ARSC

Fortran Primer

David Newman
(from slides by Tom Logan)

Resources

- **Fortran 90 for Engineers and Scientists, Larry Nyhoff and Sanford Leestma, Prentice-Hall, 1997**
- **Fortran 90/95 Explained, Metcalf & Reid, Oxford Univ Press, 1999**
- **http://www.fortran.com/fortran/tutorials.html**
  - List of Fortran Tutorials
  - The Manchester Computer Centre materials are a nice set of notes but unfortunately in PostScript format
- **http://library.lanl.gov/numerical/bookfpdf.html (Fortran77)**
- **http://library.lanl.gov/numerical/bookf90pdf.html**
  - Both of these books have many examples and useful routines, use good coding style, & are downloadable

Source Form

- **Max. line length**
  - Fortran90: 132 chars (often more)
- **Case insensitive**
- **Variable names (usual rules)**
  - Fortran90: Max length of 31 characters (more in Fortran03)
  - Convention: lower case. Only keywords in upper case
- **Comment**
  - Fortran77 & fixed format F90: C or * in first char position of line
  - Fortran90: ! to end of line. Bang, !, generally works anywhere

Source Form (cont.)

- **Long line continuation:**
  - Fortran90: Ampersand at end of line and optionally at beginning of next
- **Semicolon ends statement**
  - Usually a poor practice to use them
- **Statement labels**
  - Fortran90: like C and other names

Program Structure

```
PROGRAM <name>
   ! name should be duplicated on end statement
IMPLICIT NONE   ! Don't implicitly declare variables
   ! declarations follow and must precede use
   ! By convention and history, declarations at beginning

CONTAINS
   ! Internal subroutines and functions follow

END PROGRAM <name>  !PROGRAM, names are optional; Use them
```

Simple Fortran90 Code

```
PROGRAM main
   IMPLICIT NONE
   REAL :: a=6.0, b=30.34, c=98.98
   REAL :: mainsum
   mainsum = add()
CONTAINS
   FUNCTION add()
        REAL :: add       ! a,b,c defined in 'main'
        add = a + b + c
   END FUNCTION add
END PROGRAM main
```

Primitive Declaration Types

```
IMPLICIT NONE  ! always use

INTEGER :: i, j = 2
   ! do not forget the double colon
 REAL :: a, b, c = 1.2
 LOGICAL, PARAMETER :: debug = .true.
! Parameter indicates a constant
 CHARACTER(20) :: name = "John"
```

Assignment Statement

- **variable = expression**

```
i = 3**2          ! Double asterisk == exponentiation
j = MOD(15, 2)

a = 'Quotes delineate strings'
b = "You can also use double quotes."
```

Operators and Their Priority

- **Same as any normal language**
  - When in doubt use parens
  - Don't study the rules
  - General normal algebra rules
    - (please excuse my dear aunt sally)

Some Intrinsic Functions

FUNCTION	DESCRIPTION	ARG TYPE	RETURN TYPE
ABS(x)	Absolute value of x	INTEGER	INTEGER
		REAL	REAL
MAX(x, y, ...)	Maximum of x, y, ...	INTEGERs	INTEGER
		REALs	REAL
SIN(x), COS(x), ...	Trig functions of x (radians for angles)	REAL	REAL
ATAN(x, y)	ArcTan of x, y triangle	REAL	REAL
EXP(x)	e^x	REAL	REAL
LOG(x)		REAL	REAL

Conversion Functions

FUNCTION	DESCRIPTION	ARG TYPE	RETURN TYPE
INT(x)	Integer part of x	REAL	INTEGER
NINT(x)	Nearest integer to x	REAL	INTEGER
FLOOR(x)	Greatest integer < or = to x	REAL	INTEGER
FRACTION(x)	Fractional part of x	REAL	INTEGER
REAL(x)	Converts x to REAL	INTEGER	REAL
MAX(x1,.., xn)	Max of x1,.. xn	INTEGER	INTEGER
		REAL	REAL
MIN(x1,.., xn)	Min of x1,.. xn	INTEGER	INTEGER
		REAL	REAL
MOD(x,y)	x - INT(x/y) * y	INTEGER	INTEGER

Input/Output

- PRINT *, 'hi'
  - Shortcut for WRITE(*,*) 'hi'

- WRITE(*,*) x, y
- READ(*,*) a, b, c ! asterisks for default values
  - First asterisk says use default unit numbers. Usually
    - 5 = stdin
    - 6 = stdout
    - System unit names often for unit 24 often like ftn24, FU24, …
  - Second asterisk says use default formatting
- READ(integer_unit_number, format-format_line)
  - Fortran “unit number” functions like a file descriptor in C
  - Formats are powerful and complex like they are in C

Input/Output

- **Full treatment of I/O is not possible here, but**
  - Binary (fast but machine dependent) OR text files
    - netCDF (from NCAR) is a blend of the two
  - Sequential and direct access
  - On-the-fly conversions between binary formats
  - Setting record lengths, block sizes, etc.
  - Special instructions for asynchronous I/O

Logical Operators

TYPE	OPERATOR	ASSOCIATIVITY
Relational	<, <=, >, >=, ==, /=	
(old style)	.LT., .LE., .GT., .GE., .EQ., .NE.	
Logical	**.NOT.**	**Right-to-left**
	.AND.	**Left-to-right**
	.OR.	**Left-to-right**
	.EQV. .NEQV.	**Left-to-right**

IF Statements

- **Single line form**

```
IF(<logical-expr>)<statement>
```

- **Multiple statement form**

```
IF ( <logical-expr> ) THEN
    <statements>
END IF
```

- **If-else form**

```
IF ( <logical-expr> ) THEN
    <statements>
ELSE
    <statements>
END IF
```

- **If-else-if form**

```
IF ( <logical-expression> ) THEN
    <statements
ELSEIF(<logical-expression>)THEN
    <statements>
ELSEIF(<logical-expression>)THEN
    <statements>
ELSE
    <statements>
END IF
```

Selection

- **SELECT CASE Statement**

```
SELECT CASE (<selector>)
  CASE (<label-list-1>)
    <statements-1>          ! Note that no overlap is
  CASE (<label-list-2>)     ! allowed so only one case is
    <statements-2>          ! executed
  CASE (<label-list-3>)
    <statements-3>
    ..........
  CASE (<label-list-n>)
    <statements-n>
  CASE DEFAULT
    <statements>
END SELECT
```

Case Statement Select Values

- **Value Range**

Syntax	Meaning
:x	all values less than or equal to x
x:	all values greater than or equal to x
x:y	the inclusive range from x to y
x	the value x

SELECT Example

```
SELECT CASE(index)
        CASE(:0)
            print *, "index is equal or less than zero"
        CASE(1: maxIndex/2-1)
            print *, "index is below mean"
        CASE(int(maxIndex/2))
            print *, "index is at mean"
        CASE(maxIndex/2+1:maxIndex)
            print *, "index is above mean"
        CASE(maxIndex+1:)
            print *, "index is greater than the max"
END SELECT
```

Iteration or Looping

- **General DO-Loop w/ EXIT**

```
DO
   Statements-1
   IF (Logical-Expr) EXIT
   Statements-2
ENDDO
```

- **Nested DO-loop:**

```
Outer: DO
   IF (expressn-1) EXIT Outer !opt
   Statements-1
   Inner: DO
      IF (expr-2) EXIT Outer !req'd
      Statements-2
   ENDDO Inner
   Statements-3
ENDDO Outer
```

- **Counting Loop**

```
DO var=init-val,final-val,step-size
   Statements
ENDDO
```

- **Default step-size is 1**

```
DO var=initial-value, final-value
   Statements
ENDDO
```

- **CYCLE: start loop over (like continue in C)**

Iteration Examples

- **Classic F77 example**

```
INTEGER count, n
REAL average, input, sum

sum = 0
DO count = 1, n
   READ *, input
   sum = sum + input
END DO
average = sum / n
   ! Implicit convrs'n n to real
```

- **Fortran90 example**

```
Integer :: i, n, factorial

READ (*,*) n
factorial = 1
DO i = 1, n
   factorial = factorial * I
ENDDO
```

Subprograms

- **Subroutines**
  - Modify arguments or COMMON (global) values
  - Not typed and not declared
  - Arguments are passed by reference
  - Invoked by CALL statement
- **Functions**
  - **Conceptually** return a value, don't modify arguments; but this is **not** enforced!
  - Typed by return value; must be declared
  - Arguments are passed by reference
  - Assign return value to function name or use RESULT clause
  - Invoked by name reference

Subroutine Example

```
SUBROUTINE swap(a,b)
   IMPLICIT NONE                        ! Good habit
   INTEGER, INTENT(INOUT):: a, b        ! INTENT is optional
   INTEGER:: tmp                        ! local
   tmp = a
   a = b
   b = tmp
END SUBROUTINE swap

! Call with:
CALL swap(x,y)                          ! Call by reference!
```

Function Example

```
REAL FUNCTION fact(k)
   IMPLICIT NONE
   INTEGER, INTENT (IN) :: k
   REAL :: f
   INTEGER :: i

   IF (k .le. 1) THEN
      fact = 1.0
   ELSE
      f = 1.0
      DO i = 1, k
          f = f * i
      END DO
      fact = f
   END IF
END FUNCTION fact
```

More Fun With Functions

- **Variables declared inside a subprogram**
  - Have local scope
  - Are “automatic” (stored on the subprogram stack)

- **A local variable becomes “static” if**
  - It is initialized in the declaration

```
INTEGER :: keeper = 0
REAL :: x(123, 0: 456)
DATA x(1, 13) / 0. /
```

  - It has the SAVE attribute

```
INTEGER, SAVE :: keeper2
```

1-D Arrays

- **Syntax**
  - <type>, DIMENSION (extent) :: name-1, name-2, ...
  - <type>, DIMENSION (lower : upper) :: <list-array-names>
- **Array operands and operators**
  - Initialization

```
a = (/ 1, 2, 3 /)
```

  - Array expressions and assignments

```
a = b + c          ! These operations are done
a = b * 3.14       ! element-wise
a = b * c
```

Array Example

```
REAL FUNCTION fact(k)
   IMPLICIT NONE
   INTEGER, INTENT (IN) :: k
   INTEGER, PARAMETER :: N = 8
   REAL :: f              ! Don't use "fact" on RHS!
   REAL :: precmp(0:N)=(/1.0,1.0,2.0,6.0,24.0,120.0,720.0,5040.0,40320.0/)
   IF (k .le. N) THEN
      fact = precmp(k)
      RETURN
   ENDIF
   f = precmp(N)
   DO i = N+1, k
      f = f * i
   END DO
   fact = f
END FUNCTION fact
```

Some 1-D Array Functions

FUNCTION	RETURNS
MAXVAL(A)	Maximum value in array A
MINVAL(A)	Minimum value in array A
MAXLOC(A)	One Dimensional array of one element containing the location of the largest element
MINLOC(A)	One Dimensional array of one element containing the location of the smallest element
SIZE(A)	Number of elements in A
SUM(A)	Sum of the elements in A
PRODUCT(A)	Product of the elements in A

Dynamic Array Allocation

- **Syntax**
  - **<type>, DIMENSION(:), ALLOCATABLE :: <list-of-array-names>**
  - **ALLOCATE (list, STAT = <status-variable>)**
  - **DEALLOCATE (list, STAT = <status-variable>)**

Dynamic Array Allocation

- **Example**

```
PROGRAM main
   IMPLICIT NONE
   INTEGER, DIMENSION(:), ALLOCATABLE :: A
   INTEGER :: aStatus, N
   WRITE(*, '(1X, A)', ADVANCE = "NO") "Enter array size: "
   READ *, N                    ! Try 1 billion on your PC!
   ALLOCATE( A(N), STAT = aStatus )
   IF (aStatus /= 0) STOP "*** Not enough memory ***"
   PRINT*, 'Array allocated with size ', N

   DEALLOCATE(A)
   PRINT*, 'Array deallocated…'
```

Multidimensional Arrays

- **Syntax**
  - type, DIMENSION (dim1,dim2,...) :: <list-array-names>
    ! Up to 7 dimensions.
    Superstrings not allowed.
  - type, DIMENSION (:, :, ...), ALLOCATABLE :: <list-array-names>
    ! Some implementations may allow more. CAF does.
  - ALLOCATE(array-name(lower1: upper1, lower2: upper2, ...) , STAT = status)
- **Examples**
  - INTEGER, DIMENSION (100,200) :: a
  - INTEGER, DIMENSION(:,:), ALLOCATABLE :: a

Multidimensional Arrays

- **Column-major ordering**
  - In Fortran, it is in column-major order: the first subscript varies most rapidly
  - **NB:** C is row-major order!
  - Yes, there are situations in which we care!
    - Varying the order of loops affects performance
    - Interfacing Fortran and C programs

Multi-D Array Functions

FUNCTION	RETURNS
MAXVAL (A,D)	Array of one less dimension containing the maximum values in array A along dimension D. If D is omitted, maximum of the entire array is returned.
MINVAL (A,D)	Like MAXVAL() but returns minima
MAXLOC (A)	One Dimensional array of one element containing the location of the largest element
MINLOC (A)	Like MAXLOC() but for smallest element
SHAPE (A)	A 1-D array of the extents of (A)
SIZE (A)	Number of elements in A

Multi-D Array Fns (cont.)

FUNCTION	RETURNS
SUM(A,D)	Array of one less dimension containing the sums of the elements of A along dimension D. If D is omitted, the sum of the elements of the entire array is returned.
PRODUCT(A)	Array of one less dimension containing the products of the elements of A along dimension D. If D is omitted, the product of the elements of the entire array is returned.
MATMUL(A,B)	Matrix product of A and B (provided result is defined)

Modules (not in Fortran77)

- **Modules - used to package**
  - Type declarations
  - Subprograms
  - Data type definitions
  - Global data
- **Forms a library that can be used in other program units**
- **Creates global variables (and constants)**

Module Syntax

- **Module definition**

```
MODULE module-name
  IMPLICIT NONE
  <specification part>

  PUBLIC  :: Name-1, Name-2, ... , Name-n
  PRIVATE :: Name-1, Name-2, ... , Name-m

CONTAINS
  internal-functions
END MODULE
```

- **Module use - use the USE to use**

```
USE module-name
```

Implicit Typing

- **If you don't use IMPLICIT NONE or put the "implicit none flag" on the compilation line variables are**
  - Integer if first letter is i, j, k, l, m, or n
  - Real for all other initial letters

- **Can be changed by IMPLICIT:**
  - IMPLICIT REAL k, COMPLEX c, &
    LOGICAL b, l, t-w

Hello World

```
PROGRAM main
  IMPLICIT NONE
  PRINT *, "Hello World"
END PROGRAM main
```

```
PROGRAM main
  IMPLICIT NONE
  CHARACTER (len =33) :: name
  READ *, name
  PRINT *, "Hello, ", name
END PROGRAM main
```

Obsolescent & Redundant Features You May See

- **Arithmetic IF**
- **CONTINUE statement/shared DO loop termination**
- **GO TO**
- **Computed GO TO**
- **COMMON blocks**
- **EQUIVALENCE**
- **"Fixed Form" Source**
  - Cols 1 to 5 for statement labels that must be integers
  - Col 6 for continuation

Backup and Redundant Slides

- Mostly about old stuff that you will need to understand to read others' programs.
- Remember: Programming is about code re-use.

Fortran77 Program Structure

```
      PROGRAM <name>
C <name> is generally allowable
       IMPLICIT NONE
C Don't implicitly declare variables
C declarations follow and must precede executable code
C
       DO 100 I=1,15
       ……
100    CONTINUE
       END
```

Simple Fortran77 Code

```
PROGRAM main
      IMPLICIT NONE


      REAL a=6.0, b=30.34, c=98.98, mainsum
      DATA a/6.0/, b/30.34/, c/98.98/

      add(b, c) = b + c

C     note statement above is a BAD function defn.

C      declarations above, executable below

      mainsum = add()


      END
```

Declaration of Fortran77 Types

```
      INTEGER  i, j
      REAL a, b, c
       LOGICAL debug
      PARAMETER         (debug = .TRUE.)
C Parameter indicates a constant
      CHARACTER(20)  name

C       Before other declaratives always use:
      IMPLICIT NONE
```

Major Differences with C

Issue	C	Fortran
End of statement	;	<end of line> ;
Line length	unlimited	132 chars
Identifier length	unlimited	31 chars (soon 63)
Subprogram structures	functions	functions, subroutines declare recursion
Array storage	row-major	column-major
indexing	0-based	1-based
Looping	for, while	do I = 1, 20
Subscripts	[ ]	( ) *parens not brackets*
Statement blocking	{ }	<key words>

More Differences with C

C	Fortran
"void" functions	Call subroutine
Subscripts start with 0	Subscripts start with 1 Seven allowed x(i1, i2, i3, i4...)
for (i=0; i<10, i++) { }	DO i = 1, 10 enddo
	DO 100 i = 1, 10 100 CONTINUE
if (...) { }	IF (...) THEN ENDIF
	IF (...) ....
Use functions everywhere	Keywords for lots of stuff write, print, read, open, ... I/O formatting usually: FORMAT
Arrays stored by column	Arrays stored by row

Some Intrinsic Functions

FUNCTION	DESCRIPTION	ARG TYPE	RETURN TYPE
ABS(x)	Absolute value of x	INTEGER	INTEGER
		REAL	REAL
SQRT(x)	Square root of x	REAL	REAL
SIN(x)	Sine of x radians	REAL	REAL
COS(x)	Cosine of x radians	REAL	REAL
TAN(x)	Tangent of x radians	REAL	REAL
EXP(x)	e^x	REAL	REAL
LOG(x)		REAL	REAL

Obsolescent/Redundant Loops

- **Fortran 77 DO loops**

```
    DO 100 I=1, N
        statements
100 CONTINUE
```

- **Redundant WHILE loop**

```
DO WHILE(logical-expr)
   statements
END DO
```

- **Equivalent to**

```
DO
   IF (logical_expr) EXIT
   statements
END DO
```