Computer Science 1510

Lecture 11

Lecture Outline

- Subroutines

Subroutines

- Subroutines are the second type of subprograms in Fortran (in addition to functions).

- Some commonalities between subroutines and functions include:
  - They are program units designed to perform tasks under the control of another program unit (main program or subprogram).
  - They have a similar basic form (heading, declarations, execution part, end statement).
  - Both can be internal, module, or external.
  - The same scope rules apply.

- Differences include:
  - Functions return a single value.
    Subroutines can return many values or no values.
  - A function returns a value via the function name; subroutines return values via arguments.
  - A function is referenced by using the function name in an expression. A subroutine is referenced via a `CALL` statement.

Subroutines

- Syntax:

```
Subroutine heading
   variable-declarations
   body-of-subroutine
   RETURN
END SUBROUTINE subroutine-name
```

- Where `Subroutine heading` is one of:

```
SUBROUTINE subroutine-name(formal-argument-list)
                    or
RECURSIVE SUBROUTINE subroutine-name(formal-argument-list)
```

- A subroutine is called using a `CALL` statement:

```
CALL subroutine-name(actual-argument-list)
```

- The number and type of actual arguments must match the number and type of formal arguments (unless optional arguments are used).

- When a `CALL` statement is reached, control is passed to the subroutine, and execution follows through the subroutine up to the `RETURN` or `END SUBROUTINE` statement, at which point execution jumps to the statement following the `CALL` statement in the calling program or subprogram.

- If there are no arguments to be passed into a `SUBROUTINE` the parentheses can be omitted both in the subroutine definition and in the call statement.

- Example:

```
!-PrintDegrees-------------------------------------------------------
! Subroutine to display a measurement of Degrees, Minutes, Seconds
! as the equivalent degree measure.
!
! Accepts:  Degrees, Minutes, Seconds
! Output:   Degrees, Minutes, and Seconds and equivalent degree measure
!--------------------------------------------------------------------
SUBROUTINE PrintDegrees(Degrees, Minutes, Seconds)

   INTEGER, INTENT(IN) :: Degrees, Minutes, Seconds

   WRITE(*,10) Degrees, Minutes, Seconds, &
               REAL(Degrees) + REAL(Minutes)/60.0 + REAL(Seconds)/3600.0
10 FORMAT(I3,' degrees ',I3,' minutes ',I3,' seconds is equivalent to ',&
          F7.3,' degrees.')
END SUBROUTINE PrintDegrees
```

Example 1: Subroutine

```
PROGRAM Angles_1
!-----------------------------------------------------------------------
! Program demonstrating the use of a subroutine PrintDegrees to
! display an angle in degrees.  Variables used are:
!  NumDegrees : degrees in the angle measurement
!  NumMinutes : minutes in the angle measurement
!  NumSeconds : seconds in the angle measurement
!  Response   : user response to more-data question
!
! INPUT:  NumDegrees, NumMinutes, NumSeconds, Response
! OUTPUT: Equivalent measure in degrees (displayed by PrintDegrees)
!-----------------------------------------------------------------------
  IMPLICIT NONE
  INTEGER::NumDegrees, NumMinutes, NumSeconds
  CHARACTER::Response

  ! Read and convert angles until user signals no more data
  DO
     WRITE(*,*) 'Enter degrees, minutes, and seconds:'
     READ(*,*) NumDegrees, NumMinutes, NumSeconds
     CALL PrintDegrees(NumDegrees, NumMinutes, NumSeconds)
     WRITE (*,*) 'More angles (Y or N)?'
     READ(*,*) Response
     IF (Response /= "Y") EXIT
  END DO

CONTAINS

  !-PrintDegrees--------------------------------------------------------
  ! Subroutine to display a measurement of Degrees, Minutes, Seconds
  ! as the equivalent degree measure.
  !
  ! Accepts:  Degrees, Minutes, Seconds
  ! Output:   Degrees, Minutes, and Seconds and equivalent degree measure
  !---------------------------------------------------------------------
```

```
  SUBROUTINE PrintDegrees(Degrees, Minutes, Seconds)

    INTEGER, INTENT(IN) :: Degrees, Minutes, Seconds

    WRITE(*,10) Degrees, Minutes, Seconds, &
                REAL(Degrees) + REAL(Minutes)/60.0 + REAL(Seconds)/3600.0
10 FORMAT(I3,' degrees ',I3,' minutes ',I3,' seconds is equivalent to ',&
          F7.3,' degrees.')

  END SUBROUTINE PrintDegrees

END PROGRAM Angles_1
```

Example 2: Subroutine

```
PROGRAM Angles_2
!-----------------------------------------------------------------------
! Program demonstrating the use of an internal subroutine
! Print_Degrees_Minutes_Seconds to display an angle measured in
! radians or degrees in degrees-minutes-seconds format.
!     Angle    : angle measurement
!     Measure  : "R" if radian measure, "D" if degree measure
!     Response : user response to more-data question
! INPUT:  Angle, Measure, and Response
! OUTPUT: Value of Angle in degrees-minutes-seconds format (displayed
!         by Print_Degrees_Minutes_Seconds )
!-----------------------------------------------------------------------
   IMPLICIT NONE
   REAL::Angle
   CHARACTER::Measure, Response

   DO ! Read and convert angles until user signals no more data
      WRITE(*,*) 'Enter angle and R if in radians, D if in degrees:'
      READ(*,*) Angle, Measure
      CALL Print_Degrees_Minutes_Seconds(Angle, Measure)
      WRITE(*,*) 'More angles (Y or N)?'
      READ(*,*) Response
      IF (Response /= "Y") EXIT
   END DO

CONTAINS
  !-Print_Degrees_Minutes_Seconds-------------------------------------
  ! Subroutine to display an angular measurement Angle in either
  ! radians or degrees in a degrees-minutes-seconds format.
  ! Measure is "R" or "D" according to whether Angle is given
  ! in radians or degrees.  Local identifiers used are:
  !   Pi               : the constant pi
  !   Angle_in_Degrees : the degree equivalent of Angle
  !   Degrees          : the number of degrees
  !   Minutes          : the number of minutes
```

```
  !   Seconds          : the number of seconds
  ! Accepts:  Angle and Measure
  ! Output:   Value of Angle in degrees-minutes-seconds format
  !-----------------------------------------------------------------

   SUBROUTINE Print_Degrees_Minutes_Seconds(Angle, Measure)
      REAL,INTENT(IN)::Angle
      CHARACTER,INTENT(IN)::Measure
      REAL::Angle_in_Degrees
      REAL,PARAMETER::Pi=3.141593
      INTEGER::Degrees, Minutes, Seconds

      ! First get the degree equivalent of the angle
      IF (Measure=='R') THEN
         WRITE(*,10) Angle, 'radians'
         Angle_in_Degrees=(180.0 / Pi)*Angle
      ELSE IF (Measure=='D') THEN
         WRITE(*,10) Angle, 'degrees'
         Angle_in_Degrees=Angle
      ELSE
         WRITE(*,*) Measure,' is an illegal type of measure'
         RETURN
      END IF
10    FORMAT(F10.5,1X,A,1X,' is equivalent to')

      ! Now determine the number of degrees, minutes, and seconds
      Degrees = INT(Angle_in_Degrees)
      Angle_in_Degrees = Angle_in_Degrees - REAL(Degrees)
      Degrees = MOD(Degrees, 360)
      Angle_in_Degrees = 60.0 * Angle_in_Degrees
      Minutes = INT(Angle_in_Degrees)
      Angle_in_Degrees = Angle_in_Degrees - REAL(Minutes)
      Seconds = INT(Angle_in_Degrees * 60.0)

      WRITE(*,20) Degrees, Minutes, Seconds
20    FORMAT(I4,' degrees,',I3,' minutes,',I3,' seconds')
   END SUBROUTINE Print_Degrees_Minutes_Seconds
END PROGRAM Angles_2
```

Subroutines

- In the above examples, all of the arguments were `INTENT(IN)`, that is, their values could not be changed within the subroutine.

- However, subroutines are often used when more than one value needs to be returned.

- Subroutines return values by updating the values of the argument(s) with intent `OUT` specified.

- An argument with `INTENT(OUT)` specified will have its value set by the subroutine; no value will be passed in for this variable.

- An argument with `INTENT(INOUT)` specified will have its value passed in by the calling program, and its (possibly updated) value will be returned.

INTENT

- The `INTENT` portion of the declaration indicates how the information is shared between the subroutine and the main program

- `IN` indicates that the calling program is passing the values of the actual arguments to the subprogram. That is, the subprogram has access to the values of the actual arguments in the calling program, at the point where the subprogram was called.

- `OUT` indicates that the subprogram is permitted to modify the values of the arguments. Thus, when the subprogram returns, the values of the actual arguments are equal to the values of the formal arguments.

Example 3: Subroutine

```
PROGRAM Polar_to_Rectangular
!-------------------------------------------------------------------------
! This program accepts the polar coordinates of a point and displays
! the corresponding rectangular coordinates.  The internal subroutine
! Convert_to_Rectangular is used to effect the conversion.
! Variables used are:
!   RCoord, TCoord : polar coordinates of a point
!   XCoord, YCoord : rectangular coordinates of a point
!   Response       : user response to more-data question
!
! INPUT:  RCoord, TCoord, and Response
! OUTPUT: XCoord and YCoord
!-------------------------------------------------------------------------
   IMPLICIT NONE
   REAL::RCoord,TCoord,XCoord,YCoord
   CHARACTER::Response

! Read and convert coordinates until user signals no more data
   DO
      WRITE(*,*) 'Enter polar coordinates (in radians):'
      READ(*,*) RCoord,TCoord
      CALL Convert_to_Rectangular(RCoord,TCoord,XCoord,YCoord)
      WRITE(*,*) 'Rectangular coordinates:', XCoord, YCoord
      WRITE(*,*) 'More points to convert (Y or N)?'
      READ(*,*) Response
      IF (Response /= 'Y') EXIT
   END DO

CONTAINS
  !-Convert_to_Rectangular-----------------------------------------
  ! Subroutine to convert polar coordinates (R, Theta) to rectangular
  ! coordinates (X, Y).
  ! Accepts:  Polar coordinates R and Theta (in radians)
  ! Returns:  Rectangular coordinates X and Y
  !----------------------------------------------------------------
```

```
   SUBROUTINE Convert_to_Rectangular(R, Theta, X, Y)
      REAL,INTENT(IN)::R,Theta
      REAL,INTENT(OUT)::X,Y
      X = R * COS(Theta)
      Y = R * SIN(Theta)
   END SUBROUTINE Convert_to_Rectangular

END PROGRAM Polar_to_Rectangular
```

Example 4: Subroutine

```
PROGRAM Dice_roll
    IMPLICIT NONE
    INTEGER :: spots, count, numrolls, die1, die2, pair, roll
    REAL :: r1, r2
    CHARACTER :: response

    WRITE(*,*) 'Enter number of times to roll'
    READ(*,*) numrolls

    ! Seed the random number generator
    CALL RANDOM_SEED

    DO
        WRITE(*,*) 'Enter the number of spots to count'
        READ(*,*) spots
        count = 0
        DO roll = 1, numrolls
            ! Get random numbers between 0 and 1
            CALL RANDOM_NUMBER(r1)
            CALL RANDOM_NUMBER(r2)
            ! Compute values for each die
            die1 = 1 + INT(6*r1)
            die2 = 1 + INT(6*r2)
            pair = die1 + die2
            IF (pair == spots) count = count + 1
        END DO

        WRITE(*,*) 'The relative frequency of ',spots,' was ',&
                   REAL(count) / REAL(numrolls)
        WRITE(*,*) 'Roll again? (Y/N)'
        READ(*,*) response
        IF (response /= 'y') EXIT
    END DO
END PROGRAM Dice_roll
```