FORTRAN Functions

Syntax

Form 1

```
type FUNCTION  function-name (arg1, arg2, ..., argn)
   IMPLICIT  NONE
   [specification part]
   [execution part]
   [subprogram part]
END FUNCTION  function-name
```

Form 2

```
type FUNCTION  function-name ()
   IMPLICIT  NONE
   [specification part]
   [execution part]
   [subprogram part]
END FUNCTION  function-name
```

Function Design Guidelines

1. **Function Header**

   ```
   REAL FUNCTION  TriangleArea(a, b, c)
   ```

   (a) A function requires a *name* (`TriangleArea`).

   (b) The arguments, `a`, `b` and `c`, receive information from outside of a function.

   (c) The function uses these argument values to compute a function value, which requires a type (`REAL`).

   The arguments are called *formal arguments*.

2. **Declaring the Arguments**

   ```
   REAL, INTENT(IN) :: a, b, c
   ```

   (a) All arguments **must** be declared with their intents. For functions, using `INTENT(IN)` means that they receive information from outside and *should not* be changed.

3. **Other Specifications**
   Do it as if you are writing a program.

4. **Executable Statements**
   Do it as if you are writing a program.

5. **The End of a Function**

```
END FUNCTION  TriangleArea
```

The last line of a function must be `END FUNCTION` followed by the function name.

6. **Function Name as a Special Variable**

```
s = (a + b + c)/2.0
TriangleArea = SQRT(s*(s-a)*(s-b)*(s-c))
```

When reaches `END FUNCTION`, the *most* recent value stored in the function name is returned. **Never use the function name in the right-hand side of an expression.**

7. **A Simple Example**

```
REAL FUNCTION  TriangleArea(a, b, c)
   IMPLICIT  NONE
   REAL, INTENT(IN) :: a, b, c
   REAL             :: s
   s = (a + b + c)/2.0
   TriangleArea = SQRT(s*(s-a)*(s-b)*(s-c))
END FUNCTION  TriangleArea
```

Function Examples

1. The following computes the sum of the three arguments:

```
INTEGER FUNCTION  Sum(a, b, c)
   IMPLICIT  NONE

   INTEGER, INTENT(IN) :: a, b, c

   Sum = a + b + c
END FUNCTION  Sum
```

2. The following returns `.TRUE.` if the only argument is positive:

```
LOGICAL FUNCTION  Positive(a)
   IMPLICIT  NONE
   REAL, INTENT(IN) :: a

   IF (a > 0.0)  THEN
      Positive = .TRUE.
   ELSE
      Positive = .FALSE.
   END IF
END FUNCTION  Positive
```

The following is a shorter version which uses `LOGICAL` assignment:

```
LOGICAL FUNCTION  Positive(a)
   IMPLICIT  NONE
   REAL, INTENT(IN) :: a

   Positive = a > 0.0
END FUNCTION  Positive
```

3. The following returns the larger root of quadratic equation $ax^2 + bx + c = 0$:

```
REAL FUNCTION  LargerRoot(a, b, c)
   IMPLICIT  NONE
   REAL, INTENT(IN) :: a, b, c
   REAL             :: d, r1, r2

   d  = SQRT(b*b - 4.0*a*c)
   r1 = (-b + d) / (2.0*a)
   r2 = (-b - d) / (2.0*a)
   IF (r1 >= r2) THEN
      LargerRoot = r1
   ELSE
      LargerRoot = r2
   END IF
END FUNCTION  LargerRoot
```

4. The following computes the factorial of n:

```
INTEGER FUNCTION  Factorial(n)
   IMPLICIT  NONE
   INTEGER, INTENT(IN) :: n
   INTEGER             :: i, Ans

   Ans = 1
   DO i = 1, n
      Ans = Ans * i
   END DO
   Factorial = Ans
END FUNCTION
```

5. What is wrong with the following version?

```
INTEGER FUNCTION  Factorial(n)
   IMPLICIT  NONE

   INTEGER, INTENT(IN) :: n
   INTEGER             :: i

   Factorial = 1
   DO i = 1, n
      Factorial = Factorial * i
   END DO
END FUNCTION
```

6. The following function reads in and returns a positive number. Note that this function does not have argument.

```
REAL FUNCTION  GetNumber()
   IMPLICIT  NONE

   DO
      WRITE(*,*)  'A positive real number: '
      READ(*,*)   GetNumber
      IF (GetNumber > 0.0)  EXIT
      WRITE(*,*)  'ERROR.  Try again.'
   END DO
   WRITE(*,*)
END FUNCTION  GetNumber
```

Common Problems

1. **Forget the Type of a Function**: *BAD*

```
FUNCTION  DoSomething(a, b)
   IMPLICIT  NONE

   INTEGER, INTENT(IN) :: a, b

   DoSomething = SQRT(a*a + b*b)
END FUNCTION  DoSomething
```

2. **Forget** `INTENT(IN)`: *BAD*

```
REAL FUNCTION  DoSomething(a, b)
   IMPLICIT  NONE

   INTEGER :: a, b

   DoSomething = SQRT(a*a + b*b)
END FUNCTION  DoSomething
```

3. **Change the value of a formal argument declared with** `INTENT(IN)`: *ERROR*

```
REAL FUNCTION  DoSomething(a, b)
   IMPLICIT  NONE

   INTEGER, INTENT(IN) :: a, b

   IF (a > b) THEN
      a = a - b
   ELSE
      a = a + b
   END IF
   DoSomething = SQRT(a*a + b*b)
END FUNCTION  DoSomething
```

4. **Forget to store a value to the function name**: *BAD* and *INCORRECT*

```
REAL FUNCTION  DoSomething(a, b)
   IMPLICIT  NONE

   INTEGER, INTENT(IN) :: a, b
   INTEGER             :: c

   c = SQRT(a*a + b*b)
END FUNCTION  DoSomething
```

5. **Function name is used in the right-hand side of an expression**: *ERROR*

```
REAL FUNCTION  DoSomething(a, b)
   IMPLICIT  NONE

   INTEGER, INTENT(IN) :: a, b

   DoSomething = a*a + b*b
   DoSomething = SQRT(DoSomething)
END FUNCTION  DoSomething
```

Where Do My Functions Go?

There are two types of functions: *internal* and *external*. Internal functions are part of the main program as follows:

```
PROGRAM  main
   IMPLICIT  NONE
   ..........
CONTAINS
   INTEGER FUNCTION  Sum(...)
      IMPLICIT  NONE
      ........
   END FUNCTION  Sum

   REAL FUNCTION  Average(...)
      IMPLICIT  NONE
      ........
   END PFUNCTION Average
END PROGRAM  main
```

An Example

```
PROGRAM  TwoFunctions
   IMPLICIT  NONE
   INTEGER :: a, b, BiggerOne
   REAL    :: GeometricMean

   READ(*,*)  a, b
   BiggerOne = Large(a,b)
   GeometricMean = GeoMean(a,b)
   WRITE(*,*)  BiggerOne, GeometricMean

CONTAINS
   INTEGER FUNCTION  Large(a, b)
      IMPLICIT  NONE
      INTEGER, INTENT(IN) :: a, b
      IF (a >= b) THEN
         Large = a
      ELSE
         Large = b
      END IF
   END FUNCTION  Large

   REAL FUNCTION  GeoMean(a, b)
      IMPLICIT  NONE
      INTEGER, INTENT(IN) :: a, b
      GeoMean = SQRT(REAL(a*b))
   END FUNCTION  GeoMean
END PROGRAM  TwoFunctions
```

How to Use Functions?

1. **Use it as if you are using `SIN(x)`, `LOG(x)`, etc.**
   Functions can be used in expressions and `WRITE(*,*)`.
   The arguments are called
   actual arguments.

```
PROGRAM  Example
   IMPLICIT  NONE
   INTEGER :: a, b, c
   READ(*,*)  a, b
   c = Sum(a, b)
   WRITE(*,*)  c
   WRITE(*,*)  Sum(a, c)
   ..........
CONTAINS
   INTEGER FUNCTION  Sum(x, y)
      IMPLICIT  NONE
      INTEGER, INTENT(IN) :: x, y
      Sum = x + y
   END FUNCTION  Sum
END PROGRAM  Example
```

2. **The types of corresponding actual and formal arguments must be** *identical*:

```
PROGRAM  Example
   IMPLICIT  NONE
   INTEGER :: a, b
   REAL    :: p, q
   READ(*,*)  a, b, p, q
   c = Sum(a, p)
   ..........
CONTAINS
   INTEGER FUNCTION  Sum(x, y)
      IMPLICIT  NONE
      INTEGER, INTENT(IN) :: x, y
      Sum = x + y
   END FUNCTION  Sum
END PROGRAM  Example
```

3. **The number of actual and formal arguments must be** *equal*:

```
PROGRAM  Example
   IMPLICIT  NONE
   INTEGER :: a, b, c, d
   ..........
   d = Sum(a, b, c)
   ..........
CONTAINS
   INTEGER FUNCTION  Sum(x, y)
      IMPLICIT  NONE
      INTEGER, INTENT(IN) :: x, y
      Sum = x + y
   END FUNCTION  Sum
END PROGRAM  Example
```

Argument Association

1. In the use of a function (*i.e.*, function call), if an actual argument is an expression, it is evaluated and the result is saved to a temporary location and the value stored there is passed to the corresponding formal argument.

2. If an actual argument is a constant, it is considered as an expression. Thus, its value is saved to a temporary location and the value stored there is passed.

3. If an actual argument is a variable, its value is passed to the formal argument directly.

4. If a formal argument is declared with `INTENT(IN)`, its value is supposed to be "passed-in" and cannot be changed.

5. Consequently, if a formal argument is declared *without* `INTENT(IN)`, then its value may be changed in that function.

Argument Association Examples – Part 1

```
INTEGER :: a, b, c

a = 10
b = 5
c = 13
WRITE(*,*) Small(a,b,c)
WRITE(*,*) Small(a+b,b+c,c)
WRITE(*,*) Small(1, 5, 3)
WRITE(*,*) Small((a),(b),(c))
```

```
INTEGER FUNCTION  Small(x, y, z)
   IMPLICIT  NONE
   INTEGER, INTENT(IN) :: x, y, z

   IF (x <= y .AND. x <= z) THEN
      Small = x
   ELSE IF (y <= x .AND. y <= z) THEN
      Small = y
   ELSE
      Small = z
   END IF
END FUNCTION  Small
```

WRITE(*,*) Small(a,b,c)

```
+-----+   +-----+   +-----+
|  a  |   |  b  |   |  c  |
+-----+   +-----+   +-----+
   |         |         |
   |         |         |
   |         |         |
   V         V         V
+-----+   +-----+   +-----+
|  x  |   |  y  |   |  z  |
+-----+   +-----+   +-----+
```

Argument Association Examples – Part 2

```
INTEGER :: a, b, c

a = 10
b = 5
c = 13
WRITE(*,*) Small(a,b,c)
WRITE(*,*) Small(a+b,b+c,c)
WRITE(*,*) Small(1, 5, 3)
WRITE(*,*) Small((a),(b),(c))
```

```
INTEGER FUNCTION  Small(x, y, z)
   IMPLICIT  NONE
   INTEGER, INTENT(IN) :: x, y, z

   IF (x <= y .AND. x <= z) THEN
      Small = x
   ELSE IF (y <= x .AND. y <= z) THEN
      Small = y
   ELSE
      Small = z
   END IF
END FUNCTION  Small
```

<u>Small(a+b,b+c,c)</u>

```
temp L.   temp L.                temp L. = temporary location
+-----+   +-----+   +-----+
| a+b |   | b+c |   |  c  |
+-----+   +-----+   +-----+
   |         |         |
   |         |         |
   |         |         |
   V         V         V
+-----+   +-----+   +-----+
|  x  |   |  y  |   |  z  |
+-----+   +-----+   +-----+
```

Argument Association Examples – Part 3

```
INTEGER :: a, b, c

a = 10
b = 5
c = 13
WRITE(*,*) Small(a,b,c)
WRITE(*,*) Small(a+b,b+c,c)
WRITE(*,*) Small(1, 5, 3)
WRITE(*,*) Small((a),(b),(c))
```

```
INTEGER FUNCTION  Small(x, y, z)
   IMPLICIT  NONE
   INTEGER, INTENT(IN) :: x, y, z

   IF (x <= y .AND. x <= z) THEN
      Small = x
   ELSE IF (y <= x .AND. y <= z) THEN
      Small = y
   ELSE
      Small = z
   END IF
END FUNCTION  Small
```

Small(1, 5, 3)

```
temp L.   temp L.   temp L.     temp L. = temporary location
+-----+   +-----+   +-----+
|  1  |   |  5  |   |  3  |
+-----+   +-----+   +-----+
   |         |         |
   |         |         |
   |         |         |
   V         V         V
+-----+   +-----+   +-----+
|  x  |   |  y  |   |  z  |
+-----+   +-----+   +-----+
```

Argument Association Examples – Part 4

```
INTEGER :: a, b, c

a = 10
b = 5
c = 13
WRITE(*,*) Small(a,b,c)
WRITE(*,*) Small(a+b,b+c,c)
WRITE(*,*) Small(1, 5, 3)
WRITE(*,*) Small((a),(b),(c))
```

```
INTEGER FUNCTION  Small(x, y, z)
   IMPLICIT  NONE
   INTEGER, INTENT(IN) :: x, y, z

   IF (x <= y .AND. x <= z) THEN
      Small = x
   ELSE IF (y <= x .AND. y <= z) THEN
      Small = y
   ELSE
      Small = z
   END IF
END FUNCTION  Small
```

Small(1, 5, 3)

```
temp L.   temp L.   temp L.     temp L. = temporary location
+-----+   +-----+   +-----+
| (a)  |  | (b) |   | (c) |
+-----+   +-----+   +-----+
   |         |         |
   |         |         |
   |         |         |
   V         V         V
+-----+   +-----+   +-----+
|  x  |   |  y  |   |  z  |
+-----+   +-----+   +-----+
```

Scope Rules – 1

The scope of an entity is the program or function in which it is declared. It is *local* to that program or function.

```
PROGRAM  Scope_1
   IMPLICIT  NONE
   REAL, PARAMETER :: PI = 3.1415926
   INTEGER         :: m, n
      ...................
CONTAINS
   INTEGER FUNCTION  Funct1(k)
      IMPLICIT  NONE
      INTEGER, INTENT(IN) :: k
      REAL                :: f, g
         ..........
   END FUNCTION  Funct1

   REAL FUNCTION  Funct2(u, v)
      IMPLICIT  NONE
      REAL, INTENT(IN) :: u, v
         ..........
   END FUNCTION  Funct2
END PROGRAM  Scope_1
```

- Variables `k`, `f` and `g` are local to function `Funct1()`. Their scope is function `Funct1()`.
- Variables `u` and `v` are local to function `Funct2()`. Their scope is function `Funct2()`
- Variables `m` and `n` and `PI` are local to the main program.

Scope Rules – 2

A global entity is visible to all contained functions, including the functions in which that entity is declared.

```
PROGRAM  Scope_2
   IMPLICIT  NONE
   INTEGER :: a = 1, b = 2, c = 3

   WRITE(*,*)  Add(a)
   c = 4
   WRITE(*,*)  Add(a)
   WRITE(*,*)  Mul(b,c)

CONTAINS
   INTEGER FUNCTION  Add(q)
      IMPLICIT  NONE
      INTEGER, INTENT(IN) :: q
      Add = q + c
   END FUNCTION  Add

   INTEGER FUNCTION  Mul(x, y)
      IMPLICIT  NONE
      INTEGER, INTENT(IN) :: x, y
      Mul = x * y
   END FUNCTION  Mul
END PROGRAM  Scope_2
```

- Variables `a`, `b` and `c` are global to function `Add()` and `Mul()`.
- Variable `c` in `Add()` is the `c` declared in the main program.
- What values are displayed with the three `WRITE`s? 4, 5 and 8.
- This is called *side effect*. If it is possible, avoid using global variables.

Scope Rules – 3

An entity declared in the scope of another entity is always a different entity even if their names are identical.

```
PROGRAM  Scope_3
   IMPLICIT  NONE
   INTEGER :: i, Max = 5

   DO i = 1, Max
      Write(*,*)  Sum(i)
   END DO

CONTAINS

   INTEGER FUNCTION  Sum(n)
      IMPLICIT  NONE
      INTEGER, INTENT(IN) :: n
      INTEGER             :: i, s
      s = 0
      DO i = 1, n
         s = s + i
      END DO
      Sum = s
   END FUNCTION  Sum
END PROGRAM  Scope_3
```

- Variable `i` declared in the program has a scope of the program.
- Variable `i` declared in `Sum()` has a scope of function `Sum()`.
- Since variable `i` of `Sum()` is declared in the scope of `i` in the main program, it is a totally different entity. In function `Sum()`, when `i` is used, it always refers to the `i` declared in `Sum()` rather than the one declared in the main program.

Compute the cubes of 1, 2, 3, . . ., 10 in both `INTEGER` **and** `REAL` **types.**

```
PROGRAM  Cubes
   IMPLICIT   NONE
   INTEGER, PARAMETER :: Iterations = 10
   INTEGER            :: i
   REAL               :: x
   DO i = 1, Iterations
      x = i
      WRITE(*,*)  i, x, intCube(i), realCube(x)
   END DO

CONTAINS
   INTEGER  FUNCTION  intCube(Number)
      IMPLICIT   NONE
      INTEGER, INTENT(IN) :: Number
      intCube = Number*Number*Number
   END FUNCTION  intCube

   REAL FUNCTION  realCube(Number)
      IMPLICIT  NONE
      REAL, INTENT(IN) :: Number
      realCube = Number*Number*Number
   END FUNCTION  realCube
END PROGRAM  Cubes
```

Cm and Inch conversion.

```
PROGRAM  Conversion
   IMPLICIT  NONE
   REAL, PARAMETER :: Initial = 0.0, Final = 10.0
   REAL, PARAMETER :: Step = 0.5
   REAL            :: x
   x = Initial
   DO
      IF (x > Final)  EXIT
      WRITE(*,*)  x, 'cm = ', Cm_to_Inch(x), 'inch and ',&
                  x, 'inch = ', Inch_to_Cm(x), 'cm'
      x = x + Step
   END DO
CONTAINS
   REAL FUNCTION  Cm_to_Inch(cm)
      IMPLICIT  NONE
      REAL, INTENT(IN) :: cm
      REAL, PARAMETER  :: To_Inch = 0.3937
      Cm_to_Inch = To_Inch * cm
   END FUNCTION  Cm_to_Inch

   REAL FUNCTION  Inch_to_Cm(inch)
      IMPLICIT  NONE
      REAL, INTENT(IN) :: inch
      REAL, PARAMETER  :: To_Cm = 2.54
      Inch_to_Cm = To_Cm * inch
   END FUNCTION  Inch_to_Cm
END PROGRAM  Conversion
```

Triangle Area.

```
PROGRAM  HeronFormula
   IMPLICIT  NONE
   REAL :: a, b, c, TriangleArea

   DO
      WRITE(*,*)  'Three sides of a triangle please --> '
      READ(*,*)   a, b, c
      WRITE(*,*)  'Input sides are ', a, b, c
      IF (TriangleTest(a, b, c))  EXIT  ! exit if not a triangle
      WRITE(*,*)  'Your input CANNOT form a triangle.  Try again'
   END DO

   TriangleArea = Area(a, b, c)
   WRITE(*,*)  'Triangle area is ', TriangleArea

CONTAINS
   LOGICAL FUNCTION  TriangleTest(a, b, c)
      IMPLICIT  NONE
      REAL, INTENT(IN) :: a, b, c
      LOGICAL          :: test1, test2
      test1 = (a > 0.0) .AND. (b > 0.0) .AND. (c > 0.0)
      test2 = (a + b > c) .AND. (a + c > b) .AND. (b + c > a)
      TriangleTest = test1 .AND. test2  ! both must be .TRUE.
   END FUNCTION  TriangleTest

   REAL FUNCTION  Area(a, b, c)
      IMPLICIT  NONE
      REAL, INTENT(IN) :: a, b, c
      REAL             :: s
      s    = (a + b + c) / 2.0
      Area = SQRT(s*(s-a)*(s-b)*(s-c))
   END FUNCTION  Area
END PROGRAM  HeronFormula
```

Newton's Method for Finding Square Root.

```
PROGRAM  SquareRoot
   IMPLICIT  NONE
   REAL    :: Begin, End, Step
   REAL    :: x, SQRTx, MySQRTx, Error
   READ(*,*)  Begin, End, Step
   x = Begin
   DO
      IF (x > End)  EXIT
      SQRTx   = SQRT(x)
      MySQRTx = MySqrt(x)
      Error   = ABS(SQRTx - MySQRTx)
      WRITE(*,*)  x, SQRTx, MySQRTx, Error
      x = x + Step
   END DO

CONTAINS
   REAL FUNCTION  MySqrt(Input)
      IMPLICIT  NONE
      REAL, INTENT(IN) :: Input
      REAL             :: X, NewX
      REAL, PARAMETER  :: Tolerance = 0.00001
      IF (Input == 0.0) THEN
         MySqrt = 0.0
      ELSE
         X = ABS(Input)
         DO
            NewX  = 0.5*(X + Input/X)
            IF (ABS(X - NewX) < Tolerance)  EXIT
            X = NewX
         END DO
         MySqrt = NewX
      END IF
   END FUNCTION  MySqrt
END PROGRAM  SquareRoot
```

Greatest Common Divisor – GCD.

```
PROGRAM  GreatestCommonDivisor
   IMPLICIT  NONE
   INTEGER   :: a, b

   WRITE(*,*) 'Two positive integers please --> '
   READ(*,*)  a, b
   WRITE(*,*) 'The GCD of is ', GCD(a, b)

CONTAINS

   INTEGER FUNCTION  GCD(x, y)
      IMPLICIT  NONE
      INTEGER, INTENT(IN) :: x, y
      INTEGER             :: a, b, c

      a = x
      b = y
      IF (a <= b) THEN
         c = a
         a = b
         b = c
      END IF

      DO
         c = MOD(a, b)
         IF (c == 0) EXIT
         a = b
         b = c
      END DO

      GCD = b
   END FUNCTION  GCD
END PROGRAM  GreatestCommonDivisor
```

What is a Module?

Syntax

```
MODULE  module-name
   IMPLICIT  NONE
   [specification part]
CONTAINS
   [internal-functions]
END MODULE  module-name
```

1. The structure of a module is almost identical to the structure of a program.
2. A module starts with the keyword `MODULE` and ends with `END MODULE`.
3. A module **does not** have any executable statements.
4. As a result, a module cannot exist alone; it must be used with other modules and a main program.

Short Examples

1. A module only contains declarations

```
MODULE  SomeConstants
   IMPLICIT  NONE
   REAL, PARAMETER :: PI = 3.1415926
   REAL, PARAMETER :: g = 980
   INTEGER         :: Counter
END MODULE  SomeConstants
```

2. A module only contains internal functions

```
MODULE  SumAverage

CONTAINS
   REAL FUNCTION  Sum(a, b, c)
      IMPLICIT  NONE
      REAL, INTENT(IN) :: a, b, c
      Sum = a + b + c
   END FUNCTION  Sum

   REAL FUNCTION  Average(a, b, c)
      IMPLICIT  NONE
      REAL, INTENT(IN) :: a, b, c
      Average = Sum(a,b,c)/2.0
   END FUNCTION  Average
END MODULE  SumAverage
```

3. A module only contains declarations and internal functions

```
MODULE  DegreeRadianConversion
   IMPLICIT  NONE
   REAL, PARAMETER :: PI = 3.1415926
   REAL, PARAMETER :: Degree180 = 180.0

   REAL FUNCTION  DegreeToRadian(Degree)
      IMPLICIT  NONE
      REAL, INTENT(IN) :: Degree
      DegreeToRadian = Degree*PI/Degree180
   END FUNCTION  DegreeToRadian

   REAL FUNCTION  RadianToDegree(radian)
      IMPLICIT  NONE
      REAL, INTENT(IN) :: Radian
      RadianToDegree = Radian*Degree180/PI
   END FUNCTION  RadianToDegree
END MODULE  DegreeRadianConversion
```

How to Use a Module?

In any program or module, if you want to use a name or a function declared in a module, do the following:

Syntax

```
USE  module-name

USE  module-name, ONLY: name-1, name-2, ..., name-n
```

1. The main program can use `PARAMETER`s `PI` and `g`, and `INTEGER` variable `Counter`. That is, everything declared in module `SomeContents` can be used by the main program.

```
PROGRAM  MainProgram
   USE   SomeConstants
   IMPLICIT  NONE
   ..........
END PROGRAM  MainProgram
```

```
MODULE  SomeConstants
   IMPLICIT  NONE
   REAL, PARAMETER :: PI = 3.1415926
   REAL, PARAMETER :: g = 980
   INTEGER         :: Counter
END MODULE  SomeConstants
```

2. The main program can use only part of the module. In the following example, the main program only wants to use `PARAMETER PI` and `INTEGER` variable `Counter` of the module. Thus, in any place of the main program, `g` cannot be accessed.

```
MODULE  SomeConstants
   IMPLICIT  NONE
   REAL, PARAMETER :: PI = 3.1415926
   REAL, PARAMETER :: g = 980
   INTEGER         :: Counter
END MODULE  SomeConstants

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

PROGRAM  MainProgram
   USE   SomeConstants, ONLY: PI, Counter
   IMPLICIT  NONE
   ..........
END PROGRAM  MainProgram
```

How to Compile Programs with Modules?

- Normally, your main program and modules are in separate files. Suppose your main program is in file `main.f90` and it needs to use modules in files `a.f90`, `b.f90`, `c.f90` and `d.f90`. The following command compiles your files and generates `a.out`:

  ```
  f90 a.f90 b.f90 c.f90 d.f90  main.f90
  ```

  If you want the executable to be named `main`

  ```
  f90 a.f90 b.f90 c.f90 d.f90  main.f90 -o main
  ```

- The order of module files sometimes is important. You should **list module files that do not use any other modules first, followed by the modules use them and so on, followed by your main program.** If `c.f90` uses `a.f90`, `b.f90` uses `a.f90`, and `d.f90` uses `b.f90`, then use the following:

  ```
  f90 a.f90 b.f90 c.f90 d.f90  main.f90 -o main
  ```

  However, the following is also good:

  ```
  f90 a.f90 c.f90 b.f90 d.f90  main.f90 -o main
  ```

Factorial and Combinatorial Coefficients – Module in file `fact-m.f90`

```
MODULE  FactorialModule
   IMPLICIT  NONE

CONTAINS
   INTEGER FUNCTION  Factorial(n)
      IMPLICIT  NONE
      INTEGER, INTENT(IN) :: n
      INTEGER             :: Fact, i

      Fact = 1
      DO i = 1, n
         Fact = Fact * i
      END DO
      Factorial = Fact
   END FUNCTION  Factorial

   INTEGER FUNCTION  Combinatorial(n, r)
      IMPLICIT  NONE
      INTEGER, INTENT(IN) :: n, r
      INTEGER             :: Cnr

      IF (0 <= r .AND. r <= n) THEN
         Cnr = Factorial(n) / (Factorial(r)*Factorial(n-r))
      ELSE
         Cnr = 0
      END IF
      Combinatorial = Cnr
   END FUNCTION  Combinatorial
END MODULE  FactorialModule
```

Factorial and Combinatorial Coefficients – Main in file main.f90

```
PROGRAM  ComputeFactorial
   USE       FactorialModule

   IMPLICIT  NONE

   INTEGER :: N, R

   WRITE(*,*)  'Two non-negative integers --> '
   READ(*,*)   N, R

   WRITE(*,*)  N,   '! = ', Factorial(N)
   WRITE(*,*)  R,   '! = ', Factorial(R)

   IF (R <= N) THEN
      WRITE(*,*)  'C(', N, ',', R, ') = ', Combinatorial(N, R)
   ELSE
      WRITE(*,*)  'C(', R, ',', N, ') = ', Combinatorial(R, N)
   END IF

END PROGRAM  ComputeFactorial
```

Compile:

```
f90 fact-m.f90  main.f90
```

Trigonometric Functions Using Degrees – Module in file trigon.f90

```
MODULE  MyTrigonometricFunctions
   IMPLICIT   NONE
   REAL, PARAMETER :: PI        = 3.1415926
   REAL, PARAMETER :: Degree180 = 180.0
   REAL, PARAMETER :: R_to_D    = Degree180/PI
   REAL, PARAMETER :: D_to_R    = PI/Degree180

CONTAINS
   REAL FUNCTION  RadianToDegree(Radian)
      IMPLICIT  NONE
      REAL, INTENT(IN) :: Radian
      RadianToDegree = Radian * R_to_D
   END FUNCTION  RadianToDegree

   REAL FUNCTION  DegreeToRadian(Degree)
      IMPLICIT  NONE
      REAL, INTENT(IN) :: Degree
      DegreeToRadian = Degree * D_to_R
   END FUNCTION  DegreeToRadian

   REAL FUNCTION  MySIN(x)
      IMPLICIT  NONE
      REAL, INTENT(IN) :: x
      MySIN = SIN(DegreeToRadian(x))
   END FUNCTION  MySIN

   REAL FUNCTION  MyCOS(x)
      IMPLICIT  NONE
      REAL, INTENT(IN) :: x
      MyCOS = COS(DegreeToRadian(x))
   END FUNCTION  MyCOS
END MODULE  MyTrigonometricFunctions
```

Trigonometric Functions Using Degrees – Main in file `test.f90`

```
PROGRAM  TrigonFunctTest
   USE  MyTrigonometricFunctions

   IMPLICIT  NONE

   REAL :: Begin = -180.0
   REAL :: Final =  180.0
   REAL :: Step  =   10.0
   REAL :: x

   WRITE(*,*)  'Value of PI = ', PI
   WRITE(*,*)
   x = Begin
   DO
      IF (x > Final)  EXIT
      WRITE(*,*)  'x = ',  x, 'deg   sin(x) = ', MySIN(x), &
                  '   cos(x) = ', MyCOS(x)
      x = x + Step
   END DO

END PROGRAM  TrigonFunctTest
```

Compile:

```
f90 trigon.f90  test.f90 -o test
```

A Little Privacy: `PRIVATE` and `PUBLIC`

Syntax

```
PUBLIC  :: name-1, name-2, ..., name-n
PRIVATE :: name-1, name-2, ..., name-n
```

1. In the following, `SkyWalker` and `Princess` are public. By default, `DeathStar` and `WeaponPower` are also public.
2. `VolumeOdDeathStar`, `SecretConstant` and `BlackKnight` are private.

```
MODULE  TheForce
   IMPLICIT   NONE
   INTEGER :: SkyWalker, Princess
   REAL    :: BlackKnight
   LOGICAL :: DeathStar
   REAL, PARAMETER :: SecretConstant = 0.123456
   PUBLIC  :: SkyWalker, Princess
   PRIVATE :: VolumeOdDeathStar
   PRIVATE :: SecretConstant, BlackKnight
CONTAINS
   INTEGER FUNCTION  VolumeOfDeathStar()
      ..........
   END FUNCTION WolumeOfDeathStar

   REAL FUNCTION  WeaponPower(SomeWeapon)
      ..........
   END FUNCTION
END MODULE  TheForce
```

Trigonometric Functions Using Degrees – Module in file `trigon2.f90`

```
MODULE  MyTrigonometricFunctions
   IMPLICIT   NONE
   REAL, PARAMETER :: PI        = 3.1415926
   REAL, PARAMETER :: Degree180 = 180.0
   REAL, PARAMETER :: R_to_D    = Degree180/PI
   REAL, PARAMETER :: D_to_R    = PI/Degree180

   PRIVATE         :: Degree180, R_to_D, D_to_R
   PRIVATE         :: RadianToDegree, DegreeToRadian
   PUBLIC          :: MySIN, MyCOS
CONTAINS
   REAL FUNCTION  RadianToDegree(Radian)
      IMPLICIT  NONE
      REAL, INTENT(IN) :: Radian
      RadianToDegree = Radian * R_to_D
   END FUNCTION  RadianToDegree
   REAL FUNCTION  DegreeToRadian(Degree)
      IMPLICIT  NONE
      REAL, INTENT(IN) :: Degree
      DegreeToRadian = Degree * D_to_R
   END FUNCTION  DegreeToRadian
   REAL FUNCTION  MySIN(x)
      IMPLICIT  NONE
      REAL, INTENT(IN) :: x
      MySIN = SIN(DegreeToRadian(x))
   END FUNCTION  MySIN
   REAL FUNCTION  MyCOS(x)
      IMPLICIT  NONE
      REAL, INTENT(IN) :: x
      MyCOS = COS(DegreeToRadian(x))
   END FUNCTION  MyCOS
END MODULE  MyTrigonometricFunctions
```

Trigonometric Functions Using Degrees – Main in file `test.f90`

```
PROGRAM  TrigonFunctTest
   USE  MyTrigonometricFunctions

   IMPLICIT  NONE

   REAL :: Begin = -180.0
   REAL :: Final =  180.0
   REAL :: Step  =   10.0
   REAL :: x

   WRITE(*,*)  'Value of PI = ', PI
   WRITE(*,*)
   x = Begin
   DO
      IF (x > Final)  EXIT
      WRITE(*,*)  'x = ',  x, 'deg   sin(x) = ', MySIN(x), &
                  '    cos(x) = ', MyCOS(x)
      x = x + Step
   END DO

END PROGRAM  TrigonFunctTest
```

Compile:

```
f90 trigon2.f90  test.f90 -o test
```

Interface Block

Functions do not have to be internal to a main program or a module. They can be stand-alone (*i.e.*, not containing in any program or module). In this case, the function/program that uses a function must have an `INTERFACE` block.

Syntax

```
INTERFACE
   type FUNCTION name(arg-1, arg-2, ..., arg-n)
      type, INTENT(IN) :: arg-1
      type, INTENT(IN) :: arg-2
         ..........
      type, INTENT(IN) :: arg-n
   END FUNCTION  name

   ....... other functions .......
END INTERFACE
```

1. The `INTERFACE` block starts with `INTERFACE` and ends with `END BLOCK`.
2. In the block, it contains one or more function headers and their declarations. Note that only the declarations of the formal arguments are required.

Short Examples

Suppose the following are external functions:

```
INTEGER FUNCTION  Coin(value)
   IMPLICIT  NONE
   INTEGER, INTENT(IN) :: value
      ...........
END FUNCTION  Coin

REAL FUNCTION  Volume(a, b, c)
   IMPLICIT  NONE
   REAL, INTENT(IN) :: a, b, c
      ...........
END FUNCTION  Volume
```

The `INTERFACE` block is:

```
INTERFACE
   INTEGER FUNCTION  Coin(value)
      INTEGER, INTENT(IN) :: value
   END FUNCTION  Coin

   REAL FUNCTION  Volume(a, b, c)
      REAL, INTENT(IN) :: a, b, c
   END FUNCTION  Volume
END INTERFACE
```

If only function `Volume()` is used, then:

```
INTERFACE
   REAL FUNCTION  Volume(a, b, c)
      REAL, INTENT(IN) :: a, b, c
   END FUNCTION  Volume
END INTERFACE
```

Where Does the Interface Block Go?

Answer: after `IMPLICIT NONE`

```
PROGRAM  CoinVolume
   IMPLICIT  NONE

   INTERFACE
      INTEGER FUNCTION  Coin(value)
         INTEGER, INTENT(IN) :: value
      END FUNCTION  Coin

      REAL FUNCTION  Volume(a, b, c)
         REAL, INTENT(IN) :: a, b, c
      END FUNCTION  Volume
   END INTERFACE

   ..... other specification statements .....
   ......... executable statements ..........
END PROGRAM  CoinVolume
```

Note that functions **`Coin()`** and **`Volume()`** can be in the same file of the main program or in a separate file. If they are in a separate file, say **`funct.f90`**, use the following to compile your program:

```
f90 funct.f90 main.f90
```

Cm and Inch Conversion

```
PROGRAM  Conversion
   IMPLICIT  NONE
   INTERFACE
      REAL FUNCTION  Cm_to_Inch(cm)
         REAL, INTENT(IN) :: cm
      END FUNCTION  Cm_to_Inch
      REAL FUNCTION  Inch_to_Cm(inch)
         REAL, INTENT(IN) :: inch
      END FUNCTION  Inch_to_Cm
   END INTERFACE
   REAL, PARAMETER :: Initial = 0.0, Final = 10.0, Step = 0.5
   REAL            :: x
   x = Initial
   DO
      IF (x > Final)  EXIT
      WRITE(*,*)  x, 'cm = ',   Cm_to_Inch(x), 'inch and ',  &
                  x, 'inch = ', Inch_to_Cm(x), 'cm'
      x = x + Step
   END DO
END PROGRAM  Conversion

REAL FUNCTION  Cm_to_Inch(cm)
   IMPLICIT  NONE
   REAL, INTENT(IN) :: cm
   REAL, PARAMETER  :: To_Inch = 0.3937
   Cm_to_Inch = To_Inch * cm
END FUNCTION  Cm_to_Inch

REAL FUNCTION  Inch_to_Cm(inch)
   IMPLICIT  NONE
   REAL, INTENT(IN) :: inch
   REAL, PARAMETER  :: To_Cm = 2.54
   Inch_to_Cm = To_Cm * inch
END FUNCTION  Inch_to_Cm
```

Triangle Area

```
PROGRAM  HeronFormula
   IMPLICIT  NONE
   INTERFACE
      LOGICAL FUNCTION  TriangleTest(a, b, c)
         REAL, INTENT(IN) :: a, b, c
      END FUNCTION  TriangleTest
      REAL FUNCTION  Area(a, b, c)
         REAL, INTENT(IN) :: a, b, c
      END FUNCTION  Area
   END INTERFACE
   REAL :: a, b, c, TriangleArea
   DO
      READ(*,*)   a, b, c
      IF (TriangleTest(a, b, c))  EXIT
      WRITE(*,*)  'Your input CANNOT form a triangle.  Try again'
   END DO
   TriangleArea = Area(a, b, c)
   WRITE(*,*)  'Triangle area is ', TriangleArea
END PROGRAM  HeronFormula
LOGICAL FUNCTION  TriangleTest(a, b, c)
   IMPLICIT  NONE
   REAL, INTENT(IN) :: a, b, c
   LOGICAL          :: test1, test2
   test1 = (a > 0.0) .AND. (b > 0.0) .AND. (c > 0.0)
   test2 = (a + b > c) .AND. (a + c > b) .AND. (b + c > a)
   TriangleTest = test1 .AND. test2
END FUNCTION  TriangleTest
REAL FUNCTION  Area(a, b, c)
   IMPLICIT  NONE
   REAL, INTENT(IN) :: a, b, c
   REAL             :: s
   s    = (a + b + c) / 2.0
   Area = SQRT(s*(s-a)*(s-b)*(s-c))
END FUNCTION  Area
```