
DCSLAB

프로그래밍연습 **Lab 7**

함수와 변수

[TA] 강성민, 김기현, 최석원, 최지은, 표지원
Department of Computer Science and Engineering
Seoul National University, Korea
2022/10/26

이번 장에서 학습할 내용

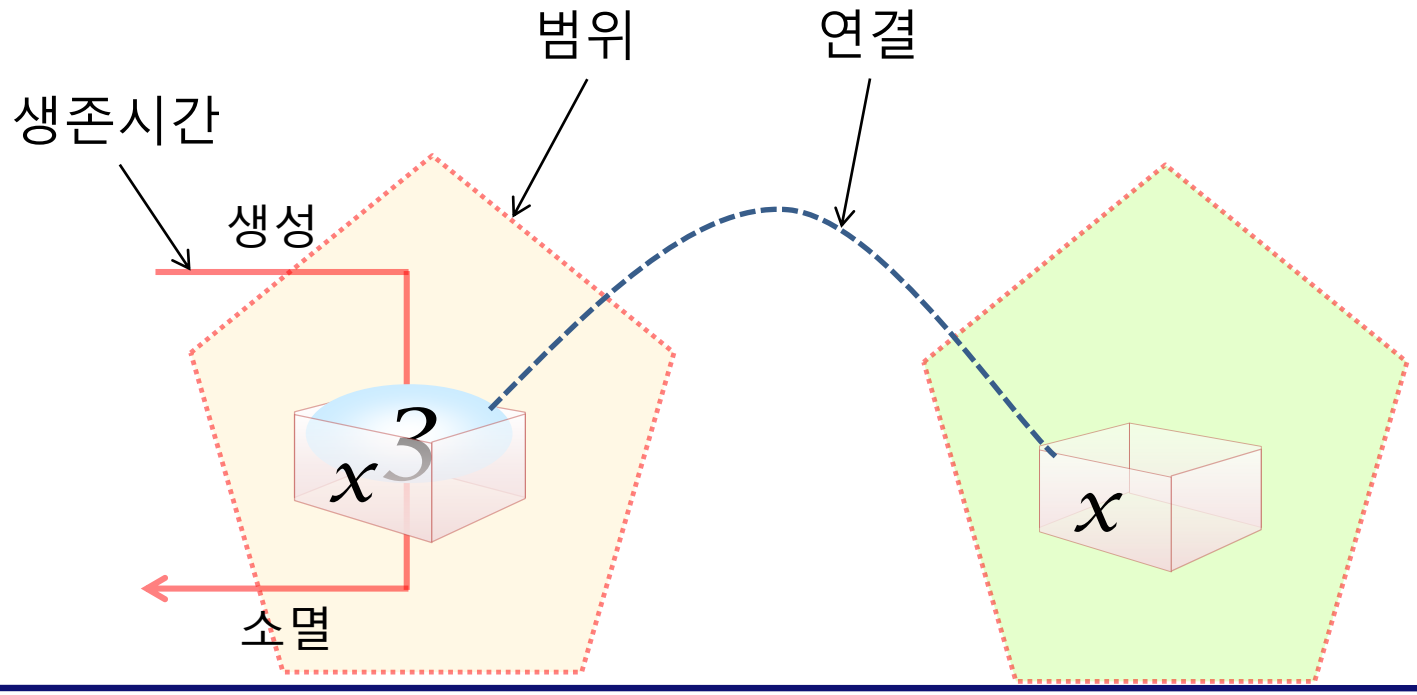
- 반복의 개념 이해
- 변수의 속성
- 전역, 지역 변수
- 자동 변수와 정적 변수
- 재귀 호출

함수와 변수와의 관계 및 함수가 자기 자신을 호출하는 재귀 호출

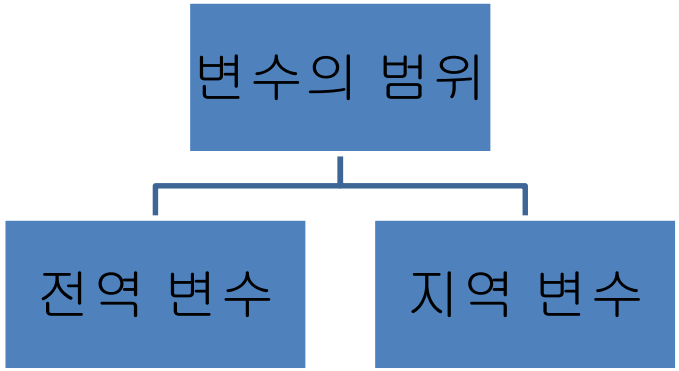
변수의 속성

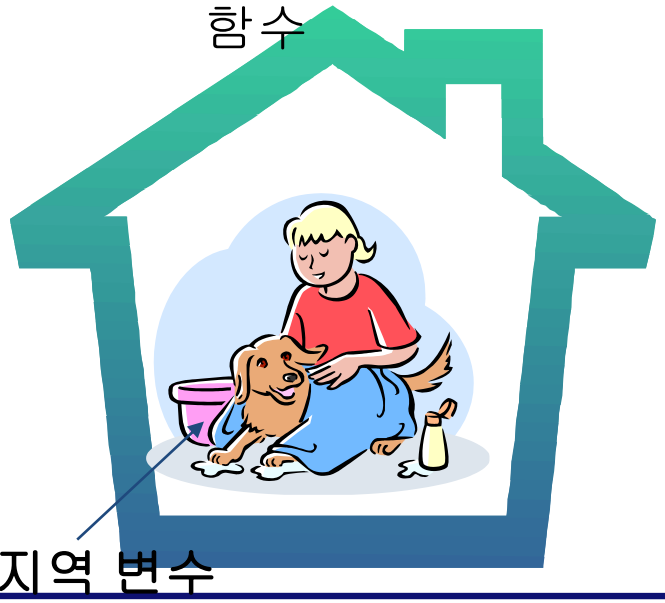
변수의 속성 : 이름, 타입, 크기, 값 + 범위, 생존 시간, 연결

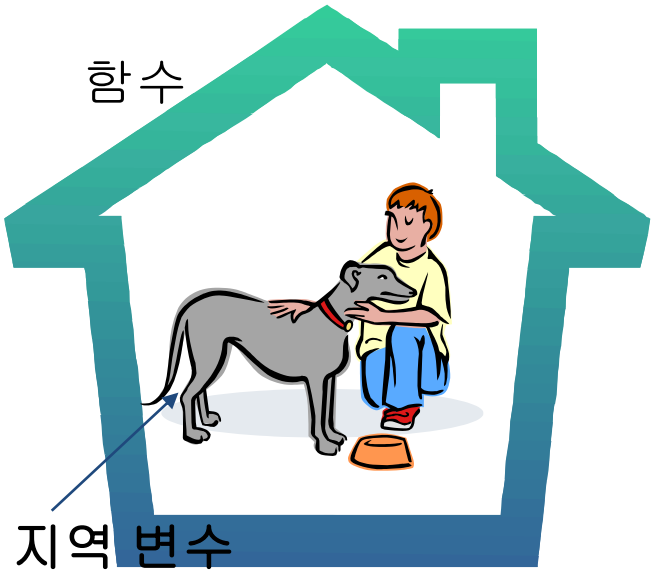
- **범위(scope)** : 변수가 사용 가능한 범위, 가시성
- **생존 시간(lifetime)** : 메모리에 존재하는 시간
- **연결(linkage)** : 다른 영역에 있는 변수와의 연결 상태

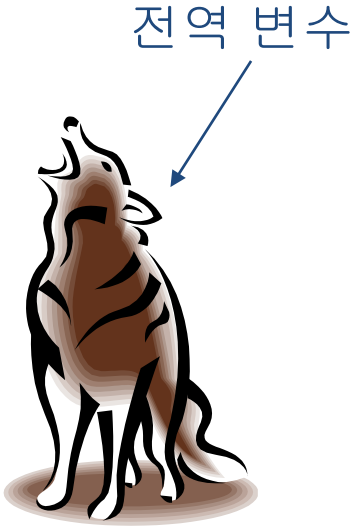


변수의 범위

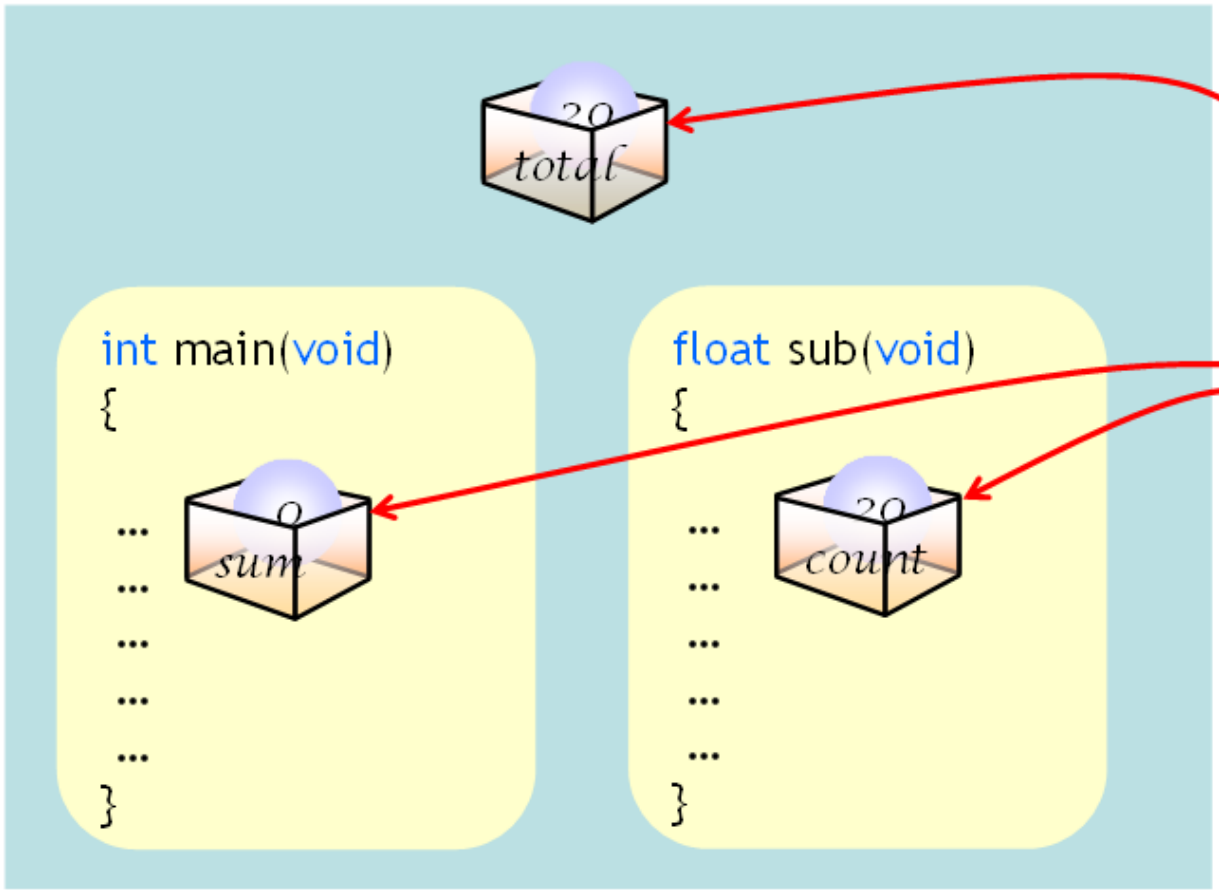








전역 변수와 지역 변수



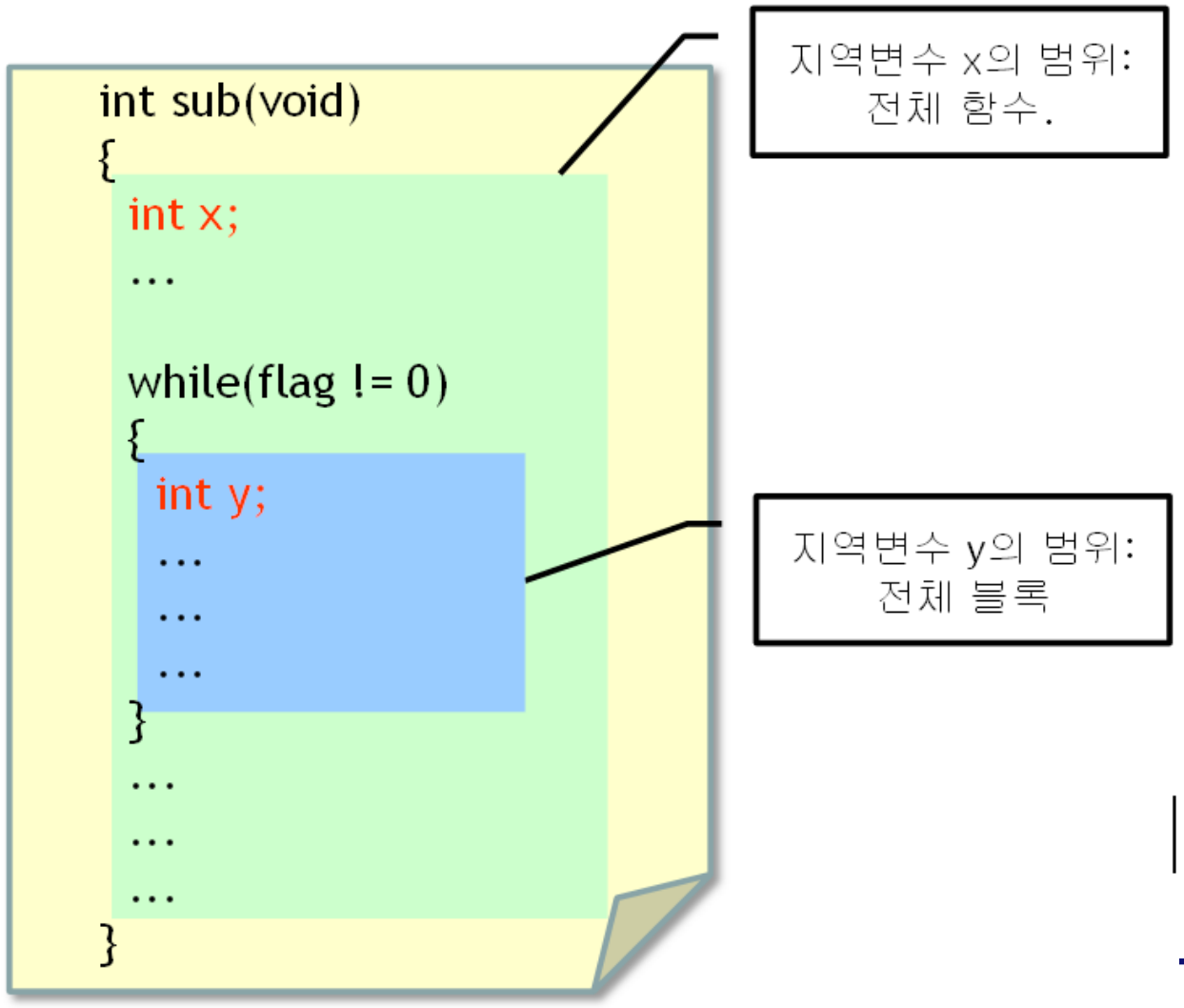
변수의 범위

전역 변수: 함수의 외부에서 정의

지역 변수: 함수의 내부에서 정의

지역 변수

지역 변수(local variable)는 블록 안에 선언되는 변수



지역 변수 선언 위치

블록 첫 부분에서 정의

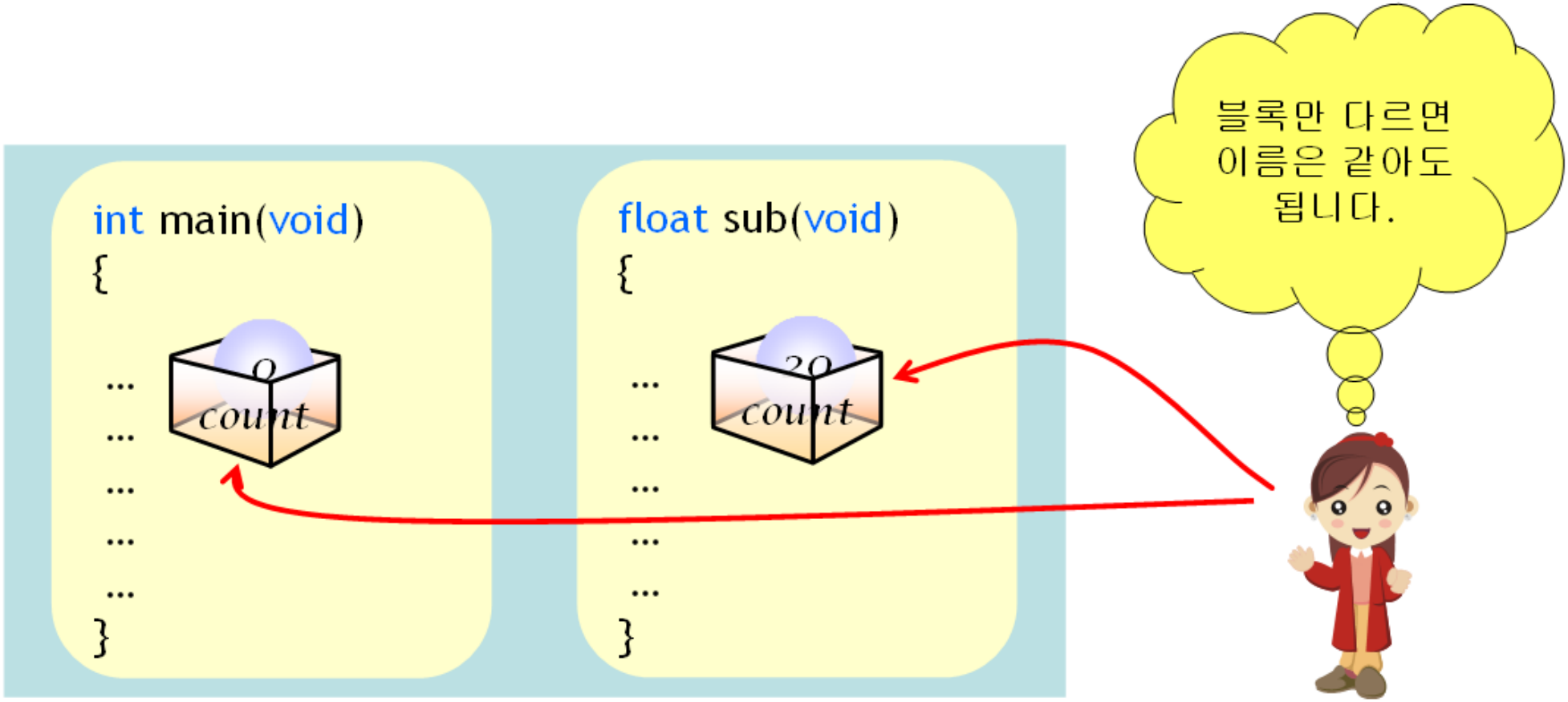
```
int sub(void)
{
        int x;      // ①
        …
        x = 100;
        int y;      // ②
}
```

변수를 함수의 첫 부분에 선언하지 않으셨군요. 컴파일 오류입니다.

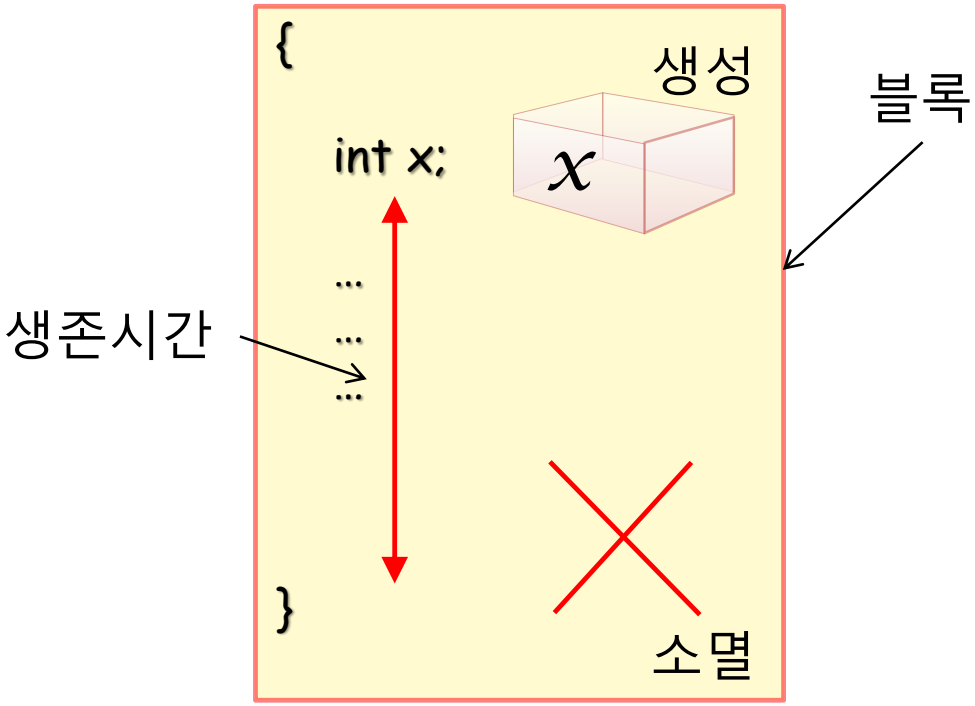
지역 변수의 범위

```
void sub1(void)
{
        {
                int y;
        …
        }
        y = 4;
}
```

이름이 같은 지역 변수


int main(void)
{
...
...
...
...
...
}
0
count
float sub(void)
{
...
...
...
...
...
}
20
count
블록만 다르면
이름은 같아도
됩니다.

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
```

지역 변수의 초기값

```
#include <stdio.h>
int main(void)
{
        int temp;
        printf("temp = %d\n", temp);
}
```

초기화 되지 않았으므로 쓰레기 값을 가진다.

함수의 매개 변수

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

매개 변수도 일종의 지역 변수

함수의 매개 변수

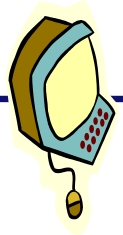
```c
#include <stdio.h>
int inc(int counter);

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

값에 의한 호출 (call by value)

매개 변수도 일종의 지역 변수임

- call-by-value 값에 의한 호출방식은 함수 호출시 전달되는 변수의 값을 복사하여 함수의 인자로 전달한다.
- 복사된 인자는 함수 안에서 지역적으로 사용되는 local value의 특성을 가진다.

```c
//두 수를 함수 내에서 바꿉니다.
void func_by_value(int a, int b)
{
        int temp = a;
        a = b;
        b = temp;
}

//두 수를 함수 내에서 바꿉니다.
void func_by_ref(int & a, int & b) //포인터로도 구현 가능
{
        int temp = a;
        a = b;
        b = temp;
}
```



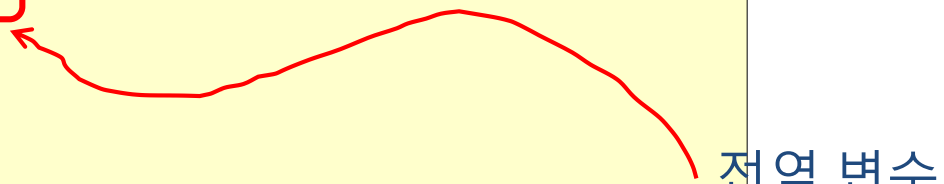
```
함수 호출전 i=10
함수 호출후 i=10
```

- 따라서 함수 안에서 인자의 값이 변경되어도, 외부의 변수의 값은 변경되지 않는다.

전역 변수

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

```
int x = 123;

void sub1()
{
        x = 456;
}

void sub2()
{
        x = 789;
}
```



전역 변수의 초기값과 생존 기간

```
#include <stdio.h>
int counter;

void set_counter()
{
        counter = 20;       // 직접 사용 가능
}
int main(void)
{
        printf("counter=%d\n", counter);
        set_counter();
        printf("counter=%d\n", counter);

        return 0;
}
```

전역 변수의 범위

전역 변수 초기값은 0

```
counter=0
counter=20
```

전역 변수의 사용

```
// 전역 변수를 사용하여 프로그램이 복잡해지는 경우
#include <stdio.h>
void f(void);

int i;
int main(void)
{
        for(i = 0;i < 5; i++)
        {
                f();        i=10 => 5가 넘어감
        }
        return 0;
}
void f(void)
{
        for(i = 0;i < 10; i++)
                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
```

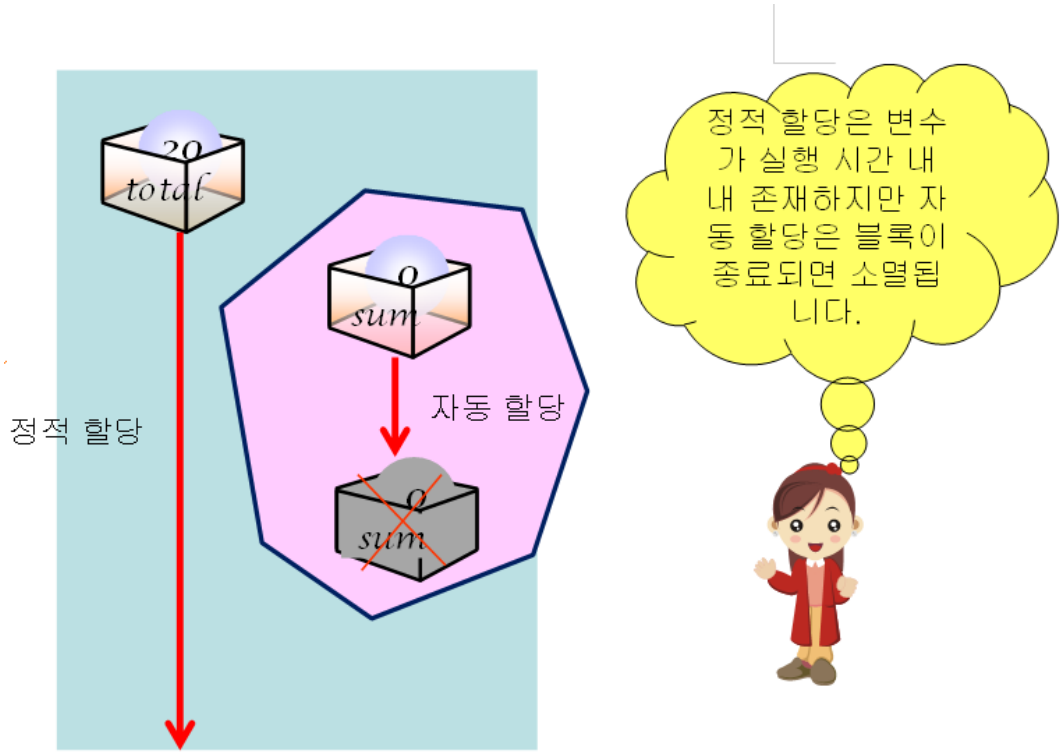
중간 점검

- 변수의 범위는 대개 무엇으로 결정되는가? 함수내부 또는 블록내부에 정의되면 지역변수, 외부에 정의되면 전역변수이다.
- 변수의 범위에는 몇 가지의 종류가 있는가? 지역변수(해당 scope 내부)/전역변수(어디서나)
- 지역 변수를 블록의 중간에서 정의할 수 있는가? 불가능
- 지역 변수가 선언된 블록이 종료되면 지역 변수는 어떻게 되는가? 지역변수가 선언된 블록이 끝나면 소멸된다.
- 지역 변수의 초기값은 얼마인가? 없다. 쓰레기값
- 전역 변수는 어디에 선언되는가? 최상단, 헤더 포함 다음. 어느 scope에도 포함되지 않음.
- 전역 변수의 생존 기간과 초기값은? 생존기간은 프로그램 시작시점부터 끝날시점까지. 초기값 0.
- 똑같은 이름의 전역 변수와 지역 변수가 동시에 존재하면 어떻게 되는가? 지역변수가 전역변수를 가린다.

생존 기간

- 정적 할당(static allocation):
  - 프로그램 실행 시간 동안 계속 유지

- 자동 할당(automatic allocation):
  - 블록에 들어갈때 생성
  - 블록에서 나올때 소멸



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 auto가 생략되어도 자동 변수가 된다.

```
int main(void)
{
  auto int sum = 0;
  int i = 0;
  ...
  ...
}
```

전부 자동 변수로서 함수가 시작되면 생성되고 끝나면 소멸된다.

저장 유형 지정자 static

```
#include <stdio.h>
void sub(void);

int main(void)
{
        int i;
        for(i = 0;i < 3; i++)
                sub();
        return 0;
}
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);
}
```

```
auto_count=1
static_count=1
auto_count=1
static_count=2
auto_count=1
static_count=3
```

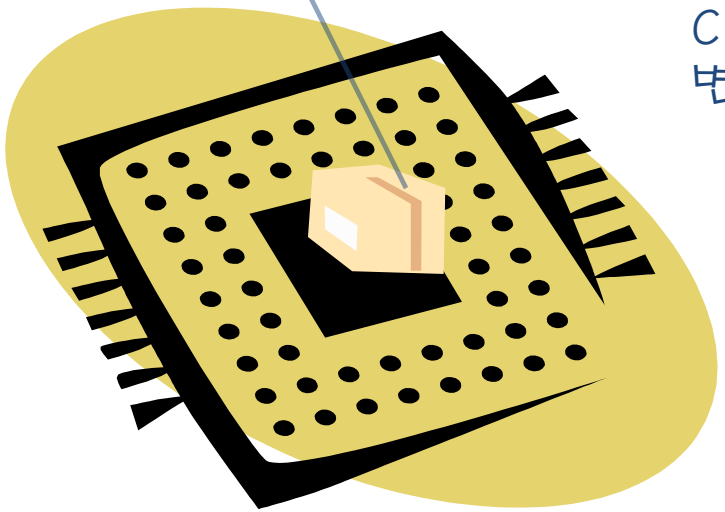
자동 지역 변수

정적 지역 변수로서 static을 붙이면 지역 변수가 정적 변수로 된다.

저장 유형 지정자 register

레지스터(register)에 변수를 저장.

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



CPU안의 레지스터에
변수가 저장됨

중간 점검

- 저장 유형 지정자에는 어떤 것들이 있는가? auto, static, register, extern
- 지역 변수를 정적 변수로 만들려면 어떤 지정자를 붙여야 하는가? Static 정적할당-
  프로그램 실행 시간 동안 계속 유지.
- 변수를 CPU 내부의 레지스터에 저장시키는 지정자는? Register 지역변수만 가능.
  &를 사용하면 오류발생. 레지스터에는 주소가 없음.
- 컴파일러에게 변수가 외부에 선언되어 있다고 알리는 지정자는? extern
- static 지정자를 변수 앞에 붙이면 무엇을 의미하는가? 지역변수가 정적변수로

실습: 로그인 횟수 제한

- 로그인 시에 제한된 횟수만큼 틀리면 로그인 시도를 막는 프로그램을 작성하여 보자.

알고리즘 & 실행결과

- *while(1)*
- *사용자로부터 아이디를 입력받는다.*
- *사용자로부터 패스워드를 입력받는다.*
- *만약 로그인 시도 횟수가 일정 한도를 넘었으면 프로그램을 종료한다.*
- *아이디와 패스워드가 일치하면 로그인 성공 메시지를 출력한다.*
- *아이디와 패스워드가 일치하지 않으면 다시 시도한다.*

시도 횟수는 3번
exit(1); 으로 종료

int check(int id, int password)함수
return 값 성공 1, fail 2,

소스

```c
#include <stdio.h>
#include <stdlib.h>
#define SUCCESS 1
#define FAIL 2
#define LIMIT 3

int check(int id, int password);

int main(void)
{	int id, password, result;

	while(1) {
		printf("id: ____\b\b\b\b");
		scanf("%d", &id);
		printf("pass: ____\b\b\b\b");
		scanf("%d", &password);
		result = check(id, password);
		if( result == SUCCESS ) break;
	}
	printf("로그인 성공\n");
	return 0;
}
```

소스

```
int check(int id, int password)
{
        static int super_id = 1234;
        static int super_password = 5678;
        static int try_count = 0;

        try_count++;
        if( try_count >= LIMIT ) {
                printf("횟수 초과\n");
                exit(1);   프로그램을 종료하며 1을 반환함 (error로 종료를 알림)
        }
        if( id == super_id && password == super_password )
                return SUCCESS;
        else
                return FAIL;
}
```

정적 지역 변수

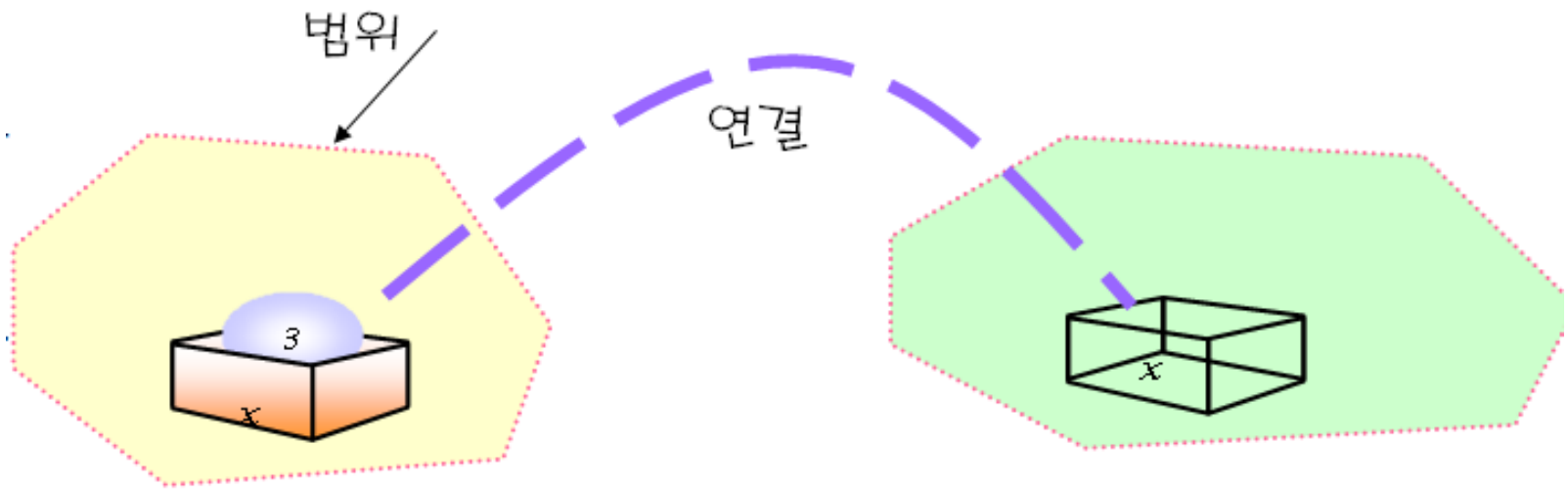
도전문제 및 과제

- 위의 프로그램은 정적 지역 변수를 사용한다. 정적 전역 변수를 사용할 수도 있다. 정적 전역 변수를 사용하도록 위의 프로그램을 변경하여 보자.
- 아이디와 패스워드에 대하여 시도 횟수를 다르게 하여 보자(2회 이상).
- 은행 입출금 시스템을 간단히 구현하고 여기에 로그인 기능을 추가하여 보자.
  - Save(int amount) 함수는 저금할 금액 amount를 받으며 save(100)과 같이 호출됨
  - Draw(int amount)은 예금 인출을 나타냄

```
Output

ID와 password를 입력해주세요.
ID:1234
PW:5555
회원정보가 일치 하지 않습니다.

ID와 password를 입력해주세요.
ID:1234
PW:5555
회원정보가 일치 하지 않습니다

ID와 password를 입력해주세요.
ID:1234
PW:5555
회원정보가 일치 하지 않습니다.
입력 횟수가 초과하였습니다.

ID와 password를 입력해주세요.
ID:1234
PW:5678
로그인에 성공하였습니다.

메뉴를 선택하세요: 저축(1), 인출(2), 종료(3):
4
잘못 입력하셨습니다.
1
저축할 금액:
100
현재 잔액은 100원입니다.
2
인출한 금액
200
잔액이 부족합니다.
현재 잔액은 100원입니다.
2
인출할 금액
100
현재 잔액은 0원입니다.
3
은행프로그램 종료
```

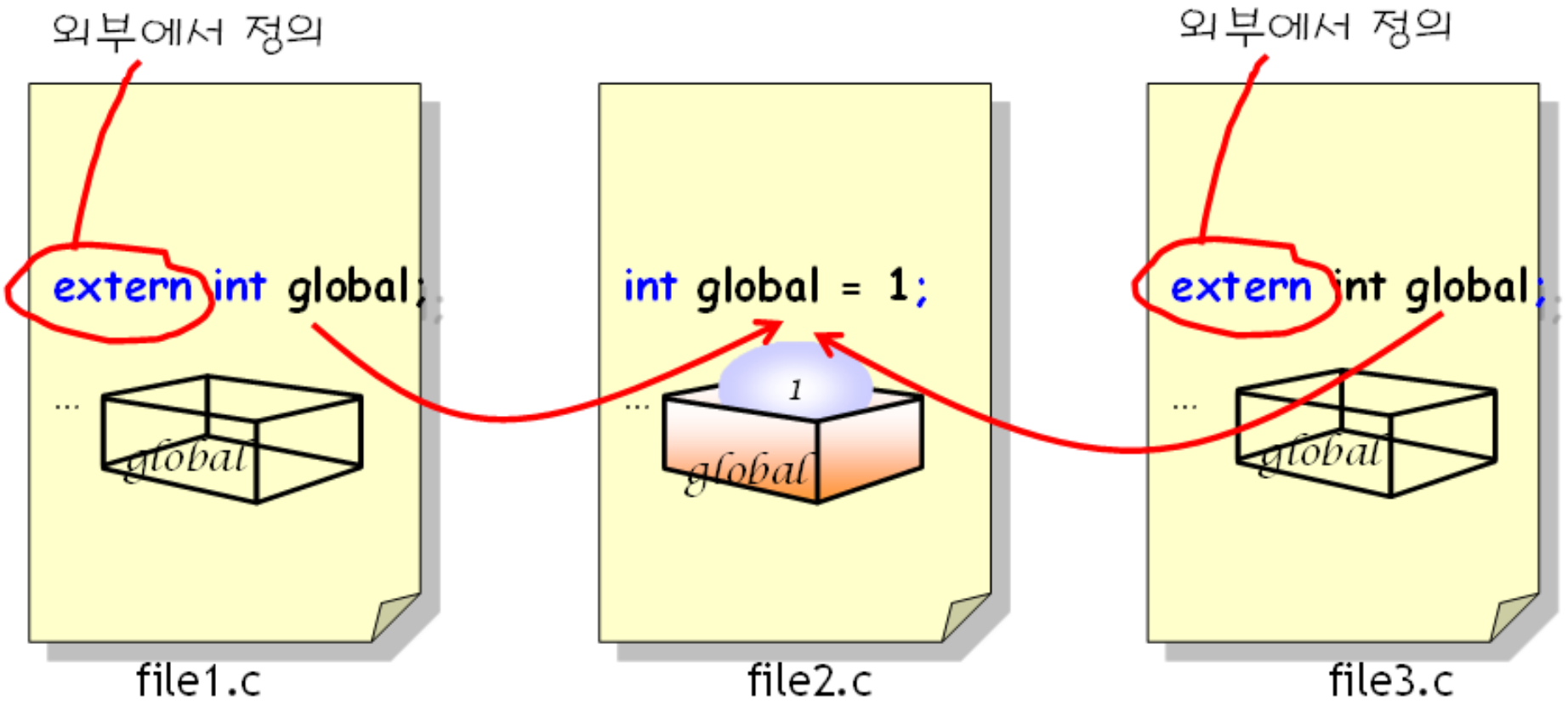
연결

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



외부 연결

- 전역 변수를 extern을 이용하여서 서로 연결



연결 예제

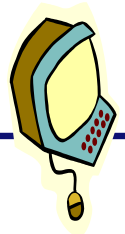
linkage1.c

```c
#include <stdio.h>
int all_files;      // 다른 소스 파일에서도 사용할 수 있는 전역 변수
static int this_file; // 현재의 소스 파일에서만 사용할 수 있는 전역 변수
extern void sub();

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

연결

linkage2.c

```c
extern int all_files;
void sub(void)
{
        all_files = 10;
}
```



저장 유형 지정자 extern

extern1.c

```
#include <stdio.h>
int x;              // 전역 변수
extern int z;       // 다른 소스 파일의 변수
extern int y;       // 현재 소스 파일의 뒷부분에 선언된 변수 전역변수를 참조(내부연결)
int main(void)
{
        extern int x;   // 전역 변수 x를 참조한다. 없어도 된다.

        x = 10;
        y = 20;
        z = 30;
        return 0;
}
int y;          // 전역 변수
```

extern2.c

```
int z;
```

함수앞의 static

main.c

```c
#include <stdio.h>

extern void f2();
int main(void)
{
        f2();
        return 0;
}
```

Static이 붙는 함수는 파일 안에서만 사용할 수 있다.

Sub.c 파일에서만 사용가능함

sub.c

```c
static void f1()
{
        printf("f1()이 호출되었습니다.\n");
}
void f2()
{
        f1();
        printf("f2()가 호출되었습니다.\n");
}
```

저장 유형 정리

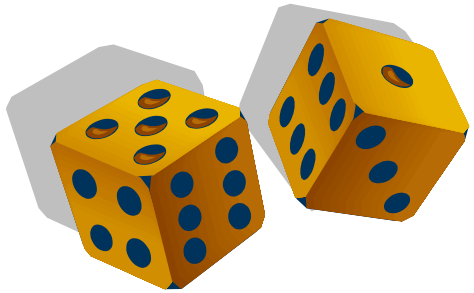
- 일반적으로는 *자동 저장 유형* 사용 권장
- 자주 사용되는 변수는 *레지스터 유형*
- 변수의 값이 함수 호출이 끝나도 그 값을 유지하여야 할 필요가 있다면 *지역 정적*
- 만약 많은 함수에서 공유되어야 하는 변수라면 *외부 참조 변수*

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 함수내부에 선언
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

예제: 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

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



예제

main.c

```c
#include <stdio.h>
unsigned random_i(void);
double random_f(void);

extern unsigned call_count;      // 외부 참조 변수

int main(void)
{
      register int i;            // 레지스터 변수

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

      printf("\n");

      for(i = 0; i < 10; i++)
            printf("%f ", random_f());

      printf("\n함수가 호출된 횟수= %d \n", call_count);
      return 0;
}
```

예제

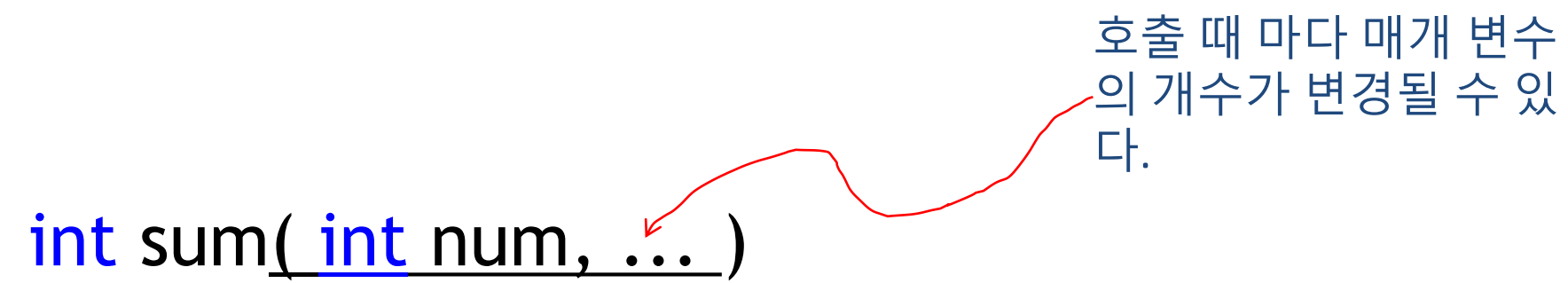
random.c

```
// 난수 발생 함수
#define SEED 17
#define MULT 25173
#define INC  13849
#define MOD  65536

unsigned int call_count = 0;      // 전역 변수
static unsigned seed = SEED;      // 정적 전역 변수

unsigned random_i(void)
{
        seed = (MULT*seed + INC) % MOD;
        call_count++;
        return seed;
}
double random_f(void)
{
        seed = (MULT*seed + INC) % MOD;
        call_count++;
        return seed / (double) MOD;
}
```

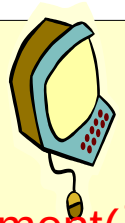
48574 61999 40372 31453 39802 35227 15504 29161
14966 52039
0.885437 0.317215 0.463654 0.762497 0.546997 0.768570
0.422577 0.739731 0.455627 0.720901
함수가 호출된 횟수 = 20

가변 매개 변수

- 매개 변수의 개수가 가변적으로 변할 수 있는 기능



가변 매개 변수



합은 10입니다.

va 는 여기 서 variable-argument(가변 매개 변수)

매개 변수의 개수

```
#include <stdio.h>
#include <stdarg.h>
int sum( int, ... );
int main( void )
{
        int answer = sum( 4, 4, 3, 2, 1 );
        printf( "합은 %d입니다.\n", answer );
        return( 0 );
}
int sum( int num, ... )
{       int answer = 0;
        va_list argptr;
        va_start( argptr, num );   첫번째 가변인수(4)를 가리 킬수 있도록 초기화
        for( ; num > 0; num-- )
                answer += va_arg( argptr, int ); 반복문을 돌리면서 다음 인자값을 받음
        va_end( argptr );
        return( answer );
}
```

순환(recursion)이란?

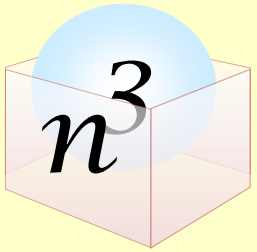
- 알고리즘이나 함수가 수행 도중에 자기 자신을 다시 호출하여 문제를 해결하는 기법
- 팩토리얼의 정의

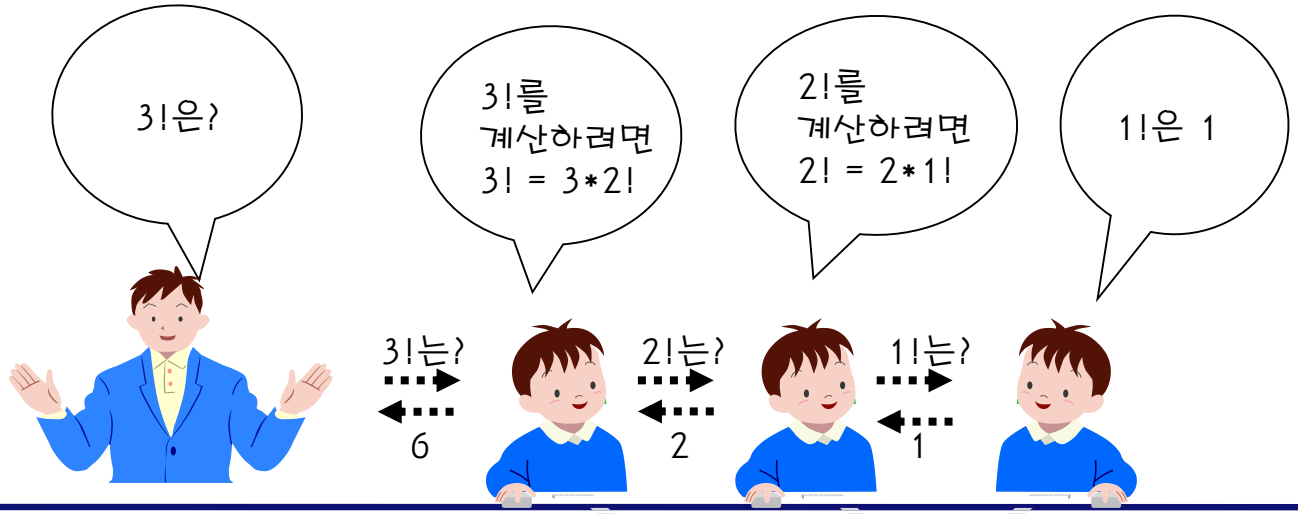
$$n! = \begin{cases} 1 & n = 1 \\ n*(n-1)! & n \geq 2 \end{cases}$$

팩토리얼 구하기

- 팩토리얼 프로그래밍 #2: (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) );
}
```

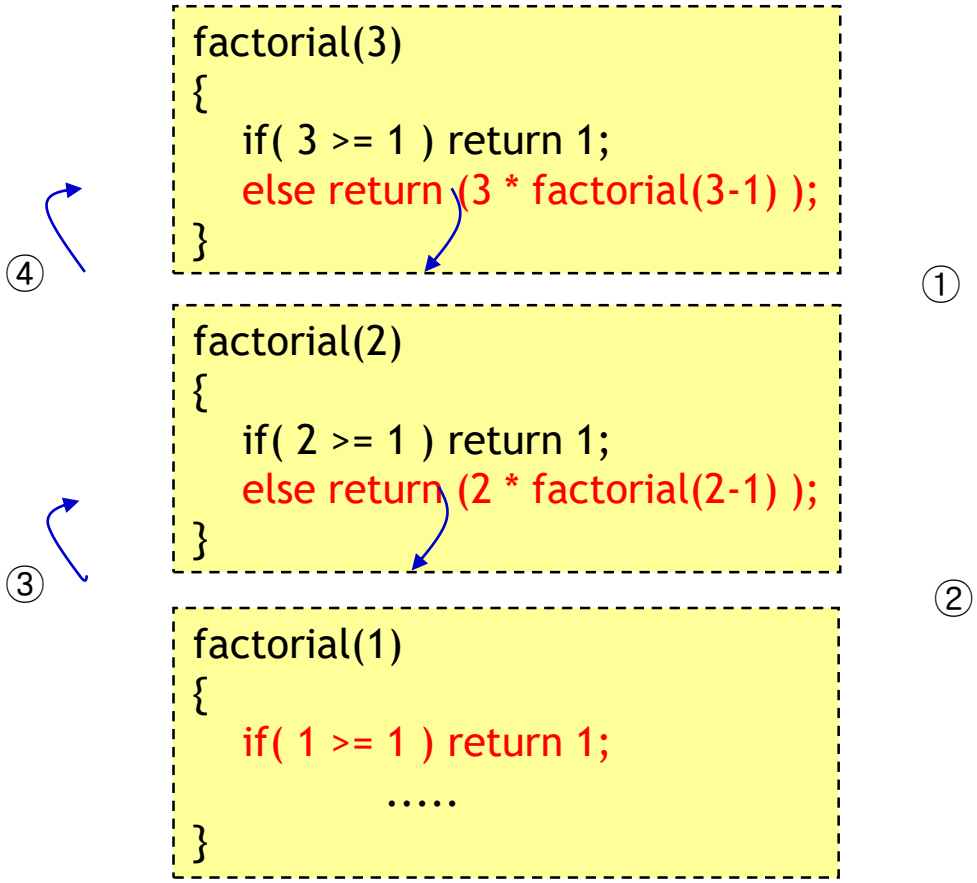




팩토리얼 구하기

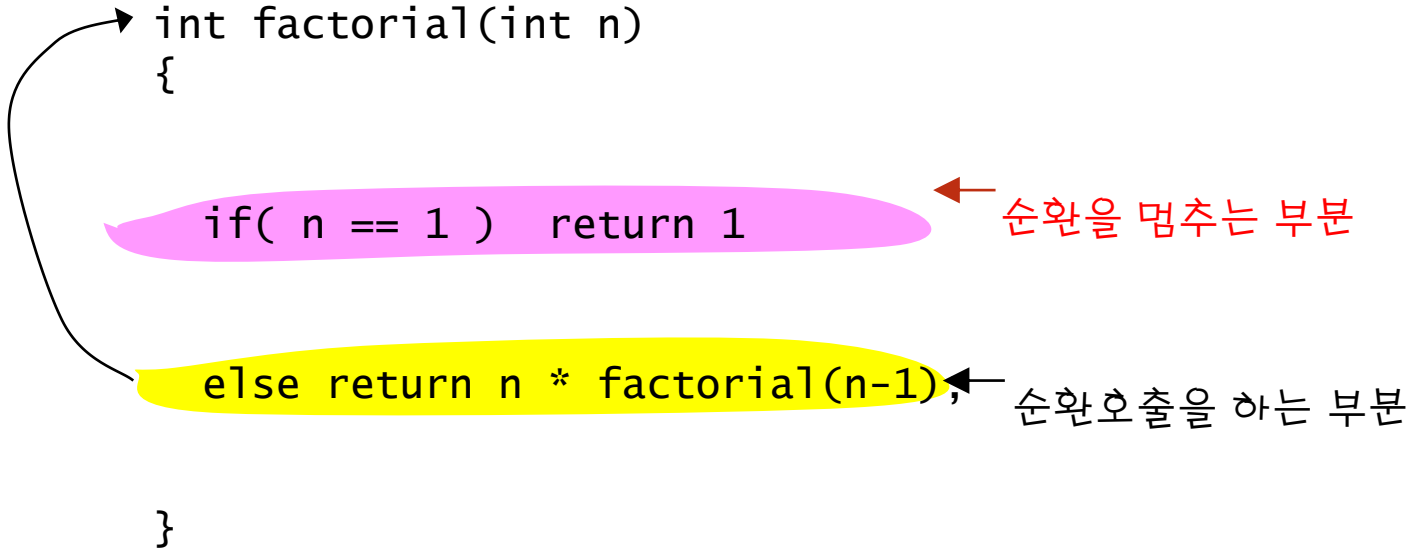
팩토리얼의 호출 순서

factorial(3) = 3 * factorial(2)
= 3 * 2 * factorial(1)
= 3 * 2 * 1
= 3 * 2
= 6



순환 알고리즘의 구조

- 순환 알고리즘은 다음과 같은 부분들을 포함한다.
  - 순환 호출을 하는 부분
  - 순환 호출을 멈추는 부분



- 만약 순환 호출을 멈추는 부분이 없다면?.
  - 시스템 오류가 발생할 때까지 무한정 호출하게 된다.

피보나치 수열의 계산 #1

- 순환 호출을 사용하면 비효율적인 예
- 피보나치 수열

  0,1,1,2,3,5,8,13,21,...

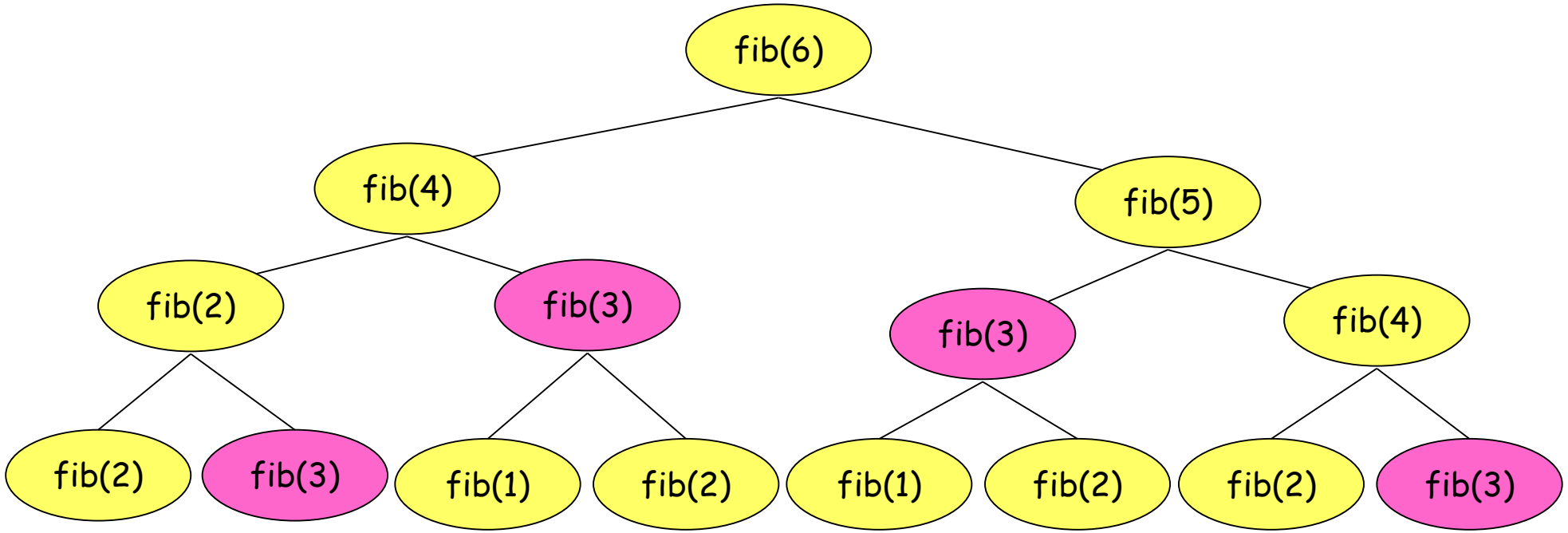
$$fib(n)\begin{cases} 0 & n=0 \\ 1 & n=1 \\ fib(n-2)+fib(n-1) & otherwise \end{cases}$$

- 순환적인 구현

```
int fib(int n)
{
    if( n==0 ) return 0;
    if( n==1 ) return 1;
    return (fib(n-1) + fib(n-2));
}
```

피보나치 수열의 계산

- 순환 호출을 사용했을 경우의 비효율성
  - 같은 항이 중복해서 계산됨
  - 예를 들어 fib(6)을 호출하게 되면 fib(3)이 4번이나 중복되어서 계산됨
  - 이러한 현상은 n이 커지면 더 심해짐



피보나치 수열의 계산(Cont'd)

피보나치의 수열의 몇 번째 값을 출력하는 프로그램 작성

```
PS C:\vscodeworkspace> gcc test.c -o test
test.c: In function 'main':
test.c:11:44: warning: implicit declaration of function 'fibo' [-Wimplicit-function-declaration]
            printf("피보나치(%d)=%d\n", i, fibo(i));
                                           ^
PS C:\vscodeworkspace> ./test
피보나치의 수열의 몇번째값까지 출력하겠습니까?:10
피보나치(0)=0
피보나치(1)=1
피보나치(2)=1
피보나치(3)=2
피보나치(4)=3
피보나치(5)=5
피보나치(6)=8
피보나치(7)=13
피보나치(8)=21
피보나치(9)=34
PS C:\vscodeworkspace>
```

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

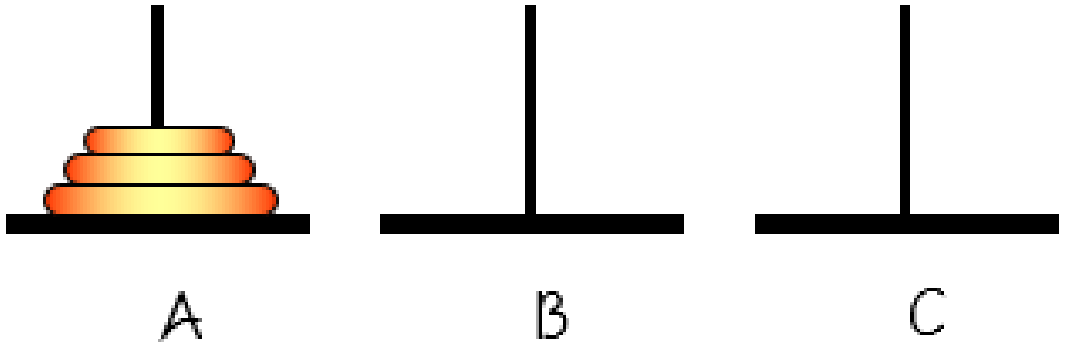
피보나치 수열의 계산(Cont'd)

test.c

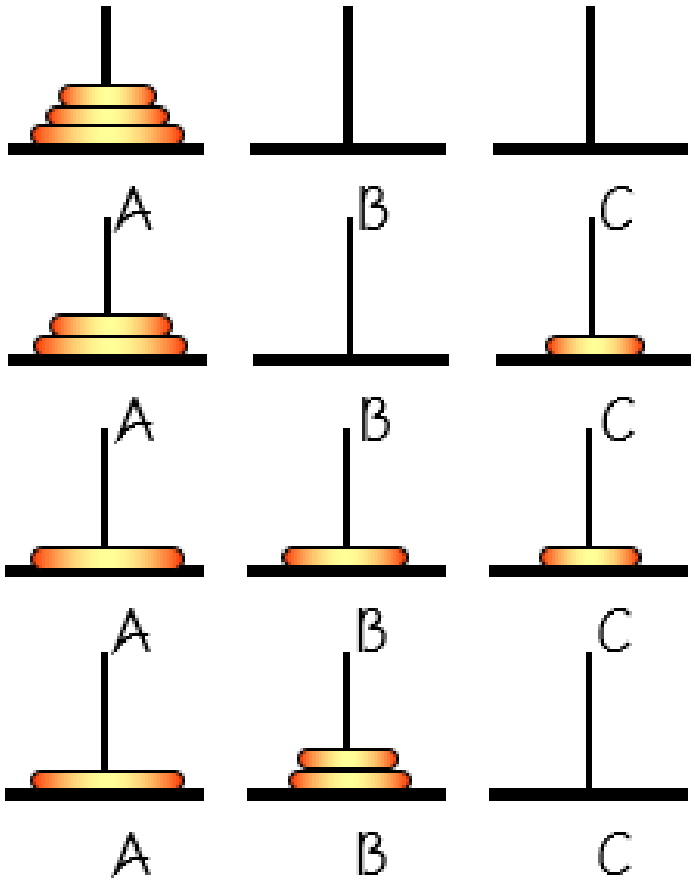
```c
#include<stdio.h>
int main()
{

	int n, i;
	printf("피보나치의 수열의 몇번째값까지 출력하겠습니까?:");
	scanf("%d", &n);

	for(i=0;i<n;i++)
	{
		printf("피보나치(%d)=%d\n", i, fibo(i));
	}


	return 0;

} //수정필요

int fibo(int n)
{
	if(n<=1)
	return n;
	else
	{
		return fibo(n-1)+ fibo(n-2);
	}

}
```

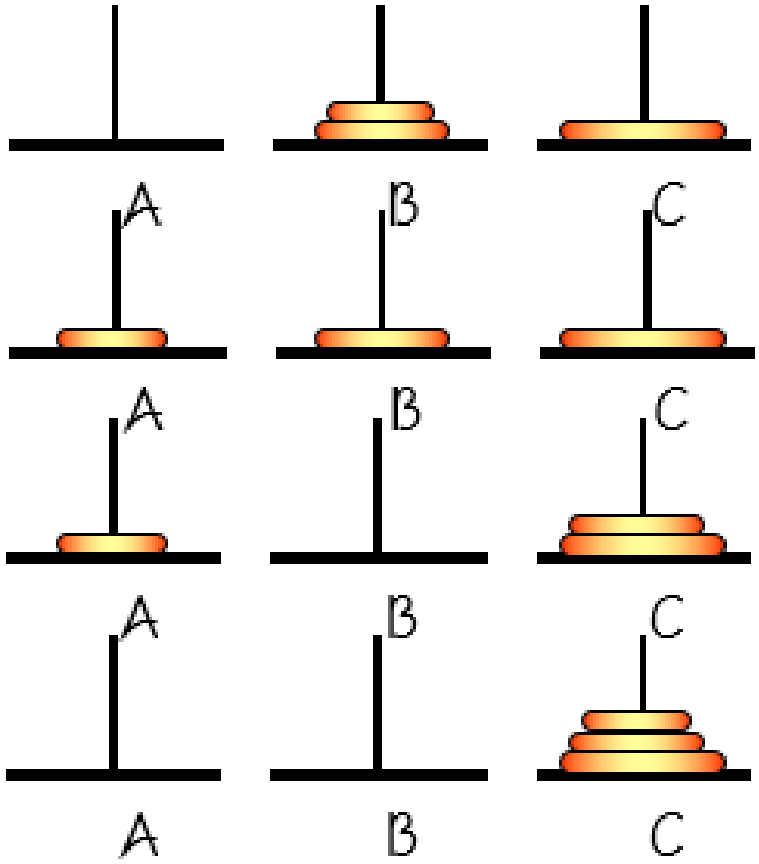
하노이 탑 문제 #1

- 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 원판위에 큰 원판이 쌓일 수 없다.
  - 중간의 막대를 임시적으로 이용할 수 있으나 앞의 조건들을 지켜야 한다.



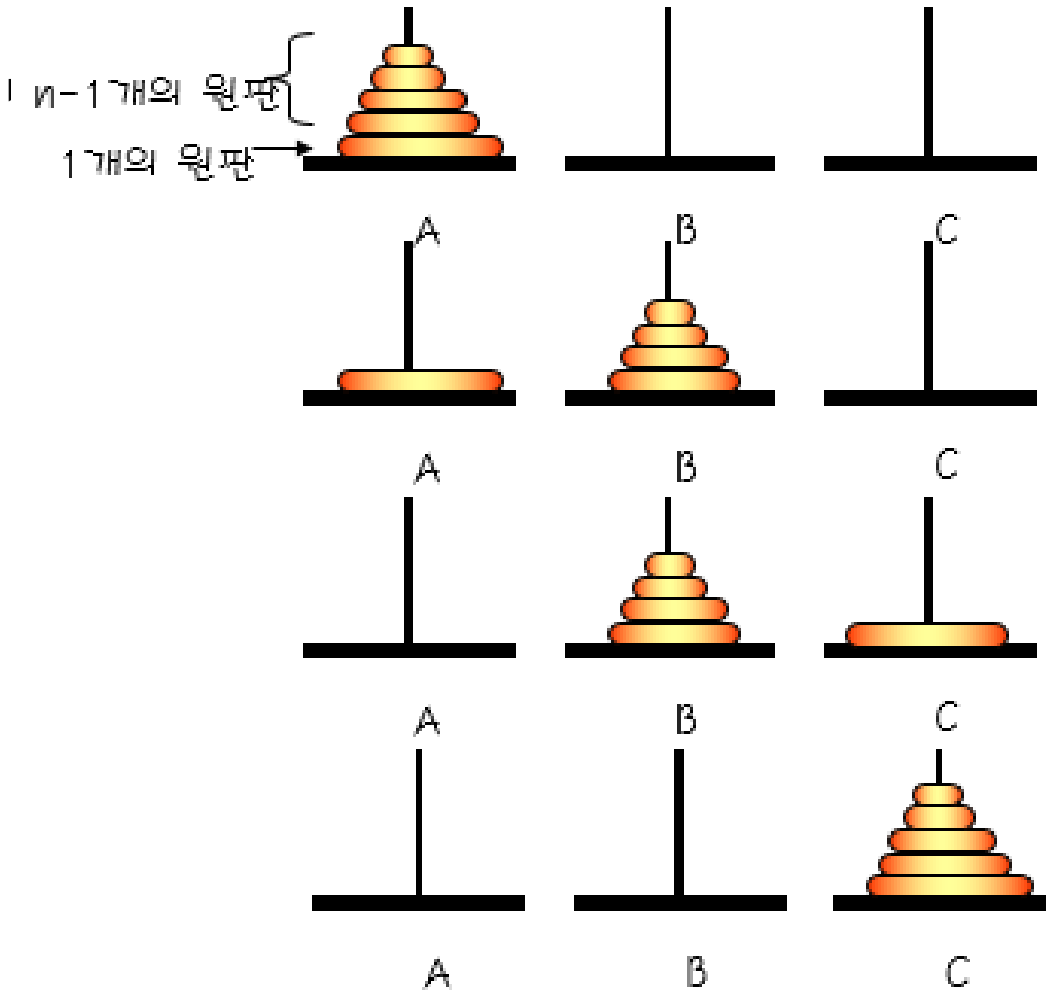
3개의 원판인 경우의 해답



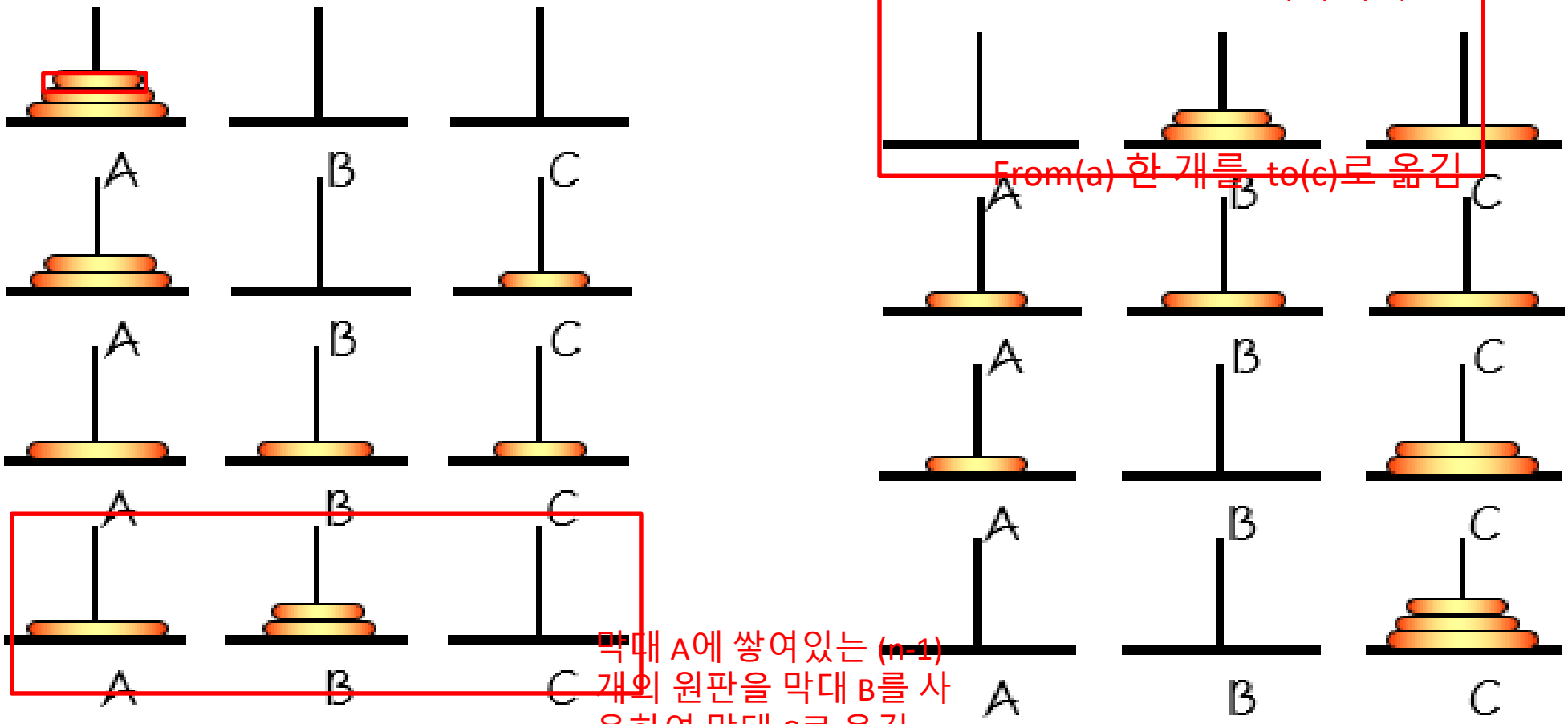


n개의 원판인 경우

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



3개의 원판인 경우의 해답



하노이탑 알고리즘

```
// 막대 from에 쌓여있는 n개의 원판을 막대 tmp를 사용하여 막대 to로 옮긴다.
void hanoi_tower(int n, char from, char tmp, char to)
{
        if (n == 1)
        {
                from에서 to로 원판을 옮긴다.
        }
        else
        {
                hanoi_tower(n-1, from, to, tmp);
                from에 있는 한 개의 원판을 to로 옮긴다.
                hanoi_tower(n-1, tmp, from, to);
        }
}
```

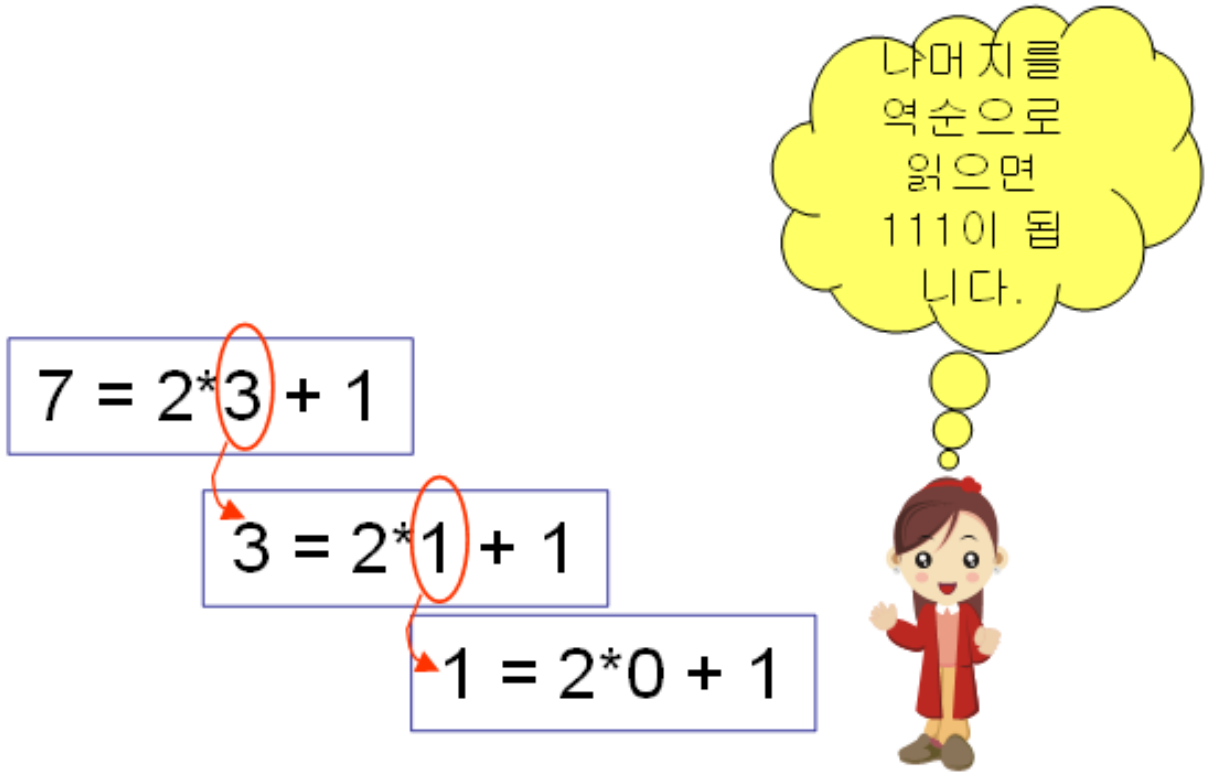
하노이탑 실행 결과

```
#include <stdio.h>

void hanoi_tower(int n, char from, char tmp, char to)
{
    if( n==1 )
            printf("원판 1을 %c 에서 %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);
  }
}


int main(void)
{
    hanoi_tower(4, 'A', 'B', 'C');
    return 0;
}
```

```
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.
```

이진수 출력하기

- 정수를 이진수로 출력하는 프로그램 작성
- 순환 알고리즘으로 가능



나누고 몫이 0일 될때 까지 나누고 나머지를 출력

순환 호출 예제

```
// 2진수 형식으로 출력
#include <stdio.h>
void print_binary(int x);
int main(void)
{
        print_binary(9);
        return 0;
}
void print_binary(int x)
{
        if( x > 0 )
        {
                print_binary(x / 2);        // 순환 호출
                printf("%d", x % 2);        // 나머지를 출력
        }
}
```

Q & A