Outline

- Introduction
- Fortran Language Basics
- Intrinsic Functions
- DO Loop Statements
- Conditional Statements
- Files and Formatted Input/Output
- Arrays
- Functions and Subroutines

Books for reference (IISEE library)

◆INTRODUCTION TO Fortran 90 for Engineers and Scientists
by Larry R. Nyhoff and Sanford C. Leestma
(New Jersey: Prentice Hall, 1996)

◆Introduction to Fortran 90/95
by Stephen J. Chapman (New York: McGraw-Hill, 1998)

◆Introduction to Programming with Fortran:
With Coverage of Fortran 90, 95, 2003, 2008 and 77
by Ian Chivers and Jane Sleightholme (Berlin: Springer, 2006)

What is programming ?

$$32 + 45 =$$

$$132.56 \times 356.32 =$$

$$\frac{\left[\log\left(1.07 \times 10^{22}\right) - 9.1\right]}{1.5} =$$

$$\sum_{m=11}^{101} \left(m + m^2 + 13\right) =$$

Complicated calculations with hand calculations

→ require a great deal of time and effort
→ human errors (careless mistakes) are unavoidable

Complicated calculations with computers

→ enables you to calculate complicated problems in a moment (save time and effort)
→ enables automatic calculations
→ enables parallel processing
→ human errors can be fixed

There are thousands of programing languages.

AWK, BASIC, C(C++), COBOL, FORTRAN, Java, Pascal, Perl, PHP, Python, R, Ruby, VJB, and more…

Programing language is…

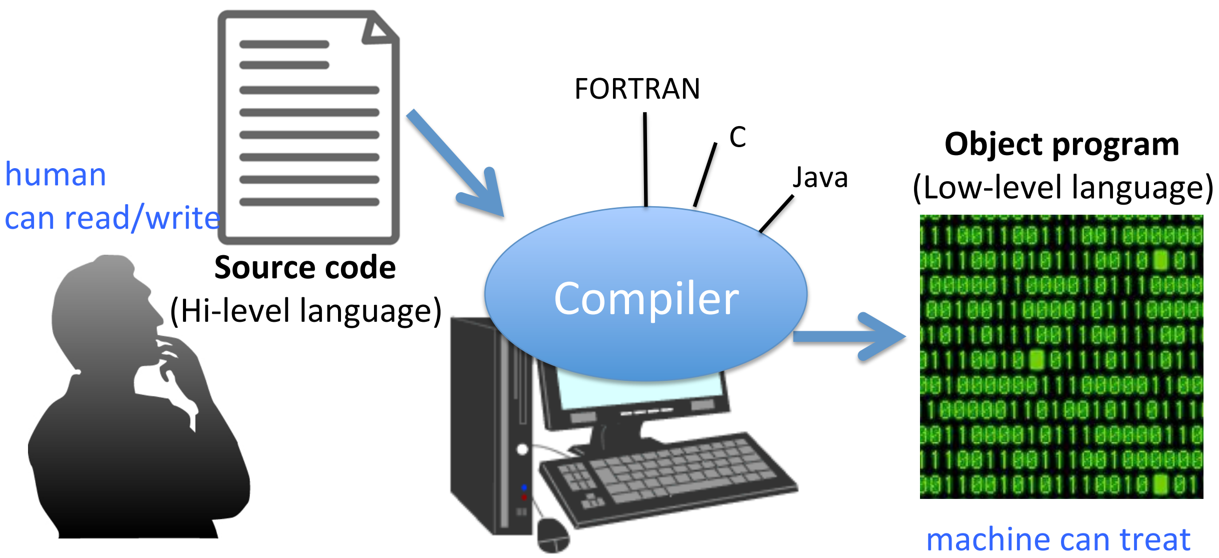
- a formal language designed to communicate instructions to a machine (computer)
- used to control behaviors of a machine
- describe computation in an imperative style (e.g. a sequence of commands), although some languages use alternative forms of description

Programming and Compilation

Inputs
- parameters
- data files

Outputs
- calculated values
- files

Programming:
the act or process of planning and writing a program for solving problems



Compilation:
the comprehensive process that leads from an original formulation of a computing problem to executable programs

1 FORTRAN LANGUAGE BASICS

FORTRAN =
mathematical **FOR**mula **TRAN**slation system

Developed
in : 1954 (appeared in 1957)
by : John Backus（IBM）
→ the first successful High-level Language

[History]
FORTRAN → FORTRAN II-IV → FORTRAN66 → FORTRAN77
Fortran90 → Fortran95 → Fortran2003 → Fortran2008 →

1.1 Why Fortran?

- Many seismological software have been developed using Fortran.
  → necessary to understand, utilize or modify existing programs

- Fortran is still very popular among scientists.

- Fortran can be used on Linux, Windows and Mac.
  → one of the versatile programming languages

- Programming is effective to improve understanding
  → it is impossible to write a code without proper understanding of mathematics

1.2 Advantages of Fortran90

- consists of simple and concise descriptions

- difficult to make errors in programs

- has convenient tools for numerical calculations

- free compilers (e.g., gfortran, g95) are available

- allows old (FORTRAN77) source codes

- designed for high performance computing (HPC) on large-scale parallel machines

1.3 Message Display

Let’s make a simple program!

Here we use “nedit” to edit programs.

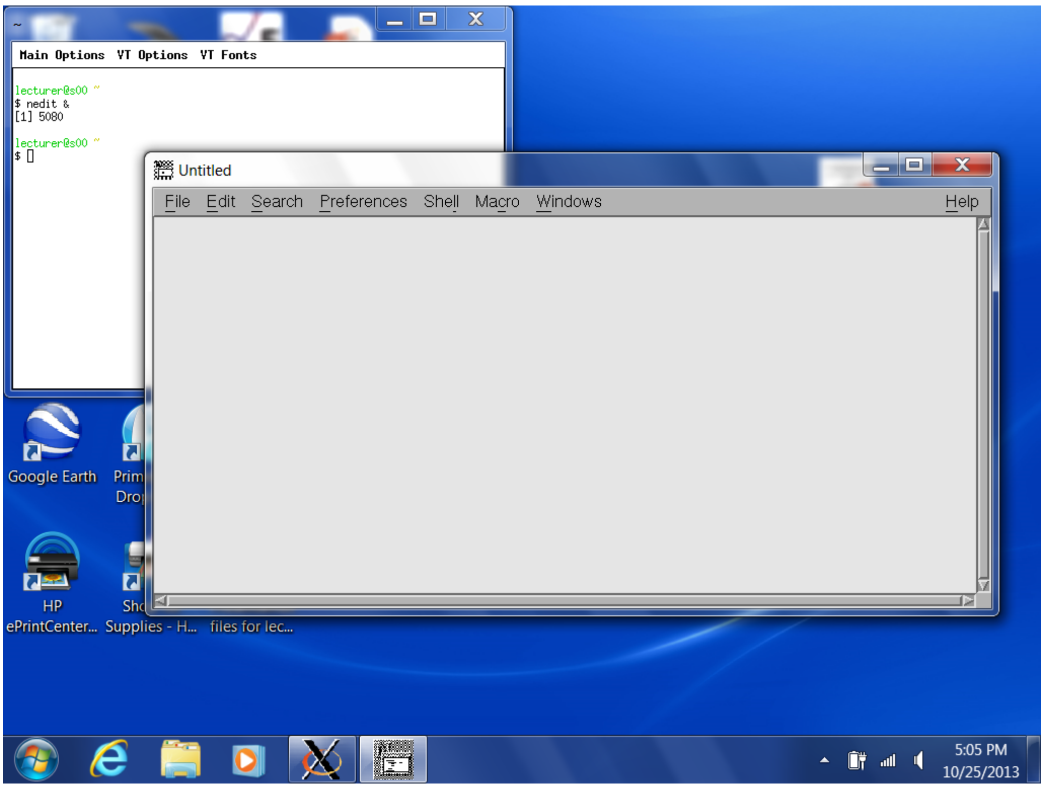
Type as:

```
$ nedit    &
```

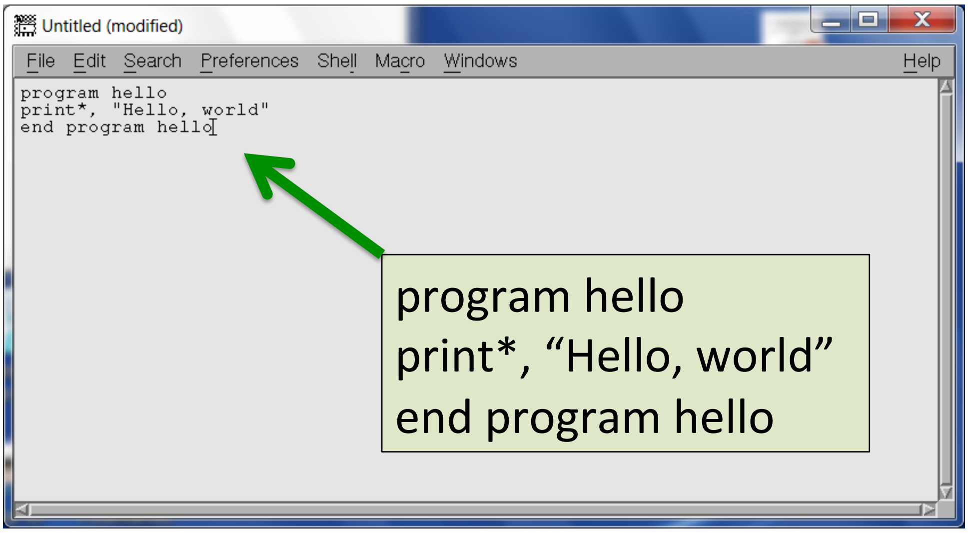
→ then you will get the window shown in the next slide.

Note: When you put “&” after the command, that command is executed as a background job.

nedit



Write the following in the *nedit* window.



program hello
print*, “Hello, world”
end program hello

Save the file

Save the program as hello.f90:

(a) Click “File” → “Save As…“ in the editor

(b) Then write the name of the file (in the below column) → “ex1_1.f90”

(c) Click “OK”

EXERCISE 1-1

a) Use the *ls* command in the *xterm* window to confirm the new file “ex1_1.f90” is created.

b) Type as “cat ex1_1.f90” in the *xterm* to see the content.

Compile and Execution

(a) Type the following command to compile the program:

```
$ gfortran ex1_1.f90
```

or gfortran -o ex1_1.exe ex1_1.f90

(b) Use the *ls* command to confirm that “a.exe” is created.

(c) Then type as:

```
$ ./a.exe
```

“./” indicates current directory

source code
(hello.f90)

Compilation
($ gfortran hello.f90)

object program
(./a.exe)

1.4 Structure of a Fortran program

```
program hello
  print*,   “Hello, world!”
end program hello
```

- Declaration section
- Execution section
- Termination section

<Declaration section>

program name
definition of functions/variables -> later

<Execution section>

read files/numbers, calculation, output files/numbers -> later

<Termination section>

stop and end

Sample program:

Declaration section

```
program quadratic_equation
 implicit none
 real :: a, b, c, D, x1, x2
```

Execution section

```
 write(*, *) 'input a, b, c : '
 read(*, *) a, b, c
 if (a == 0.0d0) stop 'stop, a should be nonzero'
   D = b * b - 4.0 * a * c
 if (D >= 0.0d0 ) then
   x1 = 0.5 * (-b + sign(sqrt(D), -b)) / a
   x2 = c / (a * x1)
   write(*, 100) 'x1 =',x1,'  x2 =',x2
 else
   D = -1.0 * D
   x1 = 0.5 * (-b) / a
   x2 = 0.5 * (sqrt(D)) / a
 write(*, 101) 'x1 =',x1,'+',x2,'i  x2 =',x1,'-',x2,'i'
 endif
 100 format(a4,e12.4,a6,e12.4)
 101 format(a4,e12.4,a1,e12.4,a7,e12.4,a1,e12.4,a1)
```

Termination section

```
stop
end program quadratic_equation
```


Fortran character set

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

Program name

```
program hello        program name
    print*,    "Hello, world"
end program hello
```

- Fortran program names may be up to 31 characters* long.
- The first character of the name must be a letter.
  (False: program 3hello)
- Cannot use intrinsic function names as program name
  (False: program sqrt)
- A clear and concise name is preferable.

*Some compilers allow names longer than 31 characters.

Comment lines

```
! The first program (2015/11/4)
program hello          !  exercise on S&T course
  print*,  "Hello, world!"    !  message
end program hello
```

- Descriptions after "!" in a line are neglected.
  → useful for leaving notes for yourself or others

EXERCISE 1-2
Insert comment lines into the program ex1_1.f90 as above and check the output remains the same (save the file as ex1_2.f90).

Continuation of lines

```
program hello
print*,      &
"Hello, world!"
end program hello
```

When you end a line with an ampersand "&" character, you can continue the statement on the next line.

EXERCISE 1-3
Modify the program ex1_1.f90 as above and check the output remains the same (save the file as ex1_3.f90).

PRINT/WRITE statements

- The following statement

  `print*, expression`

  writes the values/characters of one or more expressions to the screen.

- The following statement

  `write(*, *) expression`

  writes the values/characters of one or more expressions to the specified output device.
  In the above statement, the standard output device is specified by the first "*" and this indicates "screen".
  (The second "*" specifies the standard format.)

Example: greeting.f90

```
program greeting        !  coded by T.Hayashida
print*,  “ Good morning ! ”          !  message
write(*, *)  “ How are you ? ”
write(*, *)  “ It’s cold &
today ! ”
write(*, *)  ' I am sleepy… '
end program greeting
```

comment lines (neglected)

print statement

write statement

continuation of lines

True:	‘ I am sleepy. ‘
True:	“ I am sleepy. ”
True:	“ I’m sleepy. ”
False:	‘ I’m sleepy. ‘

EXERCISE 1-4
Use the *write* statement in place of *print* statement in “ex1_1.f90” and check the output remains the same (save the file as ex1_4.f90).

POINTS:

(a) A FORTRAN program consists of three parts, declaration, execution and termination sections.

(b) When you put an exclamation mark (!) in a program, the part after the mark is interpreted as comments.

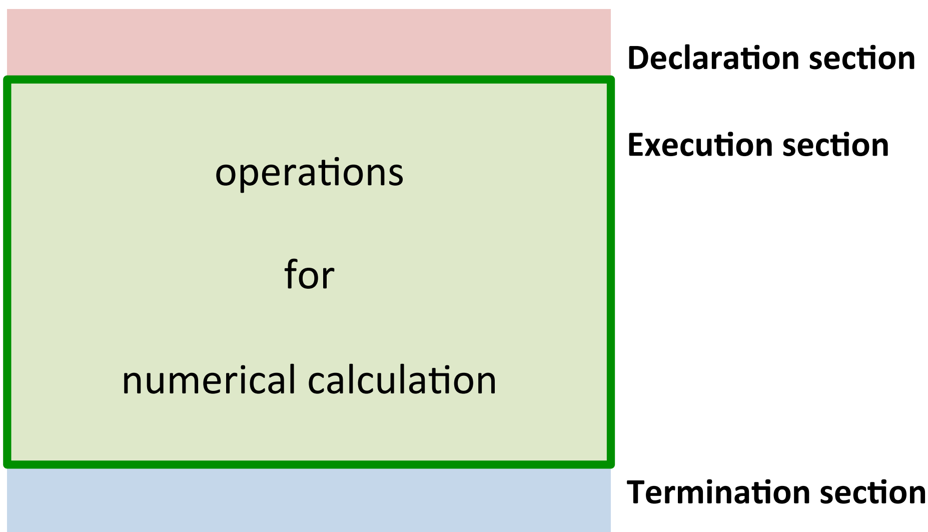
(c) When you put an ampersand (&) in a program, the part after the mark is continued in the next line.

(d) The *print* statement displays outputs on screen and the *write* statement writes outputs to the specified device (screen or file).

2. INTRINSIC FUNCTIONS

How do you perform these calculations using Fortran ?

$8 \div 2$	$\log_{10} 100$
2.1×3.5	5^2
$\sin 30°$	$\sqrt{16}$



→ To perform calculations, we have to learn arithmetic operations and intrinsic procedures in Fortran.

2.1 Arithmetic operation

Math	FORTRAN
$a+b$	`a + b`
$a-b$	`a - b`
$a \times b$	`a * b`
$a \div b$	`a / b`
a^2	`a**2  or  a*a`
a^3	`a**3`

Math	FORTRAN
$\frac{(a+b)}{c}$	`( a + b ) / c`
$a+\frac{b}{c}$	`a + b / c`

Note:

- Multiplying operations (×, ÷) have a higher priority than adding operations (+, -)
- Exponentiation (a^2, a^3) has a higher priority than multiplying operations

EXERCISE 2-1

Translate the following mathematical expressions into those of Fortran.

Example: $a+b+c$ → `a + b + c`

1. $\frac{a(b+c)}{d}$
2. $\frac{a}{b}+c$
3. $a \times b^2$
4. $a^2 + b^3$

EXERCISE 2-2

Find answers of the following Fortran mathematical expressions involving integers (a=2, b=6, c=3, d=5).

Example: `a * b * c =` **36**

1. `a+b*c+d`
2. `a*b/c*d`
3. `d*b/c/a`
4. `d**a**c` 

2.2 Integer arithmetic

<Example>

Calculate a+b, a-b, a×b, and a÷b, provided a=8 and b=2.

```
program ex2_3
implicit none
integer a, b
a = 8
b = 2
write(*, *) "a + b =", a + b
write(*, *) "a - b =", a - b
write(*, *) "a * b =", a * b
write(*, *) "a / b =", a / b
stop
end program ex2_3
```

EXERCISE 2-3

Compile and run the above program (save the file as ex2_3.f90).

Assignment statement (1)

There are four ways to assign values to variables *a* and *b*:

(1) integer :: a = 8, b = 2

(2) integer a, b
read (*, *) a, b

(3) integer a, b
a = 8
b = 2

(4) integer a, b
data a, b/8, 2/

```
program ex2_3
implicit none

(1) – (4)

write(*, *) "a + b =", a + b
write(*, *) "a - b =", a - b
write(*, *) "a * b =", a * b
write(*, *) "a / b =", a / b
stop
end program ex2_3
```

Assignment statement (2)

- `a = 8` is an example of assignment statement.

- The right hand side is evaluated first, and the result is assigned to the left hand side variable.

- The followings are invalid:

```
8 = a
a + b = 10
a = b = 2
```


READ statement

- The following READ statement

  read(*, *) variable name [, variable name, etc.]

  reads one or more values from the standard input device (i.e., keyboard) specified by the first "*", and loads them into the variables in the list.

- The second "*" specifies the standard format.

EXERCISE 2-4
Compile and run the following program (save the file as ex2_4.f90).

```
program ex2_4
implicit none
integer a, b
write(*, *) "Input a, b"
read(*, *) a, b
write(*, *) "a, b =", a, b
write(*, *) "a + b =", a + b
write(*, *) "a - b =", a - b
write(*, *) "a * b =", a * b
write(*, *) "a / b =", a / b
stop
end program ex2_4
```

* when executing the program, type "8, 2" after the message "Input a, b".

Implicit declaration

program *name*
implicit none
...variable declarations...
...statements..
stop
end program *name*

- This forces a programmer to <u>declare all variables that are used</u> and this means that <u>some potential errors may be identified</u> during compilation.
- If this is not used, variables have a data type according to the initial letter of their name: those beginning with *i* , *j* , *k* , *l* , *m* and *n* being integers; and those beginning *a* to *h* and *o* to *z* being real numbers.

EXERCISE 2-5
Compile and run the following *incorrect* programs (save the files as ex2_5_1.f90 and ex2_5_2.f90, respectively).

```
program ex2_5a
integer :: a=8, b=2
write(*,*) "a, b =", a, c
write(*,*) "a + b =", a + c
write(*,*) "a - b =", a - c
write(*,*) "a * b =", a * c
write(*,*) "a / b =", a / c
stop
end program ex2_5a
```

```
program ex2_5b
implicit none
integer :: a=8, b=2
write(*,*) "a, b =", a, c
write(*,*) "a + b =", a + c
write(*,*) "a - b =", a - c
write(*,*) "a * b =", a * c
write(*,*) "a / b =", a / c
stop
end program ex2_5b
```

**There is no declaration regarding the integer variable c.*

2.3 Real arithmetic

<Example>

Calculate a+b, a-b, a×b, and a÷b, provided a=5.2 and b=2.4.

```
program ex2_6a
implicit none
real :: a=5.2, b=2.4
write(*, *) "a + b =", a + b
write(*, *) "a - b =", a - b
write(*, *) "a * b =", a * b
write(*, *) "a / b =", a / b
stop
end program ex2_6a
```

There are four ways to assign values to variables a and b, as stated before;

(1) real :: a = 5.2, b = 2.4
(2) real a, b
    read(*, *) a, b
(3) real a, b
    a=5.2
    b=2.4
(4) data a, b/5.2, 2.4/

EXERCISE 2-6
Compile and run the above program (save the file as ex2_6a.f90).

```
program ex2_6b
implicit none
real a, b
read(*, *) a, b
write(*, *) "a + b =", a + b
write(*, *) "a - b =", a - b
write(*, *) "a * b =", a * b
write(*, *) "a / b =", a / b
stop
end program ex2_6b
```

```
a + b =    7.5999999
a - b =    2.7999997
a * b =    12.480000
a / b =    2.1666665
```

```
program ex2_6c
implicit none
real :: a=5.2, b=2.4
write(*, '(a,x,f4.1)') "a + b =", a + b
write(*, '(a,x,f4.1)') "a - b =", a - b
write(*, '(a,x,f4.1)') "a * b =", a * b
write(*, '(a,x,f4.1)') "a / b =", a / b
stop
end program ex2_6c
```

```
a + b =  7.6
a - b =  2.8
a * b = 12.5
a / b =  2.2
```

```
program ex2_6d
implicit none
double precision :: a=5.2, b=2.4
write(*, *) "a + b =", a + b
write(*, *) "a - b =", a - b
write(*, *) "a * b =", a * b
write(*, *) "a / b =", a / b
stop
end program ex2_6d
```

```
a + b =    7.5999999046325684
a - b =    2.7999997138977051
a * b =    12.480000038146954
a / b =    2.1666665010982156
```

EXERCISE 2-6
Compile and run the above three programs (save the file as ex2_6b.f90, ex2_6c.f90, and ex2_6d.f90, respectively).

2.4 Specification Statements

Integer literals

```
123   -123    0     235467     -3288
```

Single precision real (floating point numbers) literals

```
1.23     -3.24    7.998    3.24767e+3
0.4676e-2     1.00     1.0e+0     0.
```

Double precision real (floating point numbers) literals

```
3.1415926535d+0   -4.78d+6
1.0d+0
```

Single precision complex literals

```
(5.229,-4.78)   (0., 1.)
(3.2767e+2,-0.65e-2)
```


2.5 Mixed-mode operation

	Integer	Real	Double	Complex
Integer	**Integer**	Real	*Double*	Complex
Real	Real	**Real**	*Double*	Complex
Double	*Double*	*Double*	**Double**	*Double Complex*
Complex	Complex	Complex	*Double Complex*	**Complex**

Mixture of integer and real types

```
program ex2_7
implicit none
integer :: a = 2
real :: b = 4.2
!
write(*, *) "a + b =", a + b
write(*, *) "a * b =", a * b
write(*, *) "b / a =", b / a
stop
end program ex2_7
```

EXERCISE 2-7
Compile and run the above program (save the file as ex2_7.f90).

Notes:

(1) Do not divide any number by zero.

(2) When solving a complicated equation as;

$$y = \frac{(-1)^{n-1}}{(2n-1)^2}$$

It is better to solve the problem dividing the equation into several parts to avoid confusion.

a = (-1.0)**(n – 1)
b = (2.0 * n – 1)**2
y = a / b

```
program single_precision
implicit none
real pi                          Real       Real
write(*, *) 'pi =', 2.0 * acos(0.0)
stop
end program single_precision
```

pi = 3.141593

```
program double_precision
implicit none
double precision pi
write(*, *) 'pi =', 2.0d0 * acos(0.0d0)
stop                   Double        Double
end program double_precision
```

pi = 3.14159265358979

$$\arccos(-1.0) = \pi$$

```
program double_precision
implicit none
double precision pi
write(*, *) 'pi =', acos(-1.0)
stop                       Real
end program double_precision
```

pi = 3.141593

	Real	Double
Real	**Real**	*Double (unstable)*
Double	*Double (unstable)*	**Double**

```
program double_precision
implicit none
double precision pi
write(*, *) 'pi =', 2.0d0 * acos(0.0)
stop                  Double       Real
end program double_precision
```

pi = 3.14159274101257
(correct: pi = 3.14159265358979…)

2.6 Intrinsic functions

Generic name	*Type of argument	Function	Example
int	I	conversion to integer	int(−2.5) = −2
			int(−2.5d0) = −2
			int(cmplx(−2.5, 1.3)) = −2
real	R	conversion to real	real(2) = 2.0
			real(2.5d0) = 2.5
			real(cmplx(2.5, 1.3)) = 2.5
dble	D	conversion to double precision	dble(2) = 2.0d0
			dble(2.5) = 2.5d0
			dble(cmplx(2.5, 1.3)) = 2.5d0
cmplx	C	conversion to complex	cmplx(2) = cmp(2.0, 0.0)
			cmplx(2.5) = cmplx(2.5, 0.0)
			cmplx(2.5d0) = cmplx(2.5, 0.0)
aimag	R	imaginary part of complex	aimag(cmplx(2.5, 1.3)) = 1.3
conjg	C	complex conjugate	conjg(cmplx(2.5, 1.3)) = cmplx(2.5, −1.3))

*I: Integer R: Real D: Double precision C: Complex

Generic name	Type of argument	Function	Example
aint	R	truncation	aint(−2.6) = −2.0
	D		aint(−2.6d0) = −2.0d0
nint	I	rounding	nint(−2.6) = −3
	I		nint(−2.6d0) = −3
anint	R	rounding	anint(−2.6) = −3.0
	D		anint(−2.6d0) = −3.0d0
mod	I	remainder	mod(−7, 3) = −1
	R		mod(7.0, −3.0) = 1.0
	D		mod(7.0d0, −3.0d0) = 1.0d0
abs	I	absolute value	abs(−2) = 2
	R		abs(−2.0) = 2.0
	D		abs(−2.0d0) = 2.0d0
	C		abs(cmplx(−3.0, 4.0) = 5.0
* sign	I	transfer of sign	sign(−2, 3) = 2
	R		sign(−2.0, −3.0) = −2.0
	D		sign(−2.0d0, −3.0d0) = −2.0d0

*sign(*A*,*B*) returns the value of *A* with the sign of *B*.

EXERCISE 2-8
Derive remainder of 11÷4 using the intrinsic function *mod* (save the program file as ex2_8.f90).

EXERCISE 2-9
Derive remainder of 11÷4 without using the intrinsic function *mod* (save the program file as ex2_9.f90).
Hint: use intrinsic function *int*

Generic name	Type of argument	Function	Example
max	I	maximum value	max(−2, 1) = 1
	R		amax0(−2, 1, 0) = 1.0
	I		max1(−2.0, 1.0) = 1
	R		max(−2.0, 1.0) = 1.0
	D		max(−2.0d0, 1.0d0) = 1.0d0
min	I	minimum value	min(−2, 1) = −2
	R		amin0(−2, 1, 0) = −2.0
	I		min1(−2.0, 1.0) = −2
	R		min(−2.0, 1.0) = −2.0
	D		min(−2.0d0, 1.0d0) = −2.0d0
* dim	I	positive difference	dim(−1, −3) = 2
	R		dim(1.0, 3.0) = 0.0
	D		dim(1.0d0, 3.0d0) = 0.0d0

*dim (A,B) = a – b (if $A > B$)
dim (A,B) = 0. (if $A < B$)

Generic name	Type of argument	Function	Example
sqrt	R	square root	sqrt(4.0) = 2.0
	D		sqrt(4.0d0) = 2.0d0
	C		sqrt(cmplx(−1.0, 0.0)) = cmplx(0.0, 1.0)
exp	R	exponential	exp(1.0) = 2.7818282
	D		exp(0.0d0) = 1.0d0
	C		exp(cmplx(0.0, 0.0)) = cmplx(1.0, 0.0)
log	R	natural logarithm	log(1.0) = 0.0
	D		log(1.0d0) = 0.0d0
	C		log(cmplx(1.0, 0.0)) = cmplx(0.0, 0.0)
log10	R	common logarithm	log10(100.0) = 2.0
	D		log10(100.0d0) = 2.0d0

Generic name	Type of argument	Function	Example
sin	R	sine	sin(3.14159265) = 0.0
	D		sin(0.0d0) = 0.0d0
	C		sin(cmplx(0.0, 0.0)) = cmplx(0.0, 0.0)
cos	R	cosine	cos(3.14159265) = −1.0
	D		cos(0.0d0) = 1.0d0
	C		cos(cmplx(0.0, 0.0)) = cmplx(1.0, 0.0)
tan	R	tangent	tan(3.14159265) = 0.0
	D		tan(0.0d0) = 0.0d0
asin	R	arcsine	asin(0.0) = 0.0
	D		asin(0.0d0) = 0.0d0
acos	R	arccosine	acos(1.0) = 0.0
	D		acos(1.0d0) = 0.0d0
atan	R	arctangent	atan(0.0) = 0.0
	D		atan(0.0d0) = 0.0d0
*atan2	R	arctangent	atan2(0.0, 1.0) = 0.0
	D		atan2(0.0d0, 1.0d0) = 0.0d0

*atan2(a, b) = atan (a/b)

trigonometric function

θ		sinθ	cosθ	tanθ
degrees	radians			
0°	0	0	1	0
30°	1/6 π	1/2	√3/2	√3/3 (1/√3)
45°	1/4 π	√2/2 (1/√2)	√2/2	1
60°	1/3 π	√3/2	1/2	√3
90°	1/2 π	1	0	-
120°	2/3 π	√3/2	-1/2	-√3
135°	3/4 π	√2/2	-√2/2	-1
150°	5/6 π	1/2	-√3/2	-√3/3
180°	π	0	-1	0
210°	7/6 π	-1/2	-√3/2	√3/3
225°	5/4 π	-√2/2	-√2/2	1
240°	4/3 π	-√3/2	-1/2	√3
270°	3/2 π	-1	0	-
300°	5/3 π	-√3/2	1/2	-√3
315°	7/4 π	-√2/2	√2/2 (1/√2)	-1
330°	11/6 π	-1/2	√3/2	-√3/3
360°	2 π	0	1	0

definition of pi in Fortran: pi = acos (-1.0), pi = 2.0 * acos (0.0)

inputs	arcsin	inputs	arccos	inputs	arctan
0	0	1	0	0	0
1/2	1/6 π	√3/2	1/6 π	√3/3 (1/√3)	1/6 π
√2/2 (1/√2)	1/4 π	√2/2	1/4 π	1	1/4 π
√3/2	1/3 π	1/2	1/3 π	√3	1/3 π
1	1/2 π	0	1/2 π	-	-
√3/2	1/3 π	-1/2	2/3 π	-√3	-1/3 π
√2/2	1/4 π	-√2/2	3/4 π	-1	-1/4 π
1/2	1/6 π	-√3/2	5/6 π	-√3/3	-1/6 π
0	0	-1	π	0	0
-1/2	-1/6 π	-√3/2	5/6 π	√3/3	1/6 π
-√2/2	-1/4 π	-√2/2	3/4 π	1	1/4 π
-√3/2	-1/3 π	-1/2	2/3 π	√3	1/3 π
-1	-1/2 π	0	1/2 π	-	-
-√3/2	-1/3 π	1/2	1/3 π	-√3	-1/3 π
-√2/2	-1/4 π	√2/2 (1/√2)	1/4 π	-1	-1/4 π
-1/2	-1/6 π	√3/2	1/6 π	-√3/3	-1/6 π
0	0	1	0	0	0

EXERCISE 2-10

Compile and run the following program (save the files as ex2_10.f90).

```
program ex2_10
implicit none
real rad
real :: deg=30.
real, parameter :: pi = acos(-1.0)
 rad = deg * pi / 180.0
 write(*, *) 'sin30 =', sin(rad)
 write(*, *) 'cos30 =', cos(rad)
 write(*, *) 'tan30 =', tan(rad)
stop
end program ex2_10
```

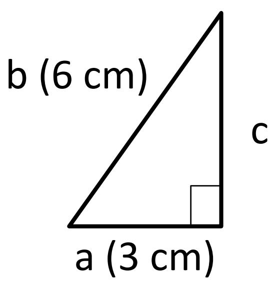
asin(x) : arcsin x

acos(x) : arccos x

EXERCISE 2-11

Compile and run the following program (save the files as ex2_11.f90).

```
program ex2_11
implicit none
real :: a=3.0, b=6.0, c
 c = sqrt(b**2 - a**2)
 write(*, *) 'c =', c
stop
end program ex2_11
```

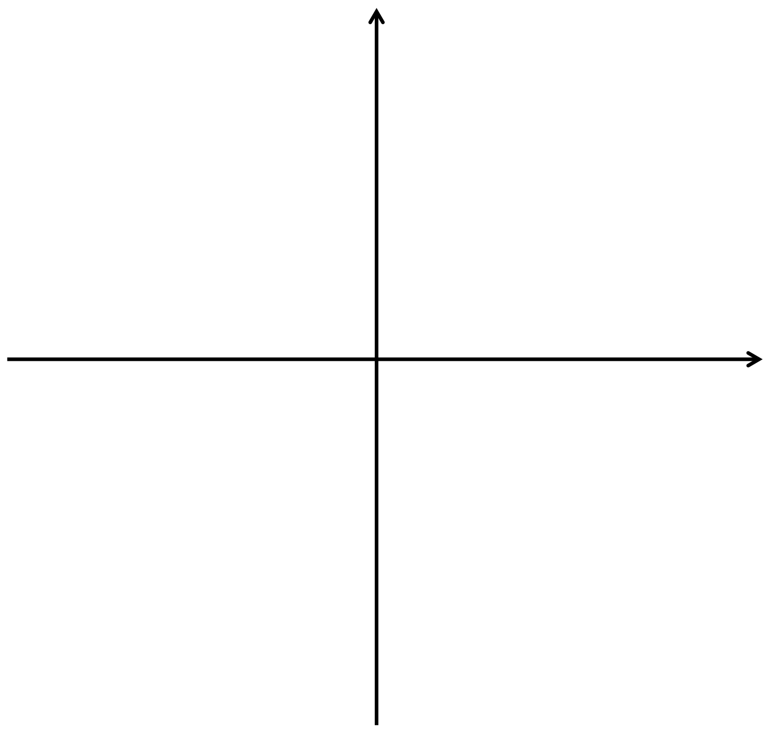


EXERCISE 2-12
Compile and run the following programs
(save the files as ex2_12a.f90 and ex2_12b.f90, respectively).

```
program ex2_12a
implicit none
real, parameter :: pi = acos(-1.0)
 write(*, *) pi
 write(*, *) sin(0.0), sin(pi/2.0), sin(pi)
stop
end program ex2_12a
```

```
program ex2_12b
implicit none
real, parameter :: pi = acos(-1.0)
 write(*, *) pi
 write(*, *) tan(0.0), tan(pi/4.0)
stop
end program ex2_12b
```


atan2



EXERCISE 2-13
Compile and run the following program (save the file as ex2_13.f90).

atan(x) : arctan x
atan2(x1,x2): arctan(x1/x2)

```
program ex2_13
implicit none
real rad2deg
real, parameter :: pi = acos(-1.0)
 write(*, *) pi
 rad2deg = 180.0/pi
 write(*, *) atan(1.0), atan(-1.0)
 write(*, *) atan(1.0)*rad2deg, atan(-1.0)*rad2deg
 write(*, *) atan2(1.0,1.0), atan2(1.0,-1.0)
 write(*, *) atan2(1.0,1.0)*rad2deg, &
             atan2(1.0,-1.0)*rad2deg
 write(*, *) atan2(-1.0,1.0), atan2(-1.0,-1.0)
 write(*, *) atan2(-1.0,1.0)*rad2deg, &
             atan2(-1.0,-1.0)*rad2deg
stop
end program ex2_13
```


EXERCISE 2-14*
Find answers of the following intrinsic functions using Fortran programs.

1. $\log_{10} 3200$
2. $\log_e 3200$
3. 7×3^4
4. $\left[\log_{10}\left(1.07\times 10^{22}\right) - 9.1\right] / 1.5$

EXERCISE 2-15*
Check the below trigonometric functions are the same (addition theorem) using Fortran programs.

1. $\sin\left(15° + 40°\right) = \sin 15° \cos 40° + \cos 15° \sin 40°$
2. $\cos\left(70° - 30°\right) = \cos 70° \cos 30° + \sin 70° \sin 30°$

POINTS:

(a) Priority of arithmetic operations
(Exponentiations > Multiple operators
> Adding operators)

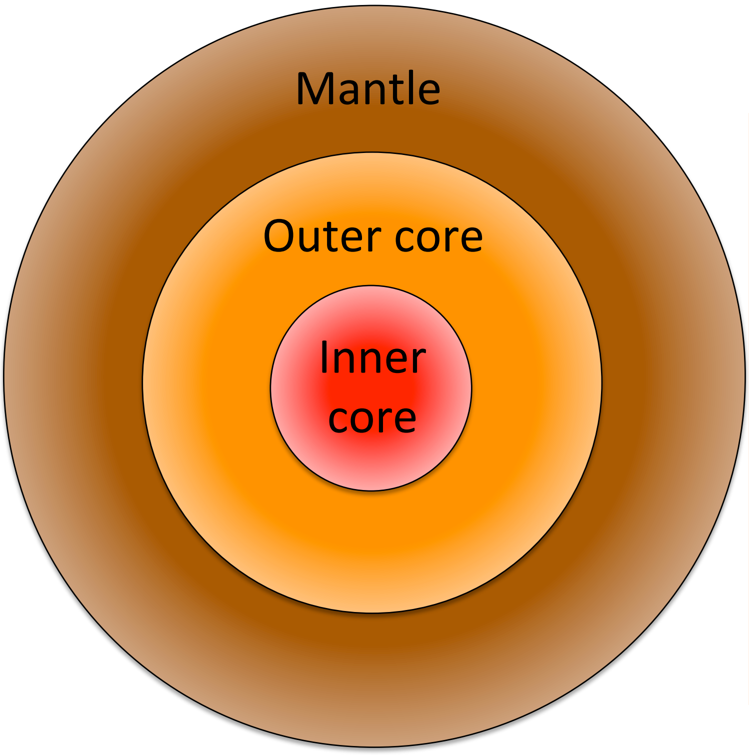
(b) Four ways to an assign value to a variable

(c) Implicit declaration (integer, real)

(d) Combination use of real/integer values

(e) Expressions of intrinsic functions

PROJECT I : Structure of the Earth



Radius of the Earth: *re*
6,371 [km]

Radius of the Outer Core: *rc*
3,482 [km]

Radius of the Inner Core: *ri*
1,217 [km]

Mass of the Earth: *me*
5.9737×10^{24} [kg]

Write a program earth.f90 to solve the following problems.

(a) Derive the volume of the earth (*ve*: in km^3), using *re*.

(b) Find the depth of the Core-Mantle Boundary (CMB).

(c) Find the depth of the Inner-Core Boundary (ICB).

(d) Find the mean density of the Earth (amass: in g/cc).
Hint: $\rho = M/V$ (M: mass, V: volume)

Procedure:

(a) Declare variables.
-> ________________________________

(b) Set known parameters (re, rc, ri, me).

(c) Derive formula to calculate the volume of the Earth.

(d) Derive formula to calculate the mean density of the Earth.

Unit conversion

ve [km^3], me [kg] -> (kg/km^3) -> (g / cc) ??

SI: kg / m^3
CGS: g / cm^3 = g / cc

$1\ kg / m^3 = 1{,}000\ g / 1{,}000{,}000\ cm^3$
$= 1.0 \times 10^{-3}\ g / cm^3$
$= 1.0 \times 10^{-3}\ g / cc$

me[kg] / ve[km^3]
$= me[kg] / (10^3 \times 10^3 \times 10^3 \times ve)\ [m^3]$ SI
$= [(1.0 \times 10^{-3}) \times me] / [(1.0 \times 10^9) \times ve]\ [g/cm^3]$ CGS
$= (1.0 \times 10^{-12}) \times (me/ve)\ [g/cm^3]$ CGS

SUPPLEMENTAL:

Fortran90

```
program product_ab
implicit none
integer :: a = 2, b = 5
write(*, *) 'a * b =', a * b
stop
end program product_ab
```

```
program product_ab
implicit none
integer a, b
a = 2
b = 5
write(*, *) 'a * b =', a * b
stop
end program product_ab
```

FORTRAN77

```
program product_ab
implicit none
integer a, b
parameter (a = 2, b = 5)
write(*, *) 'a * b =', a * b
stop
end program product_ab
```

```
program product_ab
implicit none
integer a, b
data a, b/2, 5/
write(*, *) 'a * b =', a * b
stop
end program product_ab
```

3. DO LOOP STATEMENTS

How do you solve the following problems using Fortran ?

$$\sum_{n=1}^{100} n = ?$$

$$\sum_{m=31}^{78} \left(m + m^2 + 13\right) = ?$$

3.1 Basic idea of DO loop

$$\sum_{n=1}^{100} n = 1 + 2 + 3 + \cdots + 99 + 100$$

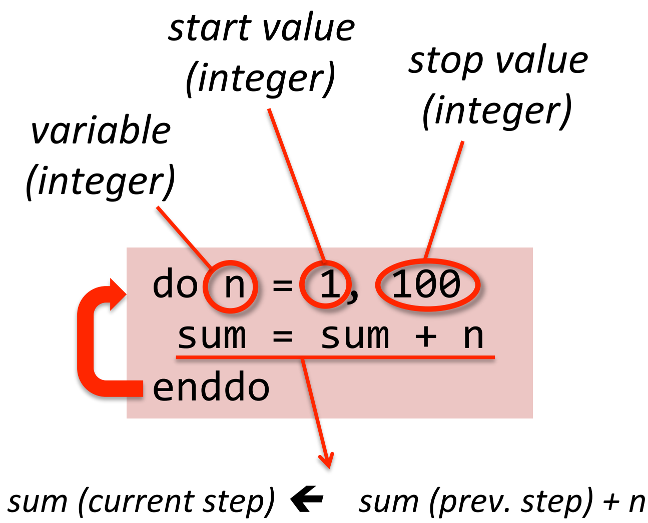
Here we define variable 'sum' that indicates sum total at each step.

before calculation: sum = 0

variable	*previous sum*				*updated sum*
n = 1	sum (=0) + n (=1)	=	1	→	sum
n = 2	sum (=1) + n (=2)	=	3	→	sum
n = 3	sum (=3) + n (=3)	=	6	→	sum
n = 4	sum (=6) + n (=4)	=	10	→	sum
	:				
n = 100	sum (=4950) + n (=100)	=	<u>5050</u>	→	sum

sum = 5050

```
program ex3_1
implicit none
integer n, sum
sum = 0
do n = 1, 100
 sum = sum + n
enddo
write(*, *) 'Sum =', sum
stop
end program ex3_1
```



EXERCISE 3-1
Compile and run the above program (save the file as ex3_1.f90).

EXERCISE 3-2
Modify the program ex3_1.f90 to derive the sum of the integers from 1 to arbitrary value *n* (save the file as ex3_2.f90).

3.2 Increment and decrement operators

(a) *start stop*

```
do n = 1, 6
  :
enddo
```

n: 1 → 2 → 3 → 4 → 5 → 6
(increment = 1)

(b) *increment*

```
do n = 1, 6, 2
  :
enddo
```

n: 1 → 3 → 5
(increment = 2)

(c)

```
do n = 6, 1, -1
  :
enddo
```

n: 6 → 5 → 4 → 3 → 2 → 1
(decrement = 1)

(d)

```
do n = 5, 5, 2
  :
enddo
```

n: 5
(increment is skipped)

EXERCISE 3-3
Make a program that displays multiples of 3 from 3 to 27 on the screen, using DO loop statement (save the file as ex3_3.f90).

EXERCISE 3-4
Make a program that displays Olympic years (Summer Olympic Games) from 1948 to 2012 on the screen, using DO loop statement (save the file as ex3_4.f90).

EXERCISE 3-5
Compile and run the below program (save the file as ex3_5.f90).

```
program ex3_5
implicit none
integer i
do i = 1, 10
 write(*, *) 0.1 * real(i)
enddo
stop
end program ex3_5
```


EXERCISE 3-6
Derive the following problems using Fortran.
(save the program file as ex3_6_1.f90, ex3_6_2.f90 and ex3_6_3.f90, respectively).

(a) $\sum_{n=1}^{100}(2n+5)$ (b) $\sum_{n=1}^{100}n^2$ (c) $\sum_{m=31}^{78}\left(m^2+7m+12\right)$

EXERCISE 3-7
Prove the following formulas using Fortran (Let *n* be a positive integer). Derive the left-hand sides by using DO loop and check both the left and right hand sides return the same value (save the program file as ex3_7_1.f90 and ex3_7_2.f90, respectively).

(a) $\sum_{k=1}^{n}k^2=\frac{1}{6}n(n+1)(2n+1)$ (b) $\sum_{k=1}^{n}k^3=\frac{1}{4}n^2(n+1)^2$

(Hint)

$$\sum_{k=1}^{n} k = \frac{1}{2} n(n+1)$$

```
program summation
 implicit none
 integer k, n, left, right
 write(*, *) 'Input n'
 read(*, *) n
 left = 0
 do k = 1, n
   left = left + k
 enddo
 right = n * ( n + 1 ) / 2
 write(*,*) 'left  = ', left
 write(*,*) 'right = ', right
 stop
end program summation
```


3.3 Double DO loops

Create a multiplication table up to 12 (1×1 ~ 12×12).

×	1	2	3	4	5	6	7	8	9	10	11	12
1	1	2	3	4	5	6	7	8	9	10	11	12
2	2	4	6	8	10	12	14	16	18	20	22	24
3	3	6	9	12	15	18	21	24	27	30	33	36
4	4	8	12	16	20	24	28	32	36	40	44	48
5	5	10	15	20	25	30	35	40	45	50	55	60
6	6	12	18	24	30	36	42	48	54	60	66	72
7	7	14	21	28	35	42	49	56	63	70	77	84
8	8	16	24	32	40	48	56	64	72	80	88	96
9	9	18	27	36	45	54	63	72	81	90	99	108
10	10	20	30	40	50	60	70	80	90	100	110	120
11	11	22	33	44	55	66	77	88	99	110	121	132
12	12	24	36	48	60	72	84	96	108	120	132	144

EXERCISE 3-8

Compile and run the below program that displays from 1*1 to 9*9 (save the file as ex3_8.f90).

```
program ex3_8
 implicit none
 integer i, j
 do i = 1, 12
   do j = 1, 12
     write(*, *) i, '*', j, '=', i * j
   enddo
 enddo
 stop
end program ex3_8
```

```
          1  *        1 =             1
          1  *        2 =             2
          :           :               :
         12  *       11 =           132
         12  *       12 =           144
```


Supplemental:

Fortran90

```
program summation
implicit none
integer n, sum
sum = 0
do n = 1, 100
  sum = sum + n
enddo
write(*, *) 'Sum =', sum
stop
end program summation
```

summation.f90

FORTRAN77

```
      program summation
      implicit none
      integer n, sum
      sum = 0
      do 100 n = 1, 100
        sum = sum + n
100    continue
      write(*, *) 'Sum =', sum
      stop
      end program summation
```

statement label (arbitrary integer)

summation.f, summation.for

STOP and END statements

There are two ends to a program:

→ The physical end (the last statement in the program)

→ The logical end (where the program stops execution)

- The **END** statement is the physical end of the program.
- The **END** statement should be last line in the program.
- The **END** statement is a non-executable statement.

- The **STOP** statement is the logical end of the main program.
- The **STOP** statement aborts the execution of the program and can display the "message" on the screen.