



인하공업전문대학
INHA TECHNICAL COLLEGE

C 프로그래밍

10주차

인하공업전문대학 컴퓨터 정보과
김한결 강사

이번 장에서 학습할 내용



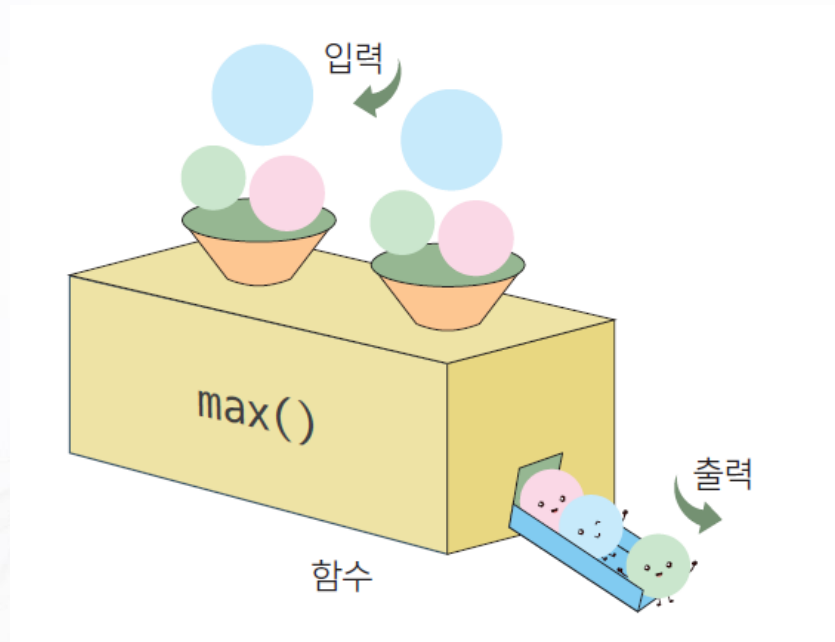
- 모듈화
- 함수의 개념, 역할
- 함수 작성 방법
- 반환값
- 인수 전달
- 함수를 사용하는 이유

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



함수의 개념

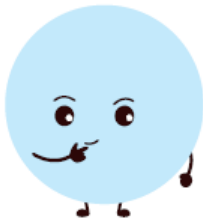
- 함수(function): 입력을 받아서 특정한 작업을 수행하여서 결과를 반환하는 블랙 박스(상자)와 같다



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!



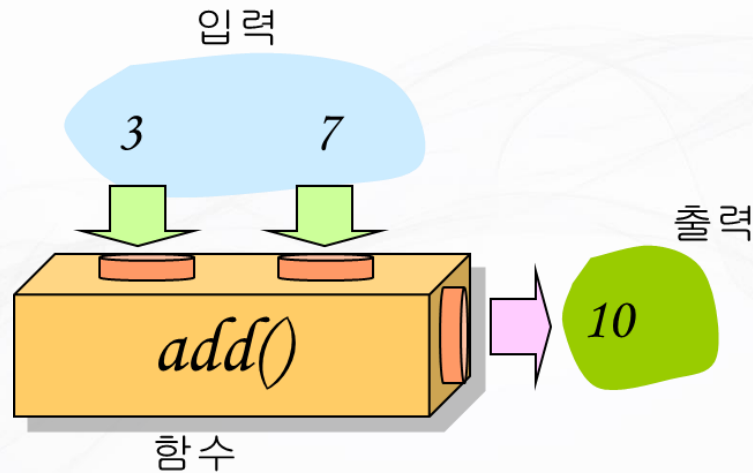
```
print_stars();
```

```
print_stars();
```

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

함수의 특징

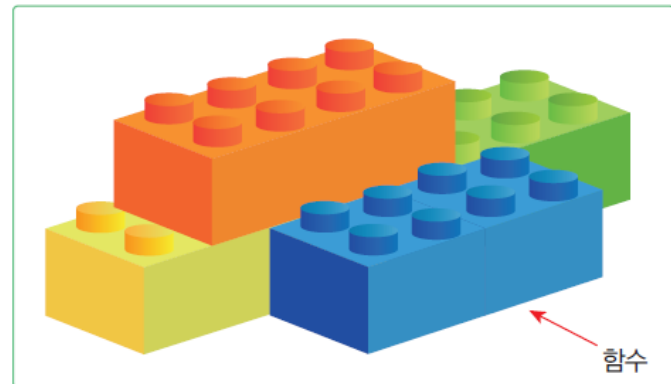
- 함수는 특정한 작업을 수행하기 위한 명령어들의 모음이다.
- 함수는 서로 구별되는 이름을 가지고 있다.
- 함수는 특정한 작업을 수행한다.
- 함수는 입력을 받을 수 있고 결과를 반환할 수 있다.



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



중간 점검

1. 함수가 필요한 이유는 무엇인가?
2. 함수와 프로그램의 관계는?
3. 컴파일러에서 지원되는 함수를 _____ 함수라고 한다.



함수의 정의

Syntax

함수 정의

예

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

반환형 함수 이름 매개 변수(현재는 없다)

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()  
double compute_average()  
void set_cursor_type()
```

// 정수를 제공하는 함수
// 평균을 구하는 함수
// 커서의 타입을 설정하는 함수

함수 이름

Tip: 함수의 길이

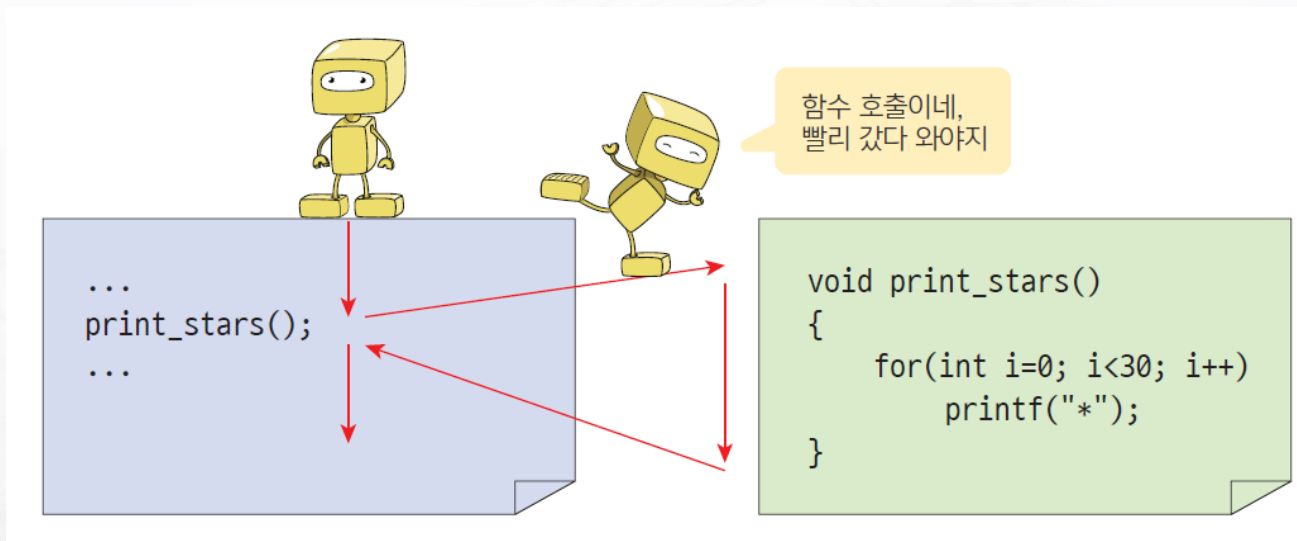


함수의 길이에는 제한이 있을까?

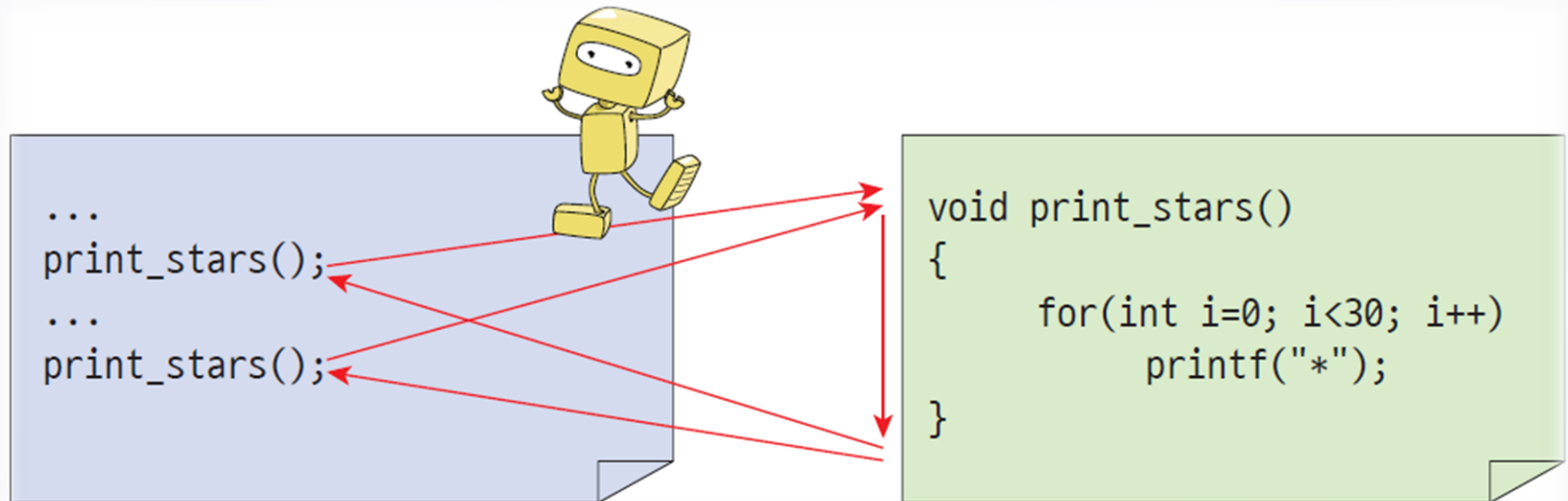
C에서는 함수의 길이에 아무런 제한을 두지 않는다. 이것은 여러분들이 하나의 함수 안에 많은 문장들을 넣을 수도 있음을 의미한다. 그러나 함수의 길이가 지나치게 길어지면 좋지 않다. 기본적으로 하나의 함수는 하나의 작업만을 수행하여야 한다. 만약 함수의 길이가 지나치게 길어진다면 하나 이상의 작업을 하고 있다고 봐야한다. 따라서 이때는 함수를 분할하여 하는 편이 낫다. 그렇다면 어느 정도의 길이가 적당할까? 물론 절대적인 기준은 없지만 30행을 넘지 않도록 하는 것이 좋다.

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 끝나면 호출한 위치로 되돌아가서 잠시 중단되었던 코드가 실행을 재개한다.

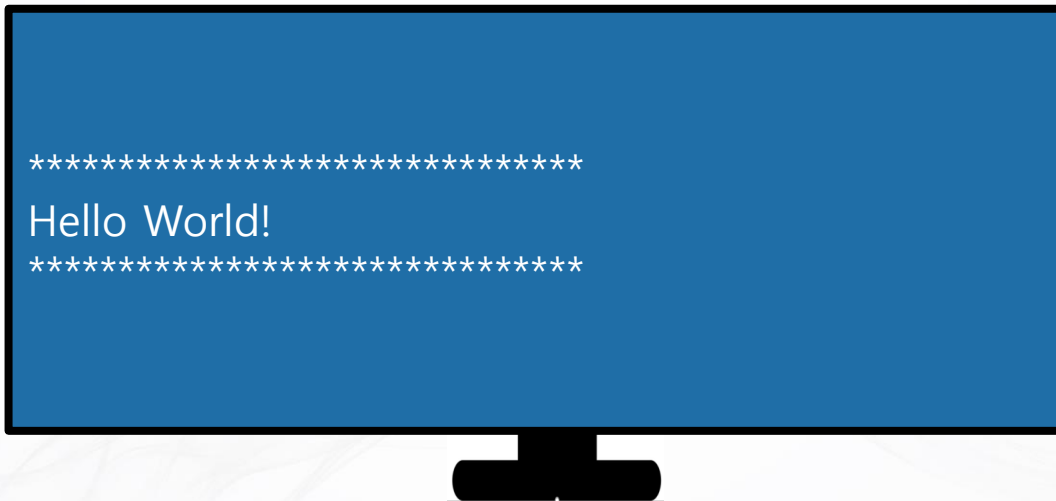


함수는 여러 번 호출될 수 있다.



예제

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



```
#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

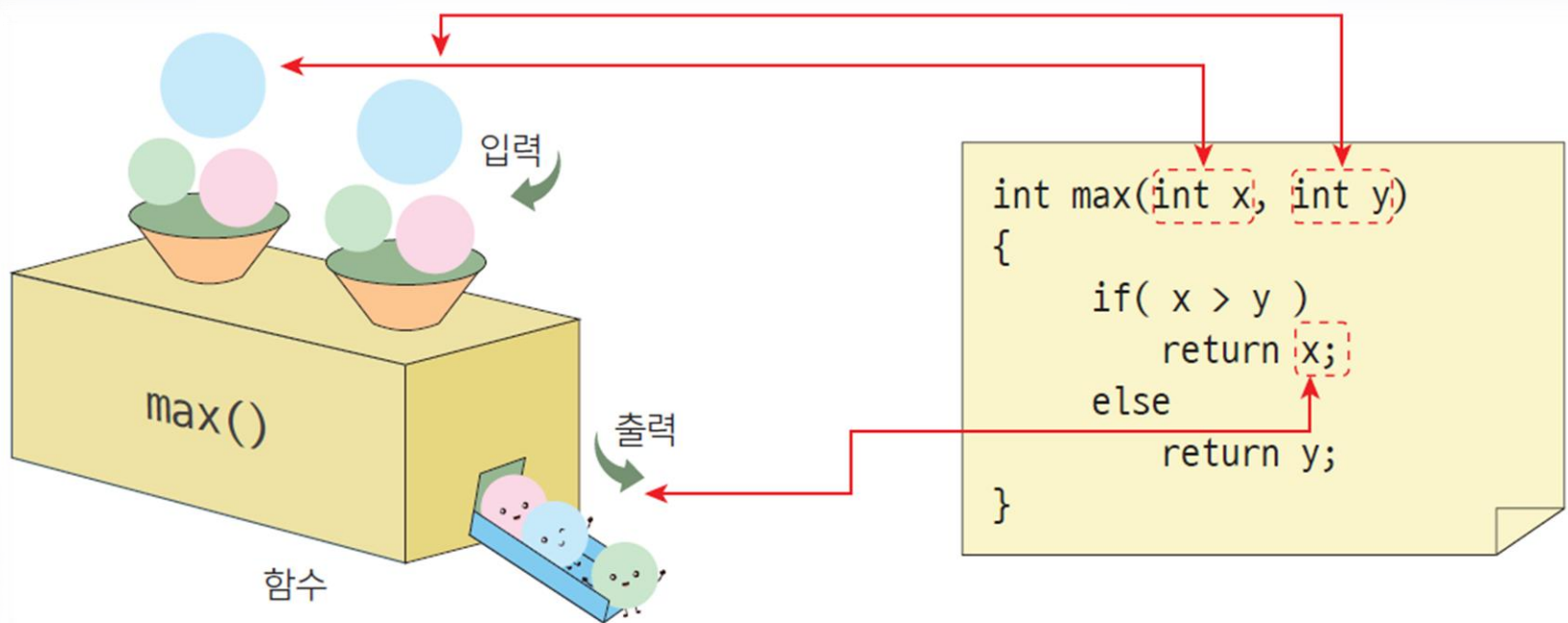
함수의 구조

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.

인수

```
value = max( 10, 20 );
```

매개 변수

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

인수와 매개 변수

- 만약 매개 변수가 없는 경우에는 `print_stars(void)`와 같이 매개변수 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.

```
value = max( 10, 20 );
```

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


중간점검

1. 함수의 반환형이 없는 경우에는 어떻게 표시하는가?
2. 수식 $(x+y+z)$ 의 값을 반환하는 문장을 작성하여 보자.
3. **double**형 변수 **f**를 매개 변수로 가지고 **double**형의 값을 반환하는 **fabs()** 함수 헤더를 정의하여 보자.



예제

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



정수 2개를 입력하시오: 10 20
더 큰 값은 20입니다.

```
#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);
```

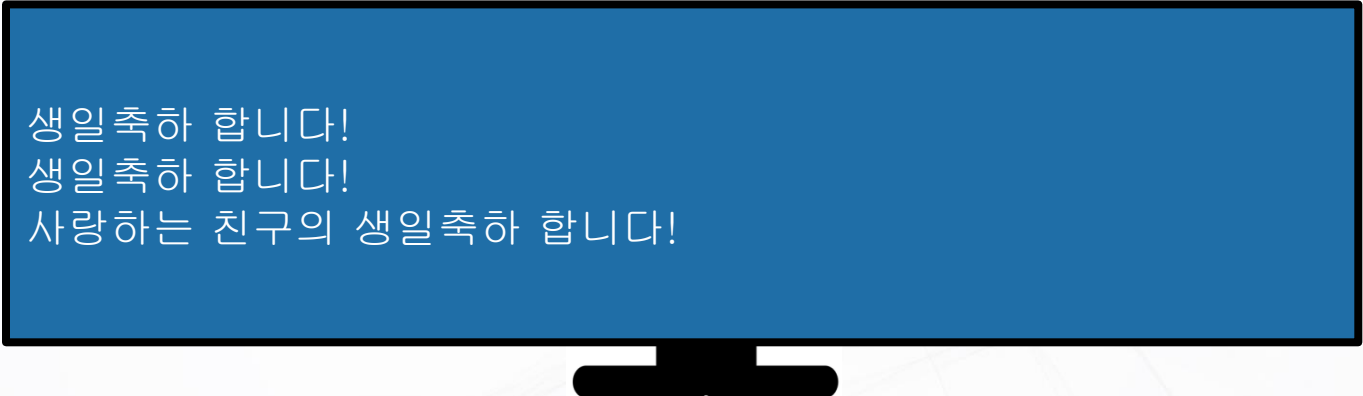
```
    printf("더 큰 값은 %d입니다. \n", larger);
```

```
    return 0;
```

```
}
```


Lab: 생일 축하 함수

- 생일 축하 메시지를 출력하는 함수 `happyBirthday()`를 작성해보자.



생일 축하 합니다!
생일 축하 합니다!
사랑하는 친구의 생일 축하 합니다!

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

```
void happyBirthday()
```

```
{
```

```
    printf("생일축하 합니다! \n");
```

```
    printf("생일축하 합니다! \n");
```

```
    printf("사랑하는 친구의 ");
```

```
    printf("생일축하 합니다! \n");
```

```
}
```

```
int main(void)
```

```
{
```

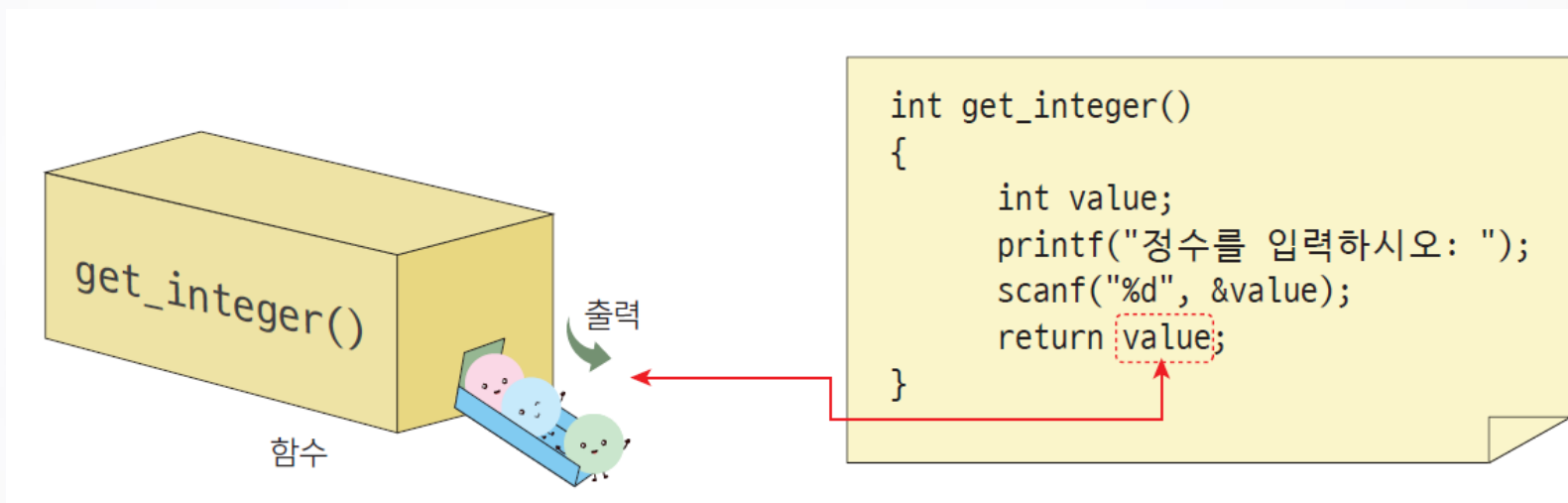
```
    happyBirthday();
```

```
    return 0;
```

```
}
```

Lab: 정수를 입력받는 get_integer() 함수

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



☆ 전역변수 = 자동 초기화

```
// 사용자로부터 정수를 받는 함수
```

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

```
int get_integer(void)
```

```
{
```

```
    int value;
```

```
    printf("정수를 입력하시오: ");
```

```
    scanf("%d", &value);
```

```
    return value;
```

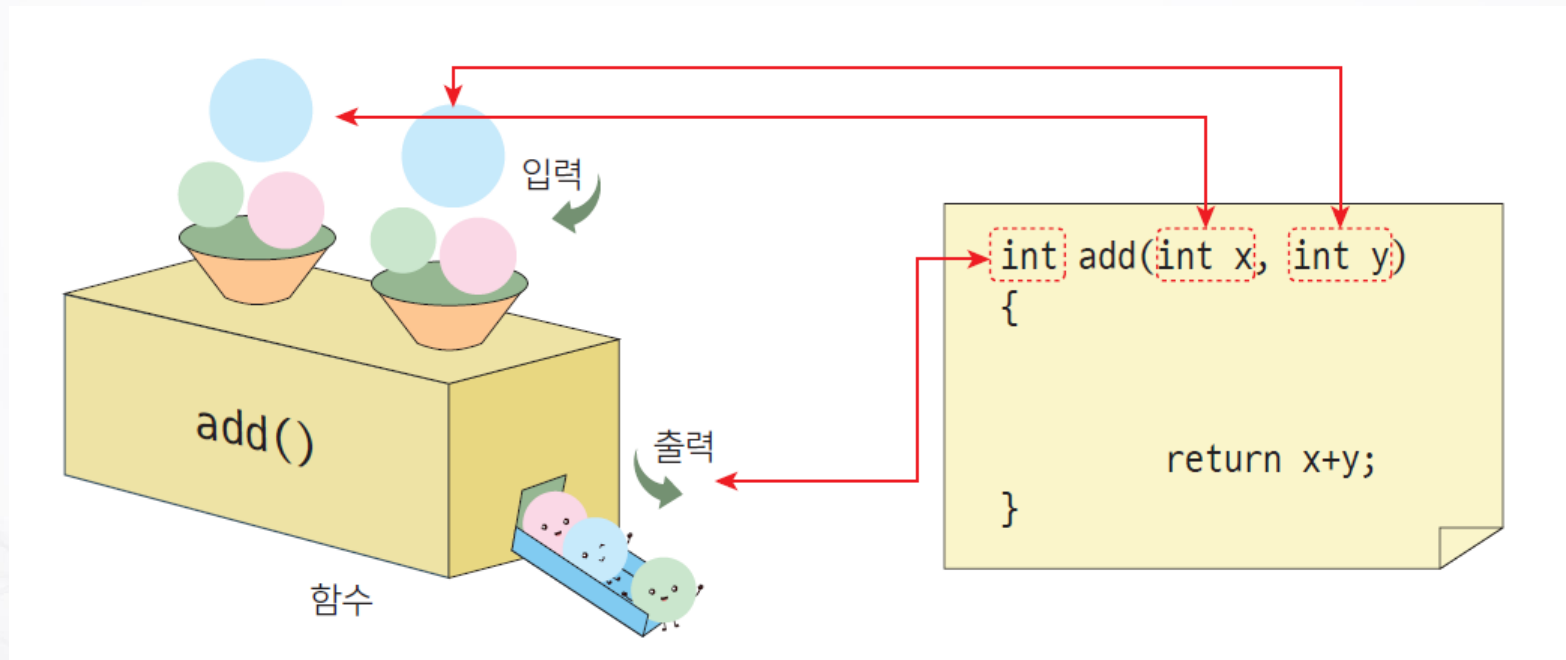
```
}
```

전역변수 초기화 VS

지역변수를 처리

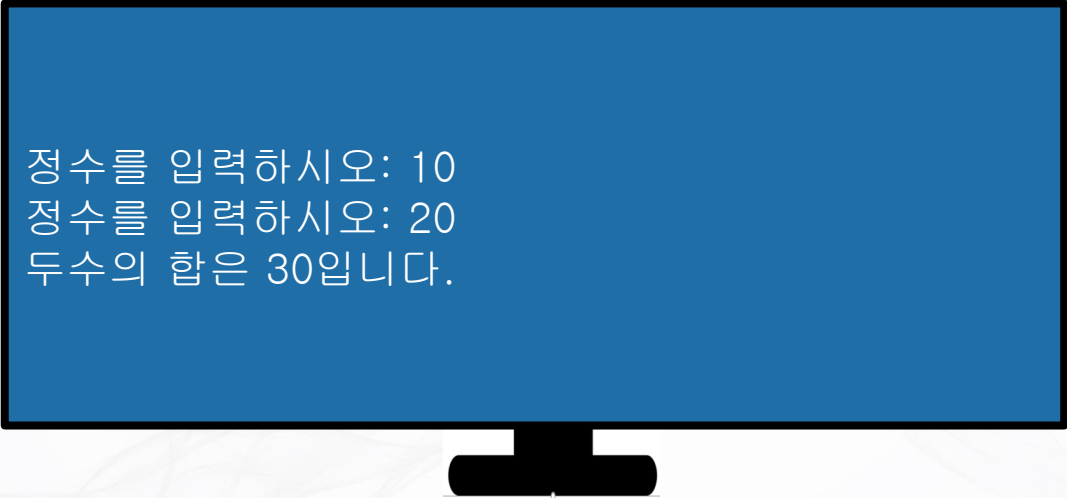
Lab: 정수의 합을 계산하는 add() 함수

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



Lab: 정수의 합을 계산하는 프로그램

- 앞에서 작성한 `get_integer()`까지 사용하여 사용자로 부터 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;
}
```

Lab: 팩토리얼 계산 함수

- 이번에는 정수 n 을 받아서 1에서 n 까지의 정수들의 곱을 구하는 팩토리얼 함수를 작성하여 보자.

$$n! = n * (n-1) * (n-2) * \dots * 1$$

알고 싶은 팩토리얼의 값은? 12
12!의 값은 479001600입니다.


```
#include <stdio.h>

int factorial(int n)
{
    int result = 1;

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

int main(void)
{
    int n;
    printf("알고 싶은 팩토리얼의 값은? ");
    scanf("%d", &n);
    printf("%d!의 값은 %d입니다. \n", n, factorial(n));
    return 0;
}
```

중간점검

1. 인수와 매개 변수는 어떤 관계가 있는가?
2. 반환값이 **double**형으로 정의된 함수에서 정수를 반환하면 어떤 일이 발생할까?



Lab: 온도 변환기

- 섭씨 온도를 화씨 온도로, 또 그 반대로 변환하는 프로그램을 작성해보자.

```
=====
'c' 섭씨온도에서 화씨온도로 변환
'f' 화씨온도에서 섭씨온도로 변환
'q' 종료
=====
메뉴에서 선택하세요: f
화씨온도: 100
섭씨온도: 37.77778
=====
'c' 섭씨온도에서 화씨온도로 변환
'f' 화씨온도에서 섭씨온도로 변환
'q' 종료
=====
메뉴에서 선택하세요:
```

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

```
void printMenu()
```

```
{
```

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

```
    printf(" 'c' 섭씨온도에서 화씨온도로 변환\n");
```

```
    printf(" 'f' 화씨온도에서 섭씨온도로 변환\n");
```

```
    printf(" 'q' 종료\n");
```

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

```
}
```

```
double C2F(double c_temp)
```

```
{
```

```
    return 9.0 / 5.0 * c_temp + 32;
```

```
}
```

```
double F2C(double f_temp)
```

```
{
```

```
    return (f_temp - 32.0) * 5.0 / 9.0;
```

```
}
```

```
int main(void)
{
    char choice;
    double temp;

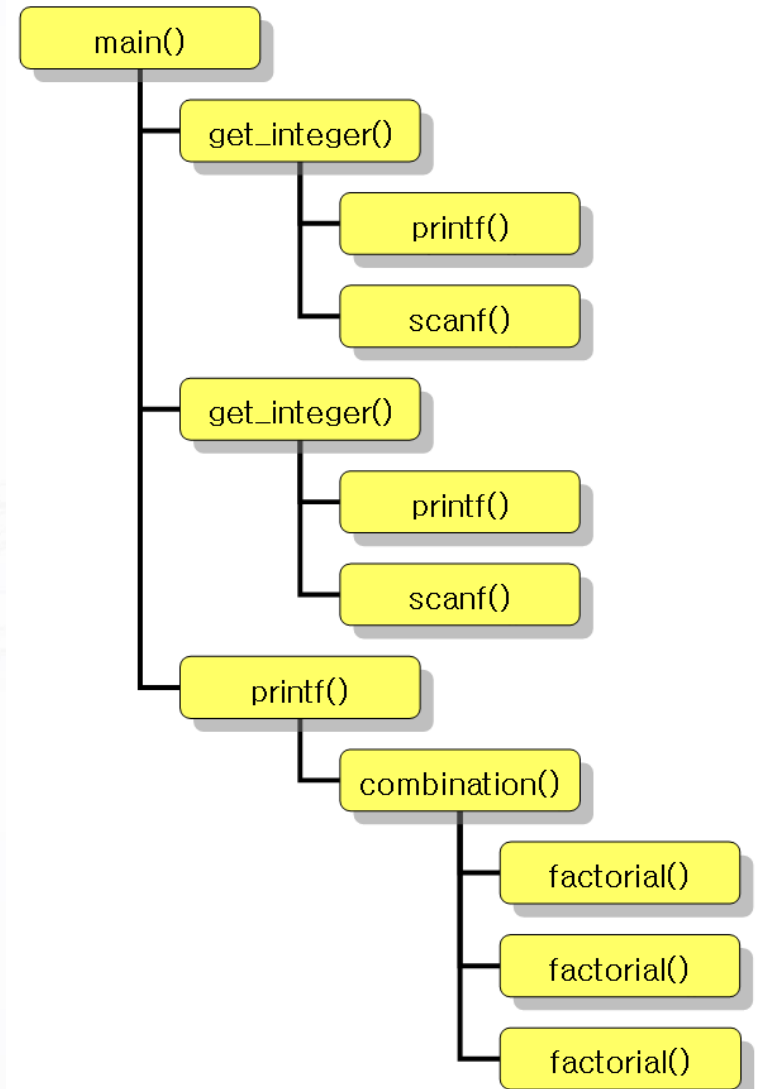
    while (1) {
        printMenu();
        printf("메뉴에서 선택하세요: ");
        choice = getchar();
        if (choice == 'q') break;
        else if (choice == 'c') {
            printf("섭씨온도: ");
            scanf("%lf", &temp);
            printf("화씨온도: %lf \n\n", C2F(temp));
        }
        else if (choice == 'f') {
            printf("화씨온도: ");
            scanf("%lf", &temp);
            printf("섭씨온도: %lf \n\n", F2C(temp));
        }
        getchar();          // 엔터키 문자를 삭제하기 위하여 필요!
    }
    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;
```

```
}
```


Lab: 소수 찾기

- 주어진 숫자가 소수(prime)인지를 결정하는 프로그램이다.
- 양의 정수 n 이 소수가 되려면 1과 자기 자신만을 약수로 가져야 한다.
- 암호학에서 많이 사용

정수를 입력하시오: 12229
12229은 소수가 아닙니다.

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

알고리즘

1. 사용자로부터 정수를 입력받아서 변수 n 에 저장한다.
2. `for(i=2; i<n ; i++)`
3. n 을 i 로 나누어서 나머지가 0인지 본다.
4. 나머지가 0이면 약수가 있는 것이므로 소수가 아니다.
5. 반복이 정상적으로 종료되고 약수가 없다면 소수이다.

```
#include <stdio.h>

int get_integer(void)
{
    int n;

    printf("정수를 입력하시오: ");
    scanf("%d", &n);
    return n;
}

int is_prime(int n)
{
    int i;

    for (i = 2; i < n; i++) {
        if (n%i == 0)
            return 0;
    }
    return 1;
}
```

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

    if (is_prime(n) == 1)
        printf("%d은 소수입니다.\n", n);
    else
        printf("%d은 소수가 아닙니다.\n", n);
    return 0;
}
```

도전문제

1. 1부터 사용자가 입력한 숫자 n 사이의 모든 소수를 찾도록 위의 프로그램을 변경하여 보자.



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

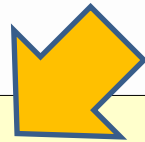
빌드 + IntelliSense

검색 오류 목록

	코드	설명	프로젝트	파일	줄	Suppress
	C4013	'c_to_f'이(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 %f도는 화씨 %f입니다. \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.


```
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.

```

#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

중간 점검

1. 함수 정의의 첫 번째 줄에는 어떤 정보들이 포함되는가? 이것을 무엇이라고 부르는가?
2. 함수가 반환할 수 있는 값의 개수는?
3. 함수가 값을 반환하지 않는다면 반환형은 어떻게 정의되어야 하는가?
4. 함수 정의와 함수 원형의 차이점은 무엇인가?
5. 함수 원형에 반드시 필요한 것은 아니지만 대개 매개 변수들의 이름을 추가하는 이유는 무엇인가?
6. 다음과 같은 함수 원형을 보고 우리가 알 수 있는 정보는 어떤 것들인가?

`double pow(double, double);`



라이브러리 함수

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



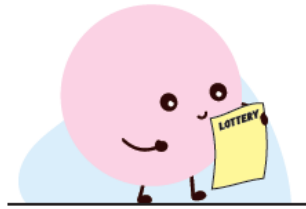
난수 함수

- 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 작성해보자. 로또 번호는 1부터 45까지의 숫자 6개로 이루어진다.



```
#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));`



42 18 35 41 45 20

- 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.

아직도 문제는 있다.

- 난수가 겹칠 수 있다.
- 겹치는 난수 검사는 뒤에서 학습하는 배열을 사용하는 것이 최선



Lab: 동전 던지기 게임

- 동전을 100번 던져서 앞면이 나오는 횟수와 뒷면이 나오는 횟수를 출력한다.

동전의 앞면: 53
동전의 뒷면: 47



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

int coin_toss(void);

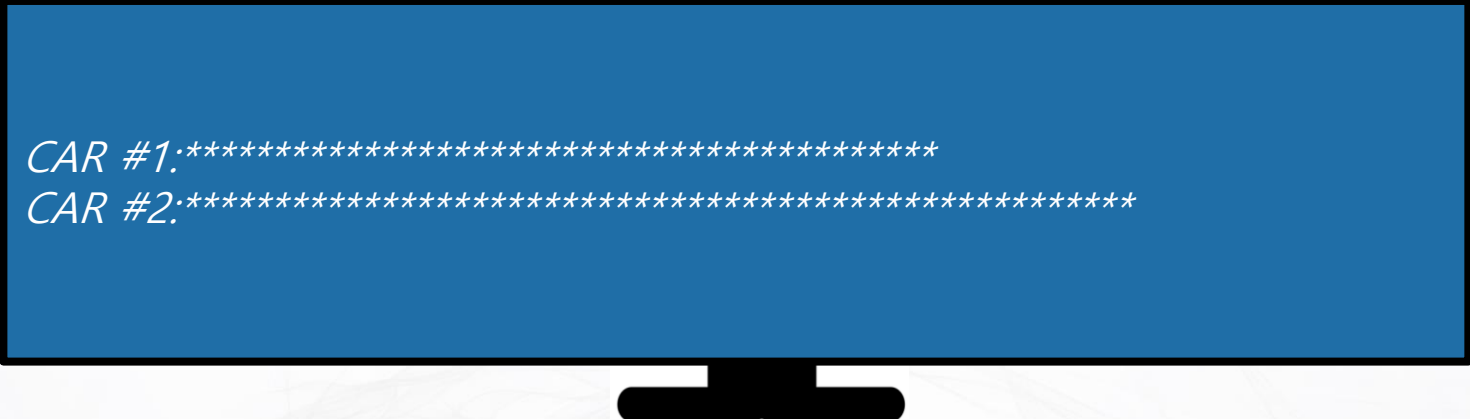
int main(void)
{
    int toss;
    int heads = 0;
    int tails = 0;
    srand((unsigned)time(NULL));

    for (toss = 0; toss < 100; toss++) {
        if (coin_toss() == 1)
            heads++;
        else
            tails++;
    }
}
```

```
printf("동전의 앞면: %d \n", heads);  
printf("동전의 뒷면: %d \n", tails);  
return 0;  
  
}  
  
int coin_toss( void )  
{  
    int head = rand() % 2;  
    return head;  
}
```

Lab: 자동차 게임

- 난수를 이용하여서 자동차 게임을 작성해보자.



```
CAR #1.*****  
CAR #2.*****
```

알고리즘

1. 난수 발생기를 초기화한다
2. `for(i=0; i<주행시간; i++)`
3. 난수를 발생하여서 자동차1의 주행거리에 누적한다.
4. 난수를 발생하여서 자동차2의 주행거리에 누적한다.
5. `disp_car()`를 호출하여서 자동차1을 화면에 *표로 그린다.
6. `disp_car()`를 호출하여서 자동차2을 화면에 *표로 그린다.


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

void disp_car(int car_number, int distance)
{
    int i;
    printf("CAR #%d:", car_number);
    for( i = 0; i < distance/10; i++ ) {
        printf("*");
    }
    printf("\n");
}
```

```
int main(void)
{
    int i;
    int car1_dist=0, car2_dist=0;

    srand( (unsigned)time( NULL ) );

    for( i = 0; i < 6; i++ ) {
        car1_dist += rand() % 100;
        car2_dist += rand() % 100;
        disp_car(1, car1_dist);
        disp_car(2, car2_dist);
        printf("-----\n");
        _getch();
    }
    return 0;
}
```

rand()를 이용하여서 난수를 발생
다. 난수의 범위는 %연산자를 사
하여서 0에서 99로 제한하였다.

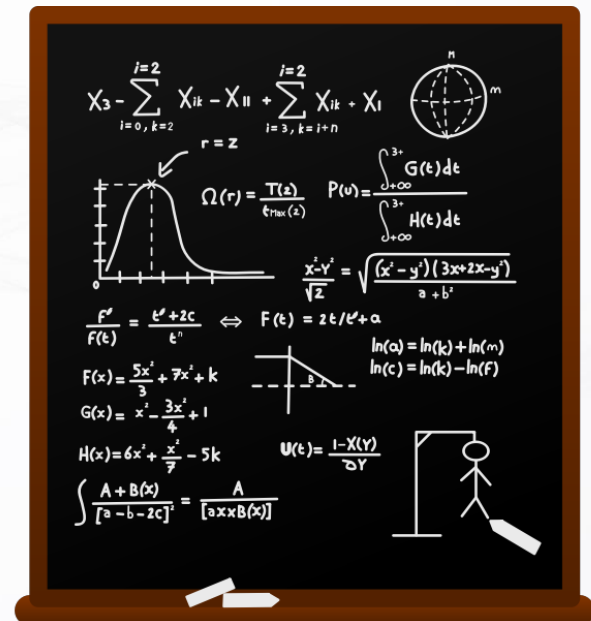
도전문제

- 자동차를 3개로 늘려보자.



표준 라이브러리 함수(수학 함수)

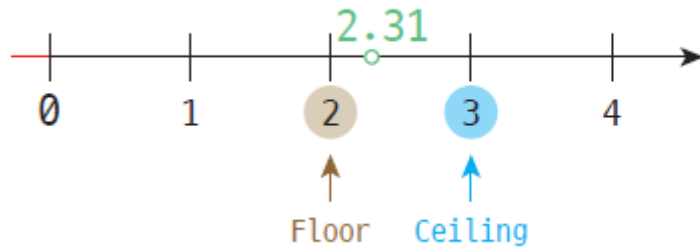
- 이번 장에서는 수치 계산을 하는 라이브러리 함수에 대하여 살펴본다. 이들 라이브러리 함수를 사용하면 복잡한 산술 연산을 할 수 있다.
- 수학 함수들에 대한 원형은 헤더파일 **math.h**에 있다. 수학 함수는 일반적으로 **double**형의 매개 변수와 반환값을 가진다.



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



floor(x)



ceil(x)

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

```
result = floor(value);  
printf("%lf ", result);
```

// result는 1.0이다.

```
result = ceil(value);  
printf("%lf ", result);
```

// result는 2.0이다.

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.*

// 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

중간 점검

1. 90도에서의 싸인값을 계산하는 문장을 작성하여 보라.
2. `rand() % 10` 이 계산하는 값의 범위는?



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.

```
#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

Lab: 시간 맞추기 게임

- 사용자에게 정확한 시간을 예측하게 하는 게임을 만들어보자. 사용자에게 10초가 지나면 엔터키를 누르라고 한 후에, 정확한 시간과 얼마나 차이가 나는지를 출력한다.



10초가 되면 엔터키를 누르세요
종료되었습니다.
경과된 시간은 6 초입니다.

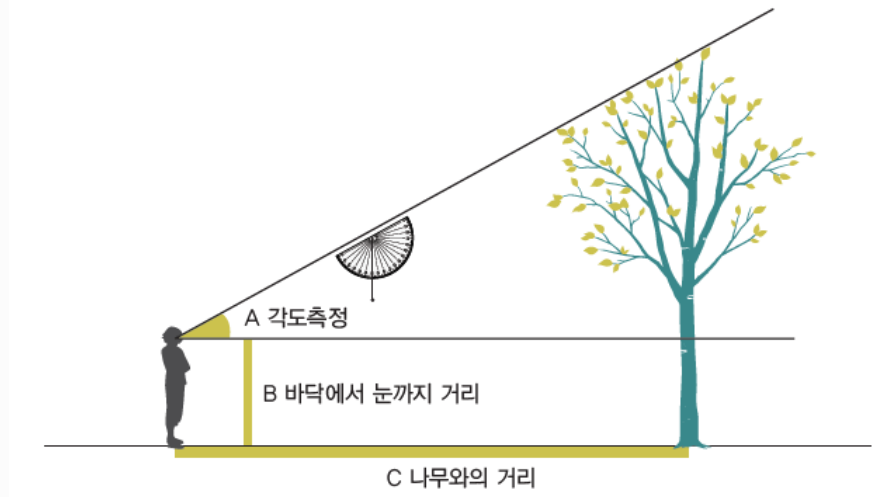
```
#include <stdio.h>
#include <time.h>

int main(void)
{
    time_t start, end; // time_t은 unsigned long과 동일하다.
    start = time(NULL);
    printf("10초가 되면 아무 키나 누르세요\n");
    while (1)
    {
        if (getchar())
            break;
    }
    printf("종료되었습니다.\n");
    end = time(NULL);
    printf("경과된 시간은 %1d 초입니다. \n", end - start);
    return 0;
}
```

C 라이브러리 함수 time()은
1970년 1월 1일 이후의 시간을
초 단위로 반환합니다.

Lab: 나무 높이 측정

- 각도기와 삼각 함수를 이용하면 나무의 높이를 측정할 수 있다. 다음 그림을 참조하여서 나무의 높이를 측정하는 프로그램을 작성해보자.



나무와의 길이(단위는 미터): 4.2
측정자의 키(단위는 미터): 1.8
각도(단위는 도): 62
나무의 높이(단위는 미터): 9.699047

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

int main(void)
{
    double height, distance, tree_height, degrees, radians;

    printf("나무와의 길이(단위는 미터): ");
    scanf("%lf", &distance);

    printf("측정자의 키(단위는 미터): ");
    scanf("%lf", &height);

    printf("각도(단위는 도): ");
    scanf("%lf", &degrees);

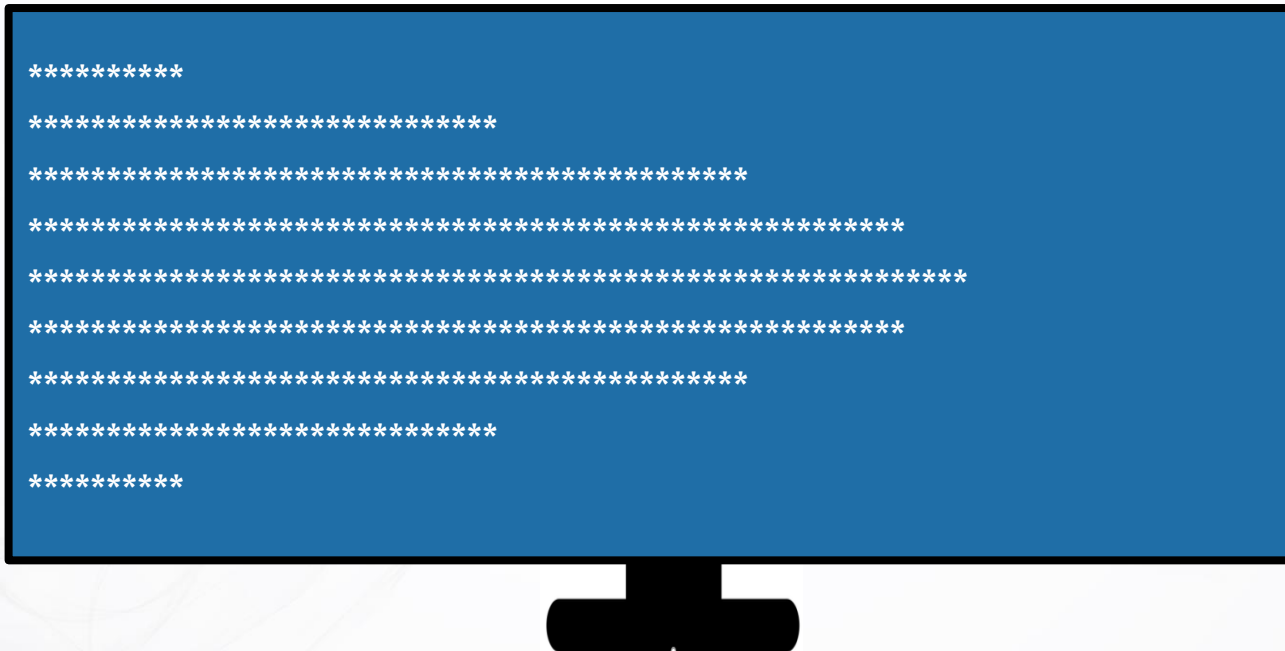
    radians = degrees * (3.141592 / 180.0);

    tree_height = tan(radians) * distance + height;
    printf("나무의 높이(단위는 미터): %lf \n", tree_height);

    return 0;
}
```


Lab: 삼각함수 그리기

- 우리는 삼각함수를 계산하는 라이브러리 함수를 학습하였다. 이것을 이용하여서 싸인함수 그래프를 90도 회전하여서 그려보자



```

#include <stdio.h>
#include <math.h>
#define PI 3.141592

double rad(double degree)
{
    return PI * degree / 180.0;
}

void drawbar(int height)
{
    for (int i = 0; i < height; i++)
        printf("*");
    printf("\n");
}

int main(void)
{
    int degree, x, y;
    for (degree = 0; degree <= 90; degree += 10) {
        // 싸인값은 -1.0에서 1.0이므로 정수로 반올림하여서 증폭한다.
        y = (int)(60 * sin(rad((double)degree)) + 0.5);
        drawbar(y);
    }
    return 0;
}

```

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.

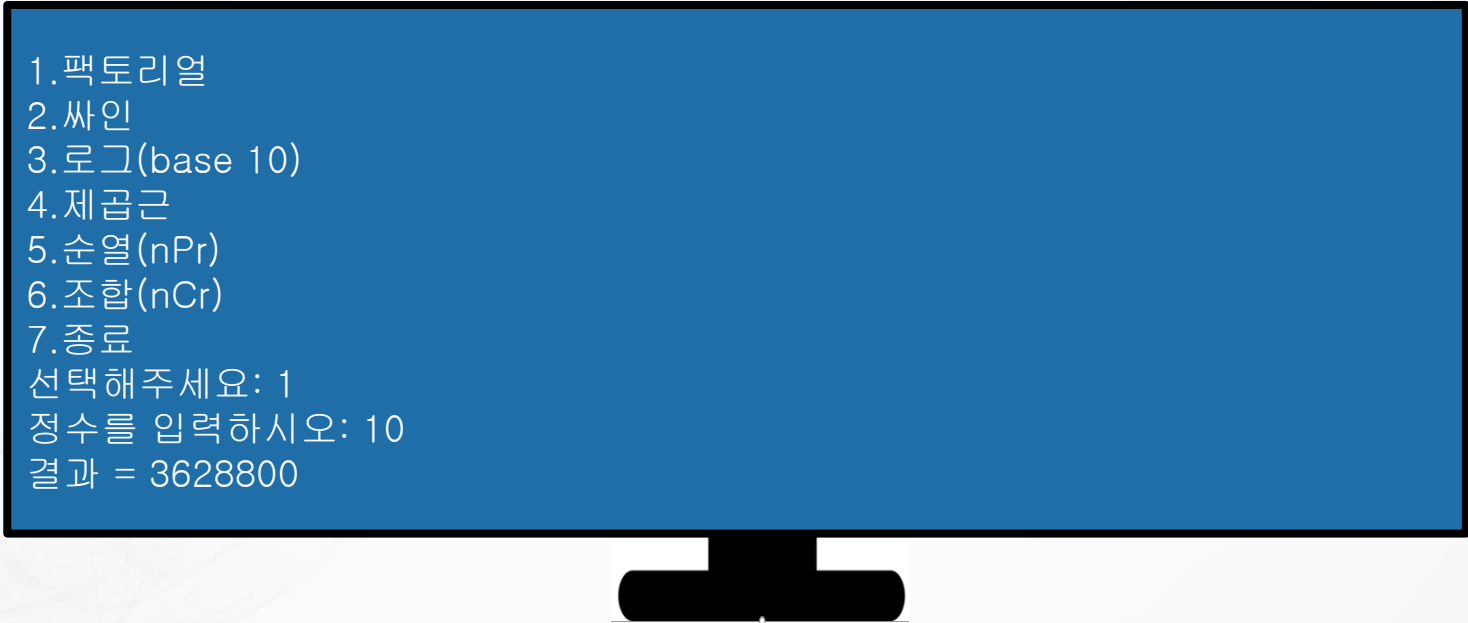
```
int main(void)
{
    // 숫자들의 리스트를 키보드에서 읽어들이는 코드
    ....
    // 숫자들을 크기순으로 정렬하는 코드
    ....
    // 정렬된 숫자들의 리스트를 화면에 출력하는 코드
    ...
}
```



```
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;
        }
    }
}
```

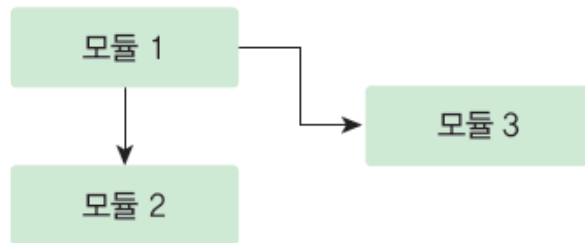
도전문제

- 아직 구현되지 않은 제곱근, 순열, 조합 계산 등을 구현해보자. 기타 공학자에게 필요한 기능을 추가해보자.

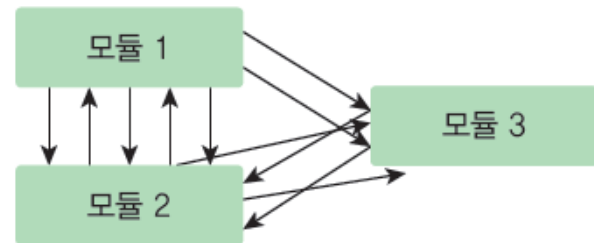


모듈화

- 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

이번 장에서 학습할 내용



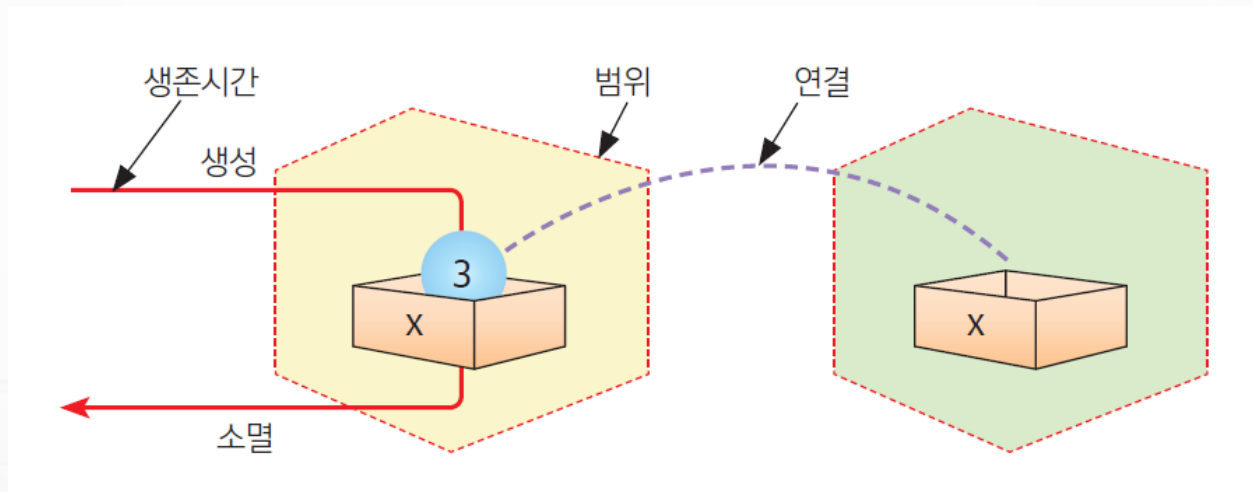
- 반복의 개념 이해
- 변수의 속성
- 전역, 지역 변수
- 자동 변수와 정적 변수
- 재귀 호출

이번 장에서는 함수와 변수와의 관계를 집중적으로 살펴볼 것이다. 또한 함수가 자기 자신을 호출하는 재귀 호출에 대하여 살펴본다.

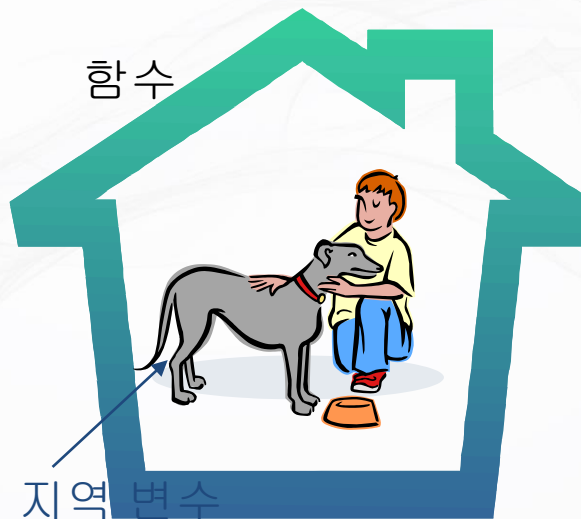
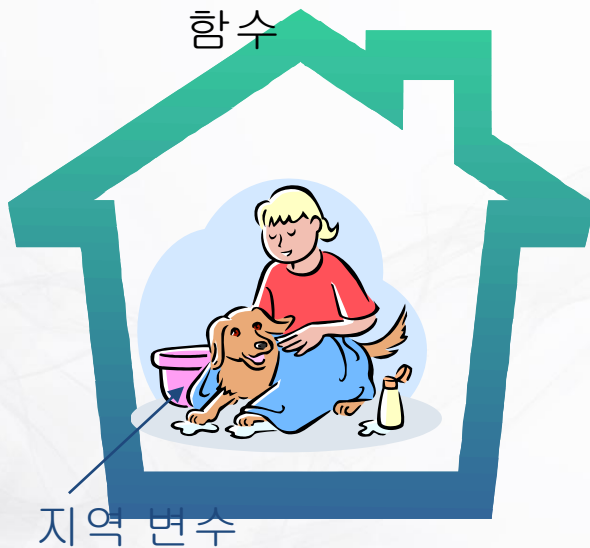
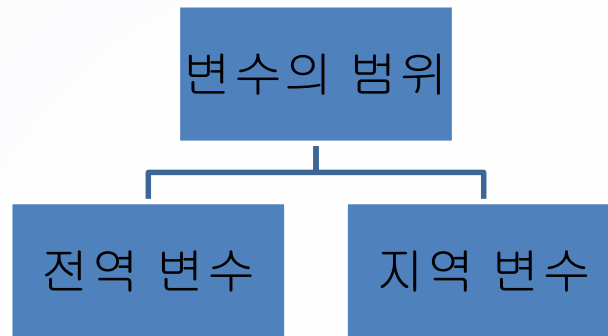


변수의 속성

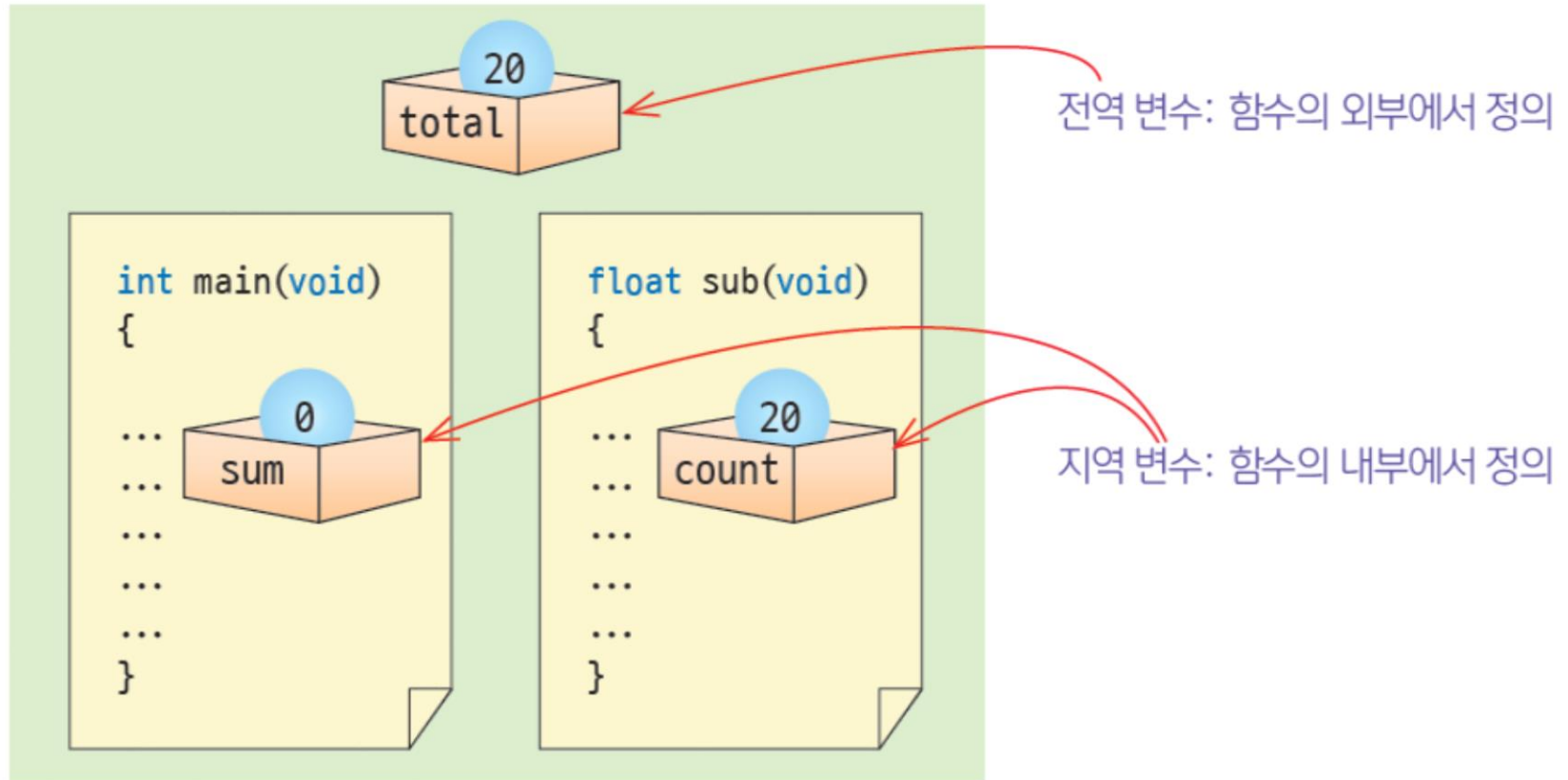
- 변수의 속성 : 이름, 타입, 크기, 값 + 범위, 생존 시간, 연결
 - 범위(scope) : 변수가 사용 가능한 범위, 가시성
 - 생존 시간(lifetime) : 메모리에 존재하는 시간
 - 연결(linkage) : 다른 영역에 있는 변수와의 연결 상태



변수의 범위



전역 변수와 지역 변수



지역 변수

- 지역 변수(**local variable**)는 블록 안에 선언되는 변수

```
int sub(void)
{
    int x = 0;

    while(flag!= 0){
        int y;
        ...
    }

    y = 0; // 오류!!
    ...
}
```

지역 변수 x가 사용가능한 범위

지역 변수 y가 사용가능한 범위

y가 선언된 블록을 벗어나서 사용하였으므로 오류!

지역 변수는 선언된 블록을 떠나면 안됩니다.



지역 변수 선언 위치

- 최신 버전의 C에서는 블록 안의 어떤 위치에서도 선언 가능!!

```
while(1) {
```

```
    ...
```

```
    ...
```

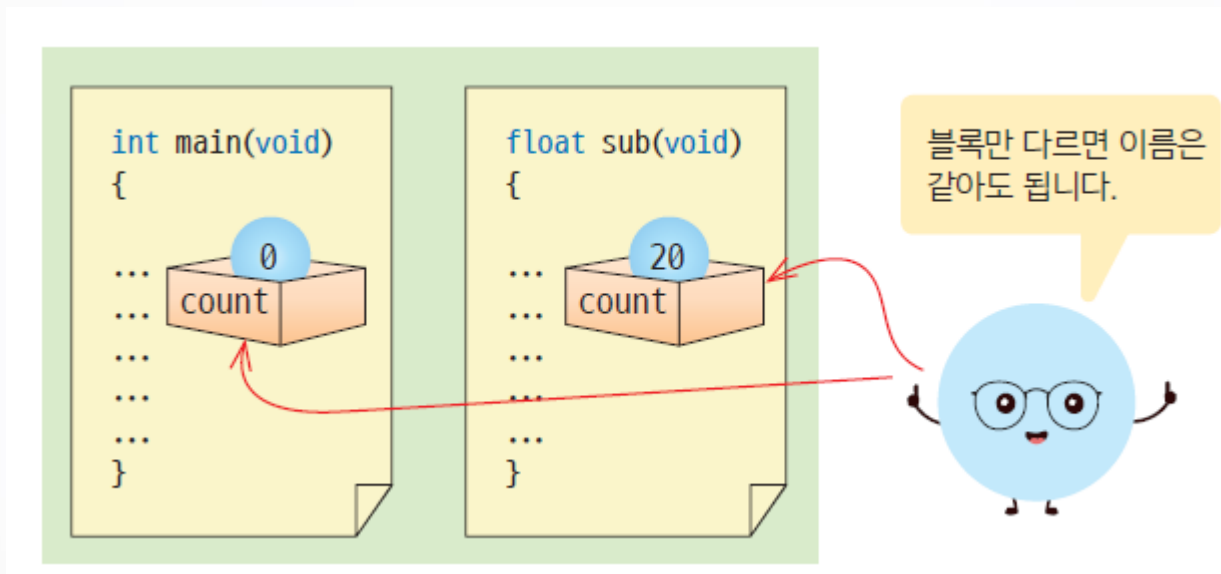
```
    int sum = 0;
```

```
    ...
```

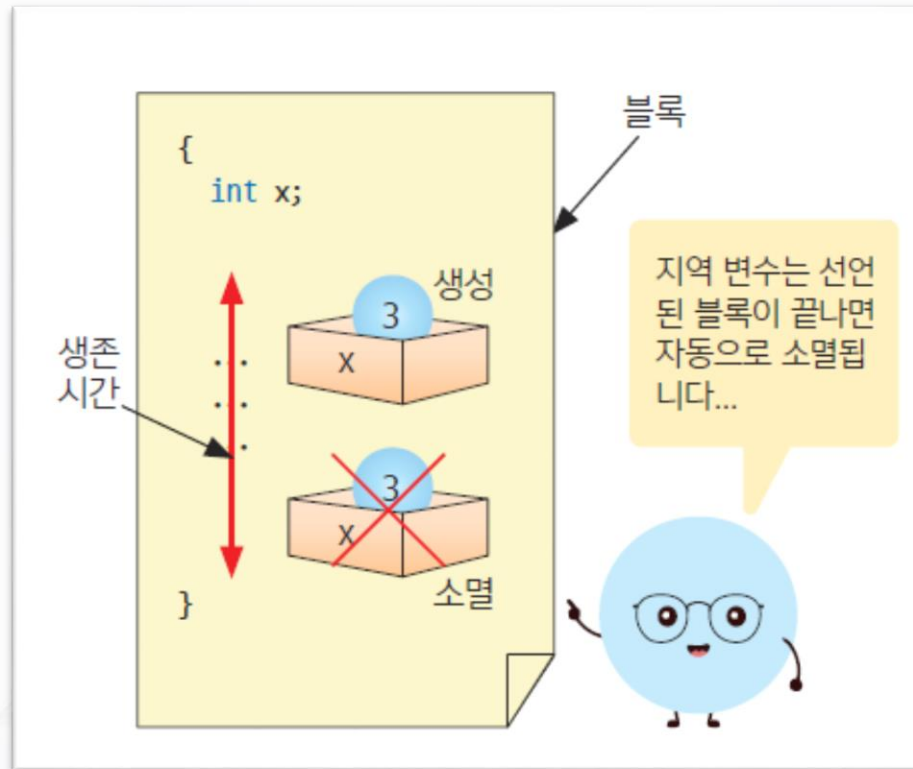
```
}
```

블록의 중간에서도 얼마든지 지역 변수
를 선언할 수 있다.

이름이 같은 지역 변수



지역 변수의 생존 기간



지역 변수 예제

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

```
int main(void)
```

```
{
```

```
    int i;
```

```
    for(i = 0; i < 5; i++)
```

```
    {
```

```
        int temp = 1;
```

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

```
        temp++;
```

```
    }
```

```
    return 0;
```

```
}
```

블록이 시작할 때 마다
생성되어 초기화된다.

```
temp = 1  
temp = 1  
temp = 1  
temp = 1  
temp = 1
```

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

```
int main(void)
```

```
{
```

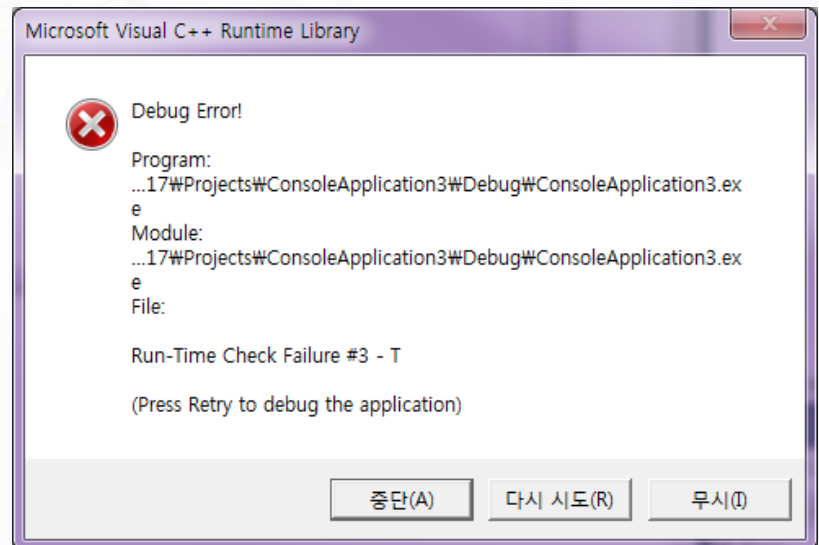
```
    int temp;
```

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

```
    return 0;
```

```
}
```

초기화 되지 않았으
므로 쓰레기 값을
가진다.



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 함수 호출시의 인수 값으로 초기화되어 있다는 점이다.

```
int inc(int counter)
{
    counter++;
    return counter;
}
```

매개 변수도 일종의
지역 변수

함수의 매개 변수

```
#include <stdio.h>
int inc(int counter);
```

```
int main(void)
{
    int i;
```

```
    i = 10;
    printf("함수 호출전 i=%d\n", i);
```

```
    inc(i);
```

```
    printf("함수 호출후 i=%d\n", i);
    return 0;
```

```
}
```

```
void inc(int counter)
```

```
{
```

```
    counter++;
```

```
}
```

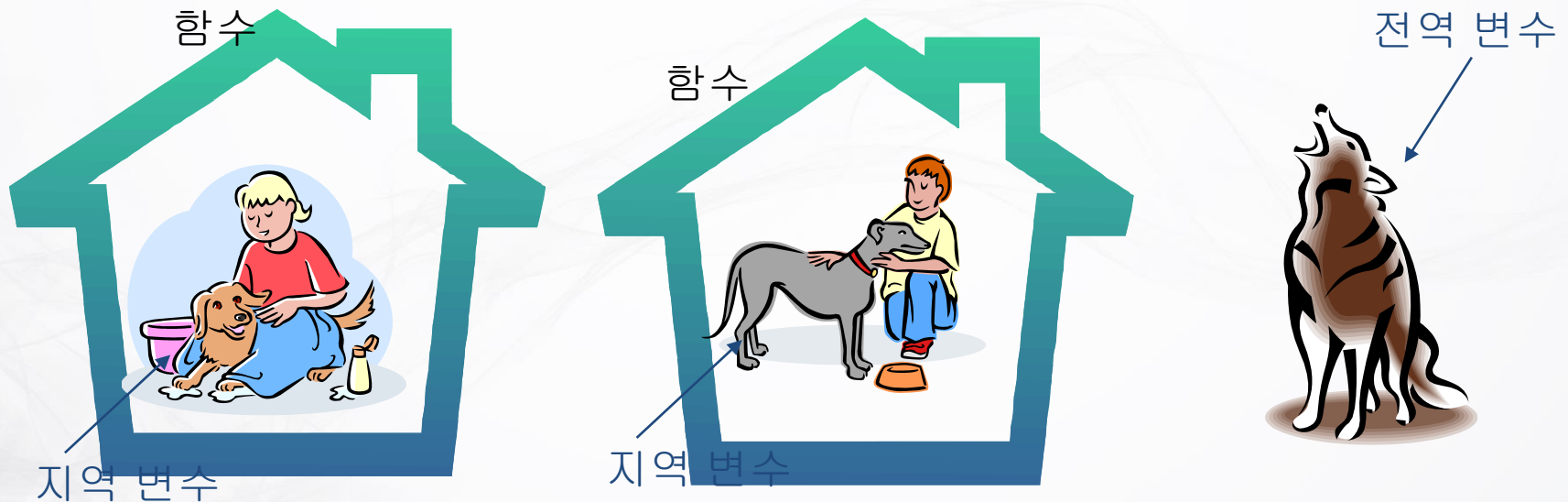
값에 의한 호출
(call by value)

매개 변수도 일종의
지역 변수임

함수 호출전 i=10
함수 호출후 i=10

전역 변수

- 전역 변수(global variable)는 함수 외부에서 선언되는 변수이다.
- 전역 변수의 범위는 소스 파일 전체이다.



전역 변수의 초기값과 생존 기간

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

```
int A;
```

```
int B;
```

```
int add()
```

```
{
```

```
    return A + B;
```

```
}
```

```
int main(void)
```

```
{
```

```
    int answer;
```

```
    A = 5;
```

```
    B = 7;
```

```
    answer = add();
```

```
    printf(" % d + % d = % d\n", A, B, answer);
```

```
    return 0;
```

```
}
```

전역 변수
초기값은 0

5 + 7 = 12

전역 변수의 초기값

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

```
int counter;
```

```
int main(void)
```

```
{
```

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

```
    return 0;
```

```
}
```

전역 변수는 컴파일러가 프로그램 실행시에 0으로 초기화한다.

counter = 0

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

```
int x;  
void sub();
```

```
int main(void)  
{  
    for (x = 0; x < 10; x++)  
        sub();  
}
```

```
void sub()  
{  
    for (x = 0; x < 10; x++)  
        printf("*");  
}
```



전역 변수의 사용

- 거의 모든 함수에서 사용하는 공통적인 데이터는 전역 변수로 한다.
- 일부의 함수들만 사용하는 데이터는 전역 변수로 하지 말고 함수의 인수로 전달한다.

같은 이름의 전역 변수와 지역 변수

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

```
int sum = 1; // 전역 변수
```

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

```
    int sum = 0; // 지역 변수
```

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

```
    return 0;
```

```
}
```

전역 변수와 지역 변수가 동일한 이름으로 선언된다.

sum = 0

중간 점검

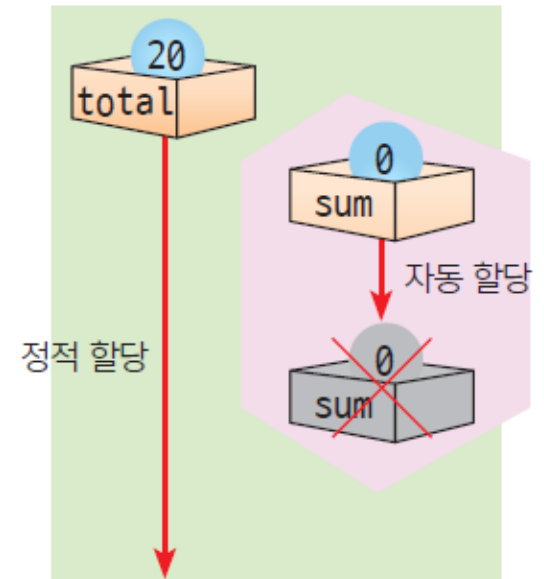
1. 변수의 범위는 대개 무엇으로 결정되는가?
2. 변수의 범위에는 몇 가지의 종류가 있는가?
3. 파일 범위를 가지는 변수를 무엇이라고 하는가?
4. 블록 범위를 가지는 변수를 무엇이라고 하는가?
5. 지역 변수를 블록의 중간에서 정의할 수 있는가?
6. 똑같은 이름의 지역 변수가 서로 다른 함수 안에 정의될 수 있는가?
7. 지역 변수가 선언된 블록이 종료되면 지역 변수는 어떻게 되는가?
8. 지역 변수의 초기값은 얼마인가?
9. 함수의 매개 변수도 지역 변수인가?
10. 전역 변수는 어디에 선언되는가?
11. 전역 변수의 생존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



생존 기간

- 정적 할당(static allocation):
 - 프로그램 실행 시간 동안 계속 유지
- 자동 할당(automatic allocation):
 - 블록에 들어갈 때 생성
 - 블록에서 나올 때 소멸

정적 할당은 변수가 실행 시간
내내 존재하지만 자동 할당은
블록이 종료되면 소멸됩니다.



생존 기간

- 생존 기간을 결정하는 요인
 - 변수가 선언된 위치
 - 저장 유형 지정자
- 저장 유형 지정자
 - auto
 - register
 - static
 - extern

저장 유형 지정자 auto

- 변수를 선언한 위치에서 자동으로 만들어지고 블록을 벗어나게 되며 자동으로 소멸되는 저장 유형을 지정
- 지역 변수는 **auto**가 생략되어도 자동 변수가 된다.

```
int main(void)
```

```
{
```

```
    auto int sum = 0;
```

```
    int i = 0;
```

```
    ...
```

```
    ...
```

```
}
```

전부 자동 변수로서 함수가
시작되면 생성되고 끝나면 소
멸된다.

저장 유형 지정자 static

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

```
void sub() {
```

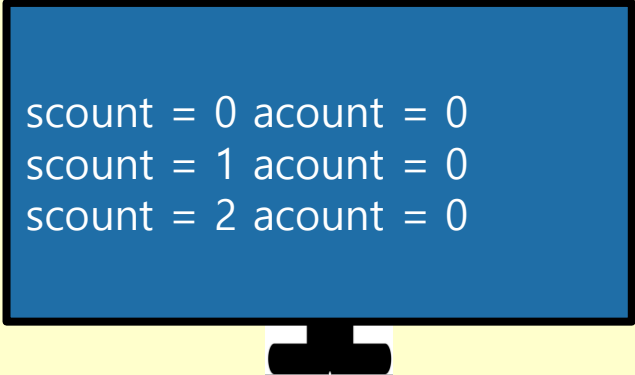
```
    static int scount = 0;  
    int account = 0;  
    printf("scount = %d\t", scount);
```

```
    printf("account = %d\n", account);  
    scount++;  
    account++;
```

```
}
```

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

정적 지역 변수로서 static을 붙이면
지역 변수가 정적 변수로 된다.



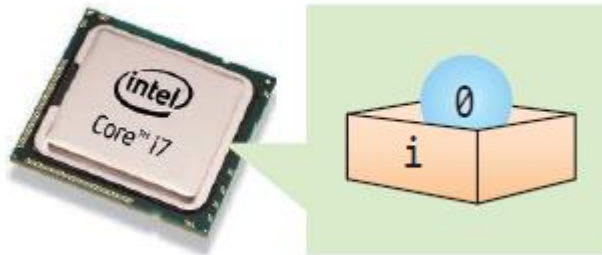
```
scount = 0 account = 0  
scount = 1 account = 0  
scount = 2 account = 0
```

저장 유형 지정자 register

- 레지스터(register)에 변수를 저장.

```
register int i;  
for(i = 0; i < 100; i++)  
    sum += i;
```

CPU안의 레지스터에
변수가 저장됨



volatile

- **volatile** 지정자는 하드웨어가 수시로 변수의 값을 변경하는 경우에 사용된다

```
volatile int io_port; // 하드웨어와 연결된 변수
```

```
void wait(void) {  
    io_port = 0;  
    while (io_port != 255)  
        ;  
}
```

volatile로 지정하면 컴파일러는 최적화를 중지하게 됩니다.



중간 점검

1. 저장 유형 지정자에는 어떤 것들이 있는가?
2. 지역 변수를 정적 변수로 만들려면 어떤 지정자를 붙여야 하는가?
3. 변수를 **CPU** 내부의 레지스터에 저장시키는 지정자는?
4. 컴파일러에게 변수가 외부에 선언되어 있다고 알리는 지정자는?
5. **static** 지정자를 변수 앞에 붙이면 무엇을 의미하는가?



Lab: 은행 계좌 구현하기

- 돈만 생기면 저금하는 사람을 가정하자. 이 사람을 위한 함수 `save(int amount)`를 작성하여 보자. 이 함수는 저금할 금액을 나타내는 인수 `amount`만을 받으며 `save(100)`과 같이 호출된다. `save()`는 정적 변수를 사용하여 현재까지 저축된 총액을 기억하고 있으며 한번 호출될 때마다 총 저축액을 다음과 같이 화면에 출력한다.

```
=====
입금   출금   잔고
=====
10000           10000
50000           60000
          10000  50000
30000           80000
=====
```

소스

```
#include <stdio.h>

// amount가 양수이면 입금이고 음수이면 출금으로 생각한다.
void save(int amount)
{
    static long balance = 0;

    if (amount >= 0)
        printf("%d \t\t", amount);
    else
        printf("\t %d \t", -amount);

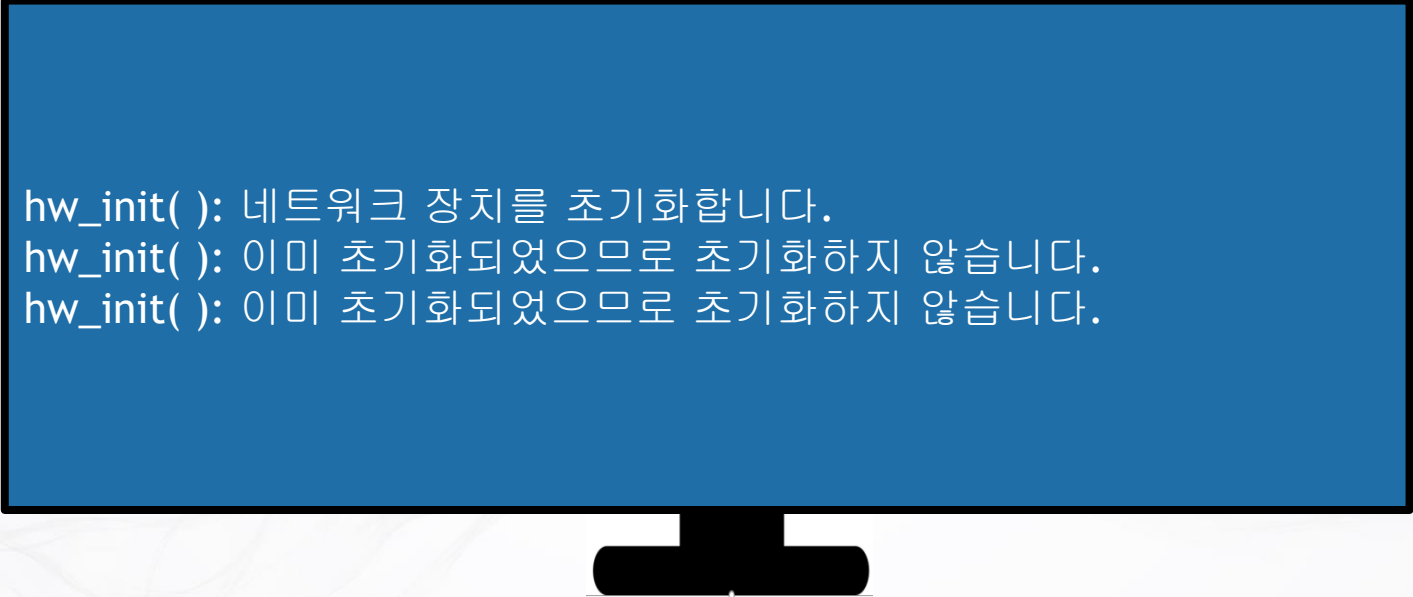
    balance += amount;
    printf("%d \n", balance);
}
```

소스

```
int main(void) {  
    printf("=====\n");  
    printf("입금 \t출금\t 잔고\n");  
    printf("=====\n");  
    save(10000);  
    save(50000);  
    save(-10000);  
    save(30000);  
    printf("=====\n");  
    return 0;  
}
```


Lab: 한번만 초기화하기

- 정적 변수는 한번만 초기화하고 싶은 경우에도 사용된다



hw_init(): 네트워크 장치를 초기화합니다.
hw_init(): 이미 초기화되었으므로 초기화하지 않습니다.
hw_init(): 이미 초기화되었으므로 초기화하지 않습니다.

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

```
void hw_init();
```

```
int main(void)
```

```
{
```

```
    hw_init();
```

```
    hw_init();
```

```
    hw_init();
```

```
    return 0;
```

```
}
```

```
void hw_init()
```

```
{
```

```
    static int initd = 0;
```

```
    if( initd == 0 ){
```

```
        printf("hw_init(): 네트워크 장치를 초기화합니다. \n");
```

```
        initd = 1;
```

```
    }
```

```
    else {
```

```
        printf("hw_init(): 이미 초기화되었으므로 초기화하지 않습니
```

```
다. \n");
```

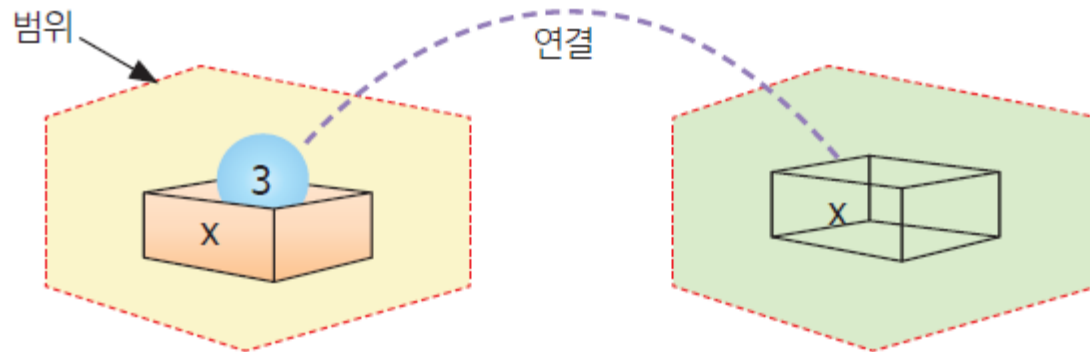
```
    }
```

```
}
```

오타! 수정해주세요.

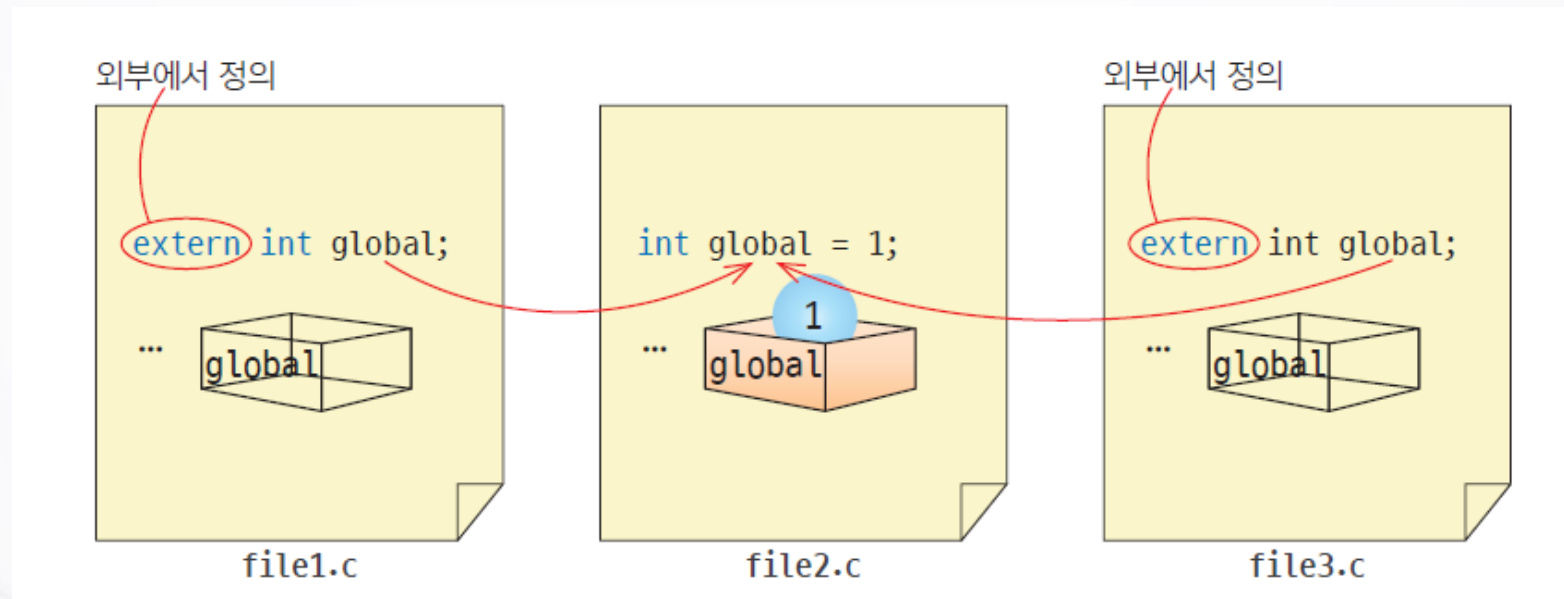
연결

- *연결(linkage)*: 다른 범위에 속하는 변수들을 서로 연결하는 것
 - 외부 연결
 - 내부 연결
 - 무연결
- 전역 변수만이 연결을 가질 수 있다.



외부 연결

- 전역 변수를 **extern**을 이용하여서 서로 연결



연결 예제

linkage1.c

```
#include <stdio.h>

int all_files;
static int this_file;
extern void sub();

int main(void)
{
    sub();
    printf("%d\n", all_files);
    return 0;
}
```

linkage2.c

```
extern int all_files;

// extern int this_file;

void sub(void)
{
    all_files = 10;
}
```

함수 앞의 static

main.c

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

```
//extern void f1();
```

```
extern void f2();
```

```
int main(void)
```

```
{
```

```
    f2();
```

```
    return 0;
```

```
}
```

sub.c

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

```
static void f1()
```

```
{
```

```
    printf("f1()가 호출되었습니다.\n");
```

```
}
```

```
void f2()
```

```
{
```

```
    printf("f2()가 호출되었습니다.\n");
```

```
}
```

f2()가 호출되었습니다.

블록에서 extern을 이용한 전역 변수 참조

- extern은 블록에서 전역 변수에 접근할 때도 사용된다.

```
#include <stdio.h>
int x = 50;

int main(void)
{
    int x = 100;
    {
        extern int x;
        printf("x= %d\n", x);
    }
    return 0;
}
```

x= 50

어떤 저장 유형을 사용하여 하는가?

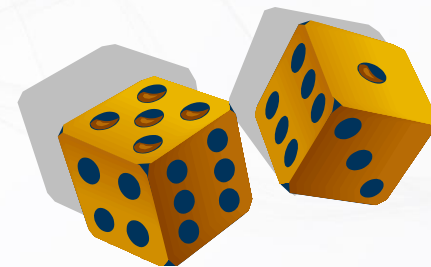
- 일반적으로는 *자동 저장 유형* 사용 권장
- 변수의 값이 함수 호출이 끝나도 그 값을 유지하여야 할 필요가 있다면 *지역 정적*
- 만약 많은 함수에서 공유되어야 하는 변수라면 *외부 참조 변수*

저장 유형	키워드	정의되는 위치	범위	생존 시간
자동	auto	함수 내부	지역	임시
레지스터	register	함수 내부	지역	임시
정적 지역	static	함수 내부	지역	영구
전역	없음	함수 외부	모든 소스 파일	영구
정적 전역	static	함수 외부	하나의 소스 파일	영구
외부 참조	extern	함수 외부	모든 소스 파일	영구

Lab: 난수 발생기

- 자체적인 난수 발생기 작성
- 이전에 만들어졌던 난수를 이용하여 새로운 난수를 생성함을 알 수 있다. 따라서 함수 호출이 종료되더라도 이전에 만들어졌던 난수를 어딘가에 저장하고 있어야 한다

$$r_{n+1} = (a \cdot r_n + b) \bmod M$$



실행 결과

- 다음과 같은 함수를 작성하여 사용해 보자.
 - 0에서 M-1 사이의 난수를 생성하는 `random_i()`
 - 0.0에서 1.0 사이의 난수를 생성하는 `random_f()`
- 0부터 100사이의 난수를 몇 개 생성해보자.

19 85 58 63 17 67 6 7 12 42

예제

random.c

```
#define SEED 17
int MULT = 25173;
int INC = 13849;
int MOD = 65536;

static unsigned int seed = SEED; // 난수 생성 시드값

// 정수 난수 생성 함수
unsigned random_i(void)
{
    seed = (MULT * seed + INC) % MOD; // 난수의 시드값 설정
    return seed;
}

// 실수 난수 생성 함수
double random_f(void)
{
    seed = (MULT * seed + INC) % MOD; // 난수의 시드값 설정
    return seed / (double)MOD; // 0.0에서 1.0 사이로 제한
}
```

예제

main.c

```
#include <stdio.h>

extern unsigned random_i(void);
extern double random_f(void);

extern int MOD;

int main(void)
{
    int i;

    MOD = 32767;
    for (i = 0; i < 10; i++)
        printf("%d ", random_i());

    return 0;
}
```

가변 매개 변수

- 매개 변수의 개수가 가변적으로 변할 수 있는 기능

```
int sum( int num, ... )
```

호출 때 마다 매개 변수의
개수가 변경될 수 있다.

가변 매개 변수

```
#include <stdio.h>
#include <stdarg.h>
int sum( int, ... );
int main( void )
```

합은 10입니다.

```
{
    int answer = sum( 4, 4, 3, 2, 1 );
    printf( "합은 %d입니다.\n", answer );
    return( 0 );
}
```

매개 변수의 개수

```
int sum( int num, ... )
{
    int answer = 0;
    va_list argptr;
    va_start( argptr, num );
    for( ; num > 0; num-- )
        answer += va_arg( argptr, int );
    va_end( argptr );
    return( answer );
}
```

순환(recursion)이란?

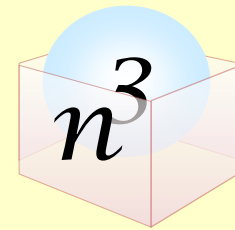
- 함수는 자기 자신을 호출할 수도 있다. 이것을 순환(recursion)라고 부른다.

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

팩토리얼 구하기

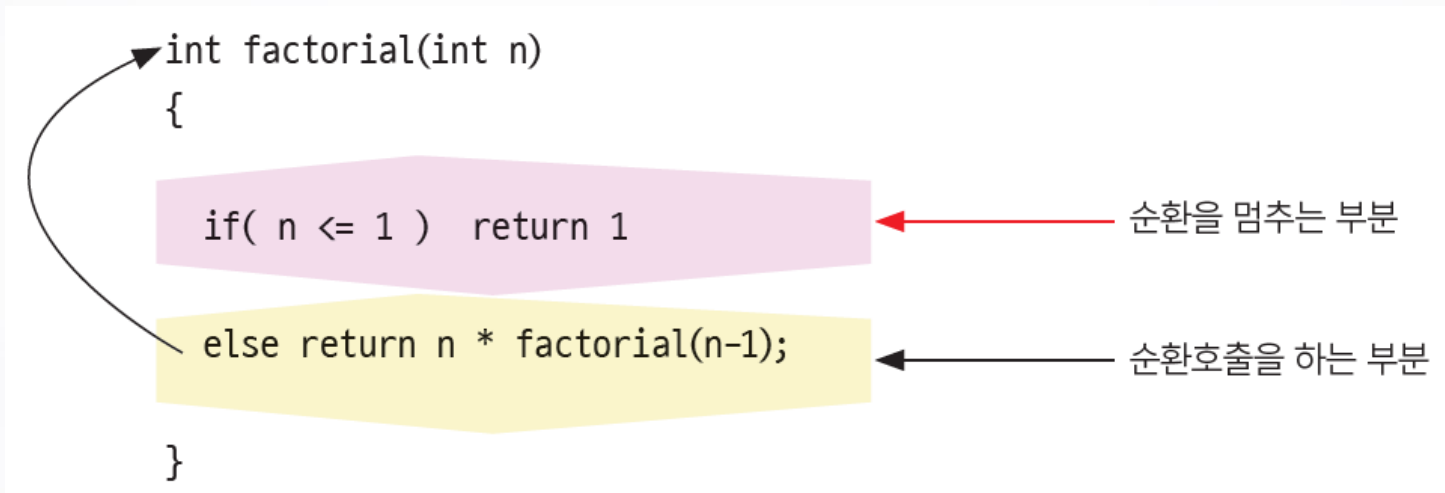
- 팩토리얼 프로그래밍: $(n-1)!$ 팩토리얼을 현재 작성중인 함수를 다시 호출하여 계산(순환 호출)

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



순환 함수의 구조

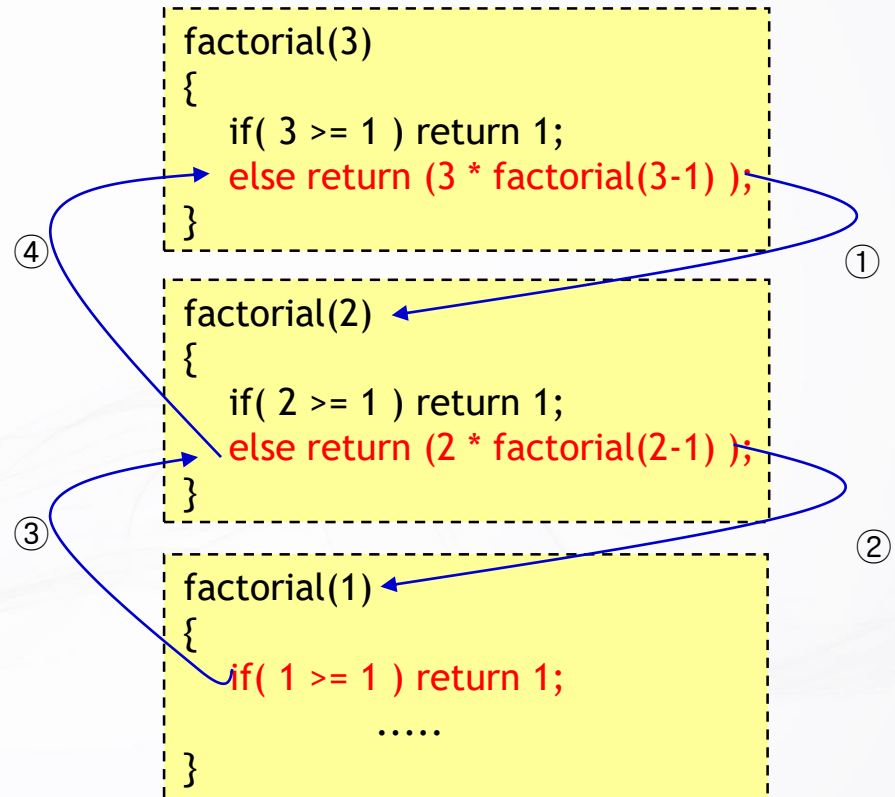
- 순환 알고리즘은 그림 9-9와 같이 자기 자신을 순환적으로 호출하는 부분과 순환 호출을 멈추는 부분으로 구성되어 있다.



팩토리얼 구하기

- 팩토리얼의 호출 순서

$\text{factorial}(3) = 3 * \text{factorial}(2)$
 $= 3 * 2 * \text{factorial}(1)$
 $= 3 * 2 * 1$
 $= 3 * 2$
 $= 6$



팩토리얼 계산

```
// 재귀적인 팩토리얼 함수 계산
#include <stdio.h>

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

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

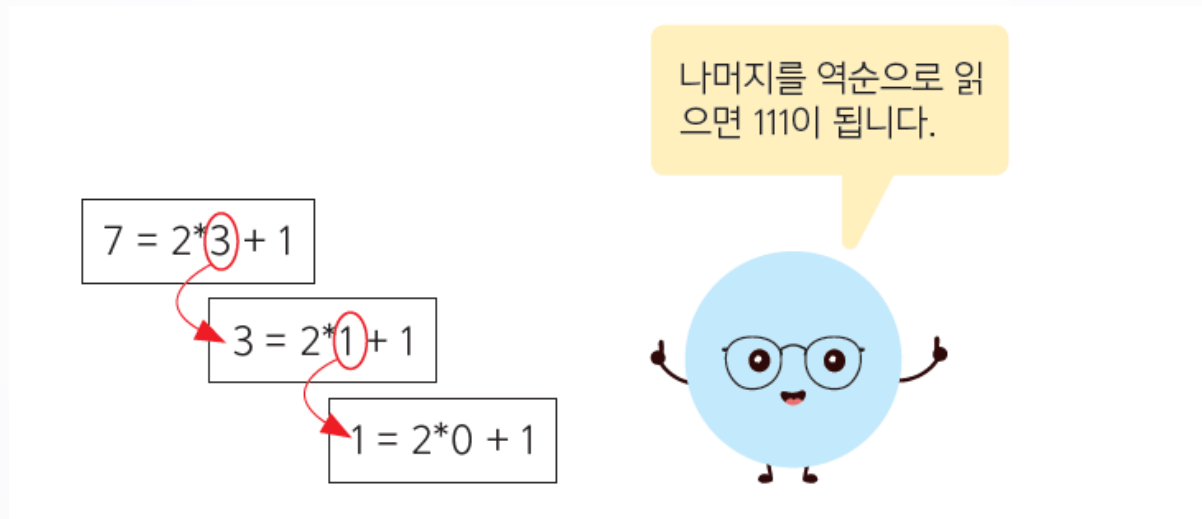
int main(void)
{
    int x = 0;
    long f;

    printf("정수를 입력하시오:");
    scanf("%d", &n);
    printf("%d!은 %d입니다. \n", n, factorial(n));
    return 0;
}
```

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

2진수 형식으로 출력하기

- C에는 정수를 2진수로 출력하는 기능이 없다. 이 기능을 순환 호출을 이용하여 구현하여 보자.



2진수 형식으로 출력하기

```
// 2진수 형식으로 출력  
#include <stdio.h>
```

```
void print_binary(int x);
```

```
int main(void)
```

```
{  
    print_binary(9);  
    printf("\n");  
    return 0;  
}
```

```
void print_binary(int x)
```

```
{  
    if (x > 0)  
    {  
        print_binary(x / 2); // 재귀 호출  
        printf("%d", x % 2); // 나머지를 출력  
    }  
}
```

1001

최대 공약수 구하기

- 최대 공약수를 구하는 방법으로 유클리드의 호제법이라는 방법이 있다. 이 방법은 두 수 x 와 y 의 최대 공약수는 y 와 $(x \% y)$ 의 최대 공약수와 같으며 x 와 0 의 최대 공약수는 x 라는 것이다

$$\text{gcd}(x, y) = \text{gcd}(y, x \% y)$$

$$\text{gcd}(x, 0) = x$$

최대 공약수 구하기

```
// 최대 공약수 구하기
#include <stdio.h>

int gcd(int x, int y);

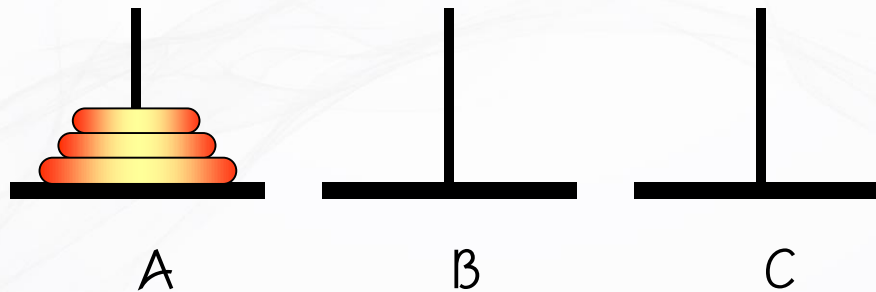
int main(void)
{
    printf("%d\n", gcd(30, 20));
}

// x는 y보다 커야 한다.
int gcd(int x, int y)
{
    if (y == 0)
        return x;
    else
        return gcd(y, x % y);
}
```

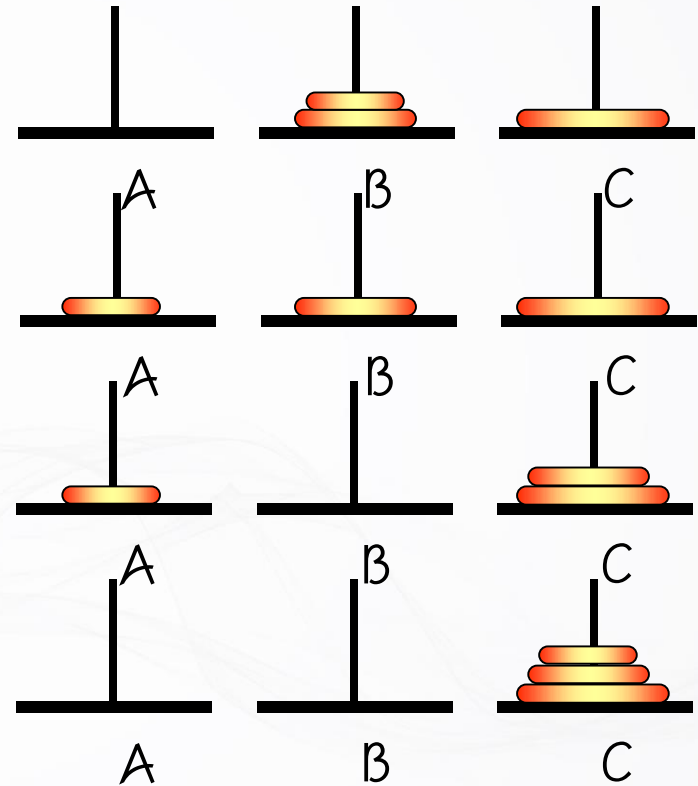
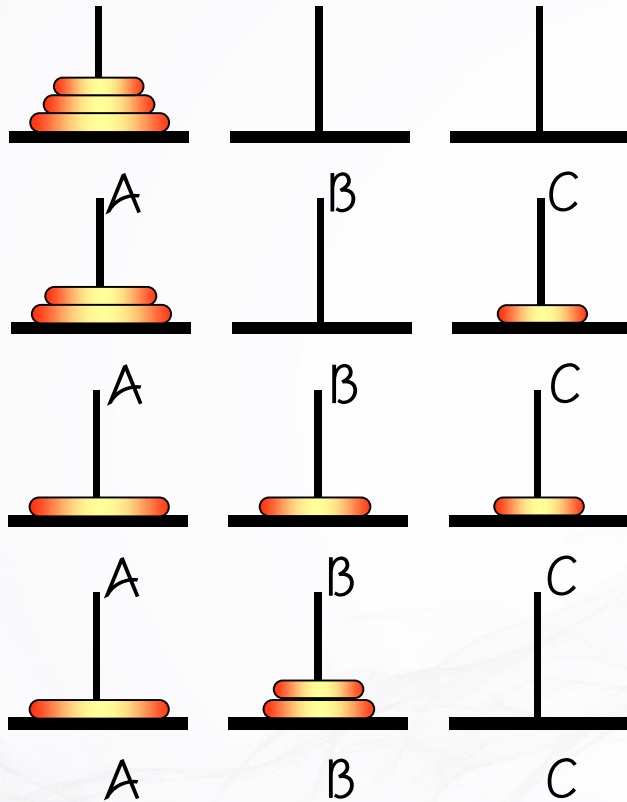
10

Mini Project: 하노이 탑 문제

- 문제는 막대 A에 쌓여있는 원판 3개를 막대 C로 옮기는 것이다. 단 다음의 조건을 지켜야 한다.
 - 한 번에 하나의 원판만 이동할 수 있다
 - 맨 위에 있는 원판만 이동할 수 있다
 - 크기가 작은 원판 위에 큰 원판이 쌓일 수 없다.
 - 중간 막대를 임시적으로 이용할 수 있으나 앞의 조건들을 지켜야 한다.

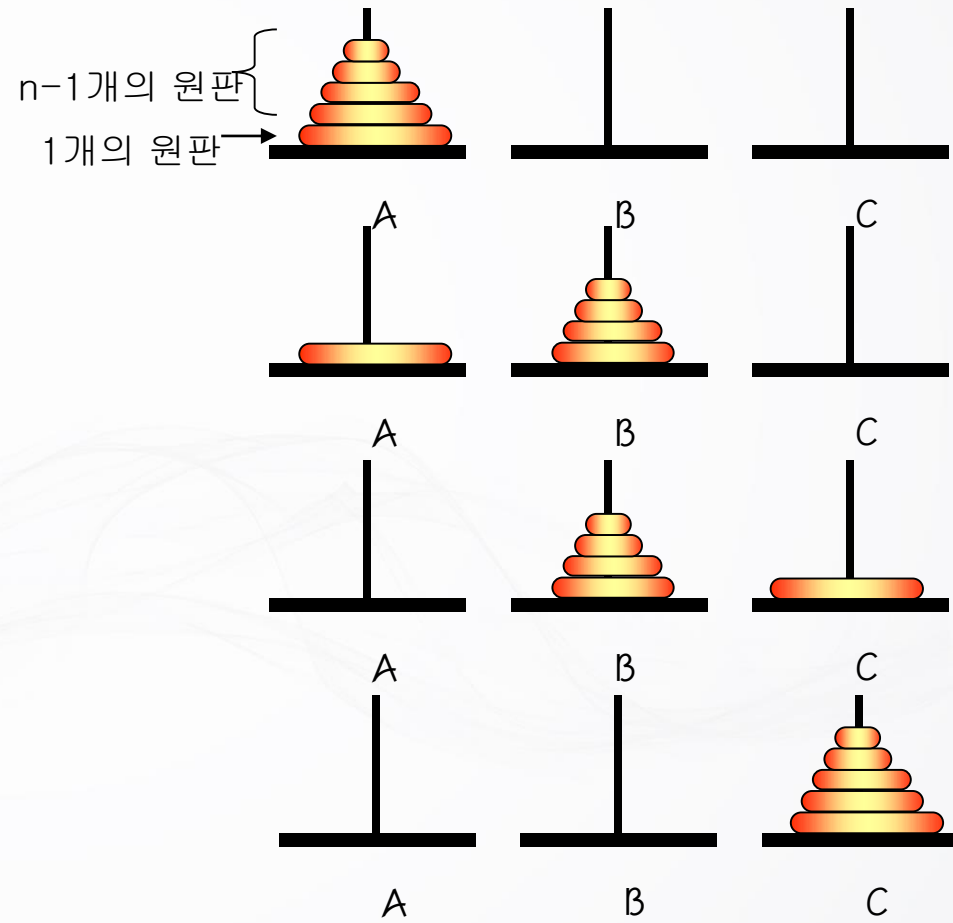


3개의 원판인 경우의 해답



n개의 원판인 경우

- $n-1$ 개의 원판을 A에서 B로 옮기고 n 번째 원판을 A에서 C로 옮긴 다음, $n-1$ 개의 원판을 B에서 C로 옮기면 된다.



하노이탑 알고리즘

```
1. // 막대 from에 쌓여있는 n개의 원판을 막대 tmp를 사용하여 막대 to로 옮긴다.  
2. void hanoi_tower(int n, char from, char tmp, char to)  
3. {  
4.     if (n == 1)  
5.     {  
6.         from에서 to로 원판을 옮긴다.  
7.     }  
8.     else  
9.     {  
10.        hanoi_tower(n-1, from, to, tmp);  
11.        from에 있는 한 개의 원판을 to로 옮긴다.  
12.        hanoi_tower(n-1, tmp, from, to);  
13.    }  
14. }
```

```
#define _CRT_SECURE_NO_WARNINGS
```

```
// 하노이의 탑 문제
```

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

```
void hanoi_tower(int n, char from, char tmp, char to);
```

```
int main(void)
```

```
{  
    hanoi_tower(4, 'A', 'B', 'C');  
}
```

```
void hanoi_tower(int n, char from, char tmp, char to)
```

```
{  
    if (n == 1)  
        printf("원판 1을 %c에서 %c으로 옮긴다.\n", from, to);  
    else  
    {  
        hanoi_tower(n - 1, from, to, tmp);  
        printf("원판 %d을 %c에서 %c으로 옮긴다.\n", n, from, to);  
        hanoi_tower(n - 1, tmp, from, to);  
    }  
}
```

원판 1을 A 에서 B으로 옮긴다.
원판 2을 A에서 C으로 옮긴다.
원판 1을 B 에서 C으로 옮긴다.
원판 3을 A에서 B으로 옮긴다.
원판 1을 C 에서 A으로 옮긴다.
원판 2을 C에서 B으로 옮긴다.
원판 1을 A 에서 B으로 옮긴다.
원판 4을 A에서 C으로 옮긴다.
원판 1을 B 에서 C으로 옮긴다.
원판 2을 B에서 A으로 옮긴다.
원판 1을 C 에서 A으로 옮긴다.
원판 3을 B에서 C으로 옮긴다.
원판 1을 A 에서 B으로 옮긴다.
원판 2을 A에서 C으로 옮긴다.
원판 1을 B 에서 C으로 옮긴다.

Q & A

