Practice of Information Processing

（IMACU）

Second lecture：
Introduction of Lectures/Basic of Program

Makoto Hirota

About lecturer

- 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

    （Please use e-mail or Google Classroom to contact me）
  - Research: theoretical and computational fluid mechanics
    - Stability and control of flows
    - Drag reduction of wing
    - Space and fusion plasmas

Lecture materials

- Google Classroom
  - [TB14131] Practice of Information Processing (情報処理演習)
  - Class code: f724i3w
  - Check announcements
  - Download slides, sample program sources
    - Slide: PIP02_01_2024.pdf
    - Program: PIP02_2024.zip
  - Submit assignments

Introduction of Lectures

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 write it by yourself.
“Learn by doing”

Syllabus

1. Guidance and introductions to the C programming language
2. Loop iterations
3. Conditional branches
4. Standard inputs and outputs
5. Arrays (1)
6. Arrays (2)
7. Functions, scope of variables, and recursive calls 1
8. Functions, scope of variables, and recursive calls 2
9. Data types, structures, and unions
10. Pointers (1)
11. Pointers (2)
12. File inputs and outputs
13. File inputs and outputs
14. Summaries and practices (1)
15. Summaries and practices (2)

Grade evaluation policy

- Attendance and Exercise submission (about 60 points)
  - Do Exercise 2-1, 2-2 etc. during the class or later at home.
  - Some of them will be requested to submit.
    - Upload your source file (*.c) and its execution result (outputs, screenshots)
    - Optionally, you can address related/in-depth tasks voluntarily and submit them as well. You might get bonus score (1~5 points in total) if they are good and appropriate.
    - Deadline: 1 week after each class
  - You'll get at least the grade "C" if you attend all classes and submit all Exercises.
- 2 report assignments (about 40 points)
  - Mid-term assignment: 15 points
  - Final assignment: 25 points
  - will be announced at a later date.
  - If you are aiming for the grade "AA", get a high score on them.

❖ Don't copy! Obvious copies will get minus points.

Schedule

- 10/2 (Wed) 14:40-17:50 (Setup)
- 10/9 (Wed) 14:40-17:50 (Introduction⇒Lecture)
- 10/16 (Wed) 14:40-17:50
- 10/23 (Wed) 14:40-17:50 (mid-term assignments)
- 10/30 (Wed) 14:40-17:50
- 11/6 (Wed) 14:40-17:50
- 11/13 (Wed) 14:40-17:50 (final assignments)
- 11/20 (Wed) 14:40-17:50 No class (spare)

Contents of today's lecture

- Review of Unix commands and C programing 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

Let's review how to use terminal

- In this exercise, the program is executed by entering text commands from the console.（Command User Interface, CUI）.
  - In the exercise, “Powershell” or “terminal in VSC” is used.
  - Example:
    - `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 text file: `less`
    - File operations: `mv, cp, rm`
    - Directory operations: `mkdir, rmdir`

Tips for UNIX commands

- 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 one level above.
  - “~(tilde)”:　Indicating the home directory of login user

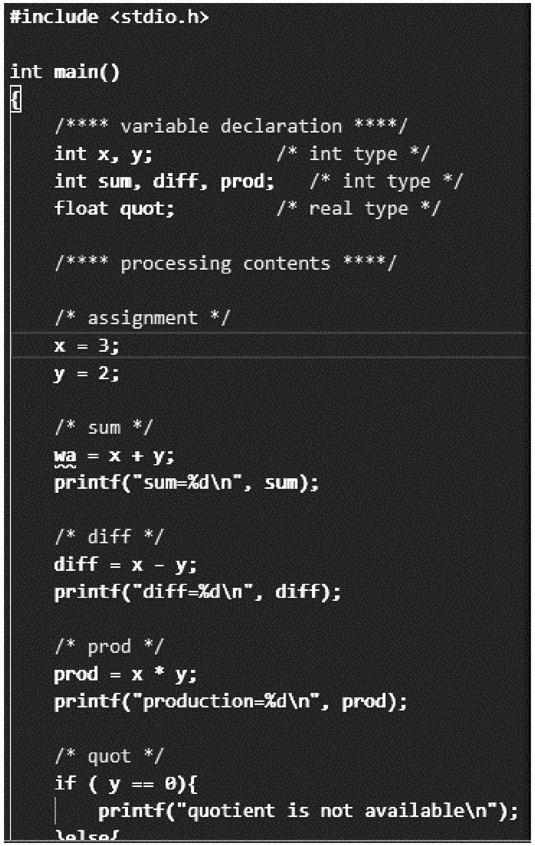
例）display the list

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

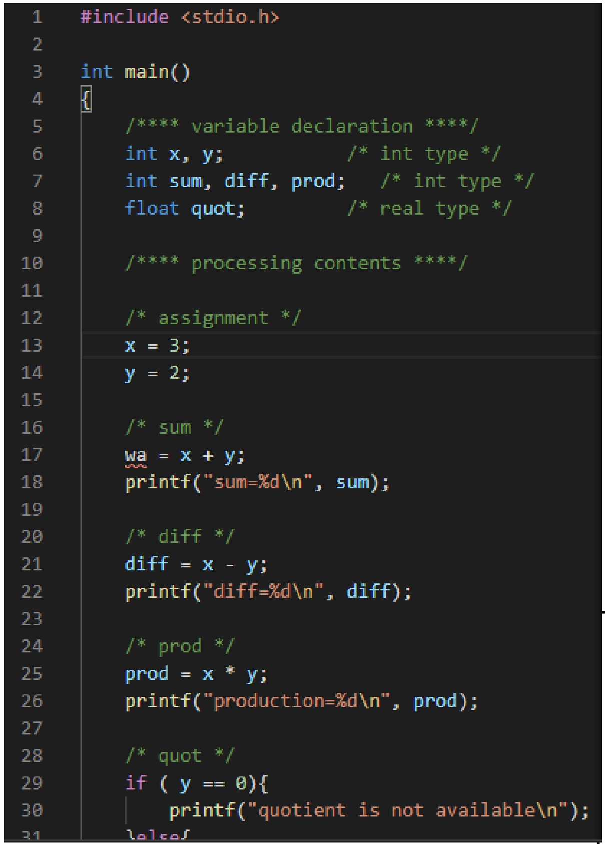
- Readline completion function of command input
  - If you press the [TAB] key while entering a command, the command name or file name being entered will be completed.
  - Completion of commands：
    - Candidates for registered commands are displayed (commands in “path”)
  - Completion of file name：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

Editor for programing

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{
```



```
#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{
```

※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

- 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) You can use Unix commands shown on the right.

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

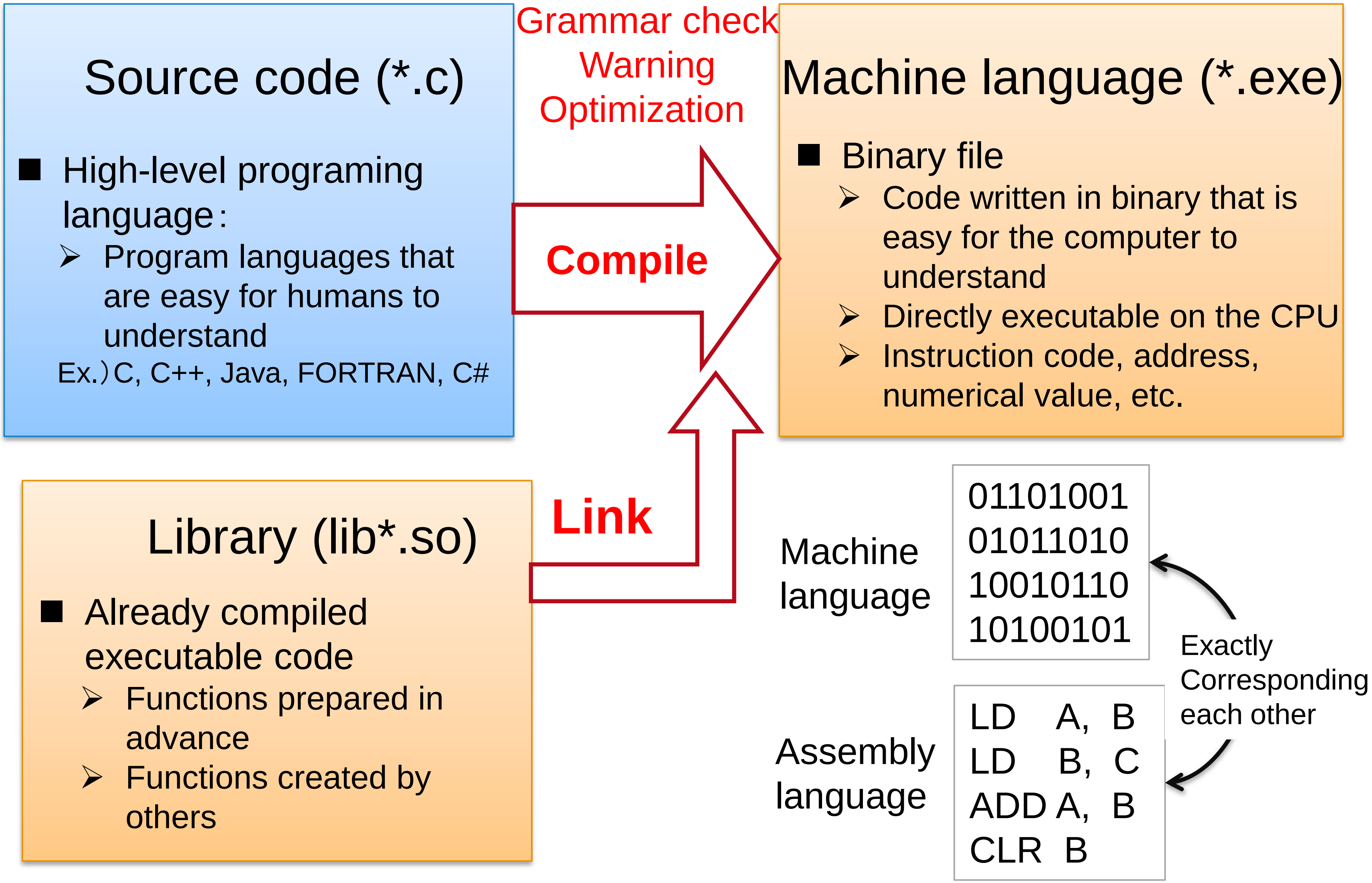
- 2.Open the program by VSC
- 3. Compile
  - Use terminal tab and gcc command

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

- 4. Execute
  - ./sample

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

Role of Compiler



Source code (*.c)

- High-level programing language：
  - Program languages that are easy for humans to understand

Ex.）C, C++, Java, FORTRAN, C#

Grammar check
Warning
Optimization

Compile

Machine language (*.exe)

- Binary file
  - Code written in binary that is easy for the computer to understand
  - Directly executable on the CPU
  - Instruction code, address, numerical value, etc.

Library (lib*.so)

- Already compiled executable code
  - Functions prepared in advance
  - Functions created by others

Link

Machine language

```
01101001
01011010
10010110
10100101
```

Assembly language

```
LD   A, B
LD   B, C
ADD A, B
CLR  B
```

Exactly Corresponding each other

How to compile

Basic form

gcc [option1] [option2] ･･･ C-source-filename

（Example）

```
$ 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

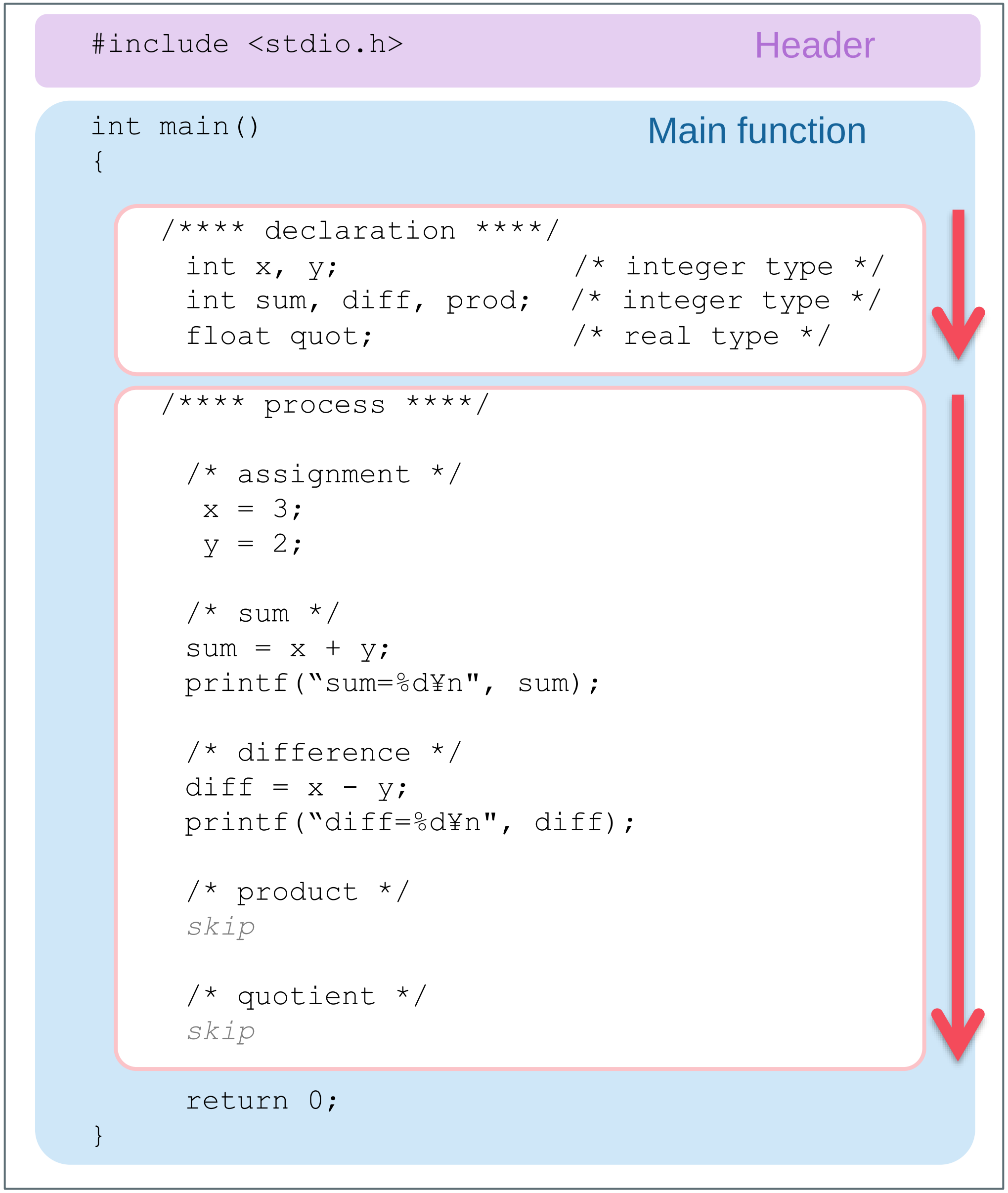
```
$ gcc -Wall -o test test.c
```

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.

Sample program: sample.c



Header

Main function

```
#include <stdio.h>

int main()
{
  /**** 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

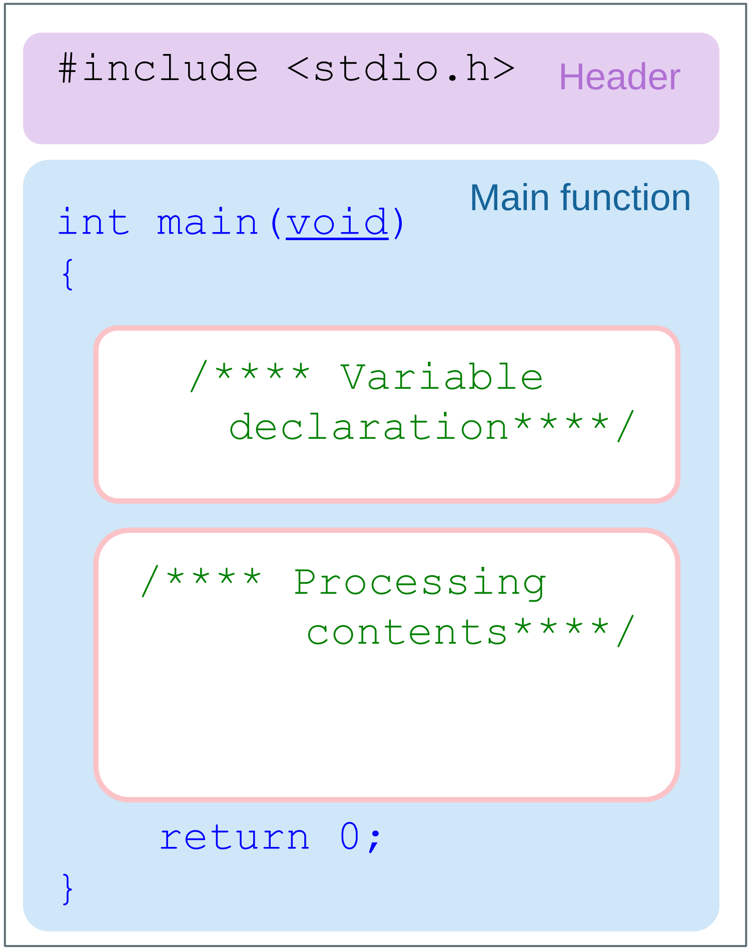
Define variables and their types to use

（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;’

Basic formats of C-source file: templete.c



```
#include <stdio.h>                Header

                             Main function
int main(void)
{

      /**** 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.

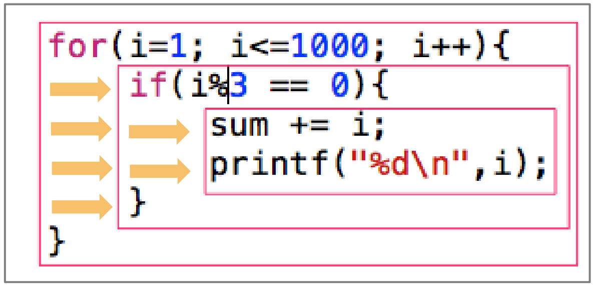
Coding style (indentation)

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)



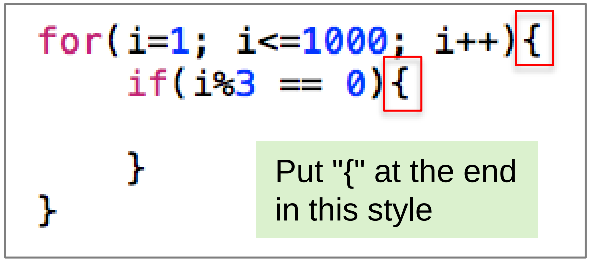
```
for(i=1; i<=1000; i++){
    if(i%3 == 0){
        sum += i;
        printf("%d\n",i);
    }
}
```

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）



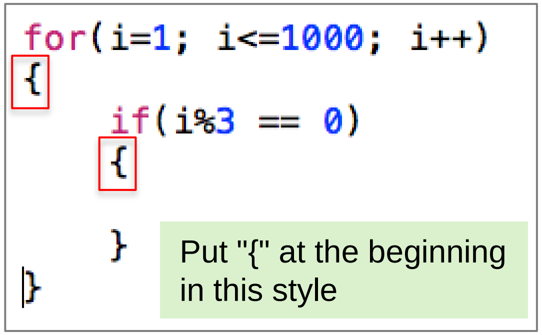
```
for(i=1; i<=1000; i++){
    if(i%3 == 0){

    }
}
```

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)
    {

    }
}
```

Put "{" at the beginning in this style

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

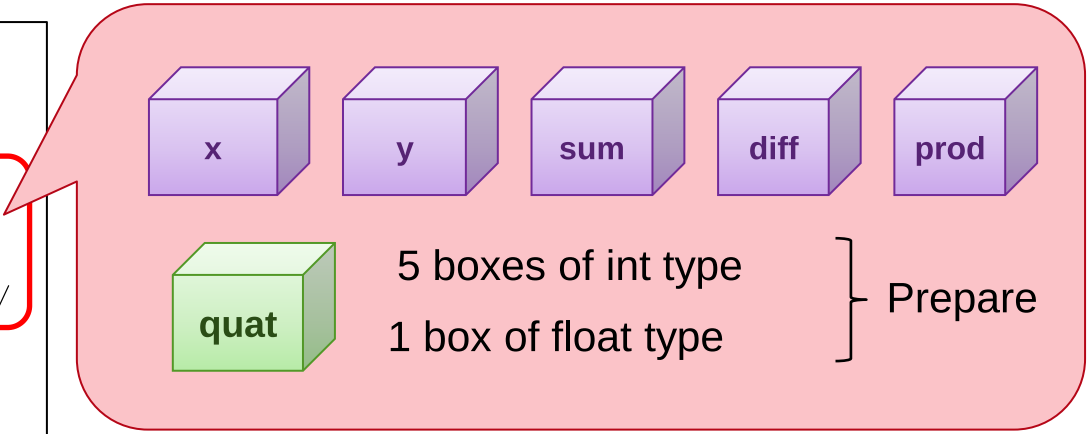
Easy to read Because the block is clear

What is variables?

- Variables are boxes for holding values in your program
  - 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)

❖ To be more precise,
Box = memory area
allocated for variable

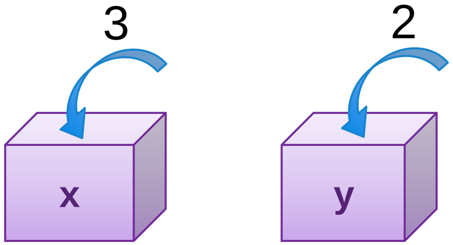
```
#include <stdio.h>

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

Data type handled by C language

- 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.
- 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

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

0	0	0	0	0	1	1	0	…	0	0	1	0	1	0	1	0

1 byte = 8 bit

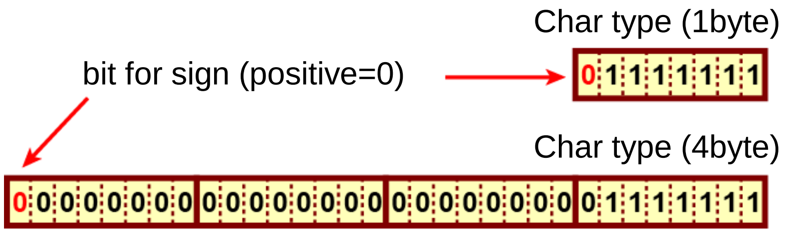
The range and the interpretation of this 0,1 block changes depending on the type specification.

Bit: 1 digital value (0 or 1)
Byte: An information unit consisting of 8 bits
(256 different values can be expressed)

Representative variable type（data type）

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

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



Char type (1byte)

bit for sign (positive=0)

0 1 1 1 1 1 1 1

Char type (4byte)

0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 0 1 1 1 1 1 1 1

※ 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

How to name the variable

- 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, dougle, else, enum, extern,
float, for, goto, if, int, long, register, return, short, signed, sizeod, 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, yAxiis　etc

❖ Emphasis on readability over writability

Standard output function: printf

```
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 justified10digit width）
%d	int, short, etc.	Print integer by decimal	“%-2d”（Left justified2digit 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 justified10digit 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

```
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.

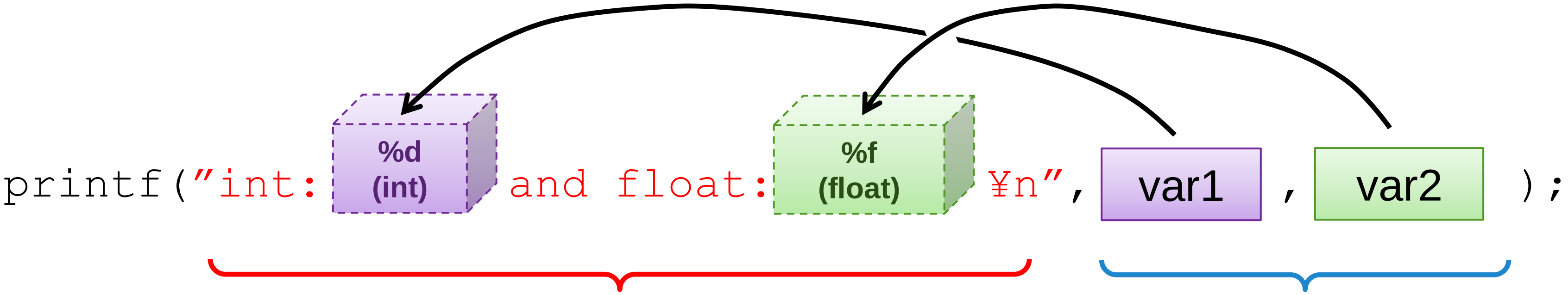
Example

```
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

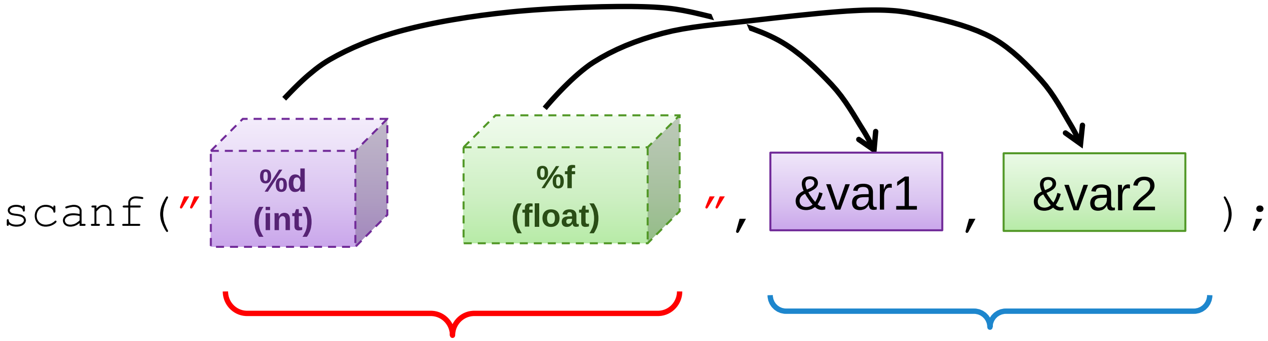


printf(”int: %d (int) and float: %f (float) ¥n”, var1 , var2);

Format:

Format displayed in terminal

arguments



scanf(” %d (int) %f (float) ”, &var1 , &var2);

Format:

Formated to enter from keyboard

arguments

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

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

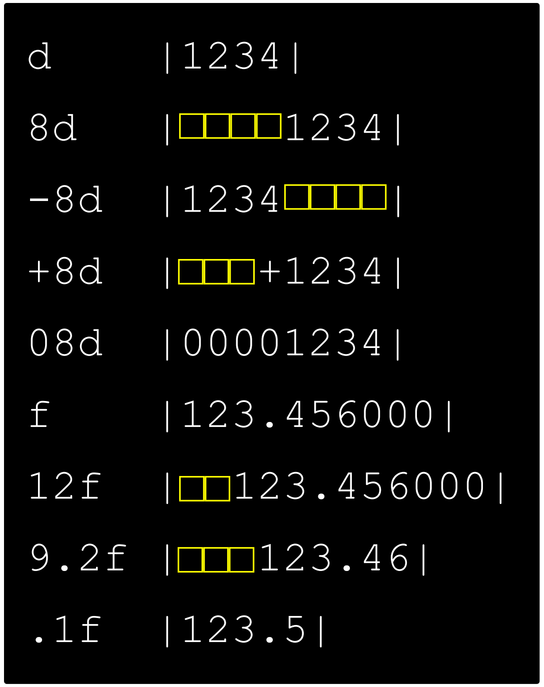
Data type		Size	Format for printf	Format for scanf
char	Character type	1 byte	″%c″	″%c″
int	Integer type	4 byte	″%d″	″%d″
long	Long int type	4 byte	″%d″	″%ld″
float	Single precision floating real	4 byte	″%f″	″%f″
double	Double precision floating point real	8 byte	″%f″	″%lf″

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

Example: Example of specifying a format expression printf_format.c

Example of results

```c
#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  |%-d¥n", int_num);
    printf("+8d  |%+d¥n", int_num);
    printf("08d  |%0d¥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;
}
```



```
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).

Exercise 2-1: sample of scanf：key_input.c

- 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

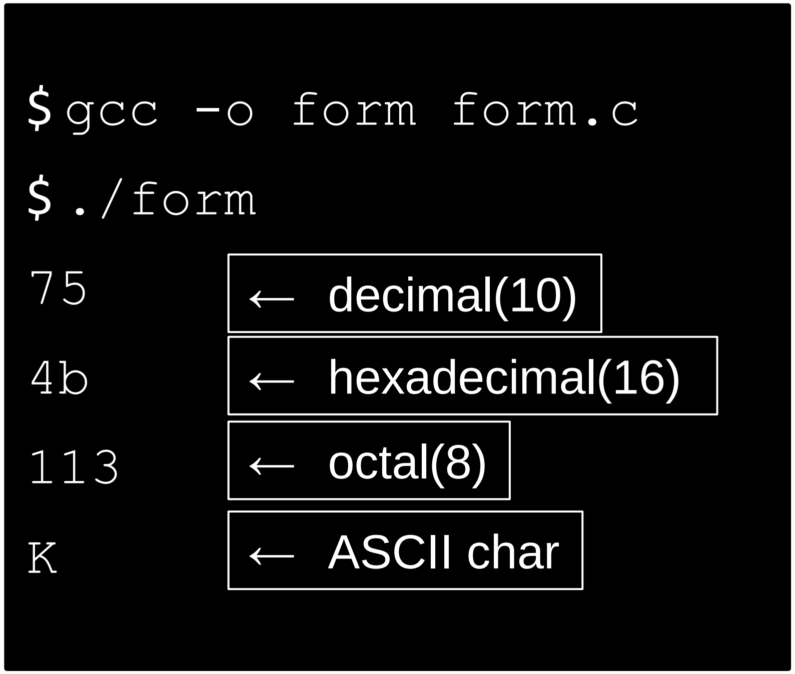
< form.c >

```
#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 templete.c and check the execution result.
- Filename：form.c
- Compile:

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

Output results



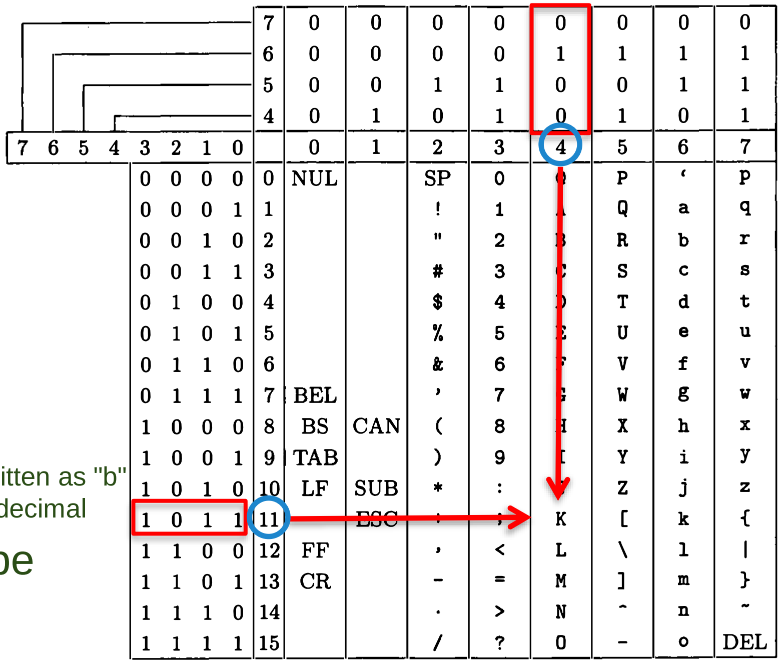
```
$ gcc -o form form.c
$ ./form
75        ← decimal(10)
4b        ← hexadecimal(16)
113       ← octal(8)
K         ← ASCII char
```

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.

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
7 6 5 4 3	2	1	0		0	1	2	3	4	5	6	7
0	0	0	0	0	NUL		SP	0	@	P	‘	p
0	0	0	1	1			!	1	A	Q	a	q
0	0	1	0	2			"	2	B	R	b	r
0	0	1	1	3			#	3	C	S	c	s
0	1	0	0	4			$	4	D	T	d	t
0	1	0	1	5			%	5	E	U	e	u
0	1	1	0	6			&	6	F	V	f	v
0	1	1	1	7	BEL		’	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	TAB		)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11		ESC	+	;	K	[	k	{
1	1	0	0	12	FF		,	<	L	\	l	|
1	1	0	1	13	CR		-	=	M	]	m	}
1	1	1	0	14			.	>	N	^	n	~
1	1	1	1	15			/	?	O	_	o	DEL

11 is written as "b" in hexadecimal

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

※ When you want to assign a character to char type

```
char c;
c = ’K’;  ←single quote(’)
```

Please try exercises 2-1 and 2-2

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

From human thinking to computer programs

- Humans can describe the flow of processing in words.

  Example) Grade evaluation policy

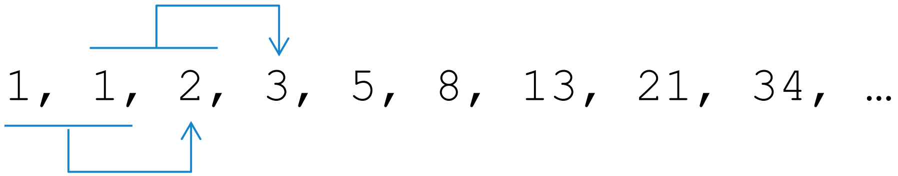
  Sometimes, it is not rigorous but ambiguous.

- How do we get the computer to perform the process?
  - The structure of the processing flow must be defined strictly.

    Flowchart
  - Process contents must be translated into mathematical forms.
  - Afterwards, we can start writing a program.
- There can be many programs that do the same job.
  - Depends on the programmer’s skill.
  - Good codes / Bad codes

Example: Fibonacci sequence (1)

- The Fibonacci sequence is a sequence in which the previous two numbers are added to obtain the next number. Here, the first and second numbers are both 1. Find the *n*th Fibonacci number



1, 1, 2, 3, 5, 8, 13, 21, 34, …

<Formula> (iterative processing)

Initial condition

$$F_1 = 1$$

$$F_2 = 1$$

Repeats

first $F_3 = F_1 + F_2$

second $F_4 = F_2 + F_3$

⋮ ⋮

(n-2)th $F_n = F_{n-2} + F_{n-1}$

<program example> fibo1.c

```
/**** variable declaration ****/
int f1, f2, f3;
int i, n;

 /**** processing contents****/
f1 = 1;
f2 = 1;
n = 8;

for(i=0;i < n-2; i++){
    f3 = f1 + f2;
    f1 = f2;
    f2 = f3;
}
printf("n=%d: fn=%d¥n", n, f3);
```

Initial condition

(the case of n=8)

repeating((n-2)times)

Example: Fibonacci sequence (2) (3)

<Math>（recursive expression）

$$F_n = F_{n-2} + F_{n-1}$$

where,

$$F_1 = 1, F_2 = 1$$

Recursive expression: A reference to the description itself appears, when describing something. (Here, "F" is used to define "F")

<program example>

fibo2.c

```
int fibonacci(int n)
{
    int fn;

    if (n == 1 || n == 2){

        fn = 1;

    }else{

        fn = fibonacci(n-2) + fibonacci(n-1);

    }

    return fn;
}
```

functionalize F

recursive call
（Calling itself in the function）

※recursive expression will be explained in the following lectures

<Math>(Expression by general term)

The general term of the Fibonacci number is expressed by the following equation

$$F_n = \frac{1}{\sqrt{5}}\left\{\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right\}$$

<program example>

fibo3.c

```
    /**** variable declaration****/
    int fn, n;

     /**** processing contents****/
    n = 8;
    fn = round((pow((1+sqrt(5))/2, n)
            - pow((1-sqrt(5))/2, n))/sqrt(5));

    printf("n=%d: fn=%d¥n", n, fn);
```

Use mathematical functions

round(): Round to the nearest whole number

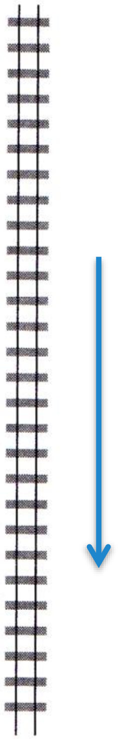
sqrt(): Square root

pow(a, n): Exponentiation （a^n）

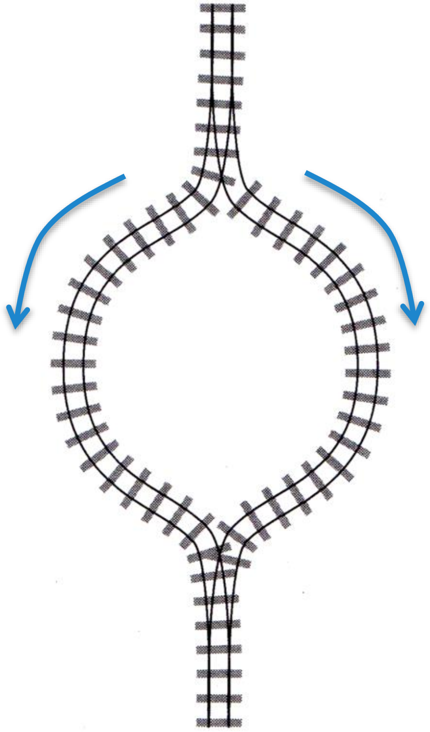
※−lm option is required for compilation

Basic structure of processing flow

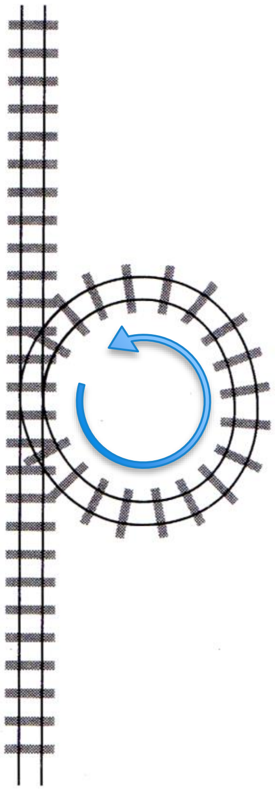
- There are only three basic forms of "processing flow" in a program



Sequence



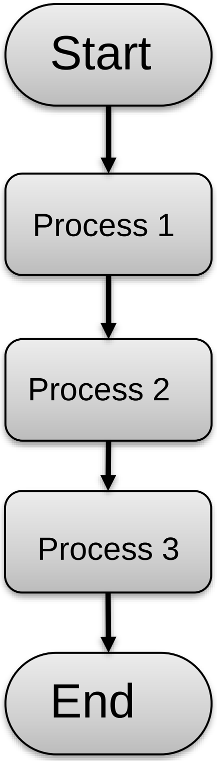
Selection



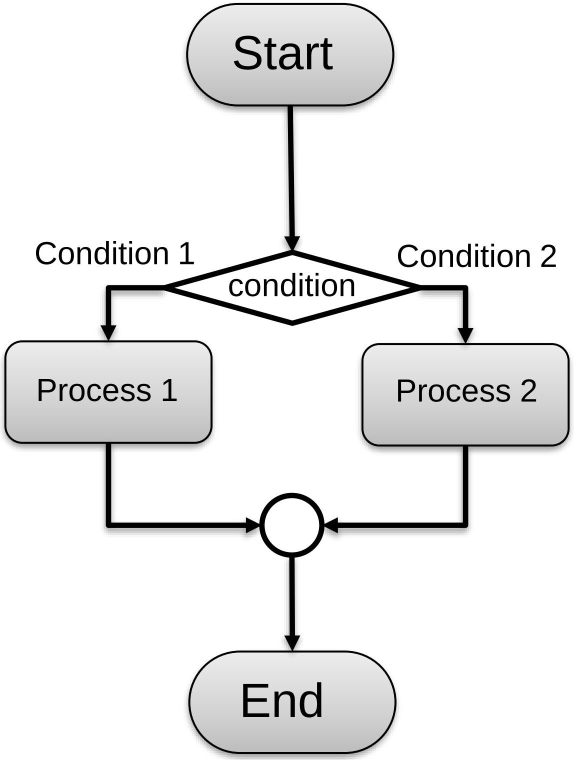
Iteration

Flowchart: processing flow

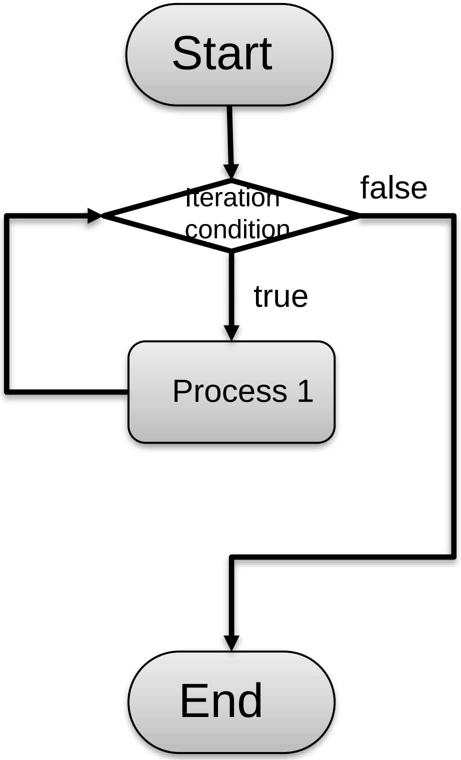
- Expression using flowchart



Sequence

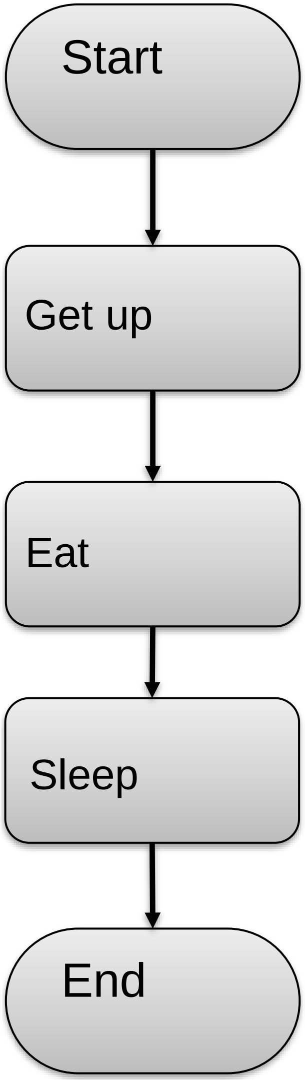


Selection



Iteration

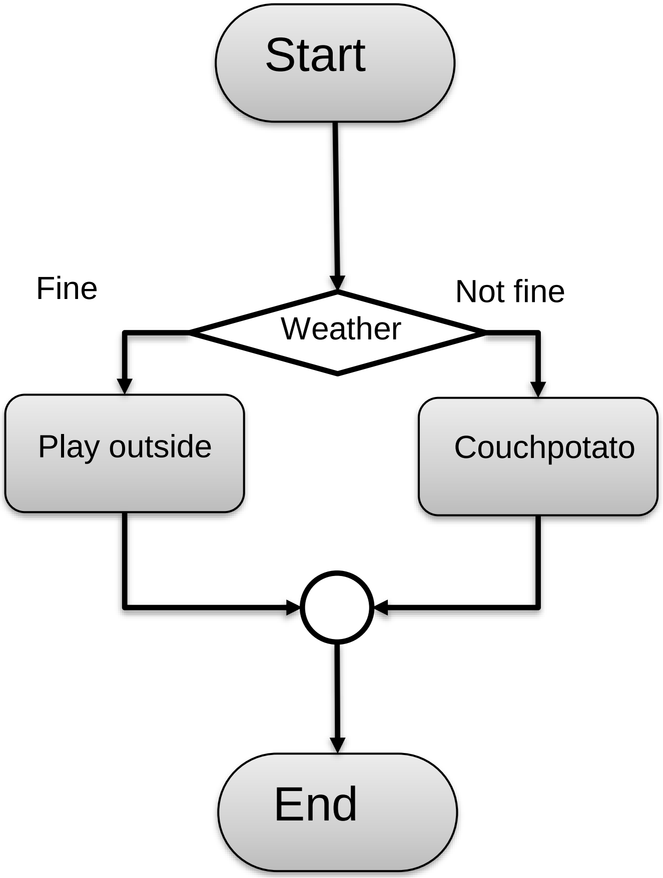
Sequence process



C-language-like description

```
Get up；
Eat；
Sleep；
```

Selection process（if statement）



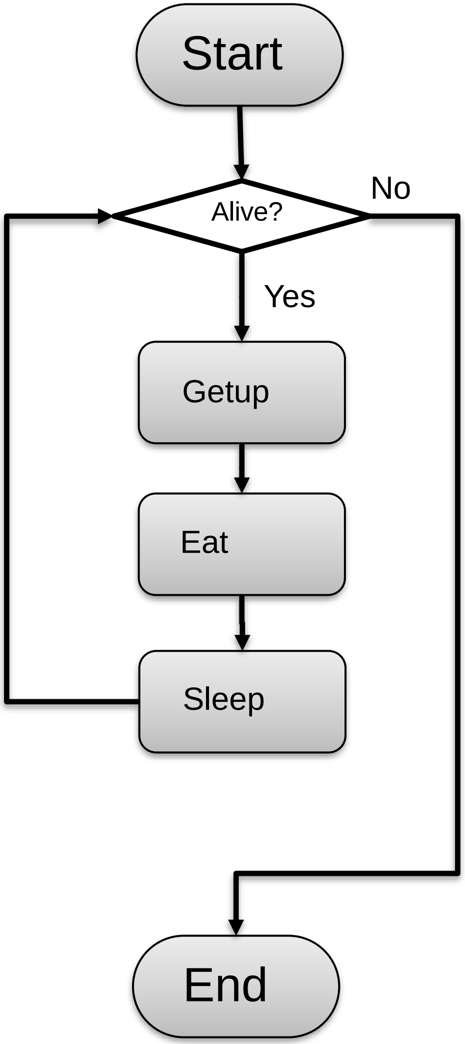
Selection

C-language-like description

```
if (weather == fine){
        play outside；
} else {
        couchpotato；
}
```

※next lecture

Iteration process（while statement）



Iteration

C-language-like description

```
while (alive){
        get up；
        eat；
        sleep；
}
```

※next lecture

Issues in flowchart

- Flowchart for complex processing is confusing

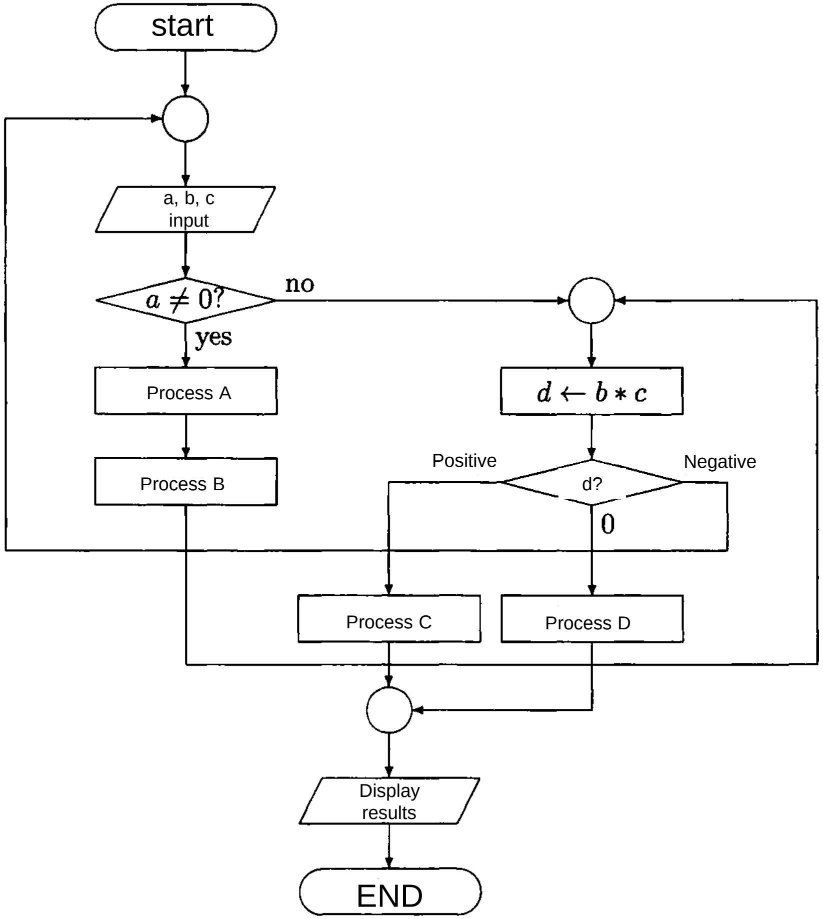
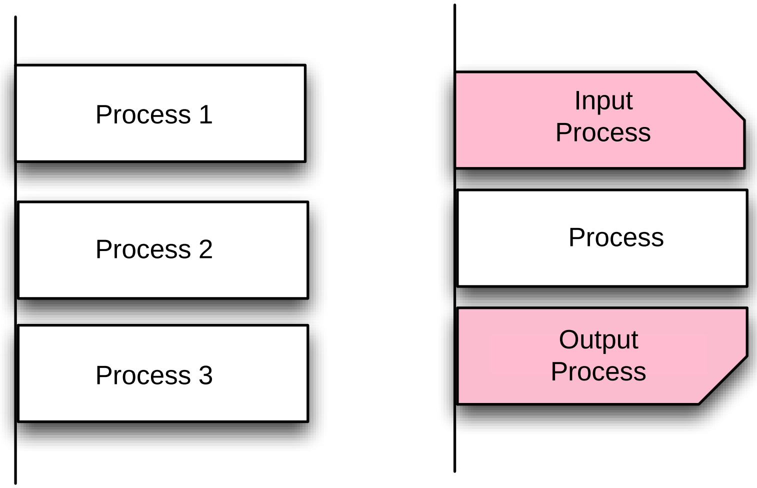


Figure indicating confusing flowchart

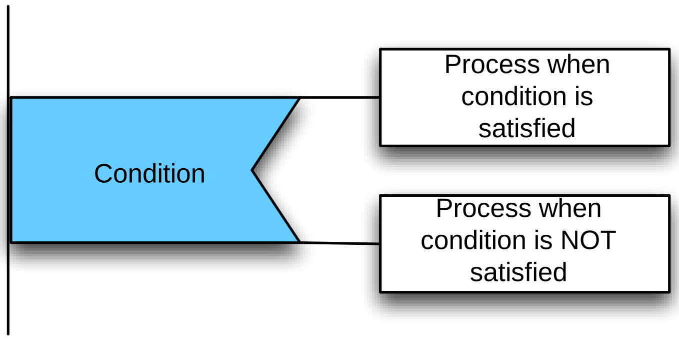
Expression using PAD (Problem Analysis Diagram)



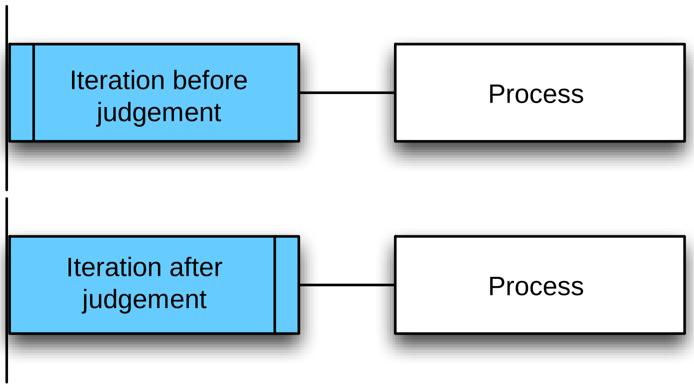
Sequence

Process in order of
Top to bottom
Left to right

Details in
next week


Condition
Process when condition is satisfied
Process when condition is NOT satisfied

Selection



Iteration

Iteration of a certain number of times

- `for` statement

```
for （Initial condition ；  continuation condition；  Incremental process）
{
    Repeating process ；
}
```

- Execution order of `for` statement

1. Execute the expression of the initial condition
2. Execute the processing in the `for` statement block
3. Perform incremental processing
4. If the continuation condition is true, return to 2.

Exercise 2-3: `for` statement

<loop.c>

```c
#include <stdio.h>

int main()
    /**** variable declaration ****/
    int i;

     /**** processing contents****/
    for(i=0;i < 10; i++){
        printf(“hello: %d¥n", i);
    }

    return 0;
}
```

- Write the following program using templete.c and check the execution result.
- Filename： loop.c
- Compile:

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

Output results

```
$ ./loop
hello: 0
hello: 1
hello: 2
hello: 3
hello: 4
hello: 5
hello: 6
hello: 7
hello: 8
hello: 9
```

Why the numbers are from 0 to 9
Please check by looking at the program

```
  for(i=0; i<10; i++){

}
```

- Here, the notation "i ++" is the same as i=i+1.

Exercise 2-4: Practice of `for` statement

Submission required

- Modify the previous loop.c to create a program that performs the following calculation.
  - loop01.c
    - A program that calculates the sum from 1 to 100 and displays the result
  - loop02.c
    - A program that calculates the sum of numbers that are multiples of 3 from 1 to 1000 and displays the result.

Exercise 2-5: Calculation of factorial：factorial.c

Submission required

- Create a program that outputs the factorial (n!) of the natural number n entered from the keyboard.

Definition of factorial

$$n! = n \times (n-1) \times (n-2) \times \cdots \times 3 \times 2 \times 1$$

Exercise 2-6. Calculation of π : circular.c

Submission required

- Calculate the circular constant π approximately by using the following facts.

$$\arctan x = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{2n+1}$$

$$\arctan 1 = \frac{\pi}{4}$$

Hints:

- $(-1)^n = 1$ when n is even.
- In the program code,
  `1/3=0` (integer) and `1.0/3=0.33333333333333` (double).

Summary

- C language basics
  - Variables and data types
  - How to use the `printf` function

- From human thinking to computer programs
  - Three basic forms of processing flow
  - “Sequence”, “Selection”, “Iteration”
  - Flowchart and PAD representation

- Basic form “Iteration of a certain number of times”
  - How to write a `for` statement

Lecture next week

- Example of model answer (`for` statement)

- Selection format
  - `if` statement
    - `if`
    - `if else`
    - `else`
  - `switch` Statement
- Iteration format No. 2
  - `while` statement
- Array(1)