Chapter 1

Basics

1.1 Program Structure

```fortran
PROGRAM my_prog
!===============================================================================
USE module1
USE module2, ONLY: sub_a, sub_b, func_e

IMPLICIT NONE

CALL main()

!===============================================================================
CONTAINS
!===============================================================================

SUBROUTINE main()
IMPLICIT NONE
END SUBROUTINE main


SUBROUTINE other_sub()
IMPLICIT NONE
END SUBROUTINE other_sub


!===============================================================================
END PROGRAM my_prog
```

Listing 1.1: Basic program structure

1.2 Data Types

```fortran
PROGRAM data_types
!===============================================================================

USE iso_fortran_env

IMPLICIT NONE

! Call main subroutine
!################################################
CALL main()

!===============================================================================
CONTAINS
!===============================================================================

SUBROUTINE main()
!################################################
IMPLICIT NONE
!################################################
REAL(kind=REAL32) ::       float32        ! 32 bit signed    1/8/23    s/e/m
REAL(kind=REAL64) ::       float64
```

```fortran
INTEGER(kind=INT8) ::      int8          ! 8 bit signed
INTEGER(kind=INT16) ::     int16
INTEGER(kind=INT32) ::     int32
INTEGER(kind=INT64) ::     int64
!################################################

float32 = 3.4028235d38
float64 = -4.29d-18
int8 = 127
int16 = 32767
int32 = 2147483647
int64 = 2**63-1

print *, "ISO FORTRAN DATA TYPES"
print *, float32
print *, float64
print *, int8
print *, int16
print *, int32
print *, int64

!################################################
END SUBROUTINE main

!===============================================================================
END PROGRAM data_types
```

Listing 1.2: Using standard data types

1.3 FORTRAN - Column Major

```fortran
FUNCTION sum_arrays(a, b)
!#######################################
REAL(kind=REAL64) ::     a(1:10, 1:4)
REAL(kind=REAL64) ::     b(1:10, 1:4)
REAL(kind=REAL64) ::     sum_arrays
!#######################################
INTEGER(kind=INT32) ::   i, j
!#######################################

sum_arrays = 0.0D0
DO j = 1, 4            ! Loop over columns (outer)
  DO i = 1, 10         ! Loop over rows (inner)
    sum_arrays = sum_arrays + a(i, j) + b(i, j)
  END DO
END DO

! Better solution
sum_arrays = 0.0D0
sum_arrays = SUM(a(:,:)) + SUM(b(:,:))

END FUNCTION sum_arrays
```

Listing 1.3: Column major language

Chapter 2

Output

2.1 Print

```
SUBROUTINE terminalprint()
!########################################################################
IMPLICIT NONE
!########################################################################
PRINT *, "Free text", 100, 1.4321D2
END SUBROUTINE terminalprint
```

Listing 2.1: Print statement

2.2 Write

```
SUBROUTINE terminalwrite()
!########################################################################
IMPLICIT NONE
!########################################################################
WRITE(*, *) "Free text"
WRITE(*, "(A20)") "20 character string."
WRITE(*, "(A24)") "20 character string."
WRITE(*, "(I4)") 1000
WRITE(*, "(I10)") 1000                             ! Integer
WRITE(*, "(F20.8)") 123456789.123456789e1          ! Fixed Point
WRITE(*, "(E20.12)") 123456789.123456789e1         ! Floating Point - Exponential form
WRITE(*, "(ES20.12)") 123456789.123456789e1        ! Floating Point - Scientific form
WRITE(*, "(EN20.12)") 123456789.123456789e1        ! Floating Point - Engineering form
WRITE(*, "(D20.12)") 123456789.123456789d1         ! Double
WRITE(*, "(E20.12E3)") 123456789.123456789e1       ! Floating Point - Exponential form
WRITE(*, "(ES20.12E3)") 123456789.123456789e1      ! Floating Point - Scientific form
WRITE(*, "(EN20.12E3)") 123456789.123456789e1      ! Floating Point - Engineering form
END SUBROUTINE terminalwrite
```

Listing 2.2: Write statement

Chapter 3

Arrays

3.1 Zero

```
SUBROUTINE zero(x)
!#################################################
IMPLICIT NONE
!#################################################
REAL(kind=REAL64), INTENT(INOUT) ::     x(:, :)
!#################################################
x(:, :) = 0.0d0
!#################################################
END SUBROUTINE zero
```

Listing 3.1: Zero (or any number)

3.2 Linspace

```
! Linspace

SUBROUTINE linspace(xmin, xmax, x)
!#################################################
IMPLICIT NONE
!#################################################
REAL(kind=REAL64), INTENT(IN) ::        xmin
REAL(kind=REAL64), INTENT(IN) ::        xmax
REAL(kind=REAL64), INTENT(INOUT) ::     x(:)
!#################################################
INTEGER(kind=INT32) ::                  n
REAL(kind=REAL64) ::                    m
!#################################################
m = (xmax - xmin) / (SIZE(x,1) - 1)
DO n = 1, SIZE(x,1)
  x(n) = xmin + (n - 1) * m
END DO
!#################################################
END SUBROUTINE linspace
```

Listing 3.2: Equally spaced linear space of points, as exists in Numpy

Chapter 4

Files

4.1 Write

```fortran
! Write file
SUBROUTINE write ()
!################################################
INTEGER(kind=INT16) ::            status, ios
INTEGER(kind=INT16) ::            n, m
INTEGER(kind=INT16) ::            lines
REAL(kind=REAL64) ::              arr(1:10,1:3)
!################################################
! Populate array with random floats
CALL RANDOM_NUMBER(arr)
arr = 1.0D10 * (0.5D0 - arr)
! Write to file
OPEN(unit=99, file='out.txt')
DO n = 1, 10
  WRITE(99, "(ES16.7, ES16.7, ES16.7)") arr(n, 1), arr(n, 2), arr(n, 3)
END DO
CLOSE(99)
END SUBROUTINE write
```

Listing 4.1: Writing to file

4.2 Read to Array

```fortran
! Read file - into array, 2 columns
SUBROUTINE readfile(filepath, arr)
!########################################################################
IMPLICIT NONE
!########################################################################
CHARACTER(LEN=*), INTENT(IN) ::                      filepath
REAL(kind=REAL64), ALLOCATABLE, INTENT(INOUT) ::     arr(:,:)
!########################################################################
INTEGER(kind=INT32), PARAMETER ::                    bsize = 1000000
CHARACTER(LEN=255) ::                                line
CHARACTER(LEN=255) ::                                buffer(1:bsize)
INTEGER(kind=INT32) ::                               status, ios, fh
INTEGER(kind=INT32) ::                               n
INTEGER(kind=INT32) ::                               lines
LOGICAL ::                                           exists
!########################################################################
! Check file exists
INQUIRE (file=filepath, exist=exists)
IF(.NOT. exists)THEN
  STOP "File " // TRIM(filepath) // " does not exist"
END IF
! Count Lines & Read To Buffer
lines = 0
OPEN(newunit=fh, file=filepath, action='read', iostat=status)
DO n = 1, bsize
  READ(fh,"(A255)",IOSTAT=ios) line
  IF(TRIM(line) .NE. "")THEN
    lines = lines + 1
    buffer(lines) = line
  END IF
  If(ios /= 0)Then
    EXIT
  End If
END DO
CLOSE(fh)
! Allocate array
IF(ALLOCATED(arr)) DEALLOCATE(arr)
ALLOCATE(arr(1:lines, 1:2))
! Read into array
DO n = 1, lines
  READ(buffer(n),"(10F8.2,10F8.2)",IOSTAT=ios) arr(n, 1), arr(n, 2)
END DO
END SUBROUTINE readfile
```

Listing 4.2: Read file into a 2D, 2 column, array

4.3 Read Namelist

```
! computer.nml
!
! &COMPUTER
! os='Ubuntu'
! ram=64
! proc%make='AMD'
! proc%model='Ryzen 9'
! proc%freq=3.8
! proc%cores=12
! proc%cache(1)=0.75
! proc%cache(2)=6
! proc%cache(3)=64
! drive(1)=256
! drive(2)=5000
! drive(3)=5000
! drive(4)=0
! drive(5)=0
! /

SUBROUTINE readnamelist(filepath)
!########################################################################
IMPLICIT NONE
!########################################################################
CHARACTER(LEN=*), INTENT(IN) ::                      filepath
!########################################################################
TYPE :: t_processor
  CHARACTER(len=16) ::                               make
  CHARACTER(len=16) ::                               model
  INTEGER(kind=INT32) ::                             cores
  REAL(kind=REAL32) ::                               freq
  REAL(kind=REAL32) ::                               cache(1:3)
END TYPE t_processor
CHARACTER(len=16) ::                                 os
INTEGER(kind=INT32) ::                               ram
TYPE(t_processor) ::                                 processor
INTEGER(kind=INT32) ::                               drive(1:5)
LOGICAL ::                                           exists
INTEGER(kind=INT32) ::                               status
INTEGER(kind=INT32) ::                               fh
!########################################################################
NAMELIST /COMPUTER/ os, ram, processor, drive     ! Define namelist
!########################################################################
! Check file exists
INQUIRE (file=filepath, exist=exists)
IF(.NOT. exists)THEN
  STOP "File " // TRIM(filepath) // " does not exist"
END IF
! Open and read file
OPEN (action='read', file=filepath, iostat=status, newunit=fh)
read (nml=COMPUTER, iostat=status, unit=fh)
CLOSE(fh)
! Check read successful
IF(status .NE. 0)THEN
  STOP "File " // TRIM(filepath) // " contains an incorrect namelist format"
```

```
END IF
! Output
WRITE (*, "(A20)") ADJUSTR(os)
WRITE (*, "(I20)") ram
WRITE (*, "(A20)") ADJUSTR(processor%make)
WRITE (*, "(A20)") ADJUSTR(processor%model)
WRITE (*, "(I20)") processor%cores
WRITE (*, "(I20)") drive(1)
WRITE (*, "(I20)") drive(2)
WRITE (*, "(I20)") drive(3)
WRITE (*, "(I20)") drive(4)
WRITE (*, "(I20)") drive(5)
END SUBROUTINE readnamelist
```

Listing 4.3: Read namelist

Chapter 5

Interpolation

5.1 Lagrange

```fortran
!  Lagrange interpolation
!
FUNCTION interpn(xi, x, y)
!################################################
IMPLICIT NONE
!################################################
REAL(kind=REAL64), INTENT(IN) ::    xi
REAL(kind=REAL64), INTENT(IN) ::    x(:)
REAL(kind=REAL64), INTENT(IN) ::    y(:)
REAL(kind=REAL64) ::                interpn
!################################################
REAL(kind=REAL64) ::                li
INTEGER(kind=INT32) ::              xsize
INTEGER(kind=INT32) ::              i, j
!################################################
IF(SIZE(x) .NE. SIZE(y))THEN
  STOP "Error interpn() unequal array sizes"
END IF
! Set values
interpn = 0.0D0
xsize = SIZE(x)
! Loop
DO i = 1, xsize
  li = 1.0D0
  DO j = 1, xsize
    IF(i .NE. j) THEN
      li = li * (xi - x(j)) / (x(i) - x(j))
    END IF
  END DO
  interpn = interpn + li * y(i)
END DO
END FUNCTION interpn
```

Listing 5.1: Interpolate small data set

5.2 Lagrange - Larger Data Set

```fortran
SUBROUTINE interpfill(x_in, y_in, k, x_out, y_out)
!################################################
IMPLICIT NONE
!################################################
REAL(kind=REAL64), INTENT(IN) ::       x_in(:)
REAL(kind=REAL64), INTENT(IN) ::       y_in(:)
INTEGER(kind=INT32), INTENT(IN) ::     k
REAL(kind=REAL64), INTENT(INOUT) ::    x_out(:)
REAL(kind=REAL64), INTENT(INOUT) ::    y_out(:)
!################################################
INTEGER(kind=INT32) ::                 s_in, s_out
INTEGER(kind=INT32) ::                 n, nn, a, b, i, j
REAL(kind=REAL64) ::                   m
REAL(kind=REAL64) ::                   li
```

```fortran
!################################################
IF(SIZE(x_in) .NE. SIZE(y_in) .OR. SIZE(x_out) .NE. SIZE(y_out))THEN
  STOP "Error interpolate() unequal array sizes"
END IF
! Array sizes
s_in = SIZE(x_in)
s_out = SIZE(x_out)
! Fill X
m = (x_in(s_in) - x_in(1)) / (s_out - 1)
DO n = 1, s_out
  x_out(n) = x_in(1) + (n - 1) * m
END DO
! Fill Y
y_out(:) = 0.0d0
nn = 1
DO n = 1, s_out
  DO WHILE(nn .LT. s_in .AND. .NOT. (x_out(n) .GE. x_in(nn) .AND. x_out(n) .LE. x_in(nn+1)))
    nn = nn + 1
  END DO
  ! Choose subset of points for interpolation
  a = MIN(nn, s_in - k + 1)
  b = a + k - 1
  DO i = a, b
    li = 1.0D0
    DO j = a, b
      IF(i .NE. j) THEN
        li = li * (x_out(n) - x_in(j)) / (x_in(i) - x_in(j))
      END IF
    END DO
    y_out(n) = y_out(n) + li * y_in(i)
  END DO
END DO
END SUBROUTINE interpfill
```

Listing 5.2: Interpolate with k points over a larger data set

5.3 Lagrange - Gradient

```fortran
!
!  Lagrange interpolation
!
FUNCTION interpngrad(xi, x, y)
!################################################
IMPLICIT NONE
!################################################
REAL(kind=REAL64), INTENT(IN) ::     xi
REAL(kind=REAL64), INTENT(IN) ::     x(:)
REAL(kind=REAL64), INTENT(IN) ::     y(:)
REAL(kind=REAL64) ::                 interpngrad
!################################################
REAL(kind=REAL64) ::                 fx, gx, psum
INTEGER(kind=INT32) ::               xsize
INTEGER(kind=INT32) ::               i, j, k
!################################################
IF(SIZE(x) .NE. SIZE(y))THEN
  STOP "Error interpngrad() unequal array sizes"
END IF
! Set values
xsize = SIZE(x,1)
interpngrad = 0.0d0
DO i=1,SIZE(x,1)
  fx = 1.0d0
  gx = 0.0d0
  DO j=1,SIZE(x,1)
    IF(i .NE. j) THEN
      fx = fx / (x(i) - x(j))
      psum = 1.0d0
      DO k=1,SIZE(x,1)
        IF((i .NE. k) .AND. (j .NE. k))THEN
          psum = psum * (xi - x(k))
        END IF
      END DO
      gx = gx + psum
    END IF
  END DO
  interpngrad = interpngrad + fx * gx * y(i)
END DO
END FUNCTION interpngrad
```

Listing 5.3: Gradient at xi using a small data set

Chapter 6

Differentiation

6.1 Zero

```fortran
FUNCTION grad(f_in, x)
!########################################################################
IMPLICIT NONE
!########################################################################
REAL(kind=REAL64), EXTERNAL ::              f_in
REAL(kind=REAL64), INTENT(IN) ::            x
REAL(kind=REAL64) ::                        grad
!########################################################################
REAL(kind=REAL64) ::                        h
!########################################################################
h = 1.0D-5
grad = (f_in(x+h) - f_in(x-h)) / (2 * h)
END FUNCTION grad
```

Listing 6.1: Calculate 1st derivative

6.2 Linspace

```fortran
FUNCTION grad2(f_in, x)
!########################################################################
IMPLICIT NONE
!########################################################################
REAL(kind=REAL64), EXTERNAL ::              f_in
REAL(kind=REAL64), INTENT(IN) ::            x
REAL(kind=REAL64) ::                        grad2
!########################################################################
REAL(kind=REAL64) ::                        h
!########################################################################
h = 1.0D-5
grad2 = (f_in(x+h) - 2.0d0 * f_in(x) + f_in(x-h)) / h**2
END FUNCTION grad2
```

Listing 6.2: Calculate 2nd derivative

Chapter 7

Integration

7.1 Simpson Integration Part 1

```
!  Simpson integration
!  Must have an even number of intervals
!  (odd number of data points)
!
FUNCTION integrate(x, y)
!################################################
IMPLICIT NONE
!################################################
REAL(kind=REAL64), INTENT(IN) ::    x(:)
REAL(kind=REAL64), INTENT(IN) ::    y(:)
REAL(kind=REAL64) ::                integrate
!################################################
REAL(kind=REAL64) ::                h, odd, even
INTEGER(kind=INT32) ::              xsize
INTEGER(kind=INT32) ::              n
!################################################
IF(SIZE(x) .NE. SIZE(y))THEN
  STOP "Error integrate() unequal array sizes"
END IF
odd = 0.0d0
even = 0.0d0
xsize = SIZE(x, 1)
h = (x(xsize) - x(1)) / (xsize - 1)
! Odd and even terms
n = 1
DO WHILE(n < xsize)
  n = n + 1
  even = even + y(n)
  n = n + 1
  odd = odd + y(n)
END DO
! Finish
integrate = (h / 3.0d0) * (y(1) + y(xsize) + 4.0d0 * even + 2.0d0 * odd)
END FUNCTION integrate
```

Listing 7.1: Using Simpson Integration

7.2 Simpson Integration Part 2

```
!  Simpson integration
!  Interpolates to odd number of points
!  Requires interpfill subroutine
!
FUNCTION integrate(x, y)
!################################################
IMPLICIT NONE
!################################################
REAL(kind=REAL64), INTENT(IN) ::    x(:)
REAL(kind=REAL64), INTENT(IN) ::    y(:)
REAL(kind=REAL64) ::                integrate
!################################################
```

```
REAL(kind=REAL64) ::                h, odd, even
INTEGER(kind=INT32) ::              xsize
INTEGER(kind=INT32) ::              n
REAL(kind=REAL64), ALLOCATABLE ::   xt(:)
REAL(kind=REAL64), ALLOCATABLE ::   yt(:)
!################################################
odd = 0.0d0
even = 0.0d0
IF(SIZE(x) .NE. SIZE(y))THEN
  STOP "Error integrate() unequal array sizes"
END IF
IF(MOD(SIZE(x, 1), 2) .EQ. 0)THEN
  ALLOCATE(xt(1:SIZE(x, 1)+1))
  ALLOCATE(yt(1:SIZE(y, 1)+1))
  CALL interpfill(x, y, 3, xt(:), yt(:))
  xsize = SIZE(xt, 1)
  h = (xt(xsize) - xt(1)) / (xsize - 1)
  ! Odd and even terms
  n = 1
  DO WHILE(n < xsize)
    n = n + 1
    even = even + yt(n)
    n = n + 1
    odd = odd + yt(n)
  END DO
ELSE
  xsize = SIZE(x, 1)
  h = (x(xsize) - x(1)) / (xsize - 1)
  ! Odd and even terms
  n = 1
  DO WHILE(n < xsize)
    n = n + 1
    even = even + y(n)
    n = n + 1
    odd = odd + y(n)
  END DO
END IF
! Finish
integrate = (h / 3.0d0) * (y(1) + y(xsize) + 4.0d0 * even + 2.0d0 * odd)
END FUNCTION integrate
```

Listing 7.2: Using Simpson Integration

Chapter 8

OpenMP

8.1 Summing in parallel

```fortran
!
! gfortran -fopenmp -o sum.x sum.f90 && ./sum.x
!

! Possibily not efficient, over head may outweigh benefit

SUBROUTINE parallelsum()
!#########################################################################
IMPLICIT NONE
!#########################################################################
REAL(kind=REAL64) ::               myarray(1:100000)
REAL(kind=REAL64) ::               s_thread, s_total
INTEGER(kind=INT64) ::             n, tid, tcount
INTEGER(kind=INT64) ::             OMP_GET_NUM_THREADS, OMP_GET_THREAD_NUM
!#########################################################################
s_total = 0.0d0
myarray(1:100000) = 1.0D0
!$OMP PARALLEL &
!$OMP PRIVATE(n, s_thread, tid) &
!$OMP SHARED(myarray, s_total)
s_thread = 0.0d0
!$OMP DO
DO n = 1, 100000
  s_thread = s_thread + myarray(n)
END DO
!$OMP END DO
!$OMP CRITICAL
tcount = OMP_GET_NUM_THREADS()
tid = OMP_GET_THREAD_NUM()
s_total = s_total + s_thread
WRITE(*,*) tid , "/" , tcount, "   ", s_thread
!$OMP END CRITICAL
!$OMP END PARALLEL
WRITE(*,*) s_total
END SUBROUTINE parallelsum
```

Listing 8.1: Using OpenMP to sum over several threads