Introduction to Fortran 77

Written by Pete Pokrandt AOS

Versions of Fortran

* Fortran 66
* Fortran 77 <---
* Fortran 90

Fortran Basics

```
c234567890123456789012345678901234567890...
      program circle
      real r, area

C This Program reads a real number r and
C prints the area of a circle with radius r.

      write (*,*) 'Please enter radius r:'
      read  (*,*) r
      area = 3.14159*r*r
      write (*,*) 'Area = ',area

      stop
      end
```

Program Organization

```
123456789012345...
      program name

      declarations

      statements

      stop
      end
```

Column Position Rules

```
Column 1     : Blank or a "C" or "*" for comments
Column 2-5   : Statement Label (optional)
Column 6     : Continuation of previous line (optional)
Column 7-72  : Statements
Column 73-80: Sequence Number (optional, rarely used)
```

Most lines start with 6 blank spaces, and end before column 72, i.e. only the statement part is used.

Blank Spaces are ignored, except in character strings.

For more info..

Check on the web.

http://www.aos.wisc.edu/~poker/fortran_tutorial/
is a FORTRAN tutorial.

http://www.aos.wisc.edu/~poker/unixhelp.html
has some links to unix tutorial and HTML tutorials as well.

How to compile Fortran under Unix:

Use a text editor to create a fortran 'source' code file.

Name the file *something*.f

Compile the source code into an executable program with the f77 command:

`f77 something.f` creates an executable called *a.out*

`f77 -o something something.f` creates an executable called *something*

`f77 -o something something1.f something2.f`
creates *something1.o*, *something2.o*, *something*

If using NCAR Graphics: substitute *ncargf77* for *f77*

Variable Names

Variable names should consist of 1-6 characters chosen from the letters `a-z` and the digits `0-9`. The first charactar MUST be a letter.

No difference between UPPER or lower case.

Longer names can be used, but may not be portable.

Types and Declarations

```
      integer           I,NLEVS,ICOUNT
      real              CP, THETA, R, PRESS
      double precision  DTHETA
      complex           IMAGIN
      logical           torf
```

character label, xaxis, yaxis

Integers and Floating Point variables

Integers - 32 bit (4 byte) Range of +/- 2*10^9
Floating point - Real (32 bit) or Double Precision (64 bit)

Parameter statement *(for constants)*

```
      real pi
      parameter (pi = 3.14159)

parameter (name=const, name=const, name=const)
```

File I/O

```
open (list-of-specifiers)

[UNIT=] u     (unit number, 9-99)
IOSTAT= ios   (integer var, i/o status, 0 if
               successful, nonzero if not)
ERR=    err   (label that program will jump to
               if there's an error)
FILE=   fname (string, name of file to open)
STATUS= sta   ('NEW', 'OLD', or 'SCRATCH')
ACCESS= acc   ('SEQUENTIAL' or 'DIRECT')
FORM=   form  ('FORMATTED' or 'UNFORMATTED')
RECL=   rl    (record length for direct access
               file)


close ([UNIT=]u)
```

A simple example

```
    open (10, file='prog1.dat')
    read (10,900) i,x,y
900 format(I4,2F6.1)
    close(10)
```

Format Statements

```
      write (*, label) list-of-variables
label format (format code)

      write (*, 900) i,x
900   format (I4, F8.3)
```

(integer of width 4, floating point with width 8, 3 digits to right of decimal point)

```
A - Text string
D - Double precision, exponent notation
E - Real numbers, exponent notation
F - Real numbers, fixed point notation
I - Integer
X - Horizontal skip (space)
/ - Vertical skip (new line)

nFw.d, nIw

n - number of them in a row
w - total width including decimal point
d - number of digits to right of decimal pt
```

Constants (6 types)

Integer

```
0      1      -100      +15    32767
```

Real

```
1.0   -0.25   2.0E6   3.333E-1   3.
```

Double Precision

```
2.0D-1    1D99
```

Complex

```
(2,-3)      (1., 9.9e-1)
```

Logical

```
.TRUE.      .FALSE.
```

Character

```
'ABC'     'Anything Goes'   'It''s a nice day'
```

Expressions

```
operand  operator  operand
x + y
x + 2 * y     [means x + (2 * y) ]
```

Operators, in order or precedence

```
**   (exponentiation)
*,/  (multiplication, division)
+,-  (addition, subtraction)

3/2=1     3./2.=1.5
```

Assignment

```
variable_name = expression

area = pi*r**2
```

Type Conversion

```
   real x
   x = x + 1
```

conversion functions:

int, real, dble, ichar, char

```
   x = x + real(1)
```

Logical Expressions (.TRUE. or .FALSE.)

```
.LT.     <
.LE.     <=
.GT.     >
.GE.     >=
.EQ.     =
.NE.     not =

logical a,b
a = .TRUE.
b = a .AND. 3 .LT. 5/2     (false)
```

Implicit types (I-> N integer, others real)

Simple I/O

```
read  (unit no, format no) list-of-vars
write (unit no, format no) list-of-vars
```

Simplify by reading from standard input (keyboard) and writing to standard output (screen/terminal) with no special formatting

```
read  (*,*) list-of-vars
write (*,*) list-of-vars
```

or

```
print *, list-of-vars
```

Subprograms

Functions - return one value

```
rh = relhum (t,r,p)
...
end

real function relhum(temp,mixrat,pres)
real temp,mixrat,pres
relhum=temp*pres+mixrat
return
end
```

Subroutines - return many values through arg list

```
real a(numpts)
call fillme (a,numpts)
...
end

subroutine fillme (a,n)
dimension a(n)
do 10 i=1,n
  a(i) = real(i)
10 continue
return
end
```

Call by Reference

The memory address for the variable is passed, not the value of the variable. So if an argument is changed in the subroutine, it has also been changed for the main program.

The `if` statement

```
if (logical expression) executable

if (x .LT. 0) x = -x


if (logical expression) then
  statements
endif


if (logical expression) then
  statements
elseif (logical expression) then
  statements
 :
 :
else
  statements
endif


if (x .GT. 0) then
  if (x .GE.y) then
    write(*,*) 'x > 0 and x >= y'
  else
    write(*,*) 'x > 0 but x < y'
  endif
else
  write(*,*) 'x < 0'
endif
```

Loops

Do loop

```
      integer i, n, sum
      sum = 0
      do 10 i = 1, n
        sum = sum + i
        write(*,*) 'i = ',i
        write(*,*) 'sum = ',sum
 10   continue
```

(10 is a statement label, col 2-5)

```
      do label var = start, end, increment
      statements
label continue
```

OR

```
      do var = start, end, increment
        statements
      enddo
```

While, Until loops - NOT standard Fortran

(while)

```
label if (logical expr) then
        statements
        goto label
      endif
```

(until)

```
label continue
      statements
      if (logical expr) goto label
```

Arrays

One-Dimensional Arrays

```
real a(20) [goes from a(1) to a(20)]
real b(0:19) [goes from b(0) to b(20) like C]
real c(-162:237) [wierd one]
```

can be of any basic data types

```
integer i(10)
logical aa(0:1)
```

each element can be thought of as a separate variable.

```
    integer i, sq(10)
    do 100 i=1,10
      sq(i) = i**2
100 continue
```

compiler will NOT check for out of bounds elements

Two-Dimensional Arrays

`real A(2,3)` two dim. array of 2*3=6 numbers.

Think of first index as ROW and second as COLUMN

```
(1,1)  (1,2)  (1,3)
(2,1)  (2,2)  (2,3)
(3,1)  (3,2)  (3,3)
```

Stored by COLUMN, so that in memory (1,2) follows (3,1)