The new features of Fortran 2003

David Muxworthy
BSI Fortran Convenor
Major extensions

- Standardized interoperability with C
- Object-oriented support
- Derived-type enhancements
- IEEE 754 support
- Numerous data manipulation enhancements
- Numerous input/output enhancements
Minor incompatibilities with Fortran 95

- Carriage control for printer output has been removed from the standard.
- List-directed and namelist output of real zeros is changed. Output of real zeros in these contexts always uses F rather than E format.
- If the processor can distinguish +0.0 and -0.0, use is made of this by ATAN2 and by LOG, SQRT if they have complex arguments. E.g., result of ATAN2 or LOG may be $-\pi$ rather than $\pi$. 

Creating the IT Profession
• the default character set is extended to include:
  ~ | \ [ ] { } @ # grave accent, circumflex accent (but not acute accent)
  only [ and ] are used and are synonyms for (/ and /)

• variable names may be to 63 characters

• statements may be up to 256 lines
'International' (non-anglophone) usage

- explicit accommodation of 10646 characters (but not necessary for processors to support 10646)

- ability to specify decimal separator in internal or external files as POINT or COMMA
Many new facilities, including:

- **deferred type parameters**
  length type parameters of certain entities may be changed during execution

- **VOLATILE attribute**
  for interacting with non-Fortran processes

- **ASYNCHRONOUS attribute**
  for specifying that a variable may be subject to asynchronous input/output

- **type specification for array constructors, e.g.**
  
  [CHARACTER(LEN=8) :: 'Fortran', 'C', 'Algol 68']
More data manipulation enhancements

• specification and initialization expressions are extended by removal of some restrictions

• complex literals are extended to accept named constants, e.g.
  \[(0., \text{PI})\] where \(\text{PI}\) is a previously declared real constant

• \text{MIN} and \text{MAX} extended to accept character arguments
Derived type enhancements

- the kind, length and shape of derived type components may be specified when the type is used
- different components may have different accessibility
- improved structure constructors
- finalizers
- derived-type input/output
- components may be allocatable
ASSOCIATE construct

This allows a complex expression or object to be denoted by a simple symbol

Examples:

associate ( z => exp(-(x**2+y**2)) * cos(theta) )
  print *, a+z, a-z
end associate

associate ( array => ax%b(i,:)%c )
  array(n)%ev = array(n-1)%ev
end associate
Enumerators are provided to interoperate with the corresponding C enumeration type

Example:

```c
enum, bind(c)
    enumerator :: red = 4, blue = 9
    enumerator yellow
end enum
```
Pointers are extended to point to procedures, as well as variables, using a new PROCEDURE statement.

Example:

```fortran
procedure (real_func), pointer :: p=> null()
where the interface to real_func has already been defined
...
p => bessel
write (*, *) p(2.5) !-- bessel(2.5)
```
There are many detailed enhancements, including:

• asynchronous transfer
• stream, rather than record, access
• named constants for preconnected units
• FLUSH statement
• access to error messages
• derived-type i/o
• control over rounding mode at internal to external real number conversion
If the processor supports some or all of IEC 60559 (IEEE 754) arithmetic, the standard provides facilities to:

- query which IEEE facilities are provided
- access the facilities for IEEE arithmetic, exception handling, rounding, use of certain IEEE functions, etc.
Enhanced access to system environment

new procedures, including

• GET_COMMAND
• GET_COMMAND_ARGUMENT
• GET_ENVIRONMENT_VARIABLE
• COMMAND_ARGUMENT_COUNT

IOMSG=character-variable specifier in OPEN, CLOSE, READ, WRITE
Interoperability with C

Uses intrinsic module ISO_C_BINDING to define named constants and derived types

allows for interoperability of:

• intrinsic types
• pointer types
• derived types
• scalars and arrays
• procedures and procedure interfaces
• C global variables
• C functions

but not (yet):
• procedures with data pointer, allocatable, assumed-shape array or optional dummy arguments
C Function Prototype:

```c
int C_Library_Function(void* sendbuf, int sendcount, int *recvcounts);
```

Fortran Module:

```fortran
module ftn_c

interface

integer (c_int) function C_Library_Function &
  (sendbuf, sendcount, recvcounts), &
bind(c,name='C_Library_Function')

use iso_c_binding

implicit none

type (c_ptr), value :: sendbuf

type (c_ptr), value :: recvcounts

end function C_Library_Function

end interface

end module ftn_c
```
Fortran Calling Sequence:

use iso_c_binding, only: c_int, c_float, c_loc
use ftn_c
...
real (c_float), target :: send(100)
integer (c_int) :: sendcount
integer (c_int), allocatable, target :: recvcounts(100)
...
allocate( recvcounts(100) )
...
call C_Library_Function(c_loc(send), sendcount, &
c_loc(recvcounts))
Object-oriented support

- enhanced data abstraction
  one type may extend the definition of another

- polymorphism
  allows type of a variable to vary at run time

- dynamic type allocation

- SELECT TYPE construct

- type-bound procedures
Further information

For a far more detailed (38 page) overview of the new features in Fortran 2003 relative to Fortran 95 see "The New Features of Fortran 2003" by John Reid, at