

1. Input and output
2. Modules

1. Input and Output

Example 1.1 Illustrating formatted output.

```
PROGRAM TRIG_TABLE
! Compiles a table of SIN, COS, TAN against angle in DEGREES
  IMPLICIT NONE
  INTEGER DEG                ! angle in degrees
  REAL RAD                   ! angle in radians
  REAL PI                     ! mathematical pi
  CHARACTER (LEN=*), PARAMETER :: FMTHEAD = '( 1X, A3, 3( 2X, A7 ) )'
  CHARACTER (LEN=*), PARAMETER :: FMTDATA = '( 1X, I3, 3( 2X, F7.4 ) )'
                                ! formats for headings and data

  PI = 4.0 * ATAN( 1.0 )

  WRITE( *, FMTHEAD ) 'Deg', 'Sin', 'Cos', 'Tan'
  DO DEG = 0, 80, 10
    RAD = DEG * PI / 180.0
    WRITE( *, FMTDATA ) DEG, SIN( RAD ), COS( RAD ), TAN( RAD )
  END DO

END PROGRAM TRIG_TABLE
```

Alternatively:

```
WRITE( *, '( 1X, A3, 3( 2X, A7 ) )' ) 'Deg', 'Sin', 'Cos', 'Tan'
...
WRITE( *, '( 1X, I3, 3( 2X, F7.4 ) )' ) DEG, SIN( RAD ), COS( RAD ), TAN( RAD )
```

or:

```
100 FORMAT( 1X, A3, 3( 2X, A7 ) )
110 FORMAT( 1X, I3, 3( 2X, F7.4 ) )
...
WRITE( *, 100 ) 'Deg', 'Sin', 'Cos', 'Tan'
...
WRITE( *, 110 ) DEG, SIN( RAD ), COS( RAD ), TAN( RAD )
...
```

Writing to file:

```
OPEN( 33, FILE = 'trig.out' )      ! open file for output
WRITE( 33, FMTHEAD ) 'Deg', 'Sin', 'Cos', 'Tan'
...
WRITE( 33, FMTDATA ) DEG, SIN( RAD ), COS( RAD ), TAN( RAD )
...
CLOSE( 33 )                        ! close file (tidiness is a virtue!)
```

Example 1.2 Illustrating floating-point output.

```
PROGRAM EXPTABLE
! Program tabulates EXP(X)
  IMPLICIT NONE
  INTEGER :: NSTEP = 15                ! number of steps
  REAL :: XMIN = 0.0, XMAX = 3.0      ! interval limits
  REAL DELTAX                          ! step size
  REAL X                                ! current X value
  INTEGER I                             ! a counter

! Format specifiers
CHARACTER (LEN=*), PARAMETER :: FMT1 = '( 1X, A4 , 2X, A10 )'
CHARACTER (LEN=*), PARAMETER :: FMT2 = '( 1X, F4.2, 2X, 1PE10.3 )'

DELTAX = ( XMAX - XMIN ) / NSTEP      ! calculate step size
WRITE( *, FMT1 ) 'X', 'EXP'         ! write headers

DO I = 0, NSTEP
  X = XMIN + I * DELTAX              ! set X value
  WRITE( *, FMT2 ) X, EXP( X )      ! write data
END DO

END PROGRAM EXPTABLE
```

Example 1.3 Illustrating reading and writing files and data analysis.

The program adds scores from a round of golf.

It reads an input file `scorecard.txt` of the form

4							
6							
	Hole:	1	2	3	4	5	6
David		5	5	11	3	3	7
Judith		3	6	5	5	1	4
Miriam		4	3	9	2	4	6
Elizabeth		4	4	5	3	2	4

where:

- the first line is the number of players (here, 4)
- the second line is the number of holes (here, 6)
- the third line is the set of hole numbers
- the remaining lines are the players' names and their scores for each hole.

The number of players and the number of holes is arbitrary.

Output consists of a table of players and total scores and is written to file `total.txt`.

```
PROGRAM GOLF
  IMPLICIT NONE
  INTEGER NPLAYERS           ! number of files
  INTEGER NHOLES             ! number of holes
  INTEGER SCORE              ! total score
  INTEGER I                  ! loop counter
  INTEGER, ALLOCATABLE :: SHOTS(:) ! score for each hole
  CHARACTER (LEN=15) PLAYER ! player name
  CHARACTER (LEN=*), PARAMETER :: FMT = '(A, 3X, I3)' ! output format

  ! Open files
  OPEN( 21, FILE='scorecard.txt' )
  OPEN( 22, FILE='total.txt' )

  ! Read numbers of players and holes
  READ( 21, * ) NPLAYERS
  READ( 21, * ) NHOLES
  READ( 21, * )           ! skip a line - nothing needed

  ! Allocate memory to record the scores of one player
  ALLOCATE( SHOTS(NHOLES) )

  ! Loop round, reading player and scores, writing player and total
  DO I = 1, NPLAYERS
    READ( 21, * ) PLAYER, SHOTS
    SCORE = SUM( SHOTS )
    WRITE( 22, FMT ) PLAYER, SCORE
  END DO

  ! Close files and tidy up
  CLOSE( 21 )
  CLOSE( 22 )
  DEALLOCATE( SHOTS )

END PROGRAM GOLF
```

Example 1.4 Illustrating formatted READ, non-advancing i/o and the IOSTAT specifier.

The program converts a passage of text to upper case.

Input file: input.txt (any favourite piece of literature will do!)

Output file: output.txt

```
PROGRAM UPPERCASE
  IMPLICIT NONE
  INTEGER :: IO = 0
  INTEGER :: INCREMENT = ICHAR( 'A' ) - ICHAR( 'a' )
  CHARACTER CH

  OPEN( 10, FILE = 'input.txt' )           ! Open files
  OPEN( 20, FILE = 'output.txt' )

  DO WHILE ( IO /= -1 )
    READ( 10, '( A1 )', IOSTAT = IO, ADVANCE = 'NO' ) CH

    IF ( IO == 0 ) THEN
      IF ( CH >= 'a' .AND. CH <= 'z' ) THEN ! Change to upper case
        CH = CHAR( ICHAR( CH ) + INCREMENT )
      END IF
      WRITE ( 20, '( A1 )', ADVANCE = 'NO' ) CH ! Write to file
    ELSE IF ( IO == -2 ) THEN
      WRITE ( 20, * ) ! Force a line feed
    END IF

  END DO

  CLOSE( 10 ) ! Close files
  CLOSE( 20 )

END PROGRAM UPPERCASE
```

2. Modules

Example 2.1 Illustrating modules used to share related parameters and variables.

Compilation and linking commands:

```
ftn95 conversion.f95
ftn95 distance.f95
slink distance.obj conversion.obj
```

conversion.f95

```
MODULE CONVERSION
! Length conversion factors
IMPLICIT NONE
REAL, PARAMETER :: MILES_TO_METRES = 1609.0
REAL, PARAMETER :: YARDS_TO_METRES = 0.9144
REAL, PARAMETER :: FEET_TO_METRES = 0.3048
REAL, PARAMETER :: INCHES_TO_METRES = 0.0254

END MODULE CONVERSION
```

distance.f95

```
PROGRAM DISTANCE
  USE CONVERSION          ! make conversion factors available

  IMPLICIT NONE
  REAL METRES             ! distance in metres
  REAL AMOUNT             ! numerical quantity
  CHARACTER UNITS         ! units (i-inches, f-feet, y-yards, m-miles)

  PRINT *, 'Input amount and units (i-inches, f-feet, y-yard, m-mile)'
  READ *, AMOUNT, UNITS

  SELECT CASE (UNITS)
    CASE ( 'i' , 'I' ); METRES = AMOUNT * INCHES_TO_METRES
    CASE ( 'f' , 'F' ); METRES = AMOUNT * FEET_TO_METRES
    CASE ( 'y' , 'Y' ); METRES = AMOUNT * YARDS_TO_METRES
    CASE ( 'm' , 'M' ); METRES = AMOUNT * MILES_TO_METRES
  END SELECT

  PRINT *, 'Distance in metres = ', METRES

END PROGRAM DISTANCE
```

Example 2.2 Illustrating modules as CONTAINers of useful routines.

Compilation and linking commands:

```
ftn95 physics.f95
ftn95 gas.f95
slink gas.obj physics.obj
```

physics.f95

```
MODULE PHYSICS
! Physical constants and some useful subprograms
  IMPLICIT NONE
  REAL, PARAMETER :: SPEED_OF_LIGHT           = 3.00E+08      ! (m/s)
  REAL, PARAMETER :: PLANCKS_CONSTANT        = 6.63E-34      ! (J s)
  REAL, PARAMETER :: GRAVITATIONAL_CONSTANT  = 6.67E-11      ! (N m2/kg2)
  REAL, PARAMETER :: ELECTRON_MASS          = 9.11E-31      ! (kg)
  REAL, PARAMETER :: ELECTRON_CHARGE        = 1.60E-19      ! (C)
  REAL, PARAMETER :: STEFAN_BOLTZMANN_CONSTANT = 5.67E-08      ! (W/m2/K4)
  REAL, PARAMETER :: IDEAL_GAS_CONSTANT     = 8.31E+00      ! (J/K)
  REAL, PARAMETER :: AVOGADRO_NUMBER        = 6.02E+23      ! (/mol)

  CONTAINS

  REAL FUNCTION PRESSURE( n, T, V )
  ! Computes pressure by ideal gas law (pV=nRT)
    REAL n, T, V                                ! moles, temperature, volume

    PRESSURE = n * IDEAL_GAS_CONSTANT * T / V
  END FUNCTION PRESSURE

  REAL FUNCTION RADIATION( T, AREA, EMISSIVITY )
  ! Computes radiative heat flux
    REAL T                                ! thermodynamic temperature
    REAL AREA                              ! area of surface
    REAL EMISSIVITY

    RADIATION = EMISSIVITY * STEFAN_BOLTZMANN_CONSTANT * T ** 4 * AREA
  END FUNCTION RADIATION

END MODULE PHYSICS
```

gas.f95

```
PROGRAM IDEAL_GAS
! Program to test module <physics>
  USE PHYSICS                                ! access module

  IMPLICIT NONE
  REAL RMM                                  ! relative molecular mass
  REAL MASS                                 ! mass (kg)
  REAL TEMPERATURE                          ! temperature (K)
  REAL VOLUME                               ! volume (m3)
  REAL MOLES                                ! moles of gas

  PRINT *, 'Input relative molecular mass'
  READ *, RMM
  PRINT *, 'Input mass(kg), temperature(K), volume(m3)'
  READ *, MASS, TEMPERATURE, VOLUME

  MOLES = 1000.0 * MASS / RMM                ! calculate moles of gas

  PRINT *, 'Pressure = ', PRESSURE( MOLES, TEMPERATURE, VOLUME ), 'Pa'

END PROGRAM IDEAL_GAS
```