

IISEE lecture for group training

# Fortran programming for beginner seismologists

## Lesson 2

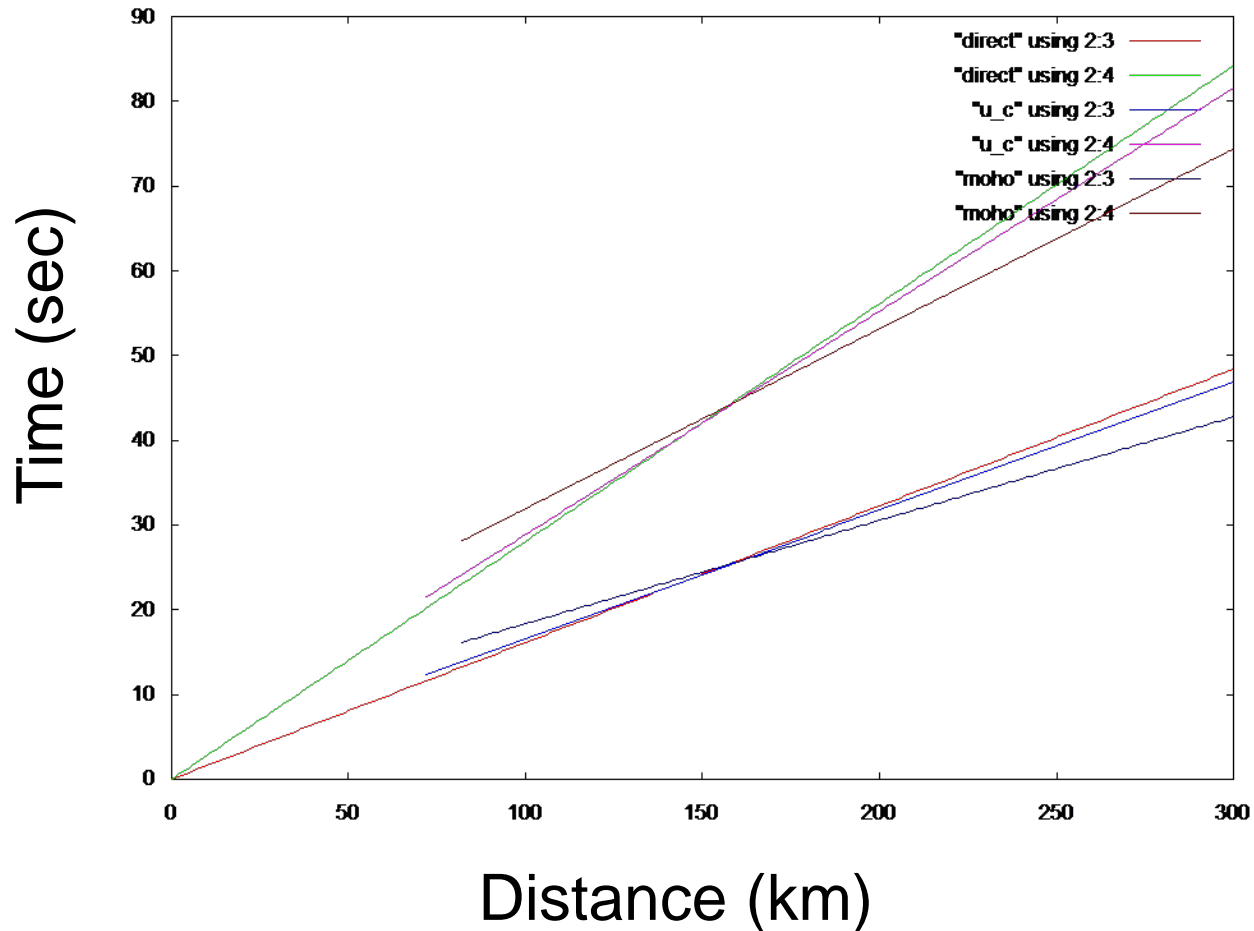
Lecturer

Tatsuhiko Hara

# A small project

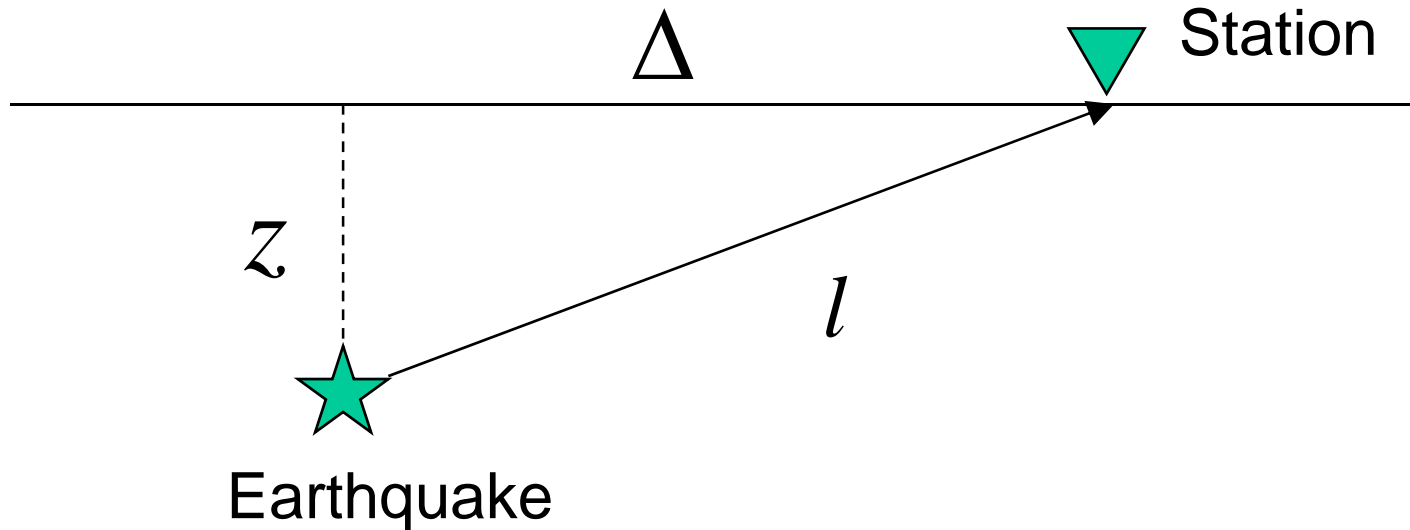
- Now we start a small project in which we make a program to compute travel times for a layered crust model to learn Fortran programming.

# An example of travel time curves



# First Step

- We start our project from the most simple case, homogeneous half space shown below.



# Travel times

The travel times of P and S waves ( $t_p$  and  $t_s$ ) are given by the following formula, respectively:

$$t_P = \frac{l}{V_P} = \frac{\sqrt{\Delta^2 + z^2}}{V_P}$$

$$t_S = \frac{l}{V_S} = \frac{\sqrt{\Delta^2 + z^2}}{V_S}$$

where  $V_p$  and  $V_s$  are P and S wave speeds, respectively.

# From maths to Fortran

- The equations in the previous slide are mathematical expressions. We have to *translate* them into those of Fortran to carry out calculations.
- Note that it is necessary to define seismic wave speeds ( $V_p$  and  $V_s$ ), epicentral distance ( $\Delta$ ), and depth ( $z$ ) to calculate  $t_p$  and  $t_s$ .

# What is a “variable”?

- As already mentioned, we have to define the values of physical parameters such as  $V_p$ . How can we do that using Fortran?
- First of all, we have to decide the name and type of a “variable” which stores a value of a certain physical parameter.
- Here, as an example, we use “vp” as a variable to store the value of  $V_p$  (although you can choose another name, it is good to choose a name to make it easy to remember which physical parameter is assigned).

# On naming a variable

- Names are case insensitive. The followings are the same in Fortran programs:

`vp`, `VP`, `Vp`, `vP`

- The maximum number of characters that you can use for a variable is 31.
- You can use alphabets, numbers, and `_` to name a variable.



# Data type (1)

- Now the name of the variable for  $V_p$  is  $v_p$ . How about the “data type” for  $v_p$ ?
- In Fortran77, the following data types are available:

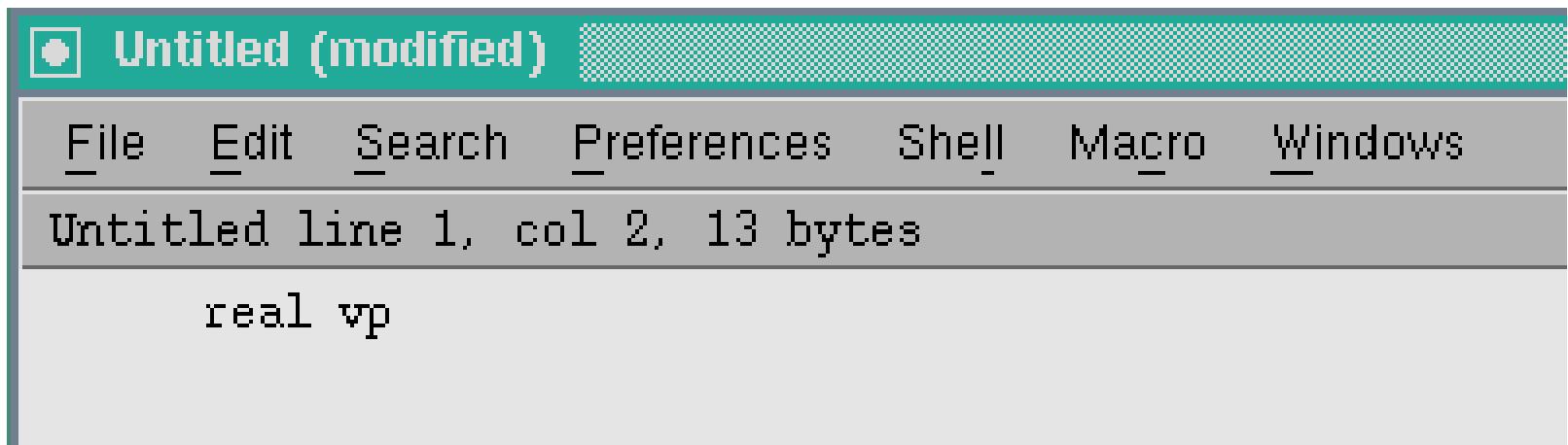
real, integer, double precision,  
complex, character, logical

## Example

'Hello!' in the hello.f is a character constant.

# Data type (2)

- We use the real type for `vp`.
- Declaration statement must be put at the beginning part of a program as shown below:



The image shows a screenshot of a text editor window. The title bar at the top reads "Untitled (modified)". Below the title bar is a menu bar with the following items: File, Edit, Search, Preferences, Shell, Macro, and Windows. The main text area contains the following text: "Untitled line 1, col 2, 13 bytes" followed by a blank line and then the declaration statement "real vp".

```
Untitled (modified)
File Edit Search Preferences Shell Macro Windows
Untitled line 1, col 2, 13 bytes
real vp
```

# *implicit typing*

- In the previous case, `vp` is explicitly declared as a `real` variable.
- In Fortran language, so called *implicit typing* is available where,
  - The names beginning with *i*, *j*, *k*, *l*, *m* and *n* are *implicitly* recognized as a `integer` variable if they are not explicitly declared.
  - Other names are *implicitly* recognized as a `real` variable.

# *IMPLICIT NONE*

- By putting *IMPLICIT NONE* statement at the top of a program, you can disable implicit typing.
- In this lecture, we use *IMPLICIT NONE*. This is because it is easy to find typos in programs (if you use an undefined variable, you will find an error message in compilation).

# Let's assign the value to vp

- There are three ways to assign a value to a variable vp:

(1) read( \*, \* ) vp

(2) data vp/6.0/

(3) vp = 6.0

# *READ* statement

- In the previous slide, there is the following expression as the first way to assign the value:

```
read( * , * ) vp
```

- As is the case for write statement, the first “\*” specifies the standard device (keyboard for *READ* statement), and the second “\*” specifies the format.

## EXERCISE 2-1

- i) Make a program to read a value of the variable  $v_p$  from the keyboard and print out it on the display.
- ii) Make a program to read values of the variables  $v_p$  and  $v_s$  from the keyboard and print out them on the display.

Hint: the following expression may be used:

```
read( *, * ) vp, vs
```

## EXERCISE 2-2

- i) Compile the following program, and see what happens:

```
implicit none
read(*,*) vp
write(*,*) vp
end
```

- ii) Correct the above program so that you can compile and run it.



# *Data* statement

- Data statement initializes variables.

## Example

```
implicit none
real vp
integer ii, jj
data vp/6.0/
data ii, jj/1, 2/
end
```

# Assignment statement

- $vp = 6.0$  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:

$$6.0 = vp$$

$$vp + vs = 10.0$$

$$vp = vs = 6.0$$

# Let's calculate travel times

A program to calculate travel times will consist of the following parts:

- Declaration of variables
- Assignment of values to each variable
- Calculation
- Output
- End statement

# Declaration of variables (1)

- As mentioned, variables for  $V_p$ ,  $V_s$ ,  $\Delta$ ,  $z$ ,  $t_p$  and  $t_s$  are necessary. The first four variables are necessary to compute travel times and the remaining two are necessary to store the results.
- Here, we develop a program only for  $T_p$ , and leave a part of  $T_s$  as an exercise.
- We use `vp`, `delta`, `z`, and `tp` for the names of  $V_p$ ,  $\Delta$ ,  $z$  and  $t_p$ , respectively.

# Declaration of variables (2)

- The part for declaration can be written as:

```
implicit none  
real vp, delta, z,
```

When you see this symbol, the code is incomplete. Please be careful. →



# Assignment of values to each variable (1)

- Here, we assume that  $v_p$  is a constant in the program, and  $z$  and  $\delta$  are read from the keyboard.
- So we use *DATA* statement for  $v_p$ , and *READ* statement for  $z$  and  $\delta$ .

# Assignment of values to each variable (2)

- The program can be developed as:

```
implicit none
real vp, delta, z, tp
data vp/6.0/
write(*,*) 'Focal depth (km):'
read(*,*) z
write(*,*) 'Epicentral distance (km):'
read(*,*)
```



# What is necessary to calculate travel times?

- Now we are ready to calculate travel times.
- To do that, we have to learn arithmetic operations and intrinsic functions in Fortran.



# Arithmetic operation (1)

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

# Arithmetic operation (2)

Math	Fortran
$\frac{x + y}{z}$	$(x+y) / z$
$x + \frac{y}{z}$	$x+y / z$

## Exercise 2-3

- Translate the following mathematical expressions into those of Fortran:

$$(1) \ xyz \quad (2) \ a - bc \quad (3) \ a + \frac{b}{c}$$

$$(4) \ a + b^3 \quad (5) \ \frac{a + b}{c} \quad (6) \ \frac{a}{b + c}$$

# Intrinsic functions

- Intrinsic functions are built-in functions in Fortran to calculate functions such as  $\sin(x)$ ,  $\cos(x)$ , and  $\log(x)$ .

## Examples of intrinsic functions

Generic name	Description	Specific names	Type of argument(s)	Type of result
<code>sin(x)</code>	Sine x in radians	<code>sin</code>	REAL	REAL
		<code>dsin</code>	DOUBLE	DOUBLE
		<code>csin</code>	COMPLEX	COMPLEX
<code>cos(x)</code>	Cosine x in radians	<code>cos</code>	REAL	REAL
		<code>dcos</code>	DOUBLE	DOUBLE
		<code>ccos</code>	COMPLEX	COMPLEX
<code>sqrt(x)</code>	Square root	<code>sqrt</code>	REAL	REAL
		<code>dsqrt</code>	DOUBLE	DOUBLE
		<code>csqrt</code>	COMPLEX	COMPLEX
<code>log(x)</code>	Natural logarithm	<code>alog</code>	REAL	REAL
		<code>dlog</code>	DOUBLE	DOUBLE
		<code>clog</code>	COMPLEX	COMPLEX

# Travel time calculations

- We can calculate travel time  $T_p$  by combining assignment statement, arithmetic expressions, and intrinsic functions as follows:

$$t_p = \text{sqrt}(z**2 + \text{delta}**2) / v_p$$

- Remember that in assignment statement, the right hand side is evaluated first, and then the value of the right hand side expression is assigned to the variable in the left hand side.

# Adding the part of calculation

- The program can be developed as:

```
implicit none
real vp, delta, z, tp
data vp/6.0/
write(*,*) 'Focal depth (km):'
read(*,*) z
write(*,*) 'Epicentral distance (km):'
read(*,*) delta
tp = sqrt(z**2+delta**2)/vp
```

# Output and End statement

- The program can be developed as:

```
implicit none
real vp, delta, z, tp
data vp/6.0/
write(*,*) 'Focal depth (km):'
read(*,*) z
write(*,*) 'Epicentral distance (km):'
read(*,*) delta
tp = sqrt(z**2+delta**2)/vp
write(*,*) 'P-wave velocity (km/s):', vp
write(*,*) 'Travel time of P-wave (s): ',
end
```





## Exercise 2-4

- Change the program developed in this lecture to calculate both  $T_p$  and  $T_s$  and print out them to the screen.

# Exercise 2-5-1

- Compile and run the following program:

```
implicit none
real pi

pi = acos(-1.0)
write(*,*) pi
write(*,*) sin(0.0), sin(pi/2.0), sin(pi)

end
```

# Exercise 2-5-2

- Compile and run the following program:

```
implicit none
real pi

pi = acos(-1.0)
write(*,*) pi
write(*,*) tan(0.0), tan(pi/4.0)

end
```

# Exercise 2-5-3

- Compile and run the following program:

```
implicit none
real pi,rad2deg

pi = acos(-1.0)
write(*,*) pi
rad2deg = 180./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

end
```