


제 8장 함수

이번 장에서 학습할 내용

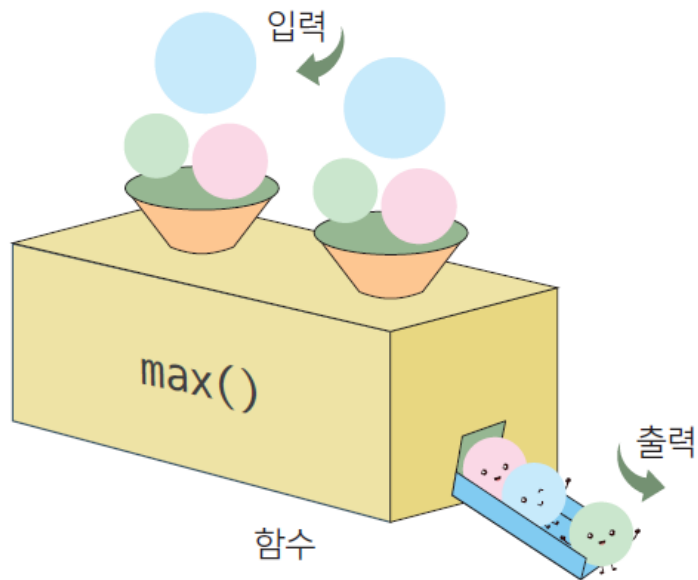
- 
- 모듈화
 - 함수의 개념, 역할
 - 함수 작성 방법
 - 반환값
 - 인수 전달
 - 함수를 사용하는 이유

규모가 큰
프로그램은 전체
문제를 보다
단순하고 이해하기
쉬운 함수로
나누어서
프로그램을
작성하여야 한다.



함수의 개념

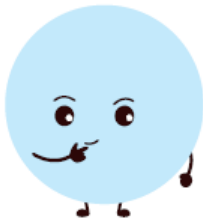
- 함수(function): 입력을 받아서 특정한 작업을 수행하여서 결과를 반환하는 블랙 박스(상자)와 같다



함수가 필요한 이유

- 동일한 코드가 여러 곳에서 사용된다고 하자.

비슷한 코드인데 하나로
합칠 수 있을까?



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

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

함수가 필요한 이유

- 함수를 작성하면 동일한 코드를 하나로 만들 수 있다.

함수를 사용하면 됩니다!



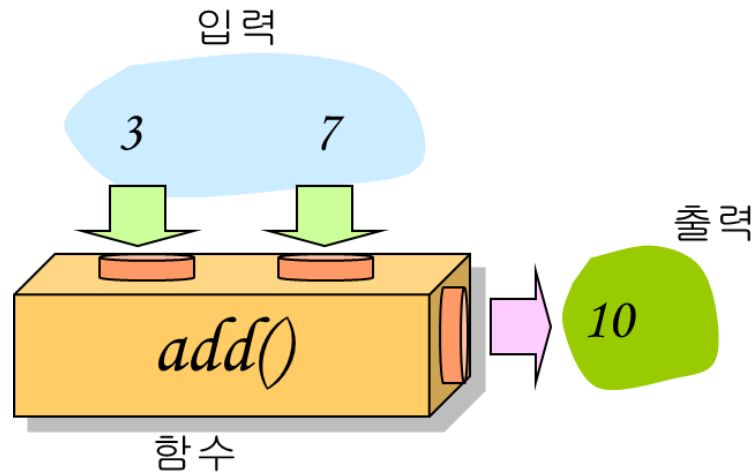
```
print_stars();
```

```
print_stars();
```

```
void print_stars()  
{  
    for(int i=0; i<30; i++)  
        printf("*");  
}
```

함수의 특징

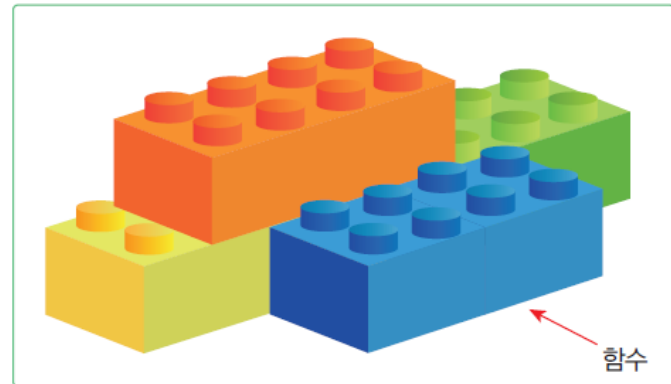
- 함수는 특정한 작업을 수행하기 위한 명령어들의 모음이다.
- 함수는 서로 구별되는 이름을 가지고 있다.
- 함수는 특정한 작업을 수행한다.
- 함수는 입력을 받을 수 있고 결과를 반환할 수 있다.



함수의 장점

- 함수를 사용하면 코드가 중복되는 것을 막을 수 있다.
- 한번 작성된 함수는 여러 번 재사용할 수 있다.
- 함수를 사용하면 전체 프로그램을 모듈로 나눌 수 있어서 개발 과정이 쉬워지고 보다 체계적이 되면서 유지보수도 쉬워진다.

프로그램



함수의 종류

함수 { 사용자 정의 함수
라이브러리 함수



내가 원하는 함수가 없으니
직접 만들어야지!



이 함수들은 기본적으로
제공됩니다.

함수의 정의

Syntax

함수 정의

예

반환형 함수 이름 매개 변수(현재는 없다)

```
void print_stars()  
{  
    for(int i=0; i<30; i++)  
        printf("*");  
}
```

함수 몸체

함수 정의

- 반환형

- 반환형은 함수가 처리를 종료한 후에 호출한 곳으로 반환하는 데이터의 유형을 말한다.

- 함수 이름

- 함수 이름은 식별자에 대한 규칙만 따른다면 어떤 이름이라도 가능하다.
- 함수의 기능을 암시하는 (동사+명사)를 사용하면 좋다.

```
int square()           // 정수를 제공하는 함수
double compute_average() // 평균을 구하는 함수
void set_cursor_type()  // 커서의 타입을 설정하는 함수
```

함수 이름

Tip: 함수의 길이

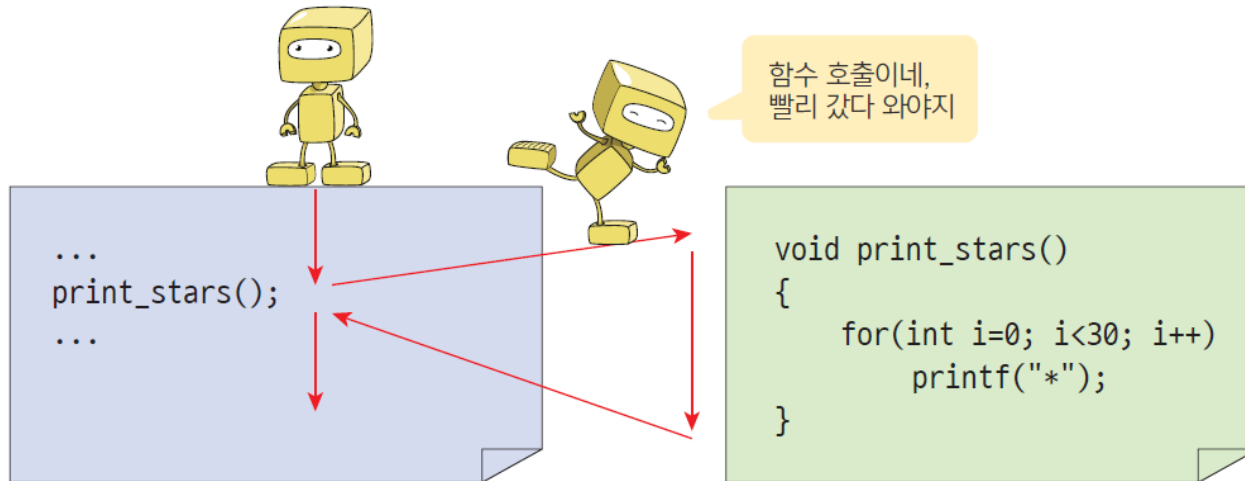


함수의 길이에 제한이 있을까?

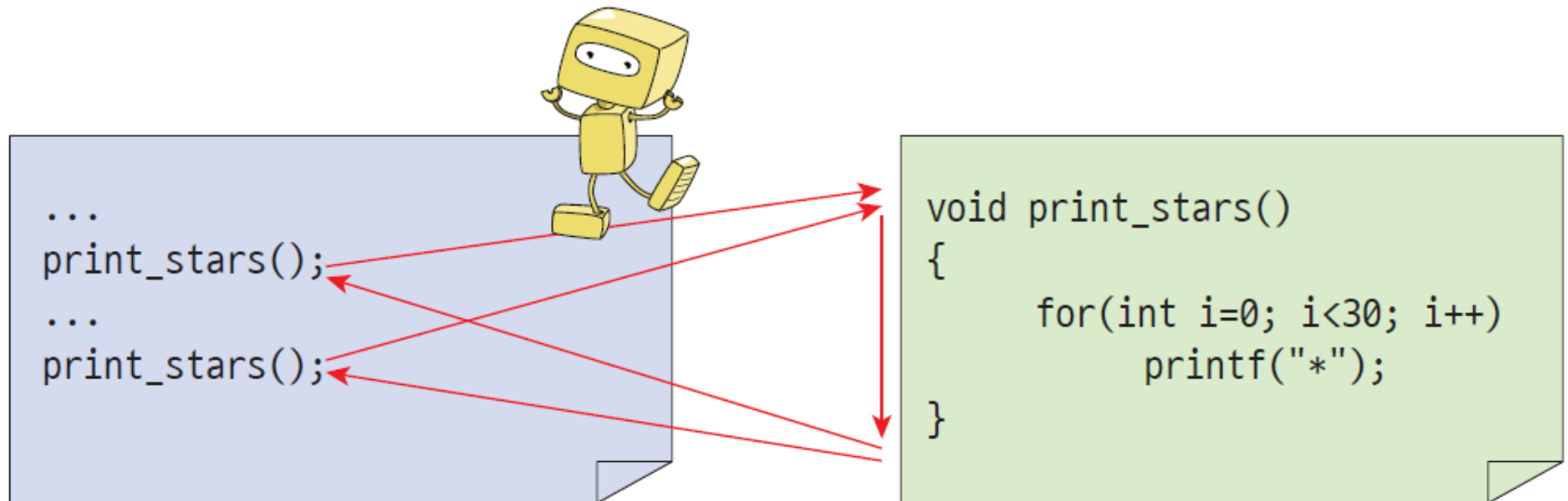
C에서는 함수의 길이에 아무런 제한을 두지 않는다. 이것은 여러분들이 하나의 함수 안에 많은 문장들을 넣을 수도 있음을 의미한다. 그러나 함수의 길이가 지나치게 길어지면 좋지 않다. 기본적으로 하나의 함수는 하나의 작업만을 수행하여야 한다. 만약 함수의 길이가 지나치게 길어진다면 하나 이상의 작업을 하고 있다고 봐야한다. 따라서 이때는 함수를 분할하여 하는 편이 낫다. 그렇다면 어느 정도의 길이가 적당할까? 물론 절대적인 기준은 없지만 30행을 넘지 않도록 하는 것이 좋다.

함수 호출

- 함수 호출(function call)이란 `print_stars()`와 같이 함수의 이름을 써주는 것이다.
- 함수 안의 문장들은 호출되기 전까지는 전혀 실행되지 않는다. 함수를 호출하게 되면 현재 실행하고 있는 코드는 잠시 중단되고, 호출된 함수로 이동하여서 함수 몸체 안의 문장들이 순차적으로 실행된다.
- 호출된 함수의 실행이 끝나며 호출한 위치로 되돌아가서 잠시 중단도

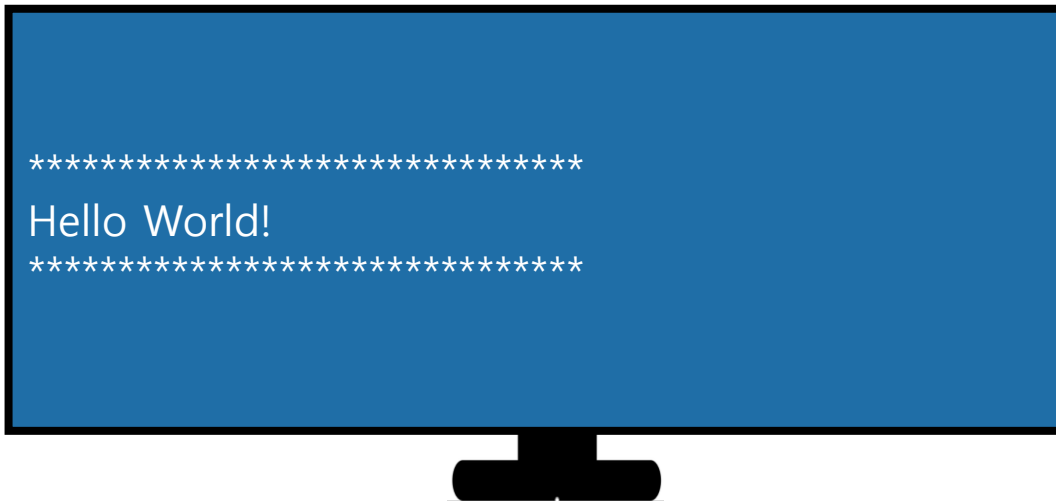


함수는 여러 번 호출될 수 있다.



예제

- `print_stars()` 함수를 2번 호출하여서 다음과 같이 출력하는 프로그램을 작성해보자.



예제

```
#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;
```

```
}
```

함수 호출

함수 호출

매개 변수와 반환값

Syntax

함수의 구조

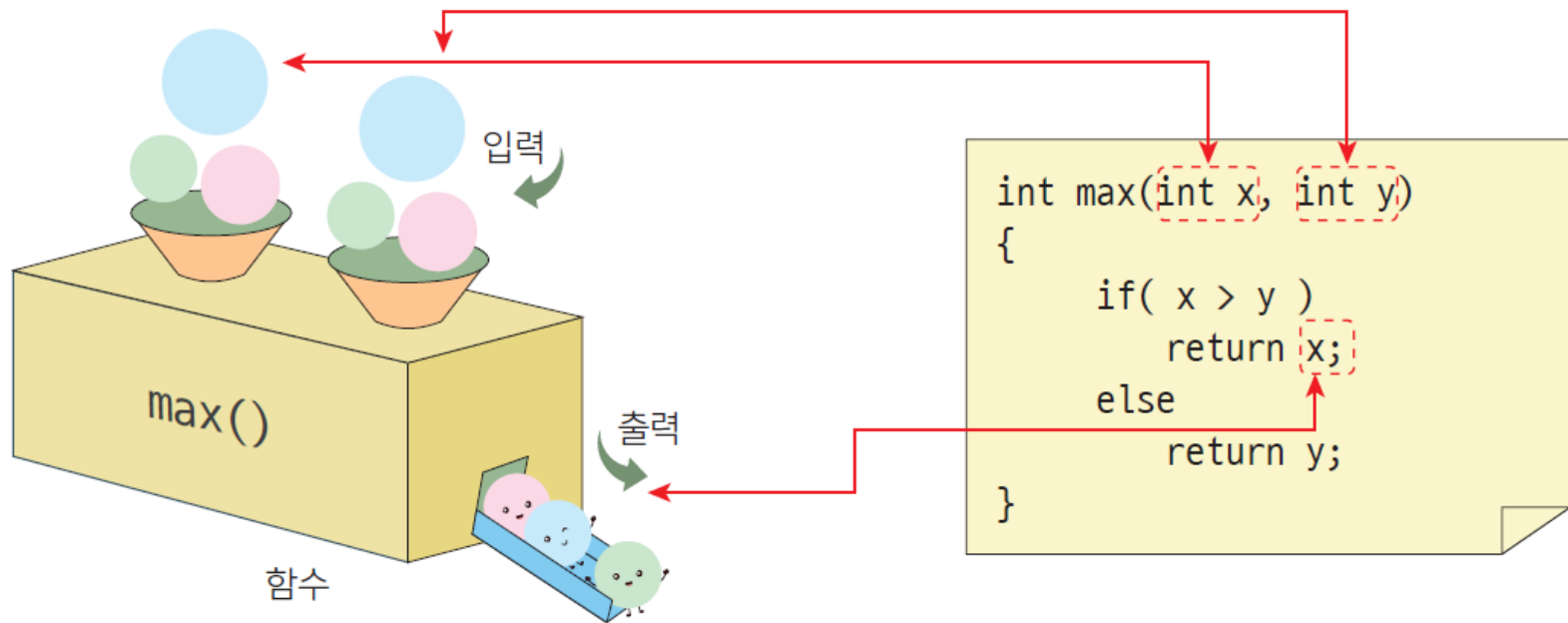
예

반환형 함수 이름 매개 변수

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

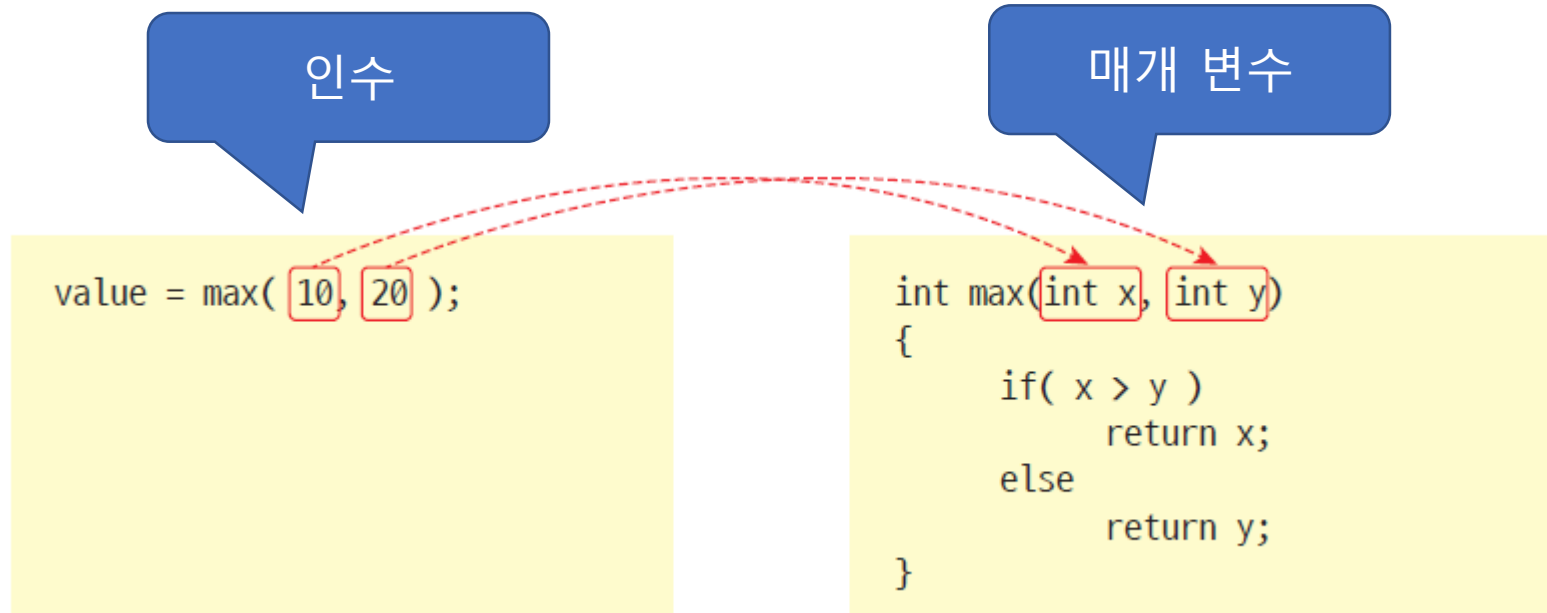
함수 몸체

매개 변수와 반환값



인수와 매개 변수

- 인수(argument)는 호출 프로그램에 의하여 함수에 실제로 전달되는 값이다.
- 매개 변수(parameter)는 이 값을 전달받는 변수이다.



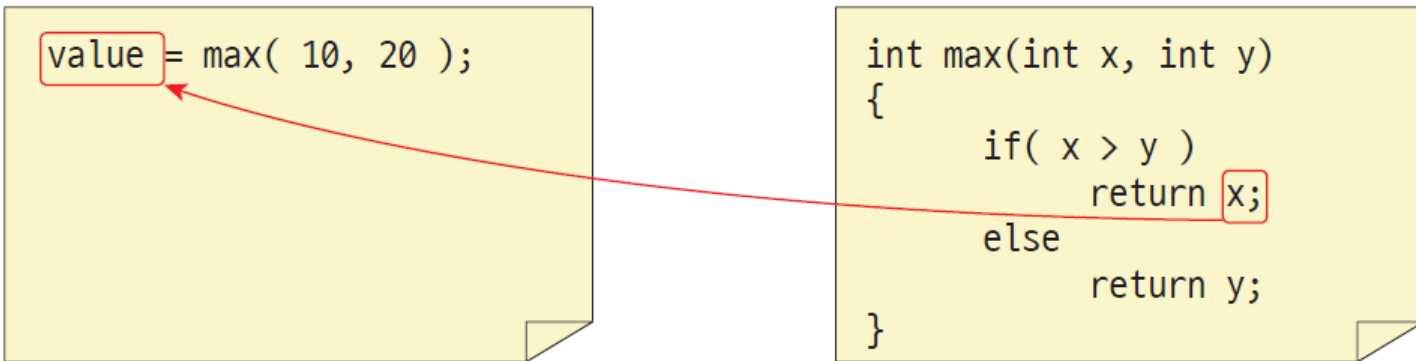
인수와 매개 변수

- 만약 매개 변수가 없는 경우에는 `print_stars(void)`와 같이 매개 변수 위치에 `void`를 써주거나 `print_stars()`와 같이 아무 것도 적지 않으면 된다.
- 함수가 호출될 때마다 인수는 달라질 수 있다.
- 매개 변수의 개수는 정확히 일치하여야 한다는 점이다. 매개 변수의 개수와 인수의 개수가 일치하지 않으면 아주 찾기 어려운 오류가 발생하게 된다.

```
max(10);      // max() 함수를 호출할 때는 인수가 두개이어야 한다.  
max( );      // max() 함수를 호출할 때는 인수가 두개이어야 한다.
```


반환값

- 반환값(return value)은 함수가 호출한 곳으로 반환하는 작업의 결과값이다.
- 값을 반환하려면 **return** 문장 다음에 수식을 써주면 수식의 값이 반환된다.
- 인수는 여러 개가 있을 수 있으나 반환값은 하나만 가능하다.



예제

- 위에서 작성한 `max()` 함수를 호출하여서 사용자가 입력한 값 중에서 더 큰 값을 찾아보자.

A computer monitor with a black frame and a black stand. The screen is blue and displays white text.

정수 2개를 입력하시오: 10 20
더 큰 값은 20입니다.

예제

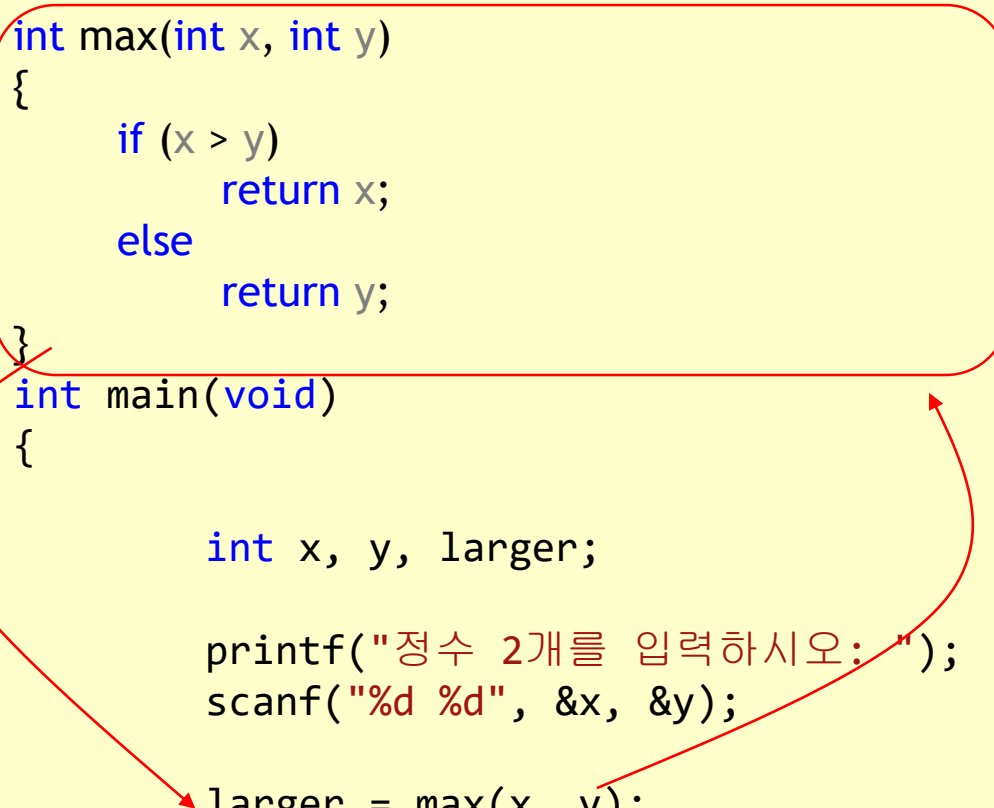
```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

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

int main(void)
{
    int x, y, larger;

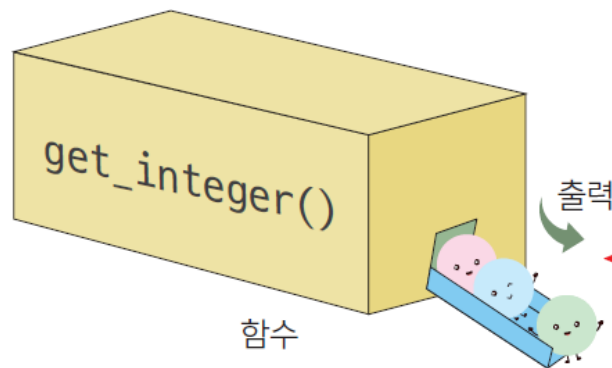
    printf("정수 2개를 입력하시오: ");
    scanf("%d %d", &x, &y);

    larger = max(x, y);
    printf("더 큰 값은 %d입니다. \n", larger);
    return 0;
}
```

A red rounded rectangle highlights the `max` function definition. A red arrow originates from the `max(x, y)` call inside the `main` function and points to the `max` function definition. Another red arrow points from the `scanf` call to the `max` function definition.

Lab: 정수를 입력받는 get_integer() 함수

- 입력 안내 메시지를 출력하고 정수를 입력받아서 우리에게 반환해주는 함수 get_integer()를 작성해보자.



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

get_integer() 함수

```
// 사용자로부터 정수를 받는 함수
#include <stdio.h>

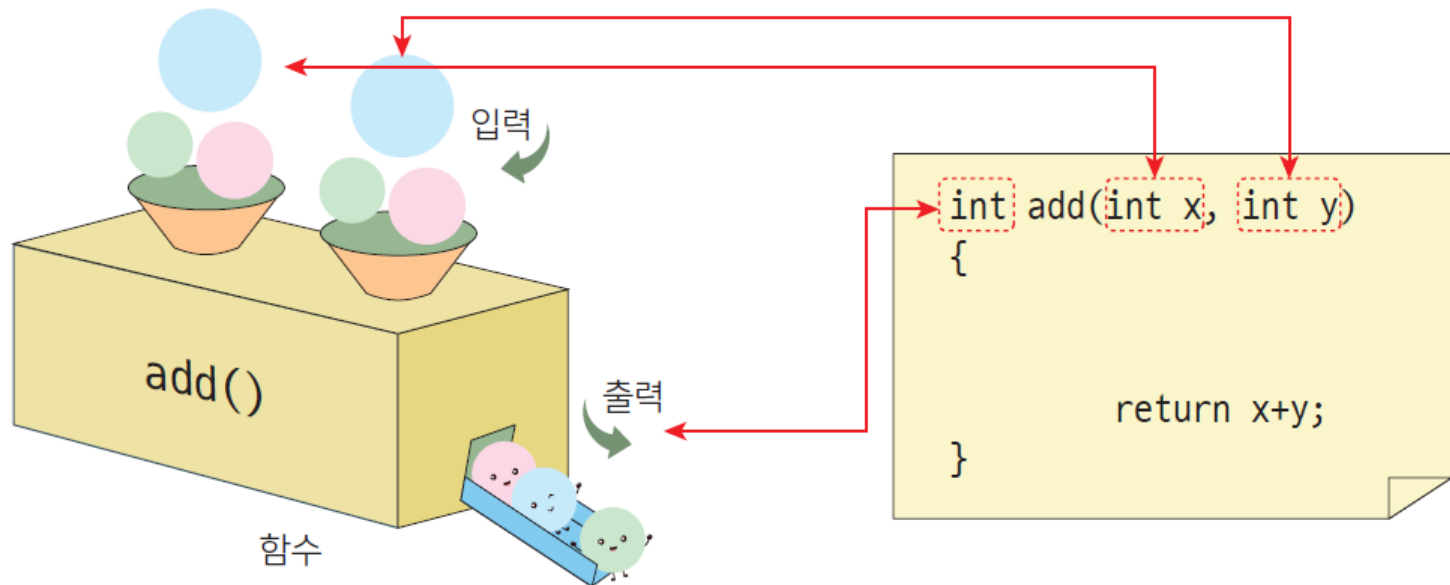
int get_integer(void)
{
    int value;

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

    return value;
}
```

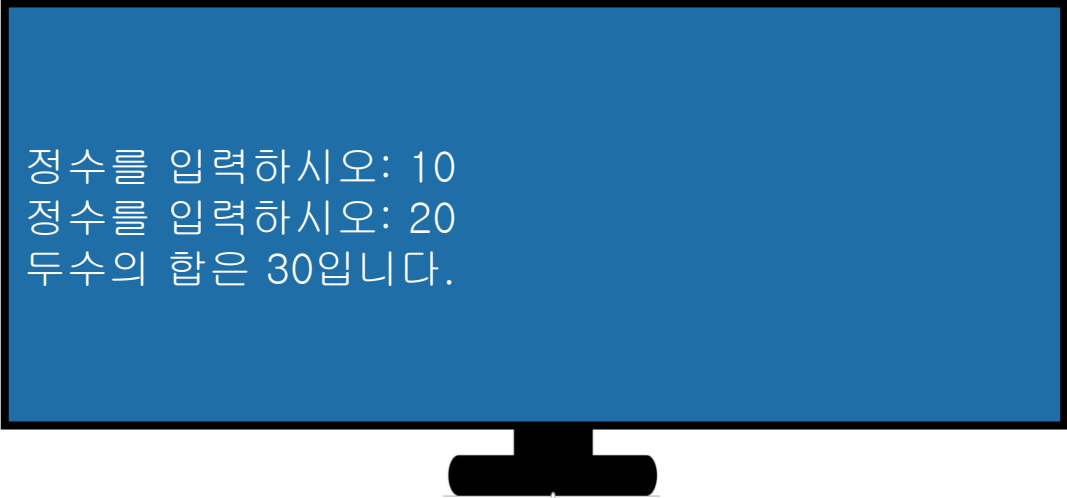

Lab: 정수의 합을 계산하는 add() 함수

- 두 개의 정수를 받아서 합을 계산하는 함수를 만들어보자. 함수 이름부터 결정하여야 한다.



Lab: 정수의 합을 계산하는 프로그램

- 앞에서 작성한 `get_integer()`까지 사용하여서 사용자로부터 받은 정수의 합을 계산하여 출력하자.



정수를 입력하시오: 10
정수를 입력하시오: 20
두수의 합은 30입니다.

```
#include <stdio.h>
//
int get_integer()
{
    int value;
    printf("정수를 입력하시오: ");
    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();

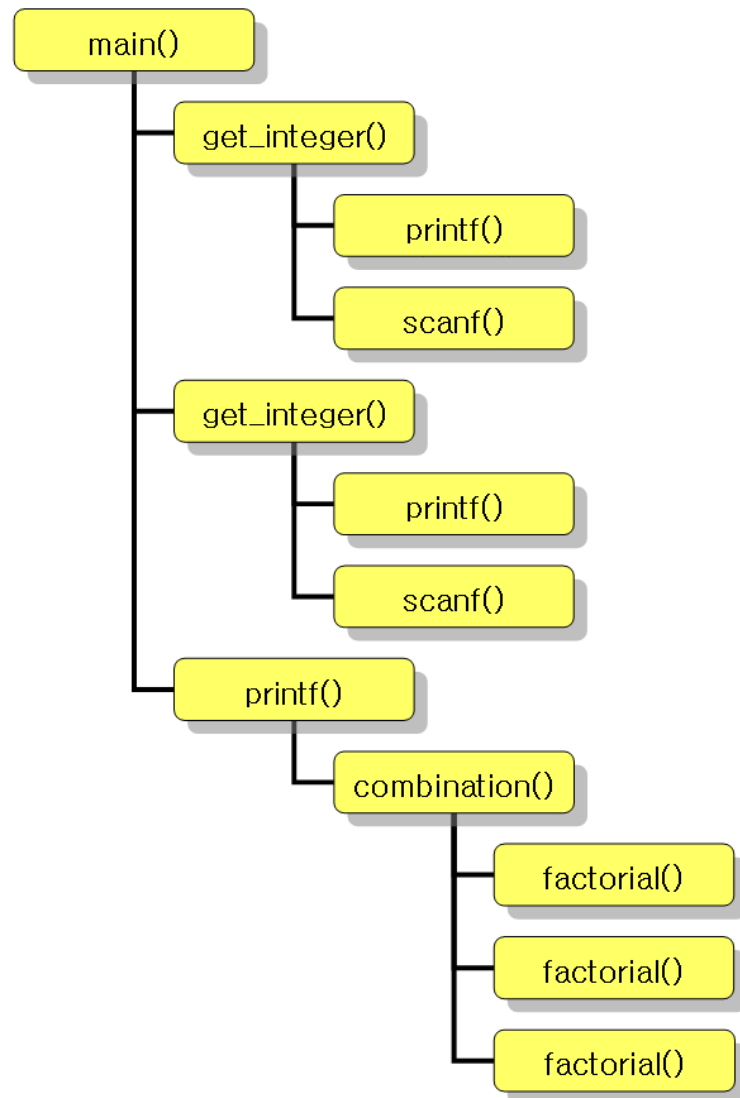
    int sum = add(x, y);
    printf("두수의 합은 %d입니다. \n", sum);
    return 0;
}
```

조합(combination) 계산 함수

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

- 팩토리얼 계산 함수와 get_integer() 함수를 호출하여 조합을 계산한다

정수를 입력하시오: 10
정수를 입력하시오: 3
C(10, 3) = 120



예제

```
// 수학적 조합 값을 구하는 예제
#include <stdio.h>

// 팩토리얼 값을 반환
int factorial(int n)
{
    int i, result = 1;

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

// 팩토리얼 값을 이용하여 조합값을 계산
int combination(int n, int r)
{
    return (factorial(n)/(factorial(r) * factorial(n-r)));
}
```

```
// 사용자로부터 값을 입력받아서 반환
```

```
int get_integer(void)
```

```
{
```

```
    int n;
```

```
    printf("정수를 입력하시오: ");
```

```
    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;
```

```
}
```

함수 원형

- 아래의 코드를 컴파일하면 오류가 발생한다.

```
#include <stdio.h>

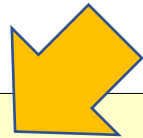
int main(void)
{
    printf("섭씨 %lf도는 화씨 %lf입니다. \n", 36.0, c_to_f(36.0));
    return 0;
}

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		

함수 원형

- 함수 원형(*function prototyping*): 컴파일러에게 함수에 대하여 미리 알리는 것



```
#include <stdio.h>
double c_to_f(double c_temp); // 함수 원형

int main(void)
{
    printf("섭씨 %lf도는 화씨 %lf입니다. \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;
}
```


함수 원형

- 함수 원형은 함수의 이름, 매개변수, 반환형을 함수가 정의되기 전에 미리 알려주는 것이다.
- 함수 원형은 함수 헤더에 세미콜론(;)만을 추가한 것과 똑같다. 다만 함수 원형에서는 매개 변수의 이름은 적지 않아도 된다. 매개 변수의 자료형만 적으면 된다.


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

매개 변수의 이름은 생략하여도 된다.
반드시 끝에 ;을 붙여야 한다.

함수 원형을 사용하지 않는 예제

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

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



함수 정의가 함수 호출보다 먼저 오면 함수 원형을 정의하지 않아도 된다.

그러나 일반적인 방법은 아니다.

함수 원형을 사용하지 않는 예제

```
#include <stdio.h>
double sub1(double d)
{
    sub2(100.0);
}
double sub2(double d)
{
    sub1(20.0);
}
int main(void)
{
    return 0;
}
```

이런 경우에는 원형 말고는 방법이 없음

오류 목록						
전체 솔루션		1 오류	1 경고	0 메시지	빌드 + IntelliSense	검색 오류 목록
코드	설명	프로젝트	파일	줄	비표시 오류(Suppr...)	
C4013	'sub2'이(가) 정의되지 않았습니다. extern은 int형을 반환하는 것으로 간주합니다.	Project14	소스.c	4		
C2371	'sub2': 재정의. 기본 형식이 다릅니다.	Project14	소스.c	6		

참고

★ 참고사항

링크 오류

소스 파일을 컴파일하여서 실행 파일을 만들다 보면 링크 오류가 발생하는 경우가 있다. 링크 오류는 컴파일러가 함수를 찾지 못했을 때 발생한다. 즉 프로그래머가 정의되지 않은 함수를 사용하였을 때 링크 오류가 발생한다. 링크 오류가 발생하면 함수의 이름이 오류 메시지로 표시된다. 따라서 오류 메시지를 보고 함수가 확실히 정의되었는지를 확인하여야 한다.

```
1>c:\users\chun\documents\visual studio 2010\projects\hello\hello\hello.c(4): warning C4013:  
'sub'이(가) 정의되지 않았습니니다. extern은 int형을 반환하는 것으로 간주합니다.
```

```
1>hello.obj : error LNK2001: _sub 외부 기호를 확인할 수 없습니다.
```

```
1>C:\Users\chun\documents\visual studio 2010\Projects\hello\Debug\hello.exe : fatal error  
LNK1120: 1개의 확인할 수 없는 외부 참조입니다.
```

```
===== 빌드: 성공 0, 실패 1, 최신 0, 생략 0 =====
```

링크 오류

라이브러리 함수

- 라이브러리 함수(library function): 컴파일러에서 제공하는 함수
 - 표준 입출력
 - 수학 연산
 - 문자열 처리
 - 시간 처리
 - 오류 처리
 - 데이터 검색과 정렬



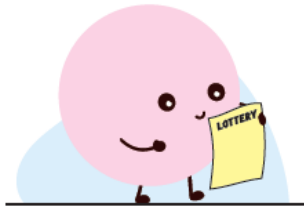
난수 함수

- 난수(**random number**)는 규칙성이 없이 임의로 생성되는 수이다.
- 난수는 암호학이나 시뮬레이션, 게임 등에서 필수적이다.
- rand()
 - 난수를 생성하는 함수
 - 0부터 RAND_MAX까지의 난수를 생성



예제: 로또 번호 생성하기

- 하나의 예제로 로또 번호를 생성하는 프로그램을 작성하여보자.
로또 번호는 1부터 45까지의 숫자 6개로 이루어진다.



실습 코드

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

    return 0;
}
```

0에서 32767 사이의 정수로 생성



41 18467 6334 26500 19169 15724

1부터 45 사이로 제한

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



- 하지만 실행할 때마다 항상 똑같은 난수가 발생된다.

실행할 때마다 다르게 하려면

- 매번 난수를 다르게 생성하려면 시드(seed)를 다르게 하여야 한다.
 - 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 );
```

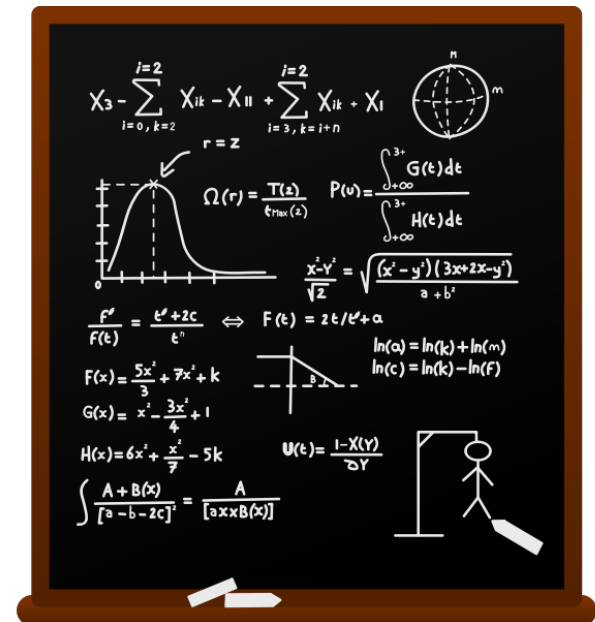
```
    return 0;
```

```
}
```

시드를 설정하는 가장 일반적인 방법은 현재의 시각을 시드로 사용하는 것이다. 현재 시각은 실행할 때마다 달라지기 때문이다.

표준 라이브러리 함수(수학 함수)

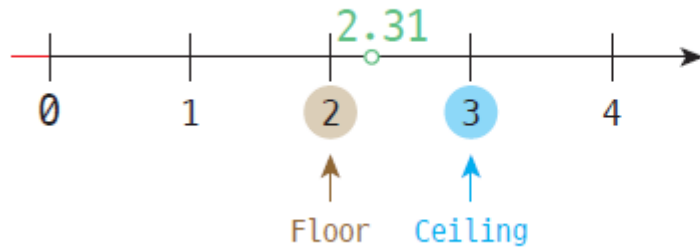
- 이번 장에서는 수치 계산을 하는 라이브러리 함수에 대하여 살펴본다. 이들 라이브러리 함수를 사용하면 복잡한 산술 연산을 할 수 있다.
- 수학 함수들에 대한 원형은 헤더파일 math.h에 있다. 수학 함수는 일반적으로 double형의 매개 변수와 반환값을 가진다.



수학 라이브러리 함수

분류	함수	설명
삼각함수	<code>double sin(double x)</code>	사인값 계산
	<code>double cos(double x)</code>	코사인값 계산
	<code>double tan(double x)</code>	탄젠트값 계산
역삼각함수	<code>double acos(double x)</code>	역코사인값 계산 결과값 범위 $[0, \pi]$
	<code>double asin(double x)</code>	역사인값 계산 결과값 범위 $[-\pi/2, \pi]$
	<code>double atan(double x)</code>	역탄젠트값 계산 결과값 범위 $[-\pi/2, \pi]$
쌍곡선함수	<code>double cosh(double x)</code>	쌍곡선 코사인
	<code>double sinh(double x)</code>	쌍곡선 사인
	<code>double tanh(double x)</code>	쌍곡선 탄젠트
지수함수	<code>double exp(double x)</code>	e^x
	<code>double log(double x)</code>	$\log_e x$
	<code>double log10(double x)</code>	$\log_{10} x$
기타함수	<code>double ceil(double x)</code>	x보다 작지 않은 가장 작은 정수
	<code>double floor(double x)</code>	x보다 크지 않은 가장 큰 정수
	<code>double fabs(double x)</code>	실수 x의 절대값
	<code>int abs(int x)</code>	정수 x의 절대값
	<code>double pow(double x, double y)</code>	x^y
	<code>double sqrt(double x)</code>	\sqrt{x}

floor()와 ceil() 함수



$\lfloor x \rfloor$
floor(x)

$\lceil x \rceil$
ceil(x)

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

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

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

fabs()

- fabs()는 실수를 받아서 절대값을 반환한다.

```
printf("12.0의 절대값은 %f\n", fabs(12.0));  
printf("-12.0의 절대값은 %f\n", fabs(-12.0));
```

pow()와 sqrt()

```
printf("10의 3승은 %.0f.\n", pow(10.0, 3.0));  
printf("16의 제곱근은 %.0f.\n", sqrt(64));
```



*10의 3승은 1000.
16의 제곱근은 4.*

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

// 삼각 함수 라이브러리

#include <math.h>

#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);

}

여러 수학 함수들을 포함하는 표준
라이브러리

sin(1.570796) = 1.000000
cos(1.570796) = 0.000000

기타 함수

함수	설명
<code>exit()</code>	<code>exit()</code> 를 호출하면, 실행 중인 프로그램을 종료시킨다.
<code>system("...")</code>	<code>system()</code> 은 운영 체제의 명령 프롬프트에게 명령어를 전달하여서 실행시키는 함수이다. 예를 들어서 DOS 명령어인 DIR이나 COPY, TYPE, CLS, DEL, MKDIR와 같은 명령어들을 실행시킬 수 있다.
<code>time(NULL)</code>	현재 시각을 반환한다. 1970년 1월 1일부터 흘러온 초를 반환한다.

예제

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

int main(void)
{
    system("dir");
    printf("아무 키나 치세요\n");
    _getch();
    system("cls");

    return 0;
}
```

C 드라이브의 볼륨에는 이름이 없습니다.
볼륨 일련 번호: C870-52ED
C:\Users\kim\source\repos\hello\hello 디렉터리
2022-10-16 오전 08:41 <DIR> .
2022-10-16 오전 08:41 <DIR> ..
2022-10-16 오전 08:41 160 hello.c
2022-10-16 오전 07:25 6,618 hello.vcxproj

함수를 사용하는 이유

- 소스 코드의 중복성을 없애준다.
- 한번 제작된 함수는 다른 프로그램을 제작할 때도 사용이 가능하다.
- 복잡한 문제를 단순한 부분으로 분해할 수 있다.

복잡한 프로그램은 함수로 분리한다.

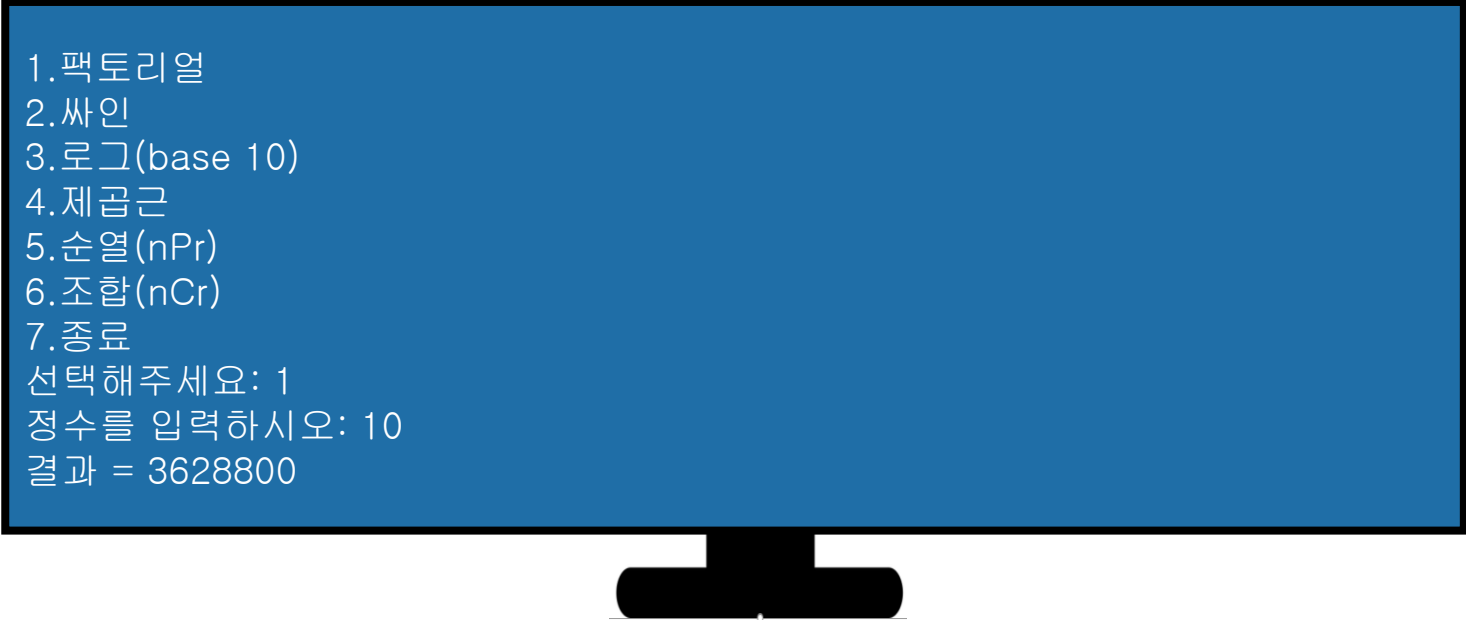
```
int main(void)
{
    // 숫자들의 리스트를 키보드에서 읽어들이는 코드
    ...
    // 숫자들을 크기순으로 정렬하는 코드
    ...
    // 정렬된 숫자들의 리스트를 화면에 출력하는 코드
    ...
}
```



```
int main(void)
{
    ...
    read_list(); // 숫자들의 리스트를 키보드에서 읽어 들이는 함수
    sort_list(); // 숫자들의 리스트를 크기순으로 정렬하는 함수
    print_list(); // 숫자들의 리스트를 화면에 출력하는 함수
    ...
}
```

Mini Project: 공학용 계산기 프로그램 작성

- 이번 장에서 학습한 함수들을 이용하여 싸인값이나 코싸인값을 계산할 수 있는 공학용 계산기를 만들어보자. 아직 구현 안 된 기능은 도전 문제에서 추가해보자.



1.팩토리얼
2.싸인
3.로그(base 10)
4.제곱근
5.순열(nPr)
6.조합(nCr)
7.종료
선택해주세요: 1
정수를 입력하시오: 10
결과 = 3628800

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

int menu(void)
{
    int n;
    printf("1.팩토리얼\n");
    printf("2.싸인\n");
    printf("3.로그(base 10)\n");
    printf("4.제곱근\n");
    printf("5.순열(nPr)\n");
    printf("6.조합(nCr)\n");
    printf("7.종료\n");
    printf("선택해주세요: ");
    scanf("%d", &n);
    return n;
}
```

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

```
void sine()
{
    double a, result;
    printf("각도를 입력하시오: ");
    scanf("%lf", &a);
    result = sin(a);
    printf("결과 = %lf\n\n", result);
}
```

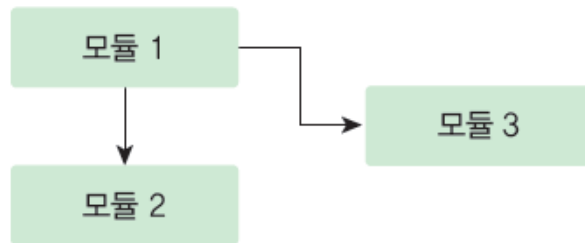
```
void logBase10()
{
    double a, result;
    printf("실수값을 입력하시오: ");
    scanf("%lf", &a);
    if (a <= 0.0)
        printf("오류\n");
    else
    {
        result = log10(a);
        printf("결과 = %lf\n\n", result);
    }
}
```



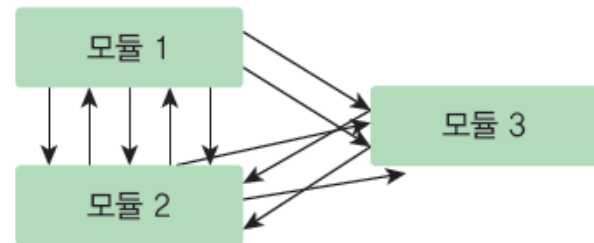
```
int main(void)
{
    while (1) {
        switch (menu()) {
            case 1:
                factorial();
                break;
            case 2:
                sine();
                break;
            case 3:
                logBase10();
                break;
            case 7:
      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

모듈화

- 모듈 내에서는 최대의 상호 작용이 있어야 하고 모듈 사이에는 최소의 상호 작용만 존재하여야 한다. 만약 모듈과 모듈 사이의 연결이 복잡하다면 모듈화가 잘못된 것이다.



(a) 좋은 모듈화



(b) 나쁜 모듈화

Q & A

