FORTRAN Subroutines

Syntax

Form 1

```
SUBROUTINE  subroutine-name (arg1, arg2, ..., argn)
   IMPLICIT  NONE
   [specification part]
   [execution part]
   [subprogram part]
END SUBROUTINE  subroutine-name
```

Form 2

```
SUBROUTINE  subroutine-name ()
   IMPLICIT  NONE
   [specification part]
   [execution part]
   [subprogram part]
END SUBROUTINE  subroutine-name
```

Form 3

```
SUBROUTINE  subroutine-name
   IMPLICIT  NONE
   [specification part]
   [execution part]
   [subprogram part]
END SUBROUTINE  subroutine-name
```

FORTRAN Subroutines

1. Subroutine can be *internal* or *external* to a program or a module.
2. A subroutine is a self-contained unit that receives some "input" from the outside world via its formal arguments, does some computations, and then returns the results, if any, with some formal arguments.
3. Unlike functions, the name of a subroutine is simply a name for identification purpose and cannot be used in computation.
4. Subroutines can only be `CALL`ed.
5. A subroutine receives its input values from formal arguments, does computations, and saves the results in some of its formal arguments. When the control of execution reaches `END SUBROUTINE`, the values stored in some formal arguments are passed back to their corresponding actual arguments.
6. Any statements that can be used in a `PROGRAM` can also be used in a `SUBROUTINE`.

Arguments' `INTENT`

1. A formal argument can be declared with `INTENT(IN)`, `INTENT(OUT)`, or `INTENT(INOUT)`.
2. If an argument only receives value from outside of the subroutine, its intent is `INTENT(IN)`. This is the simplest case.
3. An argument does not have to receive anything from outside of the subroutine. It can be used to pass a computation result back to the outside world. In this case, its intent is `INTENT(OUT)`. In a subroutine, an argument declared with `INTENT(OUT)` is supposed to hold a computation result so that the final value can be passed "out".
4. An argument can receive a value, use it for computation, and hold a result so that it can be passed back to the outside world. In this case, its intent is `INTENT(INOUT)`.

5. In subroutine `Means()`, formal arguments `a`, `b` and `c` receive values from outside and are used to compute results into `Am`, `Gm` and `Hm`. The values stored in `Am`, `Gm` and `Hm` are passed back.

```
SUBROUTINE  Means(a, b, c, Am, Gm, Hm)
   IMPLICIT  NONE
   REAL, INTENT(IN)  :: a, b, c
   REAL, INTENT(OUT) :: Am, Gm, Hm
      ..........
END SUBROUTINE  Means
```

6. In subroutine `Swap()`, formal arguments `a` and `b` receive values from outside and are used to compute some results which are stored back to `a` and `b`.

```
SUBROUTINE  Swap(a, b)
   IMPLICIT  NONE
   INTEGER, INTENT(INOUT) :: a, b
      ..........
END SUBROUTINE  Swap
```

The `CALL` Statement

Syntax

```
CALL subroutine-name (arg1, arg2, ..., argn)

CALL subroutine-name ()
CALL subroutine-name
```

- When the `CALL` statement is executed, values of actual arguments are passed to those formal arguments declared with `INTENT(IN)` and `INTENT(INOUT)`.
- Then, the statements of the called subroutine are executed.
- When the execution reaches `END SUBROUTINE`, values stored in those formal arguments declared with `INTENT(OUT)` and `INTENT(INOUT)` are passed back to their corresponding actual arguments.
- The caller executes the statement following the `CALL` statement.

Short Examples

The larger value of two.

```
PROGRAM  Example1
   IMPLICIT  NONE
   INTEGER   a, b, c
      .........
   CALL  Larger(a, b, c)
      .........
END PROGRAM  Example1
```

```
SUBROUTINE  Larger(u, v, w)
   IMPLICIT  NONE
   INTEGER, INTENT(IN)  :: u, v
   INTEGER, INTENT(OUT) :: w
   IF (u > v) THEN
      w = u
   ELSE
      w = v
   END IF
END SUBROUTINE  Larger
```

Sort two numbers into increasing order.

```
PROGRAM  Example2
   IMPLICIT  NONE
   INTEGER   a, b
      .........
   CALL  Sort(a, b)
      .........
END PROGRAM  Example2
```

```
SUBROUTINE  Sort(u, v)
   IMPLICIT  NONE
   INTEGER, INTENT(INOUT) :: u, v
   INTEGER                :: w
   IF (u > v) THEN
      w = u
      u = v
      v = w
   END IF
END SUBROUTINE  Sort
```

No particular meaning.

```
PROGRAM  Example3
   IMPLICIT  NONE
   INTEGER :: a, b, c
      ..........
   READ(*,*)  a
   b = 0
   CALL  DoSOmething(a,b,c)
   WRITE(*,*)  a, b, c
      ..........
END PROGRAM  Example3

---------------------------------

SUBROUTINE  DoSomething(p, q, r)
   IMPLICIT  NONE
   INTEGER, INTENT(IN)     :: p
   INTEGER, INTEGER(INOUT) :: q
   INTEGER, INTENT(OUT)    :: r
   IF (p > 3) THEN
      q = q + 1
      r = 1
   ELSE IF (p < -3) THEN
      q = q - 1
      r = 2
   ELSE
      r = 3
   END IF
END SUBROUTINE  DoSomething
```

Computing Means

```
PROGRAM  Mean6
   IMPLICIT  NONE
   REAL :: u, v, w
   REAL :: ArithMean, GeoMean, HarmMean

   READ(*,*)  u, v, w

   CALL  Means(u, v, w, ArithMean, GeoMean, HarmMean)

   WRITE(*,*) "Arithmetic Mean = ", ArithMean
   WRITE(*,*) "Geometric Mean  = ", GeoMean
   WRITE(*,*) "Harmonic Mean   = ", HarmMean

CONTAINS

   SUBROUTINE  Means(a, b, c, Am, Gm, Hm)
      IMPLICIT  NONE
      REAL, INTENT(IN)  :: a, b, c
      REAL, INTENT(OUT) :: Am, Gm, Hm

      Am = (a + b + c)/3.0
      Gm = (a * b * c)**(1.0/3.0)
      Hm = 3.0/(1.0/a + 1.0/b + 1.0/c)
   END SUBROUTINE  Means

END PROGRAM  Mean6
```

Triangle Area

```
PROGRAM  HeronFormula
   IMPLICIT  NONE
   REAL    :: Side1, Side2, Side3
   REAL    :: Answer
   LOGICAL :: ErrorStatus
   READ(*,*)  Side1, Side2, Side3
   CALL  TriangleArea(Side1, SIde2, Side3, Answer, ErrorStatus)
   IF (ErrorStatus) THEN
      WRITE(*,*)  "ERROR: not a triangle"
   ELSE
      WRITE(*,*)  "The triangle area is ", Answer
   END IF

CONTAINS
   SUBROUTINE  TriangleArea(a, b, c, Area, Error)
      IMPLICIT  NONE
      REAL, INTENT(IN)     :: a, b, c
      REAL, INTENT(OUT)    :: Area
      LOGICAL, INTENT(OUT) :: Error
      REAL                 :: s
      LOGICAL              :: Test1, Test2

      Test1 = (a > 0) .AND. (b > 0) .AND. (c > 0)
      Test2 = (a+b > c) .AND. (a+c > b) .AND. (b+c > a)
      IF (Test1 .AND. Test2) THEN
         Error = .FALSE.
         s     = (a + b + c)/2.0
         Area  = SQRT(s*(s-a)*(s-b)*(s-c))
      ELSE
         Error = .TRUE.
         Area  = 0.0
      END IF
   END SUBROUTINE  TriangleArea
END PROGRAM  HeronFormula
```

YYYYMMDD to Year, Month and Day

```
PROGRAM  YYYYMMDDConversion
   IMPLICIT  NONE
   INTERFACE
      SUBROUTINE  Conversion(Number, Year, Month, Day)
         INTEGER, INTENT(IN)  :: Number
         INTEGER, INTENT(OUT) :: Year, Month, Day
      END SUBROUTINE  Conversion
   END INTERFACE
   INTEGER :: YYYYMMDD, Y, M, D
   DO
      WRITE(*,*)  "A YYYYMMDD please (0 to stop) -> "
      READ(*,*)   YYYYMMDD
      IF (YYYYMMDD == 0)  EXIT

      CALL  Conversion(YYYYMMDD, Y, M, D)

      WRITE(*,*)  "Year  = ", Y
      WRITE(*,*)  "Month = ", M
      WRITE(*,*)  "Day   = ", D
      WRITE(*,*)
   END DO
END PROGRAM  YYYYMMDDConversion

SUBROUTINE  Conversion(Number, Year, Month, Day)
   IMPLICIT  NONE
   INTEGER, INTENT(IN)  :: Number
   INTEGER, INTENT(OUT) :: Year, Month, Day

   Year  = Number / 10000
   Month = MOD(Number, 10000) / 100
   Day   = MOD(Number, 100)
END SUBROUTINE  Conversion
```

```
PROGRAM  QuadraticEquation
   IMPLICIT  NONE
   INTEGER, PARAMETER :: NO_ROOT       = 0
   INTEGER, PARAMETER :: REPEATED_ROOT = 1
   INTEGER, PARAMETER :: DISTINCT_ROOT = 2
   INTEGER            :: SolutionTypee
   REAL               :: a, b, c, r1, r2
   READ(*,*)  a, b, c
   CALL  Solver(a, b, c, r1, r2, SolutionType)
   SELECT CASE (SolutionType)
      CASE (NO_ROOT)
         WRITE(*,*)  "no real root"
      CASE (REPEATED_ROOT)
         WRITE(*,*)  "repeated root ", r1
      CASE (DISTINCT_ROOT)
         WRITE(*,*)  "two roots ", r1, " and ", r2
   END SELECT
CONTAINS
   SUBROUTINE  Solver(a, b, c, Root1, Root2, Type)
      IMPLICIT  NONE
      REAL, INTENT(IN)     :: a, b, c
      REAL, INTENT(OUT)    :: Root1, Root2
      INTEGER, INTENT(OUT) :: Type
      REAL                 :: d
      Root1 = 0.0
      Root2 = 0.0
      d     = b*b - 4.0*a*c
      IF (d < 0.0) THEN
         Type  = NO_ROOT
      ELSE IF (d == 0.0) THEN
         Type  = REPEATED_ROOT
         Root1 = -b/(2.0*a)
      ELSE
         Type  = DISTINCT_ROOT
         d     = SQRT(d)
         Root1 = (-b + d)/(2.0*a)
         Root2 = (-b - d)/(2.0*a)
      END IF
   END SUBROUTINE  Solver
END PROGRAM  QuadraticEquation
```

Mean, Variance and Standard Deviation

$$
\begin{aligned}
\text{Mean} &= \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right) \\
\text{Variance} &= \frac{1}{n-1}\left(\sum_{i=1}^{n} x_i^2 - \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)^2\right) \\
\text{Standard Deviation} &= \sqrt{\text{Variance}}
\end{aligned}
$$

```
PROGRAM  MeanVariance
   IMPLICIT   NONE
   INTEGER :: Number, IOstatus
   REAL    :: Data, Sum, Sum2
   REAL    :: Mean, Var, Std

   Number = 0
   Sum    = 0.0
   Sum2   = 0.0
   DO
      READ(*,*,IOSTAT=IOstatus)  Data
      IF (IOstatus < 0)  EXIT
      Number = Number + 1
      WRITE(*,*)  "Data item ", Number, ": ", Data
      CALL  Sums(Data, Sum, Sum2)
   END DO

   CALL  Results(Sum, Sum2, Number, Mean, Var, Std)
   CALL  PrintResult(Number, Mean, Var, Std)
```

```
CONTAINS

   SUBROUTINE  Sums(x, Sum, SumSQR)
      IMPLICIT  NONE
      REAL, INTENT(IN)    :: x
      REAL, INTENT(INOUT) :: Sum, SumSQR
      Sum    = Sum + x
      SumSQR = SumSQR + x*x
   END SUBROUTINE  Sums

   SUBROUTINE  Results(Sum, SumSQR, n, Mean, Variance, StdDev)
      IMPLICIT  NONE
      INTEGER, INTENT(IN) :: n
      REAL, INTENT(IN)    :: Sum, SumSQR
      REAL, INTENT(OUT)   :: Mean, Variance, StdDev

      Mean = Sum / n
      Variance = (SumSQR - Sum*Sum/n)/(n-1)
      StdDev   = SQRT(Variance)
   END SUBROUTINE

   SUBROUTINE  PrintResult(n, Mean, Variance, StdDev)
      IMPLICIT  NONE
      INTEGER, INTENT(IN) :: n
      REAL, INTENT(IN)    :: Mean, Variance, StdDev

      WRITE(*,*)
      WRITE(*,*) "No. of data items  = ", n
      WRITE(*,*) "Mean               = ", Mean
      WRITE(*,*) "Variance           = ", Variance
      WRITE(*,*) "Standard Deviation = ", StdDev
   END SUBROUTINE  PrintResult

END PROGRAM  MeanVariance
```

More About Argument Association

The corresponding actual argument of a formal argument declared with `INTENT(OUT)` or `INTENT(INOUT)` must be a variable.

```
PROGRAM  Errors                  SUBROUTINE  Sub(u,v,w,p,q)
   IMPLICIT  NONE                   IMPLICIT  NONE
   INTEGER :: a, b, c               INTEGER, INTENT(OUT)   :: u
      ..........                    INTEGER, INTENT(INOUT) :: v
   CALL  Sub(1,a,b+c,(c),1+a)       INTEGER, INTENT(IN)    :: w
      ..........                    INTEGER, INTENT(OUT)   :: p
END PROGRAM  Errors                 INTEGER, INTENT(IN)    :: q
                                       ..........
                                 END SUBROUTINE  Sub
```

$\Rightarrow$: `INTENT(IN)` $\Leftarrow$: `INTENT(OUT)` $\Leftrightarrow$: `INTENT(INOUT)`

- `1` $\Leftarrow$ `u`: **Wrong**
- `a` $\Leftrightarrow$ `v`: **Wrong**
- `b+c` $\Rightarrow$ `w`: correct
- `(c)` $\Leftarrow$ `p`: **Wrong**
- `1+a` $\Rightarrow$ `q`: correct.