Chapter 2
Basic Fortran

- Why program need define data types
- Fortran data types: integer, real, complex, character, logical
- Declare constants and variables

Data types in Fortran

- INTEGER: between $\pm 2^{31} - 1$ (on sleet)
- REAL: 0.337456E3, 33745.6E-2
- CHARACTER: see table 2-1 in pp17.
- Character strings: `"PQS-A"`, `"Fortran class"`

Identifiers: used to identify programs, constants, variables, and other entities in a program.

Begin with a letter, followed by up to 30 letters, digits or underscores. Wrong examples: `Pd-fl, 3dimension, white.smith`

Fortran 90 makes no distinction between upper case and lower case. (except in character strings).

Variables in a Fortran program (one kind of identity)

Compiler associates a variable with a memory location (different type will be locate different size of memories).

Type statement

- `INTEGER::Sum, Age`
- `REAL::Number1, Mass`
- `CHARACTER(LEN = 16)::  Name, LastName`
- `CHARACTER(5) ::  Unit, fUnit*10`

Variable initialization:

- `INTEGER::Sum = 0, Age = 23`
- `REAL::Number1 = -13.14, Mass = 2.36E2`
- `CHARACTER(LEN = 16)::Name=" "`
- `CARACTER(15) ::  LastName = "Wei"`

It is a good idea to initialize all the variables.

The `IMPLICIT NONE` statement.

Fortran has an implicit naming convention: any undeclared identifier whose name begins with `I,J,K,L,M,N` will be typed as integer, and all others will be typed as real.

To avoid errors caused by the implicit convention, the `IMPLICIT NONE` statement should be always used.

Named constants using `PARAMETER` attribute

```
INTEGER, PARAMETER ::  limit = 66

REAL, PARAMETER ::  PI=3.14159, TwoPi = 2.0*PI

CHARACTER(2), PARAMETER ::  Unit = "cm"

CHARACTER(*),PARAMETER ::  Unit = "cm"
```

The value of a constant cannot be changed. A constant can be used to anywhere in the program.

Numeric operations:

`+` plus

`-` minus

`*` multiplication

`/` division

`**` exponentiation

Example: `B**2 - 4*A*C` means $B^2 - 4AC$.

Pay attentions to the types of constants and variables in operations: when two numbers of the same type are combined using one of the four basic arithmetic operations, the result keeps the same type.

```
9.0/4.0 -> 2.25
```

```
9/4 -> 2
```

If

```
REAL ::  X=2.0
```

```
INTEGER ::  N=2
```

what are values of

```
1.0/X
```

```
1/N  ?
```

Mixed-mode expression:

When an integer quantity is combined with a real one, the integer quantity is converted to its real equivalent, and the result is of real type.

```
1.0 / 4
```

```
3.0 + 8 / 5
```

```
3.0 + 8.0 / 5
```

Mixed-mode expression using `+, -, *, /` is considered poor programming practise.

Avoid to use integers when doing a division.

The operation of `**` can used for different types of operands.

```
2.0 ** 3
```

```
(-4.0) ** 2 !(compare to -4.0 ** 2)
```

```
2 ** 2
```

The result of `2.0 ** 29.0` and `2 ** 29` are different.

`2 ** 29 -> 2 * 2 * ...  * 2 -> 536870912`

`2.0 ** 29.0 ->` $e^{29.0 \ln(2.0)}$ `-> 536870900.0`

The follow will cause an error:

```
(-4.0)**2.0
```

Never use a real exponent in place of an integer exponent

Sometimes real exponent is useful.

$\sqrt{8.0}$ `8.0 ** 0.5` or `8.0 ** (1.0/2.0)`

$\sqrt[3]{7.0}$ `7.0 ** (1.0/3.0)`

How about `7.0 ** (1/3), 8.0 ** (1/2)` ?

Priority rules:

- All exponentiations are performed first; consecutive exponentiations are performed from right to left.
- All multiplications and divisions are performed next, in the order in which they appear from left to right.
- The additions and substitutions are performed last, in the order in which they appear from left to right.

Using parentheses can change the standard order.

What are the results of the following expressions?

```
2 ** 3 ** 2

10 - 8 - 2

10 / 5 * 2

2 + 4 / 2

2 + 4 ** 2 / 2

( 5 *(11 - 5) ** 2) * 4 + 9
```

Numeric Functions:

Fortran library provides some numeric functions, See Table 2-2 in textbook pp. 24.

Pay attentions to the **type of argument(s)**

A wrong type of an argument will cause an error.

A useful function is `REAL(x)` which converts an integer to a real.

If `Number` is an integer, then `SQRT(REAL(Number))` works, but `SQRT(Number)` doesn't work.

Character operations:

Concatenation : `//`

`"centi" // "meters"` produces the string `"centimeters"`

Suppose `Unit = "using "`

Then `Unit // "centi" // "meters"` produces the string `using centimenters"`

Substring: `(a:b)`

```
CHARACTER(15) ::  Course
```

```
Course = "Engineering"
```

Then

`Course(2:4)` is `"ngi"`

`Course(:6)` is `"Engine"`

`Course(7:)` is `"ering    "`

The assignment statement:

Used to put a value to the variable

Format: *variable = expression*

"=" does not mean "equal", but "assign"

Example:

```
R = 5.23

Number = 17

Unit = "cm"

Value = LOG(PI*R**2)

Number = Number + 1
```

In an assignment statement, if the right side is some expression of calculation, the computer will first compute the value of the expression and then assign to the variable.

If an integer-valued expression is assigned to a real variable, the integer value is converted to a real value and then assigned to the variable.

If a real-valued expression is assigned to an integer variable, the fractional part of the real value is truncated and the integer part is assigned to the variable.

Pay attentions to the data type and avoid using mixed-mode assignments. `REAL`, `INT` can be used to change types

Assignment for a character string

If the length of the value is not equal to that of the variable, then the value is padded with blanks or truncated as necessary

```
character(3) :: name, longName*10, &
 lastName*5
longName = "Fortran90"
name = longName
lastName = longName(5:)
print*, name, longName, lastName
```

The values of character strings are as follows:

`name -> "For"`

`longName -> "Fortran90 "`

`lastName -> "ran90 "`

The output of the previous slide will be

`ForFortran90 ran90`

Note: `&` means continue the line

`longName*10` means the length of longName is 10

An assignment may also used to modify part of a string.

```
CARACTER(5) :: String = "alpha"
String(3:5) = "ter"
String(2:3) = String(4:5)
```

Then `String -> "alpha" -> "alter" -> "aerer"`

Note `String(2:4) = String(3:5)` is invalid because positions the new values are being assigned can not be referenced.

Output

```
PRINT *, output-list
```

```
WRITE (*,*) output-list
```

Input

```
READ *, input-list
```

```
READ(*,*) input-list
```

" * " means default Input/Output devices, i.e., keyboard input and screen output.

Example:

```
PRINT *, name

PRINT *, "The name is ",name

WRITE (*,*) "Please input your name:"

PRINT *,

WRITE (*,*)
```

For INPUT:

- A new line of data is processed each time a READ statement is executed.
- If there are fewer entries in a line of input data than there are variables in the input list, successive lines of input are processed until values for all variables in the list have been obtained.
- If there are more entries in a line of input data than variables, the remaining values are ignored.
- The type of input should be the same as the type of the variable.
- Consecutive entries in a line of inputs must be separated by a comma or space.

Input character strings:

Need to notice the length of the strings. In the following cases, “’ ” or “ " ” should be used.

- The character value extends over more than one line.
- The character value contains blanks, commas, or slashes.
- The character value begins with an apostrophe, a double quote, or a string of digits followed by an asterisk.

```
PROGRAM CHARACTERS
IMPLICIT NONE
CHARACTER :: name*5, lastName*10
INTEGER :: birthYear
PRINT *, "input your first name:"
READ (*,*) name
PRINT *, "input your last name:"
READ (*,*) lastName
WRITE (*,*) "Input the year you born:"
READ(*,*) birthYear
PRINT *, name, ", you are ", 2010 - birthYear, "year's old."
END PROGRAM CHARACTERS
```

General form of a Fortran program:

```
PROGRAM name
IMPLICIT NONE
```

specification part (declare variables, constants)

execution part

subprogram part (we will learn that later)

```
END PROGRAM name
```

`STOP` statement will also terminate the execution of the program.

```
STOP
```

```
STOP constant
```

STOP constant will print out the constant. It can be used to test the program.

In Fortran program, a line of comment begins with exclamation mark !

The compiler will ignore the comments.

A good programmer should write necessary comments to improve the readability of the program.

```
!this program is used to calculate temperature
REAL :: temp !current temperature
```

A line may have a maximum of 132 characters.

An ampersand (&) must be used for continuation lines. At most 39 continuation lines are permitted.

For a continuation of strings, two ampersands should be used:

```
PRINT *, "Please enter your first name followed &
  & by your last name:"
```

Sometimes, a statement label can be used. A label is an integer in the range 1 through 99999.