Fortran Basics

I don't know what the programming language of the year 2000 will look like, but I know it will be called FORTRAN.

Charles Anthony Richard Hoare

F Program Structure

- A Fortran program has the following form:
  - *program-name* is the name of that program
  - *specification-part*, *execution-part*, and *subprogram-part* are optional.
  - Although `IMPLICIT NONE` is also optional, this is ***required*** in this course to write safe programs.

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

Program Comments

- Comments start with a **!**
- Everything following **!** will be ignored
- This is similar to **//** in C/C++

```
! This is an example
!

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

Continuation Lines

- Fortran is not completely format-free!
- A statement must starts with a new line.
- If a statement is too long to fit on one line, it has to be *continued*.
- The continuation character is **&**, which is not part of the statement.

```
Total = Total + &
           Amount * Payments
! Total = Total + Amount*Payments

PROGRAM  &
   ContinuationLine
! PROGRAM ContinuationLine
```

Alphabets

- **Fortran alphabets include the following:**
  - **Upper and lower cases letters**
  - **Digits**
  - **Special characters**

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

Constants: 1/6

- **A Fortran constant may be an integer, real, logical, complex, and character string.**
- **We will not discuss complex constants.**
- **An integer constant is a string of digits with an optional sign: `12345`, `-345`, `+789`, `+0`.**

Constants cont.

- **A real constant has two forms, decimal and exponential:**
  - **In the decimal form, a real constant is a string of digits with exactly one decimal point. A real constant may include an optional sign. Example: `2.45`, `.13`, `13.`, `-0.12`, `-.12`.**

Constants:

- A **real constant** has two forms, **decimal** and **exponential**:
  - **In the exponential form, a real constant starts with an integer/real, followed by a `E`/`e`, followed by an integer (*i.e.*, the exponent). Examples:**
    - `12E3` **(12×10^3)**, `-12e3` **(-12×10^3)**, `3.45E-8` **(3.45×10^{-8})**, `-3.45e-8` **(-3.45×10^{-8}).**
    - `0E0` **($0\times10^0=0$).** `12.34-5` **is wrong!**

Constants:

- **A logical constant is either `.TRUE.` or `.FALSE.`**
- **Note that the periods surrounding `TRUE` and `FALSE` are required!**

Constants:

- **A character string or character constant is a string of characters enclosed between two double quotes or two single quotes. Examples:** `"abc"`, `'John Dow'`, `"#$%^"`, **and** `'()()'`.
- **The content of a character string consists of all characters between the quotes. Example: The content of** `'John Dow'` **is** `John Dow`.
- **The length of a string is the number of characters between the quotes. The length of** `'John Dow'` **is 8, space included.**

Constants cont. :

- A string has length zero (*i.e.*, no content) is an empty string.
- If single (or double) quotes are used in a string, then use double (or single) quotes as delimiters. Examples: `"Adam's cat"` and `'I said "go away"'`.
- Two consecutive quotes are treated as one!

`'Lori''s Apple'` is `Lori's Apple`

`"double quote"""` is `double quote"`

`` `abc''def"x''y' `` is `abc'def"x'y`

`"abc""def'x""v"` is `abc"def'x"v`

Identifiers:

- **A Fortran identifier can have no more than 31 characters.**
- **The first one must be a letter. The remaining characters, if any, may be letters, digits, or underscores.**
- ***Fortran identifiers are CASE INSENSITIVE.***
- **Examples: `A`, `Name`, `toTAL123`, `System_`, `myFile_01`, `my_1st_F _program_X_`.**
- **Identifiers `Name`, `nAmE`, `naME` and `NamE` are the same.**

Identifiers:

- **Unlike Java, C, C++, etc, *Fortran does not have reserved words*. This means one may use Fortran keywords as identifiers.**
- **Therefore, `PROGRAM`, `end`, `IF`, `then`, `DO`, etc may be used as identifiers. Fortran compilers are able to recognize keywords from their "positions" in a statement.**
- **Yes, `end = program + if/(goto – while)` is legal!**
- **However, avoid the use of Fortran keywords as identifiers to minimize confusion.**

Declarations:

- **Fortran uses the following for variable declarations, where `type-specifier` is one of the following keywords: `INTEGER`, `REAL`, `LOGICAL`, `COMPLEX` and `CHARACTER`, and `list` is a sequence of identifiers separated by commas.**

```
type-specifier :: list
```

- **Examples:**

```
INTEGER :: Zip, Total, counter
REAL    :: AVERAGE, x, Difference
LOGICAL :: Condition, OK
COMPLEX :: Conjugate
```

Declarations:

- **Character variables require additional information, the *string length*:**
  - **Keyword `CHARACTER` must be followed by a length attribute `(LEN = l)` , where *l* is the length of the string.**
  - **If the length of a string is 1, one may use**

    `CHARACTER` without length attribute.
  - **Other length attributes will be discussed later.**

Declarations:

- **Examples:**
  - `CHARACTER(LEN=20) :: Answer, Quote`

    Variables `Answer` and `Quote` can hold strings up to 20 characters.
  - `CHARACTER(20) :: Answer, Quote` **is the same as above.**
  - `CHARACTER :: Keypress` **means variable `Keypress` can only hold <u>*ONE*</u> character (*i.e.*, length 1).**

The `PARAMETER` Attribute:

- A `PARAMETER` identifier is a name whose value cannot be modified. In other words, it is a *named constant (final, const)*.
- The `PARAMETER` attribute is used after the type keyword.
- Each identifier is followed by a `=` and followed by a value for that identifier.

```
INTEGER, PARAMETER :: MAXIMUM = 10
REAL, PARAMETER    :: PI = 3.1415926, E = 2.17828
LOGICAL, PARAMETER :: TRUE = .true., FALSE = .false.
```

The `PARAMETER` Attribute:

- Since `CHARACTER` identifiers have a length attribute, it is a little more complex when used with `PARAMETER`.
- Use `(LEN = *)` if one does not want to count the number of characters in a `PARAMETER` character string, where `= *` means the length of this string is determined elsewhere.

```
CHARACTER(LEN=3), PARAMETER :: YES = "yes"   ! Len = 3
CHARACTER(LEN=2), PARAMETER :: NO = "no"     ! Len = 2
CHARACTER(LEN=*), PARAMETER :: &
               PROMPT = "What do you want?" ! Len = 17
```

The **PARAMETER** Attribute:

- **Since Fortran strings are of *fixed* length, one must remember the following:**
  - **If a string is longer than the `PARAMETER` length, the right end is truncated.**
  - **If a string is shorter than the `PARAMETER` length, spaces will be added to the right.**

```
CHARACTER(LEN=4), PARAMETER :: ABC = "abcdef"
CHARACTER(LEN=4), PARAMETER :: XYZ = "xy"
```

ABC =

a	b	c	d

XYZ =

x	y		

The `PARAMETER` Attribute:

- **By convention, `PARAMETER` identifiers use all upper cases. However, this is not mandatory.**
- **For maximum flexibility, constants other than 0 and 1 should be `PARAMETER`ized.**
- **A `PARAMETER` is an alias of a value and is *not* a variable. Hence, one cannot modify the content of a `PARAMETER` identifier.**
- **One can may use a `PARAMETER` identifier anywhere in a program. It is equivalent to replacing the identifier with its value.**
- **The value part can use expressions.**

Variable Initialization:

- **A variable receives its value with**
  - ***Initialization*: It is done once before the program runs.**
  - ***Assignment*: It is done when the program executes an assignment statement.**
  - ***Input*: It is done with a `READ` statement.**

Variable Initialization:

- **Variable initialization is very similar to what we learned with `PARAMETER`.**
- **A variable name is followed by a `=`, followed by an expression in which all identifiers must be constants or `PARAMETER`s defined *previously*.**
- **Using an un-initialized variable may cause un-expected, sometimes disastrous results.**

```
REAL :: Offset = 0.1, Length = 10.0, tolerance = 1.E-7
CHARACTER(LEN=2) :: State1 = "MI", State2 = "MD"
INTEGER, PARAMETER :: Quantity = 10, Amount = 435
INTEGER, PARAMETER :: Period = 3
INTEGER :: Pay = Quantity*Amount, Received = Period+5
```

Arithmetic Operators

- **There are four types of operators in Fortran : arithmetic, relational, logical and character.**
- **The following shows the first three types:**

Type	*Operator*						*Associativity*
Arithmetic	`**`						***right to left***
	`*`			`/`			**left to right**
	`+`			`-`			**left to right**
Relational	`<`	`<=`	`>`	`>=`	`==`	`/=`	**none**
Logical	`.NOT.`						***right to left***
	`.AND.`						**left to right**
	`.OR.`						**left to right**
	`.EQV.`			`.NEQV.`			**left to right**

The Assignment Statement:

- The assignment statement has a form of

  `variable = expression`

- If the type of `variable` and `expression` are identical, the result is saved to `variable`.
- If the type of `variable` and `expression` are not identical, the result of `expression` is converted to the type of `variable`.
- If `expression` is `REAL` and `variable` is `INTEGER`, the result is truncated.

The Assignment Statement:

- The left example uses an initialized variable `Unit`, and the right uses a `PARAMETER PI`.

```
INTEGER :: Total, Amount
INTEGER :: Unit = 5

Amount = 100.99
Total = Unit * Amount
```

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

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

This one is equivalent to `Radius ** 2 * PI`

Fortran Intrinsic Functions:

- **Fortran provides many commonly used functions, referred to as *intrinsic functions*.**
- **To use an intrinsic function, we need to know:**
  - **Name and meaning of the function (*e.g.*, `SQRT()` for square root)**
  - **Number of arguments**
  - **The type and range of each argument (*e.g.*, the argument of `SQRT()` must be non-negative)**
  - **The type of the returned function value.**

Fortran Intrinsic Functions:

- **Some mathematical functions:**

Function	*Meaning*	*Arg. Type*	*Return Type*
`ABS(x)`	absolute value of `x`	`INTEGER`	`INTEGER`
		`REAL`	`REAL`
`SQRT(x)`	square root of `x`	`REAL`	`REAL`
`SIN(x)`	sine of `x` radian	`REAL`	`REAL`
`COS(x)`	cosine of `x` radian	`REAL`	`REAL`
`TAN(x)`	tangent of `x` radian	`REAL`	`REAL`
`ASIN(x)`	arc sine of `x`	`REAL`	`REAL`
`ACOS(x)`	arc cosine of `x`	`REAL`	`REAL`
`ATAN(x)`	arc tangent of `x`	`REAL`	`REAL`
`EXP(x)`	exponential e^{x}	`REAL`	`REAL`
`LOG(x)`	natural logarithm of `x`	`REAL`	`REAL`

`LOG10(x)` is the common logarithm of `x`!

Fortran Intrinsic Functions: 3/4

- **Some conversion functions:**

Function	*Meaning*	*Arg. Type*	*Return Type*
`INT(x)`	truncate to integer part `x`	`REAL`	`INTEGER`
`NINT(x)`	round nearest integer to `x`	`REAL`	`INTEGER`
`FLOOR(x)`	greatest integer less than or equal to `x`	`REAL`	`INTEGER`
`FRACTION(x)`	the fractional part of `x`	`REAL`	`REAL`
`REAL(x)`	convert `x` to `REAL`	`INTEGER`	`REAL`

Examples:

```
INT(-3.5)  ➔ -3
NINT(3.5)  ➔ 4
NINT(-3.4)  ➔ -3
FLOOR(3.6)  ➔ 3
FLOOR(-3.5)  ➔ -4
FRACTION(12.3)  ➔ 0.3
REAL(-10)  ➔ -10.0
```

Fortran Intrinsic Functions:

- **Other functions:**

Function	*Meaning*	*Arg. Type*	*Return Type*
`MAX(x1, x2, ..., xn)`	maximum of `x1`, `x2`, ... `xn`	`INTEGER`	`INTEGER`
		`REAL`	`REAL`
`MIN(x1, x2, ..., xn)`	minimum of `x1`, `x2`, ... `xn`	`INTEGER`	`INTEGER`
		`REAL`	`REAL`
`MOD(x,y)`	remainder `x - INT(x/y)*y`	`INTEGER`	`INTEGER`
		`REAL`	`REAL`

What is `IMPLICIT NONE`?

- **Fortran has an interesting tradition: all variables starting with `i`, `j`, `k`, `l`, `m` and `n`, if not declared, are of the `INTEGER` type by default.**
- **This handy feature can cause serious consequences if it is not used with care.**
- **`IMPLICIT NONE` means all names must be declared and there is no implicitly assumed `INTEGER` type.**
- **All programs in this class must use `IMPLICIT NONE`. *Points will be deducted if you do not use it*!**

List-Directed READ:

- Fortran uses the `READ(*,*)` statement to read data into variables from keyboard:

```
READ(*,*) v1, v2, …, vn
READ(*,*)
```

- The second form has a special meaning that will be discussed later.

```
INTEGER              :: Age
REAL                 :: Amount, Rate
CHARACTER(LEN=10)    :: Name

READ(*,*)  Name, Age, Rate, Amount
```

List-Directed `READ`:

- **Data Preparation Guidelines**
  - `READ(*,*)` **reads data from keyboard by default, although one may use input redirection to read from a file.**
  - **If** `READ(*,*)` **has *n* variables, there must be *n* Fortran constants.**
  - **Each constant must have the type of the corresponding variable. Integers can be read into** `REAL` **variables but not vice versa.**
  - **Data items are separated by spaces and may spread into multiple lines.**

List-Directed READ:

- *The execution of* **READ(*,*)** *<u>always starts with a new line!</u>*
- **Then, it reads each constant into the corresponding variable.**

```
CHARACTER(LEN=5)  :: Name
REAL              :: height, length
INTEGER           :: count, MaxLength

READ(*,*) Name, height, count, length, MaxLength
```

Input: `"Smith" 100.0 25 123.579 10000`

List-Directed `READ`:

- **Be careful when input items are on multiple lines.**

```
INTEGER :: I,J,K,L,M,N

READ(*,*) I, J
READ(*,*) K, L, M
READ(*,*) N
```

Input:

```
100 200
300 400 500
600
```

```
INTEGER :: I,J,K,L,M,N

READ(*,*) I, J, K
READ(*,*) L, M, N
```

ignored!

```
100   200   300   400
500   600   700   800
 0
```

`READ(*,*)` **always starts with a new line**

List-Directed READ:

- Since READ(*,*) always starts with a new line, a READ(*,*) without any variable means skipping the input line!

```
INTEGER :: P, Q, R, S

READ(*,*) P, Q
READ(*,*)
READ(*,*) R, S
```

```
100    200    300
400    500    600
700    800      0
```

List-Directed WRITE: 1/3

- **Fortran uses the `WRITE(*,*)` statement to write information to screen.**
- **`WRITE(*,*)` has two forms, where `exp1`, `exp2`, …, `expn` are expressions**

```
WRITE(*,*) exp1, exp2, …, expn
WRITE(*,*)
```

- **`WRITE(*,*)` evaluates the result of each expression and prints it on screen.**
- **`WRITE(*,*)` *always starts with a new line!***

List-Directed WRITE: 2/3

- Here is a simple example:

means length is determined by actual count

```
INTEGER :: Target
REAL :: Angle, Distance
CHARACTER(LEN=*), PARAMETER ::          &
  Time = "The time to hit target ",     &
  IS = " is ",                          &
  UNIT = " sec."

Target = 10
Angle = 20.0
Distance = 1350.0
WRITE(*,*) 'Angle = ', Angle
WRITE(*,*) 'Distance = ', Distance
WRITE(*,*)
WRITE(*,*) Time, Target, IS,          &
           Angle * Distance, UNIT
```

continuation lines

print a blank line

Output:

```
Angle =  20.0
Distance =  1350.0

The time to hit target 10 is 27000.0 sec.
```

List-Directed WRITE:

- The previous example used `LEN=*` , which means the length of a `CHARACTER` constant is determined by actual count.
- `WRITE(*,*)` without any expression advances to the next line, producing a blank one.
- A Fortran compiler will use the *best* way to print each value. Thus, indentation and alignment are difficult to achieve with `WRITE(*,*)`.
- One must use the `FORMAT` statement to produce good looking output.

CHARACTER Operator

- Fortran uses `//` to concatenate two strings.
- If strings **A** and **B** have lengths *m* and *n*, the concatenation **A // B** is a string of length *m*+*n*.

```
CHARACTER(LEN=4)  :: John = "John", Sam = "Sam"
CHARACTER(LEN=6)  :: Lori = "Lori", Reagan = "Reagan"
CHARACTER(LEN=10) :: Ans1, Ans2, Ans3, Ans4

Ans1 = John // Lori             ! Ans1 = "JohnLori[["
Ans2 = Sam // Reagan            ! Ans2 = "Sam[Reagan"
Ans3 = Reagan // Sam            ! Ans3 = "ReaganSam["
Ans4 = Lori // Sam              ! Ans4 = "Lori[[Sam["
```

Example:

- **This program uses the `DATE_AND_TIME()` Fortran intrinsic function to retrieve the system date and system time. Then, it converts the date and time information to a readable format. This program demonstrates the use of concatenation operator // and substring.**
- **System date is a string `ccyymmdd`, where `cc` = century, `yy` = year, `mm` = month, and `dd` = day.**
- **System time is a string `hhmmss.sss`, where `hh` = hour, `mm` = minute, and `ss.sss` = second.**

Example:

- **The following shows the specification part. Note the handy way of changing string length.**

```
PROGRAM DateTime
   IMPLICIT NONE
   CHARACTER(LEN = 8)   :: DateINFO                    ! ccyymmdd
   CHARACTER(LEN = 4)   :: Year, Month*2, Day*2
   CHARACTER(LEN = 10)  :: TimeINFO, PrettyTime*12 ! hhmmss.sss
   CHARACTER(LEN = 2)   :: Hour, Minute, Second*6

   CALL DATE_AND_TIME(DateINFO, TimeINFO)
      …… other executable statements ……
END PROGRAM DateTime
```

This is a handy way of changing string length

Example:

- Decompose `DateINFO` into year, month and day. `DateINFO` has a form of `ccyymmdd`, where `cc` = century, `yy` = year, `mm` = month, and `dd` = day.

```
Year  = DateINFO(1:4)
Month = DateINFO(5:6)
Day   = DateINFO(7:8)
WRITE(*,*) 'Date information -> ', DateINFO
WRITE(*,*) '            Year -> ', Year
WRITE(*,*) '           Month -> ', Month
WRITE(*,*) '             Day -> ', Day
```

Output:

```
Date information -> 19970811
            Year -> 1997
           Month -> 08
             Day -> 11
```

Example:

- Now do the same for time:

```
Hour       = TimeINFO(1:2)
Minute     = TimeINFO(3:4)
Second     = TimeINFO(5:10)
PrettyTime = Hour // ':' // Minute // ':' // Second
WRITE(*,*)
WRITE(*,*) 'Time Information -> ', TimeINFO
WRITE(*,*) ' Hour            -> ', Hour
WRITE(*,*) ' Minute          -> ', Minute
WRITE(*,*) ' Second          -> ', Second
WRITE(*,*) ' Pretty Time     -> ', PrettyTime
```

Output:

```
Time Information -> 010717.620
            Hour -> 01
          Minute -> 07
          Second -> 17.620
     Pretty Time -> 01:07:17.620
```