

Ch.8 Functions

What you will learn in this chapter



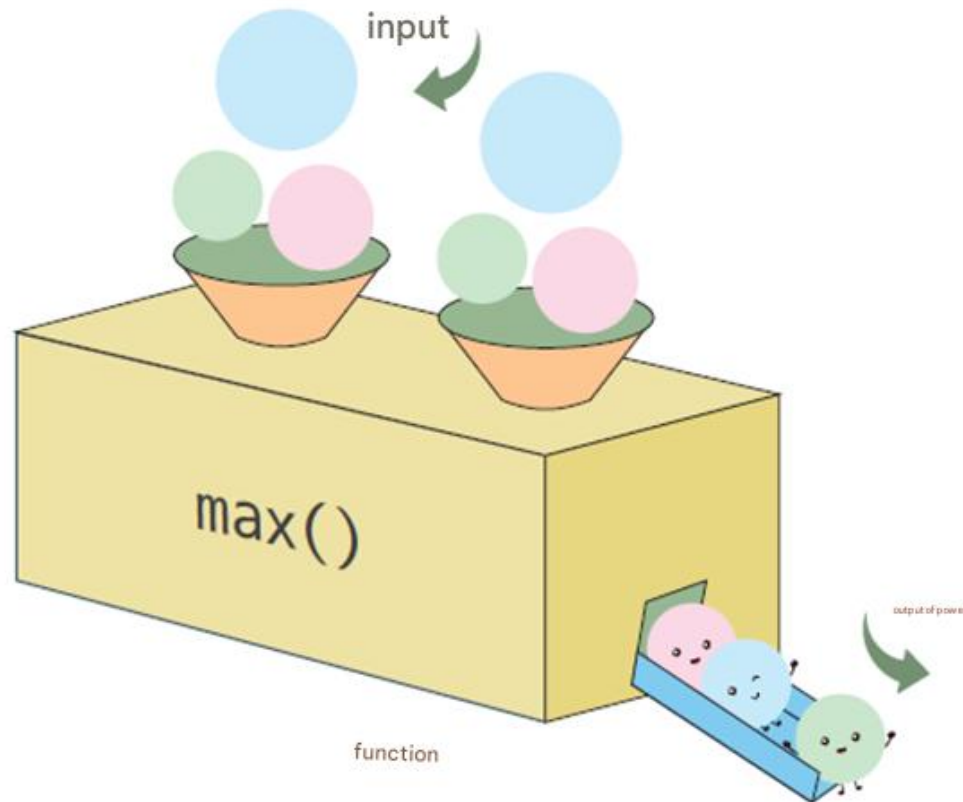
- Modularization
- Concept of function, role
- How to write a function
- Return value
- Transfer of acquisition
- Why use functions?

Large programs should be written by breaking down the overall problem into simpler, more understandable functions .



Concept of function

- **Function** : It is like a black box that takes input, performs a specific task, and returns a result.



The function Why is it needed?

- The same code is used in multiple places .

I have similar code,
Can I combine them into one?



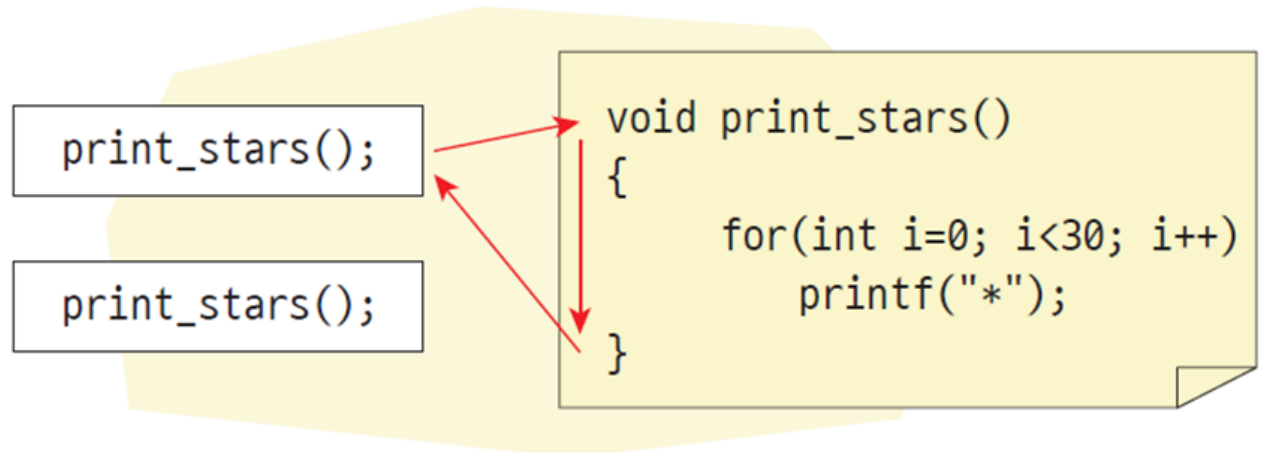
```
for(int i=0; i<30; i++)  
    printf("*");
```

```
for(int i=0; i<30; i++)  
    printf("*");
```

The function Why is it needed?

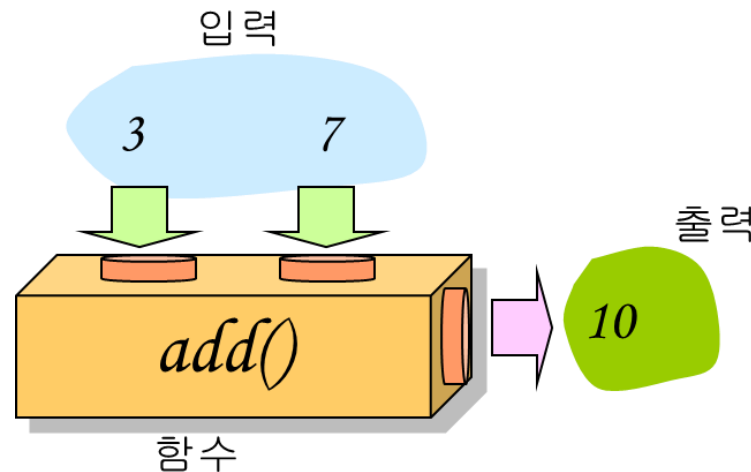
- By writing functions, you can create identical code in many place

Just use a function!



Features of functions

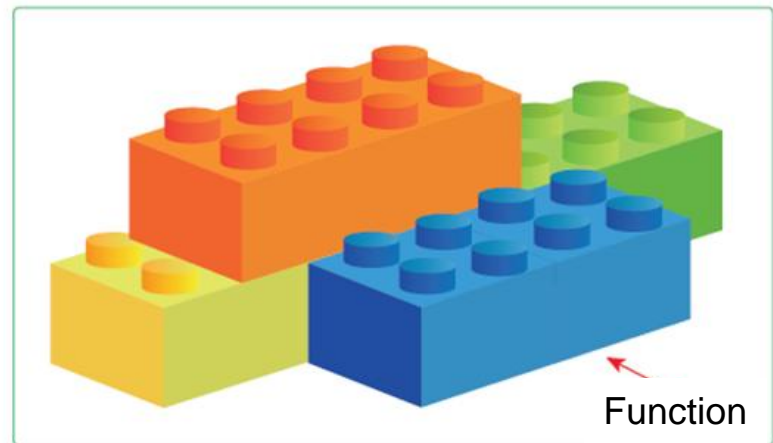
- A function is a collection of instructions to perform a specific task
- Functions have distinct names
- A function performs a specific task
- A function can take input and return a result



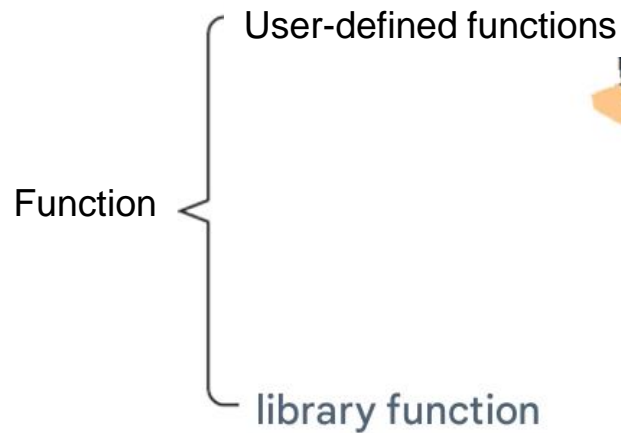
Pros of function

- Using functions can help **prevent duplication** of code .
- Once a function is written, it can be **reused** multiple times .
- Using functions allows you to divide your entire program into modules, making the **development process easier, more organized, and easier to maintain.**

Program



Types of functions



Since the function I want isn't available, I have to create it myself.



These functions are provided By default.

Definition of a function

Syntax

Function definition

Return type Function name Parameter (none)

`void print_stars()`

`{`

`for(int i=0; i<30; i++)
 printf("*");`

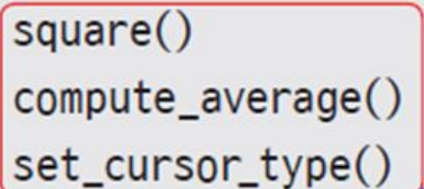
Function body

`}`

Definition of a function

- Return type
 - The return type refers to the type of data that a function returns to the caller after completing processing.
- Function name
 - A function name can be any name that follows the rules for identifiers .
 - It is a good idea to use (verb + noun) that implies the function's functionality.

```
int    square()  
double compute_average()  
void   set_cursor_type()
```



// Function to square an integer

// Function to calculate the average

// Function to set the type of cursor



Function name

Tip: Function's length

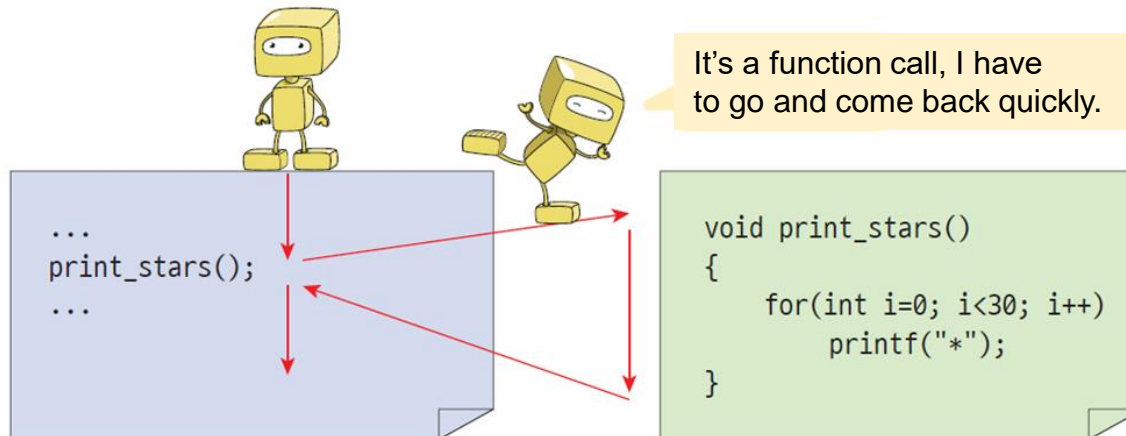


Is there a limit to the length of a function?

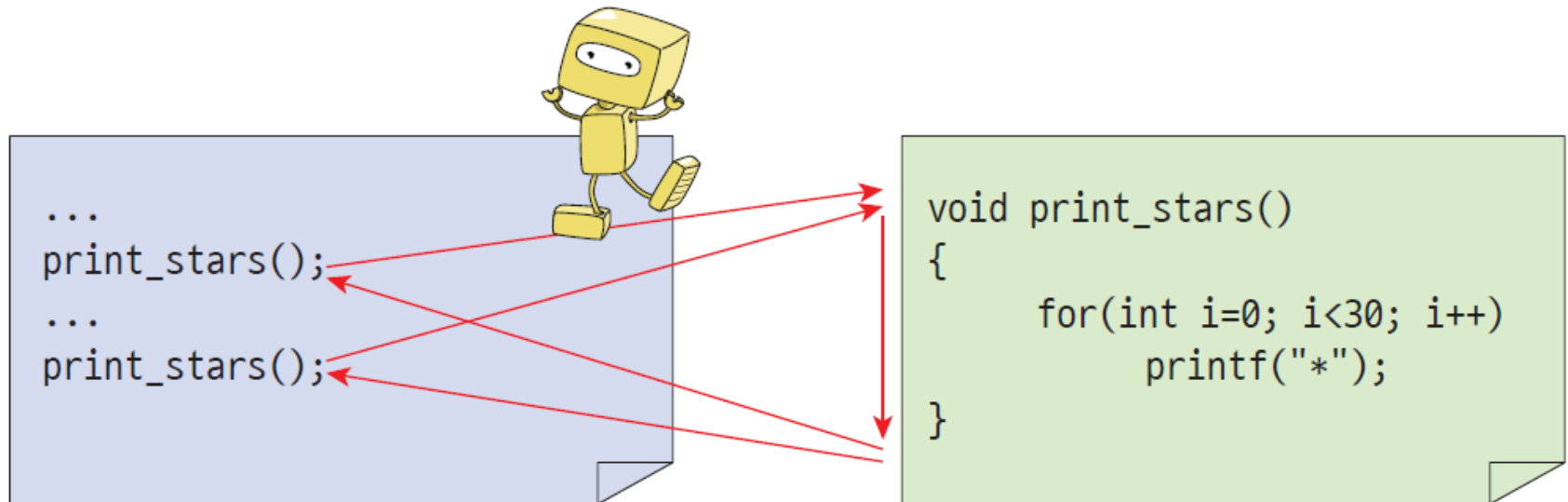
In C, there is no limit to the length of a function. This means that you can put many statements in a single function. However, **it is not good if the function length becomes too long**. Basically, **a function should perform only one task**. If the function length becomes too long, it should be considered as performing more than one task. Therefore, it is better to split the function. So, what is the appropriate length? Of course, there is no absolute standard, but it is good to keep it from exceeding 30 lines.

Function call

- A function call is writing the name of a function, such as `print_stars()` .
- The statements within a function are not executed at all until they are called. When a function is called, the currently executing code is suspended for a moment, and execution moves to the called function, where the statements within the function body are executed sequentially.
- When the execution of the called function is finished, it returns to the calling location and resumes execution of the code that was paused.

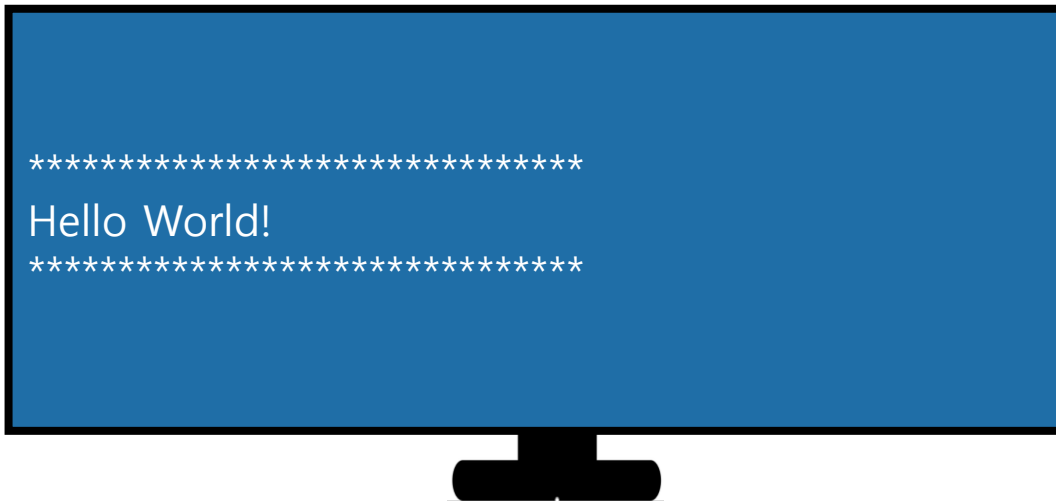


A function can be called multiple times.



Example

- the `print_stars ()` function twice to output the following :



Example

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

```
void print_stars()
```

```
{
```

```
    for( int i = 0; i < 30; i++)  
        printf("*");
```

```
}
```

```
int main( void )
```

```
{
```

```
    print_stars ();
```

```
    printf("\nHello World!\n");
```

```
    print_stars ();
```

```
    printf("\n");
```

```
    return 0;
```

```
}
```

Function call

Function call

Parameters and return values

Syntax

Structure of a function

Return type Function name Parameters

`int max(int x, int y)`

{

`if(x > y)`

`return x;`

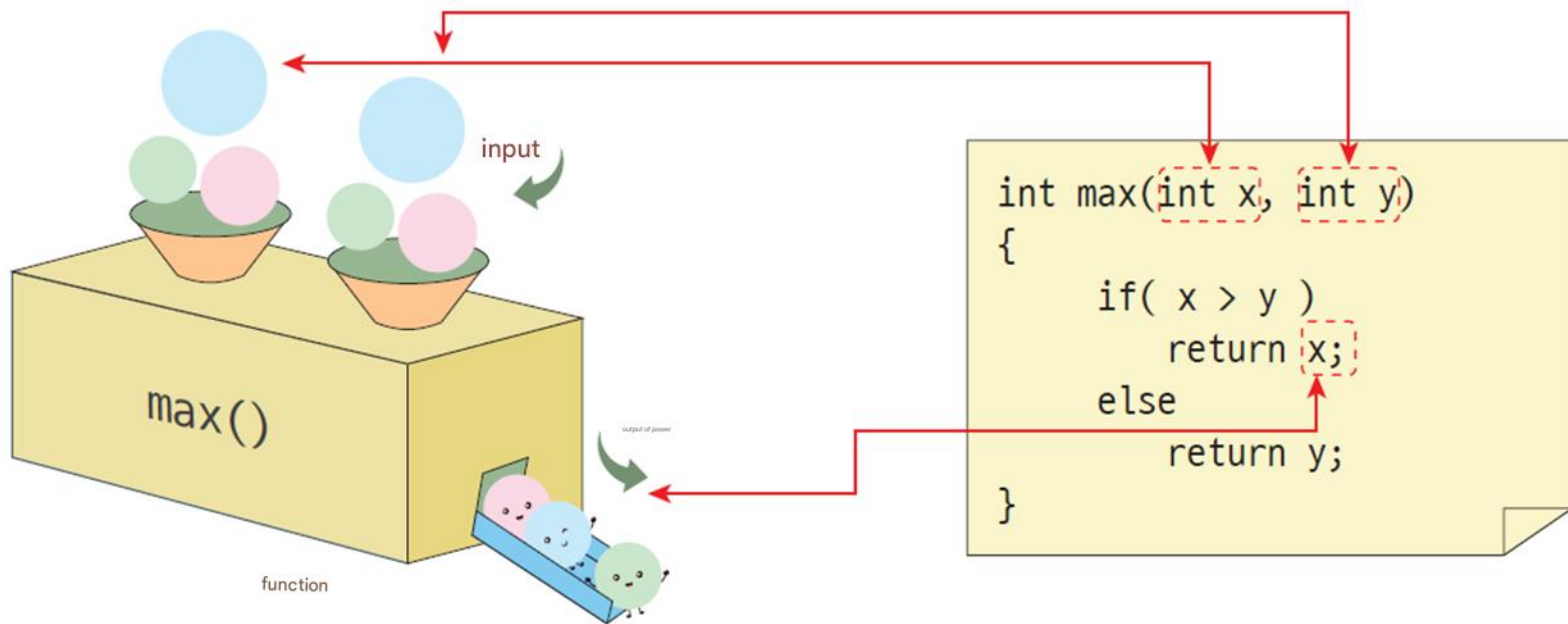
`else`

`return y;`

}

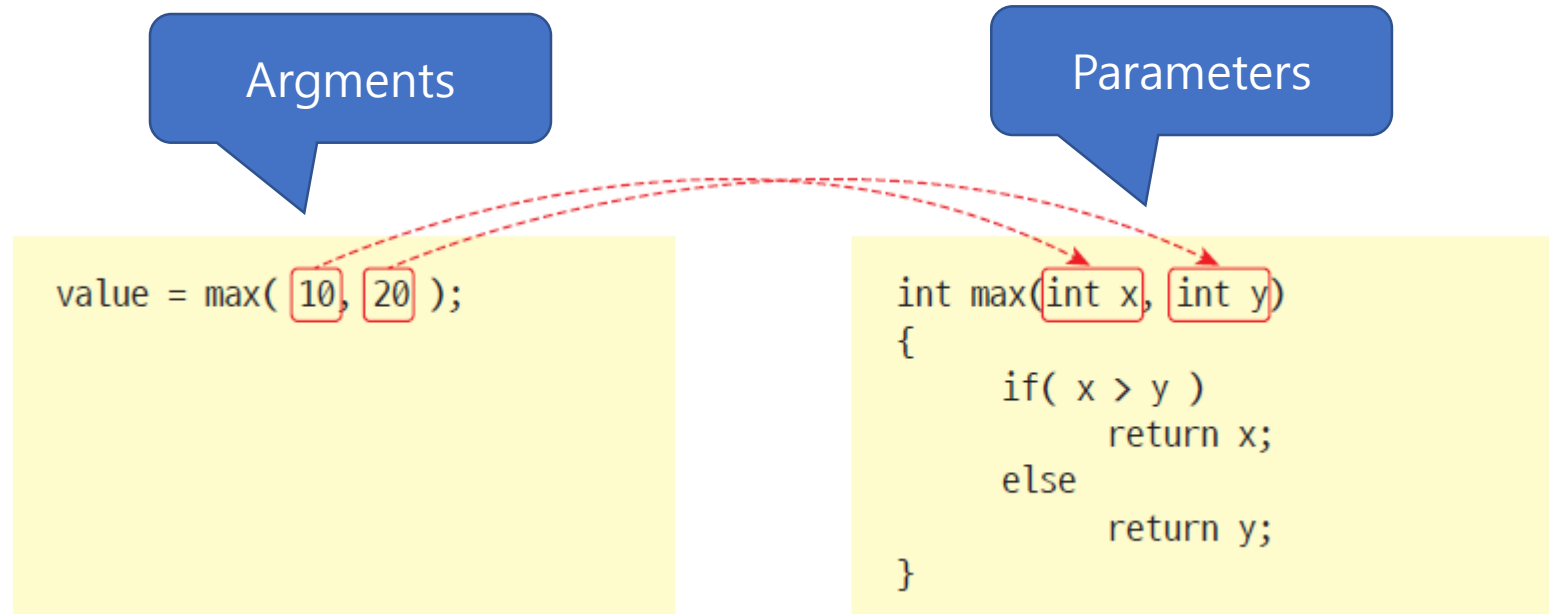
Function body

Parameters and return values



Arguments and parameters

- An argument is a value actually passed to a function by the calling program.
- A parameter is a variable that receives this value.



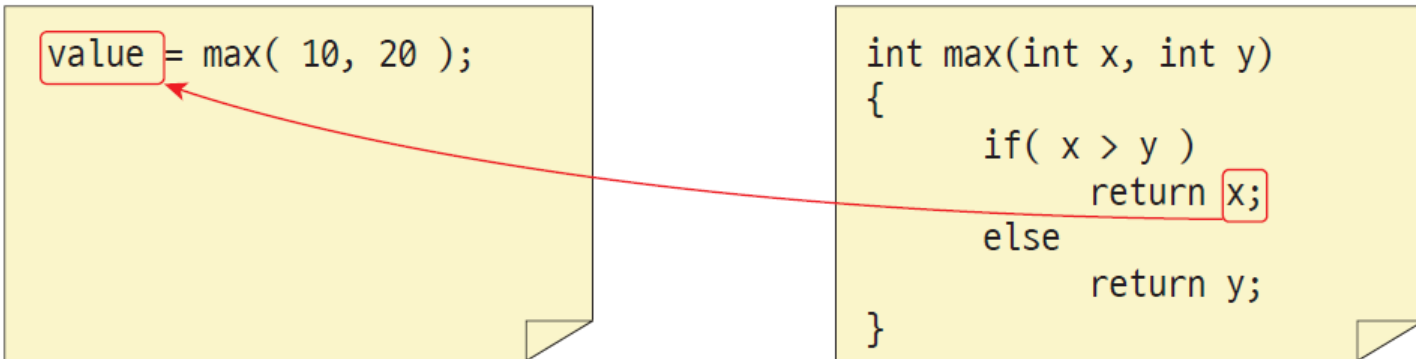
Arguments and parameters

- If there are no parameters, you can write `void` in the parameter position, like `print_stars(void)`, or write nothing, like `print_stars()`
- The arguments may vary each time the function is called. (...)
- The number of parameters must match exactly. If the number of parameters and the number of arguments do not match, an error will occur that is very difficult to find.

```
max(10);           // When calling the max() function, there must be two arguments.  
max( );           // When calling the max() function, there must be two arguments.
```

Return value

- The return value is the result of the operation that the function returns to the place where it called.
- To return a value, write a formula after the return statement and the value of the formula will be returned.
- There can be **multiple arguments, but only one return value.**
(possible to return multiple values with pointer-parameter or structure)



Example



Practice

- max() function written above to find the larger value among the values entered by the user.

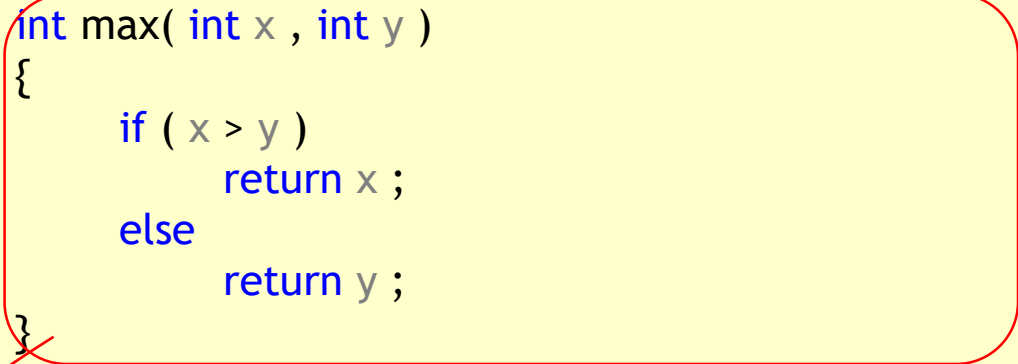


Enter two integers : 10 20
The larger value is 20 .

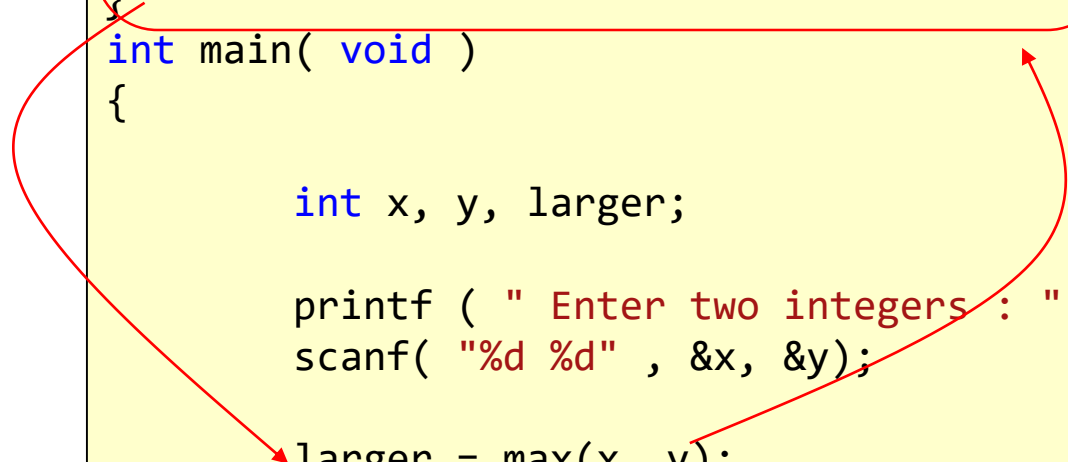
Example

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

```
int max( int x , int y )  
{  
    if ( x > y )  
        return x ;  
    else  
        return y ;  
}
```

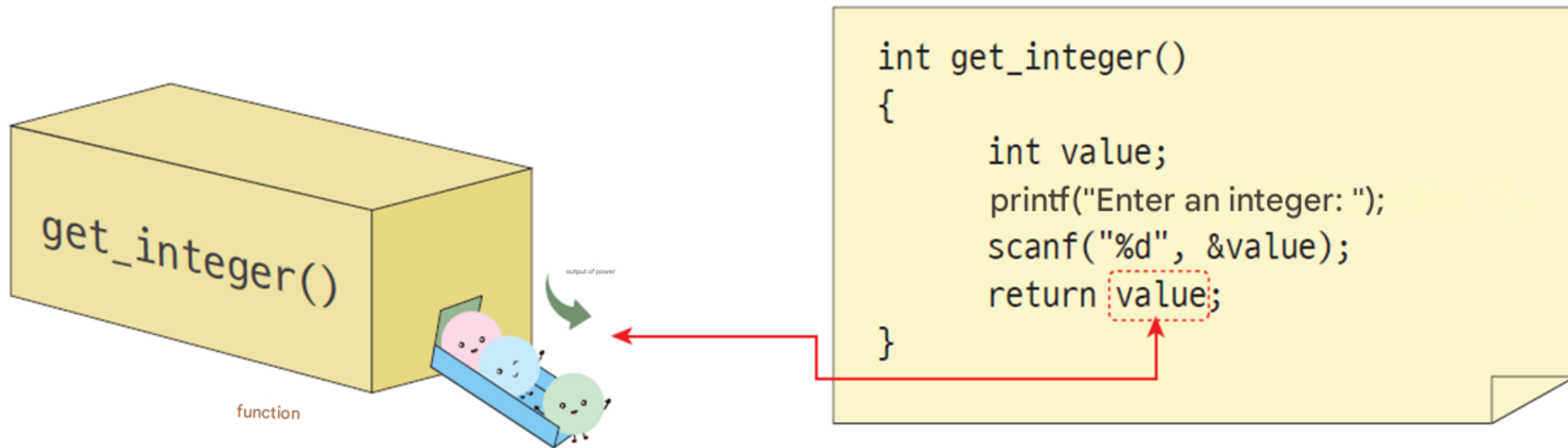


```
int main( void )  
{  
  
    int x, y, larger;  
  
    printf ( " Enter two integers : " ) ;  
    scanf( "%d %d" , &x, &y);  
  
    larger = max(x, y);  
    printf ( " The larger value is %d . \n" , larger);  
    return 0;  
}
```



Lab: Inputting an integer get_integer () function

- Let's write a function `get_integer ()` that prints an input guidance message and takes an integer as input and returns it to us.



get_integer () function

```
// Function that receives an integer from the user
#include <stdio.h>

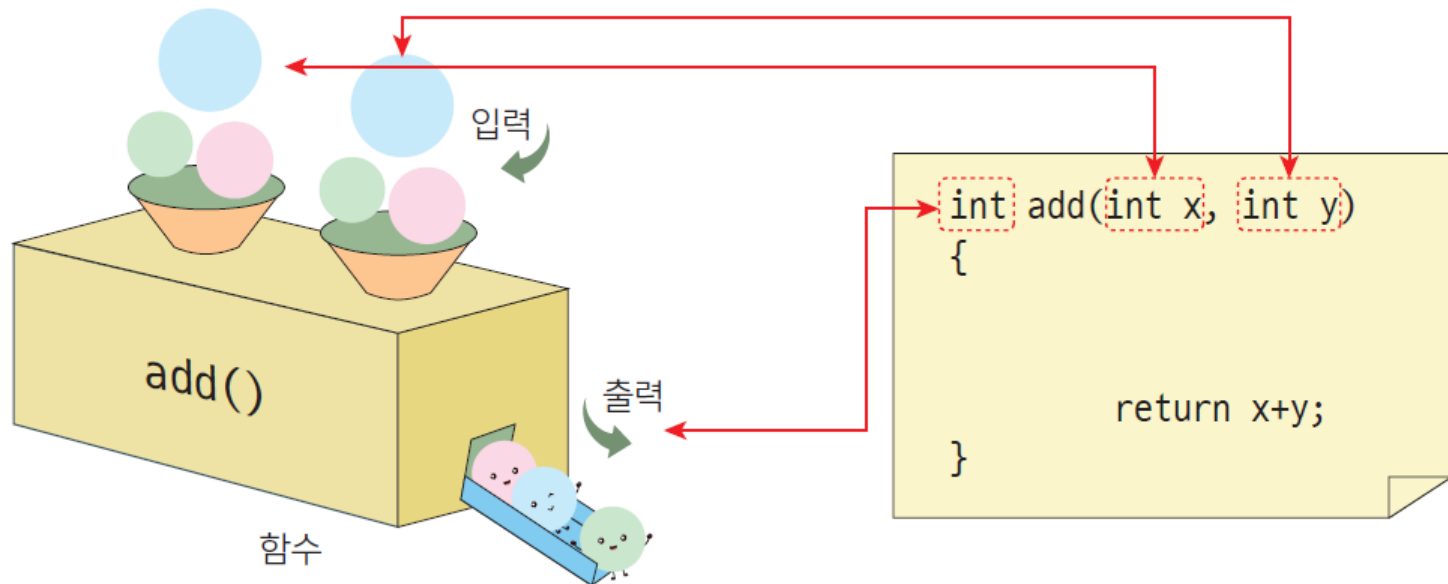
int get_integer ( void )
{
    int value;

    printf ( " Enter an integer : " );
    scanf ( "%d" , &value);

    return value;
}
```

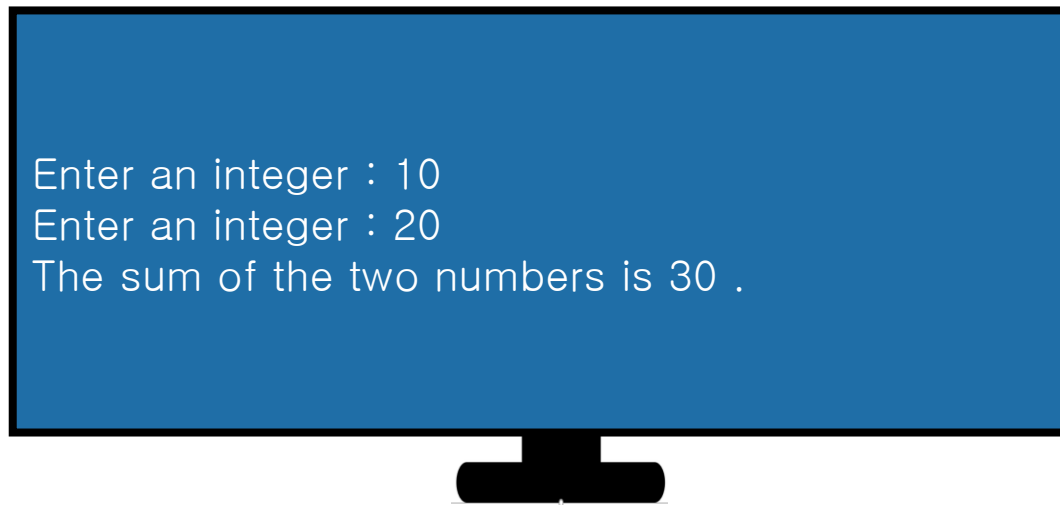

Lab: add() function to calculate the sum of integers

- Let's create a function that takes two integers and calculates their sum. We need to decide on a name for the function first.



Lab: Program to calculate the sum of integers

- Let's calculate and print the sum of the integers received from the user using the `get_integer ()` written above.



Practice

```
#include < stdio.h >
//
int get_integer ()
{
    int value;
    printf ( " Enter an integer : " );
    scanf ( "%d" , &value);
    return value;
}

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

int main( void )
{
    int x = get_integer ();
    int y = get_integer ();

    int sum = add(x, y);
    printf ( " The sum of the two numbers is %d . \n" , sum);
    return 0;
}
```

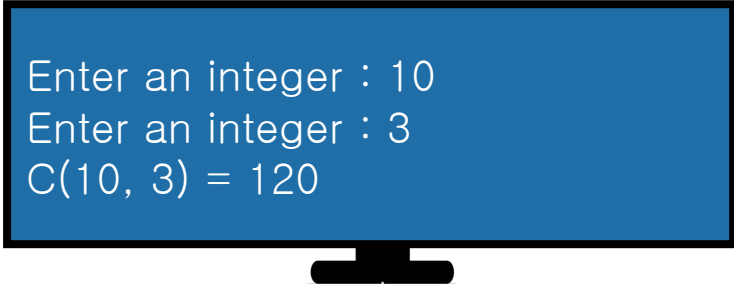
Combination calculation function

$$C(n,r) = \frac{r!}{(n-r)!r!}$$

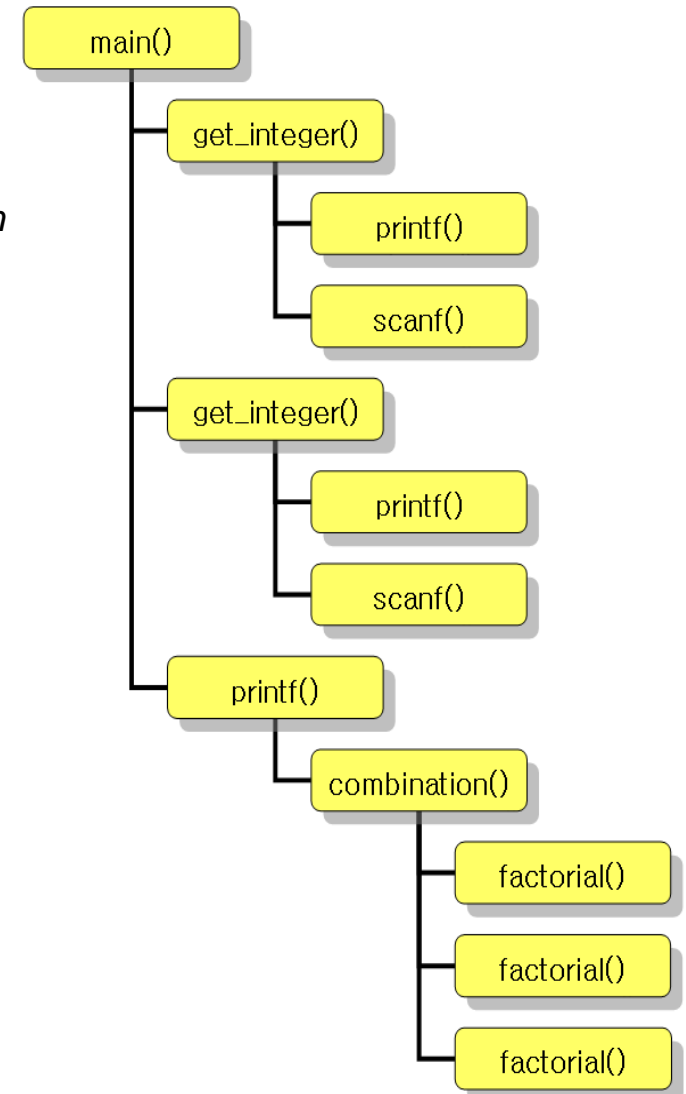
n = total number of items
 r = number of items chosen

combination is a way of selecting items from a group, where the order does not matter.

- Calculate the combination by calling the factorial calculation function and the get_integer () function.



```
Enter an integer : 10
Enter an integer : 3
C(10, 3) = 120
```



Example

```
// Example of finding mathematical combination values
#include < stdio.h >

// Returns the factorial value
int factorial( int n )
{
    int i , result = 1;

    for (i = 1; i <= n ; i++)
        result *= i ; // result = result * i
    return result;
}

// Calculate
int combination( int n, int r)
{
    return (factorial(n)/(factorial(r) * factorial(n-r)));
}
```

```
// Get a value from the user and return it
int get_integer ( void )
{
    int n;

    printf ( " Enter an integer : " );
    scanf ( "%d" , &n);
    return n;
}

int main( void )
{
    int a, b;
    a = get_integer ();
    b = get_integer ();

    printf ( "C(%d, %d) = %d \n" , a, b, combination(a, b));
    return 0;
}
```

Function prototype

- When I compile the code, an error occurs. Why?

```
#include <stdio.h>

int main( void )
{
    printf ( " Celsius is %lf and Fahrenheit is %lf . \n" , 36.0, c_to_f (36.0));
    return 0;
}

double c_to_f ( double c_temp )
{
    return 9.0 / 5.0 * c_temp + 32;
}
```

전체 솔루션

1 오류

2 경고

0 메시지

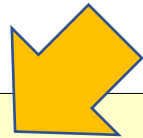
빌드 + IntelliSense

검색 오류 목록

	코드	설명	프로젝트	파일	줄	Suppress
	C4013	'c_to_f'(가) 정의되지 않았습니다. extern은 int형을 반환하는 것으로 간주합니다.	ConsoleApplication3	test.c	5	
	C4477	'printf' : 서식 문자열 '%lf'에 'double' 형식의 인수가 필요하지만 variadic 인수 2의 형식이 'int'입니다.	ConsoleApplication3	test.c	5	
	C2371	'c_to_f': 재정의. 기본 형식이 다릅니다.	ConsoleApplication3	test.c	9	

Function prototype

- *Function prototyping* : Telling the compiler about a function in advance



```
#include <stdio.h>
double c_to_f ( double c_temp ); // Function prototype

int main( void )
{
    printf ( " Celsius is % lf and Fahrenheit is % lf . \n" , 36.0, c_to_f (36.0));
    return 0;
}
double c_to_f ( double c_temp )
{
    return 9.0 / 5.0 * c_temp + 32;
}
```


Function prototype


- A function prototype is something that tells you the function's name, parameters, and return type before the function is defined.
- A function prototype is the same as adding a semicolon(;) to the function header. However, in a function prototype, you don't need to write the parameter names. You only need to write the parameter data types.

```
double c_to_f(double);  
int get_integer(void);
```

The parameter name may be omitted.
You must add ; at the end.

Example without using function prototype

```
int compute_sum ( int n)
{
    int i ;
    int result = 0;
    for ( i = 1; i <= n; i ++ )
        result += i ;
    return result;
}
```



If the function definition comes before the function call, there is no need to define a function prototype .

But this is not the general method .

```
int main( void )
{
    int sum;
    sum = compute_sum (100);
    printf ("sum=%d \n", sum);
}
```

reference

link error

When compiling a source file to create an executable file, a link error may occur. A link error occurs when the compiler cannot find a function.

In other words, a link error occurs when a programmer uses an undefined function.

When a link error occurs, the function name is displayed in the error message. Therefore, you should check the error message to see if the function is definitely defined.

```
1>c:\users\chun\documents\visual studio 2010\projects\hello\hello\hello.c(4): warning C4013:  
'sub' is not defined. extern is assumed to return int.
```

```
1>hello.obj : error LNK2001: unresolved external symbol _sub
```

```
1>C:\Users\chun\documents\visual studio 2010\Projects\hello\Debug\hello.exe : fatal error
```

```
LNK1120: 1 unresolved external reference.
```

```
===== Build: Success 0, Failure 1, Latest 0, Skip 0 =====
```

link error

Library functions

- *Library function* : Functions provided by the compiler
 - Standard Input/Output
 - Mathematical Operations
 - String processing
 - Time handling
 - Error Handling
 - Searching and Sorting Data



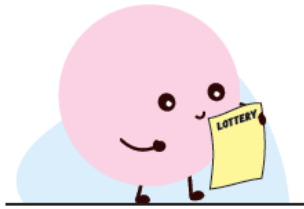
Random number function

- A random number is a number that is generated randomly without any pattern .
- Random numbers It is essential in cryptography, simulations, games, etc.
- rand()
 - Function to generate random numbers
 - Generate a random number from 0 to RAND_MAX



Example : Generating Lotto Numbers

- let's write a program to generate lottery numbers. Lotto numbers are made up of six numbers from 1 to 45.



Practice Code

```
#include <stdio.h>
#include <stdlib.h>
int main( void )
{
    int i ;
    for ( i = 0; i < 6; i ++ )
        printf ( "%d ", rand());

    return 0;
}
```

Generate an integer between
0 and 32767



41 18467 6334 26500 19169 15724

Limited to 1 to 45

- `printf ("%d ", 1+(rand()%45));`



- But every time I run it, the same random number is always generated .

Different random number each time

- If you want to generate different random numbers each time, you need to use **a different seed** .
 - `srand ((unsigned)time(NULL));`

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

```
#define MAX 45
```

```
int main( void )
```

```
{
```

```
    int i ;
```

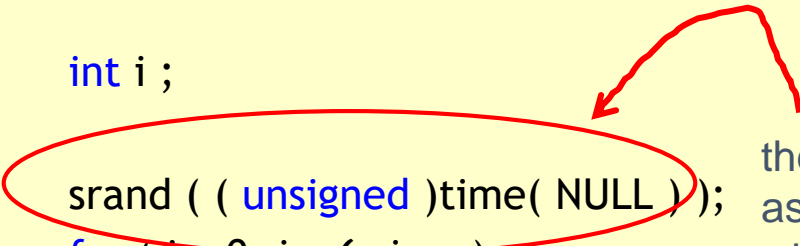
```
    srand ( ( unsigned )time( NULL ) );
```

```
    for ( i = 0; i < 6; i ++ )
```

```
        printf ( "%d " , 1+rand()%MAX );
```

```
    return 0;
```

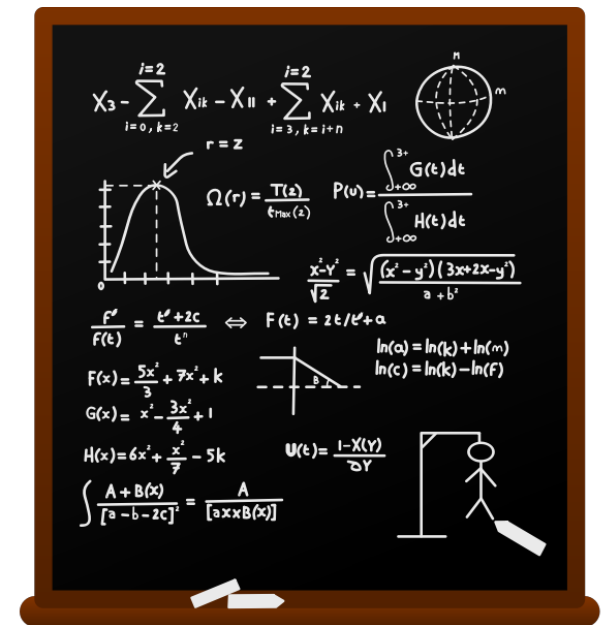
```
}
```



the seed is to use the current time as the seed , since the current time will change each time you run it .

Standard library functions (mathematical functions)

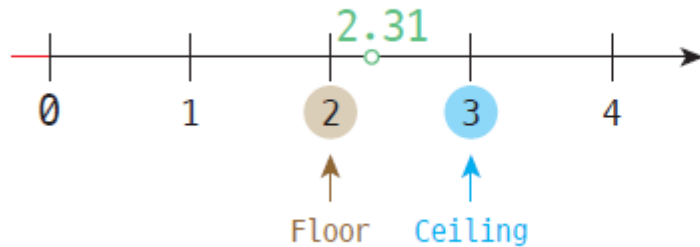
- In this chapter, we will look at library functions that perform numerical calculations. These library functions allow you to perform complex arithmetic operations .
- Prototypes for mathematical functions are in the header file math.h. Mathematical functions generally have parameters and return values of type double .



Math library functions

classification	function	explanation
trigonometry	double sin(double x)	Sine calculation
	double cos(double x)	Cosine calculation
	double tan(double x)	Calculate tangent value
inverse trigonometric functions	double acos(double x)	The range of results for calculating the inverse cosine value is $[0, \pi]$
	double asin(double x)	The range of results for calculating the historical value is $[-\pi/2, \pi]$
	double atan(double x)	The range of results for calculating the inverse tangent is $[-\pi/2, \pi]$
hyperbolic function	double cosh(double x)	hyperbolic cosine
	double sinh(double x)	hyperbolic sine
	double tanh(double x)	hyperbolic tangent
exponential function	double exp(double x)	e^x
	double log(double x)	$\log_e x$
	double log10(double x)	$\log_{10} x$
Other functions	double ceil(double x)	The smallest integer not less than x
	double floor(double x)	The largest integer not greater than x
	double fabs(double x)	absolute value of real x
	int abs(int x)	absolute value of integer x
	double pow(double x, double y)	x^y
	double sqrt(double x)	\sqrt{x}

floor() and ceil() functions



$\lfloor x \rfloor$
floor(x)

$\lceil x \rceil$
ceil(x)

```
double result, value = 1.6;
```

```
result = floor(value);           // result is 1.0 .  
printf ("%lf ", result);
```

```
result = ceil(value);            // result is 2.0 .  
printf ("%lf ", result);
```

fabs ()

- fabs() takes a real number and returns its absolute value .

```
printf (" The absolute value of 12.0 is %f\n", fabs (12.0));  
printf (" The absolute value of -12.0 is %f\n", fabs (-12.0));
```

pow() and sqrt ()

```
printf ("10 to the power of 3 is %.0f.\n", pow(10.0, 3.0));  
printf (" The square root of 16 is %.0f.\n", sqrt (64));
```



*10 to the power of 3 is 1000.
The square root of 16 is 4.*

cos(double x), sin(double x), tan(double x)

```
// Trigonometry function library
```

```
#include < math.h >
```

A standard library containing several mathematical functions.

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

```
int main( void )
```

```
{
```

```
    double pi = 3.1415926535;
```

```
    double x, y;
```

```
    x = pi / 2;
```

```
    y = sin( x );
```

```
    printf ( "sin( %f ) = %f\n" , x, y );
```

```
    y = cos( x );
```

```
    printf ( "cos( %f ) = %f\n" , x, y );
```

```
}
```

```
sin( 1.570796 ) = 1.000000
```

```
cos(1.570796) = 0.000000
```

Etc Function

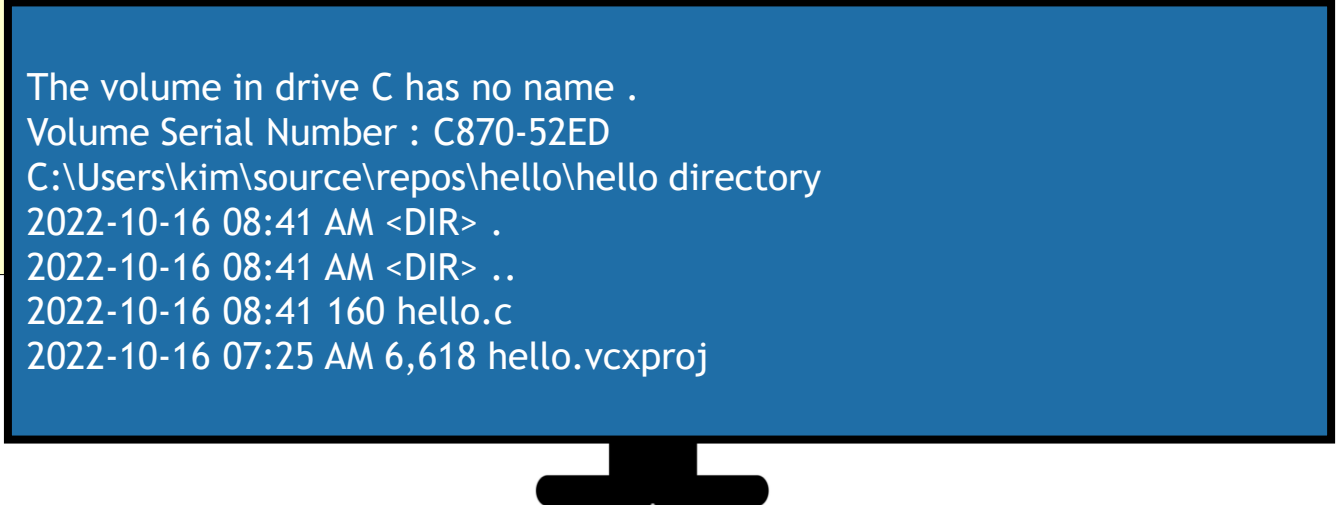
function	explanation
<code>exit()</code>	Calling <code>exit()</code> terminates the running program.
<code>system("...")</code>	<code>system()</code> is a function that passes a command to the operating system's command prompt and executes it. Example You can use it to execute DOS commands such as <code>DIR</code> , <code>COPY</code> , <code>TYPE</code> , <code>CLS</code> , <code>DEL</code> , and <code>MKDIR</code> .
<code>time(NULL)</code>	Returns the current time. Returns the number of seconds elapsed since January 1, 1970.

Example

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

int main( void )
{
    system( "dir " );
    printf ( " Press any key \n" );
    _getch ();
    system( "cls " );

    return 0;
}
```



The volume in drive C has no name .
Volume Serial Number : C870-52ED
C:\Users\kim\source\repos\hello\hello directory
2022-10-16 08:41 AM <DIR> .
2022-10-16 08:41 AM <DIR> ..
2022-10-16 08:41 160 hello.c
2022-10-16 07:25 AM 6,618 hello.vcxproj

Why use functions?

- Eliminates duplication of source code .
- Once a function is created, it can be used when creating other programs .
- Able to break down complex problems into simpler parts.

Complex programs are divided into functions

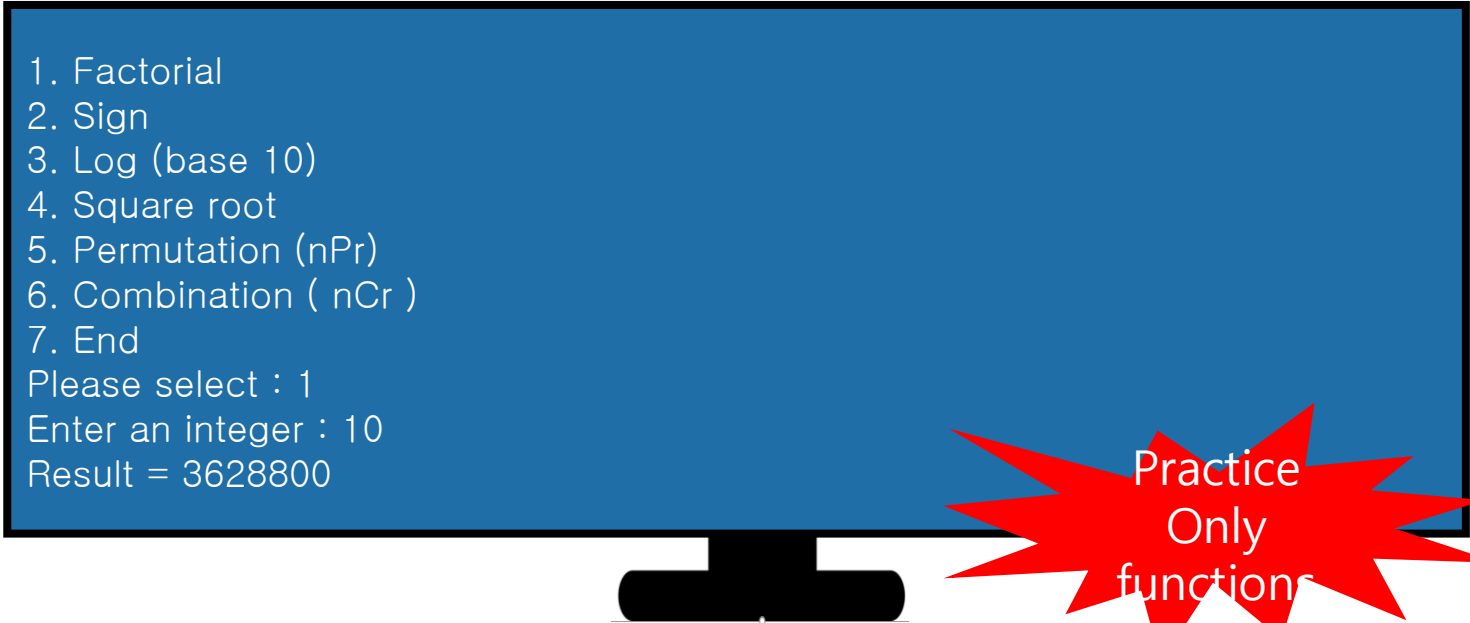
```
int main( void )
{
    // Code to read a list of numbers from the keyboard
    ...
    // Code to sort numbers by size
    ...
    // Code that prints a list of sorted numbers on the screen
    ...
}
```



```
int main( void )
{
    ...
    read_list (); // Function to read a list of numbers from the keyboard
    sort_list (); // Function to sort a list of numbers by size
    print_list (); // Function that prints a list of numbers to the screen
    ...
}
```

Mini Project: Writing an engineering calculator program (Refer ch8-9.c)

- Let's create an engineering calculator that can calculate cosine values. Let's add features that are not implemented yet in the challenge problem .



```
1. Factorial
2. Sign
3. Log (base 10)
4. Square root
5. Permutation (nPr)
6. Combination ( nCr )
7. End
Please select : 1
Enter an integer : 10
Result = 3628800
```



Practice
Only
functions

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

int menu( void )
{
    int n;
    printf ( "1. Factorial \n" );
    printf ( "2. Sign \n" );
    printf ( "3. log (base 10)\n" );
    printf ( "4. Square root \n" );
    printf ( "5. Permutation (nPr)\n" );
    printf ( "6. Combination ( nCr )\n" );
    printf ( "7. End \n" );
    printf ( " Please select : " );
    scanf ( "%d" , &n);
    return n;
}
```

```
void factorial()
{
    long long n, result=1, i ;
    printf ( " Enter an integer : " );
    scanf ( "%lld" , &n);
    for (i = 1; i <= n; i++)
        result = result * i ;
    printf ( " result = %lld \n\n" , result);
}
```

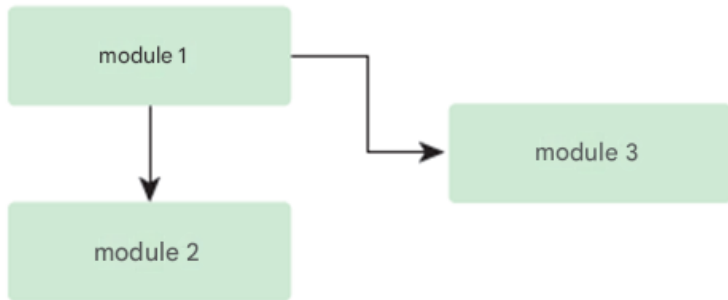
```
void sine()
{
    double a, result;
    printf ( " Enter the angle : " );
    scanf ( "%lf" , &a);
    result = sin(a);
    printf ( " result = %lf \n\n" , result);
}
```

```
void logBase10()
{
    double a, result;
    printf ( " The real number Enter : " );
    scanf ( "% lf " , &a);
    if (a <= 0.0)
        printf ( " Error \n" );
    else {
        result = log10(a);
        printf ( " result = % lf \n\n" , result);
    }
}
```

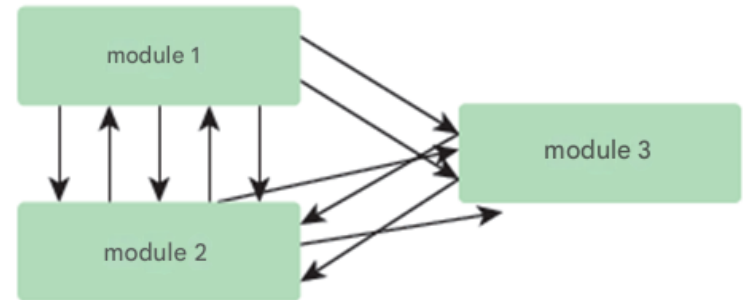
```
int main( void )
{
    while (1) {
        switch (menu()) {
            case 1:
                factorial();
                break ;
            case 2:
                sine();
                break ;
            case 3:
                logBase10();
                break ;
            case 7:
                printf ( " Quitting .\n" );
                return 0;
            default :
                printf ( " Bad choice .\n" );
                break ;
        }
    }
}
```


Modularization

- There should be maximum interaction within a module and minimum interaction between modules.
If the connections between modules are complex, modularization is wrong .



(a) Good modularity



(b) bad modularity

Q & A

