Introduction to Fortran

Ed Kornkven
PHYS F693
Spring 2004

Resources

• http://www.csi.mtu.edu/cs2911/www/FortranReview.htm
  – The source of these notes - some bugs
• 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
• Fortran 90 for Engineers and Scientists, Larry Nyhoff and Sanford Leestma, Prentice-Hall, 1997
Source Form

- Max. line length of 132 characters
- Case insensitive
- Identifiers
  - Begin with alpha
  - Contain alphanumeric and underscore
  - Max. length of 31 characters
- Exclamation begins comment

Source Form (cont.)

- Ampersand (&) at end of line indicates continuation
- Semicolon separates multiple statements on a line
- Statement labels are integers
Source Form (cont.)

PROGRAM name
   ! Exclamation mark indicates comment

   IMPLICIT NONE   ! Don't implicitly declare variables
   ! declarations follow and must precede use

   CONTAINS
   ! Internal subroutines and functions follow

   END PROGRAM name

Prototype 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
Declaration of Primitive Types

INTEGER :: i, j = 2  
    ! do not forget the double :: without spaces
REAL :: a, b, c = 1.2
LOGICAL, PARAMETER :: debug = .true.  
    ! Parameter indicates a constant
CHARACTER(20) :: name = "John"

IMPLICIT NONE  ! always use

Assignment Statement

• variable = expression

I = 3**2
J = MOD(15, 2)

A = 'Quotes delineate character literal'
B = "Can also use double quotes."
### Operators and Their Priority

<table>
<thead>
<tr>
<th>Arithmetic Operator</th>
<th>Symbol</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exponentiation</td>
<td>**</td>
<td>Right-to_left</td>
</tr>
<tr>
<td>Multiplication and division</td>
<td>* /</td>
<td>Left-to-right</td>
</tr>
<tr>
<td>Addition and subtraction</td>
<td>+ -</td>
<td>Left-to-right</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>EXP(x)</td>
<td>ex</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

- 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
- READ(*,*) a, b, c ! asterisks for default values
  - Traditional default unit numbers are
    - 5 = stdin
    - 6 = stdout
- PRINT format,
- PRINT *, 'hi'
  - Shortcut for WRITE(*,*)
Input/Output

• Full treatment of I/O is important for scientific computing but not possible here -- e.g.:
  – Binary and text files
  – Sequential and direct access
  – On-the-fly conversions between different binary formats
  – Setting record lengths, block sizes, etc.
  – Special instructions for asynchronous I/O

Logical Operators and Precedence

<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>None</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

• **Else-if form**
  IF (logical-expression) THEN
  statements
  ELSE IF (logical-expression) THEN
  statements
  ELSE IF (logical-expression) THEN
  statements
  ELSE
  statements
  END IF

Selection

• **SELECT CASE Statement**
  SELECT CASE (selector)
  CASE (label-list-1)
    statements-1
  CASE (label-list-2)
    statements-2
  CASE (label-list-3)
    statements-3
    ..........
  CASE (label-list-n)
    statements-n
  CASE DEFAULT
    statements-DEFAULT
  END CASE
SELECT Labels

- A label has one of the following four forms:

<table>
<thead>
<tr>
<th>Label</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>:x</td>
<td>all values less or equal x</td>
</tr>
<tr>
<td>x:</td>
<td>all values greater or equal to x</td>
</tr>
<tr>
<td>x:y</td>
<td>all values in the range of x and y</td>
</tr>
<tr>
<td>x</td>
<td>the values of x itself</td>
</tr>
</tbody>
</table>

SELECT Example

```plaintext
SELECT CASE(index)
    CASE(:0)
        print *, "index is equal or less then zero"
    CASE(1: maxIndex)
        print *, "index is in range"
    CASE( int(maxIndex/2) )
        print *, "index is at mean"
    CASE(maxIndex+1:)
        print *, "index is greater the max"
END CASE
```
Iteration

- General DO-Loop w/ EXIT
  DO
  Statements-1
  IF (Logical-Expr) EXIT
  Statements-2
  END DO

- Nested DO-loop:
  Outer: DO
  IF (expression-1) EXIT Outer
  Statements-1
  Inner: DO
    IF (expression-2) EXIT Inner
    Statements-2
  END DO Inner
  Statements-3
  END DO Outer

- Counting Loop
  DO var=init-val,final-val,step-size
  Statements
  END DO

- Default step-size is 1
  DO var=initial-value, final-value
  Statements
  END DO

- CYCLE: jump up to the top of loop and increment (like continue in C)

Iteration Examples

- Classic example
  Sum = 0
  DO Count = 1, N
    READ*, Input
    Sum = Sum + Input
  END DO
  Average = REAL(Sum) / REAL(N)

- Another example
  READ(*,*) N
  Factorial = 1
  DO I = 1, N
    Factorial = Factorial * I
  END DO
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
  ```

Subprograms

- Subroutines
  - Don’t return a value except by modifying arguments
  - Therefore, 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
  - Invoked by name reference
**Subroutine Example**

```fortran
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**

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

  IF (k .lt. 1) THEN
    fact = 0.0
  ELSE
    f = 1.0
    DO i = 1, k
      f = f * i
    END DO
    fact = f
  END IF
END FUNCTION fact
```
More Fun With Functions

• **Look, Ma! Fortran can do recursion!**
  – Just declare the routine to be recursive and specify the result value

```fortran
RECURSIVE REAL FUNCTION fact3(k) RESULT (fact)
IMPLICIT NONE
INTEGER, INTENT (IN) :: k

IF (k .lt. 1) THEN
  fact = 0.0
ELSE IF (k .eq. 1) THEN
  fact = 1.0
ELSE
  fact = k * fact3(k-1)
END IF
END FUNCTION fact3
```

More Fun With Functions

• **By default, variables declared inside a subprogram have**
  – Local scope
  – “Automatic” lifetime

• **A local variable can be given a “static” lifetime by either**
  – Initializing it
    INTEGER :: keeper = 0
  – Or giving it the SAVE attribute when declaring it
    INTEGER, SAVE :: keeper
Expressions & Assignments

• Data types
  – Implicit typing rules

• Operators

• Library functions

• Assignments

1-D Arrays

• Syntax
  – type, DIMENSION ( extent ) :: name-1, name-2, ...
  – type, DIMENSION( lower : upper) :: list-array-names

• Array operands and operators
  – Initialization
    \[ A = (\langle 1, 2, 3 \rangle) \]
  – Array expressions and assignments
    \[ A = B + C \quad \text{! These operations are done} \]
    \[ A = B \times 3.14 \quad \text{! element-wise} \]
### Array Example

```fortran
REAL FUNCTION fact(k)
INTEGER, INTENT (IN) :: k
INTEGER, PARAMETER :: N = 8
REAL :: f
! Don't use "fact" on RHS!
REAL :: precmp(N)=(/1.0, 2.0, 6.0, 24.0, 120.0, 720.0, 5040.0, 40320.0/)
IF (k .lt. 1) THEN
  fact = 0.0
ELSE IF (k .le. N) THEN
  fact = precmp(k)
ELSE
  f = precmp(N)
  DO i = N+1, k
    f = f * i
  END DO
  fact = f
END IF
END FUNCTION fact
```

### Some 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)

• Example

PROGRAM main
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...
END PROGRAM main
Multidimensional Arrays

• Syntax
  – type, DIMENSION( dim1,dim2,...) :: list-array-names
  
  – type, DIMENSION(:,;,...), ALLOCATABLE :: list-array-names
  
  – ALLOCATE(array-name( lower1: upper1, lower2: upper2) ,
    STAT = status)

• Examples
  – INTEGER, DIMENSION(100,200) :: a
  – INTEGER, DIMENSION(:,;), ALLOCATABLE :: a

Multidimensional Arrays

• Column-major ordering
  – Suppose we have the declaration
    INTEGER, DIMENSION(100,200) :: a
  – The usual way of viewing a 2-D array reference is
    • A(row,column) - first dimension is “rows”, second is “columns”
  – Of course, computer memory is 1-D. How is virtual 2-D array mapped to real memory?
    • In Fortran, it is in column-major order -- i.e., column-by-column
    • NB: C is row-major order!
  – Yes, there are situations in which we care!
    • Certain performance situations
    • Interfacing Fortran and C
## 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 for 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>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

- **Modules - used to package together**
  - Type declarations
  - Subprograms
  - Data type definitions
- **Forms a library that can be used in other program units**

Module Syntax

- **Module definition**
  
  MODULE module-name
  IMPLICIT NONE
  specification part

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

  CONTAINS
  internal-functions
  END MODULE

- **Module use - use the USE to use**
  USE module-name
New Fortran Features That We Can’t Cover

- Pointers
- User-defined types and operations
- INTERFACE blocks
- KIND
- Others -- can you say “C++”? 

Obsolescent & Redundant Features

- Arithmetic IF
- PARAMETER
- DATA
- GO TO
- Computed GO TO
- COMMON blocks
- EQUIVALENCE
NEC/Cray SX-6 Architecture Overview

SX-6 Architecture

Arctic Region Supercomputing Center

AF
SX-6 Overview

• SX-6 Node
  – 8 CPUs
  – 64 GB Memory

• Performance
  – 64 GFLOPS per Node

• SUPER-UX operating system
  – 32- and 64-bit IEEE

SX-6 Programming Environment

<table>
<thead>
<tr>
<th>SX-6 Languages / Programming Models</th>
<th>Fortran 90</th>
<th>C++ / C</th>
<th>HPF</th>
<th>Co-array Fortran</th>
<th>UPC</th>
<th>Auto par.</th>
<th>OpenMP</th>
<th>MPI</th>
<th>MPI-2</th>
<th>SHMEM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

("almost all")
### CPU Performance Overview

<table>
<thead>
<tr>
<th></th>
<th>SX-6</th>
<th>SV1ex</th>
<th>X1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector pipes</td>
<td>8</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>Vector regs.</td>
<td>8 x 256 x 64</td>
<td>8 x 64 x 64</td>
<td>128 x 64 x 64</td>
</tr>
<tr>
<td>Clock (mHz)</td>
<td>500</td>
<td>500</td>
<td>800</td>
</tr>
<tr>
<td>Peak perf. (GFLOPS)</td>
<td>8</td>
<td>2</td>
<td>12.8</td>
</tr>
<tr>
<td>Mem. BW (GB/sec)</td>
<td>31.9</td>
<td>3.6</td>
<td>34.1</td>
</tr>
</tbody>
</table>

**Earth Simulator - 40 TFlops**

- 640 SX-6 cabinets
- x 8 Pes each
- = 40960 GFLOPS
- > combined 5 nearest competitors

“Real Performance”?  
- SC 2002 Gordon Bell prize winner  
- Sustained rates exceeding 25 TFLOPS
Compiling on SX-6

• Gateway machine
  – rimegate.arsc.edu
  – An SGI front end
  – Fast compiles using SX cross-compilers

• SX-6
  – rime.arsc.edu
  – Limited disk space, no backups
  • [link](http://www.arsc.edu/support/howtos/usingsx6.html)

Compiling on SX-6 (cont.)

• Summary
  – Give Fortran source files a suffix of .f90
  – Rime compile: f90 file.f90
  – Rimegate compile: sxf90 file.f90
  – Execute (must be on Rime!): ./a.out
  – For compiler options: man {sx}f90
  – NQS batch system
**Compiling on SX-6 (cont.)**

- **Important compiler options**
  - `-ew`: 64-bit
  - `-Wf",-pvctl fullmsg -L fmtlist"` : listing details
  - `-pi`: automatic inlining

**SX-6 Source File Name Defaults**

<table>
<thead>
<tr>
<th></th>
<th>Fixed Format</th>
<th>Free Format</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Preprocessor Invoked</strong></td>
<td>.f</td>
<td>.f90</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.ftn</td>
</tr>
<tr>
<td><strong>Preprocessor Not Invoked</strong></td>
<td>.F</td>
<td>.F90</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.FTN</td>
</tr>
</tbody>
</table>