

Practice of Information Processing

(IMACU)

Sixth lecture, part 1

Previous exercises

Makoto Hirota

Contents of this lecture

2

■ Review of previous lecture

■ Mechanism of computer

■ Pointer

- memory
 - Variable area in memory and address
- Pointer to variable
- Pointer to structure

■ File input/output

- Read/write file
- Access option of file
- Functions of read/write file

How to define a function

3

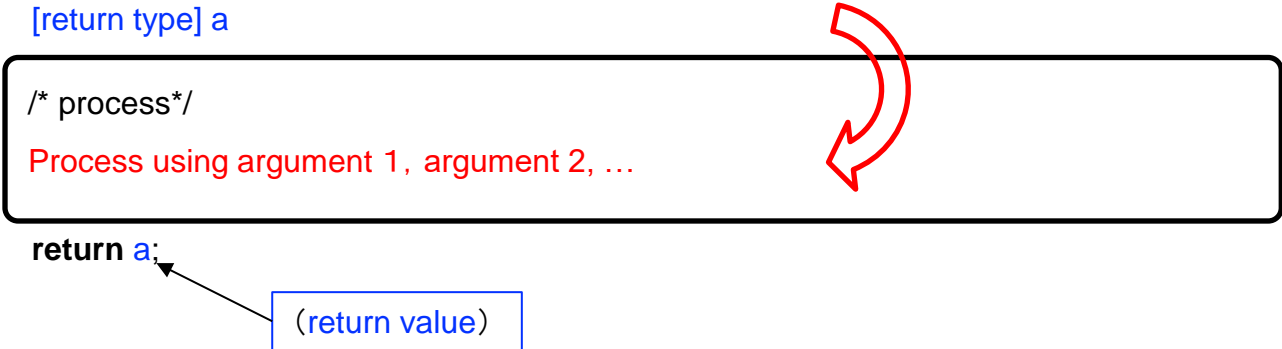
■ General form of function definition

```
[type of return value] function_name([type of argument1] argument1, [type of argument2] argument2, ...)
{
  /* Variable declaration */
  [return type] a

  /* process*/
  Process using argument 1, argument 2, ...

  return a;
}
```

The arguments described here can be used without declaring them as variables in the program.



Example (a function that adds two variables)

```
/* User defined add_func */

int add_func(int x, int y)
  /***** variable declaration*****/
  int a;
  /***** processing contents*****/
  a = x + y;
  /***** return value *****/
  return a;
}
```

Exercise 5-1 Practice your own function: calc_func.c

4

- Modify the sample program add_func2.c (with prototype declaration) and create a program calc_func.c that adds the three functions of
 - Subtraction function: sub_func
 - Multiplication function: multi_func
 - Division function: div_func
- Also, in the main function, execute the following calculation using the above functions.

$$y = (2.0 + 1.0) \times 6.0 / 1.5 - 3.0$$

- All the variables are double type.
- Define the functions after the main function and use a prototype declaration.

[Tips] For example, $y = 2.0 \times 3.0 + 4.0$ is calculated as follows.

```
double p;  
p = add_func( multi_func(2.0, 3.0), 4.0)
```

The return value of ① is directly substituted into ②

Exercise 5-1: calc_func.c

5

Function of four rules function

```
double add_func(double x, double y)
```

```
double a
a = x + y
return a
```

```
double sub_func(double x, double y)
```

```
double a
a = x - y
return a
```

```
double multi_func(double x, double y)
```

```
double a
a = x * y
return a
```

```
double div_func(double x, double y)
```

```
double a
a = x / y
return a
```

Ex1. of main function

```
double p,q,r,s,t
p = 2.0
q = add_func(p, 1.0)
r = multi_func(q, 6.0)
s = div_func(r, 1.5)
t = sub_func(s, 3.0)
printf("%f\n", t)
return 0
```

Ex2. of main function

```
double p
p = 2.0
Next value      Current value
p = add_func(p, 1.0)
p = multi_func(p, 6.0)
p = div_func(p, 1.5)
p = sub_func(p, 3.0)
printf("%f\n", p)
return 0
```

One variable is enough

Ex3. of main function

```
double p
p = sub_func(div_func(multi_func(add_func(2.0, 1.0), 6.0), 1.5) , 3.0)
printf("%f\n", p)
return 0
```

Functions can be called in a nested structure
(put the function inside the function argument)

Exercise 5-2 Finding the least common multiple lcm.c

6

■ Create a program lcm.c that displays the least common multiple of two integers input from the keyboard.

- The least common multiple m of the two integers a and b is expressed by $m = ab / d$ using the greatest common divisor d of a and b .
- Create and use the function $d = \text{gcd}()$ to find the greatest common divisor
- The greatest common divisor can be calculated using the following algorithm called the Euclidean algorithm.

- ① Let two integers a and b ($a > b$), and let r be the remainder of dividing a by b .
- ② If r is 0, then b is the greatest common divisor
- ③ If r is not 0, return to ① with $a = b$ and $b = r$

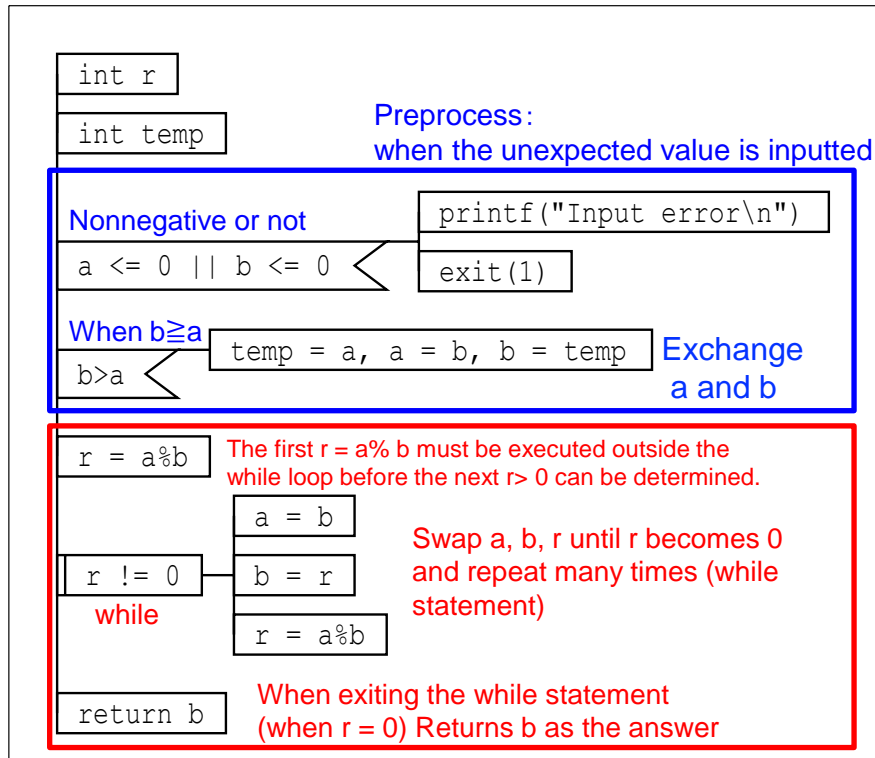
- If 0 or a negative value is entered in either of the two integers, it should terminate abnormally ($\text{exit}(1)$) in main function.
- Implement $\text{gcd}()$ without using recursive calls

Exercise 5-2: lcm.c example answer

7

PAD representation of the function gcd () for finding the greatest common divisor

```
int gcd(int a, int b)
```



Algorithm: Euclidean algorithm

- ① Let two integers a and b ($a > b$), and let r be the remainder of dividing a by b .
- ② If r is 0, then b is the greatest common divisor
- ③ If r is not 0, return to ① with $a = b$ and $b = r$

✂ Refer to previous lecture for the `main` function

Exercise 5-3: lcm2.c (recursive call)

8

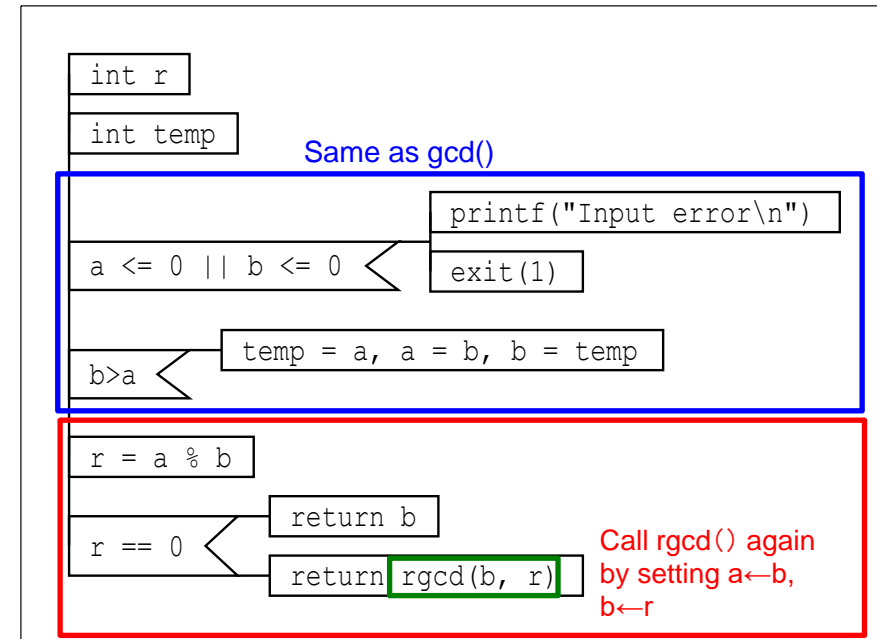
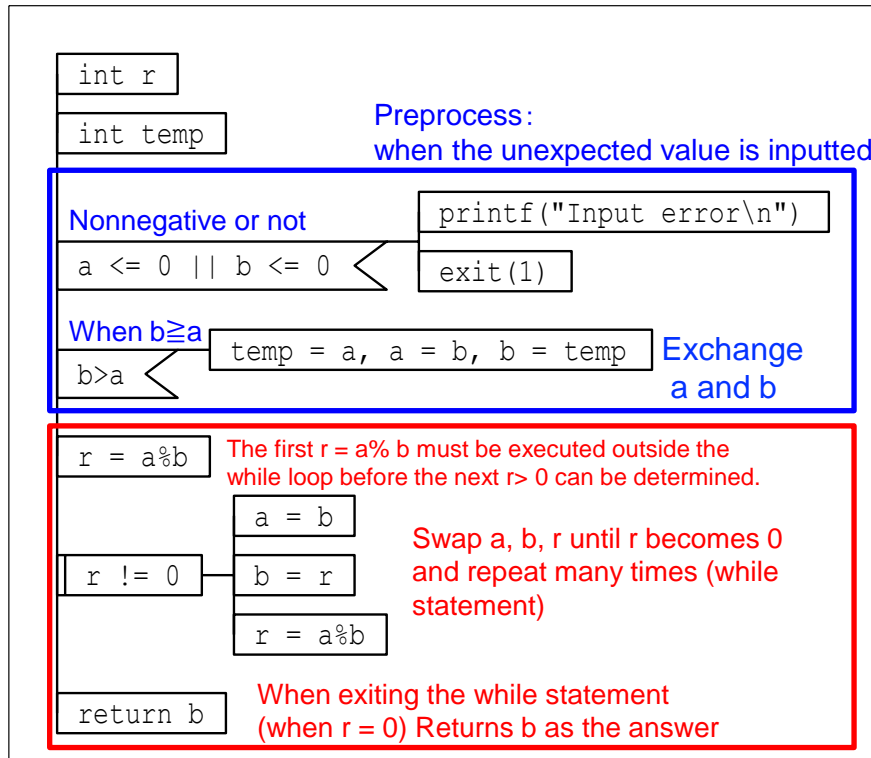
Exercise 5-2: gcd() without recursive call

```
int gcd(int a, int b)
```



Exercise 5-3: rgcd() with recursive call

```
int rgcd(int a, int b)
```



Effective when the output value is used as the next input and the same pattern of processing (= function) is repeated.

Previous lecture: Structure

9

- Defined as a new type (= structure) to handle multiple different variables with one name

We can refer the member (component) by “Variable_name.(dot)member_name”

Defined outside the main function at the beginning of the program

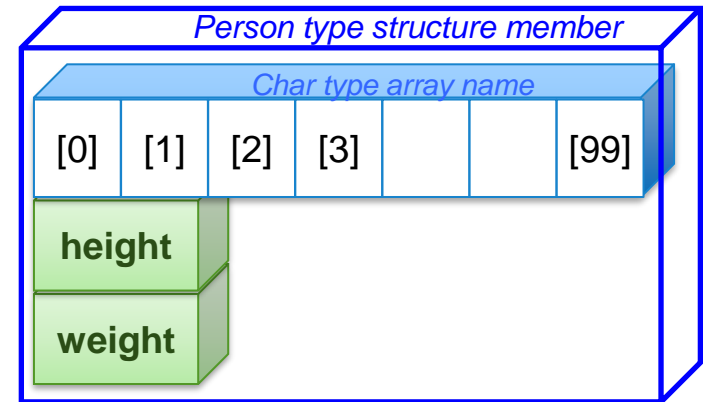
```
struct person {  
    char name[100];  
    double height;  
    double weight;  
};
```

←semicolon “;” is required



Can be declared as a variable type in a program

```
struct person member1, member2; (Declare 2 variables)  
    type name      variable      variable  
  
member1.height = 170.5; (first person)  
member1.weight = 62.0;  
  
member2.height = 165.0; (second person)  
member2.weight = 55.3;
```



Person type structure

Previous lecture: Supplement: Initialize when declaring a structure₁₀

- Initial values can be assigned with {,,} (bracket) at the time of declaration

```
struct person{  
    char name[NAME_LEN];  
    double height;  
    double weight;  
};  
typedef struct person st_person;
```

Structure definition

```
main() {  
  
    st_person member = {"Ichiro", 170, 50};  
  
    printf("Name:    %s\n", member.name);  
    printf("Height: %.1f\n", member.height);  
    printf("Weight:  %.1f\n", member.weight);  
  
}
```

Declaration of structure

Previous lecture: Array of structure

11

- Since a structure is a type of variable, it can also be an array.

```
main() {
    int i;
    struct person member[3];

    for (i=0; i<3; i++){
        scanf("%s", member[i].name);
        scanf("%lf", &member[i].height);
        scanf("%lf", &member[i].weight);
    }
}
```

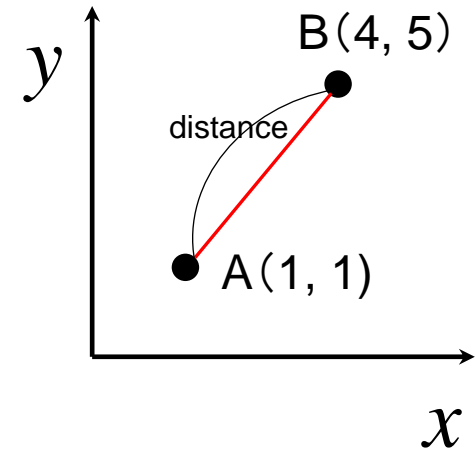
```
main() {
    int i;
    st_person member[3];    (alias of structure name)

    for (i=0; i<3; i++){
        scanf("%s", member[i].name);
        scanf("%lf", &member[i].height);
        scanf("%lf", &member[i].weight);
    }
}
```

Exercise 5-4 struct_point.c

12

- Create a program `struct_point.c` that finds the distance between two points in a two-dimensional plane.
 - For the coordinates of a certain point, define a structure `Point` having x and y coordinates as members, and use it for distance calculation.
 - Use double type coordinates
 - The square root uses the mathematical function `sqrt()`
 - The square can be " $x * x$ " or "`pow(x, 2)`" using the mathematical function `pow()`.
 - Coordinates are assigned directly in the main function. You may substitute (1.0, 1.0), (4.0, 5.0) at the time of declaration.



For mathematical functions
`#include <math.h>`
is necessary
(-lm for compilation)

Example of declaration: `struct Point a = {1.0, 1.0};`
`struct Point b = {5.0, 4.0};`

- Display the two coordinate values and the distance in the output.

Display example

```
a = (1.000000, 1.000000)
b = (5.000000, 4.000000)
distance = 5.000000
```

Exercise 5-4 : struct_point example

13

```
struct Point {  
    double x;  
    double y;  
};
```

Definition of structure

```
int main()
```

```
    struct point a = {1.0, 1.0}
```

```
    struct point b = {5.0, 4.0}
```

```
    double dist
```

```
    printf("a = (%lf, %lf)\n", a.x, a.y)
```

```
    printf("b = (%lf, %lf)\n", b.x, b.y)
```

```
    dist = sqrt((b.x - a.x) * (b.x - a.x) + (b.y - a.y) * (b.y - a.y))
```

```
    printf("distance = %lf\n", dist)
```

```
    return 0
```

Exercise 5-5: struct_array.c

14

- Rewrite the sample program struct_ex.c and create a program struct_array.c that inputs and displays the data of N people.
 - Define the number of people N as a macro (N = 3 is sufficient)
 - Also display the average height and weight of N people

Exercise 5-5: struct_array.c example

15

Main function

```
struct person member[NUMBER]
int i
double ave_h=0, ave_w=0

    printf("Name? ")
    scanf("%s", member[i].name)
    printf("Height? ")
    scanf("%lf", &member[i].height)
    printf("Weight? ")
    scanf("%lf", &member[i].weight)

i=0; i<NUMBER; i++

printf("-----\n")

    printf("Name:   %s\n", member[i].name)
    printf("Height: %.1f\n", member[i].height)
    printf("Weight: %.1f\n", member[i].weight)
    ave_h += member[i].height/NUMBER
    ave_w += member[i].weight/NUMBER

i=0; i<NUMBER; i++

printf("Ave height%.2f\n", ave_h)
printf("Ave weight%.2f\n", ave_w)

return 0
```

Practice Information

Practice

(IMACU)

Sixth lecture, part 2

Pointer

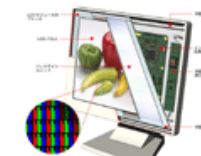
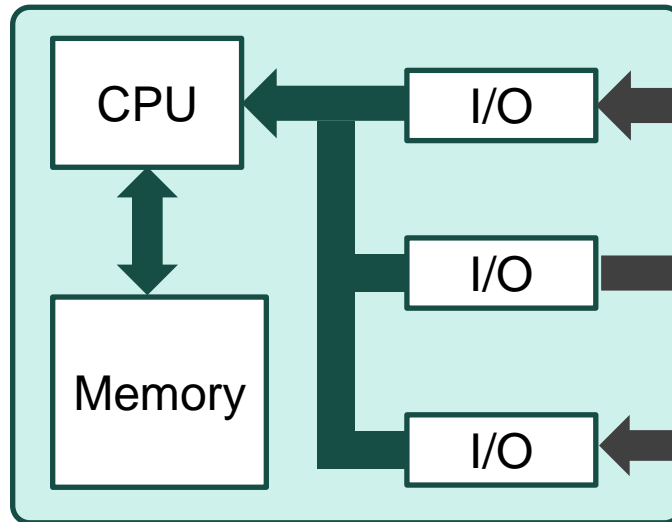
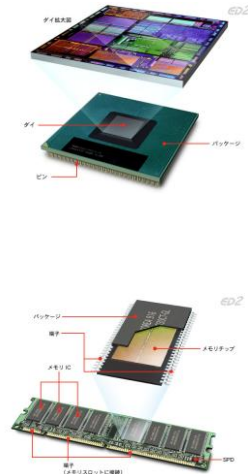
Makoto Hirota

- Review of previous lecture
- Memory address in computer
- Pointer
 - memory
 - Variable area in memory and address
 - Pointer to variable
 - Pointer to structure
- File input/output
 - Read/write file
 - Access option of file
 - Functions of read/write file

Memory address in computer

18

Computer



Input device
keyboard
mouse

Output device
monitor

Auxiliary storage
Harddisk

“Drawer”
(files)

“Desktop”

(variables, arrays)

Memory

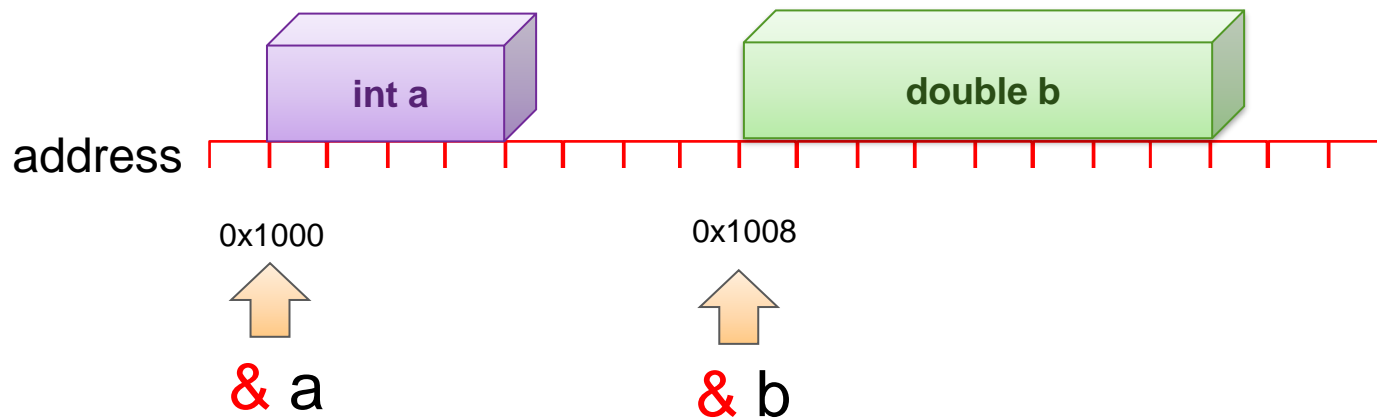
address	
	⋮
0x1000	data1
0x1004	data2
0x1008	data3
0x100f	data4
0x1010	⋮

- Tentative data are stored in memory (while PC is on).
- **Addresses** are assigned to memory space
 - 32bit OS: Address is expressed in 32bit width (= the memory size is limited to about 4GB.)
 - 64bit OS: Address is expressed in 64bit width (= the maximum memory size is theoretically 17.1 billion GB.)
- CPU reads data from a specific address and stores calculated results at a specific address.

How to obtain the address?

19

If you put the address operator (&) before a variable, you can refer to the **start address** of the variable



```
int a;  
double b;  
  
printf("%ld", &a);  
printf("%ld", &b);
```

→ The start address of the variable a is displayed

→ The start address of the variable b is displayed

※ The address is unsigned long type.

Confirmation of variable/array address var_address.c

20

- Execute the sample program var_address.c and check the output.

```
#include <stdio.h>

int main() {

    int    a;
    double b;
    int    c[3];
    double d[3];
    /**** processing contents****/
    printf("-----\n");
    printf("address of int type a :%p\n", &a);
    printf("address of double type b :%p\n", &b);
    printf("-----\n");
    printf("size of int type :%ld\n", sizeof(a));
    printf("size of double type :%ld\n", sizeof(b));
    printf("-----\n");
    printf("address of int type c[0] :%p\n", &c[0]);
    printf("address of int type c[1] :%p\n", &c[1]);
    printf("address of int type c[2] :%p\n", &c[2]);
    printf("-----\n");
    printf("address of int type d[0] :%p\n", &d[0]);
    printf("address of int type d[1] :%p\n", &d[1]);
    printf("address of int type d[2] :%p\n", &d[2]);
    return 0;
}
```

The size of the variable (number of bytes) can be calculated with the sizeof () function.

Format when displaying the address: "%p"
The output result has "0x" at the beginning to indicate that it is a hexadecimal number.

Display example:

```
-----
address of int type a :0xbf8b542c
address of double type b :0xbf8b5420
-----
size of int type :4
size of double type :8
-----
address of int type array c[0]:0xbf8b5414  +4
address of int type array c[1]:0xbf8b5418  +4
address of int type array c[2]:0xbf8b541c
-----
address of double type array d[0]:0xbf8b53f8  +8
address of double type array d[1]:0xbf8b5400  +8
address of double type array d[2]:0xbf8b5408
```

0x represents hexadecimal display

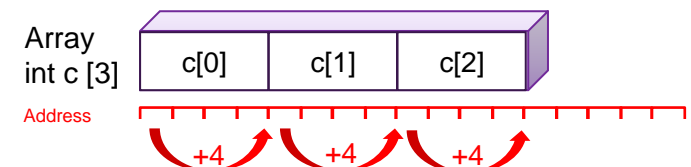
The displayed address is different each time it is executed

Int type: 4byte
Double type: 8byte

You can see that the address of each element of the array increases with the size of the data type:

- int type is 4 bytes each
- double type is 8 bytes each

The array exists in a continuous memory area and is arranged by data size.



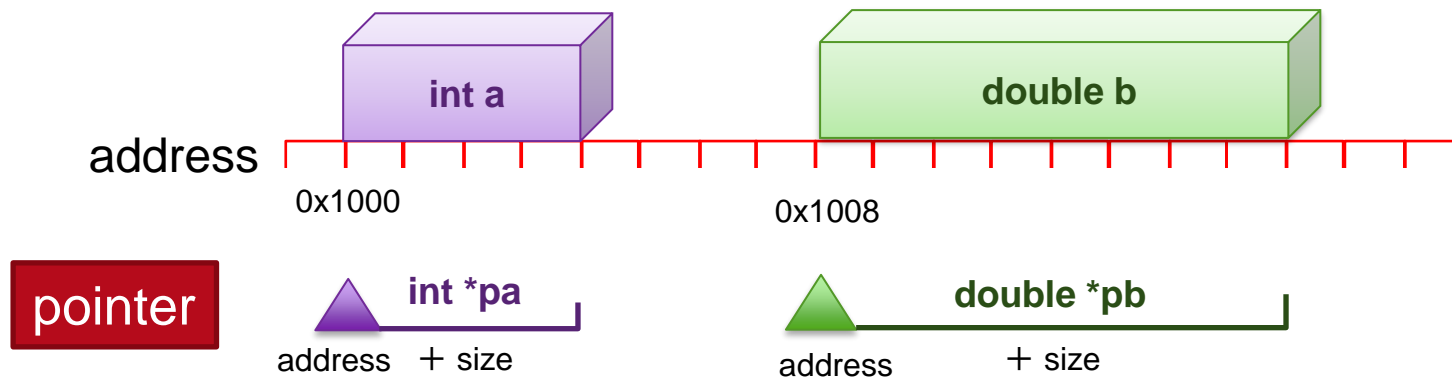
What is pointer?

21

In C, addresses can be saved as pointers.

```
int    a; int *pa;  
double b; double *pb;  
pa=&a;  
pb=&b;
```

- A special variable that holds the **start address** (address) of a "variable (data type)" in memory.
- When declaring a pointer, it is necessary to specify the **"type"** of the data to be pointed to.

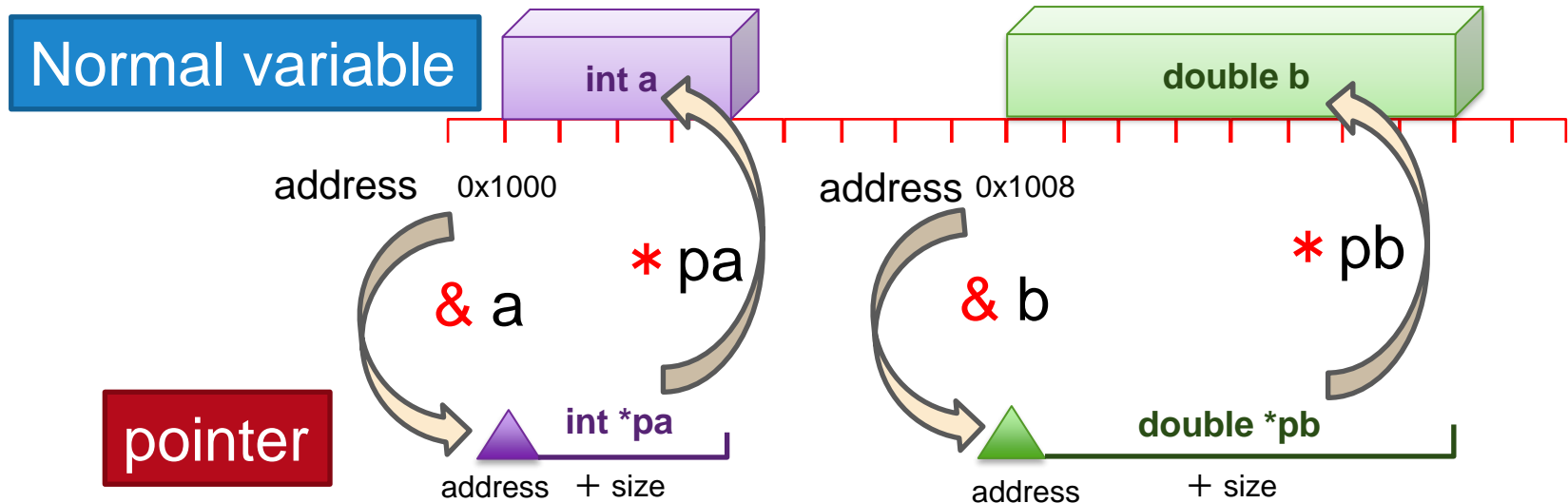


“Pointer” contains information on the "address" at the beginning of the variable and the "size" of the type.

Variables can be referred by Addresses (pointers)

22

The variable at the pointer can be obtained by the indirect operator (*).



&operator: “variable” \Rightarrow “pointer” (first address)
(address operator)

***operator:** “pointer” \Rightarrow “variable” at the address
(indirect operator)

Ex. When `pa=&a` , `*pa` can be used in place of `a`.

How to declare a pointer?

- You can declare a pointer to a data type by adding "*" in front of the pointer name.

```
Data_type *pointer_name;
```

ex)

```
int *pa;  
double *pb;
```

How about structure?

```
struct person *member;
```

This is type name

Similarly, if you add "*" after the type name, it becomes a pointer to the structure.
(Pointers to structures will be dealt with in the next lecture)

Note: Position of asterisk when declaring pointer ²⁴

- The following three declarations have the same meaning.

```
int *a;  
int* a;  
int * a;
```

Styles

- ① Just before the variable *
- ② Immediately after the mold *
- ③ Leave a space between the type and the variable *

Note

```
int* a, b;
```

only a is a pointer. b is an ordinary int type

【Note】 There is no type of "int*"

If you want both to be pointers, you should write

```
int *a, *b;
```

Therefore, the style of ① is recommended to avoid mistakes.

■ Assignment of address

```
pointer = &variable;
```

■ Reference to the variable pointed to by the pointer

```
variable = *pointer;
```

Ex)

<code>int x = 10;</code>	Declaration of int type variable x
<code>int *px;</code>	Declaration of pointer variable px for int type
<code>px = &x;</code>	Substitute the address of the variable x into the pointer variable px
<code>*px += 1;</code>	Refer to the contents of the pointer variable px and add 1 = Equivalent to adding 1 to the variable x (x = 11)

Exercise 6-1 var_pointer.c

26

- Create and execute the following program, and consider the results.

```
#include <stdio.h>

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

    /*** processing contents***/
    printf("Input variable: ");
    scanf("%d", &x);

    p = &x;
    printf(" x = %d \n", x);
    printf("*p = %d \n", *p); //indirect reference by pointer

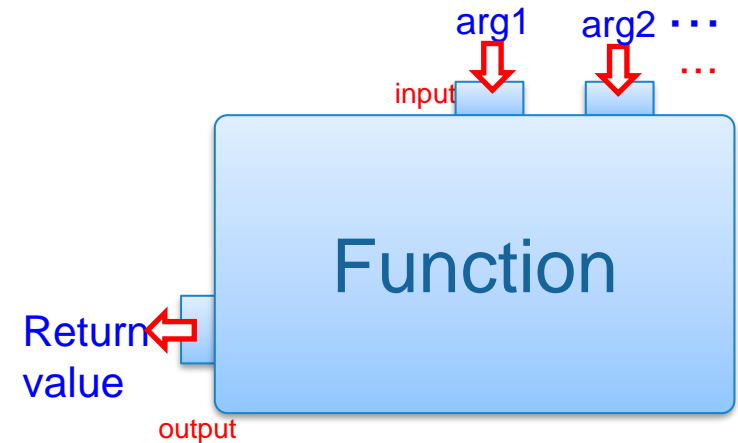
    printf("&x = %p \n", &x); // address for variable x
    printf(" p = %p \n", p); //value for the pointer p

    *p = 20;
    printf(" x = %d \n", x);
    // Please write here in comments why the results above obtained
    return 0;
}
```

Review: Problems of using function (without pointer)

27

- Only **one return** value can be output from a function.
- Only the **values** of the arguments are passed to a function (“call-by-value”).
 - If a function is called by `func(x)`, this function can read the value of `x` but **cannot change it**.



When you want to return multiple results:

Method 1: Use **pointers** in arguments and return value. “call-by-reference”


Method 2: Use **global variables**

- The way using **pointers** for arguments (= inputs) and return values (= outputs)

Function with
call-by-value

```
int func(int var1)
```

Output Input



Function with
call-by-reference

```
int func(int var1, double *var2, double *var3, ...)
```

Output Input Output Output

For example, var1 is input and the results are output in *var2, *var3 and so on.

Advantages

- Without using global variables, this function can read and write variables that are defined outside of its scope.
- No extra memory space is needed.

Q. Have you ever used “call-by-reference”? ⇒ Yes! `scanf ("%d", &x)`

Exercise 6-2. Difference in call-by-(value/ref.)

29

- Write and execute program callby.c

```
#include <stdio.h>
/* prototype declaration */
int add(int v);
void addp(int *p);

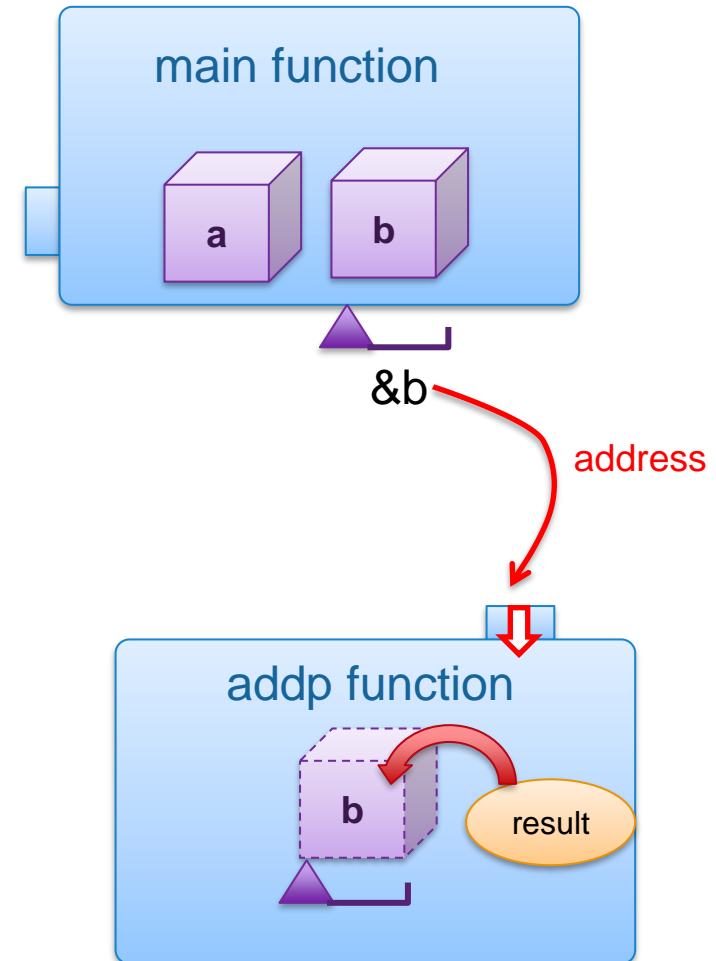
int main(){
    int a,b;

    printf("Input value: ");
    scanf("%d", &a);

    b = add(a);
    printf("Call by value: %d\n", b);
    b = a;
    addp(&b);
    printf("Call by reference: %d\n", b);
    return 0;
}
/* call by value function */
int add(int v){
    v += 10; // add 10
    return v;
}
/* call by reference(pointer) */
void addp(int *p){
    *p += 10; // add 10
}
```

value

reference



Since there is an int type variable b at the end of the sent pointer, the result can be assigned to the contents (* p) of the pointer.

Exercise 6-3 Swap values swap.c

30

- Create a program swap.c that inputs two integers and displays them in a different order.
 - Define the swap function according to the main function on the right
 - The swap function takes pointers as arguments as shown in the prototype declaration

```
#include <stdio.h>
/* prototype declaration */
void swap(int *x, int *y);

int main(void) {
    int a,b;

    /* Input 2 integers */
    scanf("%d %d" &a, &b);

    /* Exchange the contents */
    swap(&a, &b);

    /* Display results */
    printf("%d %d\n", a, b);

    return 0;
}
```

Display example

```
10 20 (enter)
20 10
```

Practice of Information Processing

(IMACU)

Sixth lecture, part 3 File input/output

Makoto Hirota

- Review of previous lecture
- Mechanism of computer
- Pointer
 - memory
 - Variable area in memory and address
 - Pointer to variable
 - Pointer to structure
- File input/output
 - Read/write file
 - Access option of file
 - Functions of read/write file

- In C language, by using a pointer to a structure called FILE type (file pointer), input / output can be performed in the same manner using `fscanf` and `fprintf`.

- By default, keyboard and terminal are opened as standard input / output with the file pointers `stdin` and `stdout`.
- "Output" to the screen (terminal console): Function `printf` → Output to file pointer `stdout`
- "Input" from the keyboard: Functions `scanf`, `getchar`, etc. → Input from the file pointer `stdin`

■ How to access the file

1. Declare a file pointer

```
FILE *fp;      declare file pointer
```

2. "Open" the target file

- Associate a file name with a file pointer
- Access mode ("r": read-only, "w": writable)

```
fp = fopen(filename,  
              accessmode);
```

3. Reading and writing to files:

- Specify a file pointer for input / output functions (`fprintf`, `fscanf`, `fgets`, etc.)

ex)

```
fprintf(fp, "Hello¥n");  
fscanf(fp, "%d %d", x1, x2);
```

4. When to finish:

- "Close" the file. (Disable the file pointer)

```
fclose(fp);
```

Open and close of files

34

■ fopen function

- Opens a file and returns information about its FILE structure as a file pointer

```
FILE* fopen ( filename, access_mode);
```

When it cannot open it,
NULL is returned

- access mode:

- Read-only or writable, or select the mode according to the operation when the file exists / does not exist.

Mode	Action	File exists	File does not exists
"r"	Read	Normally executed	Return NULL(=no file generated)
"w"	Write new file	Remove contents and write from top	New file generated
"a"	Write additionally into existing file	Keep contents and write from the bottom	New file generated
"r+"	Update mode (read/write)	Normally executed	Return NULL(=no file generated)
"w+"	Update mode(New write/read)	Remove contents and write from top	New file generated
"a+"	Update made (Additional write/read)	Keep contents and write from the bottom	New file generated

Advanced usage

■ fclose function

※All update modes can be read / written, but the operation differs depending on the presence or absence of a file.

```
int fclose (FILE*);
```

After fopen (), be sure to fclose ()

Exercise 6.4

- Create “file_open.c” below, and “sample.txt” which includes characters(ex. your name, address, or poem), and executes the program.
- Then, change the name of sample.txt and re-execute the program.

```
#include <stdio.h>

int main(){
    FILE *fp;

    fp = fopen("sample.txt", "r");
    if( fp == NULL){
        printf("can't open file\n");
        return 1;
    } else {
        printf("success\n");
        fclose(fp);
        return 0;
    }
}
```

■ fprintf / fscanf

- Write / read by specifying the format for each line
- After the first argument FILE *, it is the same as printf and scanf.
 - `int fprintf(FILE*, "format", argument, ...);`
 - `int fscanf(FILE*, "format", argument, ...);`

■ fputc / fgetc

- 1 character write / read
 - `int fputc(int c, FILE*);`
 - `int fgetc(FILE*);`

Note that the position of the argument of the file pointer differs depending on the function.

```
fprintf(*fp, "Hello");  
fputs("Hello", *fp);
```

■ fputs / fgets

- Write one line of character string / read the specified size (including line feed code at the end of the line)
 - `int fputs(str[], FILE*);`
 - `int fgets(str[], int size, FILE*);`

Practical example: Writing to a file

37

```
/*
file_write.c
Sample program for file output
*/

#include <stdio.h>

int main(void) {
    FILE *fp;

    fp = fopen("sample_out.txt", "w");
    if (fp == NULL) {
        printf("Can't open file ¥n");
        return 1;
    }

    fprintf(fp, "Hello!¥n");

    fclose(fp);

    return 0;
}
```

All file access is treated
by the file pointer fp

- 1) declaration FILE type fp
- 2) open file (access mode is set to be "w:writable")
(Error handling when the file cannot be opened)
- 3) write into file
- 4) close file

38

- Write and execute this code (file_write.c). Then confirm the contents of “sample_out.txt.”
- Note that the contents of the file will be overwritten by this program!
- Change access mode “a” to see what happen

Difference in access mode for writing to a file

39

- Change the access mode of the sample program file_write.c

```
fp = fopen("sample_out.txt", "w");
```

access
mode

New write

```
fp = fopen("sample_out.txt", "a");
```

Additional write

Check how the contents of sample_out.txt change when file_write is repeatedly executed.

Write to file: sample file_write2.c

40

Output strings examples

```
char str[] = "I got it¥n";  
char str2[] = "an apple";
```

■ When writing each character

fputc

- int fputc(int c, [FILE *fp](#));

Return value: Returns the written character. Returns EOF in case of error

※ The specified character is written as is, regardless of whether it contains a null character.

```
for(i=0;str[i]!='¥0';i++){  
    fputc(str[i],fp);  
}
```

A null character (¥0) is included at the end of the character string. If you write it as is, even the last NULL character will be written, so you need to stop before that.

Output results

I got it ↵

■ When writing for each character string

(1) When writing as it is without formatting

fputs

- int fputs(char *str, [FILE *fp](#));

(Return value: Returns a positive number for normal termination, EOF for error)

※ Output without null characters from the string

```
fputs(str, fp);  
fprintf(fp, "%s", str);
```

Output results

I got it ↵
I got it ↵

Exactly the same results

(2) When writing by specifying the format

fprintf

- int fprintf([FILE *fp](#), "format", argument, ...);
 - After the first argument FILE *, it is the same as printf

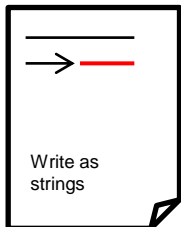
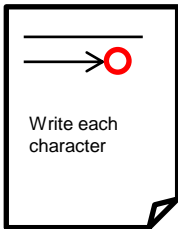
(Return value: Returns the number of written characters. Returns a negative value in case of error)

※ Output according to the format specification

```
fprintf(fp, "I got %s¥n", str2);
```

Output results

I got an apple ↵



Actual example: Reading file contents

41

```
/*
file_read.c
Sample program for file unput
*/

#include <stdio.h>

int main(void) {
    FILE *fp;
    char buf[256]

    fp = fopen("sample_read.txt", "r");

    if(fp == NULL) {
        printf("Can't open file %n");
        return 1;
    }

    while(fscanf(fp, "%s", buf) != EOF) {
        printf("%s\n", buf);
    }

    fclose(fp);

    return 0;
}
```

1) declare file pointer **fp**

2) open file (access mode "r: readonly")

(Error handling when the file cannot be opened)

3) Read strings until the file end (EOF; End Of File) appears

4) close file

Read from file: Sample file_read2.c

42

■ When reading one character by one

fgetc

- `int fgetc(FILE *fp);`

(Return value: Returns the read character.
Returns EOF at the end of the file or in case of an error)

```
int code;

while((code = fgetc(fp)) != EOF){
    printf("%c",code);
}
```

■ When reading line by line

(1) When reading one line as it is

fgets

- `char *fgets(char *str, int size, FILE *fp);`

(Return value: If the result is normal, a pointer to the read character string is returned.
Returns NULL at the end of the file or in case of an error)

※ One line including the line feed code is read

For size, specify the maximum file size to be read. Normally, specify the size of the array (= buffer) to be prepared.
It is convenient to find the size of the array using the `sizeof ()` function.

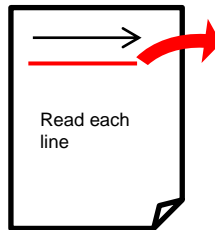
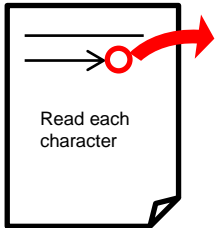
(2) Read one line by specifying the format

fscanf

- `int fscanf(FILE *fp, "format", args, ...);`

(Return value: Returns the number of assigned input items.
Returns the EOF value in case of an error)

`fscanf` is difficult to use because it cannot read spaces and line feed codes. Normally, use a combination of `fgets ()` and `sscanf ()`



Read file: fscanf and fgets

43

- Try to read sample_read2.txt by using file_read2.c
- Use fgets when read one line by one

```
while(fgets(buf, sizeof(buf), fp) != NULL){  
    printf("%s¥n", buf);  
}
```

Since the return value of fgets is char, it is necessary to judge by NULL (empty pointer) instead of EOF (int type).

Exercise 6.6

44

Submission required

- Create a program 'pi_count.c' which can count the number of "0," "1," "2," "3," "4," "5," "6," "7," "8," "9" in the "pi.txt". "pi.txt" can be downloaded from Google classroom.

```
int main(void) {
    int c;
    FILE* fp;
    int count[10]={0,0,0,0,0,0,0,0,0,0};
    .
    .
    while( (c=fgetc(fp)) !=EOF) {
        if( c == '0' ) count[0]=count[0]+1;
        else if(c =='1' ) count[1]=count[1]+1;
        else if(c =='2' ) count[2]=count[2]+1;
        .
        .
        .
    }
}
```

Exercise 6.7

45

Submission required

■ Create the following program 'addone.c'.

- Read a text file 'input.txt' that contains one line:

Input Parameters: 5, 7, 3

- and store 5, 7 and 3 into the variables `a`, `b` and `c`, respectively.
- `a`, `b` and `c` are defined as shown at right. You must use `pc` instead of `c`.
- Write a function that increases `a`, `b` and `c` by one.
- Namely, the output should be

6 8 4

```
#include <stdio.h>

int a;

/* define addone function here */

int main(void) {
    FILE *fp;
    int b, c;
    int *pc; pc=&c;

    /* Read input.txt */

    /* call addone */

    printf("%d %d %d\n", a, b, *pc);
    return 0;
}
```

■ Review of previous Exercises

■ Pointer

- memory
 - Variable area in memory and address
- Pointer to variable
- Pointer to structure

■ File input/output

- Read/write file
- Access option of file
- Functions of read/write file

- Review of exercises
- Pointer(2)
- Final report