The Abdus Salam
International Centre for Theoretical Physics

United Nations
Educational, Scientific and
Cultural Organization

IAEA
International Atomic Energy Agency

1856-29

2007 Summer College on Plasma Physics

30 July - 24 August, 2007

Elementary Programming in FORTRAN 90

D. Shaikh
University of California
Riverside, CA, USA

Elementary Programming in FORTRAN 90

Dastgeer Shaikh

Summer College on Plasma Physics
International Center of Theoretical Physics (ICTP)
Trieste, Italy

August 2007

FORTRAN Compiler for LINUX/WINDOWS

Intel FORTRAN Compiler for LINUX operating system is FREE.

Download from :

`http://www.intel.com/cd/software/products/asmo-na/eng/compilers/flin/index.htm`

WINDOWS NT/2000/XP Users:

GNU FORTRAN compiler (fortran77, some feature of f90) is FREE.
Download from

`http://www.cs.yorku.ca/~roumani/fortran/ftn.htm`

LINUX operating system is also FREE.

FEDORA (32/64 bits) : `http://fedora.redhat.com/Download/`
CENTOS : `http://www.centos.org/`

References

- Fortran 90 handbook

  J.C. Adams et. al., McGraw-Hill, 1992

- Programmer's Guide to Fortran 90

  W.S. Brainerd et. al., Unicomp, 1994

- Fortran 90

  M. Counihan, Pitman, 1991

- Fortran 90 programming

  T.M.R. Ellis et. al., Wesley, 1994

- Fortran 90 for Scientists and Engineers

  B.D. Hahn, Edward Arnold, 1994

- Fortran 90 Explained

  M. Metcalf and J. Ried, Oxford University Press, 1992

- Programming in Fortran 90

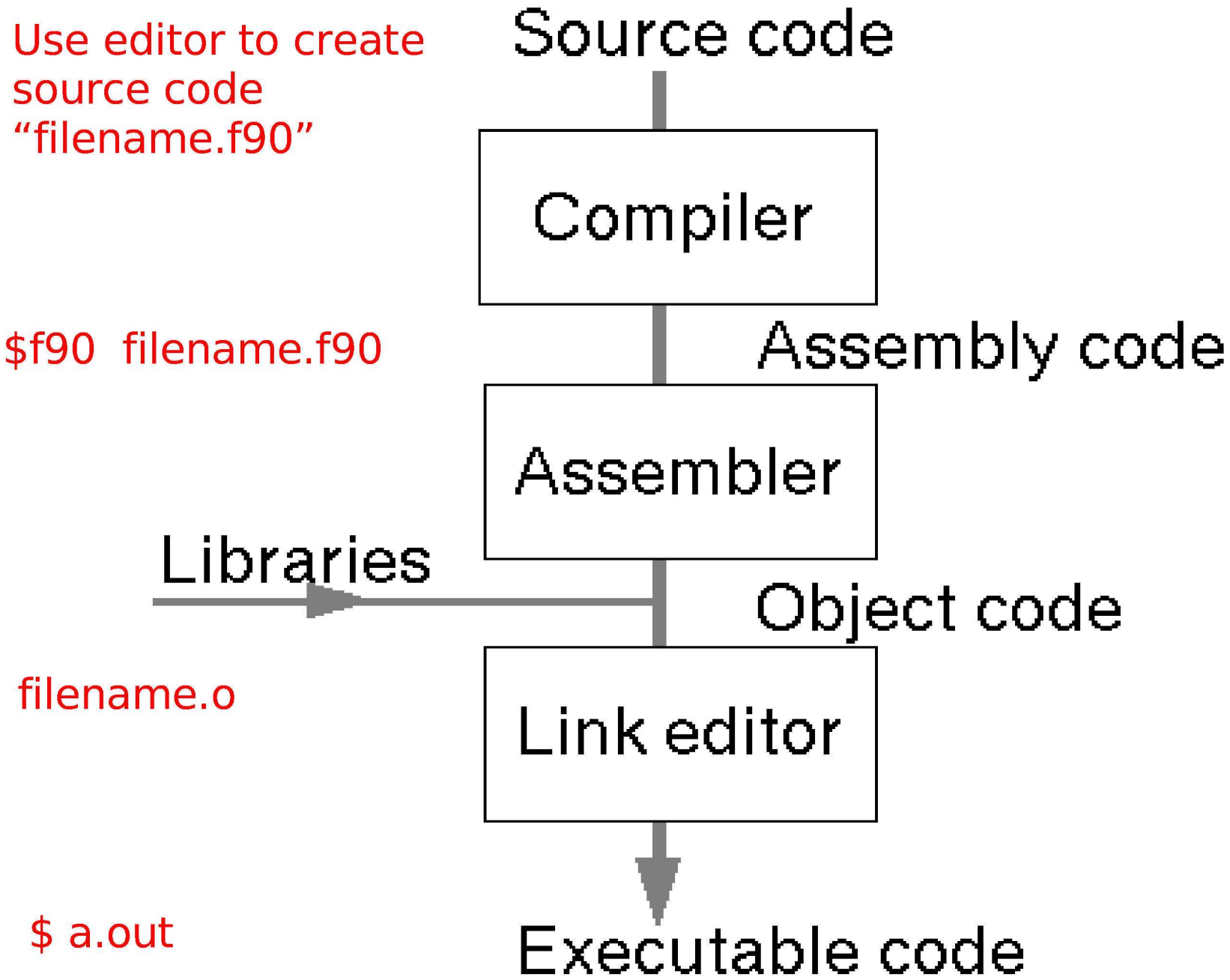
  J.S. Morgan and J.L. Schonfelder, Alfred Walker Ltd, 1993

- Programming in Fortran 90

  I.M. Smith, Wiley (online course material)

Your program should have the following form:

```
PROGRAM    program-name
IMPLICIT   NONE
  [specification part]
  [execution part]
  [subprogram part]
END PROGRAM program-name
```


Use editor to create source code "filename.f90"
Source code
Compiler
$f90 filename.f90
Assembly code
Assembler
Libraries
Object code
filename.o
Link editor
$ a.out
Executable code

Example: A program to print a message.

```
PROGRAM hi
!displays a message
WRITE(*,*) 'Hello World'
END PROGRAM hi
```

(See example1.f90)

What more a program contains?

Identifying name tags

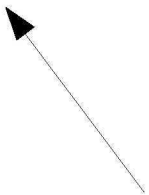
```
daysinyear temperature pi
```

integer numbers:

```
-3124 -96 0 10 365
```

Real numbers:

```
10.3 -8.45 0.00002
2.576x1032 1.3x10-22
2.576E32 1.3E-22
```



Exponential numbers are real numbers

Naming Convention

- Names upto 31 characters long:

  Letters `a...z, A...Z,`

  Numerals `0, 1, 2...9,`

  Underscore _
- First character must be a letter.
- No case sensitivity.
- No reserved words.
- Examples:

  Valid:

  `X, x1, mass, pressure`

  invalid:

  `1x, _time, ten.green.bottles`

Variables

A variable is a user defined entity whose value changes while a program runs.

Declared at the start of the program.

General form:

```
type [,attribute] :: variable list
```

Examples

```
REAL :: temperature, pressure
INTEGER :: count, hours, minutes
```

Variables may be given a value on declaration:

```
REAL :: temperature=96.4
INTEGER :: months=12, weeks=52
```

Parameters

Parameters have a constant value.

Requires the PARAMETER attribute.

Examples:

```
REAL,    PARAMETER :: pi=3.141592
INTEGER, PARAMETER :: maxvalue=1024
INTEGER, PARAMETER :: repeat=1000
```

Parameters must be given an initial value.

Arithmetic Expressions

Operators

- **+** Addition
- **-** Subtraction
- ***** Multiplication
- **/** Division
- ****** Exponentiation

Examples:

```
cost * number

cost * number + postage

10 - 3

4 * pi

1 / pressure

pi * radius **2
```

Assignment Statement

General form:

```
variable = expression
```

Used to assign a value to a variable:

```
pi = 3.141592
radius = 10.0
```

Used to assign the result of an expression to a variable:

```
area = pi*radius*radius
```

The result of the expression should be the same type as the variable.

Simple Input and Output (READ and WRITE)

Input from keyboard via READ statement:

```
READ(5,*) radius
```

Output to screen via WRITE statement:

```
WRITE(6,*) area
WRITE(*,*) area
```

6 or * represent output on screen

Mutliple variables may be specified on the same READ/WRITE statement:

```
READ(5,*) length, breadth
WRITE(6,*) temp, pressure, mass
WRITE(6,*) pi*radius**2, 2.0
```

Example Program

$$\begin{aligned}\text{Arithmetic mean} &= \frac{1}{3}(x+y+z)\\ \text{Geometric mean} &= (x\cdot y\cdot z)^{1/3}\\ \text{Harmonic mean} &= \frac{3}{\frac{1}{x}+\frac{1}{y}+\frac{1}{z}}\end{aligned}$$

(see example2.f90)

```
!-----------------------------------------------------
!   Computes arithmetic, geometric and harmonic means
!-----------------------------------------------------

PROGRAM  ComputeMeans
   IMPLICIT  NONE
   REAL  :: X = 1.0, Y = 2.0, Z = 3.0
   REAL  :: ArithMean, GeoMean, HarmMean
   WRITE(*,*)  'Data items: ', X, Y, Z
   WRITE(*,*)
   ArithMean = (X + Y + Z)/3.0
   GeoMean   = (X * Y * Z)**(1.0/3.0)
   HarmMean  = 3.0/(1.0/X + 1.0/Y + 1.0/Z)
   WRITE(*,*)  'Arithmetic mean = ', ArithMean
   WRITE(*,*)  'Geometric mean  = ', GeoMean
   WRITE(*,*)  'Harmonic Mean   = ', HarmMean
END PROGRAM ComputeMeans
```

Example 3: Quadratic Equation Solution (Real Roots only)

(See example3.f90)
x^2+4x+4=0
(x+2)^2=0
x=2, 2

$$ax^2 + bx + c = 0$$

$$\text{root 1} = \frac{1}{2a}\left(-b + \sqrt{b^2 - 4ac}\right)$$
$$\text{root 2} = \frac{1}{2a}\left(-b - \sqrt{b^2 - 4ac}\right)$$

```
! ------------------------------------------------
!   Solve  Ax^2 + Bx + C = 0 given B*B-4*A*C >= 0
! ------------------------------------------------
PROGRAM  QuadraticEquation
   IMPLICIT  NONE
   REAL  :: a, b, c
   REAL  :: d
   REAL  :: root1, root2
!  read in the coefficients a, b and c
   WRITE(*,*) 'A, B, C Please : '
   READ(*,*)  a, b, c
!  compute the square root of discriminant d
   d  = SQRT(b*b - 4.0*a*c)
!  solve the equation
   root1 = (-b + d)/(2.0*a)    ! first root
   root2 = (-b - d)/(2.0*a)    ! second root
!  display the results
   WRITE(*,*)
   WRITE(*,*)  'Roots are ', root1, ' and ', root2
END PROGRAM  QuadraticEquation
```

Intrinsic function

Fortran Intrinsic Functions

Function	Generic name	Specific name	Data type Arg	Res
Square root	SQRT	SQRT	R	R
		DSQRT	D	D
		CSQRT	C	C
Exponential	EXP	EXP	R	R
		DEXP	D	D
		CEXP	C	C
Natural logarithm	LOG	ALOG	R	R
		DLOG	D	D
		CLOG	C	C
Common logarithm	LOG10	ALOG10	R	R
		DLOG10	D	D

R=REAL, D=DOUBLE PRECESION, C=COMPLEX

Sine	SIN	SIN	R	R
		DSIN	D	D
		CSIN	C	C
Cosine	COS	COS	R	R
		DCOS	D	D
		CCOS	C	C
Tangent	TAN	TAN	R	R
		DTAN	D	D
Arcsine	ASIN	ASIN	R	R
		DASIN	D	D
Arccosine	ACOS	ACOS	R	R
		DCOS	D	D
Arctangent	ATAN	ATAN	R	R
		DATAN	D	D

Hyperbolic sine	SINH	SINH	R	R
		DSINH	D	D
Hyperbolic cosine	COSH	COSH	R	R
		DCOSH	D	D
Hyperbolic tangent	TANH	TANH	R	R
		DTANH	D	D

Programming Example 4: Projectile Motion

This program computes the position (x and y coordinates) and the velocity (magnitude and direction) of a projectile, given t, the time since launch, u, the launch velocity, a, the initial angle of launch (in degree), and g=9.8, the acceleration due to gravity.

The horizontal and vertical displacements are given by the following formulae:

$$x = ut\cos(a)$$
$$y = ut\sin(a) - \frac{gt^2}{2}$$

The horizontal and vertical components of the velocity vector

$$V_x = u\cos(a)$$
$$V_y = u\sin(a) - gt$$

the magnitude of the velocity vector is

$$V = \sqrt{V_x^2 + V_y^2}$$

the angle between the ground and the velocity vector is

$$\tan(\theta) = \frac{V_x}{V_y}$$

```
!-----------------------------------------------------
!Given t, the time since launch, ......
!-----------------------------------------------------
PROGRAM  Projectile
   IMPLICIT    NONE
   REAL, PARAMETER :: g  = 9.8          ! acceleration due to gravity
   REAL, PARAMETER :: PI = 3.1415926    ! you knew this.  didn't you
   REAL            :: Angle             ! launch angle in degree
   REAL            :: Time              ! time to flight
   REAL            :: Theta             ! direction at time in degree
   REAL            :: U                 ! launch velocity
   REAL            :: V                 ! resultant velocity
   REAL            :: Vx                ! horizontal velocity
   REAL            :: Vy                ! vertical velocity
   REAL            :: X                 ! horizontal displacement
   REAL            :: Y                 ! vertical displacement
   READ(*,*)  Angle, Time, U
   Angle = Angle * PI / 180.0           ! convert to radian
   X     = U * COS(Angle) * Time
   Y     = U * SIN(Angle) * Time - g*Time*Time / 2.0
   Vx    = U * COS(Angle)
   Vy    = U * SIN(Angle) - g * Time
   V     = SQRT(Vx*Vx + Vy*Vy)
   Theta = ATAN(Vy/Vx) * 180.0 / PI
   WRITE(*,*)  'Horizontal displacement : ', X
   WRITE(*,*)  'Vertical displacement   : ', Y
   WRITE(*,*)  'Resultant velocity      : ', V
   WRITE(*,*)  'Direction (in degree)   : ', Theta
END PROGRAM  Projectile
```

Program Output (see example4.f90)

If the input to the program consists of the following three real values:

```
   45.0  6.0  60.0
```

The program will generate the following output:

```
Horizontal displacement : 254.558472
Vertical displacement   : 78.158432
Resultant velocity      : 45.4763107
Direction (in degree)   : -21.1030636
```

Mixed Mode Arithmetic Expressions

If operands in an expression contains both **INTEGER** and **REAL** constants or variables, this is a mixed mode arithmetic expression.

In mixed mode arithmetic expressions, **INTEGER** operands are always converted to **REAL** before carrying out any computations. The result is of **REAL** type.

Operator	**INTEGER**	**REAL**
INTEGER	**INTEGER**	**REAL**
REAL	**REAL**	**REAL**

Simple Examples:

- 1 + 3 is 4
- 1.23 - 0.45 is 0.78
- 3 * 8 is 24
- 6.5/1.25 is 5.2
- 8.4/4.2 is 2.0 rather than 2, since the result must be of **REAL** type.
- -5**2 is -25
- 12/4 is 3
- 13/4 is 3 rather than 3.25. Since 13/4 is a single mode arithmetic expression and since all of its operands are of **INTEGER** type, the result must also be of **INTEGER** type. The computer will *truncate* the mathematical result (3.25) making it an integer. Therefore, the result is 3.
- 3/5 is 0 rather than 0.6.

LOGICAL Type and Variables

LOGICAL values are either *true* or *false*. In Fortran, they must be written as **.TRUE.** and **.FALSE.** Note that the two periods must be there; otherwise, they become identifiers.

A variable that can hold one of the logical values is a logical variable and it is of type **LOGICAL**.

To declare a **LOGICAL** variable, do it as what you did for **INTEGER** and **REAL** variables. But, use the type name **LOGICAL** instead.

```
LOGICAL  :: Answer, Condition, Test

LOGICAL  :: Value, Yes_and_No

LOGICAL, PARAMETER :: Answer = .TRUE., Condition = .FALSE.
```

- LOGICAL variables

```
LOGICAL  :: Test = .TRUE., PreTest = .FALSE.
```

- LOGICAL variables Cond_1, Cond_2 and Total are assigned with .TRUE., .TRUE. and .FALSE., respectively:

```
LOGICAL  :: Cond_1, Cond_2, Total
Cond_1 = .TRUE.
Cond_2 = .TRUE.
```

Relational Operators

There are six relational operators:

< : less than

<= : less than or equal to

> : greater than

>= : greater than or equal to

== : equal to

/= : not equal to

- Each of these six relational operators takes two operands. must both be arithmetic or both be strings. For arithmetic operands, if they are of different types (i.e., one INTEGER and the other REAL), the INTEGER operand will be converted to REAL.

- The outcome of a comparison is a LOGICAL value. For example, 5 /= 3 is .TRUE. and 7 + 3 >= 20 is .FALSE.

Example

3**2 + 4**2 == 5**2 is .TRUE.

LOGICAL Operators and Expressions

Fortran has five LOGICAL operators that can only be used with expressions whose results are logical values (i.e., .TRUE. or .FALSE.).

These five logical operators are

.NOT. : logical not

.AND. : logical and

.OR. : logical or

.EQV. : logical equivalence

.NEQV. : logical not equivalence

IF-THEN-ELSE-END IF

```
IF (logical-expression) THEN
   statements-1
ELSE
   statements-2
END IF
```

If logical-exp is true then state1 is executed, if false then state2 is executed

Examples

```
INTEGER :: Number
READ(*,*)  Number
IF (MOD(Number, 2) == 0) THEN
   WRITE(*,*)  Number, ' is even'
ELSE
   WRITE(*,*)  Number, ' is odd'
END IF
```

If an input number is even or odd!!

(See example5.f90)

Following example program finds the smaller number

```
INTEGER  :: a, b, Smaller
READ(*,*)  a, b
IF (a <= b) THEN
   Smaller = a
ELSE
   Smaller = b
END IF
Write(*,*)  'The smaller of ', a, '
and ', &
            b, ' is ', Smaller
```

(See example6.f90)

Counting DO-Loop

```
DO control-var = initial-value, final-value, [step-size]
   statements
END DO
```

Example i: (see example7.f90)

```
INTEGER  :: Count
DO Count = -3, 4, 2
   WRITE(*,*)  Count, Count*Count, Count*Count*Count
END DO
```

Example ii:

```
INTEGER :: Count, Number, Sum, Input
Sum = 0
DO Count = 1, Number
   READ(*,*)  Input
   Sum = Sum + Input
END DO
```

General DO-Loop with EXIT

```
DO
 statements
END DO
```

```
DO
   statements-1
   EXIT
   statements-2
END DO
```

Example :

```
REAL  :: x, y, z
DO
   READ(*,*)  x
   y = x*x
   z = x*x*x
   WRITE(*,*)  x, ' square = ', y, ' cube  = ', z
END DO
```

Example : (See example8.f90)

```
!-----
!Learning DO loop
!--------

INTEGER :: x, Sum
Sum = 0
DO
   READ(*,*)  x
   IF (x < 0)  EXIT
   Sum = Sum + x
   WRITE(*,*)'SUM = ', sum
END DO
```

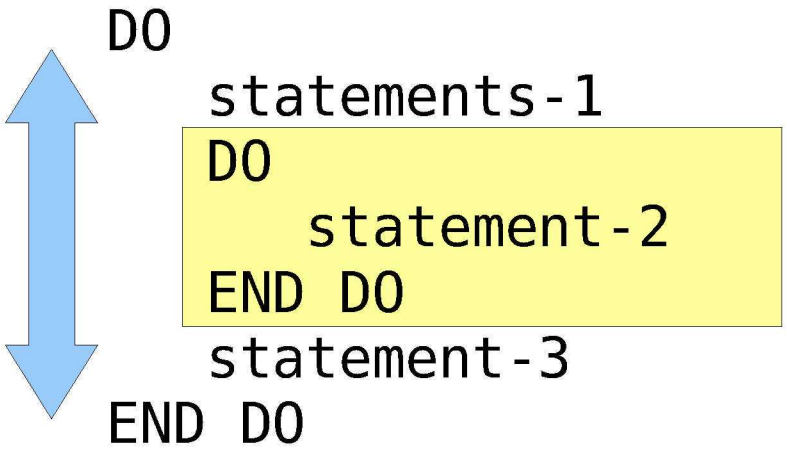
Computing Factorial

(See example9.f90)

$$n! = 1 \times 2 \times 3 \times 4 \times \ldots \times n \quad if\ n > 1$$
$$n! = 1 \quad if\ n = 0$$

```
PROGRAM  Factorial
   IMPLICIT  NONE
   INTEGER :: N, i, Answer
   WRITE(*,*)  'This program computes the factorial of'
   WRITE(*,*)  'a non-negative integer'
   WRITE(*,*)
   WRITE(*,*)  'What is N in N! --> '
   READ(*,*)   N
   WRITE(*,*)
   IF (N < 0) THEN                    ! input error if N < 0
      WRITE(*,*)  'ERROR: N must be non-negative'
      WRITE(*,*)  'Your input N = ', N
   ELSE IF (N == 0) THEN              ! 0! = 1
      WRITE(*,*)  '0! = 1'
   ELSE                               ! N > 0 here
      Answer = 1                      ! initially N! = 1
      DO i = 1, N                     ! for each i = 1, 2, ..., N
         Answer = Answer * i          ! multiply i to Answer
      END DO
      WRITE(*,*)  N, '! = ', Answer
   END IF
END PROGRAM  Factorial
```

Nested DO-Loops



Completes inner loop first

Example : (see example10.f90)

```
INTEGER :: i, j

DO i = 1, 3
   DO j = 1, 3
     WRITE(*,*)  i*j
   END DO
END DO
```

Array Fundamentals

An array is a collection of data of the same type. Arrays can be 1d, 2d, or 3d.

Array elements are indexed or subscripted, just like x1, x2, ..., xn in mathematics. Or x(1,2,3,....n)

```
array(smaller-integer : larger-integer)
```

Declaring an Array

```
type, DIMENSION( extent ) :: name-1, name-2, ..., name-n
```

Example :

```
REAL, DIMENSION(-1:1)      :: a, Sum
INTEGER, DIMENSION(0:100) :: InputData
```

Array Input/Output

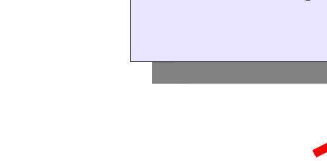
```
INTEGER, DIMENSION(1:10) :: x
INTEGER                  :: n, i

READ(*,*)  n
DO i = 1, n
   READ(*,*)  x(i)
END DO
```

Above representation is same as the following

```
INTEGER,DIMENSION(1:10) :: x
INTEGER :: n, i

READ(*,*)  n
READ(*,*)  (x(i), i=1, n)
```



Implied DO-loop

Array Input/Output

```
INTEGER, DIMENSION(1:10) :: x
INTEGER                  :: n, i


DO i = 1, 4
   WRITE(*,*)  x(i), y(i)
END DO
```

```
WRITE(*,*)  (x(i), y(i), i=1, n)
```

Using Arrays in Computation

```
INTEGER, PARAMETER                    :: LOWER = -100, UPPER = 100
INTEGER, DIMENSION(LOWER:UPPER) :: a
INTEGER                                    :: i

DO i = LOWER, UPPER
   a(i) = 0
END DO
```

```
REAL, PARAMETER                                :: MAX_SIZE = 100
REAL, DIMENSION(-MAX_SIZE:MAX_SIZE) :: DataArray
REAL                                                :: Sum
INTEGER                                           :: m, n, k

READ(*,*)  m, n
Sum = 0.0
DO k = m, n
   Sum = Sum + DataArray(k)
END DO
```

Multi Dimensional Arrays

Examples :

```
INTEGER,DIMENSION(1:10, 1:10)   :: x
REAL,   DIMENSION(1:100, 1:100) :: data
REAL,   DIMENSION(-10:10, 1:50) :: SUM
```

In computation,

```
DO i = 1, 100
   DO j = 1, 200
      matrix(i, j) = i*j
      WRITE(*,*) matrix(i, j)
   END DO
END DO
```

Functions

In addition to intrinsic functions, Fortran allows you to design your own functions. Returns only one value.

A Fortran function subprogram, has the following syntax:

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

Example :

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

How to use Functions in a Program

Example :

(example11.f90)

```
PROGRAM  Avg

   IMPLICIT  NONE
   REAL :: a, b, c, Mean
   READ(*,*)  a, b, c
   Mean = Average(a, b, c)
   WRITE(*,*) a, b, c, Mean

CONTAINS

   REAL FUNCTION  Average(a, b, c)
      IMPLICIT  NONE
      REAL, INTENT(IN) :: a, b, c
      Average = (a + b + c) / 3.0
   END FUNCTION  Average

END PROGRAM  Avg
```

Subroutines

Returns more than one values

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

INTENT(IN) : It indicates that an argument will receives some input from outside of the function and its value will not, actually cannot, be changed within the function.

INTENT(OUT) : is supposed to hold a computation result so that its value can be passed "out".

Example:

```
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
```

How to use subroutine in a program?

The subroutines are called in a program.

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

Example :

```
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
```

Example: Computing Means - Revisited (Again)

```
! ------------------------------------------------------
!    This program contains one subroutine for computing the
! arithmetic, geometric and harmonic means of three REALs.
! ------------------------------------------------------
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():
!    This subroutine receives three REAL values and computes
! their arithmetic, geometric, and harmonic means.
! ------------------------------------------------------
   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
```

(See: example12.f90)

Formatted Input and Output

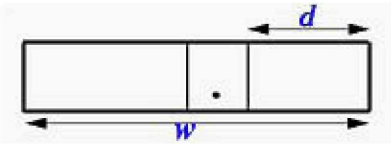
So far, we have seen *list-directed* or *free-format* I/O. Easy to use. Let us see formatted I/O.

```
READ(*,'(2I5,F10.2)')  ... variables ...
READ(*,"(5F10.2)")     ... variables ...
WRITE(*,'(A,I5)')    ... variable and expressions ...
WRITE(*,"(10F5.2)")  ... variable and expressions ...
```

'*' means standard output is screen.

Can be replaced by filename. E.g. '**10**' means output file '**fort.10**' will be created.

rFw.d



Real out put: F Descriptor

r = repetition indicator
F = Real
w = width
. = decimal point
d = digits after decimal points

I Descriptor = rIw.m (integer)
E Descriptor = rEw.d (exponential)
L Descriptor = rLw (logical)
A Descriptor = rAw (character)

Example :

There are two **REAL** variables **a** and **b** with values 123.345 and -123.345, respectively. In the following table, the **WRITE** statements are shown in the left and their corresponding output, all using five positions, are shown in the right.

```
REAL :: a = 123.345, b = -123.345
```

1	WRITE(*,"(F10.0)") a							1	2	3	.
2	WRITE(*,"(F10.1)") a						1	2	3	.	3
3	WRITE(*,"(F10.2)") a					1	2	3	.	3	5
4	WRITE(*,"(F10.3)") a				1	2	3	.	3	4	5
5	WRITE(*,"(F10.4)") a			1	2	3	.	3	4	5	0
6	WRITE(*,"(F10.5)") a		1	2	3	.	3	4	5	0	0
7	WRITE(*,"(F10.6)") a	1	2	3	.	3	4	5	0	0	0
8	WRITE(*,"(F10.7)") a	*	*	*	*	*	*	*	*	*	*
9	WRITE(*,"(F10.4)") b		-	1	2	3	.	3	4	5	0
10	WRITE(*,"(F10.5)") b	-	1	2	3	.	3	4	5	0	0
11	WRITE(*,"(F10.6)") b	*	*	*	*	*	*	*	*	*	*
		1	2	3	4	5	6	7	8	9	10

Printing INTEGERs and REALs with the I and F Descriptors

Example : (see example13.f90)

```
PROGRAM  Squares_and_Roots
   IMPLICIT  NONE
   INTEGER, PARAMETER :: MAXIMUM = 10
   INTEGER            :: i
   CHARACTER(LEN=30)  :: Format
   Format = "(3I6, 2F12.7)"
   DO i = 1, MAXIMUM
      WRITE(*,Format)  i, i*i, i*i*i, SQRT(REAL(i)), &
                       SQRT(SQRT(REAL(i)))
   END DO
END PROGRAM  Squares_and_Roots
```

Continuation on the line below

Output :

```
     1     1     1   1.0000000   1.0000000
     2     4     8   1.4142135   1.1892071
     3     9    27   1.7320508   1.3160740
     4    16    64   2.0000000   1.4142135
     5    25   125   2.2360680   1.4953488
     6    36   216   2.4494898   1.5650846
     7    49   343   2.6457512   1.6265765
     8    64   512   2.8284271   1.6817929
     9    81   729   3.0000000   1.7320508
    10   100  1000   3.1622777   1.7782794
```