

Fortran Reference Sheet

by Mark R. Petersen

References

- man page for your Fortran compiler. At UNIX prompt, type `man f77` or `man f90`
- on-line: search for Fortran reference. My favorite is <http://docs.sun.com/db/coll/34.4>
- *Fortran 90/95 Explained*, 2nd ed. by Michael Metcalf and John Reid, 1999
- *Fortran 90 Language Guide* by Wilhelm Gehrke, 1995 (this is technical but has everything)

Compiling

Command to compile is usually `f77`, `f90`, or `f95` (they are equivalent)

Say you have code in two files: `main.f90` and `subs.f`. At the UNIX prompt,

```
>f90 main.f90 subs.f           creates the executable a.out
>f90 main.f90 subs.f -o prog   creates the executable prog
>f90 main.f90 subs.f -c        creates object files main.o and subs.o
>f90 main.o subs.o -o prog     links the object files to create executable prog
```

Sample Code

- Fortran 77 uses a fixed column format. Code text begins in column 7. Comment lines have a character in column 1, col 2-5 are for line numbers, and a character in col 6 is for line continuation. Fortran 90 is free form, as shown.
- Code is not case sensitive, so variables `dia`, `DIA`, `Dia` are all the same.

Fortran 77	Fortran 90
<pre> program write_output c Fortran 77 example code c234567 c declarations: implicit none double precision pi,d integer j pi = atan(1.0)*4 j = 2 d = pi*j c write output to screen write(*,100) ' d = ',d 100 format (A,F10.5) end </pre>	<pre> program write_output ! Fortran 90 example code ! comments begin with ! ! declarations: implicit none real(8) :: d,pi ! comments can be integer :: j=2 ! anywhere with ! ! multiple commands on same line: pi = atan(1.0)*4; d = pi*j ! write output to screen write(*,'(A,F10.5)') ' d = ',d end program write_output </pre>

Data Types

- real - 4 byte floating point number, usually has 8 digits of accuracy.
synonyms: `real*4` in `f77`, `real(4)` in `f90`, `float` in `c`
- double precision - 8 byte floating point number, usually has 16 digits of accuracy.
synonyms: `real*8` in `f77`, `real(8)` in `f90`, `double` in `c`
- integer - 4 byte integer (positive or negative whole number)
synonyms: `integer(4)` in `f90`, `int` in `c`
- logical - single bit, value is either `.true.` or `.false.`
- character - string of characters, for example `file_name='velocity_02.dat'`

Declarations

All variables in a program or subroutine should be declared using `real`, `integer`, etc. Older code often uses implicit variables, where variables beginning with `i-n` are integers, and all others are real unless declared. Implicit variables are not recommended, as typos in variables do not cause compiler errors. The command `implicit none` forces all variables to be declared, and should always be used.

Mathematical Statements

Fortran uses standard order of operations:

parentheses, exponents, multiplication/division, addition/subtraction

- Exponent operator is `**`, not `^` as in other languages
- Integer division will round down. For example `5/4` will be 1. Avoid integer division by converting to reals first: `5.0/4.0` or `real(5)/real(4)`.

Input and Output

writing to the screen without formatting

```
write(*,*) ' index j = ',j,' and a,b are ',a,b
```

writing to the screen with formatting

```
write(*,100) ' index j = ',j,' and a,b are ',a,b
100 format (A,I4,A,2F10.5)
```

writing formatted text to an ASCII file

```
open(unit=5,file='output.txt')
write(5,100) ' index j = ',j,' and a,b are ',a,b
close(5)
100 format (A,I4,A,2F10.5)
```

reading text to an ASCII file

```
open(unit=5,file='input.txt',status='old')
read(5,*) a, b
close(5)
```

writing data to a binary file

```
open(unit=5,file='output.dat',form='UNFORMATTED')
write(5) a,b
close(5)
```

reading data from a binary file

```
open(unit=5,file='output.dat',form='UNFORMATTED',status='old')
read(5) a,b
close(5)
```

Formating

`2F7.3` produces `__3.146__3.146`

Diagram illustrating the components of the format string `2F7.3`:

- `2`: repeat factor
- `F`: variable type
- `7`: field width
- `.`: width after decimal point
- `3`: width after decimal point

variable types:

F	floating point
I	integer
E	exponential
A	characters
L	logical

other formats:

/	carraige return
X	space

Arrays

Variables of any type can be indexed arrays. The array's size must be included in the declaration

```
real a(100,100)
```

If Fortran 90, arrays can be allocated dynamically

```
real, allocatable :: a(:, :)
n=100
allocate(a(n,n))
... more code ...
deallocate(a)
```

Flow Control: DO for loops

Fortran 77	Fortran 90
<pre>do 50 j=1,10 b(j) = 3*a(j) 50 continue</pre>	<pre>do j=1,10 b(j) = 3*a(j) enddo</pre>

- Indenting should be used for readable code.
- Fortran 90 supports implicit array indexing, so the above statements could be replaced by `b=3*a` for the full array or `b(2:5)=3*a(2:5)` for part of the array.

Flow Control: IF for conditional branching

```
if (score.ge.90) then
  grade = 'A'
else if (score.ge.80) then
  grade = 'B'
else
  grade = 'C'
endif
```

Conditions:

f77 or f90	.lt.	.le.	.gt.	.ge.	.eq.	.ne.	.and.	.or.	.not.
f90 only	<	<=	>	>=	==	/=	.and.	.or.	.not.

Flow Control: EXIT from a DO loop

```
do j=1,1000
  sum = sum + a(j)
  if (sum >= 300) then
    exit ! this exits from do loop.
  endif
enddo
```

- Exit will move control to the command after the next `enddo`.
- Note multiple indenting for readable code.

Flow Control: GOTO (not recommended)

```
if (score.ge.90) then
  goto 30
else if (score.ge.80) then
  goto 40
endif
30 continue
c  more code ...
40 continue
```

- GOTO is considered poor programming. IF and EXIT statements should be used instead.

Subroutines and Functions

- These should be used for any repetitive tasks.
- Subroutines can have any number of input and output arguments
- Functions have any number of inputs, but only one output.

```

c      program factorial_chart
      create a chart of factorials
      implicit none
      integer j, f(10), n,
      integer factorial

      n=10
      do 5 j=1,n
         f(j) = factorial(j)
5      continue
      call chart(f,n)
      end

```

```

c      integer function factorial(k)
      calculate k factorial
      integer k, index

      factorial = 1
      do 3 index = 2,k
         factorial = factorial*index
3      continue
      return
      end

```

```

c      subroutine chart(data,m)
      print chart with two columns
      integer m, data(m), j

      write(*,*) '  n      n! '
      do 7 j=1,m
         write(*,110) j, data(j)
7      continue
110 format(i5,i10)
      return
      end

```

Make Files

The UNIX command `make` automates the process of compiling and linking code. This is particularly useful for large codes which are separated into many files. The `make` utility saves time by only compiling files which have changed since the last compilation.

```

# A simple make file
# Compile main.f90, subs1.f90, and subs2.f
# to generate the executable named Model

Model: main.o subs1.o subs2.o
<tab>f90 main.o subs1.o subs2.o -o Model

main.o: subs1.o subs2.o main.f90
<tab>f90 main.f90 -c

subs1.o: subs1.f90
<tab>f90 subs1.f90 -c

subs2.o: subs2.f
<tab>f90 subs2.f -c

```

- Each group of lines is
 component: files component depends on
 (tab character) commands to create this component
- Run the make file using `make make_file_name` at the UNIX prompt
- Type `man make` at the UNIX prompt for more information.
- Note that Fortran 90 and 77 code in separate files can easily be compiled together.