Program Structure and Format

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

Comments

Comments should be used liberally to improve readability. The following are the rules for making comments:

1. **All characters following a exclamation mark !, except in a character string, are commentary, and are ignored by the compiler.**

```
PROGRAM  TestComment1
   ..........
   READ(*,*)  Year   ! read in the value of Year
   ..........
   Year = Year + 1   ! add 1 to Year
   ..........
END PROGRAM  TestComment1
```

2. **An entire line may be a comment**

```
! This is a comment line
!
PROGRAM  TestComment2
   .........
   ! This is a comment line in the middle of a program
   .........
END PROGRAM  TestComment2
```

3. **A blank line is also interpreted as a comment line**

```
PROGRAM  TestComment3
   ..........
   READ(*,*)  Count

! The above blank line is a comment line
   WRITE(*,*)  Count + 2
END PROGRAM  TestComment3
```

Continuation Lines

If a statement is too long to fit on a line, it can be continued with the following methods:

1. **If a line is ended with an ampersand &, it will be continued on the next line.**

```
A = 174.5 * Year   &          A = 174.5 * Year  + Count / 100
    + Count / 100
```

2. **Continuation is normally to the first character of the next <u>non-comment</u> line**

```
A = 174.5 * Year   &           A = 174.5 * Year  + Count / 100
!  this is a comment line
    + Count / 100
```

3. **If the first non-blank character of the continuation line is &, continuation is to the first character after the &:**

```
A = 174.5 + ThisIsALong&        A = 174.5 + ThisIsALongName
     &Name
```

There should be no spaces between the last character and the & on the first line.

```
A = 174.5 + ThisIsALong   &     A = 174.5 + ThisIsALong   Name
     &Name
```

An Example

```
! Calculates number of accumulated
! AIDS cases in USA

PROGRAM  AidsCases
   IMPLICIT NONE          ! this is required
   INTEGER  :: Year
   REAL     :: Ans

   READ(*,*)  Year        ! Read in a value for Year
   Ans = 174.6 * (Year - 1981.2) ** 3
   WRITE(*,*) 'AIDS cases by year ', Year, ':', Ans
END PROGRAM AidsCases     ! end of program
```

FORTRAN Alphabets

- **Letters:** Upper and lower case letters:

  A B C D E F G H I J K L M
  N O P Q R S T U V W X Y Z
  a b c d e f g h i j k l m
  n o p q r s t u v w x y z

- **Digits:**

  0 1 2 3 4 5 6 7 8 9

- **Special Characters:**

  space
  ' " ( ) * + - / : = _
  ! & $; < > % ? , .

Data Types and Constants

1. `INTEGER` – a string of digits with an optional sign
   - **Correct:** `0`, `-345`, `768`, `+12345`
   - **Incorrect:**
     - `1,234` (comma not allowed)
     - `12.0` (no decimal point),
     - `--4` and `++3` (too many signs),
     - `5-` and `5+` (sign must precede the string of digits)
2. `REAL` – may be in scientific (exponential) form
   - **Correct:**
     - Decimal Form: `123.45`, `.123`, `123.`, `-0.12`, `+12.0`
     - Exponential Form: `12.34E3`, `12.3E+3`, `-1.2E-3`, `45.67E0`, `123E4`, `-123E3`. (`E-3` is equal to $\times 10^{-3}$)
   - **Incorrect:**
     - `12,345.99` (no comma)
     - `65` (this is an `INTEGER`)

3. `CHARACTER`: a string of characters enclosed between apostrophes or double quotes:

- **Correct:**
  - `'John'` and `"John"` (content: `John` and length: 4),
  - `' '` and `" "` (content: space and length: 1),
  - `'John Dow #2'` and `"John Dow #2"` (content: `John Dow #2` and length: 11)
- **Special Case:**
  - `'Lori''s apple'` (content: `Lori's apple` and length: 12), or equivalently, `"Lori's apple"`
  - `'Don''t forget Jim''s book'` (content: `Don't forget Jim's book` and length: 23), or equivalently, `"Don't forget Jim's book"`
- **Incorrect:**
  - `'you and me` (missing a closing apostrophe – **a common error**)
  - `'Tech's seminar'` – **another common error**.
  - `'Have a nice day"` – don't mix appostrophes and double quotes

Identifiers

A FORTRAN identifier consists of

1. No more than 31 characters
2. The first character must be a letter
3. The remaining, if any, may be letters, digits or underscores
   - **Correct:**
     - `MTU`, `MI`, `John`, `Count`
     - `I`, `X`
     - `I1025`, `a1b2C3`, `X9900g`
     - `R2_D2`, `R2D2_`, `A__` (yes, this one is correct)
   - Incorrect:
     - `M.T.U.`, `R2-D2` – only letters, digits and underscore are allowed.
     - `6feet` – cannot start with a digit
     - `_System` – the first character must be a letter
4. **Please use meaningful identifiers**
   - **Good:** `Total`, `Rate`, `Length`
   - **Not so good:** `ThisIsALongIdentifierName`, `X321`, `A_B_012`, `OPQ`

Declare the Type of an Identifier

```
PROGRAM   program-name
IMPLICIT  NONE
INTEGER          ::  name, name, name, ...
REAL             ::  name, name, name, ...
CHARACTER        ::  name, name, name, ...
CHARACTER(LEN=n) ::  name, name, name, ...
CHARACTER(n)     ::  name, name, name, ...
          ................
END PROGRAM program-name
```

1. Variables `ZIP`, `Mean`, and `Total` are of type `INTEGER`:

   ```
   INTEGER  :: ZIP, Mean, Total
   ```

2. Variables `Average`, `error`, `sum` and `ZAP` are of type `REAL`:

   ```
   REAL  :: Average, error, sum, ZAP
   ```

3. Variables `Name` and `Street` can hold a character string up to 15 characters

   ```
   CHARACTER(LEN = 15) :: Name, Street
   ```

   The following is an equivalent form:

   ```
   CHARACTER(15) :: Name, Street
   ```

4. Variables `letter` and `digit` can only hold one character:

   ```
   CHARACTER  :: letter, digit
   ```

5. Variables `City` and `BOX` can hold a character string up to 10 characters, `Nation` can hold a character string up to 20 characters and `bug` can only hold one character.

   ```
   CHARACTER(LEN = 10) :: City, Nation*20, BOX, bug*1
   ```

Giving Constants Names
The `PARAMETER` Attribute

Syntax

`type, PARAMETER ::` *name* = *value*, *name* = *value*, ...

Examples

- ```
  INTEGER, PARAMETER ::  Limit = 100
  REAL, PARAMETER    ::  PI = 3.14159, TWOPI = 2.0 * PI
  CHARACTER(LEN = 4), PARAMETER ::  Name = 'Smith', city = "LA"
  CHARACTER(*), PARAMETER ::  NAME = 'Smith', CITY = "LA"
  ```

  `CHARACTER(*)` is an **assumed length specifier**. That is, the length of a constant is determined by the lengths of the string.

- ```
  INTEGER, PARAMETER :: Count = 10
  REAL, PARAMETER    :: degree = 37.5, total = Count * degree
  CHARACTER(*), PARAMETER :: FirstName = 'John', MiddleName = 'F'
  CHARACTER(*), PARAMETER :: LastName = 'Kennedy'
  ```

Important Notes

- **If string length is longer, truncation to the right will happen:** `Name = 'Smit'`
- **If string length is shorter, spaces will be added to the right:** `city = 'LA  '`

Variable Initialization

The way of initializing a variable is very similar to the use of `PARAMETER` attribute. More precisely, to initial variable with the value of an expression, do the following:

1. add an equal sign = to the right of the variable name
2. to the right of the equal sign, write an expression.

Initializing a variable is only done exactly <u>once</u> when the computer loads the program into memory for execution.

1. Initializes variables `Offset` to 0.1, `Length` to 10.0, and `tolerance` to 1.E-7.

   ```
   REAL :: Offset = 0.1, Length = 10.0, tolerance = 1.E-7
   ```

2. Initializes variables `State1` to `"MI"`, `State2` to `"MN"`, and `State3` to `"MD"`.

   ```
   CHARACTER(LEN=2) :: State1 = "MI", State2 = "MN", State3 = "MD"
   ```

3. The following defines three named integer constants using `PARAMETER` and initializes `Pay` and `Received` to 4350=10*435 and 15 =3*5.

   ```
   INTEGER, PARAMETER :: Quantity = 10, Amount = 435, Period = 3
   INTEGER            :: Pay = Quantity*Amount, Received = Period+5
   ```

4. The following example contains a mistake.

   ```
   INTEGER, PARAMETER :: Quantity = 10, Amount = 435
   INTEGER            :: Pay = Quantity*Amount, Received = Period+5
   INTEGER, PARAMETER :: Period = 3
   ```

Operators and Their Priority

Type	*Operators*	*Associativity*
Arithmetic	`**`	right-to-left
	`*` `/`	left-to-right
	`+` `-`	left-to-right
Relational	`<` `<=`	left-to-right
	`>` `>=`	
	`==` `/=`	
Logical	`.NOT.`	right-to-left
	`.AND.`	left-to-right
	`.OR.`	left-to-right
	`.EQV.` `.NEQV.`	left-to-right

Important Note

`A**B**C` is equal to `A**(B**C)` rather than `(A**B)**C` since `**` is **right** associative!

Single Mode Arithmetic Expressions

1. The result is 4 rather than 4.444444 since the operands are all integers.

```
2 * 4 * 5 / 3 ** 2
     --> [2 * 4] * 5 / 3 ** 2
     --> 8 * 5 / 3 ** 2
     --> [8 * 5] / 3 ** 2
     --> 40 / 3 ** 2
     --> 40 / [3 ** 2]
     --> 40 / 9
     --> 4
```

2. As in mathematics, subexpressions in parenthesis must be evaluated first.

```
100 + (1 + 250 / 100) ** 3
     --> 100 + (1 + [250 / 100]) ** 3
     --> 100 + (1 + 2) ** 3
     --> 100 + ([1 + 2]) ** 3
     --> 100 + 3 ** 3
     --> 100 + [3 ** 3]
     --> 100 + 27
     --> 127
```

3. In the following example, `x**0.25` is equivalent to $\sqrt[4]{x}$ In general, taking the k-th root of `x` can be done with `x**(1.0/k)`.

```
1.0 + 2.0 * 3.0 / ( 6.0*6.0 + 5.0*44.0) ** 0.25
     --> 1.0 + [2.0 * 3.0] / (6.0*6.0 + 5.0*44.0) ** 0.25
     --> 1.0 + 6.0 / (6.0*6.0 + 5.0*55.0) ** 0.25
     --> 1.0 + 6.0 / ([6.0*6.0] + 5.0*44.0) ** 0.25
     --> 1.0 + 6.0 / (36.0 + 5.0*44.0) ** 0.25
     --> 1.0 + 6.0 / (36.0 + [5.0*44.0]) ** 0.25
     --> 1.0 + 6.0 / (36.0 + 220.0) ** 0.25
     --> 1.0 + 6.0 / ([36.0 + 220.0]) ** 0.25
     --> 1.0 + 6.0 / 256.0 ** 0.25
     --> 1.0 + 6.0 / [256.0 ** 0.25]
     --> 1.0 + 6.0 / 4.0
     --> 1.0 + [6.0 / 4.0]
     --> 1.0 + 1.5
     --> 2.5
```

Mixed Mode Arithmetic Expressions

Operation	*Conversion*	*Result*
INTEGER $\otimes$ REAL	REAL $\otimes$ REAL	REAL

- `6.0 ** 2` is not converted to `6.0 ** 2.0`. It is computed as `6.0 * 6.0`.

```
5 * (11.0 - 5) ** 2 / 4 + 9
     --> 5 * (11.0 - {5}) ** 2 / 4 + 9
     --> 5 * (11.0 - 5.0) ** 2 / 4 + 9
     --> 5 * ([11.0 - 5.0]) ** 2 / 4 + 9
     --> 5 * 6.0 ** 2 / 4 + 9
     --> 5 * [6.0 ** 2] / 4 + 9
     --> 5 * 36.0 / 4 + 9
     --> {5} * 36.0 / 4 + 9
     --> 5.0 * 36.0 / 4 + 9
     --> [5.0 * 36.0] / 4 + 9
     --> 180.0 / 4 + 9
     --> 180.0 / {4} + 9
     --> 180.0 / 4.0 + 9
     --> [180.0 / 4.0] + 9
     --> 45.0 + 9
     --> 45.0 + {9}
     --> 45.0 + 9.0
     --> 54.0
```

- In the following, `25.0 ** 1` is not converted, and `1 / 3` is zero.

```
25.0 ** 1 / 2 * 3.5 ** (1 / 3)
     --> [25.0 ** 1] / 2 * 3.5 ** (1 / 3)
     --> 25.0 / 2 * 3.5 ** (1 / 3)
     --> 25.0 / {2} * 3.5 ** (1 / 3)
     --> 25.0 / 2.0 * 3.5 ** (1 / 3)
     --> 12.5 * 3.5 ** (1 / 3)
     --> 12.5 * 3.5 ** ([1 / 3])
     --> 12.5 * 3.5 ** 0
     --> 12.5 * [3.5 ** 0]
     --> 12.5 * 1.0
     --> 12.5
```

Assignment Statement

Syntax

$$\text{variable} = \text{expression}$$

The result will be <u>converted</u> to the variable's type

1. The following computes `Unit * Amount` and saves the answer to variable `Total` (= `5 * 100 = 500`)

```
INTEGER :: Total, Amount, Unit

Unit   = 5
Amount = 100.99
Total  = Unit * Amount
```

2. In the following, `PI` is a `PARAMETER` and is an alias of 3.1415926. The first assignment statement puts 5 into integer variable `Radius`. The second assignment computes `(Radius ** 2) * PI` and saves the result to real variable `Area`.

```
REAL, PARAMETER :: PI = 3.1415926
REAL            :: Area
INTEGER         :: Radius

Radius = 5
Area   = (Radius ** 2) * PI
```

3. The initial value of integer variable `Count` is zero. The first assignment adds 1 to it, yielding a new result 1 =0+1. The second assignment adds 3 to `Count`, yielding a result of $4 = 1 + 3$.

```
INTEGER  ::  Counter = 0

Counter = Counter + 1
Counter = Counter + 3
```

4. The following three assignments swap the values of `A` and `B`. That is, after these assignments are done, `A` and `B` have values 5 and 3.

```
INTEGER  :: A = 3, B = 5, C

C = A
A = B
B = C
```

Common Functions

Function	*Description*	*Arg. Type*	*Return Type*
`ABS(x)`	absolute value of `x`	`INTEGER` `REAL`	`INTEGER` `REAL`
`SQRT(x)`	square root of `x`	`REAL`	`REAL`
`SIN(x)`	Sine of `x` radians	`REAL`	`REAL`
`COS(x)`	Cosine of `x` radians	`REAL`	`REAL`
`TAN(x)`	tangent of `x` radians	`REAL`	`REAL`
`EXP(x)`	exp(`x`)	`REAL`	`REAL`
`LOG(x)`	ln(`x`)	`REAL`	`REAL`

Conversion Functions

Function	*Description*	*Arg. Type*	*Return Type*
`INT(x)`	integer part of `x`	`REAL`	`INTEGER`
`NINT(x)`	nearest integer to `x`	`REAL`	`INTEGER`
`FLOOR(x)`	greatest integer $\leq$ `x`	`REAL`	`INTEGER`
`FRACTION(x)`	fractional part of `x`	`REAL`	`REAL`
`REAL(x)`	convert `x` to `REAL`	`INTEGER`	`REAL`
`MAX(x1, .., xn)`	max of `x1`, . . ., `xn`	`INTEGER` `REAL`	`INTEGER` `REAL`
`MIN(x1, .., xn)`	min of `x1`, . . ., `xn`	`INTEGER` `REAL`	`INTEGER` `REAL`
`MOD(x,y)`	`x - INT(x/y)*y`	`INTEGER` `REAL`	`INTEGER` `REAL`

Free Format Output
The `WRITE` Statement

Syntax

```
WRITE(*,*)
WRITE(*,*)  expr-1, expr-2, .... , expr-n
```

1. `WRITE(*,*)`
   Just output a blank line
2. `WRITE(*,*) exp-1, exp-2, ... , exp-n`
   (a) Each expression will be evaluated and display on screen.
   (b) After all expressions have been displayed, **advance to next line**.
   (c) Thus, a `WRITE` produces **at least** one line.
3. The mean of `*`
   (a) The first `*` means the output is sent to screen. Technically the screen is referred to as **`stdout`**, standard out, in UNIX.
   (b) The second `*` means the `WRITE` is a free format.
4. The actual appearance of the output depends on your compiler.

Example 1

```
PROGRAM     FORTRAN_Traps
   IMPLICIT    NONE

   INTEGER, PARAMETER          :: A = 2, B = 2, H = 3
   INTEGER, PARAMETER          :: O = 4, P = 6
   CHARACTER(LEN=5), PARAMETER :: M = 'Smith', N = 'TEXAS'
   CHARACTER(LEN=4), PARAMETER :: X = 'Smith'
   CHARACTER(LEN=6), PARAMETER :: Y = 'TEXAS'

!  The exponential trap

   WRITE(*,*)   "First, the exponential trap:"
   WRITE(*,*)   A, ' ** ', B, ' ** ', H, ' = ', A**B**H
   WRITE(*,*)   '( ', A, ' ** ', B, ' ) **', H, ' = ', (A**B)**H
   WRITE(*,*)   A, ' ** ( ', B, ' ** ', H, ' ) = ', A**(B**H)
   WRITE(*,*)

!  The integer division trap.  Intrinsic function REAL() converts
!  an integer to a real number

   WRITE(*,*)   "Second, the integer division trap:"
   WRITE(*,*)
   WRITE(*,*)   O, ' / ', P, ' = ', O/P
   WRITE(*,*)   'REAL( ', O, ' ) / ', P, ' = ', REAL(O)/P
   WRITE(*,*)   O, ' / REAL( ', P, ' ) = ', O/REAL(P)
   WRITE(*,*)

!  The string truncation trap

   WRITE(*,*)   "Third, the string truncation trap:"
   WRITE(*,*)   'IS ', M, ' STILL IN ', N, '?'
   WRITE(*,*)   'IS ', X, ' STILL IN ', Y, '?'

END PROGRAM  FORTRAN_Traps
```

Example 2

Compute the arithmetic, geometric, and harmonic means of three real numbers:

$$\begin{aligned} \text{arithmetic mean} &= \frac{1}{3}(a+b+c) \\ \text{geometric mean} &= (a \cdot b \cdot c)^{1/3} \\ \text{harmonic mean} &= \frac{3}{\frac{1}{a}+\frac{1}{b}+\frac{1}{c}} \end{aligned}$$

```
! -------------------------------------------------------
!   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
```

Free Format Input
The `READ` Statement

Syntax

```
READ(*,*)  var-1, var-2, .... , var-n
```

1. A new line of data item is processed each time a `READ` statement is executed.
2. In the input, consecutive data items must be separated with comma or by one or more spaces.
3. Relationship between the number of variables, n, in `READ` and the number of data items, d, in input:
4. Cannot supply `REAL` data items to an `INTEGER` variable.
5. Character string data can only be read into a `CHARACTER` variable.
   (a) $n = d$: The variables receive the corresponding values in the input. This is the perfect case.
   (b) $n < d$: The variables in `READ` will receive values from the input and the remaining data will be ignored.
   (c) $n > d$: After consuming all data items in the input, successive input lines will be processed until all variables in a `READ` receive their value.

Example 1

The roots of a quadratic equation $ax^2 + bx + c = 0$ can be expressed as follows:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

In order to use the square root, $b^2 - 4ac$ must be positive.

```
PROGRAM  QuadraticEquation

   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
```

Example 2

Given a parabola whose base length is $2b$ and height is h, the length of the parabola can be computed as

$$\sqrt{4h^2 + b^2} + \frac{b^2}{2h} \ln\left(\frac{2h + \sqrt{4h^2 + b^2}}{b}\right).$$

```
PROGRAM  ParabolaLength
   IMPLICIT  NONE

   REAL  :: Height, Base, Length
   REAL  :: temp, t

   WRITE(*,*)  'Height of a parabola : '
   READ(*,*)   Height

   WRITE(*,*)  'Base of a parabola   : '
   READ(*,*)   Base

! ... temp and t are two temporary variables

   t      = 2.0 * Height
   temp   = SQRT(t**2 + Base**2)
   Length = temp + Base**2/t*LOG((t + temp)/Base)

   WRITE(*,*)
   WRITE(*,*)  'Height = ', Height
   WRITE(*,*)  'Base   = ', Base
   WRITE(*,*)  'Length = ', Length

END PROGRAM  ParabolaLength
```