

Practice of Information Processing

(IMACU)

Fifth lecture(part 1)
Previous exercise

Makoto Hirota

Contents of this lecture

2

- Example answers of previous exercises
- Definition of function
 - What is function
 - Definition of self-made function
 - Prototype declaration
 - Role of function
- Function with no return value “procedure”
- Recursive call of function
- Data type
- What is structure?

Previous lecture Sort(1) Bubble sort sort_bubble.c

3

■ Compile and execute source file sort_bubble.c

A program that sorts in ascending order

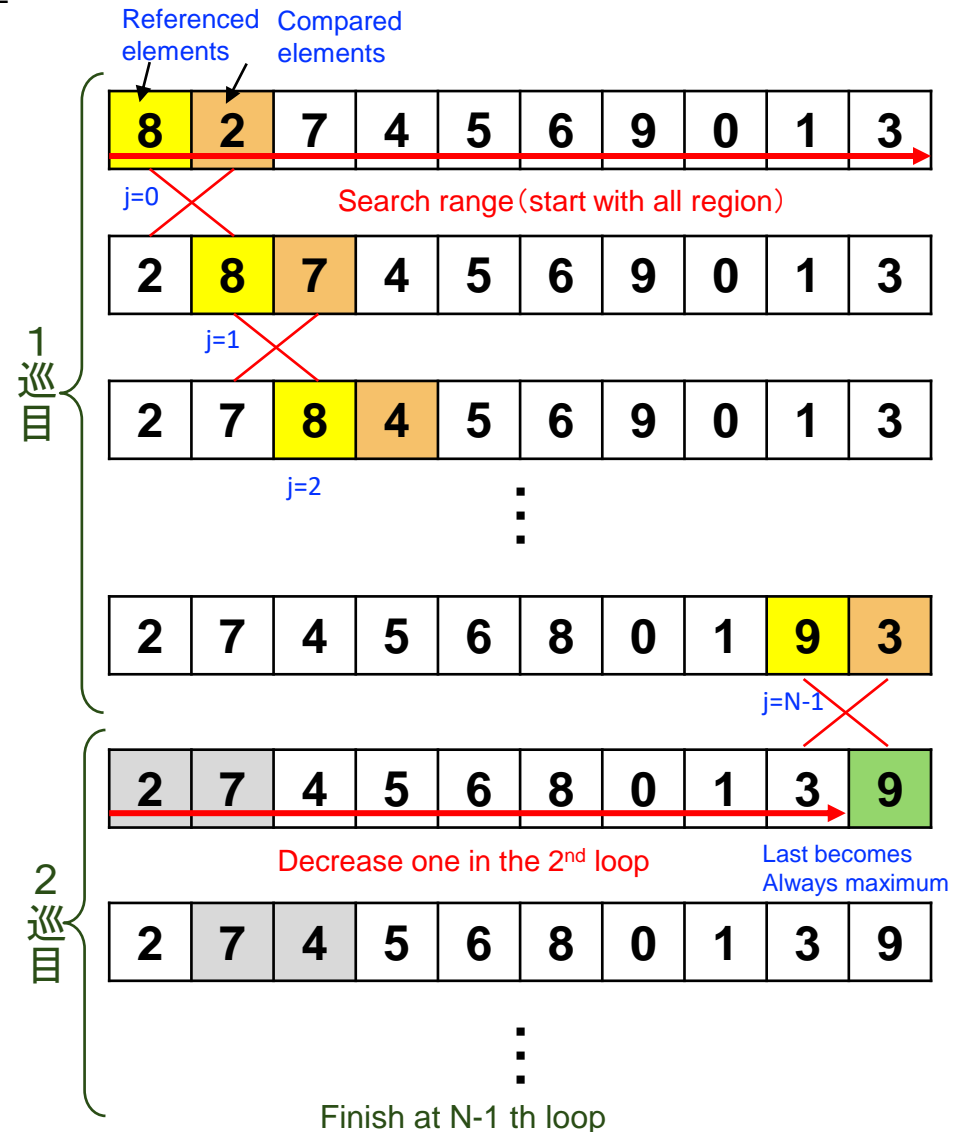
algorithm:

- Compare adjacent elements in order from the front
- If the reference element is larger than the comparison target, replace it.
- Repeat the comparison while reducing the exploration range by one

```
int A[NUM] = {8,2,7,4,5,6,9,0,1,3};
```

```
int i, j, temp;
```

```
for (i = 0; i < NUM-1; i++) { ← repeat N-1 loop
    for (j = 0; j < NUM-i-1; j++) { ← With decreasing the
        if (A[j] > A[j+1]) { ← If the referenced
            temp = A[j];
            A[j] = A[j+1];
            A[j+1] = temp; ← exchange
        }
    }
}
```



Exersice 4-1: Visualization of bubble sort

4

- Modify sort_bubble.c and create a program sort_bubble1.c that visualizes bubble sort.
- Also, count and display the number of comparisons and the number of replacements.

– ex)

- "*" Before the reference element
- ">" Before the element to be compared
- [Comparison count] and [Replacement count] are displayed at the beginning of the array.

- The origin of "bubble" is from the appearance that the larger values move (emerge) to the edge in order.
- Bubble sort always makes $n(n-1)/2$ comparisons

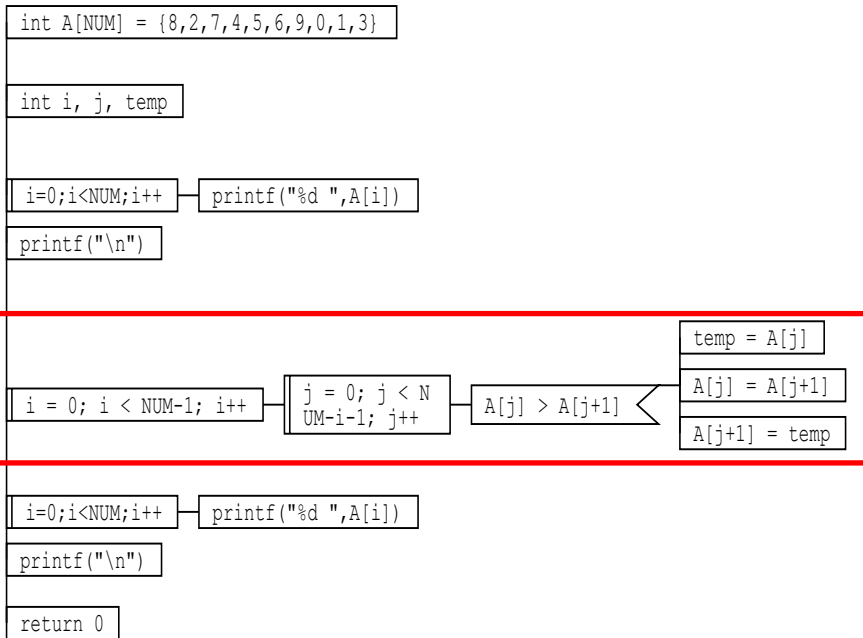
```
[ 1][ 0] *8>2 7 4 5 6 9 0 1 3
[ 2][ 1] 2 *8>7 4 5 6 9 0 1 3
[ 3][ 2] 2 7 *8>4 5 6 9 0 1 3
[ 4][ 3] 2 7 4 *8>5 6 9 0 1 3
[ 5][ 4] 2 7 4 5 *8>6 9 0 1 3
[ 6][ 5] 2 7 4 5 6 *8>9 0 1 3
[ 7][ 5] 2 7 4 5 6 8 *9>0 1 3
[ 8][ 6] 2 7 4 5 6 8 0 *9>1 3
[ 9][ 7] 2 7 4 5 6 8 0 1 *9>3
```

....

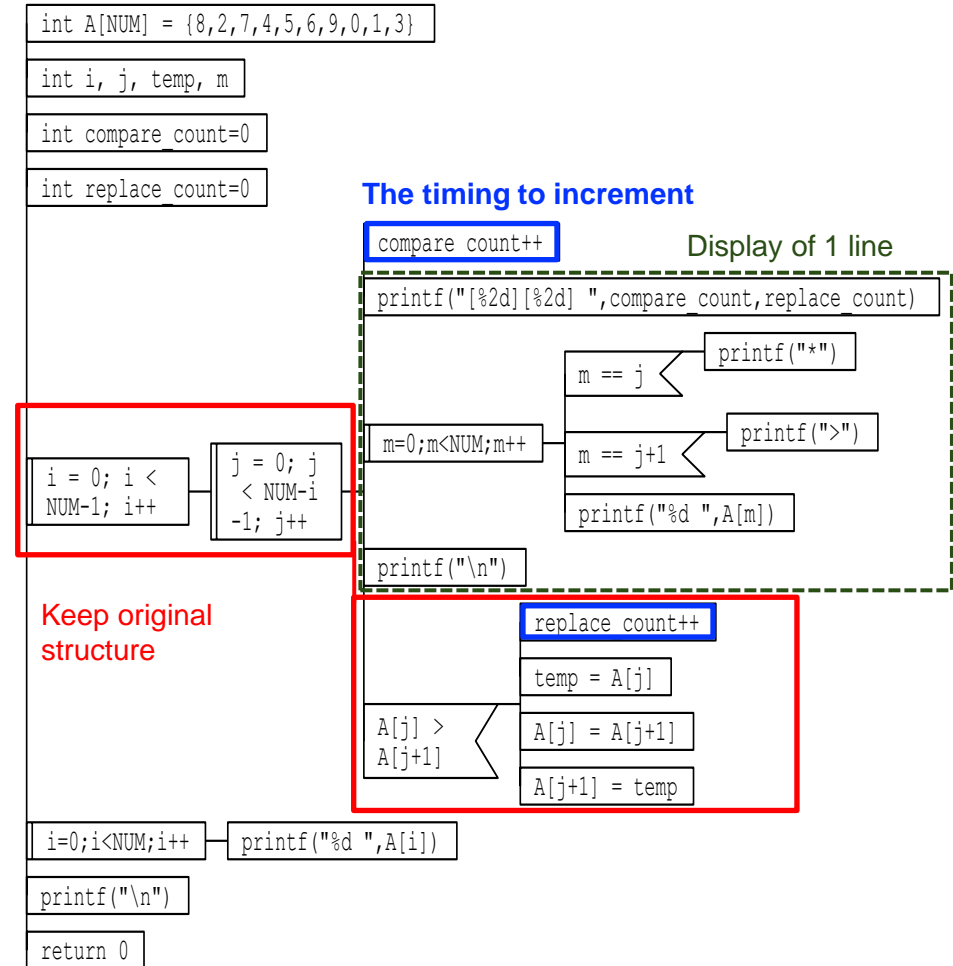
```
[38][24] 2 0 *4>1 3 5 6 7 8 9
[39][25] 2 0 1 *4>3 5 6 7 8 9
[40][26] *2>0 1 3 4 5 6 7 8 9
[41][27] 0 *2>1 3 4 5 6 7 8 9
[42][28] 0 1 *2>3 4 5 6 7 8 9
[43][28] *0>1 2 3 4 5 6 7 8 9
[44][28] 0 *1>2 3 4 5 6 7 8 9
[45][28] *0>1 2 3 4 5 6 7 8 9
```

Exercise 4-1: sort_bubble1.c : PAD expression₅

Original sort_bubble.c



Visualized sort_bubble1.c



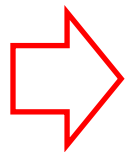
Previous lecture Exercise 4-2: turn over of array sort_bubble2.c

- Modify sort_bubble1.c to create a program sort_bubble2.c that starts the comparison from the back of the array and replaces the smaller elements in the forward.

sort_bubble1.c

$A[j] > A[j+1]$ then exchange

| | j | j+1 | |
|----------|----|---------------|-------------------|
| [1][0] | *8 | >2 | 7 4 5 6 9 0 1 3 |
| [2][1] | 2 | *8 | >7 4 5 6 9 0 1 3 |
| [3][2] | 2 | 7 | *8 >4 5 6 9 0 1 3 |
| [4][3] | 2 | 7 4 | *8 >5 6 9 0 1 3 |
| [5][4] | 2 | 7 4 5 | *8 >6 9 0 1 3 |
| [6][5] | 2 | 7 4 5 6 | *8 >9 0 1 3 |
| [7][5] | 2 | 7 4 5 6 8 | *9 >0 1 3 |
| [8][6] | 2 | 7 4 5 6 8 0 | *9 >1 3 |
| [9][7] | 2 | 7 4 5 6 8 0 1 | *9 >3 |



sort_bubble2.c

$A[j-1] > A[j]$ then exchange

| | j-1 | j | |
|----------|-----|----|-------------------|
| [1][0] | 8 | 2 | 7 4 5 6 9 0 <1 *3 |
| [2][0] | 8 | 2 | 7 4 5 6 9 <0 *1 3 |
| [3][0] | 8 | 2 | 7 4 5 6 <9 *0 1 3 |
| [4][1] | 8 | 2 | 7 4 5 <6 *0 9 1 3 |
| [5][2] | 8 | 2 | 7 4 <5 *0 6 9 1 3 |
| [6][3] | 8 | 2 | 7 <4 *0 5 6 9 1 3 |
| [7][4] | 8 | 2 | <7 *0 4 5 6 9 1 3 |
| [8][5] | 8 | <2 | *0 7 4 5 6 9 1 3 |
| [9][6] | <8 | *0 | 2 7 4 5 6 9 1 3 |

```
for (i = 0; i < (NUM - 1); i++) {
    for (j = 0; j < NUM-i-1; j++) {

        /* compare and exchange */
        if (A[j] > A[j+1]) {

            }

    }
}
```

```
for (i = 0; i < (NUM - 1); i++) {
    for (j = ?; j > ?; j--) {

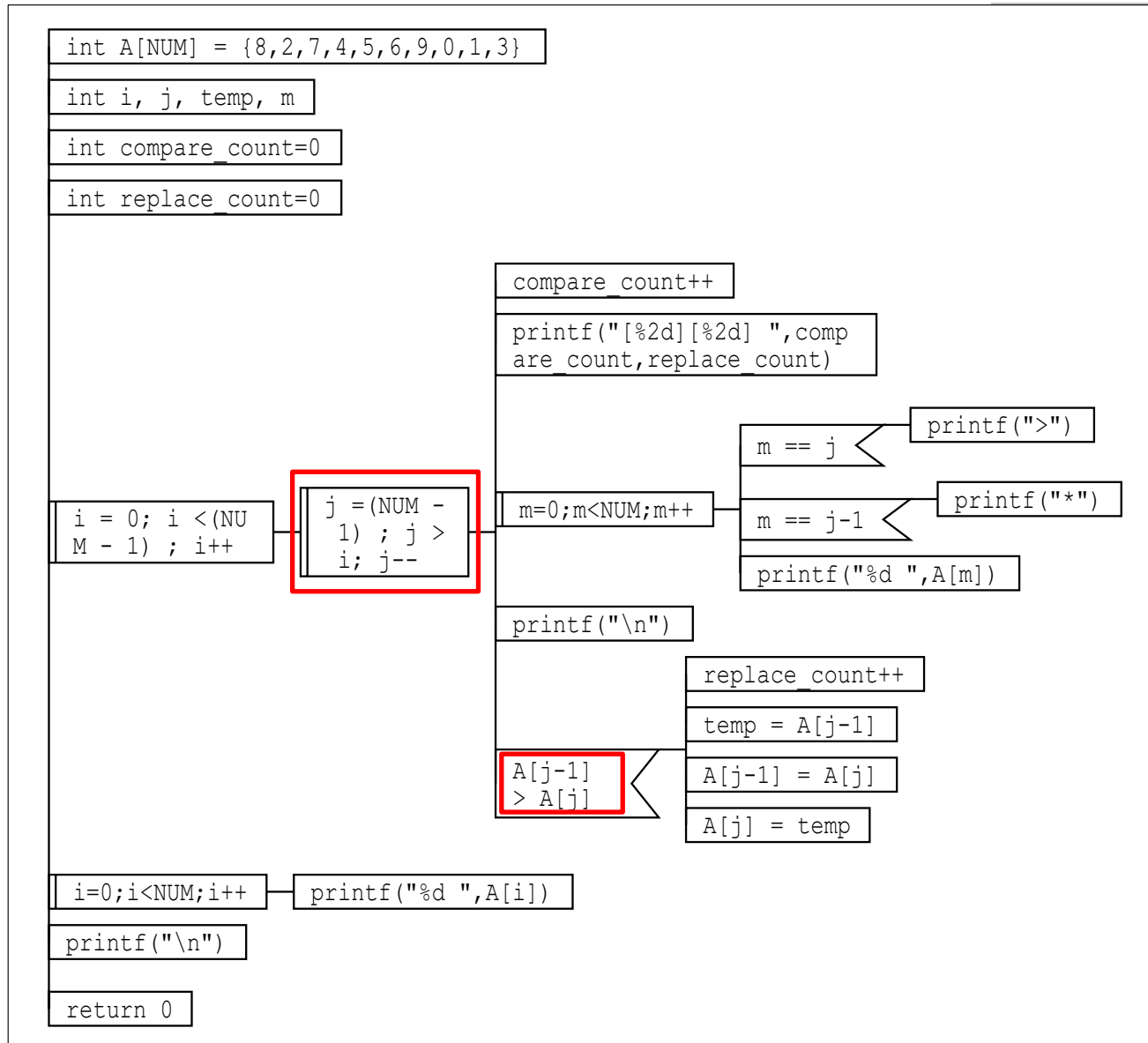
        /* compare and exchange */
        if (A[j-1] > A[j]) {

            }

    }
}
```

Exercise 4-2: PAD expression of sort_bubble2.c

7



Exercise 4-3: Selection sort sort_select.c

8

■ Replace bubble sort with selection sort.

- The program name is sort_select.c
- Visualize and compare the difference with bubble sort
- (Check the number of comparisons and the number of replacements)

(Example)

"*" For the element to be replaced (0, 1, 2 ...)

"!" For the element of the minimum value at the time before comparison

">" For the element to be compared

(want to show that "!" And ">" are compared)

[Comparison count] and [Replacement count] are displayed at the beginning of the array.

Let's check the difference in the number of replacements compared to bubble sort

Ref: Select-sort with Gypsy folk dance

<https://www.youtube.com/watch?v=Ns4TPTC8whw>

Algorithm of selection sort

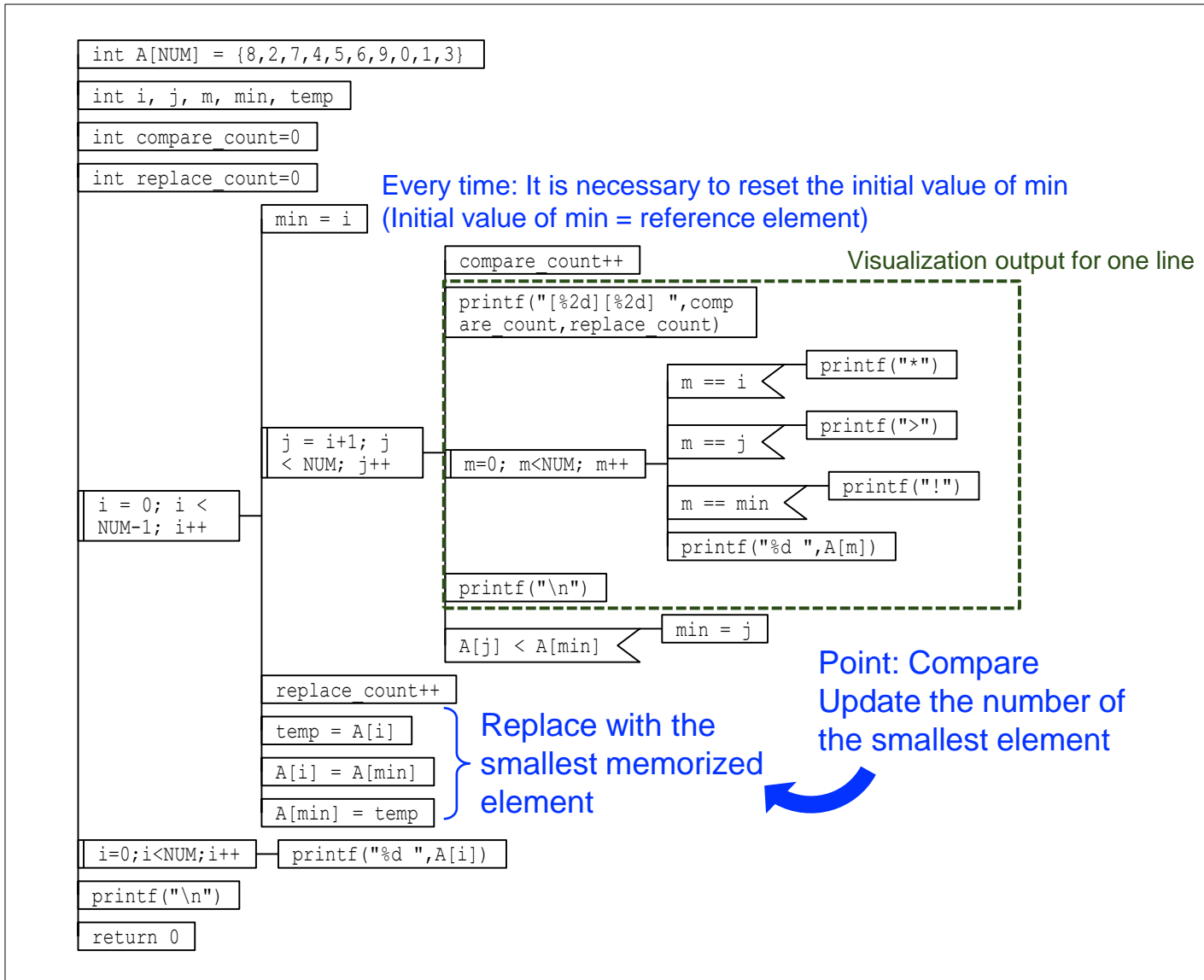
1. Find the smallest value in the data column and exchange it for the first element.
2. Next, find the smallest value in the second and subsequent data columns and exchange it for the second element.
3. Repeat this until the end of the data string

```
[ 1][ 0] *!8 >2 7 4 5 6 9 0 1 3
[ 2][ 0] *8 !2 >7 4 5 6 9 0 1 3
[ 3][ 0] *8 !2 7 >4 5 6 9 0 1 3
[ 4][ 0] *8 !2 7 4 >5 6 9 0 1 3
[ 5][ 0] *8 !2 7 4 5 >6 9 0 1 3
[ 6][ 0] *8 !2 7 4 5 6 >9 0 1 3
[ 7][ 0] *8 !2 7 4 5 6 9 >0 1 3
[ 8][ 0] *8 2 7 4 5 6 9 !0 >1 3
[ 9][ 0] *8 2 7 4 5 6 9 !0 1 >3
[10][ 1] 0 *!2 >7 4 5 6 9 8 1 3
[11][ 1] 0 *!2 7 >4 5 6 9 8 1 3
[12][ 1] 0 *!2 7 4 >5 6 9 8 1 3
[13][ 1] 0 *!2 7 4 5 >6 9 8 1 3
[14][ 1] 0 *!2 7 4 5 6 >9 8 1 3
[15][ 1] 0 *!2 7 4 5 6 9 >8 1 3
[16][ 1] 0 *!2 7 4 5 6 9 8 >1 3
[17][ 1] 0 *2 7 4 5 6 9 8 !1 >3
[18][ 2] 0 1 *!7 >4 5 6 9 8 2 3
...
```

Minimum value is determined when it reaches to end

Exercise 4-3 PAD expression of sort_select.c

9



Exercise 4-4: Polynomial approximation

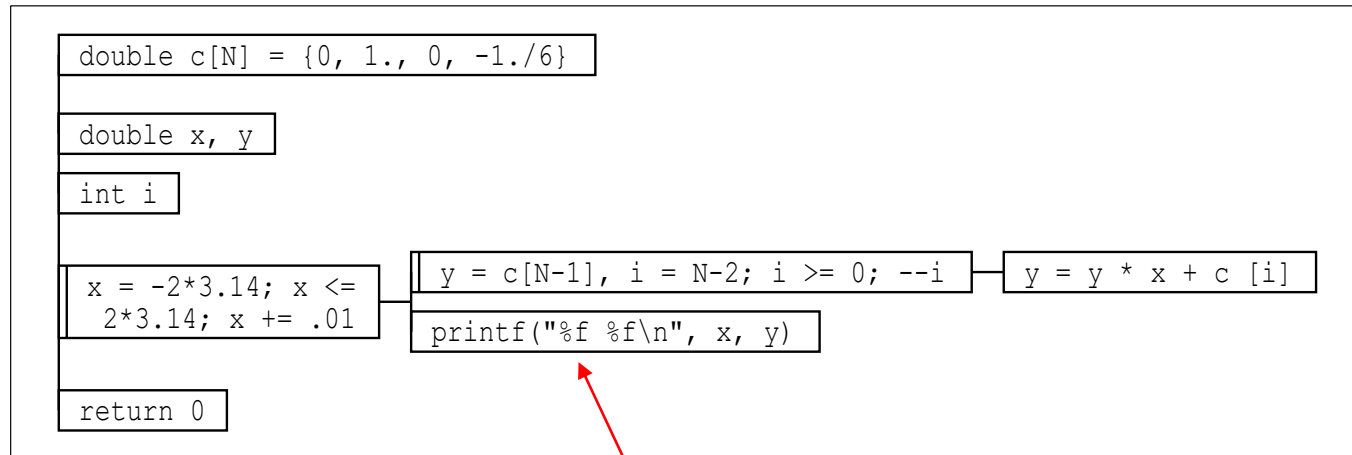
10

- Modify the polynomial program `poly.c` to calculate the 3rd and 5th order McLaughlin expansions of the sine function $\sin(x)$. Create a graph with Excel, and compare the degree of approximation.
 - The calculation range of x is between -2π and 2π .
- Submit the followings.
 1. `poly_sin3.c` that expands $\sin(x)$ into 3rd order McLaughlin series
 2. `poly_sin5.c` which expands $\sin(x)$ into 5th order McLaughlin series
 3. Make a graph of these functions using any plot software

Exercise 4-4: PAD expression

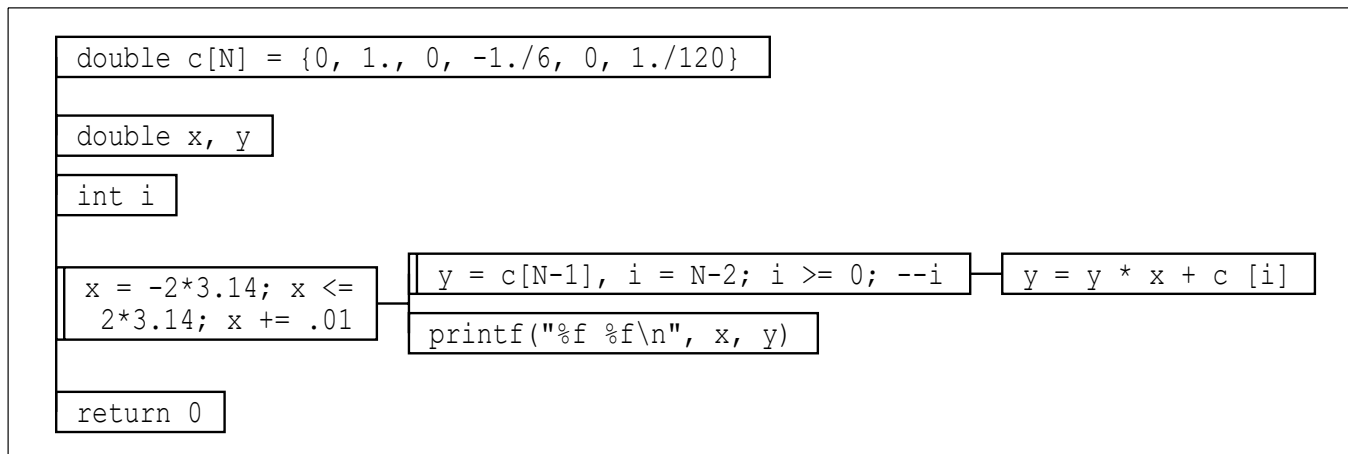
11

poly_sin3.c



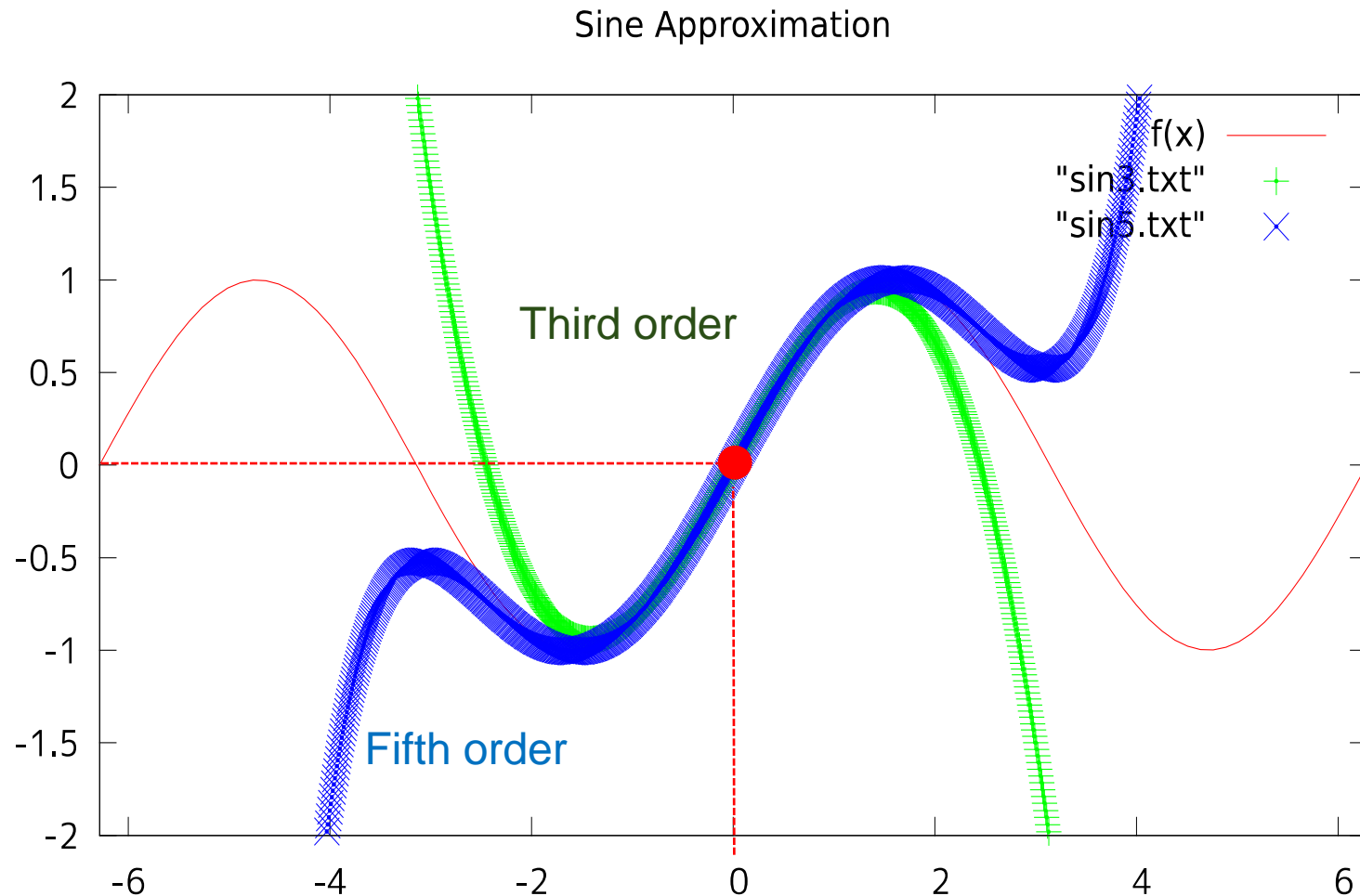
You can write `printf("%f %f %f\\n", x, y, sin(x))` using `<math.h>`

poly_sin5.c



Exercise 4-4: Example of graph

12



McLaughlin expansion is approximation around $x=0$.

More accurate with increasing order

Practice of Information Processing

(IMACU)

Fifth lecture(Part 2)

Function definition, call and, recursive call

Makoto Hirota

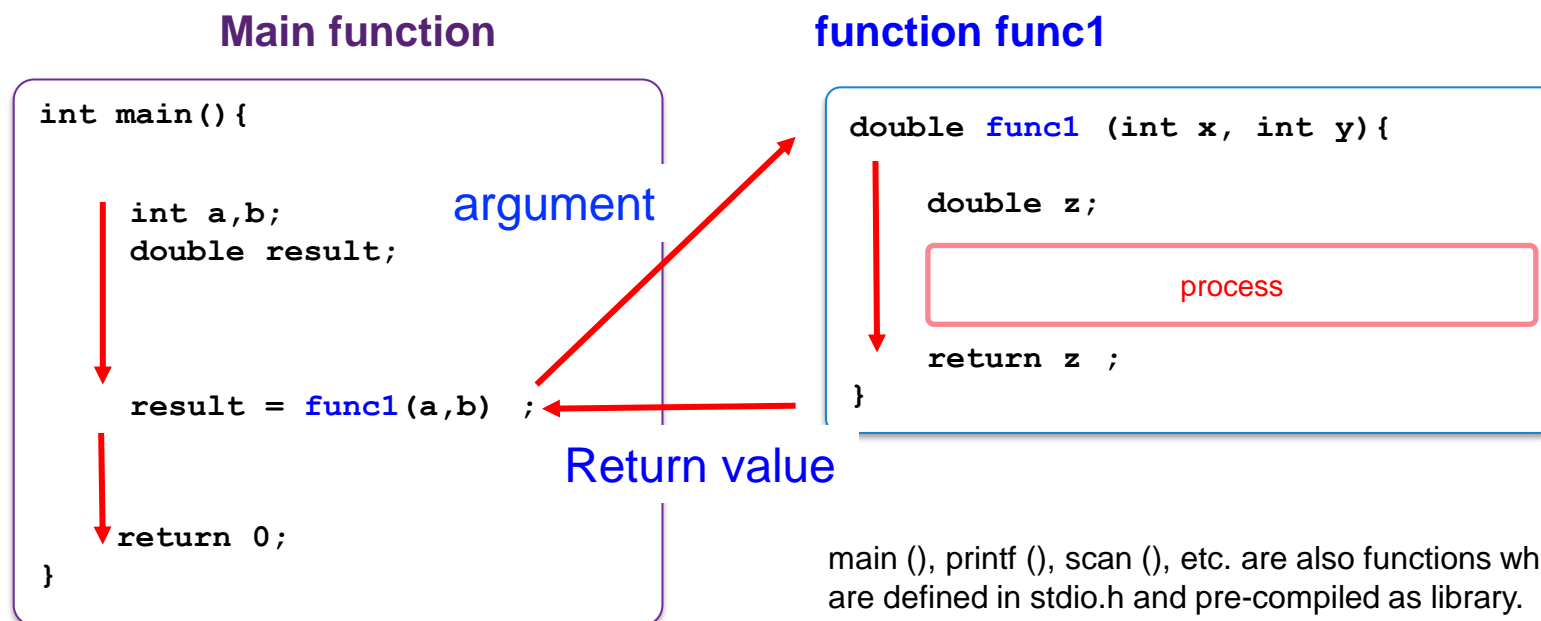
- Example answer of previous exercise
- Definition of function
 - What is function
 - Definition of self-made function
 - Prototype declaration
 - Role of function
- Function with no return value “procedure”
- Recursive call of function
- Data type
- What is structure?

What is function?

15

■ function

- A program that extracts **specific processes** and puts them together as a separate program.
- By **calling a function** in the middle of a program, a specific process is run and the result is returned (**return value**).
- When calling a function, you can specify the value (**argument**) to be input.



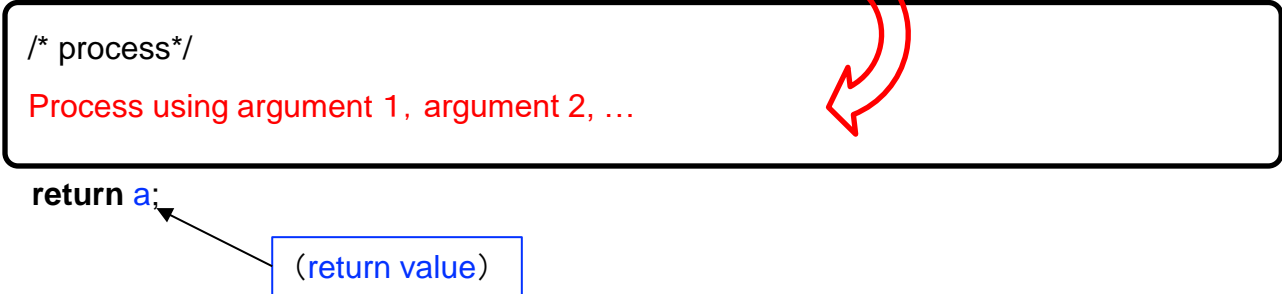
■ Functions have four main roles:

- The **structure of a program** is made easy to understand by isolating a specific process into a function
- A function can be **used repeatedly** at different locations of a program.
- A function can be **reused** in another program.
- **Black boxing**: you can use a function created by others without knowing the process inside it. (= Library)

■ 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.



(return value)

Example (a function that adds two variables)

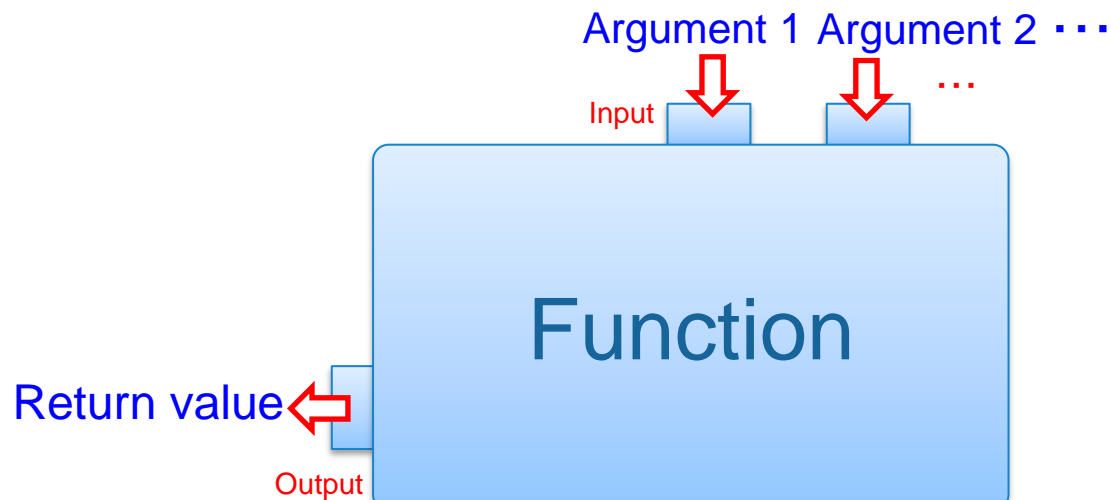
```
/* User defined add_func */  
  
int add_func(int x, int y)  
  /***** variable declaration*****/  
  int a;  
  /***** processing contents*****/  
  a = x + y;  
  /***** return value *****/  
  return a;  
}
```

Number of arguments and return values

18

In C language,

- **Multiple arguments** can be given to a function.
(no argument is also OK)
- **At most one return value** can be output.



If it is zero, do not write anything or write "void"

What if we want to return multiple results?

Method 1: Return only the “first address of the output data” (**pointer**)
(next lectures)

Method 2: Use **global variables** (this lecture)

Ex) Function add_func.c

19

- Compile and execute the sample program add_func.c, and check the contents of the program.

```
#include <stdio.h>

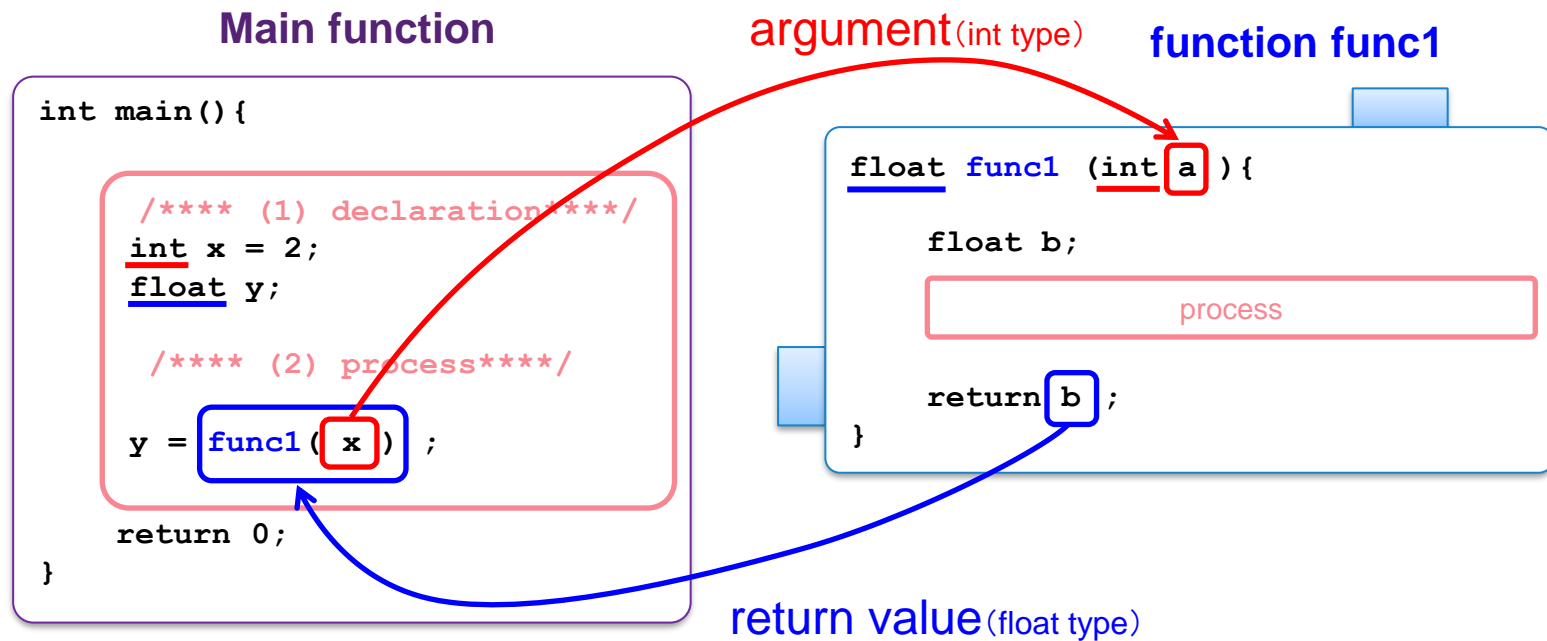
/* User defined add_func */
int add_func(int x, int y){
    /*** variable declaration***/
    int a;
    /*** processing contents***/
    a = x + y;
    /*** return value ***/
    return a;
}

int main(){
    /*** variable declaration***/
    int p,q;
    /*** processing contents***/
    q = 2;
    p = add_func(q,3);
    printf("%d\n", p);

    return 0;
}
```

Supplement: Matching the types of arguments and return values ²⁰

- When calling a function, the "types" of the arguments and return value must match the definition of the function.



If there is a mismatch of type, the compiler shows a warning message.
Use the explicit type conversion ("cast") if you really want to change the type.

How to add a user function

21

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

```
int func1 (int x){  
    process  
    return y;  
}
```

User-function func1

```
int func2 (int x, int y){  
    process  
    return z;  
}
```

User-function func2

```
int main(){  
    process  
    y = func1 (x);  
    z = func2 (p, q);  
    return 0;  
}
```

Main function

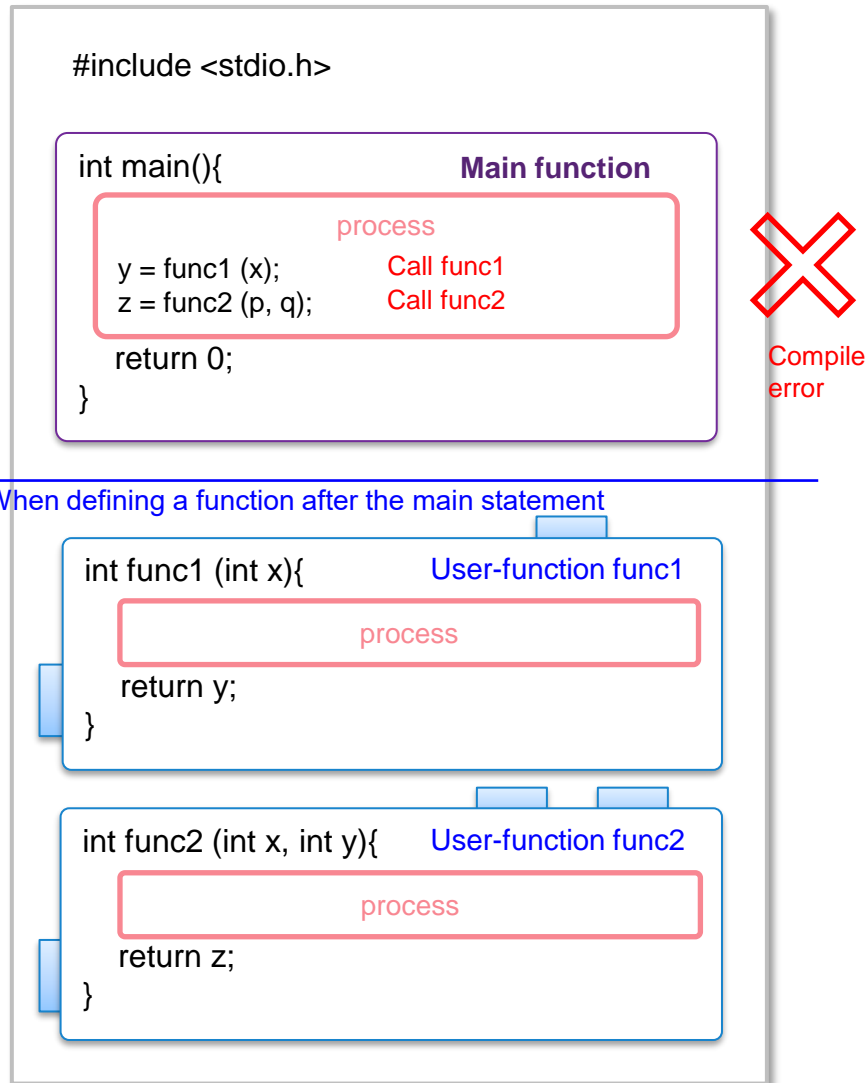
- In fact, functions can be defined in a variety of positions in the code (even in a separate file).
- However, the simplest way is

**Method1: Write the definition before the main function
(To be precise, before being called by other functions)**

* The main function is also a kind of function, but it is always called first in a program.

When a function is defined “after” the main function, ...

22



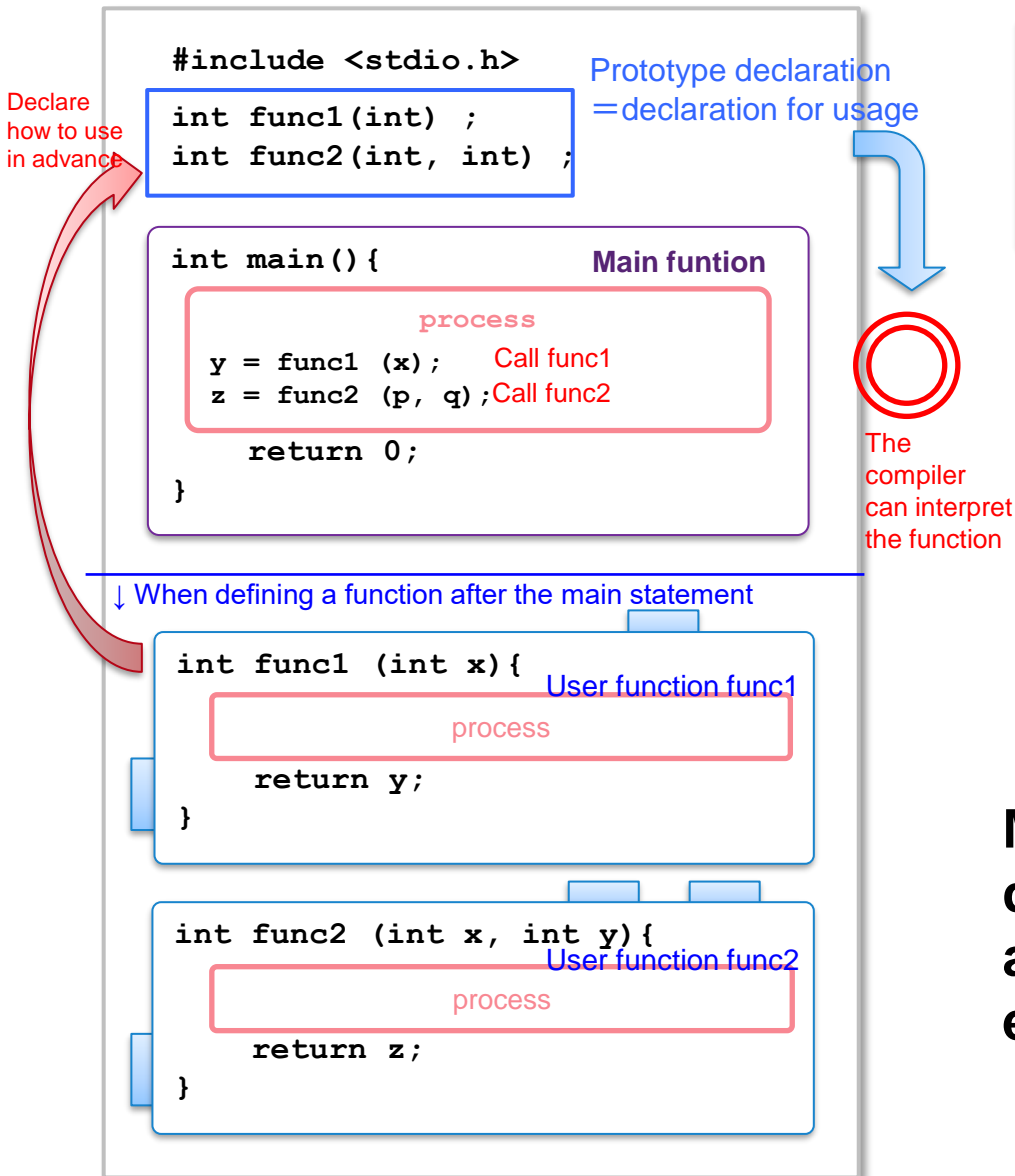
If you write your own function after the main function, a compilation error will occur.



Your function is not yet defined when it is called.

Workaround: Function prototype declaration

23



If you define a function after the main function, you need to teach the compiler how to use the function with a prototype declaration.

◆ Function prototype declaration

Return type **Function name** (**type of argument 1**, **type of argument 2**, ...);

Only the function name and the "types" of the argument and return value are written.
No variable name for argument required.
";" is required at the end.

Method2: Write the prototype declaration at the beginning and write the main definition elsewhere

Example: Sample code add_func2.c

Check it out for yourself

- Move the code of the user function `add_func.c` of the sample program after the main function and compile it.
- Add the following sentence to `add_func.c` and compile again.

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

```
int add_func(int, int);
```

```
int main() {
```

```
...
```

```
}
```

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

```
/* User defined add_func */
```

```
int add_func(int x, int y)
```

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

```
int a;
```

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

```
a = x + y;
```

```
/* return value */
```

```
return a;
```

```
}
```

```
int main() {
```

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

```
int p, q;
```

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

```
q = 2;
```

```
p = add_func(q, 3);
```

```
printf("%d\n", p);
```

```
return 0;
```

```
}
```

Move here

What is the role of header file?

25

- The prototype declaration of many functions is written in a header file (`myfunc.h`), which can be read at once by

```
#include "myfunc.h"
```

- This is the reason for including the header file at the beginning.
- Then, the definitions of functions may be written in a separate file, say `myfunc.c` (Method3).

```
$ gcc program.c myfunc.c -o program
```

You can reuse your functions as “library”.

(Have a look at `C:\mingw64\include\math.h`)

Function with no return value

- In C language, you can define a function with no return value.

```
void func(type of argument1 argument1,  
          type of argument 2 argument2, ...)  
{  
    process  
  
    (no return statement)  
}
```

Similarly, you can define a function that takes no argument.

```
void func(void)  
{  
    process  
}
```

- Parentheses () are required even when calling without arguments

```
value = func ();
```

- Use “void” in prototype declaration

- When there is no argument: `int func (void);`
- When there is no return value: `void func (int x);`
- When neither is available: `void func (void);`

- It is customary for the return value (int) of `main` function to be used for **error handling**.

- By adding

```
#Include <stdlib.h>
```

in the header part, you can use the `exit` function at any position to terminate the program.

- `exit (0)` ends normally (same as exiting with `return 0` at the end of the main statement).
- `exit (1)` ends abnormally due to `error1`. (You may define the error type.)
- `exit (2)` ends abnormally due to `error2`.
- ...

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

28

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

29

■ 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: Tips

■ Example of main function

```
int main() {  
  
    /*** variable declaration***/  
    int a,b,d;  
  
    /*** processing contents***/  
    printf("input a ->");  
    scanf("%d", &a);  
    printf("input b ->");  
    scanf("%d", &b);  
  
    d = gcd(a,b);  
  
    printf("Least common multiple of %d and %d is %d\n", a, b, a*b/d);  
    return 0;  
}
```

■ Regular use

```
f () {  
    g () ; Other functions  
    ...  
    h () ; Other functions  
    ...  
}
```

■ Recursive calling of function

```
f () {  
    ...  
    f () ; Call the same  
           function in the  
           function definition  
    ...  
}
```

When the same function f is called again in the definition of function f , it is called a recursive call.

* In the Euclidean algorithm used in the previous exercise 5-2, the inductive algorithm that repeatedly calls the same process can be executed by recursive call of the function.→ Exercise 5-3

Factorial calculation (recursive call version) r_factorial.c

32

Sample code: r_factorial.c

```
#include <stdio.h>

int rfact(int n)
{
    int m;

    if (n == 0){ /* when n=0(0!=1)*/
        m = 1;
    }
    else{ /* when not n=0*/
        m = n * rfact(n - 1);
    }
    return m;
}

int main()
{
    int num, ans;

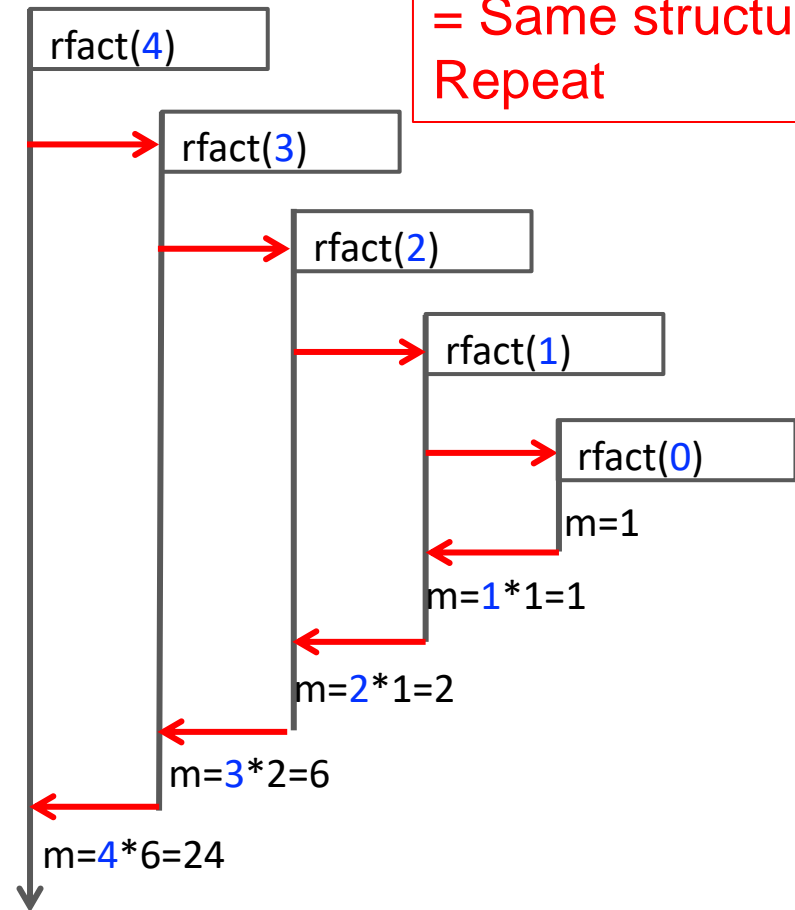
    printf("Input natural number= ");
    scanf("%d", &num);

    /* factorial by func. rfact()*/

    ans = rfact(num);

    /* output results*/
    printf("%d! = %d\n", num, ans);
    return 0;
}
```

ex: case of num = 4



Exercise 5-3: Recursive call version of the least common multiple lcm2.c

33

Submission required

- Create a program lcm2.c that finds the least common multiple by rewriting the function gcd that finds the greatest common divisor in the previous exercise 5-2 using the recursive call of the function.
- Name the function rgcd()

Greatest_commen_divisor = rgcd(a,b)

- 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 $b = r$
 - ➡ Call rgcd () again with $a \leftarrow b$ and $b \leftarrow r$
 - Greatest_commen_divisor = rgcd(b,r)

Effective range (“scope”) of variables

34

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

```
int total_number; ← location of declaration
```

Function **add_total**

```
int add_total (int x){
```

```
    int answer; ← location of declaration
```

```
    answer = total_number + x;
```

```
    return answer;
```

```
}
```

Main function

```
int main(){
```

```
    int answer; ← location of declaration
```

```
    total_number = 0;
```

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

```
        answer = add_total(i);
```

```
    }
```

```
    printf("Total is %d\n", answer);
```

```
    return 0;
```

```
}
```

total_number is defined outside functions



Variables that can be referenced by any function
= **Global variable**

answer is defined in each function



Variables that can only be referenced within that function
= **Local variable**

answer in the `add_total` function and **answer** in the `main` function are treated as different things (= different memory areas) even if they have the same name.

Variable sharing between functions using global variables

35

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

```
int g_count;
```

```
int func1 (void) {    User defined func1
```

```
    /* process */
```

```
    g_count ++;
```

```
    return 0;
```

```
}
```

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

```
    /* process */
```

```
    g_count =0;
```

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

```
        func1();
```

```
    }
```

```
    printf("%d\\n", g_count);
```

```
    return 0;
```

```
}
```

- Variables declared outside the function are called global variables and can be called from all functions.

Example

- Since the variable `g_count` is declared outside the function, it can be called and operated from both the main function and the `func1` function.
- In this example, `g_count` is incremented by 1 each time `func1 ()` is called, and finally the value of `g_count` is displayed and the process ends.

Variable name used in the function = local variable

36

```
#include <stdio.h>

int rfact(int n)
{
    int m;

    if (n == 0){ /* n=0(0!=1)*/
        m = 1;
    }
    else{ /* NOT n=0 */

        m = n * rfact(n - 1);
    }
    return m;
}

int main()
{
    int num, ans;

    printf("Input natural number= ");
    scanf("%d", &num);

    /* factorial is computed by
       function rfact()*/
    ans = rfact(num);

    /* display results */
    printf("%d! = %d\n", num, ans);
    return 0;
}
```

What if we
rename the
variable m in
rfact () to
ans?



```
#include <stdio.h>

int rfact(int n)
{
    int ans;

    if (n == 0){ /* n=0(0!=1)*/
        ans = 1;
    }
    else{ /* NOT n=0 */

        ans = n * rfact(n - 1);
    }
    return ans;
}

int main()
{
    int num, ans;

    printf("Input natural number= ");
    scanf("%d", &num);

    /* factorial is computed by
       function rfact()*/
    ans = rfact(num);

    /* display results */
    printf("%d! = %d\n", num, ans);
    return 0;
}
```

Nothing wrong!

- Example answer of previous exercise
- Definition of function
 - What is function
 - Definition of self-made function
 - Prototype declaration
 - Role of function
- Function with no return value “procedure”
- Recursive call of function
- Data type
- What is structure?

Practice of Information Processing

(IMACU)

Fifth Lecture(Part 3) Data type, structure

Makoto Hirota

- Example answer of previous exercise
- Definition of function
 - What is function
 - Definition of self-made function
 - Prototype declaration
 - Role of function
- Function with no return value “procedure”
- Recursive call of function
- Data type
- What is structure?

Array (review)

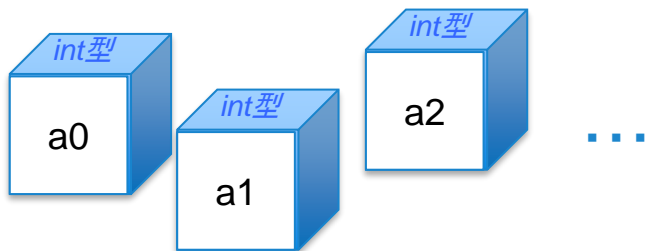
40

- The way managing many variables of the same type by assigning **one name** and **consecutive numbers** (index)

Ex) When you want to prepare 100 int type variables

<Previous variable declaration>

```
int a0, a1, a2, ..., a99;  
a0=10;  
a1=20;
```



schematic of single variable

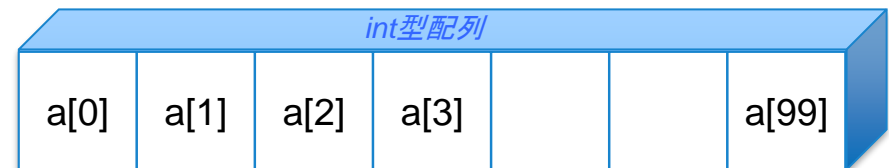
A separate data storage location (memory) is prepared for each variable



<Variable declaration using array>

```
int a[100];  
a[0]=10;  
a[1]=20;
```

Add [] (braces) after the variable name



Schematic of array

Data in a continuous memory space
Storage space is secured

Data type summary

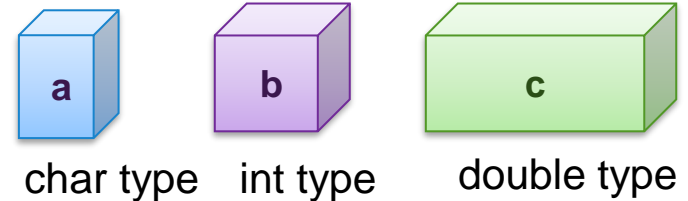
41

Variable

- Data type (box) whose size differs depending on the type of variable

```
char a; int b; double c;
```

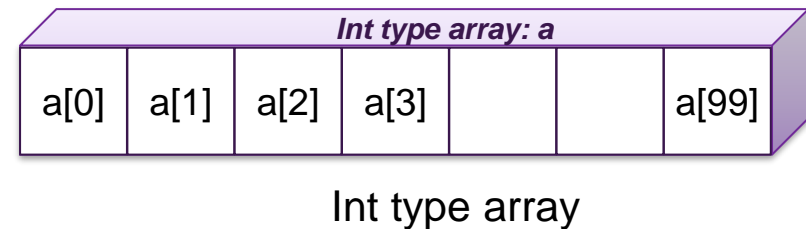
- located at random places of memory space



Array

- Data type (box) in which variables of the **same type** are arranged

```
int a[100];
```



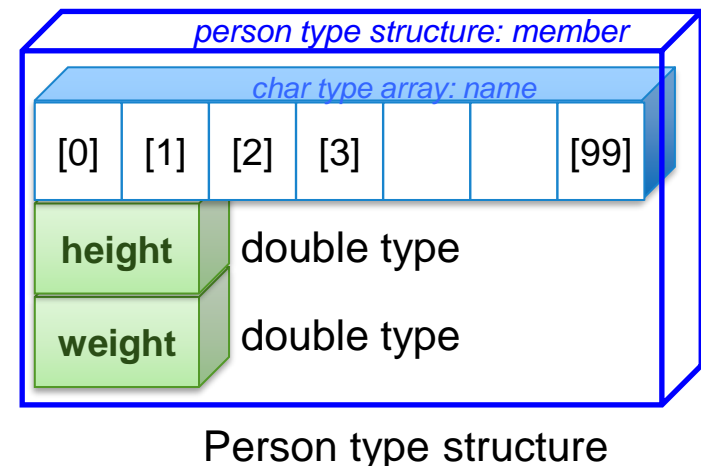
Structure

- Data type (box) that collects variables of **different types**
- You can define the name of the **"type"**.

```
struct person member;
```

= Variable named "member" of "person" type structure

* Be careful not to confuse the structure name because it is a definition of "type" and not a variable name.



- 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)
```

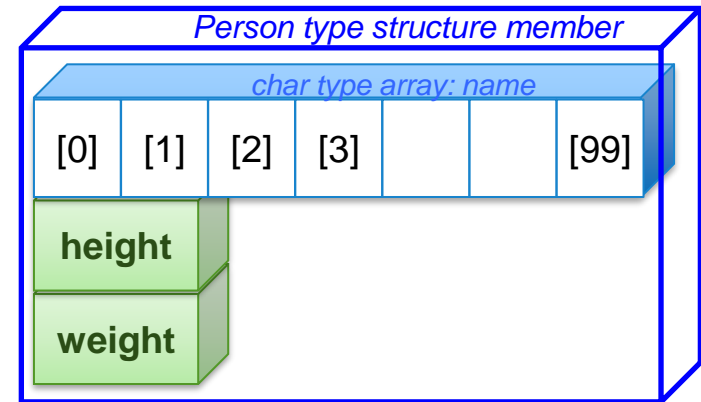
Type name

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

Ex: Program example using a structure struct_ex.c

43

```
#include <stdio.h>
#define NAME_LEN 100

/* definition of sturucture person*/
struct person{
    char name[NAME_LEN];
    double height;
    double weight;
};

int main() {

    /* variable declaration */
    struct person member;

    /* processing contents */
    /* data input */
    printf("Name? ");
    scanf ("%s",member.name);
    printf("Height? ");
    scanf ("%lf",&member.height);
    printf("Weight? ");
    scanf ("%lf",&member.weight);

    /* data display */
    printf("Name:   %s\n",member.name);
    printf("Height: %.1f\n",member.height);
    printf("Weight: %.1f\n",member.weigth);

    return 0;
}
```

- Structure definition
- Assignments and references to struct members

※ Since name is an array, & is not required for scanf

※ To treat the name including spaces, write the scan part as follows.

fgets(member.name, sizeof(member.name), stdin);

Supplement: Define an alias for the structure

44

- You can shorten the name of data type `struct person` into one word, using `typedef`

<1> No alias

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

```
int main(){

    //declaration
    struct person member;

    .
    .

}
```

<2> Alias after declaration

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

```
int main(){

    //declaration
    st_person member;

    .
    .

}
```

<3> Alias with declaration

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

```
int main(){

    //declaration
    st_person member;

    .
    .

}
```

Supplement: Initialize when declaring a structure

45

- 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);  
  
}
```

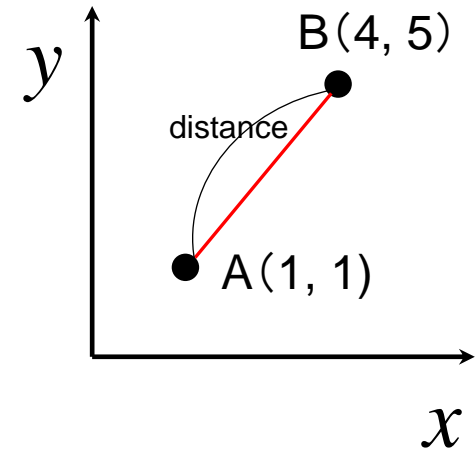
Declaration of structure

Exercise 5-4 struct_point.c

46

Submission required

- 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 " $\text{pow}(x, 2)$ " 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
```

Array of structure

- 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);
    }
}
```

Exercise5-5: struct_array.c

- 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

- Example answer of previous exercise
- Definition of function
 - What is function
 - Definition of self-made function
 - Prototype declaration
 - Role of function
- Function with no return value “procedure”
- Recursive call of function
- Data type
- What is structure?

■ Mechanism of computer

■ Pointer

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