Practice of Information Processing
（IMACU）

Fifth lecture(Part 2)
Function definition, call and, recursive call

Makoto Hirota

Contents of this lecture

- Example answer of previous exercise

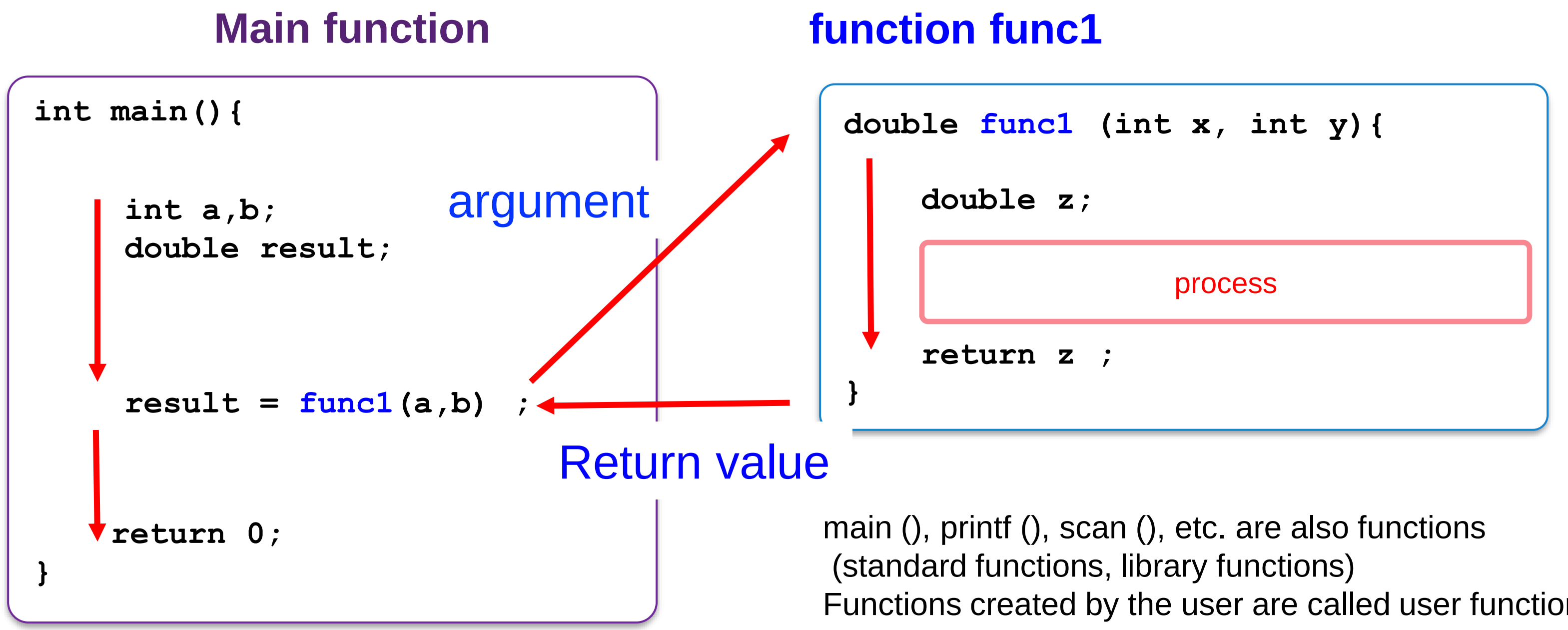
- Definition of function
  - What is function
  - Definition of self-made function
  - Prototype declaration
  - Role of function
- Function with no return value “procedure“
- Recursive call of function

- Data type
- What is structure?

What is function?

- 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 executed and the result is returned (return value).
  - When calling a function, you can specify the value (argument) to be input.

<Technical terms>
Value to enter
= Argument, actual arguments
Output value
= Return value value,
function value



```
int main(){

    int a,b;
    double result;

    result = func1(a,b) ;

    return 0;
}
```

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

main (), printf (), scan (), etc. are also functions
(standard functions, library functions)
Functions created by the user are called user functions.

How to define a function

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)

(For a function that adds two variables)

1) Determine the function name (`add_func`)
2) Determine arguments
   (2 variables of `int type x, y`)
3) Determine the return value
   (`int type a` as a solution)
4) Write the process
5) Return the return value

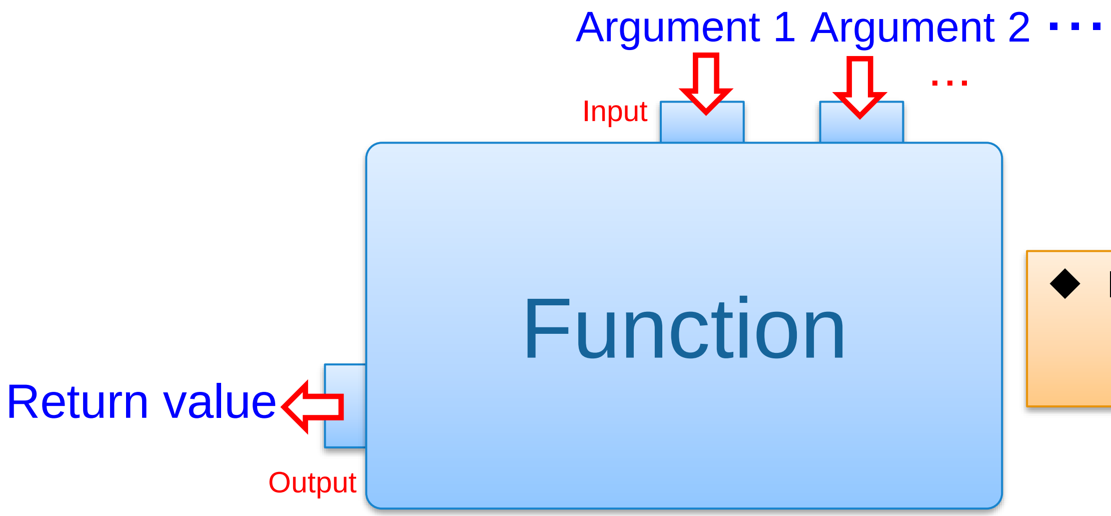
Example

```
/* User defined add_func */

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

Function: Number of arguments and return values

- Number of arguments and return values
  - Multiple arguments are possible (no argument is also OK)
  - Only one return value (no return value is also OK)

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



- In C language
  - Multiple arguments are possible
  - Only one return value

What if we want to return multiple results?
Method 1: Specify the data storage destination with a pointer and have the result entered (next lectures)
Method 2: Rewrite global variables (next lectures)

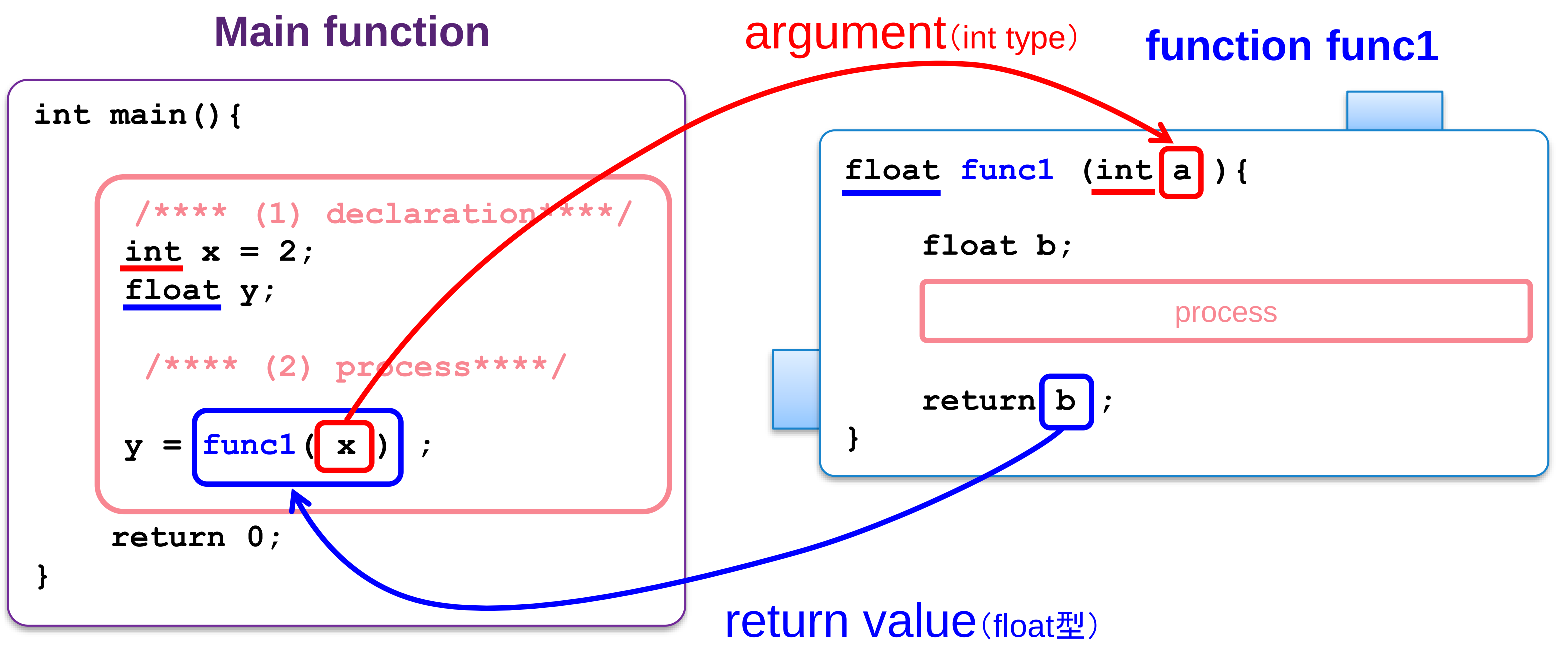
Ex) Function add_func.c

- 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

- When calling a function, it is necessary to match the "type" of the argument / return value of the defined function.

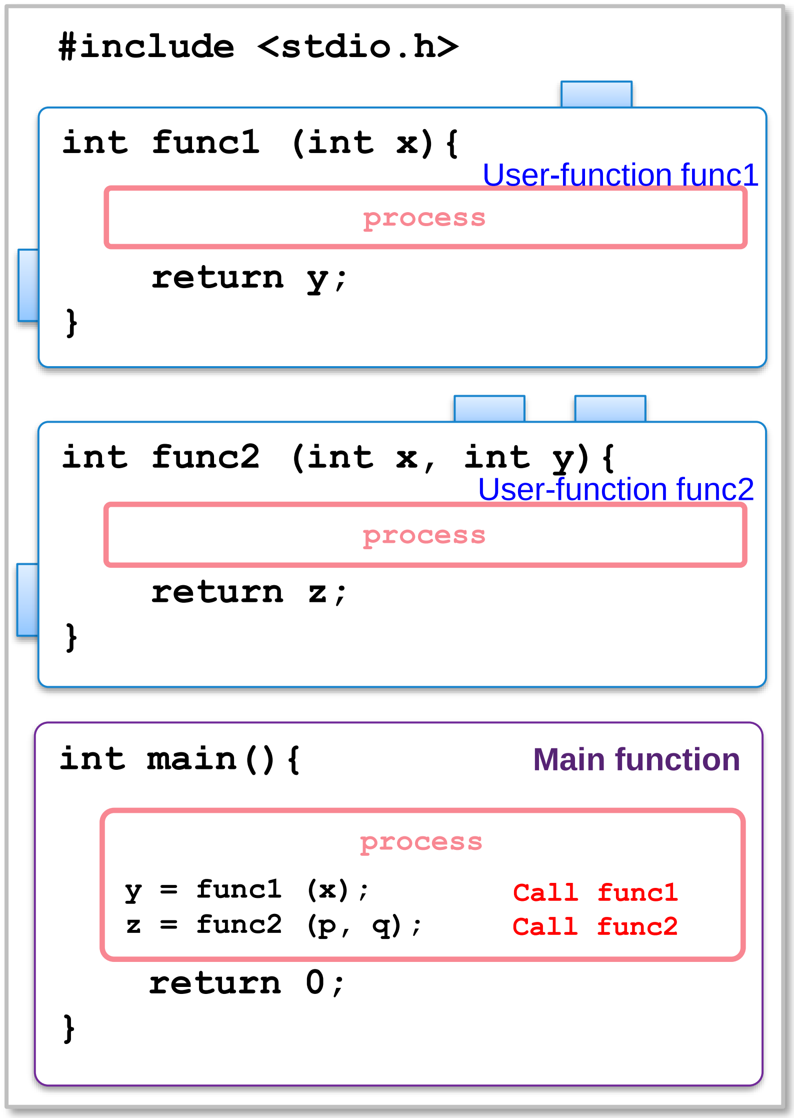


```
int main(){

    /**** (1) declaration****/
    int x = 2;
    float y;

    /**** (2) process****/

    y = func1( x ) ;

    return 0;
}
```

```
float func1 (int a ){
    float b;

    return b ;
}
```

In this example, the argument defined by the function is int type and the return value is float type, so in the main function, the types of the argument x (int type) when calling the function func1 and the return value assignment destination y (float type) are matched between at `main` and at `func1`.

How to add a user function



```
#include <stdio.h>

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

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

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

Write outside the main function in order

- In fact, the function definition can be written at any position as long as it is the main function. You can write it in a separate file.
- However, it is necessary to share conventions such as function arguments and return value form (= header file that #includes the file that describes the convention at the beginning).

How to share a defined function type
= Prototype declaration

* The main function is also a type of function, but in a program, there is a rule that the main function is executed first, so we will create a processing flow in the main function.

Example: If you need a prototype declaration

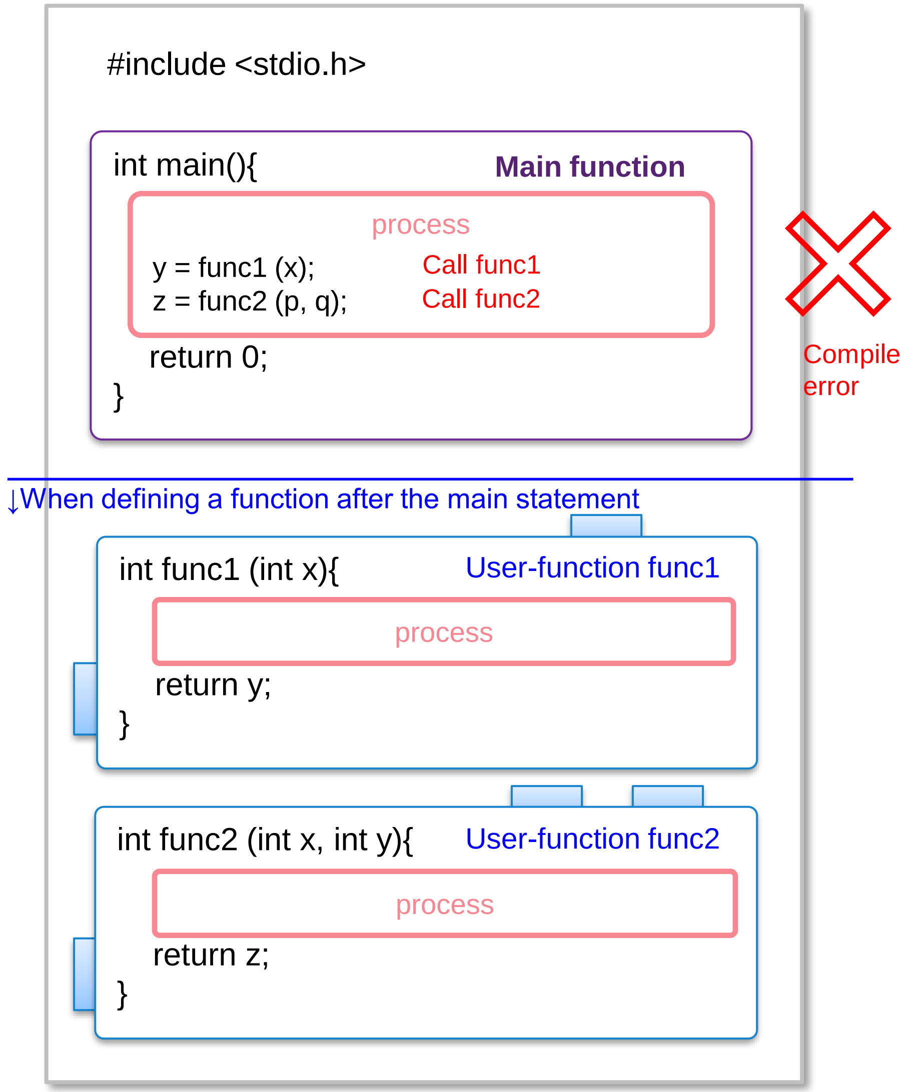
- Move the code of the user function add_func.c of the sample program after the main function and check the operation.
- Add the following sentence to add_func.c and compile again.

```
#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

In the case the compiler does not know the definition of the function



```
#include <stdio.h>

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

↓When defining a function after the main statement

```
int func1 (int x){

  return y;
}

int func2 (int x, int y){

  return z;
}
```

The definition of the function can be anywhere in the program. However, if you write your own function after the main function, a compile error will occur.

Because the compiler doesn't know the definition of the user function.

Workaround: Function prototype declaration (prototype declaration)

Declare how to use in advance

Prototype declaration
＝declaration for usage

```
#include <stdio.h>
int func1(int) ;
int func2(int, int) ;
```

Main funtion

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

The compiler can interpret the function

↓ When defining a function after the main statement

User function func1

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

User function func2

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

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 "type" of the argument and return value are shown.
No variable name for argument required
";" Is required at the end

In many programs, the prototype declaration is written in the header file (.h). This is the reason for including the header file when using the function.

Example: Sample code add_func2.c

Function with no return value (= "procedure")

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

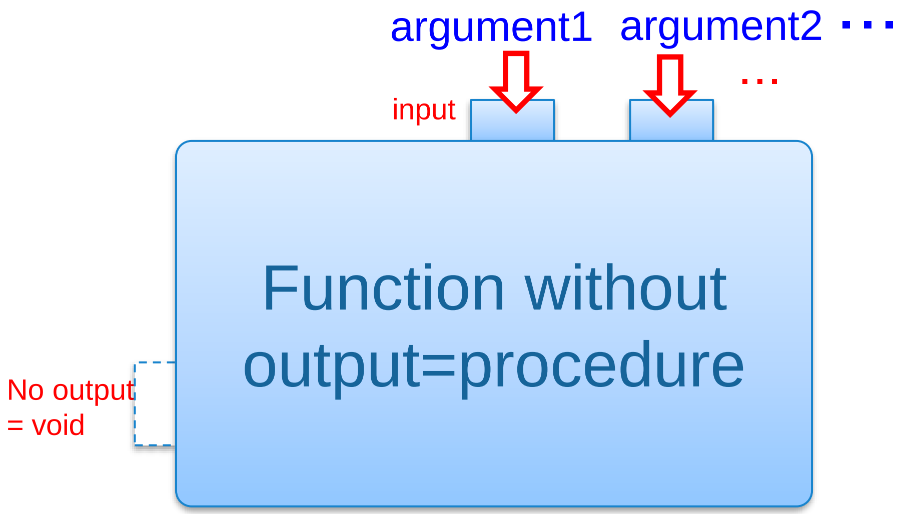
```
void func(type of argument1 argument1,
         type of argument 2 argument2,・・・)
{
    process

    (no return statement)
}
```

Such a function is sometimes called a "procedure"

* The term "Procedure" seems to be a remnant of the programming language Pascal. In C language, a function with no return value can be considered to be the same kind of the function.

Similarly, you can define a return value or a function that takes no arguments.

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



Supplement: Omission of arguments and return values

- Why the return type of main is omitted
  - If you do not declare the type of the function, there is a rule that the int type is used by default, so you often see a program in which nothing is written before the main function (also in textbooks). However, in order to clarify the intention of the creator's function definition, it is recommended to write it in full.
  - In other words, it is not void main (), so be careful. This is why return 0 is needed at the end of the main function.
- How to use a function with no arguments
  - Parentheses () are required even when calling without arguments
    - `value = func ();`
- Function prototype declaration with no return value or arguments
  - Use "void"
    - 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);`

Reason using functions

- Functions have four main roles:
  - Structure: The structure of a program is made easy to understand by dividing complicated and intertwined procedures into elemental functions (= functions).
    - It makes it easier to create long programs.
  - Repeated use: Use the function many times in the same program.
    - Program gets shorter
  - Reuse: Reuse a function in another program
    - The effort of creating a program is reduced.
  - Black boxing: Makes the function available even if you do not know the processing inside the function.
    - Facilitates program collaboration.

Review: A predefined function

- Math library
  - A collection of math-related functions prepared in C language
  - The following two are required to use
    - Write #include <math.h> at the beginning of the program
    - Add -lm option at compile time
      - gcc –Wall–o test test.c –lm
  - Argument and return value are double type

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

- 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 without using arithmetic operators.
  - Y = (2.0 + 1.0) x 6.0 / 1.5 – 3.0
- However, the variable of each function is double type.
- The function should be defined after the main function and a prototype declaration should be used.

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

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

```
y = add_func(multi_func(2.0, 3.0), 4.0)
```

Exercise 5-1 : Tips

Function calls and assigned variables

Ex) : $y = \underbrace{\underbrace{2.0 \times 3.0}_{\text{①}} + 4.0}_{\text{②}}$

(solution 1)

```
   double p, q  ;
①  q = multi_func(2.0, 3.0);
②  p = add_func(q, 4.0);
```

The way writing in a sample program style
(Prepare different variables each time you assign)

Better

(solution 2)

```
   double p;
①  p = multi_func(2.0, 3.0);
②  p = add_func(p, 4.0);
```

Next value　　Original value

One variable to assign is enough
(There is a context for assignment operators)

Best

(solution3)

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

$\text{p} = \underbrace{\text{add\_func(} \underbrace{\text{multi\_func(2.0, 3.0)}}_{\text{①}}\text{, 4.0)}}_{\text{②}}$

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

- Create a program lcm.c that displays the least common multiple of two integers input from the keyboard.
- However,
  - 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 = 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 (exit (1)) in main function.
  - Implement 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;
}
```

- Use of exit function
  - Note that it is necessary to add
    `#Include <stdlib.h>`
    At the beginning to use the exit function

Recursive call of function

- Regular functions

```
f(){

    g();  Other functions
     …
    h();  Other functions
     …
}
```

It's not strange to call another function in a function definition

- 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

Recursive use of function

$${}_nC_r = \frac{\boxed{n!}}{\boxed{r!}\boxed{(n-r)!}}$$

```
/* combination.c : nCr */

#include <stdio.h>
#include <stdlib.h>
long factorial(int x)
{
    int     i;
    long    f;

    i = 0; f = 1;
    while (i < x) {
        ++i; f = f * i;
    }
    printf("factorial called, %3d!=%12ld\n", x, f);
    return(f);
}

main()
{
    int     n, r;
    long    ncr;

    if (scanf("%d %d", &n, &r) != 2) {
        printf("Input 2 integers\n"); exit(1);
    }
    if (n > 12) {
        printf("Too large n=%d > 12\n", n); exit(1);
    }
    if (n < r) {
        printf("Invalid inputs: n=%d < r=%d\n", n, r); exit(1);
    }
    ncr = factorial(n) / (factorial(r) * factorial(n - r));
    printf("n=%d, r=%d, nCr=%ld\n", n, r, ncr);
}
```

Functionize the calculation of factorial() that appears repeatedly (= factorial)

Error handling: Meaning of exit (1)

You can use the exit () statement to exit at any point in the program

- exit (1) ends abnormally
- exit (0) ends normally (same as exiting with return 0 at the end of the main statement).

※Note
Note that you need to add
#Include <stdlib.h>
in the head part of program to use the exit function

Ex: Factorial calculation (recursive call version) r_factorial.c

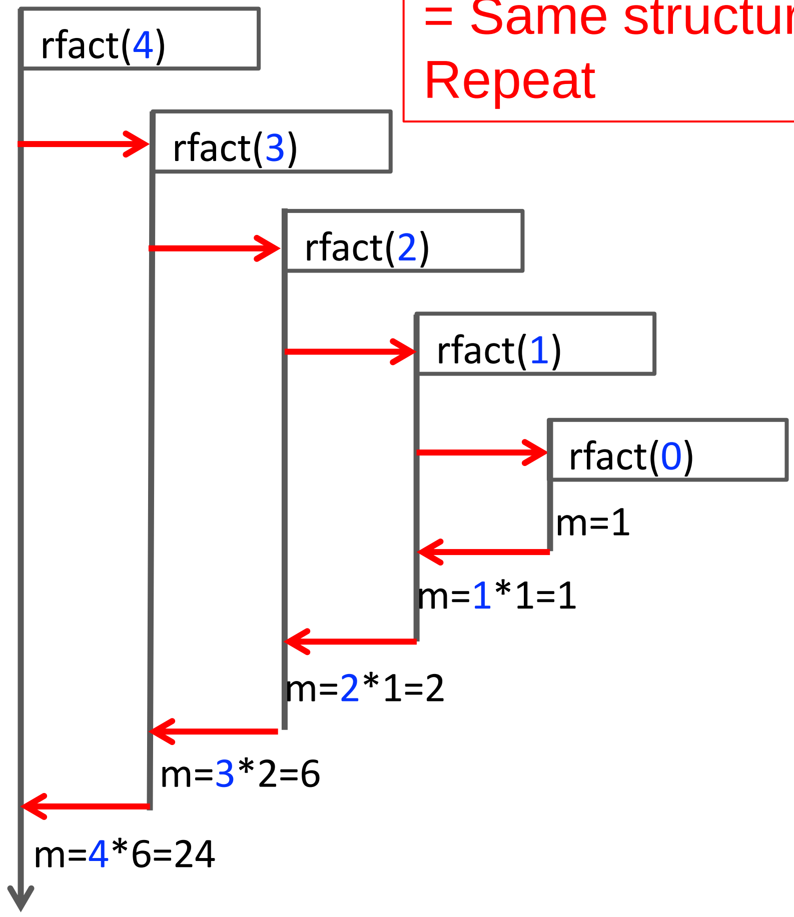
- Sample code: r_factorial.c

```
#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);

    /* fatorial by func. rfact()*/

    ans = rfact(num);

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

Calling the same function in the function definition

ex：case of num = 4

Recursive call
= Same structure
Repeat



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

- 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 funtion 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 ← b and b ← r

Greatest_commen_divisor＝ rgcd(b,r)

Effective range (scope) of variables

```
#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;
}
```

Two variables in the program on the left

`total_number`   `answer`

What is the effective range (= range treated as the same variable)?

`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 use the same name..

Variable sharing between functions using global variables

```
#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.

Will be explained in detail in 11th lecture

Variable name used in the function = local variable

```
#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;
}
```

Duplicate with the variable ans of the main function?

Summary

- 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?