Object oriented programming In C++

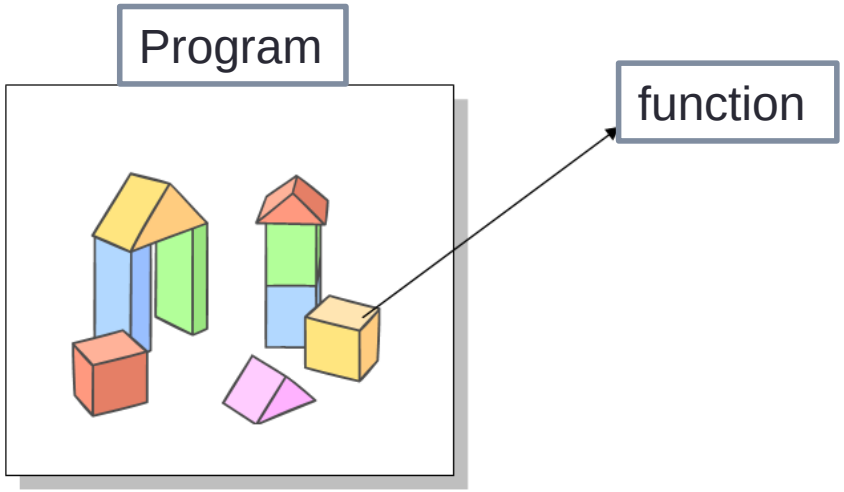
Function 1

Professor 최학남
xncui@inha.ac.kr
Office: high-tech 401

Contents

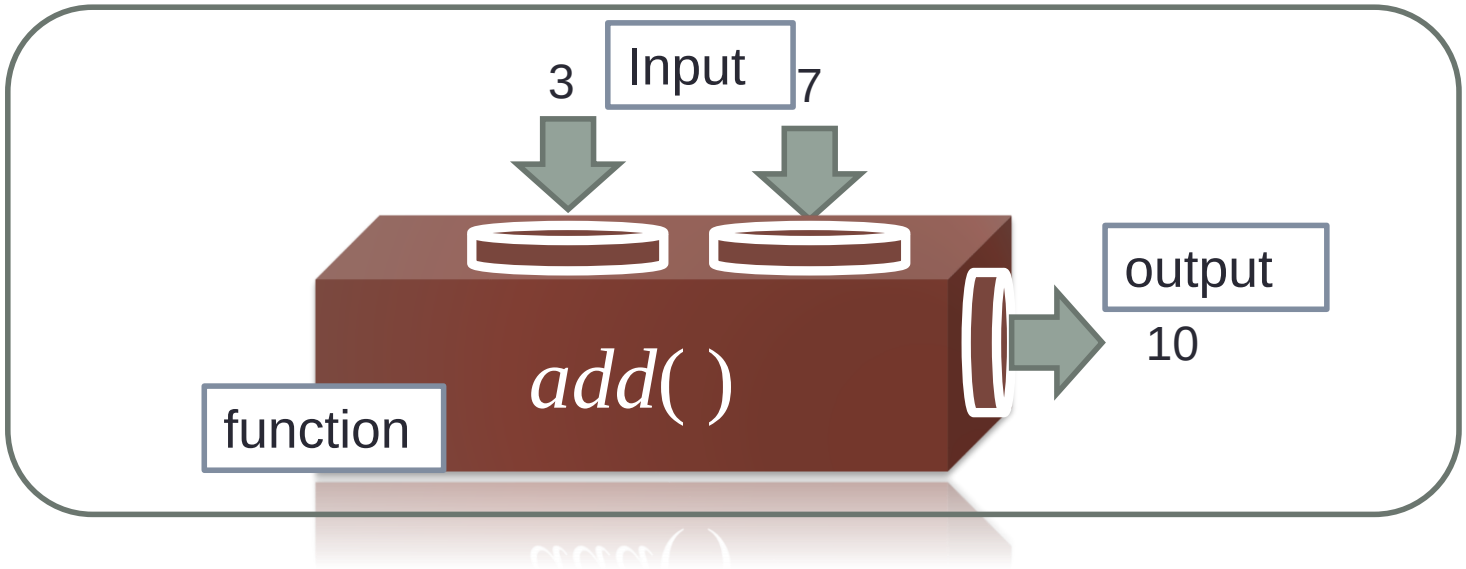
Definition of Function

- What is function(module)?
  - ✓ A function is a self-contained block or a sub-program of one or more statements that performs a special task when called.



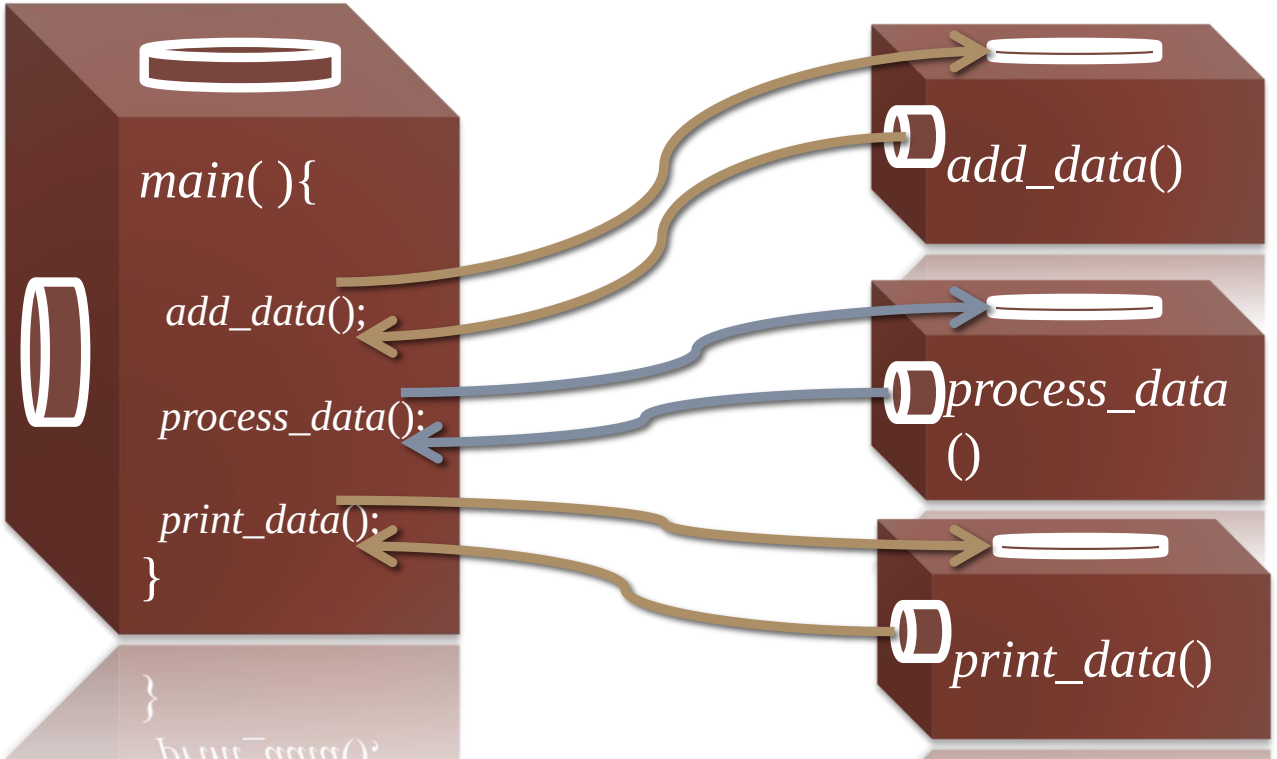
Definition of Function

- What is function call?
  - ✓ A *function call* is an expression containing a simple type name and a parenthesized argument list. The argument list can contain any number of expressions separated by commas.



Definition of Function

- Function execution
  - ✓Execute the *main*() function firstly
  - ✓Each function connected by function call

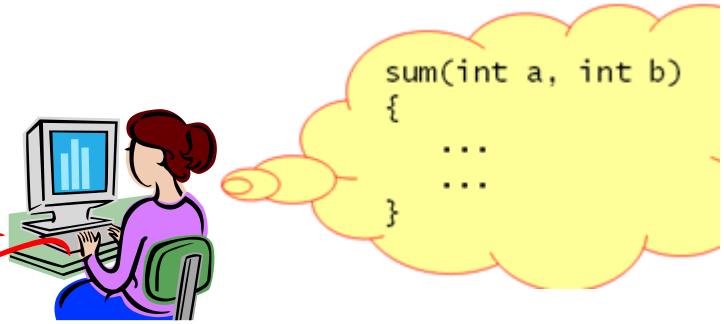


Types of Function

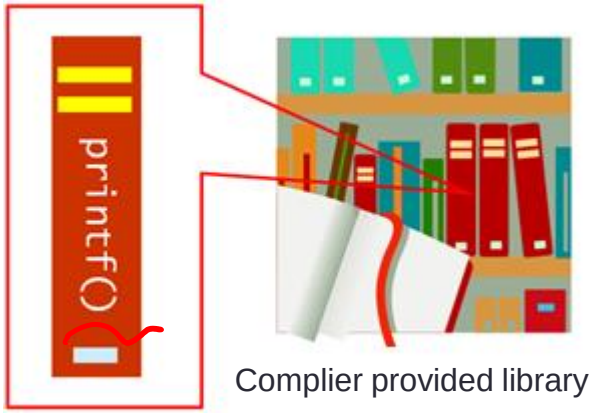
- C language supports two types of function

function

User-defined



Library



Complier provided library

Types of Function

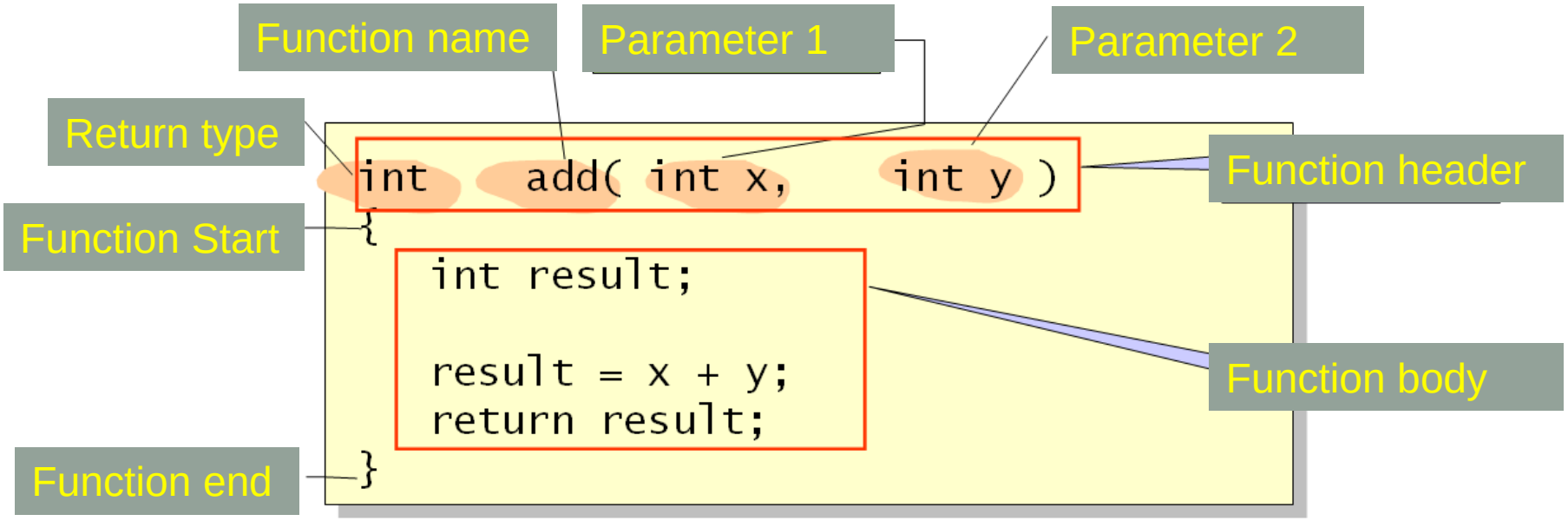
- C language supports two types of functions
  - ✓ Library functions
    - Pre-defined set of functions
    - Their task is limited
    - The user can use them but can’t modify them
  - ✓ User-defined functions
    - User defines according to the requirements
    - Can modify as per requirements
    - One should include the file in which the user-defined functions are stored to call the function in the program

Why use Functions?

- The function defined can be used for any number of times to perform the task
- Using functions large programs can be reduced to smaller ones.
- It is easy to debug and test them.
- Also increases the readability.

Function Declaration

- Structure of function



Function Declaration

- return types
  - ✓ char, int, long, double, float, void etc.

```
int square(int n)
{
    return(n*n);
}
```

```
double square(double n)
{
    return(n*n);
}
```

Function Declaration

- Function_name(arguments/parameter list)

```
[return_type] function_name(data_type param1, data_type param2, …)
{
    declaration;
    statements;
   [return expression;]
}
```

Example 0-1

Returen type: int

Function name : square

Parameter : int n

```
int square(int n)
{
  return(n*n);
}
```

Example 0-2

Returen type: int

Function name : get_max

Parameter : int x, int y

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

Example 0-3

Returen type: int

Function name : absolute

Parameter : int x

```
int absolute(int x)
{
   if( x > 0 )
        return x;
   else
        return -x;
}
```

Example 0-4

Returen type: void
Function name : draw_rect
Parameter : int side

```
void draw_rect(int side)
{
        int x, y;
        for(y = 0; y < side; y++)
        {
                for(x = 0; x < side; x++)
                        printf("*");
                printf("\n");
        }
        return;
}
```

Example 0-5

Returen type: int
Function name : get_integer
Parameter : void

```
int get_integer(void)
{
        int n;

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

        return n;
}
```

Example 0-6

Returen type: int
Function name : power
Parameter : int x, int y

```
int power(int x, int y)
{
        int i;
        long result = 1;

        for(i = 0; i < y; i++)
                result *= x;
        return result;
}
```

Example 0-7

Returen type: int
Function name : factorial
Parameter : int n

```
int factorial(int n)
{
    int i;
    long result = 1;

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

Function Prototype

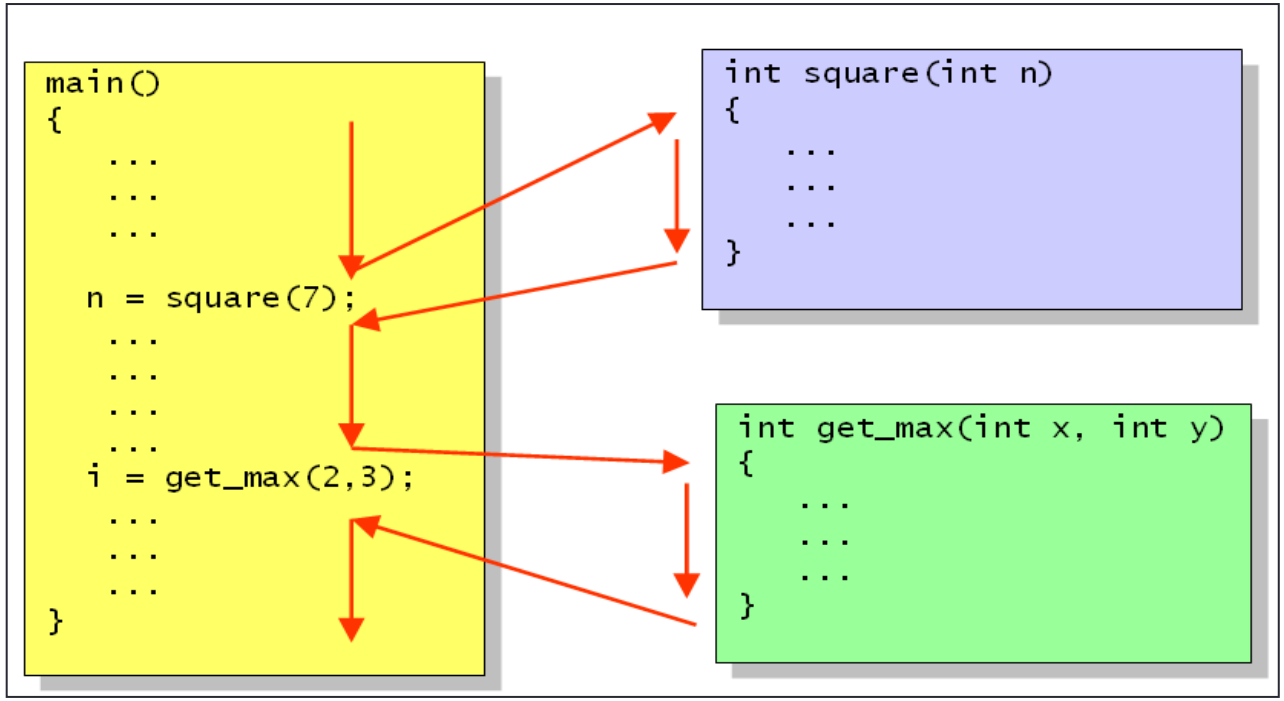
- A prototype statement helps the compiler to check the return type and argument type of the function
- Located before the function call

[**return_type**] **function_name**(data_type param1, data_type param2, …) ;

- Example
  - ✓ float sum(float, int);
  - ✓ float sum(float x, int y);

Function call

- A function call is an expression that passes control and arguments (if any) to a function
- The statements execute the step by step in the function
- Transfer the return-value to the calling function



Example 1

```
2  #include <stdio.h>
3  int add(int a,int b); //Function prototype
void main(){
5      int x = 1,y=2,z;
6      z = add(x,y); // function call
7      printf("z=%d\n",z);
8  }
9
10 int add(int a,int b){ //Function definition
11     return(a+b);
12 }
```

```
z=3
계속하려면 아무 키나 누르십시오 . . .
```

Example 2

```
1  #include <stdio.h>
2  int square(int a);
void main(){
4      int x = 15,z;
5      z = square(x); // function call
6      printf("z=%d\n",z);
7  }
8
9  int square(int a){
10     return(a*a);
11 }
```

```
z=225
계속하려면 아무 키나 누르십시오 . . .
```

Arguments and parameters

- Arguments
  - ✓ Actual arguments
  - ✓ Actual parameters
    - The arguments of calling function are actual arguments

- Parameters
  - ✓ Formal parameters
  - ✓ Formal arguments:
    - The arguments of called function are formal arguments

Arguments and parameters

```
int main(void)
{

  ...
  i = get_max(2, 3);
  ...
}
```

Arguments

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

parameters

```
#include <stdio.h>
int add(int x, int y)
{
      return (x + y);
}

int main(void)
{
      // 2and 3 is the argument of the
     function add().
      add(2, 3);

      // 5and 6 is the argument of the
     function add().
      add(5, 6);
      return 0;
}
```

Example 2

```
1  #include <stdio.h>
2  int square(int a);
void main(){
4      int x = 15, z;
5      z = square(x); // function call
6      printf("z=%d\n",z);
7  }
8
9  int square(int a){
10     return(a*a);
11 }
```

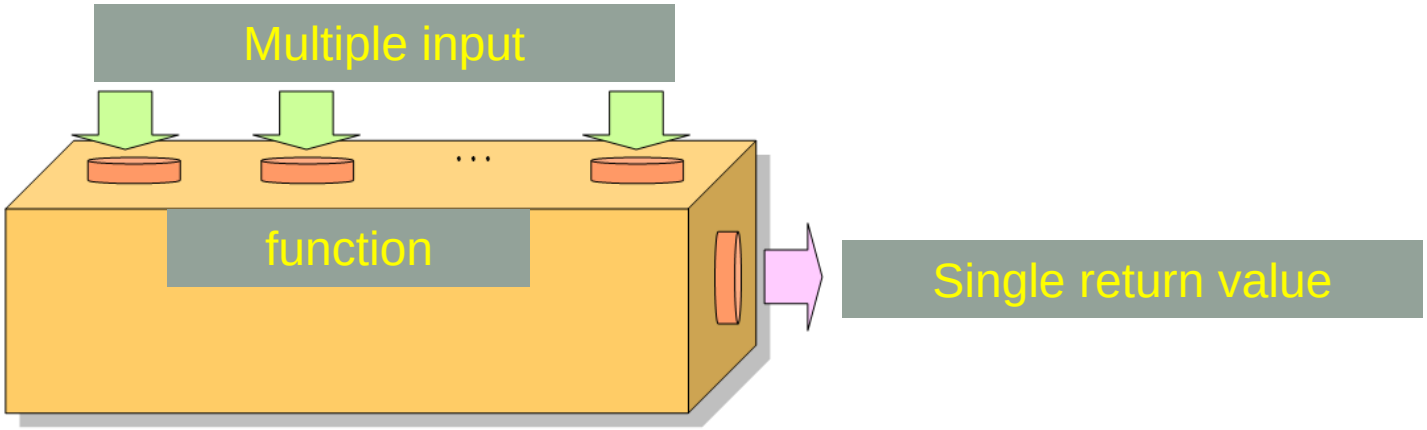
Actual argument

Formal argument

x is the actual argument
a is the formal argument

Return value

- Only single return value
- Default is *int* type – in C
- Must be set the return type in C++

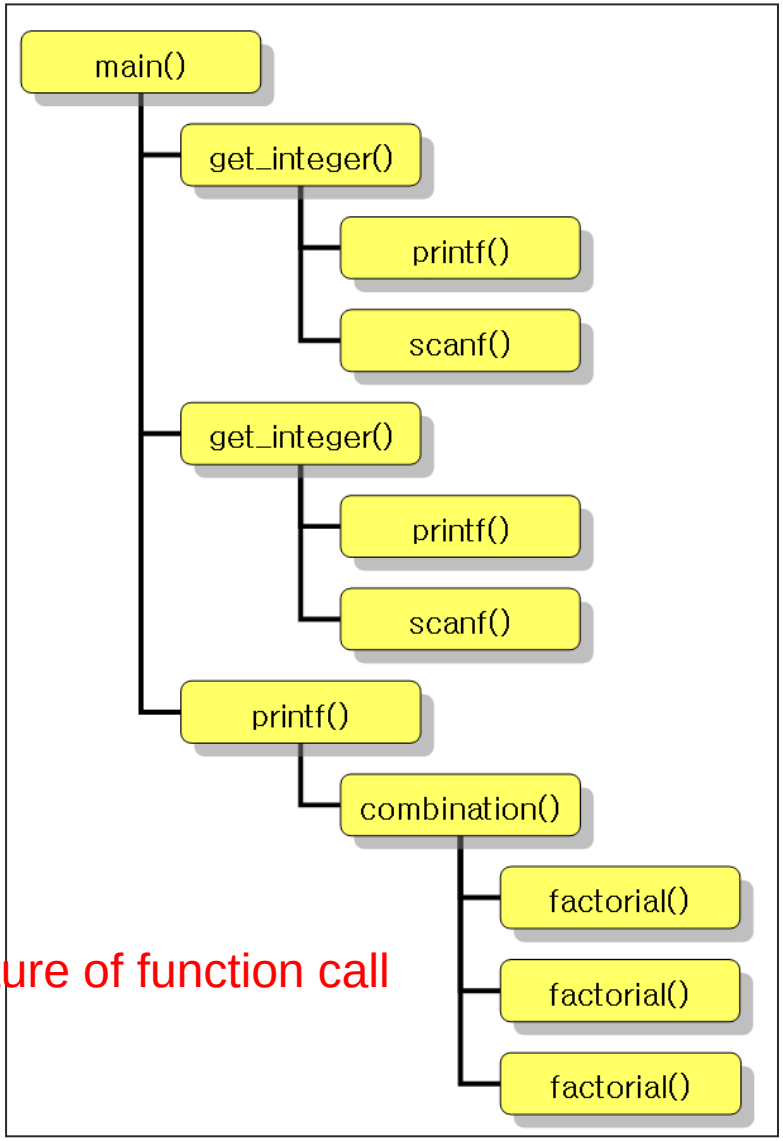



```
return 0;
return(0);
return x;
return x+y;
return x*x+2*x+1;
```

Example 3: Combination

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

$$C(3,2) = \frac{3!}{(3-2)!2!} = \frac{6}{2} = 3$$



Structure of function call

Example 3: Combination

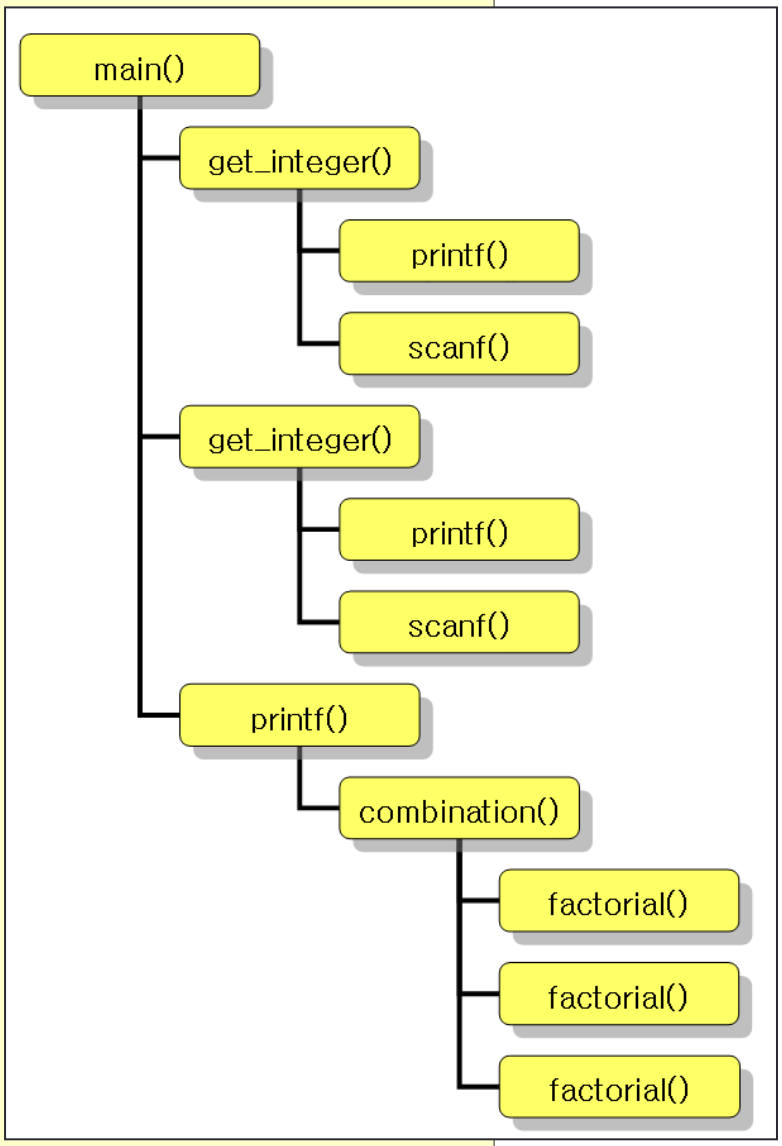
```
#include <stdio.h>

int get_integer(void);
int combination(int n, int r);
int factorial(int 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;
}

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

Example 3: Combination

```
int get_integer(void)
{
        int n;

        printf("insert the integer value: ");
        scanf("%d", &n);
        return n;
}
int factorial(int n)
{
        int i;
        long result = 1;

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

Example 3: Combination

```c
#include <stdio.h>
int get_integer(void);
int combination(int n, int r);
int factorial(int 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;
}

int combination(int n, int r)
{
	return (factorial(n)/(factorial(r) * factorial(n-r)));
}
int get_integer(void)
{
	int n;

	printf("insert the integer value: ");
	scanf("%d", &n);
	return n;
}
int factorial(int n)
{
	int i;
	long result = 1;

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



HW6-1

- Example 0-1~7 함수를 실행하는 코드 작성, 실행
- Example 3 실행 및 function call structure 그리기