Fortran 90/95 Programming
High Performance Computing and Programming, Winter 2005
http://www.it.uu.se/edu/course/homepage/hpb/vt05

Why?
- Lots of legacy code
  - Mainly Fortran 77
- Good support for arrays
  - Especially multi-dimensional ones
- Made for number crunching
  - Intrinsic functions
- Looks like Matlab
  - Faster
  - Less coding compared to C

History of FORTRAN
- FORTRAN
  - FORMula TRANslator
  - John Backus (IBM) 1952
- One of the first programming languages
  - Several standards

<table>
<thead>
<tr>
<th>Year</th>
<th>Standard</th>
</tr>
</thead>
<tbody>
<tr>
<td>1966</td>
<td>FORTRAN66</td>
</tr>
<tr>
<td>1978</td>
<td>FORTRAN77</td>
</tr>
<tr>
<td>1991</td>
<td>Fortran90</td>
</tr>
<tr>
<td>1997</td>
<td>Fortran95</td>
</tr>
<tr>
<td>Nov 2004</td>
<td>Fortran2003</td>
</tr>
</tbody>
</table>

FORTRAN77
- Fortran77 is about the size of C
  - No dynamic memory allocation
  - No pointers
  - No derived types
- Implicit typing
  - Variable names starting with letters i,j,n are integers otherwise floating point
- Fixed source form
  - Every line 72 characters
  - Position 1 to 5 reserved for labels
  - Positions 7 to 72 FORTRAN statements
### Fortran 90/95

- Fortran95 “bugfix” to Fortran90
  - Small differences
- Modern programming language
  - Derived types
  - Dynamic memory allocation
  - Pointers
  - Operator overloading
- Most compilers commercial
- Open Source compilers
  - http://gfortran.org
  - http://www.g95.org

### Writing Fortran Programs

- Source is not case-sensitive
- Lines are terminated after 132 tokens (words or symbols)
  - No semicolon like in C
- Lines can be continued up to 39 times using the & sign
- Comments are written using !
- Separate short statements using ;

```fortran
A line containing only a comment
B = c ; q = 1.0 ! Comment
c = 3.1415926*sin(log(atan(q_1))) &
    q_2*cos(q_3) ! A continuation
```

### Intrinsic types

- Basic data types
- Precision and size determined by `kind`
  - More on that later

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>Integer type, both signed and unsigned</td>
</tr>
<tr>
<td>real</td>
<td>Floating Point number</td>
</tr>
<tr>
<td>logical</td>
<td>Boolean type (.true. and .false.)</td>
</tr>
<tr>
<td>character</td>
<td>ASCII</td>
</tr>
<tr>
<td>complex</td>
<td>Built in support for complex numbers</td>
</tr>
</tbody>
</table>

### Program Units

```fortran
PROGRAM program_name

<statements>

END PROGRAM program_name
```

"main()"
### Exponentiation Operator

No implicit typing

#### Variable Declaration

```fortran
integer :: n, k
real :: x, y
```

Here comes the code

```fortran
write(*,*) 'Give to numbers'
read(*,*) k, x
n = 2**k
y = sin(x)
write(*,*) n, y
```

### Example, program

```fortran
PROGRAM my_program
  implicit none
  integer :: n, k
  real :: x, y
  ! Here comes the code
  write(*,*) 'Give to numbers'
  read(*,*) k, x
  n = 2**k
  y = sin(x)
  write(*,*) n, y
END PROGRAM my_program
```

### More on types

- Every variable has a **kind**, a positive integer that determines its precision and range
- Use the intrinsic function **kind()** to check the kind of a variable
- In some implementations, kind() returns the number of bytes reserved for the representation.

### Kind, examples

- To allocate an integer that can handle six digits, i.e., \(-999999 < n < 999999\)

```fortran
INTEGER, PARAMETER  :: I6 = SELECTED_INT_KIND(6)
INTEGER(KIND=I6) :: n
n = 456_I6
```

### Floating Point Types

- Old codes can use the old Fortran 77 way of declaring real variables of double precision
- To create a type that follows the IEEE standard, use the following:

```fortran
INTEGER, PARAMETER  :: OLD = KIND(0.0)
INTEGER, PARAMETER  :: SP = SELECTED_REAL_KIND(6, 37)
INTEGER, PARAMETER  :: DP = SELECTED_REAL_KIND(15, 307)
INTEGER, PARAMETER  :: QP = SELECTED_REAL_KIND(33, 4931)
REAL(KIND=SP)       :: x
REAL(KIND=QP)       :: y
x = 3.141459_SP
y = 0.343234_QP
```
Type conversion

• Types can be converted using the `real()` and `int()` intrinsics.
• Always use explicit type conversion

```fortran
n = int(a)
y = real(z)
```

• Dangers of implicit type conversion:

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>3/6</td>
<td>0</td>
</tr>
<tr>
<td>3/6.0</td>
<td>0.5</td>
</tr>
<tr>
<td>2**(-2)</td>
<td>0.25</td>
</tr>
</tbody>
</table>

Complex Numbers

• Fortran 90/95 has an intrinsic complex data type

```fortran
COMPLEX :: z = ( 1., .99e-2 )
```

• COMPLEX variables can be used just like other variables

Intrinsic Arithmetical Functions

- `abs(x)`, `nint(x)`, `round(x)`, `floor(x)`, `ceiling(x)`, `mod(i,j)`, `sqrt(x)`, `sin(x)`, `exp(x)`, `log(x)`, `log10(x)`

• They also work for complex numbers

Conditionals

```fortran
if (condition) then
  <statements>
else if (condition) then
  <statements>
  ... else <statements> end if
```

• A “select” statement is also available
Loops

do <variable> = <start>,<stop>,<step>
<statements>
if (condition) exit
<statements>
if (condition) cycle
<statements>
end do

• The loop control block (start,stop,step)
  may be omitted to form a C-style while
  loop

Functions in Fortran 95

PROGRAM my_program
IMPLICIT NONE
REAL, PARAMETER :: PI = 3.14159
REAL :: area, radius
area = calculate_area(r)
CONTAINS
FUNCTION calculate_area(radius) result r
IMPLICIT NONE
REAL :: r
REAL, INTENT(IN) :: radius
r = PI * (radius**2)
END FUNCTION calculate_area
END PROGRAM my_program

Subroutines in Fortran 95

PROGRAM my_program
IMPLICIT NONE
REAL, PARAMETER :: PI = 3.14159
REAL :: area, radius
CALL calculate_area(radius,area)
CONTAINS
SUBROUTINE calculate_area(radius,area)
IMPLICIT NONE
REAL, INTENT(OUT) :: area
REAL, INTENT(IN) :: radius
area = PI * (radius**2)
END SUBROUTINE calculate_area
END PROGRAM my_program

Dummy Arguments

• Subroutines and functions are called
  procedures
• Arguments to procedures are called
dummy arguments and they are
  associated with actual arguments each
time the call is executed
• It is up to the implementation to decide if
  the dummy arguments are copies or
  references
  – Most implementations use references
• The type of actual arguments and dummy
  arguments must agree
More on Arguments

- Arguments can be explicitly associated at the call
  - Keyword Parameters
  - Optional Parameters
- Arguments can have intents to show the compiler how they are accessed
  - INTENT(IN)
  - INTENT(OUT)
  - INTENT(INOUT)
- Used frequently in intrinsic procedures

Parameters, example

```fortran
PROGRAM parameter_demo
  integer :: m
  real :: x, y, z
  m = 1; x = -1.0; y = -2.0; z = -3.0
  call sub(m,x,y,z)
  call sub(m,a=x,b=y,c=z)
  call sub(m,c=z,a=x,b=y)
  call sub(m,x,y)
  call sub(m,b=y)
END PROGRAM parameter_demo
```

Arrays, terminology

- Rectangular set of elements, all of the same type
  ```fortran
  REAL, DIMENSION(10,20) :: matrix
  ```
- The number of elements of an array is called its size
- The number of dimensions of an array is called its rank
  - Maximum rank is 7
- The number of elements along each dimension is called its extent

Arrays, indexing

- By default, arrays are indexed from 1 to its extent
  - Not from 0 as in C, C++, or JAVA
- However, indexing can be arbitrarily stated at the declaration
  ```fortran
  REAL, DIMENSION(-10:5,-20:-1,0:1,8) :: grid
  ```
- The sequence of extents is called the shape of an array
  ```fortran
  (16,20,2,8)
  ```
Array Constants

- Arrays can be initialized using array constants
  
  ```fortran
  integer, dimension(1:3) :: a
  a = (/ 10, 20, 40 /)
  ```

- Or, using the data statement
  
  ```fortran
  integer, parameter :: length = 200
  real, dimension(1:length) :: r
  data r(1:length)/length*0.0/
  ```

Array Syntax

- Arrays can be indexed using Matlab-style indices
  
  ```fortran
  integer, dimension(1:3) :: index
  real, dimension(1:10) :: V
  integer, dimension(0:2,1:10) :: A
  V(1:5)   First five elements of V
  V(:5)    As above
  V(6:10)  Last five elements of V
  V(:)     All of V
  V(1:5:2) V(1), V(3), V(5)
  V(2::2)  Even indices
  A(:,1:5) First five columns of A
  V(index) V(index(1)), V(index(2)), V(index(3))
  ```

Array Arithmetic

- Arrays can be used in arithmetic expressions
- They are evaluated elementwise
- Arrays must conform, ie have the same shape
- In an assignment, the right-hand side is completely evaluated before the assignment is performed
- Some intrinsics perform linear algebra

```fortran
integer, dimension(1:3) :: index
real, dimension(1:10) :: V, W
integer, dimension(0:2,1:10) :: A
complex, dimension(1:10,1:10) :: C
V(1:4) + W(3:6)
2*V + A(1,:)
C = C**(-2)
sin(A)
dot_product(V,W)
matmul(C,C)
transpose(A)
maxval(A)
sum(A)
```