Chapter 1

Basic Elements of Fortran

1.1 Variable Names

A name is a symbolic link to a location in memory. A variable is a memory location whose value may be changed during execution. Names must:

- have between 1 and 63 alphanumeric character (alphabet, digits and underscore).
- start with a letter.

One should not use Fortran keyword or standard intrinsic (in-built) function as a variable name. Tempting names that should be avoided in this respect include: counts, len, product, range, scale, size, sum, tiny. The following are valid variable names: `SUST_UNITED`, `as_easy_as_123`. The following are not: `Math+Physics` ('+' is not allowed), `999help` (starts with a number), `Hello!` ('!' would be treated a comment not as a part of the variable name)

1.2 Data Types

In Fortran there are 5 intrinsic data types.

1. Integer
2. Real
3. Complex
4. Character
5. Logical

The first three are numeric types while the other two are non-numeric types. It is also possible to have derived data types and pointers.

1.2.1 Integer

Integer constants are whole numbers without a decimal point. e.g. 100, +16, -14, 0, 666. They are stored exactly, but their range is limited; typically -2^{n-1} to $2^{n-1}-1$. Where n is either 16 (for 2-byte integer) or 32 (for 4-byte integer). It is possible to change the default range using the `kind` type parameter.

1.2.2 Real

Real constant has a decimal point and maybe entered as either fixed point, eg. 442.2 or floating point, eg. 4.122e+02. Real constants are stored in exponential form in memory, no matter how they are entered. They are accurate only to a finite machine precision (which again, can be changed using the `kind` type parameter).

1.2.3 Complex

Complex constants consist of paired real number corresponding to real and imaginary parts. eg. (2.0, 3.0) corresponds to $2 + 3i$.

1.2.4 Character

Character constants consists of strings of characters enclosed by a pair of delimiters, which may be either single (’) or double (”) quotes. eg. `"This is a string"`, `'Department of Mathematics'`. The delimiter themselves are not part of the string.

1.2.5 Logical

Logical constants may be either true or false.

1.3 A program to compute square root of a number

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{a}{x_n}\right) \to \sqrt{a}$$

Using `if (...) exit`

```fortran
program newton
    implicit none
    real a
    real x, xold
    real change
    real, parameter :: tolerence=10e-6
    print*, "Enter a number:"
    read*, a
    x=1.0
    do
        xold=x
        x=0.5*(x+a/x)
        print*, x
        change=abs((x-xold)/x)
        if(change<tolerence) exit
    end do
end program newton
```

Listing 1.1: Newton’s method for finding square root (using if)

Using `do while`

```fortran
program newton
    implicit none
    real a
    real x, xold
    real change
    real, parameter :: tolerence=10e-6
    print*, "Enter a number:"
    read*, a
    x=1.0
    do while(change>tolerence)
        xold=x
        x=0.5*(x+a/x)
        print*, x
        change=abs((x-xold)/x)
    end do
end program newton
```

Listing 1.2: Newton's method for finding square root (using do while)

1.4 Declaration of Variable

1.4.1 Type

Variables should be declared (that is, have either data types defined and memory set aside for them) before any executable statements. This is achieved by a type declaration statement of form, eg.

```
integer num
real x
complex z
logical answer
character letter
```

More than one variable can be declared in each statement, eg. `integer i,j,k`

1.4.2 Initialization

If desired variables can be initialized in their type-declaration statement. In this case a double colon (: :) must be used. Thus, the above examples might become

```
integer :: num=2
real :: x=0.5
complex :: z=(0.0, 1.0)
logical :: answer=true
character :: letter='A'
```

Variables can also be initialized with a data statement. eg.
`data, num, x, z, answer, letter / 20,50, (0.09,1.0), .false., 'B'/`
The data statement must be placed before any executable statement.

1.4.3 Attributes

Various attributes may be specified for variables in their type-declaration statement. One such is `parameter`. A variable declaration with this attribute may not have its value changed within the program unit. It is often used to emphasize key physical or mathematical constants. eg. `real, parameter :: gravity=9.81`

1.4.4 Precision and Kind

By default, `real x` will occupy 4 bytes of computer memory and will be inaccurate in the sixth significant figure. The accuracy can be increased by replacing this type statement by `double precision x` with the floating-point variable now requiring twice as many bytes of memory. Better portability can be used using `kind` parameters. Avoid `double precision` statement by using

```
integer, parameter :: rkind=kind(1.0d0)
```

followed by the declaration for all floating point variable like :

```
real (kind=rkind) x
```

To switch to single precision for all floating-point variable just replace `1.0d0` by `1.0` in the first statement.

Intrinsic functions which allow you to determine the `kind` parameter for different types are

```
selected_char_kind(name)
selected_int_kind(range)
selected_real_kind(precision,range)
```

1.5 Operators and Expression

1.5.1 Numeric Operator

A numeric expression is a formula combining constants, variables and functions using the numeric intrinsic operators given in the following table:

Operator	Meaning	Precedence (1=highest)
**	x^y (Exponential)	1
*	xy (Multiplication)	2
/	$\frac{x}{y}$ (Division)	2
+	$x + y$ (Addition) or $(+x)$ unary plus	3
-	$x - y$ (Subtraction) or $(-x)$ unary minus	3

Repeated exponential is the single exception to the left-to-right rule for equal precedence.

$$a**b**c \rightarrow a^{b^{c}}$$

1.5.2 Type Coercion

When a binary operator has operands of different type, the weaker type is coerced to the stronger type and the result is of the stronger type. eg. 3/10.0 → 3.0/10.0

1.5.3 Character Operator

There is only one character operator, concatenation, //, eg. `"shah"//"jalal"` gives `"shahjalal"`

1.6 Line Discipline

The usual layout of statement is one pre line. However, there may be more than one statement per line separated by a semi colon; eg, a=1.0;b=1.0; c=100

```
radius = degrees * Pi &
                  / 180.0
```

is same as the single line statement `radius = degrees * Pi / 180.0`

1.7 Remarks

1.7.1 Pi

The constant π appears a lot in mathematical programming. eg, whenever converting between degrees and radians.
If a `real` variable Pi is declared then its value can be set within the program: Pi=3.14159. But it is good to declare it as a `parameter` in its type statement. eg, `real, parameter :: Pi=3.14159`. Alternatively, a popular method to obtain an accurate value is to insert the result

$$\tan(\frac{\pi}{4}) = 1.0$$
$$\Rightarrow Pi = 4.0 * atan(1.0)$$

1.7.2 Exponents

If an exponent (”Power”) is coded as an integer it will be worked out by repeated multiplication.
eg, $a**3$ will be worked out as a*a*a
$a**(-3)$ will be worked out as 1/(a*a*a)

For non-integer powers (including whole numbers if a decimal point is used) the result will be worked out by $a^b = (e^{\ln a})^b$

a**3.0 will be worked out something akin to $e^{3.0 \ln a}$. However, the logarithms of negative numbers don't exist. So the following Fortran statement is legitimate:

$$x = (-1) * *2$$

but the next one isn't

$$x = (-1) * *2.0$$

The bottom line is that

- If the exponent is genuinely a whole number, then don't use a decimal point or for small powers, simply write it explicitly as a repeated multiple. eg, a*a*a

- Take special care with odd roots of negative numbers. e.g, $(-1)^{1/3}$; ypu should work out the fractional power of the magnitude, then adjust the sign. eg, write $(-8)^{1/3}$ as $-(8)^{1/3}$

Remember because of the integer arithmetic the Fortran statement x**(1/3) actually evaluates to x**0(=1.0; presumably not intended). To ensure real arithmetic code as x**(1.0/3.0).

A useful intrinsic function for setting sign of an expression as `sign(x,y)` → absolute value of `x` times the sign of `y`.