<u>Basic Elements of FORTRAN</u>

FORTRAN is one of the computer languages commonly used by scientists and engineers for technical calculations. These technical calculations are too difficult or take too long to be performed by hand.

FORTRAN has its own special alphabet. The special alphabet used with the FORTRAN 90/95 language is known as the FORTRAN character set. It consists of 86 symbols shown below.

<u>FORTRAN 90/95 character set</u>

26	Uppercase letters of the alphabet: A through Z
26	Lowercase letters of the alphabet: a through z
10	Digits: 0 through 9
1	Underscore character: _
5	Arithmetic symbols: + − * / **
18	Miscellaneous symbols: ( ) . = , ‘ $: ! “ % & ; < > ? $

FORTRAN is case insensitive. (For example, the uppercase letter A is equivalent to the lowercase letter a.) This behavior is in contrast with such case sensitive languages as C in which A and a are two totally different things.

A FORTRAN program consists of a series of statements designed to accomplish the goal of the programmer. The two basic types of statements are **executable statements** and **nonexecutable statements**. Executable statements describe the actions a program takes when it is executed. (e.g. additions, subtractions, multiplications, divisions). Nonexecutable statements provide information necessary for the proper operation of the program. FORTRAN 90/95 is a free-source form, where FORTRAN statements may be entered anywhere on a line, and each line may be up to 132 characters long. A too long statement can be continued on the next line by ending the current line (and optionally starting the next line) with an ampersand (&) character. A statement can continue over 40 lines. A statement label is a number between 1 and 99999. It is the “name” of the FORTRAN statement and may be used to refer to a statement from other parts of the program. It is **not** a line number, and it tells nothing about the order in which the statements are executed. Statement labels are optional, and most Fortran 90/95 statements will not have one. If a statement label is used, it must be unique within a given program unit. Any characters following a exclamation point are comments and are ignored by the FORTRAN compiler. Comments may appear on the same line as an executable statement. They are very important because they help us to document the proper operation of the program.

Constants and Variables

A FORTRAN constant is a data object that is defined before a program is executed and that does not change value during the execution of the program. When FORTRAN compiler encounters a constant, it places constant value in a known location in memory and then references that memory location whenever the constant is used in the

program. A FORTRAN Variable is a data object that can change value during the execution of a program. When a FORTRAN compiler encounters a variable, it reserves a known location in memory for the variable and then references that memory location whenever the variable is used in the program. FORTRAN names may be up to 31 characters long and may contain any combination of alphabetic characters, digits, and the underscore (_) character. First character in a name must be always an alphabetic.
Examples of valid variable names: time distance z123456789 I_want_to_go_home
Examples of invalid variable names:

This_is_a_very_long_variable_name	(Name is too long)
3_days	(First character is a number)
A$	($ is a illegal character)
my-help	(- is a illegal character)

Use meaningful variable names whenever possible.

There are five intrinsic or built-in types of FORTRAN constants and variables. Three of them are numeric (types INTEGER, REAL, and COMPLEX), one is logical (type LOGICAL), and one consists of string of characters (type CHARACTER).

Integer Constants and Variables

An integer constant is any number that does not contain a decimal point. It can be positive, negative or zero. A positive constant may be written either with or without a plus sign. No commas may be embedded within an integer constant.
Examples of valid integer constants: 0 -999 123456789 +17
Examples of invalid integer constants:

1,000,000	(Embedded commas are illegal)
-100.	(If it has a decimal point, it is not an integer constant)

An integer variable is a variable containing a value of the integer data type.

Real Constants and Variables

A real constant is any number with a decimal point. It can be positive, negative or zero and can be written with or without an exponent. A positive constant may be written with or without a plus sign.
Examples of valid real constants: 10. -999.9 1.0E-3 0.12E+1 123.45E20
Examples of not valid real constants:

1,000,000.	(Embedded commas are illegal)
111E3	(A decimal point is required in the mantissa)
-12.0E1.5	(Decimal points are not allowed in exponents)

A real variable is a variable containing a value of the real data type.

Character Constants and Variables

A character constant is a string of characters enclosed in single (‘) or double (“) quotes.

Examples of valid character constants:

‘This is a test !’	
“This is a test !”	
‘ ’	(A single blank)
‘{^}’	(Legal in character context)
‘3.141593’	(A character string, not a number)

Examples of not valid character constants:

This is a test !	(No single or double quotes.)
‘This is a test !”	(Mismatched quotes.)

Logical Constants and Variables

A logical constant is a constant that can take on one of two possible values: .TRUE. or .FALSE.

Valid logical constants:

.TRUE.
.FALSE.

Not valid logical constants:

TRUE	(No periods – this is a variable name)
.FALSE	(Unbalanced periods)

Default and Explicit Variable Typing

Two possible ways in which the type of a variable can be defined: default typing and explicit typing. If the type of a variable is not explicitly specified in the program, then default typing is used.

Default: Any variable names beginning with the letters I,J,K,L,M or N are assumed to be a type INTEGER. Any variable names starting with another letter are assumed to be type REAL.

```
Explicit:   INTEGER :: var1, var2, var3, ….
            REAL :: var1, var2, var3, …..
            LOGICAL :: var1,  var2,  var3, …
            Character(len= <len>) :: var1, var2, var3, ….
```

Type declaration statement with a PARAMETER attribute

```
Type, PARAMETER :: name = value
```

Example,

```
REAL, PARAMETER :: pi = 3.141593
CHARACTER,   PARAMETER  :: error message = ‘Unknown error !’
```

Keep your physical constants consistent and precise throughout a program. To improve the consistency and understandability of your code, assign a name to all important constants and refer to them by name in the program.

Assignment Statements

General Form is variable_name = expression

The assignment statement calculates the value of the expression to the right of the equal sign and assigns that value to the variable named on the left of the equal sign. The equal sign means "store the value of expression into location variable_name."

The standard arithmetic operators included in FORTRAN are

+	Addition
−	Subtraction
*	Multiplication
/	Division
**	Exponentiation

The above are binary operators.
In addition, the + and − symbols can occur as unary operators. Examples +23 −a

Rules when using FORTRAN arithmetic operators:

1. No two operators may occur side by side.
2. Implied multiplication is illegal in FORTRAN. Example x * (y + z)
3. Parenthesis may be used to group terms whenever desired. When parenthesis are used, the expressions inside the parenthesis are evaluated before the expression outside the parenthesis. Example 2 ** ((8+2) / 5)

Integer Arithmetic

Integer Arithmetic always produces a result that is an integer. Therefore, if the division of two integers is not itself an integer, the computer automatically truncates the fractional part of the answer.

Real Arithmetic (or floating-point arithmetic)

Real Arithmetic always produces a result that is real. Because of the finite word length of a computer, some real numbers cannot be represented exactly. For example, the number 1/3 is equal to 0.33333333333…. , but since the numbers stored in the computer have limited precision, the representation of 1/3 in the computer might be 0.3333333. As a result of this limitation in precision, some quantities that are theoretically equal will not be equal when evaluated by the computer. For example, on some computers
3. * (1./3.) ≠ 1. but 2. * (1./2.) = 1. Tests for equality must be performed very cautiously when working with real numbers.

Beware of real arithmetic: Limited precision can cause two theoretically identical expressions to give slightly different results.

Hierarchy of Operations

Arithmetic operations in an expression are evaluated in the following order:

1. The contents of the parenthesis are evaluated first, starting from the innermost parenthesis and working outward.
2. All exponentials are evaluated, working from right to left.
3. All multiplications and divisions are evaluated, working from left to right.
4. All additions and subtractions are evaluated, working from left to right.

Use parenthesis as necessary to make your expressions clear and easy to understand.

Mixed-Mode Arithmetic

An operation between an integer and a real number is called a mixed-mode operation, and an expression containing one or more such operations is called a mixed-mode expression. When a mixed-mode operation is encountered, FORTRAN converts the integer into a real number and then performs the operation to get a real result. The automatic mode conversion does not occur until a real number and an integer both appear in the same operation. Therefore, a portion of an expression can be evaluated in integer arithmetic, and another portion can be evaluated in real arithmetic.

Use integer exponents instead of real exponents whenever possible.
Never raise a negative number to a real power.

Assignment Statements and Logical Calculations

Logical variable name = logical expression

The above assignment statement calculates the value of the expression to the right of the equal sign and assigns that value to the variable named on the left of the equal sign. The expression to the right of the equal sign contains any combination of valid logical constants, logical variables, and logical operators.

The two basic types of logical operators are **relational operators** and **combinational operators**.

Relational Operator

General form: a_1 op a_2

Where a_1 and a_2 are arithmetic expressions, variables, constants, or character strings, and op is one of the relational logic operators as shown below

= =	Equal to
/ =	Not equal to
>	Greater than
> =	Greater than or equal to

<	Less than
< =	Less than or equal to

If the relationship between a_1 and a_2 expressed by the operator is true, then the operation returns a value of .TRUE. ; otherwise, the operation returns a value of .FALSE.

Be careful not to confuse the equivalence relational operator (= =) with the assignment operator.

Combinational Logic Operators

General form: l_1 .op. l_2

Where l_1 and l_2 are logical expressions, variables, or constants and .op. is one of the combinational operators as shown below

l_1 .AND. l_2	Logical AND	Result is TRUE if both l_1 and l_2 are TRUE; otherwise, it is FALSE
l_1 .OR. l_2	Logical OR	Result is TRUE if both l_1 and l_2 are TRUE; otherwise, it is FALSE
l_1 .EQV. l_2	Logical equivalence	Result is TRUE if l_1 is same as l_2 (either both TRUE or both FALSE); otherwise, it is FALSE
l_1 .NEQV. l_2	Logical non-equivalence	Result is TRUE if one of l_1 and l_2 is TRUE and the other one is FALSE; otherwise, it is FALSE
.NOT. l_1	Logical NOT	Result is TRUE if l_1 is FALSE, and FALSE if l_1 is TRUE

The periods are part of operator and must always be present.

Logical operators in an expression are evaluated in the following order.

1. All arithmetic operators are evaluated first in the order previously described.
2. All relational operators (= =, / = , >, >=, <, <=) are evaluated from left to right.
3. All .NOT. operators are evaluated.
4. All .AND. operators are evaluated from left to right.
5. All .OR. operators are evaluated from left to right.
6. All .EQV. and .NEQV. operators are evaluated from left to right.

Assignment Statements and Character Variables

General form: character variable name = character expression

The above assignment statement calculates the value of the character expression to the right of the equal sign and assigns that value to the variable named on the left of the equal sign. The expression to the right of the equal sign contains any combination of valid character constants, character variables, and character operators.

The two basic types of logical operators are **substring specifications** and **concatenation**.

Substring Specification

It selects a portion of a character variable and treats that portion as if it were an independent character variable. For example, variable str1 contains ‘123456’, then the substring str1(2:4) would be ‘234’. *Note colon in substring specification*

Concatenation (//) Operator

It combines two or more strings or substrings into a single large string. *For example str1 // str1(2:4) would result as ‘123456234’*

Intrinsic Functions

Most common functions used in scientific calculations are the trigonometric functions, logarithms and square roots. Rarer functions are hyperbolic functions and Bessel functions. These functions are built in directly into the FORTRAN language.

List-directed Input and Output Statements

The input statement is of the form *READ (*,*) i, j , a, chars*
The parenthesis *(*,*)* in the statement contains control information for the read. The first field in the parenthesis specifies the input/output unit from which the data is to be read. An asterisk in this field means that data is read from the standard input device for the computer (keyboard). The second field specifies the format in which the data is to be read. An asterisk in this field means that list-directed input (free-format input) is to be used. This means that type of the variables in the variable list determine the required format of the input data.

The output statement is of the form *WRITE (*,*) i, j, a, chars*
In above explanation replace read by write for the given output statement.

Always initialize all variables in a program before using them.

The Implicit None Statement

When used it disables the default typing provisions of FORTRAN. When included in program, any variable that does not appear in an explicit type declaration statement is considered an error. It should appear after the program statement and before any type declaration statements.

For example,

```
 PROGRAM test_1
IMPLICIT NONE
REAL :: time
time = 10.0
Write (*,*) ‘Time =’ , time
END PROGRAM
```

Always explicitly define every variable in your programs and use the IMPLICIT NONE statement to help you spot and correct typographical errors before they become program execution errors.