhancements

- VALUE attribute on dummy argument
  - call by value
- abstract interface
  - declares a procedure interface without declaring an actual procedure
- declare procedures using procedure interface name (prototype)

  procedure(problem_solver) :: forward_euler, backward_euler
- procedure pointer

```
procedure(problem_solver), pointer :: solution
solution => forward_euler
```

Parameterized derived types

- derived type allowed to have KIND and LENGTH type parameters

```
integer, parameter::dp = selected_real_kind(15)
type matrix(kind,m,n)
  integer, kind :: kind=dp
  integer, len :: m, n
  real(kind) :: element(m,n)
end type

type(matrix(dp,10,20)) :: a
```

declares a double-precision matrix of size 10 x 20

```
type(matrix(dp, :, :)), allocatable :: mat
...
ALLOCATE (matrix(dp, m, n) :: mat)
```

size of matrix mat is determined at runtime

IBM XL Fortran

- standard compliance is one of the XL compiler priorities
- IBM actively participates in the standard development
- implement Fortran 2003 features since V8.1 (GA 2002)
- V12.1 contains all Fortran 2003 features except parameterized derived types
  - First OO Fortran compiler in the industry
- V12.1 supports most OpenMP3.0 features
- implemented features are available on all supported platforms
- for more information, please visit

  http://www-01.ibm.com/software/awdtools/fortran/

Q & A