Basics of Fortran programming **Notes**

- Fortran is not case sensitive, but for clarity built-in commands are uppercase, and variable names in lowercase
- File names should end in `.f90` to make it clear that you're using modern Fortran 90, not Fortran 77

Programs

All Fortran codes must have one `PROGRAM` block where execution will begin:

```
PROGRAM myprogram
  IMPLICIT NONE
  <commands>
END PROGRAM myprogram
```

The `IMPLICIT NONE` is optional, but recommended. To compile a program:

```
$ gfortran mycode.f90 -o mycode
```

will produce an executable called `mycode`. To run the compiled program:

```
$ ./mycode
```

Comments

Comments start with '`!`'

```
! What the program does
PROGRAM myprogram
  IMPLICIT NONE
  ! What this step means
  <commands>
END PROGRAM myprogram
```

Variables

Before using a variable, you need to *declare* it by giving it a type e.g.

```
  REAL :: r
```

creates a variable `r` which is a REAL type, i.e. a number with decimal places like a FLOAT or DOUBLE in IDL.

Type	Explanation
INTEGER	Whole numbers $\mathbb{Z}$
REAL	Numbers with decimal point $\mathbb{R}$
COMPLEX	Complex numbers $\mathbb{C}$ with real and imaginary part
CHARACTER	Letters of the alphabet

Printing and input

The `PRINT` command is followed by a comma-separated list of values or expressions:

```
  PRINT *, "Hello world"
  PRINT *, "Result: ", r
```

To get input from the user, there is the `READ` command:

```
  READ *, r
```

Arrays

Array sizes can either be fixed

```
  REAL, DIMENSION(3) :: a
```

creates an array '`a`' with 3 elements. In Fortran these are numbered $1 \ldots n$ i.e. `a(1)` to `a(3)`. If an input to a function is an array, but could be any size, use e.g.

```
  REAL, DIMENSION(:), INTENT(IN) :: val
```

If you want to change the size of the array, it needs to be ALLOCATABLE:

```
  REAL, DIMENSION(:), ALLOCATABLE :: x
  INTEGER :: n
  n = 10 ! Could be input from user
  ALLOCATE(x(n)) ! X now has n elements
  ! Use x for some calculations
  DEALLOCATE(x) ! X now has no elements
```

You can also create arrays with more than one dimension in a very similar way

Subroutines

Subroutines take zero or more inputs, perform calculations on them, and produce (return) zero or more outputs.

```
! Sets intitial values of x and y
SUBROUTINE initial(x, y)
  REAL, INTENT(OUT) :: x, y
  READ *, x
  READ *, y
END SUBROUTINE initial
```

The INTENT for each parameter (x and y here) specifies whether it's an input (IN), an output (OUT), or both (INOUT). To use a subroutine, you need to CALL it:

```
PROGRAM myprogram
  IMPLICIT NONE
  REAL :: a, b
  CALL initial(a, b)
  PRINT *, "a = ", a, " b = ", b
END PROGRAM myprogram
```

The name of the variables in the brackets can be different inside the subroutine and where it's called; only the position matters.

Conditionals

The IF construct has the following syntax:

```
  IF (<condition>) THEN

  END IF
```

where condition can use either old or new style comparisons:

Old style	New style	Meaning
a .eq. b	a == b	'a' equal to 'b'?
a .ne. b	a /= b	'a' not equal 'b'?
a .lt. b	a < b	'a' less than 'b'?
a .gt. b	a > b	'a' greater than 'b'?

Loops

DO loops repeat a set of commands, each time using a different value of a variable (in this case i):

```
PROGRAM myprogram
  INTEGER :: i
  DO i=1,100
    PRINT *, i
  END DO
END PROGRAM myprogram
```

Another way is to keep looping until you EXIT

```
PROGRAM myprogram
  INTEGER :: i
  i = 1
  DO
    PRINT *, i
    IF (i .EQ. 100) THEN
      EXIT
    END IF
    i = i + 1
  END DO
END PROGRAM
```