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
  - 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](http://library.lanl.gov/numerical/bookfpdf.html) (Fortran77)
- [http://library.lanl.gov/numerical/bookf90pdf.html](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`

```python
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

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>DESCRIPTION</th>
<th>ARG TYPE</th>
<th>RETURN TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS(x)</td>
<td>Absolute value of x</td>
<td>INTEGER</td>
<td>INTEGER</td>
</tr>
<tr>
<td>MAX(x, y, ...)</td>
<td>Maximum of x, y, ...</td>
<td>INTEGERs</td>
<td>INTEGER</td>
</tr>
<tr>
<td>SIN(x), COS(x), ...</td>
<td>Trig functions of x (radians for angles)</td>
<td>REALs</td>
<td>REAL</td>
</tr>
<tr>
<td>ATAN(x, y)</td>
<td>ArcTan of x, y triangle</td>
<td>REAL</td>
<td>REAL</td>
</tr>
<tr>
<td>EXP(x)</td>
<td>$e^x$</td>
<td>REAL</td>
<td>REAL</td>
</tr>
<tr>
<td>LOG(x)</td>
<td></td>
<td>REAL</td>
<td>REAL</td>
</tr>
</tbody>
</table>
## Conversion Functions

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>DESCRIPTION</th>
<th>ARG TYPE</th>
<th>RETURN TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT(x)</td>
<td>Integer part of x</td>
<td>REAL</td>
<td>INTEGER</td>
</tr>
<tr>
<td>NINT(x)</td>
<td>Nearest integer to x</td>
<td>REAL</td>
<td>INTEGER</td>
</tr>
<tr>
<td>FLOOR(x)</td>
<td>Greatest integer &lt; or = to x</td>
<td>REAL</td>
<td>INTEGER</td>
</tr>
<tr>
<td>FRACTION(x)</td>
<td>Fractional part of x</td>
<td>REAL</td>
<td>INTEGER</td>
</tr>
<tr>
<td>REAL(x)</td>
<td>Converts x to REAL</td>
<td>INTEGER</td>
<td>REAL</td>
</tr>
<tr>
<td>MAX(x1,.., xn)</td>
<td>Max of x1,.. xn</td>
<td>INTEGER</td>
<td>INTEGER</td>
</tr>
<tr>
<td>MIN(x1,.., xn)</td>
<td>Min of x1,.. xn</td>
<td>INTEGER</td>
<td>INTEGER</td>
</tr>
<tr>
<td>MOD(x,y)</td>
<td>x - INT( x/y) * y</td>
<td>INTEGER</td>
<td>INTEGER</td>
</tr>
</tbody>
</table>
Input/Output

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

- WRITE(*,*) x, y

- READ(*,*) a, b, c
  - 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

<table>
<thead>
<tr>
<th>TYPE</th>
<th>OPERATOR</th>
<th>ASSOCIATIVITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relational</td>
<td>&lt;, &lt;=, &gt;, &gt;=, ==, /=</td>
<td></td>
</tr>
<tr>
<td>(old style)</td>
<td>.LT., .LE., .GT., .GE., .EQ., .NE.</td>
<td></td>
</tr>
<tr>
<td>Logical</td>
<td>.NOT.</td>
<td>Right-to-left</td>
</tr>
<tr>
<td></td>
<td>.AND.</td>
<td>Left-to-right</td>
</tr>
<tr>
<td></td>
<td>.OR.</td>
<td>Left-to-right</td>
</tr>
<tr>
<td></td>
<td>.EQV. .NEQV.</td>
<td>Left-to-right</td>
</tr>
</tbody>
</table>
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-n>)

<statements-n>

CASE DEFAULT

<statements>

END SELECT
Case Statement Select Values

- Value Range

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>:x</td>
<td>all values less than or equal to x</td>
</tr>
<tr>
<td>x:</td>
<td>all values greater than or equal to x</td>
</tr>
<tr>
<td>x:y</td>
<td>the inclusive range from x to y</td>
</tr>
<tr>
<td>x</td>
<td>the value x</td>
</tr>
</tbody>
</table>
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!
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
    \[ a = b * c \]
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

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>RETURNS</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAXVAL(A)</td>
<td>Maximum value in array A</td>
</tr>
<tr>
<td>MINVAL(A)</td>
<td>Minimum value in array A</td>
</tr>
<tr>
<td>MAXLOC(A)</td>
<td>One Dimensional array of one element containing the location of the largest element</td>
</tr>
<tr>
<td>MINLOC(A)</td>
<td>One Dimensional array of one element containing the location of the smallest element</td>
</tr>
<tr>
<td>SIZE(A)</td>
<td>Number of elements in A</td>
</tr>
<tr>
<td>SUM(A)</td>
<td>Sum of the elements in A</td>
</tr>
<tr>
<td>PRODUCT(A)</td>
<td>Product of the elements in A</td>
</tr>
</tbody>
</table>
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(:,:) :: 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

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>RETURNS</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAXVAL (A,D)</td>
<td>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.</td>
</tr>
<tr>
<td>MINVAL (A,D)</td>
<td>Like MAXVAL() but returns minima</td>
</tr>
<tr>
<td>MAXLOC (A)</td>
<td>One Dimensional array of one element containing the location of the largest element</td>
</tr>
<tr>
<td>MINLOC (A)</td>
<td>Like MAXLOC() but for smallest element</td>
</tr>
<tr>
<td>SHAPE (A)</td>
<td>A 1-D array of the extents of (A)</td>
</tr>
<tr>
<td>SIZE (A)</td>
<td>Number of elements in A</td>
</tr>
</tbody>
</table>
### Multi-D Array Fns (cont.)

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>RETURNS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUM(A,D)</td>
<td>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.</td>
</tr>
<tr>
<td>PRODUCT(A)</td>
<td>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.</td>
</tr>
<tr>
<td>MATMUL(A,B)</td>
<td>Matrix product of A and B (provided result is defined)</td>
</tr>
</tbody>
</table>
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
  - Col 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

<table>
<thead>
<tr>
<th>Issue</th>
<th>C</th>
<th>Fortran</th>
</tr>
</thead>
<tbody>
<tr>
<td>End of statement</td>
<td>;</td>
<td>&lt;end of line&gt; ;</td>
</tr>
<tr>
<td>Line length</td>
<td>unlimited</td>
<td>132 chars</td>
</tr>
<tr>
<td>Identifier length</td>
<td>unlimited</td>
<td>31 chars (soon 63)</td>
</tr>
<tr>
<td>Subprogram structures</td>
<td>functions</td>
<td>functions, subroutines</td>
</tr>
<tr>
<td></td>
<td></td>
<td>declare recursion</td>
</tr>
<tr>
<td>Array storage</td>
<td>row-major</td>
<td>column-major</td>
</tr>
<tr>
<td>index,ing</td>
<td>0-based</td>
<td>1-based</td>
</tr>
<tr>
<td>Looping</td>
<td>for, while</td>
<td>do l = 1, 20</td>
</tr>
<tr>
<td>Subscripts</td>
<td>[ ]</td>
<td>( ) parens not brackets</td>
</tr>
<tr>
<td>Statement blocking</td>
<td>{ }</td>
<td>&lt;key words&gt;</td>
</tr>
</tbody>
</table>

Arctic Region Supercomputing Center
## More Differences with C

<table>
<thead>
<tr>
<th>C</th>
<th>Fortran</th>
</tr>
</thead>
<tbody>
<tr>
<td>“void” functions</td>
<td>Call subroutine</td>
</tr>
<tr>
<td>Subscripts start with 0</td>
<td>Subscripts start with 1</td>
</tr>
<tr>
<td></td>
<td>Seven allowed x(i1, i2, i3, i4...)</td>
</tr>
<tr>
<td>for (i=0; i&lt;10, i++) { }</td>
<td>DO i = 1, 10</td>
</tr>
<tr>
<td></td>
<td>enddo</td>
</tr>
<tr>
<td></td>
<td>DO 100 i = 1, 10</td>
</tr>
<tr>
<td></td>
<td>100 CONTINUE</td>
</tr>
<tr>
<td>if ( ... ) { }</td>
<td>IF (...) THEN</td>
</tr>
<tr>
<td></td>
<td>IF (...) ....</td>
</tr>
<tr>
<td></td>
<td>ENDIF</td>
</tr>
<tr>
<td>Use functions everywhere</td>
<td>Keywords for lots of stuff</td>
</tr>
<tr>
<td></td>
<td>write, print, read, open, ...</td>
</tr>
<tr>
<td></td>
<td>I/O formatting usually: FORMAT</td>
</tr>
<tr>
<td>Arrays stored by column</td>
<td>Arrays stored by row</td>
</tr>
</tbody>
</table>
### Some Intrinsic Functions

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>DESCRIPTION</th>
<th>ARG TYPE</th>
<th>RETURN TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS(x)</td>
<td>Absolute value of x</td>
<td>INTEGER</td>
<td>INTEGER</td>
</tr>
<tr>
<td>SQRT(x)</td>
<td>Square root of x</td>
<td>REAL</td>
<td>REAL</td>
</tr>
<tr>
<td>SIN(x)</td>
<td>Sine of x radians</td>
<td>REAL</td>
<td>REAL</td>
</tr>
<tr>
<td>COS(x)</td>
<td>Cosine of x radians</td>
<td>REAL</td>
<td>REAL</td>
</tr>
<tr>
<td>TAN(x)</td>
<td>Tangent of x radians</td>
<td>REAL</td>
<td>REAL</td>
</tr>
<tr>
<td>LOG(x)</td>
<td></td>
<td>REAL</td>
<td>REAL</td>
</tr>
<tr>
<td>EXP(x)</td>
<td>$e^x$</td>
<td>REAL</td>
<td>REAL</td>
</tr>
</tbody>
</table>
Obsolescent/Redundant Loops

- **Fortran 77 DO loops**
  
  ```fortran
  DO 100 I=1, N
  statements
  100 CONTINUE
  ```

- **Redundant WHILE loop**
  
  ```fortran
  DO WHILE(logical-expr)
    statements
  END DO
  ```

- **Equivalent to**
  
  ```fortran
  DO
    IF (logical_expr) EXIT
    statements
  END DO
  ```