1. Character Handling

Example 1.1 Basic character-handling operations.

```fortran
PROGRAM CHARACTER_FUNCTIONS
! Program illustrating strings and character functions
IMPLICIT NONE
CHARACTER (LEN=72) SCHOOL

SCHOOL = 'School of Mechanical, Aerospace and Civil Engineering'

PRINT *, '1: ', SCHOOL
PRINT *, '2: ', SCHOOL(11:20)
PRINT *, '3: ', SCHOOL(37:)
PRINT *, '4: ', LEN(SCHOOL), LEN_TRIM(SCHOOL), LEN(TRIM(SCHOOL))
PRINT *, '5: ', INDEX(SCHOOL, 'Civil')
PRINT *, '6: ', SCAN(SCHOOL, 'PQR'), SCAN(SCHOOL, 'pqr')
PRINT *, '7: ', SCAN(SCHOOL, 'e'), SCAN(SCHOOL, 'e', .TRUE.)
PRINT *, '8: ', VERIFY(SCHOOL, 'Superb scholars')
END PROGRAM CHARACTER_FUNCTIONS
```

Example 1.2 The Fortran character set.

```fortran
PROGRAM CHARACTER_SET
! Program prints (most of) the Fortran character set, 10 to a line
IMPLICIT NONE
INTEGER N
CHARACTER (LEN=*)
PARAMETER :: FMT = '( 10( 2X, I3, 1X, A1 ) )'

WRITE( *, FMT ) ( N, CHAR(N), N = 20, 127 )

END PROGRAM CHARACTER_SET
```
Example 1.3 Date, time and character variables.

```
PROGRAM CLOCK
! Program asking the computer for date and time
IMPLICIT NONE
CHARACTER (LEN=8) DATE            ! date in format ccyymmdd
CHARACTER (LEN=10) TIME           ! time in format hhmmss.sss
CHARACTER (LEN=5) ZONE            ! time zone (rel to UTC) as Shhmm
INTEGER VALUES(8)                 ! year, month, day, mins from UTC,
!       hours, min, sec, msec
CHARACTER (LEN=8) TIMESTRING      ! time in digital form
CHARACTER (LEN=20) DATESTRING     ! date in various forms
CHARACTER (LEN=9) MONTH(12)       ! months of the year
INTEGER FIRST, LAST               ! system clock counts
INTEGER COUNTS_PER_SECOND         ! clock counts per second
REAL SECONDS                      ! time taken
DATA MONTH / 'January', 'February', 'March', 'April', 'May', 'June', &
    'July', 'August', 'September', 'October', 'November', 'December' /

! Ask for date and time (DATE and TIME as text, VALUES as integers)
CALL DATE_AND_TIME( DATE, TIME, ZONE, VALUES )

! Put the time in digital form
TIMESTRING = TIME( 1:2 ) // ':' // TIME( 3:4 ) // ':' // TIME( 5:6 )

! Date in abbreviated form from character string DATE
DATESTRING = DATE( 7:8 ) // '-' // DATE( 5:6 ) // '-' // DATE( 1:4 )
WRITE( '*', '' ) 'It is ', TIMESTRING, ' on ', DATESTRING

! Date from integer array VALUES
WRITE( DATESTRING, '(I2, 1X, A, 1X, I4)') VALUES(3), TRIM(MONTH(VALUES(2))), VALUES(1)
WRITE( '*', '' ) 'It is ', TIMESTRING, ' on ', DATESTRING

! Illustrates timing
CALL SYSTEM_CLOCK( FIRST, COUNTS_PER_SECOND )
WRITE( '*', '' ) 'Hit the enter button'
READ( *, * )
CALL SYSTEM_CLOCK( LAST, COUNTS_PER_SECOND )
SECONDS = REAL( LAST - FIRST ) / COUNTS_PER_SECOND
WRITE( '*', '' ) 'It took ', SECONDS, ' seconds to run this stupid program'
END PROGRAM CLOCK
```
2. Subprograms (Subroutines and Functions)

Example 2.1 Use of functions to allow general methods to be applied to particular problems.

(a) Trapezium rule for integration:
\[
\int_{a}^{b} f(x) \, dx = \frac{\Delta x}{2} \left[ f(a) + f(b) + 2 \sum_{i=1}^{N-1} f(x_i) \right]
\]
where, with \(N\) intervals,
\[
\Delta x = \frac{b-a}{N}, \quad x_i = a + i\Delta x
\]

```fortran
PROGRAM TRAPEZIUM_RULE
 IMPLICIT NONE
 REAL, EXTERNAL :: F                     ! function to be integrated
 INTEGER N                               ! number of intervals
 REAL A, B                               ! limits of integration
 REAL INTEGRAL                           ! value of integral
 REAL DX                                 ! interval
 REAL X                                  ! an ordinate
 INTEGER I                               ! a counter

 PRINT *, 'Input A, B, N'
 READ *, A, B, N
 DX = (B - A) / N                        ! calculate interval
 INTEGRAL = F(A) + F(B)                  ! contribution from ends
 DO I = 1, N - 1
    X = A + I * DX                       ! calculate intermediate point
    INTEGRAL = INTEGRAL + 2.0 * F(X)     ! add contribution to sum
 END DO
 INTEGRAL = INTEGRAL * DX / 2.0          ! convert sum to integral
 PRINT *, 'Integral = ', INTEGRAL
 END PROGRAM TRAPEZIUM_RULE

!================================================== =====================
REAL FUNCTION F( X )                       ! Function to be integrated
 IMPLICIT NONE
 REAL X
 F = X ** 2
END FUNCTION F
```

To change the function to be integrated just change the \(F = \ldots\) line.
(b) Newton-Raphson iteration to solve \( f(x) = 0 \):

\[
x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}
\]

Program the method to work for arbitrary \( f \); use functions to return particular \( f \) and \( f' \).

```fortran
PROGRAM NEWTON_RAPHSON
  ! Program finds a root of the equation f(x) = 0 by the Newton-Raphson method
  IMPLICIT NONE
  REAL, EXTERNAL :: F, DFDX           ! function and its derivative
  REAL, PARAMETER :: TOLERANCE = 1.0E-6 ! tolerance for near-zero
  INTEGER, PARAMETER :: ITERMX = 200   ! maximum number of iterations
  REAL X                               ! current x
  REAL FVALUE                          ! current value of function f
  INTEGER :: ITER = 0                  ! current iteration number

  PRINT *, 'Input initial X'
  READ *, X                            ! input first guess
  FVALUE = F( X )                       ! value of function
  PRINT *, 'X, F(X) =', X, FVALUE

  ! Loop until root found or maximum iterations reached
  DO WHILE ( ABS( FVALUE ) > TOLERANCE .AND. ITER <= ITERMX )
    X = X - FVALUE / DFDX( X )         ! update x by Newton-Raphson formula
    FVALUE = F( X )                    ! update value of function
    ITER = ITER + 1                    ! update iteration number
    PRINT *, 'X, F(X) =', X, FVALUE    ! output current values
  END DO

  ! Output answer (or warn if not converged)
  IF ( ABS( FVALUE ) > TOLERANCE ) THEN
    PRINT *, 'Not converged'
  ELSE
    PRINT *, 'Answer =', X
  END IF

END PROGRAM NEWTON_RAPHSON

!===========================================
REAL FUNCTION F( X )
  ! This function should return the value of the function at X
  IMPLICIT NONE
  REAL X
  F = 16.0 * X * X - 4.0
END FUNCTION F

!===========================================
REAL FUNCTION DFDX( X )
  ! This function should return the derivative of the function at X
  IMPLICIT NONE
  REAL X
  DFDX = 32.0 * X
END FUNCTION DFDX
```

To change the equation to be solved just change the \( F = \ldots \) and \( DFDX = \ldots \) lines.
Example 2.2 Use of subroutines.

```fortran
PROGRAM EXAMPLE
! Program to swap two numbers
IMPLICIT NONE
EXTERNAL SWAP ! (optionally) declare routine to be used
INTEGER I, J

PRINT *, 'Input integers I and J'
READ *, I, J

CALL SWAP( I, J )

PRINT *, 'Swapped numbers are ', I, J

END PROGRAM EXAMPLE

!=======================================
SUBROUTINE SWAP( M, N )
IMPLICIT NONE
INTEGER M, N ! numbers to be swapped
INTEGER TEMP ! temporary storage

TEMP = M ! store number before changing it
M = N
N = TEMP

END SUBROUTINE SWAP
```
Example 2.3 Specifying INTENT for subprogram arguments.

```
PROGRAM COORDINATES
! Program to convert from Cartesian to polar coordinates
IMPLICIT NONE
EXTERNAL POLARS
REAL X, Y
REAL R, THETA

PRINT *, 'Input coordinates X and Y'
READ *, X, Y

CALL POLARS( X, Y, R, THETA )
PRINT *, 'R, THETA =', R, THETA

END PROGRAM COORDINATES

SUBROUTINE POLARS( X, Y, R, THETA )
! Subroutine transforming input (X, Y) to output (R, THETA)
IMPLICIT NONE
REAL, INTENT(IN) :: X, Y ! cartesian coordinates (input)
REAL, INTENT(OUT) :: R, THETA ! polar coordinates (output)
REAL, PARAMETER :: PI = 3.141593 ! the constant pi

R = SQRT( X ** 2 + Y ** 2 ) ! radius
THETA = ATAN2( Y, X ) ! inverse tangent between -pi and pi
IF ( THETA < 0.0 ) THETA = THETA + 2.0 * PI ! angle between 0 and 2 pi
THETA = THETA * 180.0 / PI ! convert to degrees

END SUBROUTINE POLARS
```
Example 2.4 Subroutines with array arguments.

Mean: \[ \bar{X} = \frac{\sum X}{N} \], sample variance: \[ \sigma^2 = \frac{\sum X^2}{N} - \bar{X}^2 \], standard deviation: \( \sigma \)

(Note: for an unbiased estimate of the population variance multiply \( \sigma^2 \) by \( \frac{N}{N-1} \).)

PROGRAM EXAMPLE
!

! Program computes mean, variance and standard deviation

IMPLICIT NONE
EXTERNAL STATS
INTEGER NVAL
REAL, ALLOCATABLE :: X(:)
REAL MEAN, VARIANCE, STANDARD_DEVIATION
INTEGER N

! Open data file
OPEN( 10, FILE = 'stats.dat' )

! Read the number of points and set aside enough memory
READ( 10, * ) NVAL
ALLOCATE( X(NVAL) )

! Read data values
READ( 10, * ) ( X(N), N = 1, NVAL )
CLOSE( 10 )

! Compute statistics
CALL STATS( NVAL, X, MEAN, VARIANCE, STANDARD_DEVIATION )

! Output results
PRINT *, 'Mean = ', MEAN
PRINT *, 'Variance = ', VARIANCE
PRINT *, 'Standard deviation = ', STANDARD_DEVIATION

! Recover computer memory
DEALLOCATE( X )

END PROGRAM EXAMPLE

=========================================

SUBROUTINE STATS( N, X, M, VAR, SD )
!

! This works out the sample mean, variance and standard deviation

IMPLICIT NONE
INTEGER, INTENT(IN) :: N
REAL, INTENT(IN) :: X(N)
REAL, INTENT(OUT) :: M, VAR, SD

! Calculate statistics using array operation SUM
M = SUM( X ) / N
VAR = SUM( X ** 2 ) / N - M ** 2
SD = SQRT( VAR )

END SUBROUTINE STATS