The format, code, declaration of variables for programming languages

Declarations of the variable types for Fortran and C/C++

	Fortran	C/C++
Short integer	integer*2	short short int
Integer with single precision	integer integer*4	int long long int
Integer with double precision	integer*8	long long int
Real with single precision	real real*4	float
Real with double precision	real*8 double precision	double
More precise than double (quad precision)	real*16 (The numerical expression is 0.q0 and built-in functions are the same as those of single precision.)	long double
Complex variables	complex*8 (single) complex*16 or double complex (double) complex*32 (quadruple)	complex double complex long double complex (It needs complex.h.)
One character	integer*1 byte	char
String	character*n character(len= 10)	char * char[ ]
External function	external	extern
Boolean variables	logical e.g. logical b b = .true.	bool (This requires a header file, stdbool.h.)

Fortran input/output formats and its commands

The input/output commands take three arguments, but the third one is usually optional. For instance:

```
read(number1,number2)
write(number1,number2)
```

Number1 specifies where to input/output. Number2 sets up the format of the input/output. When number1 is 5 for "read" command, it specifies the input from the keyboard. When number1 is 6 for "write" command, the output will be displayed on the screen. However, these numbers can be replaced by asterisk, *, for the standard screen I/O.

For one of the third option is

```
write(*, fmt = '(i5)', advance = "no")
```

This does not go to a new line to output the data. You cannot use * for format option when using advance = 'no'. The "fmt = '(i5)'" specifies the format of the output, which is for a 5-digit integer. (The item, "fmt = ", can usually be omitted.)

The following third option specifies the numbered command when the program is forcibly ended by ctrl+C. This end option is only for the read command. (The key depends on the operating system. It may be ctrl+D or ctrl+Z.)

```
          do while(.true.)
             read(*,*, end = 10)
          enddo
10        write(*,*) "The program is terminated."
```

The following code notifies a user after reading 15 data:

```
          do i=1,15
              read(*,*,end=10) a
          end do
10        write(*,*) "Reading's done."
```

Without a do-loop, you can express the data of a vector array:

```
write(*,*) (x(i), i=1,5)
```

The data above are placed in a line. The following puts the data in the next line:

```
write(*,"(f5.3)",advance="yes") (x(i),i=1,5)
```

The “print” command only gives a screen output. The option of print command is:

```
print ‘format’, ‘variable’
```

For example,

```
print *, i
print “(i5)”, i
print “(i5,i7)”, i, j
print *, “Hello”
print “(‘ The value is ‘, i5)”, i
```

The format code letters are used in the format option as mentioned above. The code letter must be corresponded to the type of values. For an integer, i5 creates a 5-digit space to express or read the data. For a float number, f10.6 means that there is a 10-digit space for the value and 6 digits are kept for the decimal places. If you use the same format with multiple values, it can be made as 3f10.6, which is for 3 different variables.

Code	Description	Example
i	Integer	i5
f	Real, floating point	f10.6
e	Single precision real with exponential notation	e12.10
d	Double precision real with exponential notation	d26.20
a	Character	a15
x	Space	7x
/	Vertical space	/
t	Tab	t12
l	Logical	l4
n(‘*symbol*’)	Repeating a symbol	8(‘-‘)

A file can be specified as follows:

```
open(unit = 8,file = “out.dat”,status = “old”)
```

Then, either “write” or “read” specifies the unit number of the above open command:

```
write(8,*) “a =  “, a
```

For an example of reading a file,

```
open(unit = 7,file = “input.dat”,status = “old”)
```

```
read(7,*) a
close(7)
```

A file is opened, it must be closed with the unit number.

You can make arbitrary output without the open command. For instance,

```
write(10,*) a
```

Then, it creates a new file named as fort.10.

The status option in the open command has following choices:

Options	If there is no file, …	If there is the file, …
Old	*The error occurs.*	*Open the file.*
New	*Create a new one.*	*The error occurs.*
Replace	*Create a new file.*	*The file is replaced.*
Unknown	*Create a new one.*	*Open or overwrite the file*
Scratch	*Create a new one.*	*The error occurs while compiling.*

For closing file, there are status options as follows. The option, "keep", is to keep the generated files and it is the default.

```
close(unit = 7, status = "keep")
```

Then, delete option is to delete the opened file after executing the program.

```
close(unit = 7, status = "delete")
```

The following code lets it skip the first line to read the data:

```
open(11, file = ' test.dat ' )
read (11,'( )')      ! Skip the line
read (11,*) j
close(11)
```

C/C++ input/output formats and its commands

C/C++ general

When input and output data on the command prompt (screen), use printf and scanf commands as follows:

```
#include<stdio.h>
int main( ){
   double f;
   scanf("%lf", &f);
```

```
   printf("The ans is %lf\n", f);
}
```

For the input command (scanf), the address of the variable should be used with &. The items, \n and %lf, are an escape sequence and a specifier, respectively, and the others can be referred to in the tables below.

Format specifiers for C

Specifier	Correspondent type	Description	Example
%c	char	For one character	"%c"
%s	char *	For a string	"%7s", "%-12s"
%d	int	For an integer in decimal	"%5d", "%-2d"
%u	unsigned int unsigned short	For an unsigned integer in decimal	"%2u"
%o	int short unsigned int unsigned short	For an integer in octal	"%08o"
%x	int short unsigned int unsigned short	For an integer in hexadecimal	"%08x"
%f	float	For a real	"%8.6f"
%e	float	For a real with exponential	"%10.7e"
%g	float	For a real with an optimal form	"%g"
%ld	long	For a double precision integer in decimal	"%-12ld"
%lu	unsigned long	For an unsigned double precision integer in decimal	"%08lu"
%lo	long unsigned long	For an unsigned double precision	"%14lo"

		integer in octal	
%lx	long unsigned	For an unsigned double precision integer in hexadecimal	“%10lx”
%lf	Double	For a double precision real	“%26.20lf”
%Lf	Long double	For a quadruple precision real	“%33.32Lf”
%p	Pointer	For printing a pointer value	“%p”

Escape sequences of C

Escape sequence	Representation
\ a	Alert (beep sound)
\ b	Backspace
\ f	Form feed (new page)
\ n	Line feed (new line)
\ r	Carriage return
\ t	Horizontal tab
\ v	Vertical tab
\ \	Backslash
\ ?	Question mark
\"	Double quotation mark
\'	Single quotation mark
\ 0	Null (string terminator)
\ xhh	Hexadecimal bit pattern
\ ooo	Octal bit pattern

When reading or storing data in files, the following procedure is required: First, make a pointer variable by using the FILE struct.

```
FILE *fp;
```

Then open a file to input the variable.

```
fp = fopen(“out.dat”,”w”);
```

The argument, w, indicates writing. The command, fprintf, stores the data in the pointer file.

```
fprintf(fp, "A = %lf. \ n ", a);
```

You can do the same way to read a set of data from a provided file. Declare a pointer and open the correspondent file.

```
FILE *fi;
fi = fopen("input.dat","r");
```

Note that the option, r, represents reading. Then, read the data from the file using fscanf command:

```
fscanf(fi, "%lf", &t);
```

C++ specifics

On the command prompt, use "iostream" for the header file. Commands, "cout" and "cin", are used for output and input, respectively. The following code is an example:

```
#include<iostream>
using namespace std;
int main( ){
   int i;
   cout << "Enter an integer." << endl;
   cin >> i;
   cout << "You entered " << i << "." << endl;
}
```

The "endl" denotes the end of the line and the command feeds a new line.

When using an external file to input data, use "fstream" for the header file. To open a file, use commands ifstream or ofstream for input or output, respectively. For instance,

```
#include<iostream>
#include<fstream>
using namespace std;
int main( ){
   int i;
   ifstream fin("input.d");
   ofstream fout("output.d");
   fin >> i;
   fout << "You entered " << i << "." << endl;
}
```

The input data is stored in the specified file, input.d. The comment and the data are stored in the file named output.d.