

# **Basic Computer Programming**

## Lecture 8

Electrical & Electronics Engineering  
Chung-Ang University

# Contents

- Understand the concept of a function.
- Write a function.
- Understand the return values and parameters of the function.
- Understand global and local variables.
- Understand and use circular calls.

# The program we will make in this chapter

- Define and use a function that converts temperature.



```
Microsoft Visual Studio 디버그 콘솔
화씨 온도 32.000000은 섭씨 온도 0.000000에 해당한다.
```

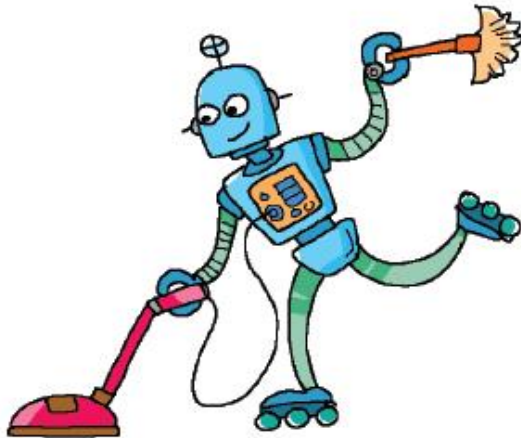
- Calculate factorial with a function



```
Microsoft Visual Studio 디버그 콘솔
정수를 입력하시오:5
factorial(5)
factorial(4)
factorial(3)
factorial(2)
factorial(1)
5!은 120입니다.
```

# A repetitive task

- Just as some tasks are repeated in everyday life, there are repetitive tasks in programs.



# The necessity of a function

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

```
int main(void)
```

```
{
```

```
printf("*****\n");  
printf("한국대학교 컴퓨터 공학과 \n");  
printf("홍길동 \n");  
printf("*****\n");
```

```
printf("*****\n");  
printf("한국대학교 컴퓨터 공학과 \n");  
printf("홍길동 \n");  
printf("*****\n");
```

```
return 0;
```

```
}
```

# The necessity of a function

A function was defined. Once defined, the function can be called several times to execute.

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

```
void print_name()  
{  
    printf("*****\n");  
    printf("한국대학교 컴퓨터 공학과 \n");  
    printf("홍길동 \n");  
    printf("*****\n");  
}
```

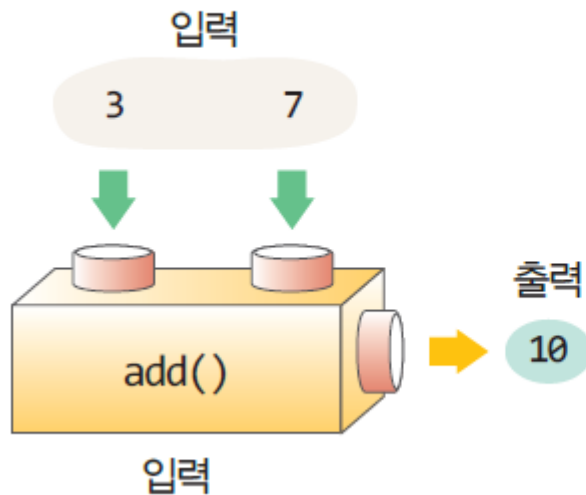
```
int main(void)  
{  
    print_name();  
    print_name();  
  
    return 0;  
}
```

# Advantages of Functions

- The function can be used to prevent code duplication.
- Once a function is written, it can be reused several times.
- The function allows the entire program to be divided into modules, making the development process easier and more systematic, making maintenance easier.

# The concept of a function

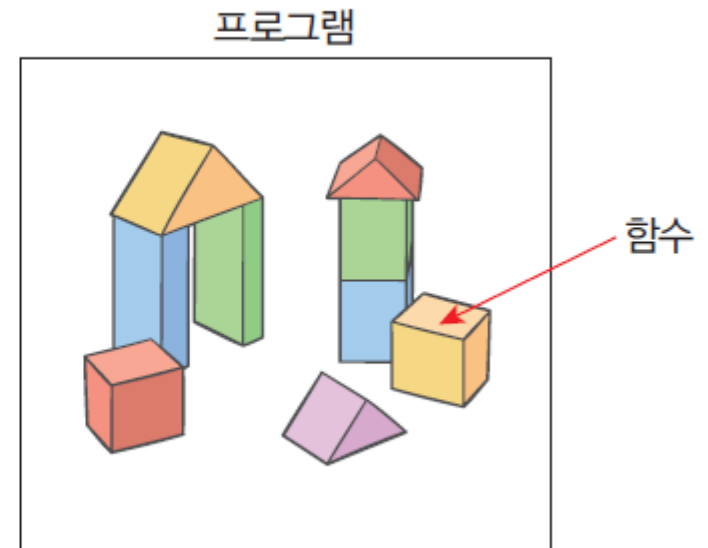
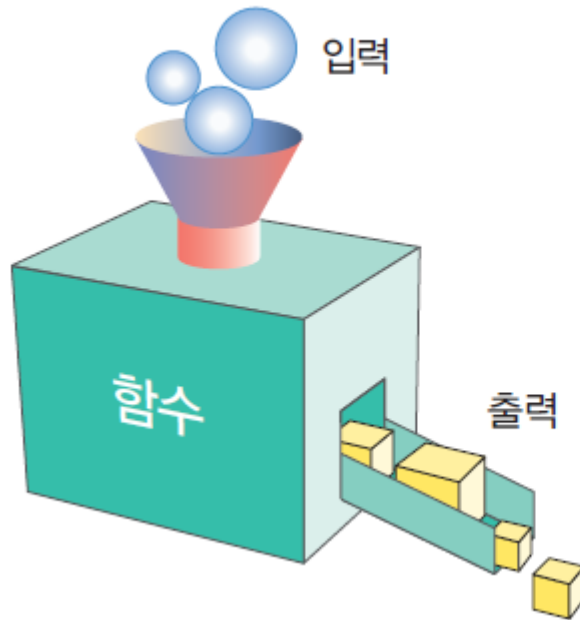
- *Function*: A set of commands that perform a particular task and return the result





# A function is a block that make up a program

- A program consists of several functions.
- The function acts like a Lego block. Functions are easy to understand if you think of them as parts that make up a program.



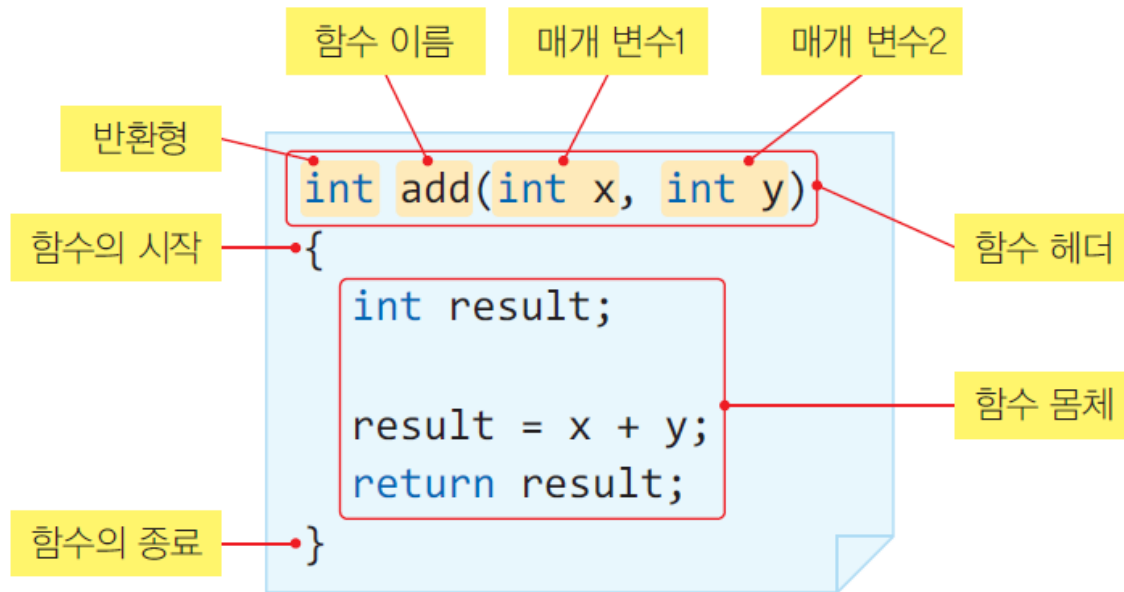
# Interim check



## 중간점검

1. 함수가 필요한 이유는 무엇인가?
2. 함수와 프로그램의 관계는?

# Definition of a function



# Return type


- Indicates the type of data returned before the function name.
- char, int, long, double ...
- void if no return type exists

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

```
void print_info()  
{  
    ....  
}
```

# Function Name

- Verbs + nouns in general
- (Example)
  - square() // a function that squares integers
  - compute\_average() // function to obtain the average
  - get\_integer() // function that accepts integers

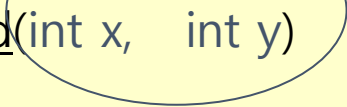


```
int add(int x, int y)
{
    int result;

    result = x + y;
    return result;
}
```

# Parameters

- Parameter: a variable whose function has data from the outside



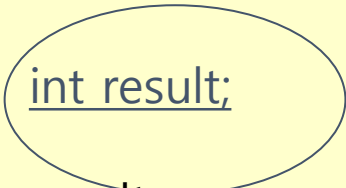
```
int add(int x, int y)
{
    int result;

    result = x + y;
    return result;
}
```

# Local Variables

- Local variable: a variable defined in a function

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

A diagram illustrating a local variable. The code is enclosed in a yellow rectangular box. Inside the box, the function definition for 'add' is shown. The line 'int result;' is underlined and circled with a blue oval, highlighting it as a local variable. The rest of the function code, including the assignment 'result = x + y;' and the 'return' statement, is not circled.

# Check



## 참고

함수에는 얼마든지 많은 문장들을 넣을 수 있지만 함수의 길이가 지나치게 길어지면 좋지 않다. 기본적으로 하나의 함수는 하나의 작업만을 수행하여야 한다. 일반적으로 하나의 함수는 30행을 넘지 않도록 하는 것이 좋다.



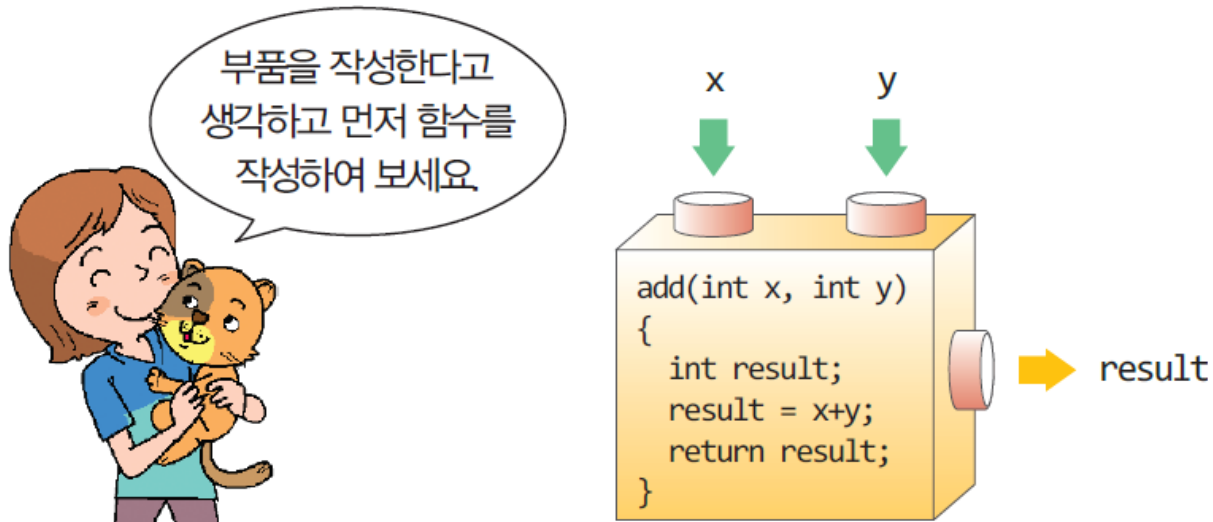
## 참고

만약 함수의 반환형을 명시하지 않으면 C 컴파일러는 `int` 형을 가정한다. 그러나 특별한 경우가 아니면 반환형을 생략하면 안 된다. 반환형이 `int`일지라도 항상 반환형을 명시하는 것이 좋다. 또한 반환 값이 있는 함수에서 값을 반환하지 않으면 예측 불가능한 값이 전달될 수 있다. 또 반대로 반환 값이 `void`로 지정된 함수에서 값을 반환하면 문법적인 오류가 발생한다.



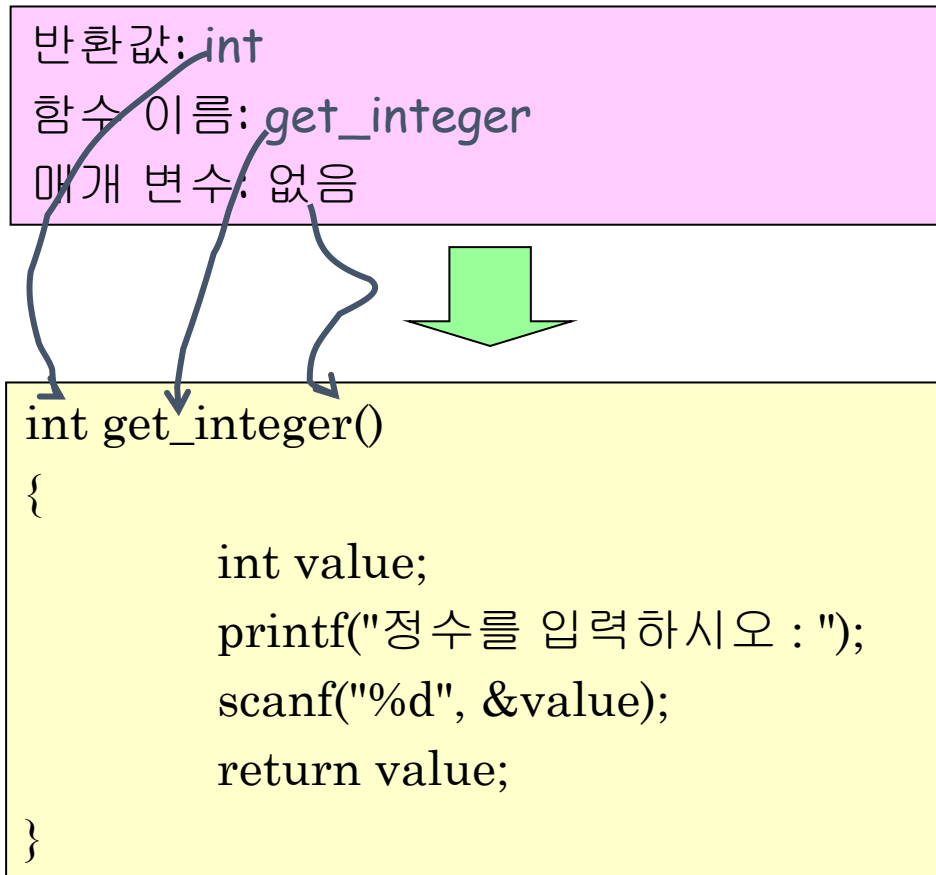
# Example of defining a function

- Suppose a function is a part of a program.
- It receives input, works, and generates results.



# Example #1

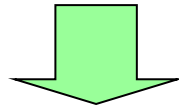
- Returns an integer



# Example #2

- A function that calculates the larger of two integers

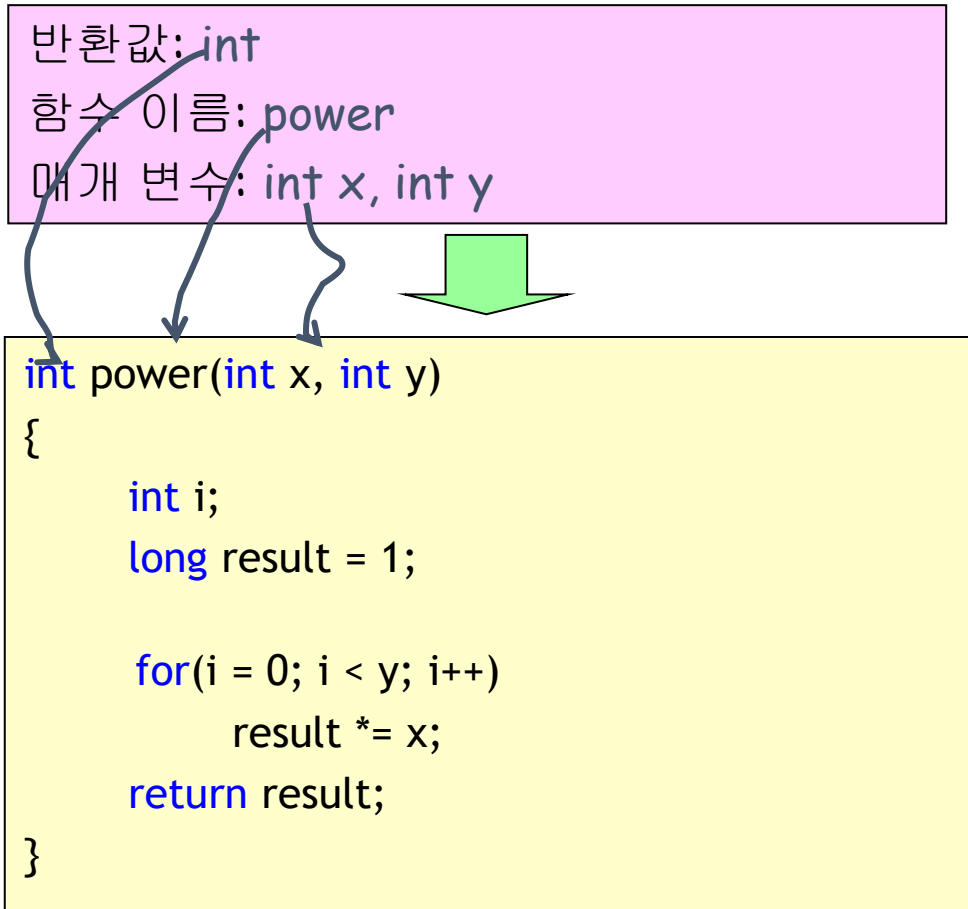
반환값: `int`  
함수 이름: `get_max`  
매개 변수: `int x, int y`



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

# Example #3

- Function that calculates the power of an integer ( $x^y$ )



# Interim check



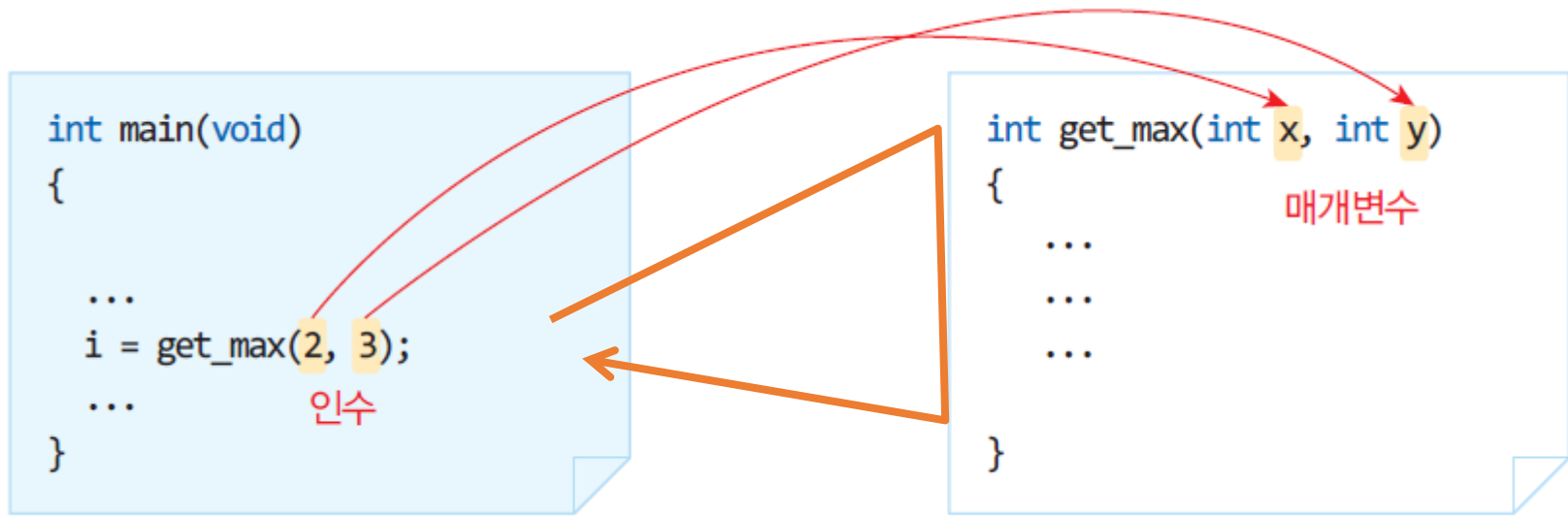
## 중간점검

1. 함수 이름 앞에 `void`가 있다면 무슨 의미인가?
2. 함수가 작업을 수행하는데 필요한 데이터로서 외부에서 주어지는 것을 무엇이라고 하는가?
3. 함수 몸체는 어떤 기호로 둘러싸여 있는가?
4. 함수의 몸체 안에서 정의되는 변수를 무엇이라고 하는가?



# Function Call and Return

- Function call:
- To write down the name of a function to use it.
- The commands in the function are executed sequentially.
- After executing the sentence, return to the called position.
- Results can be transferred.



# Arguments and Parameters

- *Argument*: Values of parameters.
- *Parameter*: Variables of a function.

```
int main(void)
{
    ...
    i = get_max(2, 3);
    ...
}
```

인수

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

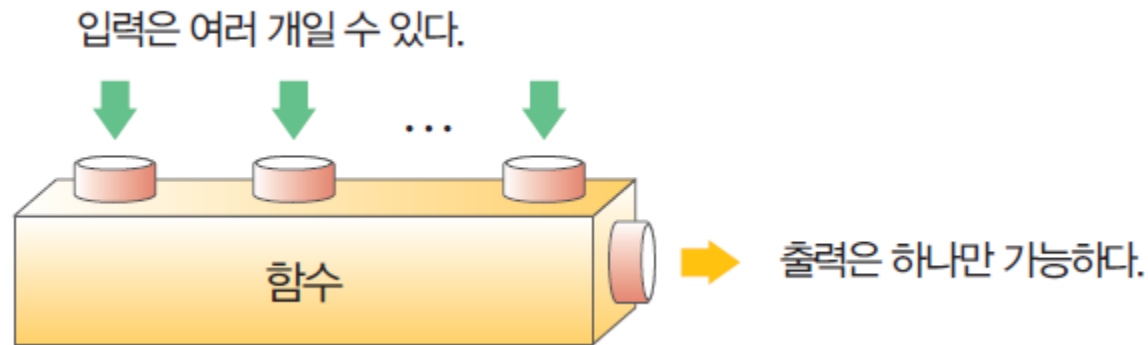
매개변수



get\_max(10); // 인수가 두 개이어야 한다.  
get\_max(0.1, 0.2); // get\_max() 인수의 타입은 정수이어야 한다.  
get\_max( ); // 인수가 두 개이어야 한다.

# Return value

- Return value: Forwarding the result value of a task to the place where the function is called
- Multiple arguments are possible, but there should be only one return value



```
return 0;  
return (x);  
return x+y;  
return;
```



# Example #1

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

// 함수를 정의한다.
int get_integer()
{
    int value;
    printf("정수를 입력하시오 : ");
    scanf("%d", &value);
    return value;
}

int main(void)
{
    int x = get_integer(); // 함수를 호출한다.
    int y = get_integer(); // 함수를 호출한다.
    int result = x + y;
    printf("두수의 합 = %d \n", result);

    return 0;
}
```

# Results



The screenshot shows the 'Microsoft Visual Studio 디버그 콘솔' (Debug Console) window. It contains three lines of text: '정수를 입력하시오 : 10', '정수를 입력하시오 : 20', and '두 수의 합 = 30'. The window has a standard Windows title bar with minimize, maximize, and close buttons.

```
Microsoft Visual Studio 디버그 콘솔
정수를 입력하시오 : 10
정수를 입력하시오 : 20
두 수의 합 = 30
```

# Example #2

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

```
// 함수를 정의한다.
```

```
int get_integer()
{
    int value;
    printf("정수를 입력하시오 : ");
    scanf("%d", &value);
    return value;
}
```

```
// 함수를 정의한다.
```

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

# Example #2

```
int main(void)
{
    int a = get_integer();    // 함수 호출
    int b = get_integer();    // 함수 호출

    printf("두수 중에서 큰 수는 %d입니다.\n", get_max(a, b)); // 함수 호출
    return 0;
}
```

Microsoft Visual Studio 디버그 콘솔

정수를 입력하시오 : 10  
정수를 입력하시오 : 20  
두수 중에서 큰 수는 20입니다.

# Example #3

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
// 함수를 정의한다.
int get_integer()
{
    int value;
    printf("정수를 입력하시오 : ");
    scanf("%d", &value);
    return value;
}
// 함수를 정의한다.
int power(int x, int y)
{
    int i, result = 1;

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

# Example #3

```
int main(void)
{
    int x = get_integer(); // 함수를 호출한다.
    int y = get_integer(); // 함수를 호출한다.
    int result = power(x, y);
    printf("%d의 %d승 = %d \n", x, y, result);

    return 0;
}
```



The screenshot shows the 'Microsoft Visual Studio 디버그 콘솔' (Debug Console) window. It contains the following text: '정수를 입력하시오 : 10', '정수를 입력하시오 : 2', and '10의 2승 = 100'. The window has a standard Windows title bar with minimize, maximize, and close buttons.

```
C:\> Microsoft Visual Studio 디버그 콘솔
정수를 입력하시오 : 10
정수를 입력하시오 : 2
10의 2승 = 100
```

# Interim check



## 중간점검

1. 인수와 매개 변수는 어떤 관계가 있는가?
2. 사용자로부터 실수를 받아서 반환하는 함수 `get_real()`을 작성하고 테스트하라.
3. 함수 정의의 첫 번째 줄에는 어떤 정보들이 포함되는가? 이것을 무엇이라고 부르는가?
4. 함수가 반환할 수 있는 값의 개수는?
5. 함수가 값을 반환하지 않는다면 반환형은 어떻게 정의되어야 하는가?



# function prototyping

- *Function prototyping*: Informing the compiler about the function in advance

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

```
int compute_sum(int n);
```

```
int main(void) {
```

```
    int sum;
```

```
    sum = compute_sum(100);
```

```
    printf("1부터 100까지의 합 = %d \n", sum);
```

```
    return 0;
```

```
}
```

```
int compute_sum(int n) {
```

```
    int i, result = 0;
```

```
    for (i = 1; i <= n; i++)
```

```
        result += i;
```

```
    return result;
```

```
}
```

함수 원형



# Interim check



## 참고

함수 원형을 사용하지 않는 방법도 있다. 함수 원형이란 근본적으로 컴파일러에게 함수에 대한 정보를 주기 위하여 만들어진 것이다. 따라서 사용하려는 함수의 정의가 먼저 등장한다면 구태여 함수 원형을 표시할 필요가 없다. 하지만 함수 호출이 서로 물고 물리는 경우에는 이 방법이 불가능하다. 따라서 대부분의 경우에 먼저 함수 원형을 적어 주는 것이 권장된다.



## 중간점검

1. 함수 정의와 함수 원형 선언의 차이점은 무엇인가?
2. 함수 원형에 반드시 필요한 것은 아니지만 대개 매개 변수들의 이름을 추가하는 이유는 무엇인가?
3. 다음과 같은 함수 원형을 보고 우리가 알 수 있는 정보는 어떤 것들인가?

```
double pow(double, double);
```

# Lab: Temperature conversion function

- Write and test a function `FtoC()` that converts Celsius temperature to Fahrenheit.

```
Microsoft Visual Studio 디버그 콘솔
화씨 온도 32.000000은 섭씨 온도 0.000000에 해당한다.
```



# Sol.

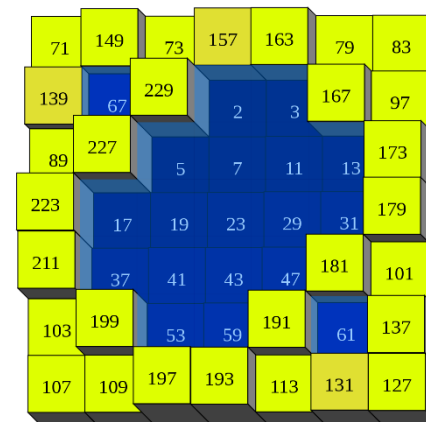
```
#include <stdio.h>
double FtoC(double temp_f); // 함수 원형 정의

int main(void)
{
    double c, f;
    f = 32.0;
    c = FtoC(f);           // 함수 호출
    printf("화씨온도 %lf은 섭씨온도 %lf에 해당한다. \n", f, c);
    return 0;
}

double FtoC(double temp_f) // 함수 정의
{
    double temp_c;
    temp_c = (5.0 * (temp_f - 32.0)) / 9.0;
    return temp_c;
}
```

# Lab: Prime number check function

- Write a function that receives an integer and returns whether it is a prime number or not.



# Sol.

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
int check_prime(int);

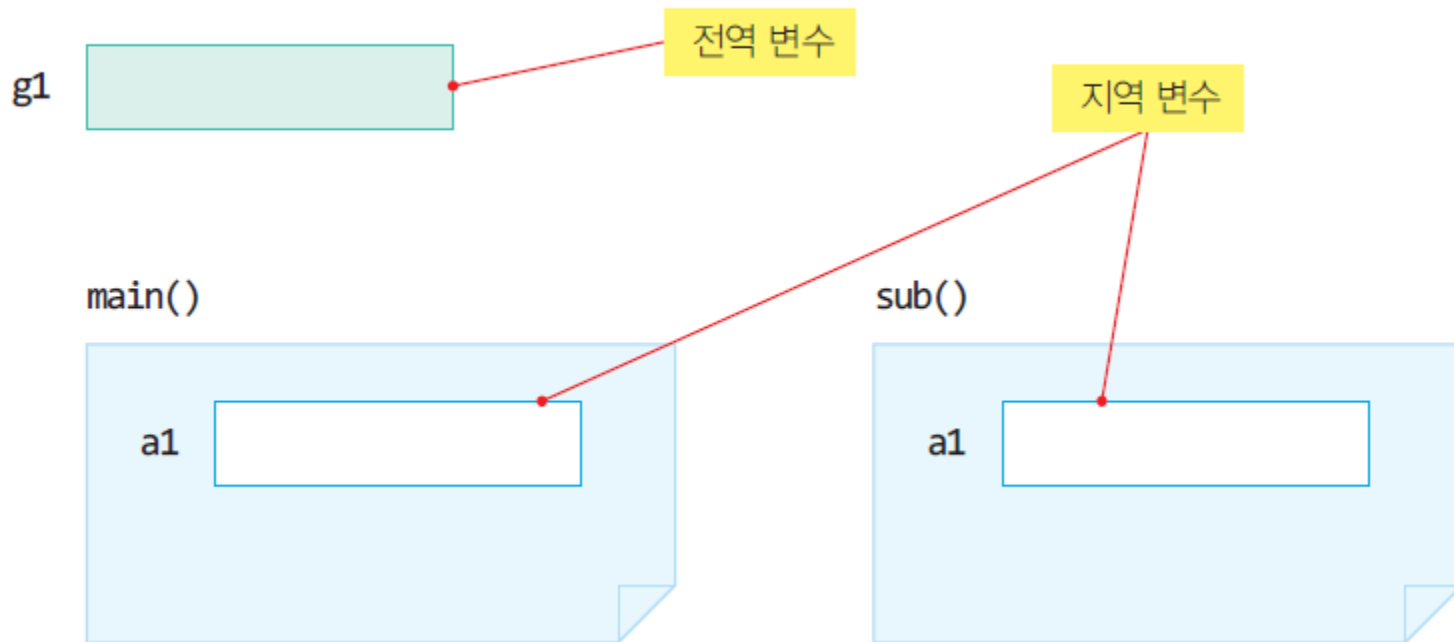
int main(void)
{
    int k;
    printf("정수를 입력하시오: ");
    scanf("%d", &k);
    if (check_prime(k) == 1) printf("소수입니다. \n");
    else printf("소수가 아닙니다. \n");
    return 0;
}
```

# Sol.

```
int check_prime(int n) {  
    int is_prime = 1;    // 일단 소수라고 가정한다.  
    for (int i = 2; i < n; ++i) {  
        if (n % i == 0) {  
            is_prime = 0;  
            break;  
        }  
    }  
    return is_prime;  
}
```

# Local and Global Variables

- Variables defined within a function are called local variables and can only be used within that function.
- Variables declared outside the function are called global variables.



# Local Variables

- Local variable: a variable declared within a function or block

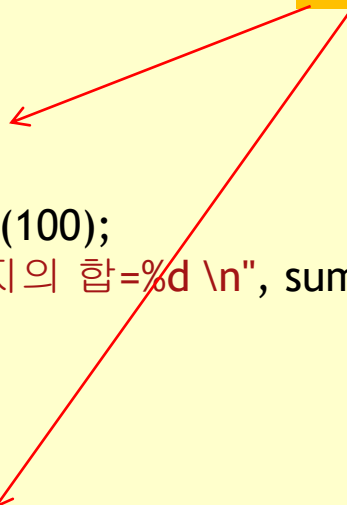
```
#include <stdio.h>
int compute_sum(int n);

int main(void)
{
    int sum;
    sum = compute_sum(100);
    printf("1부터 100까지의 합=%d \n", sum);
    return 0;
}

int compute_sum(int n)
{
    int i;
    int result = 0;

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

지역 변수





# Range of use of local variables

```
int sub1()
```


```
{
```

```
    int l;
```

```
    ...
```

```
}
```

지역 변수  $x$ 의 유효한  
범위



```
void sub2()
```

```
{
```

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

```
}
```

# Initial value of the local variable

```
int compute_sum(int n)
{
    int i, result;

    for(i=0; i<=n; i++)
        result += i;

    return result;
}
```



## Debug Error!

Program:

...15\Projects\ConsoleApplication1\Debug\ConsoleApplication1.exe

Module:

...15\Projects\ConsoleApplication1\Debug\ConsoleApplication1.exe

File:

Run-Time Check Failure #3 - T

(Press Retry to debug the application)



# Check



## 참고: 블록에서도 지역 변수를 선언할 수 있다.

블록은 중괄호 {}을 사용하여 만드는 구간이다. 여기서도 지역 변수를 선언할 수 있다. 블록이 종료되면 지역 변수도 같이 사라진다.

```
while(1) {  
    int x; // 블록 안에서 선언된 지역 변수  
    ...  
} // 변수 x는 여기서 사라진다.
```



## 참고: 함수의 매개 변수

함수의 헤더 부분에 정의되어 있는 매개 변수도 일종의 지역 변수이다. 즉 지역 변수가 지니는 모든 특징을 가지고 있다. 지역 변수와 다른 점은 함수를 호출할 때 넣어주는 인수 값으로 초기화되어 있다는 점이다.

```
int inc(int counter) // counter도 지역 변수의 일종이라고 생각할 수 있다.  
{  
    counter++; // 지역 변수처럼 사용할 수 있다.  
    return counter;  
}
```

# Global Variables

- Global variable: variable declared outside of function
- If you don't give the initial value, it's zero.

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

```
int global = 123;
```



```
void sub1()
{
    printf("global=%d\n", global);
}
void sub2()
{
    printf("global=%d\n", global);
}
int main(void)
{
    sub1();
    sub2();
    return 0;
}
```

```
Microsoft Visual Studio 디버그 콘솔
global=123
global=123
```

# Characteristics of global variables

- Global variables are created at the same time as the program starts and exist in memory until the program ends.
- Global variables are thought to be quite convenient, but experts do not recommend using them. The reason is that the advantage of being accessible from anywhere can be a disadvantage. As programs become more complex, it is often difficult to know where the global variables are being changed.



# Global and local variables with the same name

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

```
int sum = 123;
```

```
int main(void)
```

```
{
```

```
    int sum = 321;
```

```
    printf("sum=%d \n", sum);
```

```
    return 0;
```

```
}
```

지역 변수가  
전역 변수를  
가린다.

Microsoft Visual Studio 디버그 콘솔

sum=321

# Interim check



## 중간점검

1. 변수의 범위에는 몇 가지의 종류가 있는가?
2. 블록 범위를 가지는 변수를 무엇이라고 하는가?
3. 지역 변수를 블록의 중간에서 정의할 수 있는가?
4. 똑같은 이름의 지역 변수가 서로 다른 함수 안에 정의될 수 있는가?
5. 지역 변수가 선언된 블록이 종료되면 지역 변수는 어떻게 되는가?
6. 지역 변수의 초기값은 얼마인가?
7. 전역 변수는 어디에 선언되는가?
8. 전역 변수의 생존 기간과 초기값은?
9. 똑같은 이름의 전역 변수와 지역 변수가 동시에 존재하면 어떻게 되는가?



# Lab: Find the sum of prime numbers

- Let's write a program to test which integers can be represented by the sum of two prime numbers.



```
Microsoft Visual Studio 디버그 콘솔
양의 정수를 입력하시오: 33
33 = 2 + 31
33 = 31 + 2
```

$$33 = 2 + 31$$



# Sol.

```
#include <stdio.h>
int check_prime(int n);

int main(void) {
    int n, flag = 0;
    printf("양의 정수를 입력하시오: ");
    scanf("%d", &n);

    for (int i = 2; i < n; i++) {
        if (check_prime(i) == 1) {
            if (check_prime(n - i) == 1) {
                printf("%d = %d + %d\n", n, i, n - i);
                flag = 1;
            }
        }
    }
    if (flag == 0)
        printf("%d은 소수들의 합으로 표시될 수 없습니다.\n", n);
    return 0;
}
```

# Sol.

```
int check_prime(int n) {  
    int is_prime = 1;    // 일단 소수라고 가정한다.  
    for (int i = 2; i < n; ++i) {  
        if (n % i == 0) {  
            is_prime = 0;  
            break;  
        }  
    }  
    return is_prime;  
}
```

# Static local variables

- Static variables: variables that are used only in blocks but are not automatically deleted when out of blocks
- Write static in front of it.

```
void sub()
{
    static int count;
    ....

    return;
}
```

정적 변수

# An example of static

```
#include <stdio.h>
void sub(void)
{
    int auto_count = 0;
    static int static_count = 0;

    auto_count++;
    static_count++;
    printf("auto_count=%d\n", auto_count);
    printf("static_count=%d\n", static_count);
}

int main(void) {
    sub();
    sub();
    sub();
    return 0;
}
```

# Result



```
Microsoft Visual Studio 디버그 콘솔
auto_count=1
static_count=1
auto_count=1
static_count=2
auto_count=1
static_count=3
```

# Circular call

- Circulation is a programming technique in which a function invokes itself to solve a problem

$$n! = \begin{cases} 1 & n=0 \\ n*(n-1)! & n \geq 1 \end{cases}$$



# Calculating factorial

$$\begin{aligned}\text{factorial}(5) &= 5 * \text{factorial}(4) \\ &= 5 * 4 * \text{factorial}(3) \\ &= 5 * 4 * 3 * \text{factorial}(2) \\ &= 5 * 4 * 3 * 2 * \text{factorial}(1) \\ &= 5 * 4 * 3 * 2 * 1 \\ &= 120\end{aligned}$$

# Circular call

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
int factorial(int n);

int main(void)
{
    int x = 0, result;

    printf("정수를 입력하시오:");
    scanf("%d", &x);

    result = factorial(x);
    printf("%d!은 %d입니다.\n", x, result);

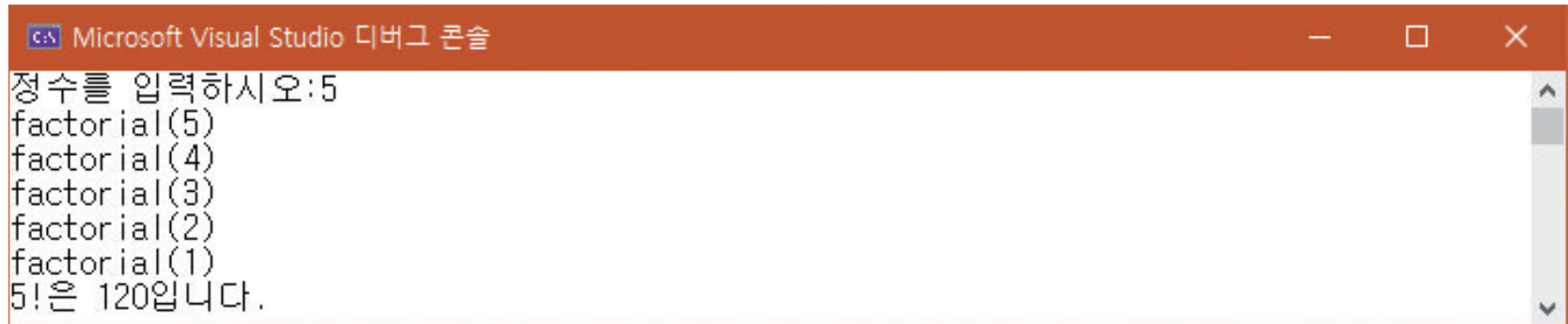
    return 0;
}

int factorial(int n)
{
    printf("factorial(%d)\n", n);

    if (n <= 1) return 1;
    else return n * factorial(n - 1);
}
```



# Results



A screenshot of the Microsoft Visual Studio Debug Console window. The window has a title bar with the text "Microsoft Visual Studio 디버그 콘솔" and standard Windows window controls (minimize, maximize, close). The console output shows the following text:

```
정수를 입력하시오:5  
factorial(5)  
factorial(4)  
factorial(3)  
factorial(2)  
factorial(1)  
5!은 120입니다.
```

The text is displayed in a monospaced font. A vertical scrollbar is visible on the right side of the console window.

# Interim check



## 중간점검

1. factorial() 함수를 순환을 사용하지 않고 반복문으로 다시 작성하여 보자.
2. factorial() 함수 안에 `if( n <= 1 ) return;`이라는 문장이 없으면 어떻게 될까?

# Lab: Fibonacci Sequence

- Write a program that uses circular calls to output Fibonacci sequences. A Fibonacci sequence is a sequence defined as:

$$fib(n) \begin{cases} 0 & n=0 \\ 1 & n=1 \\ fib(n-2) + fib(n-1) & otherwise \end{cases}$$

# Sol:

```
#include <stdio.h>

int fibonacci(int n) {
    if (n == 0) {
        return 0;
    }
    else if (n == 1) {
        return 1;
    }
    else {
        return (fibonacci(n - 1) + fibonacci(n - 2));
    }
}

int main(void)
{
    for (int i = 0; i < 10; i++) {
        printf("%d ", fibonacci(i));
    }
}
```

# Library Functions

- *Library functions*: functions provided by the compiler
  - Standard Input/Output
  - Mathematical operations
  - String Processing
  - Time Processing
  - Error Handling
  - Retrieving and sorting data

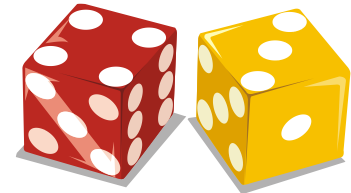


# Library Functions

함수	설명	사용예	반환 값
double sin(double x)	사인값 계산	sin(3.14/2.0)	1.0
double cos(double x)	코사인값 계산	cos(3.14/2.0)	0.0
double tan(double x)	탄젠트값 계산	tan(0.5)	0.546302
double exp(double x)	$e^x$	exp(10.0)	22026.5
double log(double x)	$\log_e x$	log(10.0)	2.30259
double log10(double x)	$\log_{10} x$	log10(100.0)	2.0
double ceil(double x)	x보다 작지 않은 가장 작은 정수	ceil(3.8)	4.0
double floor(double x)	x보다 크지 않은 가장 큰 정수	floor(3.8)	3.0
double fabs(double x)	x의 절대값	fabs(-3.67)	3.67
double pow(double x, double y)	$x^y$	pow(3.0,2.0)	9.0
double sqrt(double x)	$\sqrt{x}$	sqrt(4.0)	2.0

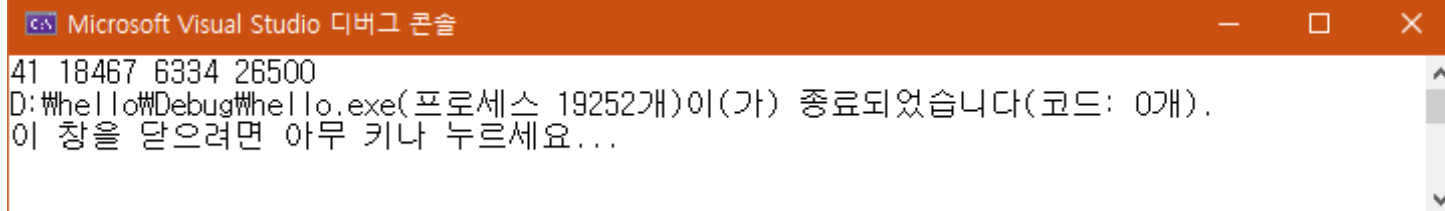
# Random Function

- A random number is a number that is randomly generated without an order.
- Random numbers are essential in cryptography, simulation, and games.
- rand()
  - Functions that generate random numbers
  - Generate random numbers from 0 to RAND\_MAX
- `number = rand();`



# Example

```
int main(void)
{
    printf("%d ", rand());
    printf("%d ", rand());
    printf("%d ", rand());
    printf("%d ", rand());
}
```

The image shows a screenshot of the Microsoft Visual Studio Debug Console window. The title bar is orange and contains the text "Microsoft Visual Studio 디버그 콘솔" along with standard window control buttons (minimize, maximize, close). The console area has a white background and displays the following text: "41 18467 6334 26500", "D:\helloworld\Debug\helloworld.exe( 프로세스 19252개)이(가) 종료되었습니다(코드: 0개).", and "이 창을 닫으려면 아무 키나 누르세요...". A vertical scrollbar is visible on the right side of the console area.

Microsoft Visual Studio 디버그 콘솔

41 18467 6334 26500  
D:\helloworld\Debug\helloworld.exe( 프로세스 19252개)이(가) 종료되었습니다(코드: 0개).  
이 창을 닫으려면 아무 키나 누르세요...



# To limit the value of a random number to a certain range

전체적으로는 1에서 45까지의 값이 생성된다.

$1 + ( \text{rand()} \% 45 )$

0에서 44까지의 값이 생성된다.

# Random number seed

- How do we create different random numbers each time we run a program? To generate random numbers differently each time, we use the concept of seed.
- Seed is a reference value at random number generation.

난수의 시드를 설정한다.

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

현재 시간을 얻는다.

# To make it different each time to run

- To generate random numbers differently each time, the seed must be different.

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

```
#define MAX 45
int main( void )
```

```
{
```

```
    int i;
```

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

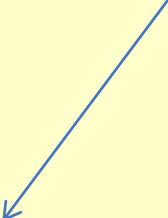
```
    for( i = 0; i < 6; i++ )
```

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

```
    return 0;
```

```
}
```

The most common way to set a seed is to use the current time as a seed. This is because the current time varies from time to time.



# Lab: Create Lotto Number

- Random number between 1 and 45
- Let's prevent duplication.




# Sol:

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

#define MAX 45

int main(void)
{
    int i, k, lotto[6] = { 0 };
    int dup_check[MAX + 1] = { 0 };


    srand(time(NULL));
    for (i = 0; i < 6; i++)
    {
        k = 1 + (rand() % MAX);
        while (dup_check[k] == 1)
            k = 1 + (rand() % MAX);
        lotto[i] = k;
        dup_check[k] = 1;
        printf("%d ", lotto[i]);
    }
    return 0;
}
```



# Lab: calculate the Taylor series

지수 함수  $e^x$  에 대한 테일러 급수는 다음과 같다.

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = \frac{x^0}{0!} + \frac{x^1}{1!} + \frac{x^2}{2!} + \dots$$

 Microsoft Visual Studio 디버그 콘솔

```
x와 n의 값을 입력하시오: 2 100  
e^2 = 7.389
```

# Sol:

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

```
double power(int x, int y) {
    double result = 1.0;
    for (int i = 0; i < y; i++)
        result *= x;
    return result;
}
```

```
double factorial(int n) {
    double result = 1.0;
    if (n <= 1) return 1;
    for (int i = 1; i <= n; i++)
        result *= i;
    return result;
}
```

# Sol:

```
int main(void) {  
    double sum = 0.0;  
    int x, n;  
    printf("x와 n의 값을 입력하시오: ");  
    scanf("%d %d", &x, &n);  
    for (int i = 0; i <= n; i++)  
        sum += power(x, i) / factorial(i);  
  
    printf("e^%d = %.3lf\n", x, sum);  
    return 0;  
}
```



# Q & A

