

# **Practice of Information Processing**

(IMACU)

## **Second lecture (part 1): Introduction of Lectures/Basic of Program**

---

Makoto Hirota

## ■ Makoto Hirota

- Institute of Fluid Science, Associate Professor
- Office: Institute of Fluid Science, 2nd building, 4th floor 406 (in Katahira campus)
- Tel: 022-217-5251
- E-mail: [makoto.hirota.d5@tohoku.ac.jp](mailto:makoto.hirota.d5@tohoku.ac.jp)  
(Please use e-mail or Google Classroom to contact us)
- Research: theoretical and computational fluid mechanics
  - Stability and control of flows
  - Magnetohydrodynamics (MHD)

## ■ Google Classroom

- [TB14131] Practice of Information Processing (情報処理演習)
- Class code: [g6g4oj2](#)
- Check announcements
- Download slides, sample program sources
  - Slide: [PIP02\\_01\\_2023.pdf](#)
  - Program: [PIP02\\_2023.zip](#)
- Submit assignments
  - Do Exercise 2-1, 2-2 etc. during the class or at home.
  - Upload your source file (\*.c) and expository writing (\*.txt, \*.doc)
  - Explain what you did in arbitrary format. Don't copy!
  - Deadline: 1 week after each class

## ■ Objectives

- Acquire knowledge of programming languages that are necessary for information processing.
- Lectures on basic grammar of C language and simple algorithm design methods, and exercises in programming.

## ■ Goals

- Understand grammar of C language and create simple programs.
- Write programs and execute it in a UNIX-like environment.

## ■ Course requirements

- Review the basic C language grammar learned in previous courses, how to use the computer system, etc.
- Find time to prepare and review, not just during the lecture time. Also, try to find and tackle advanced problems by yourself.

You can't learn programming unless you actually write it.  
It is important for beginners to imitate the code written by experts.

## ■ Policy

- Focus on attendance. Basically, I want to pass the students who attended every time.
- I will add points to your voluntary efforts. In order to take AA, please think that it is necessary to tackle voluntary challenge tasks and self-study tasks.
- No final exam

## <Rating criteria>

### ■ Attendance points (about 60 %)

- However, attendance points will be given by conducting exercises during class and submitting the exercises each time.
- Earlier submission gets higher score.
- Points will be deducted if you do not submit an assignment.

### ■ 2 report assignments (about 40 %)

- Intermediate report 15 points
- Final report 25 points

- ❖ Obvious copies will get minus points.
- ❖ Please attach your own explanation to your report.

- 10/4 (Wed) 14:40-17:50
- 10/11 (Wed) 14:40-17:50
- 10/18 (Wed) 14:40-17:50
- 10/23 (Mon) 14:40-17:50 Makeup class
- ~~■ 10/25 (Wed)~~
- 11/1 (Wed) 14:40-17:50 (mid-term assignments with TA)
- 11/8 (Wed) 14:40-17:50
- ~~■ 11/15 (Wed)~~
- 11/22 (Wed) 14:40-17:50
- 11/30 (Thu) 14:40-17:50 Makeup class

If you can't attend 10/23 and 11/30 for some reason, let me know.

- Review of Unix commands and C programming language
- C language basics
  - Variables and data types
  - How to use the standard I / O functions `printf` and `scanf`
- From human thinking to computer programs
  - Three basic forms of processing flow
    - “Sequence”, “Select”, “Iteration”
    - Flowchart and PAD representation
- Loop structure “Iteration of a certain number of times”
  - `for` statement

- In this exercise, the program is executed by entering text commands from the console. (Command User Interface, CUI).
  - In the exercise, “Powershell”, “terminal in VSC” is used.

## <basic commands ? >

- `cd ~/` : move to home directory (folder)
- `cd ./` : move to current directory (folder)
- `cd ../` : move to the directory (folder) one-level above
- `cd lec02` : move to the directory (folder) named lec02

## – UNIX basic commands

- Move directory: `cd`
- View list of files: `ls`
- View the contents of the file: `less`
- File operations: `mv`, `cp`, `rm`
- Directory operations: `mkdir`, `rmdir`



## ■ How to use commands?

- `man` “command”
  - `man` (the command name you want to know)

## ■ Relative directory

- “.”: indicates the current directory where you are.
- “..”: indicates the directory up one stage.
- “~(tilde)”: Indicating the home directory of login user

例) display the list

```
% ls .  
% ls ..  
% ls ../..
```

## ■ Readline completion function of command input

- If you press the [TAB] key while entering a command, the command name or file name being entered will be completed.
- Completion of commands :
  - Candidates for registered commands are displayed (commands in “path”)
- Completion of file name :
  - Candidates of the files in current directory are displayed
- Use this function to prevent typos in commands and filenames.

## ■ Color coding function for source

```
#include <stdio.h>

int main()
{
    /***** variable declaration *****/
    int x, y;          /* int type */
    int sum, diff, prod; /* int type */
    float quot;        /* real type */

    /***** processing contents *****/

    /* assignment */
    x = 3;
    y = 2;

    /* sum */
    wa = x + y;
    printf("sum=%d\n", sum);

    /* diff */
    diff = x - y;
    printf("diff=%d\n", diff);

    /* prod */
    prod = x * y;
    printf("production=%d\n", prod);

    /* quot */
    if ( y == 0){
        printf("quotient is not available\n");
    }
    else{
```



```
1  #include <stdio.h>
2
3  int main()
4  {
5      /***** variable declaration *****/
6      int x, y;          /* int type */
7      int sum, diff, prod; /* int type */
8      float quot;        /* real type */
9
10     /***** processing contents *****/
11
12     /* assignment */
13     x = 3;
14     y = 2;
15
16     /* sum */
17     wa = x + y;
18     printf("sum=%d\n", sum);
19
20     /* diff */
21     diff = x - y;
22     printf("diff=%d\n", diff);
23
24     /* prod */
25     prod = x * y;
26     printf("production=%d\n", prod);
27
28     /* quot */
29     if ( y == 0){
30         printf("quotient is not available\n");
31     }
32     else{
```

※There are other useful functions if you use the dedicated editor of the program development environment.

- Completion input
- Automatic indent
- Definition call
- Reference search, etc

例) Develop environment

- Eclipse
- Visual Studio (Win)
- Xcode (Mac)
- Emacs (if you configure it)

Recommended

**Visual Studio Code**

- Compatible with Windows / Mac / Linux
- Quick movement
- Supports various languages
- Function extension is possible

# Exercise 2-0 Compile sample program again

12

## ■ 1. Prepare file

- 1) Start and check if PIP is in your desktop directory.
- 2) Make subdirectory 'lec02' in 'PIP' and copy all the file of extracted zip into lec02 by explorer
- 3) Of course you can use linux commands shown right hand side->.

## ■ 2. Open the program by VSC

## ■ 3. Compile

- Use **terminal tab** and **gcc** command

## ■ 4. Execute

- ./sample

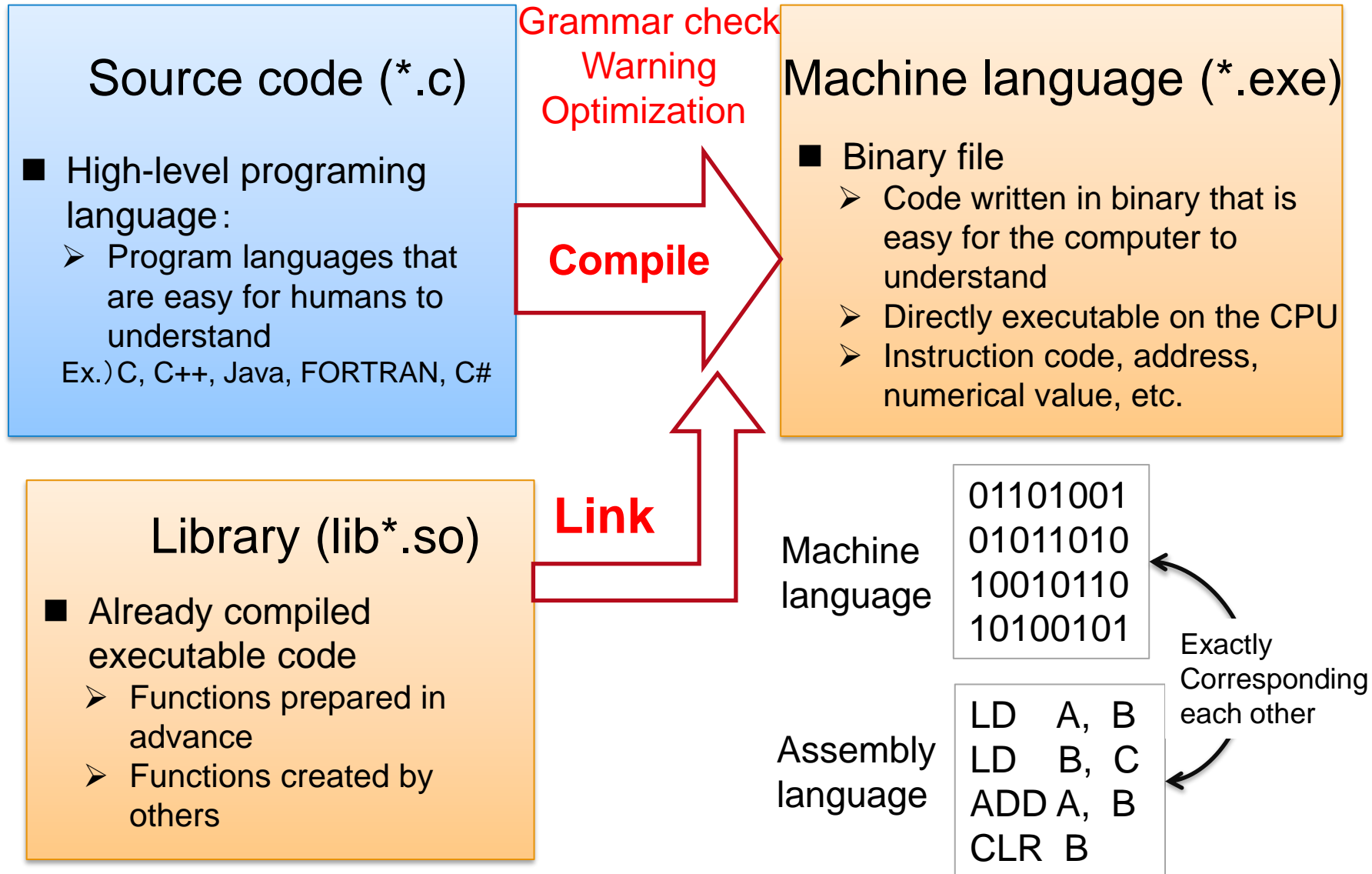
## ■ 5. No submission is required.

- If you cannot do this, let us know it. We will help you.

```
$ cd ./Desktop/PIP
$ cp ../../Downloads/PIP02_* ./
$ ls
PIP02_2023.zip
$ Expand-Archive PIP02_2023.zip .
$ cd lec02
```

```
$ gcc -o sample sample.c
$ ls
sample.c sample
$ ./sample
```

❖ When executing the program that you created, be sure to specify the program from the current directory (./).



## Basic form

```
gcc [option1] [option2] ... C-source-filename
```

(入力例)

```
$ gcc -Wall -o test test.c
```

- [ -o executable-file-name] Specify the generated executable file name

```
$ gcc -o test test.c
```

Option C-source-filename

If you do not specify the executable file name, a file called a.exe will be created.

- [-Wall] Strictly point out the possibility of program errors

```
$ gcc -Wall -o test test.c
```

Option C-source-filename

-Wall is easy to find mistakes in the program, so it is recommended to always add it in this exercise.

※ It works even if the order of options and C file name is changed, but it leads to confusion. So, enter in this order in this lecture.

# Sample program: sample.c

15

## Sample program

```
#include <stdio.h>
```

Header

```
int main()  
{
```

Main function

```
/**** declaration ****/
```

```
int x, y;           /* integer type */  
int sum, diff, prod; /* integer type */  
float quot;         /* real type */
```

```
/**** process ****/
```

```
/* assignment */  
x = 3;  
y = 2;
```

```
/* sum */  
sum = x + y;  
printf("sum=%d¥n", sum);
```

```
/* difference */  
diff = x - y;  
printf("diff=%d¥n", diff);
```

```
/* product */  
skip
```

```
/* quotient */  
skip
```

```
return 0;
```

```
}
```

## Processing flow of main function

### (1) Variable declaration

Define variables and  
their types to use

### (2) Processing contents

Describe the processing  
content you want to  
execute  
(Executed in order from  
the top)

In the main function, the program ends  
when it reaches 'return 0;'

```
#include <stdio.h> Header
```

```
int main(void) Main function  
{
```

```
    /*** Variable  
    declaration***/
```

```
    /*** Processing  
    contents***/
```

```
    return 0;
```

```
}
```

## ■ Header

- `#include`: Specify information (header file) to call a function created by another person
- `stdio.h` (Standard Input / Output) is required to use the standard input / output functions (printf function in sample.c)
- In addition, `#define (macro)`, common variables (global variables), etc. are also entered here.
- The part starting with `#` is called the preprocessor.

## ■ Main function

- The main function is the first function to be executed when the program is started, and it is a necessary function.
- `int main (void) {... return 0;}` is the standard form.
  - ❖ In some textbooks, "int" and "return 0;" are also omitted, but it is recommended that you first write the formal notation without omitting it.
- At first, you can create a working program by rewriting only the "variable declaration" and "processing contents" of template.c.

## 【Important】Insert comment (`/* comment */`)

Statements in between `/*` and `*/` are recognized as comment statements and do not affect compilation. In order to write an easy-to-understand program, it is important to frequently write comments.

# Important: Coding style (indentation)

17

We May call  
“code”=“a program”  
“coding”=“writing a program”  
“style”=“notation”

To write easy-to-read code  
It is necessary to describe program  
▪ in a general style, and  
▪ consistently.  
This makes it easier to find bugs

## ■ General style rules

1) Indent each block (indentation)

```
for(i=1; i<=1000; i++){  
    if(i%3 == 0){  
        sum += i;  
        printf("%d\n", i);  
    }  
}
```

➡ To indent  
Hit the "TAB key"

❖ If you use a program editor, the indent will be inserted automatically, but if the indent collapses during editing, be sure to correct it.

2) Put brackets { } in style: Two different ways

### K&R style

(Format used in classic C books)

```
for(i=1; i<=1000; i++){  
    if(i%3 == 0){  
        sum += i;  
        printf("%d\n", i);  
    }  
}
```

Put "{" at the end  
in this style

Standard in UNIX  
kernel and Java  
language

Save  
space

### BSD/Allman style

(Format used in C Bible)

```
for(i=1; i<=1000; i++)  
{  
    if(i%3 == 0)  
    {  
        sum += i;  
        printf("%d\n", i);  
    }  
}
```

Put "{" at the beginning  
in this style

Standard in the  
environment of  
Microsoft system  
e.g. C # etc.

Easy to read  
Because the  
block is clear



# What is variables?

18

- Variables are **boxes** for holding values in your program ❖ To be more precise,  
Box = memory area allocated for variable
  - The variable name is the name of the box
  - Declare the type of box (= variable type) to be used according to the range of numerical values to be handled
  - It is uncertain what is in the box, Immediately after the declaration.  
(Be sure to explicitly substitute some value before using it = initialization)

```
#include <stdio.h>
```

```
int main(){
```

```
    /**** variable declaration****/  
    int x, y;           /* int type */  
    int sum, diff, prod; /* int type */  
    float quat;        /* real type */
```

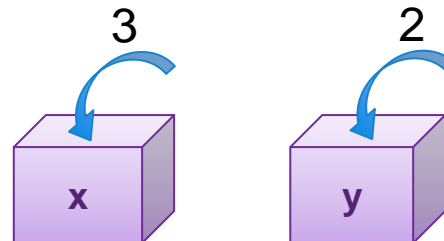
```
    /**** processing contents****/  
    /* assignment */  
    x = 3;  
    y = 2;  
  
    return 0;
```

```
}
```



5 boxes of int type  
1 box of float type

} Prepare



**“=” means  
“assignment”**

❖ Use “==” for the equality  
“=” in mathematics

## ■ There are three main types

- integer type
  - Example: 10, 123, -10,...
- Real type (floating point type)
  - Example: 1.05, 3.14, -12.34,...
- Char type
  - Example: 'A', 'b', 'C', '#',
    - ❖ The characters are the same as the integer type because the numbers are read in a special table.



In a computer, they are all the same set of digital (0 and 1)



1 byte = 8 bit

The range and the interpretation of this 0,1 block changes depending on the type specification.

Bit: 1 digital value (0 or 1)

Byte: An information unit consisting of 8 bits  
(256 different values can be expressed)

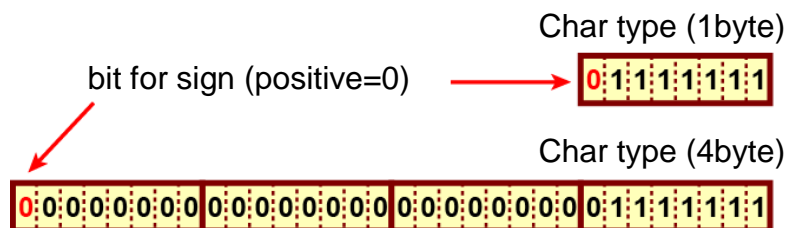
## ■ Why do we need to specify a data type?

- To determine the interpretation of the digital set (integer / real number / character)
- To change the size (number of digits) of the set by specifying the type
  - (= The range and accuracy of numbers that can be expressed changes)
- char: 1 byte, int: 4 byte, double: 8

## 20

Data type		Byte	Range of variables
<b>char</b>	Character type	1	-128~127 (If you use the ASCII code table, it will be treated as a character.)
<b>short</b>	Integer type	2	-32768 ~ 32767 (±30 thousand)
<b>int</b>	Integer type (32bit CPU)	4	-2147483648 ~ 2147483647 (±billion ) (16bit CPU: 2byte)
<b>long</b>	Long integer type (32bit CPU)	4	-2147483648 ~ 2147483647
<b>long</b>	Long integer type (64bit CPU)	8	-9223372036854775808~9223372036854775807 ("long long" type is 8digit integer, 32bit CPU environment)
<b>float</b>	Single precision real type	4	-3.40282 × 10 <sup>38</sup> ~3.40282 × 10 <sup>38</sup> (Effective digit of approx. 7)
<b>double</b>	Double precision real type	8	-1.79769 × 10 <sup>308</sup> ~1.79769 × 10 <sup>308</sup> (Effective digit of approx. 14)

$$127 = 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0$$



※ The size of the data type depends on the execution environment (CPU). Normally, you don't need to be aware of it because the compiler has a definition that matches the CPU.

The significant digit also changes depending on whether the most significant bit is treated as a sign.

unsigned int type: 0 ~ 4294967295 (32bit)

Sign is not used

## ■ Characters available for variable name

- numbers (0~9), alphabet (A~Z, a~z), underline (\_)

## ■ Restriction

- NG for the name start with numbers: **Variable name start with number should not be defined**
- Number of characters: less than 33 chars
- Command words used in syntax and type names are defined as "reserved words" and cannot be used as variable names.

Example of reserved words

```
auto, break, case, char, const, continue, default, do, double, else, enum, extern,  
float, for, goto, if, int, long, register, return, short, signed, sizeof, static,  
struct, switch, typedef, union, unsigned, void, volatile, while
```

## ■ Tips

- Give a clear name to the variable name as to what kind of value is stored.
  - Ex) score, or points etc. for score of tests
- As a general rule, variables start with a lowercase letter (to distinguish them from the uppercase constant representation (MACRO)).
- The meaning becomes clear when words are connected.
  - The way using underline: x\_axis, y\_axis, etc.
  - The way capitalize the first word after the second word: arrayName, arrayScore, etc. : xAxis, yAxis etc

❖ **Emphasis on readability over writability**

# Standard output function: printf

22

```
printf ("format", argument1, argument2, ...);
```

- A function that outputs a "formatted expression" as a character string to standard output (= console)
  - If you put "%" (specifier)" in the format expression, the argument value will be inserted.
  - If there are multiple "%" (specifiers)", they are inserted according to the order of the arguments.
  - The number of "%" (specifier)" and the number of arguments must match.

specifier	type	explanation	example to use
%c	char	Print 1 character	"%c"
%s	char * (Pointer to strings)	Print strings	"%8s" (8 digit width), "%-10s" (Left justified 10 digit width)
%d	int, short, etc.	Print integer by decimal	"%-2d" (Left justified 2 digit width), "%03d" (3 digit width with 0 for blank)
%o	int, short, etc.	Print integer by octal (8)	"%06o" (6 digit width with 0 for blank)
%x	int, short, etc.	Print integer by hexadecimal (16)	"%04x" (4 digit width with 0 for blank)
%f	float	Print real number	"%5.2f" (Display with 5 digits including a decimal point, 2 digits after the decimal point)
%e	float	Print real number in exponential notation	"%5.3e"
%ld	long	Print long integer by decimal	"%-10ld" (Left justified 10 digit width)
%lf	double	Print double precision real number in exponential	"%8.3lf" (Display with 8 digits including a decimal point, 3 digits after the decimal point)

# Formatted Input function: scanf

23

```
scanf ( "format" , &argument1 , &argument2 , ... )
```

- A function that reads a value from standard input (= keyboard) according to the "format expression"
  - Substitute the "% expression" part in the "format expression" into the argument variable (specified by the pointer)
  - If there are multiple "% expressions", they are inserted according to the order of the arguments.
  - The number of "% expressions" and the number of arguments must match.

## Example

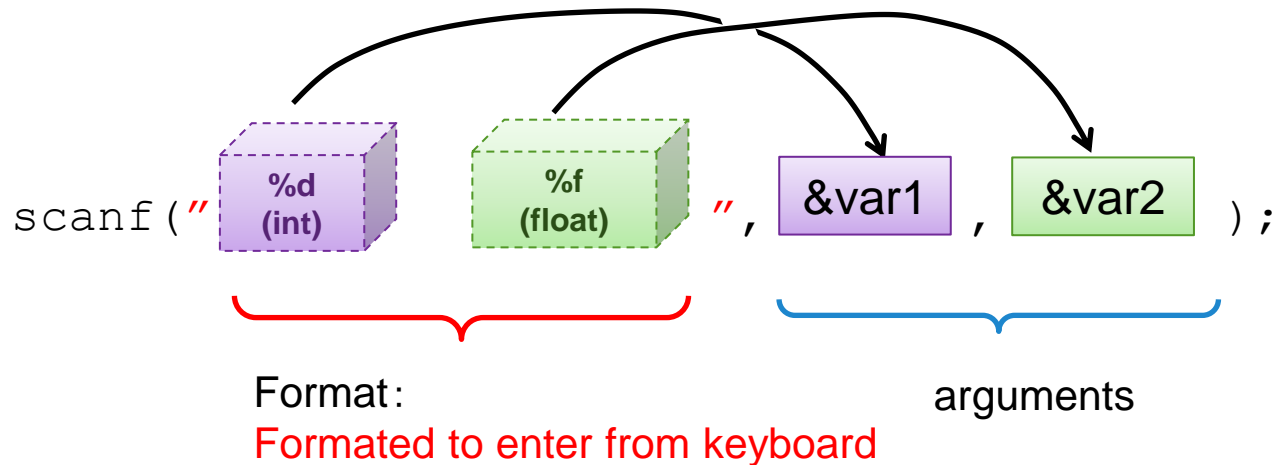
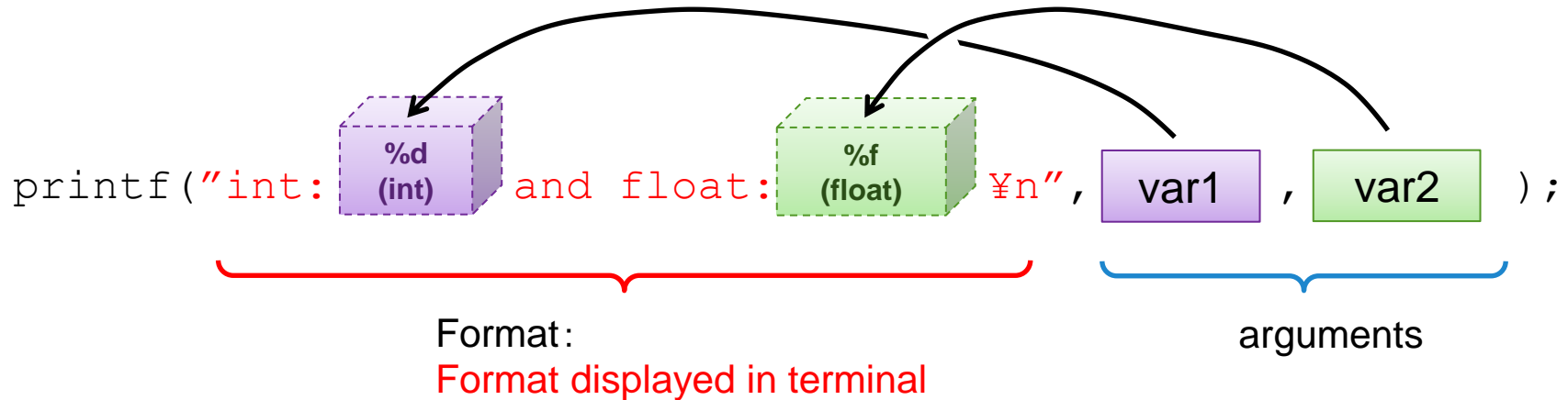
```
char operator;  
float value;  
  
scanf ("%c %f", &operator, &value)
```

We need to specify the pointer to variables, and so "&" is required in front of the name of variable

Specifier	Types	explanation
%c	char	Input one character
%s	char * (pointer to strings)	Input strings
%d	int, short など	Input integer
%o	int, short など	Input octal integer
%x	int, short など	Input hexagonal integer
%f	float	Input real number
%ld	long	Input long integer
%lf	double	Input double precision real number

# Images for printf and scanf

24



Arrange the same number of arguments so that they correspond to the order of the "% expression" described in the format part.

# Difference in between printf and scanf

25

The printf function prints the "contents" of a variable according to a format expression  
The scanf function specifies the "storage destination" of the variable according to the format expression.

## Difference in arguments given

```
printf ( "format", arg1, arg2, ... ) ;
```

Arguments can be variable, calculation results

```
scanf ( "format", &arg1, &arg2, ... ) ;
```

Pointer address operator "&" is required

We will learn the meaning of this at next lectures

## Difference in format

Data type		Size	Format for printf	Format for scanf
<b>char</b>	Character type	1 byte	"%c"	"%c"
<b>int</b>	Integer type	4 byte	"%d"	"%d"
<b>long</b>	Long int type	4 byte	"%d"	"%ld"
<b>float</b>	Single precision floating real	4 byte	"%f"	"%f"
<b>double</b>	Double precision floating point real	8 byte	"%f"	"%lf"

Note that scanf changes the format expression depending on the size of the variable type



```
#include <stdio.h>

int main()
/* **** variable declaration ****/
int int_num = 1234;
double real_num =123.456;

/* processing contents****/
printf("d      |%d¥n", int_num);
printf("8d     |%8d¥n", int_num);
printf("-8d    |%-8d¥n", int_num);
printf("+8d    |%+8d¥n", int_num);
printf("08d    |%08d¥n", int_num);

printf("f      |%f¥n", real_num);
printf("12f     |%12f¥n", real_num);
printf("9.2f    |%9.2f¥n", real_num);
printf(".1f     |%.1f¥n", real_num);
return 0;
}
```

## Example of results

```
d      |1234|
8d     |1234|
-8d    |1234|
+8d    |1234|
08d    |00001234|
f      |123.456000|
12f     |123.456000|
9.2f    |123.46|
.1f     |123.5|
```

← If nothing is specified, it will be displayed in the required width

← Displayed in 8 character width

← (-) is left-justified display

← (+) is signed and right-justified

← 8 characters wide, 0 is displayed if blank

← Displayed in standard floating point width

← Display with 12 digits including a decimal point

← 9 digits with a decimal point, 2 digits after the decimal point

← Required width for integer part, 1 digit after decimal point

- ❖ The "¥ n" at the end of the format expression is a special character that represents a line break.
- ❖ In Linux, it is displayed as "\n" (backslash).

- Compile key\_input.c , and check the results
- Change to a program key\_input2.c that supports decimal numbers

```
$ gcc -Wall -o key_input key_input.c
$ ./key_input
```

```
#include <stdio.h>

int main()
{
    /***** variable declaration *****/
    int x, y;          /* integer type */

    /***** processing contents*****/
    printf("Input two numbers: ");

    /* keyboard input */
    scanf("%d,%d", &x, &y);

    /* display the input values */
    printf("x=%d\n", x);
    printf("y=%d\n", y);

    return 0;
}
```

- Enter two integers and check the results
  - 10 20
- Check what happens if you enter two decimals



Change to a program that supports decimal numbers  
Let's save it as key\_input2.c

# Exercise 2-2: Formatting the printf function

28

## < form.c >

```
#include <stdio.h>

int main()
{
    /***** variable declaration ****/
    char x;

    /* processing contents****/
    x = 75;

    printf("%d\n", x);
    printf("%x\n", x);
    printf("%o\n", x);
    printf("%c\n", x);
    return 0;
}
```

- Write the following program based on template.c and check the execution result.
- Filename: form.c
- Compile:

```
$ gcc -Wall -o form form.c
$ ./form
```

```
$ gcc -o form form.c
```

```
$ ./form
```

```
75      ← decimal(10)
```

```
4b      ← hexadecimal(16)
```

```
113     ← octal(8)
```

```
K       ← ASCII char
```

`char` is a data type that represents one character, but its substance is a one-byte number. In `printf`, we select how to display the number by specifying the format.

※ When you want to assign a character to `char` type

```
char c;
```

```
c = 'K'; ←single quote(')
```

There is no distinction between letters and numbers in the contents of integers.

The output format is determined by the variable type

ASCII character code:

8-bit character conversion table  
75(decimal)=0100 1011(binary)

ascii character table

7	0	0	0	0	0	0	0	0
6	0	0	0	0	1	1	1	1
5	0	0	1	1	0	0	1	1
4	0	1	0	1	0	1	0	1
3	0	1	2	3	4	5	6	7
2	0	0	0	0	0	0	0	0
1	0	0	0	1	1	1	1	1
0	0	0	1	2	3	4	5	6
0	0	1	0	4	5	6	7	8
0	1	0	0	4	5	6	7	8
0	1	0	1	5	6	7	8	9
0	1	1	0	6	7	8	9	10
0	1	1	1	7	8	9	10	11
1	0	0	0	8	9	10	11	12
1	0	0	1	9	10	11	12	13
1	0	1	0	10	11	12	13	14
1	0	1	1	11	12	13	14	15
1	1	0	0	12	13	14	15	16
1	1	0	1	13	14	15	16	17
1	1	1	0	14	15	16	17	18
1	1	1	1	15	16	17	18	19

11 is written as "b"  
in hexadecimal

Codes in 0 and 1 column and DEL are called control code

# Please try exercises 2-1 and 2-2

30

- If you finish it, have a break and come back to the latter half of lecture.
- See you later!