# C Programming & Lab

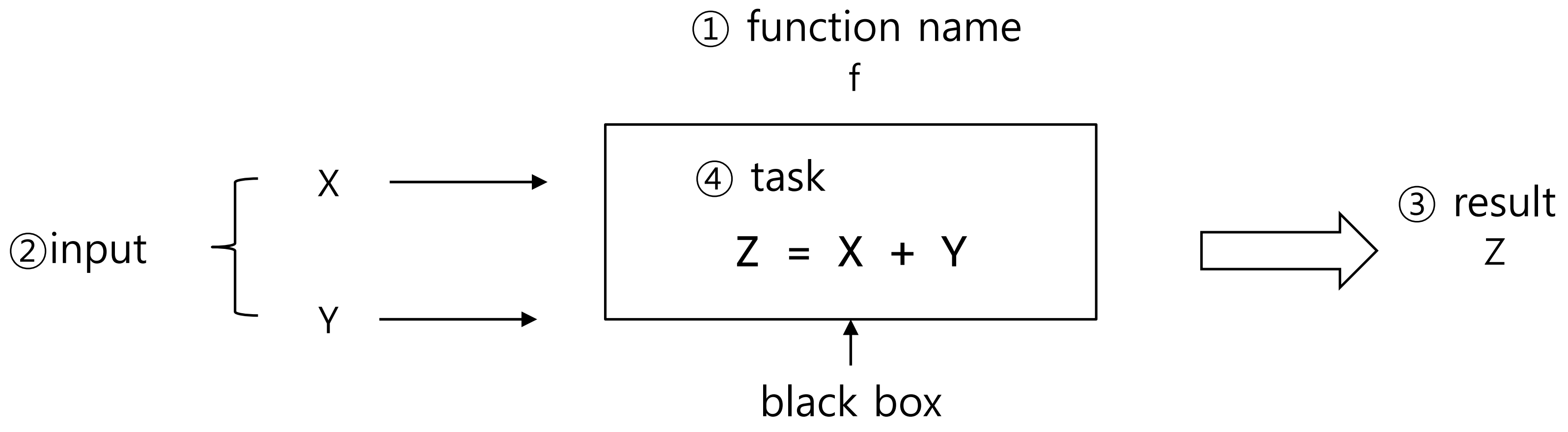
## 7. Functions

Sejong University

## Outline

## 1) Functions?

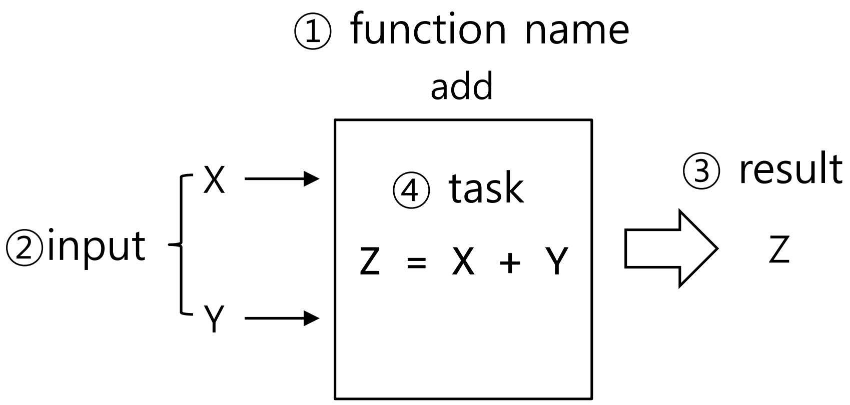
- **Functions in C Language**
  - A group of statements that perform a task
  - Example : printf(), scanf(), main()
  - Similar to a function in Mathematics
- **Function in Mathematics**
  - f(x,y) = x + y : compute the sum of two numbers
  - (black box) f: input x and y, return sum of two

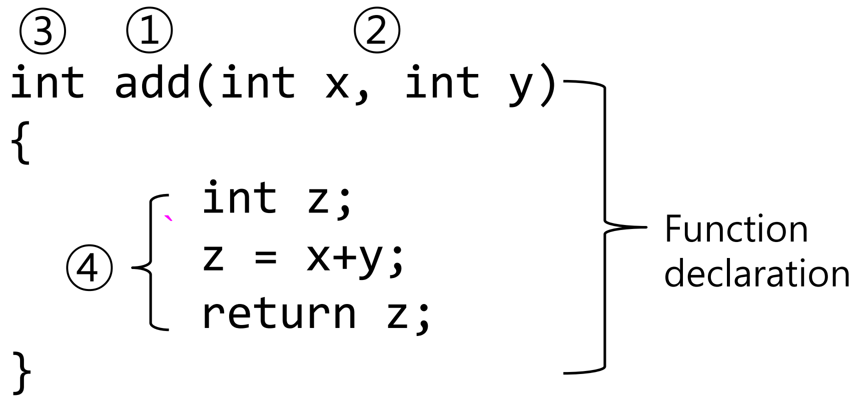


- **A function comprises 4 elements :** ①, ②, ③, ④

## 1) Functions?

- **Represent f(x,y) in C Language?**
  - Use 'add' for a function name f





```
int add(int x, int y)
{
    int z;
    z = x+y;
    return z;
}
```

- **A function comprises 4 elements :** ①, ②, ③, ④

## 1) Functions?

- **A function consists of function definition and function call**
  - **Definition** : Implement a task (i.e., inside of a black box)
    - 4 elements of a function
  - **Call** : Perform the task (Use the black box)
    - Need to know the name, inputs, outputs (data type) of the function
    - Do not care the internal implementation of the function

```c
int add(int x, int y)
{
    int z;
    z = x+y;

    return z;
}
```

Function definition

```c
int main()
{
    int c;
    c = add(3,4);
    printf("3 + 4 = %d\n", c);

    return 0;
}
```

Function call

## 2) Function Definition

- **Function**
  - Standard library functions are included with C compilers
  - User-defined functions: need to specify the internal code

```
int add (int x, int y)
{
        int z;
        z = x+y;

        return z;
}
```

- 4 elements
  - Name
  - Inputs
  - Task
  - Ouputs

## 2) Function Definition

- **Function Definition**

```
③   ①         ②
int add (int x, int y)
{
            int z;
    ④       z = x+y;

            return z;
}
```

① **Function name**

- Same rules with defining a variable name

- Likely to show the task of the function

② **Function argument or parameter**

- Inputs to perform the task of the function

- Multiple arguments: use comma (,) to separate them

※ No input: still need a parenthesis

## 2) Function Definition

- **Function Definition**

```
③   ①        ②
int add (int x, int y)
{
        int z;
   ④    z = x+y;

        return z;
}
```

③ **Return type**

- Function needs to return the result

- Need to specify the data type of the result

- Return type comes before the function name

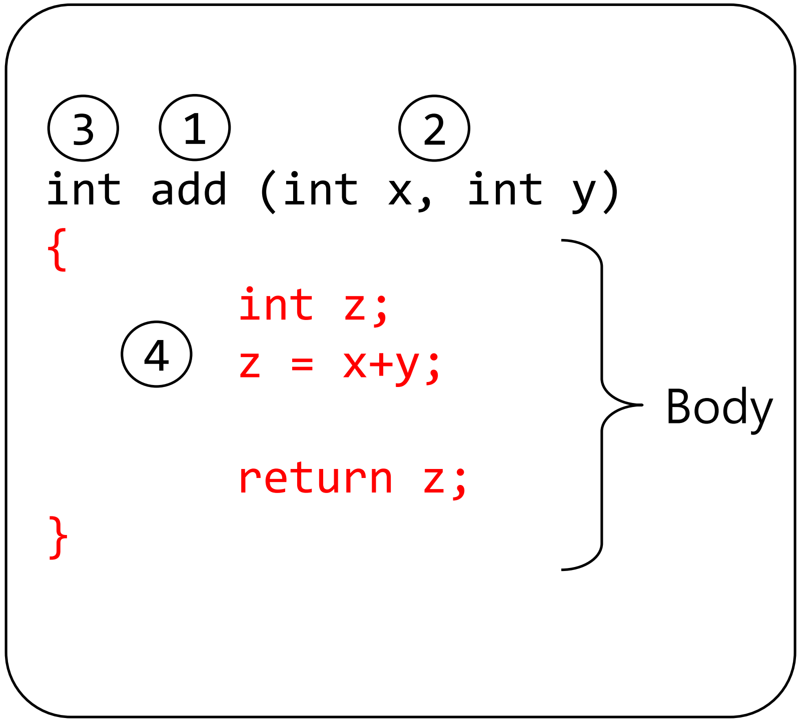
- No return: write “void”

※ Omit return type:
assume return type to be int type

※ ①,②,③: function header

# 2) Function Definition

## ▪ Function Definition



```
int add (int x, int y)
{
        int z;
        z = x+y;

        return z;
}
```

**④ Body**

- Specify the task of a function within a curly bracket { }

- Execute the code sequentially

- Use 'return' to return the result

**※return statement**

- Can place anywhere in the body
- Terminate the function when it meets return statement
- When return type is void, there is no return, i.e., do not need return statement

## 2) Function Definition

- **Function Definition**

```
Return-type Function-name (argument1, argument2, …)
{

       declarations
       statements

       return statement;
}
```

Number of arguments can vary

## 2) Function Definition

- **Example1**

```
char next_char(char c, int num)
{
    char c1;

    . . .

    return c1;
}
```

Example 1

1. return type : char
2. function name : next_char
3. function argument : char type c, int type num

※ Since return type is char type, no return statement causes an compilation error!!

- **Example2**

```
void print_heading( void )
{

   printf("\n==========\n");
   printf("    heading   ");
   printf("\n==========\n");

}
```

Example 2

1. return type : void
2. function name : print_heading
3. function argument : none

※ Since return type is void, do not need return statement

※ if omit return type void, assumed to be int type

※ if no argument, write void or nothing

## 2) Function Definition

- **[Practice1] Write a function.**
  - max function
    - ✓ Function name : max
    - ✓ Function arguments : int type variables *a* and *b*
    - ✓ Return type : int type
    - ✓ Return the maximum of *a* and *b*

- **[Practice2] Write a function.**
  - print_characters function
    - ✓ Function name : print_characters
    - ✓ Function arguments : char type variable *c*, int type variable *n*
    - ✓ Return type : void
    - ✓ Print the character stored in *c* n times in one line

## 2) Function Definition

- **[Practice3] Write a function.**
  - divide function
    - ✓ Function name : divide
    - ✓ Function arguments : int type variables *a* and *b*
    - ✓ Return type: double
    - ✓ Divide a by b and return the result as a real number
    - ✓ Ex) 3/2 -> return 1.5

- **[Practice4] Write a function.**
  - add3 function
    - ✓ Function name : add3
    - ✓ Function arguments : float type variables *a, b, c*
    - ✓ Return type : float
    - ✓ Return sum of *a, b, c*

## 2) Function Definition

- **[Practice5] Write a function.**
  - atoA function
    - ✓ Function name : atoA
    - ✓ Return type : char
    - ✓ Function arguments : char type variable *ch*
    - ✓ Convert a lower-case letter *ch* to an upper-case letter and return the letter

## 3) Function Calls and Returns

- **How to call a function**
  - Write a function name, enter function arguments within a parenthesis
  - add(3, 4): call a function ‘add’, first argument is 3, second argument is 4

```
int add(int x, int y)
{
    int z;

    z = x+y;

    return z;
}
```

```
int main()
{
    int c;

    c = add(3, 4);
    …
    return 0;
}
```

Function call

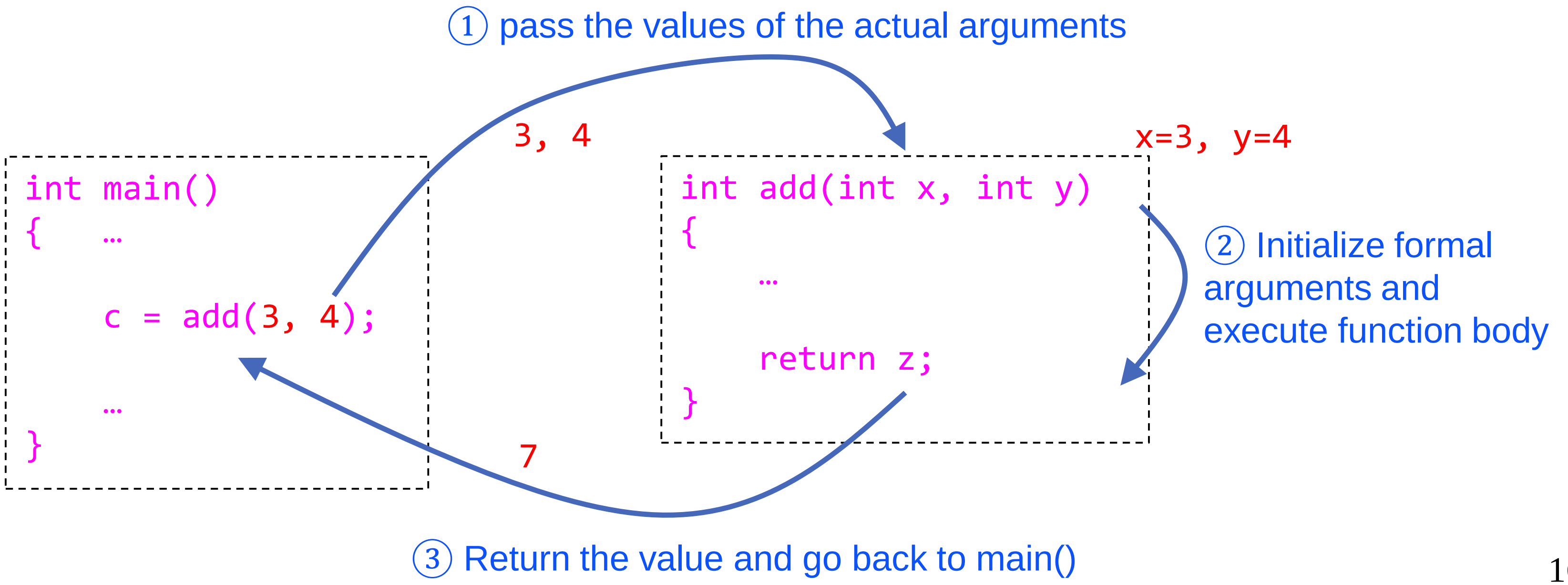
add (3, 4)

Function name

arguments

# 3) Function Calls and Returns

- **Function Calls**
  ① In main(), call a function add(3,4) -> arguments 3 and 4 are passed to a function 'add', execute statements in a function 'add'
  ② In add(), set arguments x=3, y=4, execute statements
    - Formal argument:  x and y in function definition, add()
    - Actual argument : values (3 and 4) passed to the function add()
  ③ Terminating the function add(), go back to main() and execute the next line


① pass the values of the actual arguments
3, 4
x=3, y=4
int main()
{ …
c = add(3, 4);
…
}
int add(int x, int y)
{
…
return z;
}
② Initialize formal arguments and execute function body
7
③ Return the value and go back to main()

## 3) Function Calls and Returns

- **Mechanism of function call**
  ① Pass the values of actual arguments
  - If a function A calls a function B, the actual arguments are passed to the function B and execute the body of the function B

  ② Initialize formal arguments and execute the function body
  - Initialize formal arguments with the actual arguments
  - Execute the function B body
  - If encounter a return statement or end of the function, terminate the function B
  - If there is a value in a return statement, pass the value to the function A

  ③ Return the value
  - If terminate the function B, Back to the function A
  - The returned value from the function B is the result of the function B

## 3) Function Calls and Returns

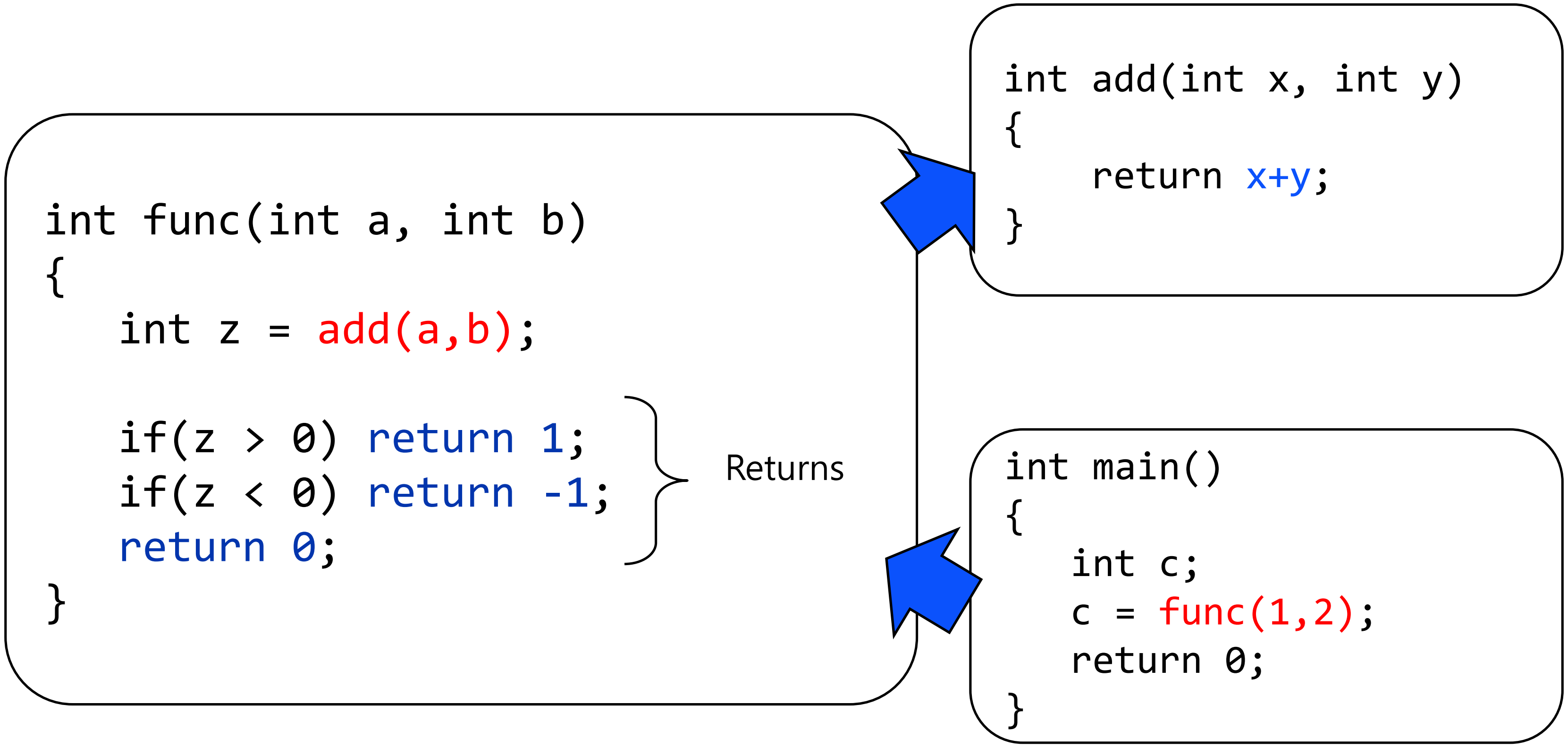
- **Actual arguments**

```
int add(int x, int y)
{
    …
    return z;
}
```

```
int main()
{
    int a=4, b=3;
    int v1,v2,v3,v4,sum;

    v1 = add(a, a+b);
    v2 = add(1, a+2);
    v3 = add(1+2, a) - 3;
    sum = add(1, b)+add(a, 2);
    v4 = add(a, add(1, 2));

    return 0;
}
```

## 3) Function Calls and Returns

- **Calls and Returns**
  - main( ) → func( ) → add( )



```
int add(int x, int y)
{
    return x+y;
}
```

```
int func(int a, int b)
{
   int z = add(a,b);

   if(z > 0) return 1;
   if(z < 0) return -1;
   return 0;
}
```

```
int main()
{
   int c;
   c = func(1,2);
   return 0;
}
```

## 3) Function Calls and Returns

- **[Practice6] Using functions in pages 12~14, write a program.**
  - Print a character 'a' once, character 'b' twice, …, character 'z' 26times.
    - ✓ Print one types of characters in a line
    - ✓ Use a function print_characters

Display

```
a
bb
ccc
dddd
…
```

## 3) Function Calls and Returns

- **[Practice7] Using functions in pages 12~14, write a program.**
  - Read 4 integers a, b, c, d, print the maximum value.
    ① Use a function max to find the larger value of a and b
    ② Use a function max to find the larger value of c and d
    ③ Use a function max to find the larger value of ① and ②

Input Example 1

```
5 2 9 4
```

Output Example 1

```
9
```

✓ (Advance) use function call max as an argument. Use one statement to find the maximum value.

## 3) Function Calls and Returns

- **Function Definition/Declaration**
  - Compile the following two codes?
    - Order of function definitions is different

```
int add (int x, int y)
{
    . . .
}

int main()
{
    . . .
    c = add(3,4);
    . . .
}
```

Succeeded

```
int main()
{
    . . .
    c = add(3,4);
    . . .
}

int add (int x, int y)
{
    . . .
}
```

Error
add() has not been defined

Compilation error or warning

## 3) Function Calls and Returns

- **Function Definition/Declaration**

```
int main()
{
    . . .
    c = add(3,4);
    . . .
}

int add (int x, int y)
{
    . . .
}
```

Error
add() has not
been defined

- Compilation error or warning
- In C, should define a function before calling the function
- Using multiple functions, should we define the functions in the same order with the function calls?
- How to handle this?

# 3) Function Calls and Returns

- **Function Definition/Declaration**

```
int add (int x, int y);←    Function
                             Declaration
int main(){
   . . .
   c = add(3,4); ← OK!
   . . .
}

int add (int x, int y){ ← Function
   . . .                   Definition
}
```

- Declare a function
- The function is defined elsewhere
- That is, a function add() with two arguments of int type, return type of int type is defined elsewhere

## 3) Function Calls and Returns

- **Function Definition/Declaration**
  - Return type, function name, arguments, semi-colon (;)

Function header

```
Return-type function-name ( argument1, argument2,…) ;
```

  - Omit variable names
  - OK to use different variable names from the variable names used in the function definition

```
int add(int x, int y);
int add(int a, int b);
int add(int, int);

- Equivalent!
```

## 3) Function Calls and Returns

- **[Practice8] using functions in the slides 12~14, write a program.** **(function declaration)**
  - Read 10 alphabet letters, convert them to upper-case letters.
    - ✓ Call atoA() to convert a lower-case letter to a upper-case letter

Input Example 1

```
HellOWorLd
```

Output Example 1

```
HELLOWORLD
```

## 3) Function Calls and Returns

- **[Practice9] using functions in the slides 12~14, write a program. (function declaration)**
  - Read 6 integers a, b, c, d, e, f, print the result of the following expression.

    a/b + c/d + e/f (real number operation)

    - ✓ Division: divide function
    - ✓ Addition: add3 function

Input Example 1

```
5 2 9 4 6 4
```

Output Example 1

```
6.25
```

- ✓ (Advance) use divide function call as an argument of add3 function. Use one statement.

## 4) Local Variable and Global Variable

- **Scope of a variable**
  - Variables declared in a function only used inside the function
  - Sometimes, need a variable that can be used anywhere in the program
- **Types**
  - Local variable
  - Global variable

## 4) Local Variable and Global Variable

- **Local variable**
  - ✓ Declared in a function
  - ✓ Used inside the function
  - ✓ When a function is called, it is automatically created. When a function is terminated, it is automatically removed – automatic variable
  - ✓ Variables x and y in add() are local variables

- Compilation error?

Error →

```
int add (int x, int y)
{
   int z;
   c = x + y;

   return c;
}
```

```
int main()
{
   int c;
   c = add(3,4);
   . . .
}
```

## 4) Local Variable and Global Variable

- **Result of the following code?**

```
int add (int x, int y)
{
   int c;

   c = x + y;

   return c;
}
```

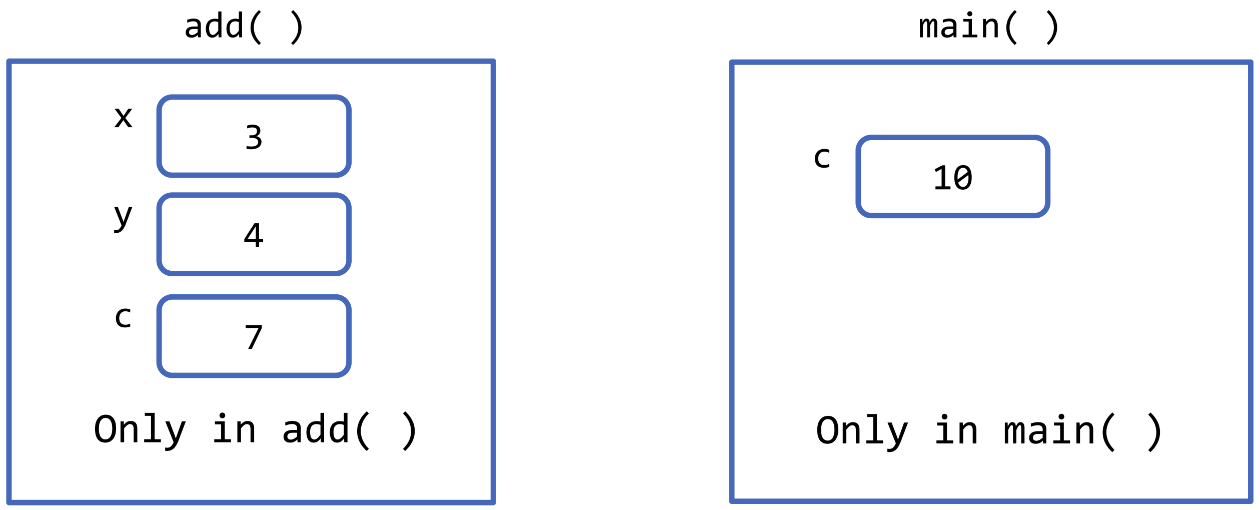
```
int main()
{
   int c = 10;

   printf("3 + 4 = %d\n", add(3,4));
   printf("c = %d\n", c);
   return 0;
}
```

```
[Result]
3 + 4 = 7
c = 10
```

- Local variable!

## 4) Local Variable and Global Variable

- 4 local variables
- A variable c in main() and a variable c in add() are not the same
- Change the value of the variable c in add() does not affect the variable c in main()



- Do not need to consider what variables are used by other functions

## 4) Local Variable and Global Variable

- **Global variable**
  - ✓ Can be used anywhere in the program
  - ✓ Declared outside a function
  - ✓ Initialized with 0
    - ✓ Better to explicitly initialize any variables.

```
int c; ←—— Global variable

int add (int x, int y)
{
   c = x + y;

   return c;
}
```

## 4) Local Variable and Global Variable

- **Result of the following code?**

```
int c;  ← Global variable

int add (int x, int y)
{
   c = x + y;

   return c;
}
```

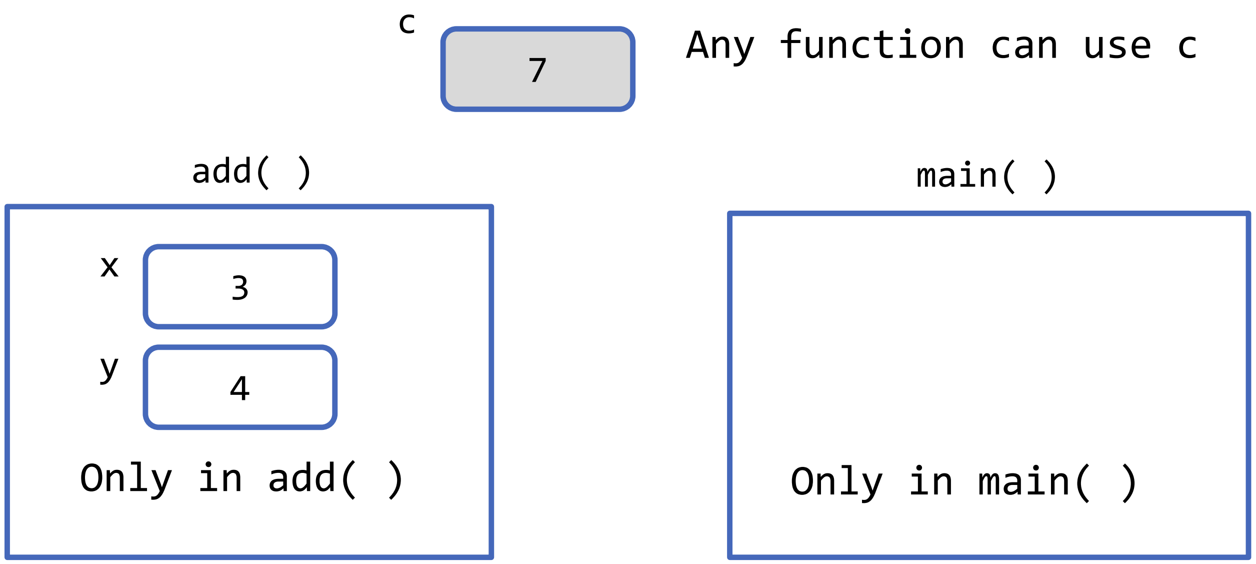
```
int main()
{
   c = 10;

   printf("3 + 4 = %d\n", add(3,4));
   printf("c = %d\n", c);

   return 0;
}
```

```
[Result]
3 + 4 = 7
c = 7
```

- Global variable!

## 4) Local Variable and Global Variable

- 2 local variables, 1 global variable
- c is a global variable
- c can be used anywhere in the program
- c in add() is the same with c in main()



# 4) Local Variable and Global Variable

- **Compilation Error?**

```
int c;          ← Global variable

void add (int x, int y)
{
   c = x + y;

   printf("add: c = %d\n", c);
}
```

```
int main()
{
   int c = 10;

   add(3,4);
   printf("main: c = %d\n", c);

   return 0;
}
```

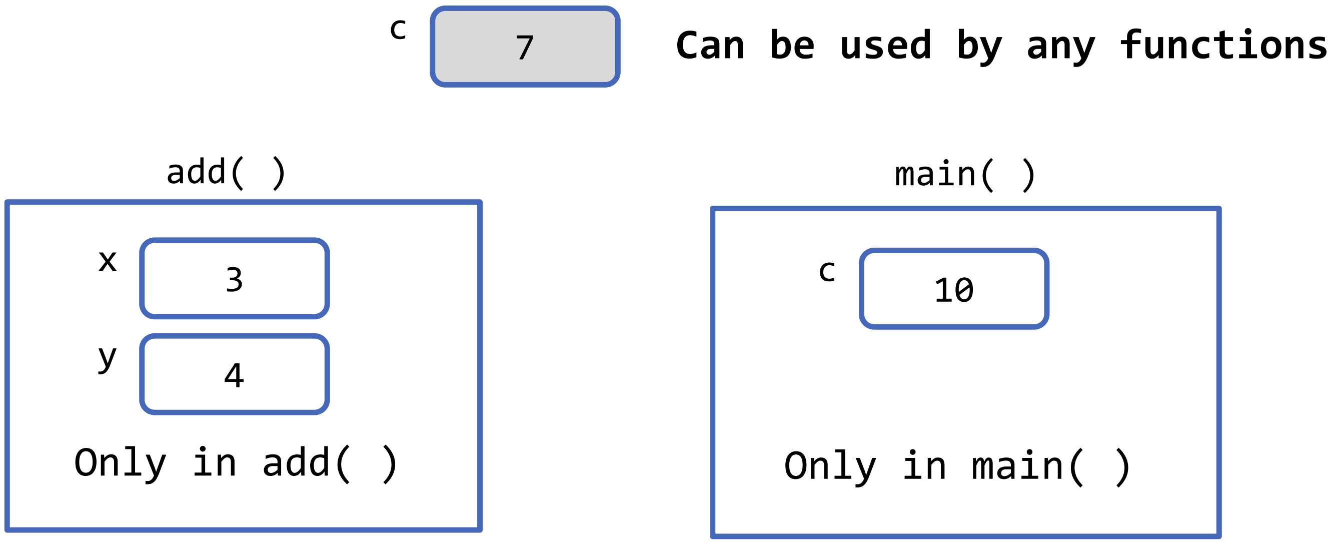
```
[Result]
add: c = 7
main: c = 10
```

- A global variable and a local variable can have the same name: no compilation error

- Result?

## 4) Local Variable and Global Variable

- If a local variable and a global variable are declared using the same name, the local variable will be given priority
- In add(), c is a global variable. In main(), c is a local variable
- In add(3,4), assigning 3+4 to c does not affect c in main()



## 4) Local Variable and Global Variable

- **Role of a global variable**
  - Use a global variable c, instead of return statement, to pass the result of the operation in add() to main()

```
int c;        ← Global variable

void add (int x, int y)
{
   c = x + y;
}
```

```
int main()
{
   add(3,4)
   printf("3 + 4 = %d\n", c);

   return 0;
}
```

```
[Result]
3 + 4 = 7
c = 7
```

## 4) Local Variable and Global Variable

- **[Practice10] Define the following functions.**
  - div( ) function
    - ✓ Return type: int type, arguments: two int types
    - ✓ Divide one by the other, return the quotient, and store the remainder in a global variable
  - main( ) function
    - ✓ Read two integers, call div(), print the quotient and remainder

Input Example 1

```
5 3
```

Output Example 1

```
1 2
```

Input Example 2

```
4 2
```

Output Example 2

```
2 0
```

## 4) Local Variable and Global Variable

- **[Practice11] Define the following functions.**
  - swap( ) function
    - ✓ Return type: void type, No argument
    - ✓ Global variables a and b, swap values of the two variables
  - main() function
    - ✓ Read two integers and store them in global variables a and b
    - ✓ Call swap( )
    - ✓ Print the swapped two variables

Example
(Red: use input)

```
a : 6
b : 8
Call swap
a = 8
b = 6
```

## 4) Local Variable and Global Variable

- **Caution: global variable**
  - Convenient to use, but functions are not independent any more
  - Careful with using global variables, especially write a huge program
  - When call a function, need to consider whether other functions use global variables or not
- **Local variable vs. Global variable**

| | Local Variable | Global Variable |
|---|---|---|
| Declaration | Inside of a function | Outside of a function |
| Scope | Within a function declaring the variable | Anywhere in a program |
| Automatic initialization | X (User needs to initialize) | O (automatically set to 0) |
| Termination | When a function terminates | When a program terminates |

## 5) Function and Library

- **Library**
  - Collection of functions
  - Call functions if needed
- **Standard Library**
  - Standard functions that are implemented in C: printf(), scanf()

- **Usage**
  - Need to know the form and role of the functions
  - Do not need to know the implementation details of the functions
  - Must declare the functions

  - Have we declared the functions such as printf(), scanf()?

## 5) Function and Library

- **Compile?**

```
~~#include <stdio.h>~~    // Remove this line

int main()
{
    printf("Hello, World!!\n");
    return 0;
}
```

✓ Compilation error: cannot find printf()

- #include statement includes 'stdio.h' in a source code
- printf() is defined in 'stdio.h'

- Call 'stdio.h' a header file ( extension: .h)

## 5) Function and Library

- **Standard function and header file**
  - ✓ Need to include header files to use standard functions by using #include statement
- **Frequently used header files and standard functions in C**
  - ✓ Refer "help" in C

| Header file | Role | Standard function |
|---|---|---|
| stdio.h | Input, Output, File | printf, scanf, putc, getc, fopen, etd |
| stdlib.h | Number conversion, Dynamic allocation | atoi, rand, srand, malloc, free, etd |
| ctype.h | Character | isalnum, isalpha, islower, toupper, etd |
| math.h | Mathematical operations | sin, asin, exp, log, pow, sqrt, abs, etd |
| time.h | Time | clock, time, difftime, etd |
| string.h | Character string, memory | strcpy, strcat, strcmp, strlen, memcpy, etd |