Fortran Programming Language

Fortran (FORmula TRANslation) is a high level language aimed at numerical calculations. Evolved over 30-40 years. All compilers should conform to the Fortran standard so that code is completely portable.

Recommended textbooks:

- Fortran 90 Explained, Metcalf and Reid, Oxford Science Publications
- Fortran 90/95 Explained, Metcalf and Reid, Oxford University Press
- Fortran 95/2003 Explained, Metcalf, Reid and Cohen, Oxford University Press

Internet resources are

- http://www.fortran.com contains a FAQ and lots of other information.
- news://comp.lang.fortran to speak directly to those who develop Fortran

Fortran Program Structure

Here is an example Fortran code.

```
program circle
real  :: r, area
!This program reads a real number r and prints
!the area of a circle with radius r
read (*,*) r
area = 3.14159*r*r
write (*,*)' Area = ',area
stop
end program circle
```

Note that all variables are declared at the beginning of the program and before they are used.

Declaration of Variables

Start every program with the statement

```
implicit none
```

This tells the compiler that all variables are to be declared and the compiler will report an error if any variable is not declared. This is good programming practice. Do not rely on the compiler to decide if variables are integers or real. This is different from e.g. Matlab

Declaration of Variables (Contd)

Variables can be of the following types:

- LOGICAL
- CHARACTER
- INTEGER
- REAL
- COMPLEX
- user constructed called derived types

Declaration of Variables (Contd)

For example the following all declare x as a single precision real variable, that is, its representation in memory is 4 bytes long

- real :: x
- real(4) :: x
- real*4 :: x

A double precision variable z which takes 8 bytes of memory can be represented in any of the following ways:

- real(8) :: z
- real*8 :: z
- double precision :: z

Declaration of Variables (Contd)

- COMPLEX (COMPLEX*8 or COMPLEX(4)) - single precision complex number
- COMPLEX(8) or COMPLEX*16 - double precision complex number
- CHARACTER(LEN=n), CHARACTER(n), or CHARACTER*n where n is an integer value represents the number of bytes in the character variable or can be *.
- LOGICAL can be .TRUE. or .FALSE.

Declaring arrays

Here are some examples of array declarations, there are several ways to do this.

- a is a real one-dimensional array of length 4, indexed 1 to 4.

```
real :: a(4)
```

- a is a real one-dimensional array of length 20

```
integer, parameter :: n=20
real  :: a(n)
```

- b is an integer one-dimensional array of length 20, indexed 0 to 19

```
integer :: b(0:19)
```

- c is double precision 2-dimensional array

```
double precision, dimension(10,10) :: c
```

In these the array a has elements `a(1)` to `a(20)`, b has elements `b(0)` to `b(19)` and c has elements `c(1,1)` to `c(10,10)`

Example 1

Compile and execute the following code `variables.f90` by doing

```
>f90 variables.f90 -o variables
>./variables
```

```
    program variables

    implicit none

    integer    :: i
    integer(2) :: j
    integer(4) :: k
    integer(8) :: l

    real    :: a
    real(4) :: b
    real(8) :: c

    write (*,*) ' Huge:   ',huge(i),huge(j),huge(k),huge(l)
    write (*,*) ' Digits: ',digits(i),digits(j),digits(k),digits(l)

    write (*,*) ''

    write (*,*) 'Huge:   ',huge(a),huge(b),huge(c)
    write (*,'(a,i,i,i)') 'Digits: ',digits(a),digits(b),digits(c)
    write (*,'(a8,e,e,e)') 'Epsilon: ',epsilon(a),epsilon(b),epsilon(c)
    write (*,'(a9,en,en,en)') 'Epsilon: ',epsilon(a),epsilon(b),epsilon(c)
    write (*,'(a10,3es)') 'Epsilon: ',epsilon(a),epsilon(b),epsilon(c)
    write (*,'(a11,3e20.10e4)') 'Epsilon: ',epsilon(a),epsilon(b),epsilon(c)

    end program variables
```

Example 1 continued

The output from variables.f90 is

```
 Huge:     2147483647  32767  2147483647   9223372036854775807
 Digits:           31          15          31          63

 Huge:     3.4028235E+38  3.4028235E+38  1.797693134862316E+308
Digits:           24          24          53
Epsilon:  0.1192093E-06  0.1192093E-06   0.2220446049250313E-15
Epsilon: 119.2092896E-09119.2092896E-09 222.0446049250313081E-18
 Epsilon:   1.1920929E-07  1.1920929E-07   2.2204460492503131E-16
  Epsilon:   0.1192092896E-0006  0.1192092896E-0006  0.2220446049E-0015
```

Note how the output is presented and how this relates to the `write` statements.

- * is a special character meaning standard or default
- `a` represents ASCII characters
- `i` represents integers
- `e` represents exponential notation
- `es` represents scientific notation (similar to `e`)
- `en` represents engineering notation
- and more

`huge` and `digits` represent intrinsic functions in the Fortran language and are defined in the standard.

`huge` returns the largest number representable by a number of the same type and kind as the argument.

`digits` returns the number of significant digits for numbers of the same type and kind as the argument.

Numeric Expressions

+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation

There is an order of precedence with ** the highest followed by * and /. Next are the unary operators + and - and last the binary operators + and -. For example,

A**B*C is evaluated as (A**B)*C

A/B*C is evaluated as (A/B)*C

Use parantheses to force a particular order of evaluation.

Data Type of Numeric Expressions

If every operand in a numeric expression is of the same type, the result is also of that type.

If operands of different data types are combined in an expression, the data type of the result is the same as the highest ranking operand. For example,

```
double precision :: x, y
y = x*2
```

The integer constant 2 will be promoted to double precision 2.0D0 before doing the multiplication.

Best not to rely on this but be sure that all expressions contain variables or constants of the same type.

Integer Arithmetic

Care must be taken when using all integer variables. The code segment

```
real    :: a
a = 3/2
```

will produce the result a=1.0 not a=1.5

Compile and run the code exercise_int.f90 and see how the value of a can be calculated.

Do Loops

Loops are used for repeated of execution of similar instructions. For example,

```
program looping
implicit none
integer  :: i, n
real*8   :: sum
n = 10
sum = 0.0D0
do i=1,n
  sum = sum + dble(i)
  write (*,*) ' Value of sum at step ',i,' is ',sum
enddo
stop
end program looping
```

Do Loops (Contd)

Things to note:

- Loop is bounded by do and enddo
- This can be replaced by a statement number such as

```
      do 5 i=1,n
         sum = sum + dble(i)
5     continue
```

- Loop index is an integer.
- There can be loops within loops using a different index.
- Note 0.0D0 meaning zero in double precision.
- Note the matching of types in the summation step.

Exercise 1

Write Fortran code to calculate and print out the value of the variable a where a=5+1/i for i=1,..,n and n=10.

Then modify the code so that a is a one-dimensional array and the iteration of the do loop for i calculates a(i).