

# 제 9장 함수와 변수

# 이번 장에서 학습할 내용



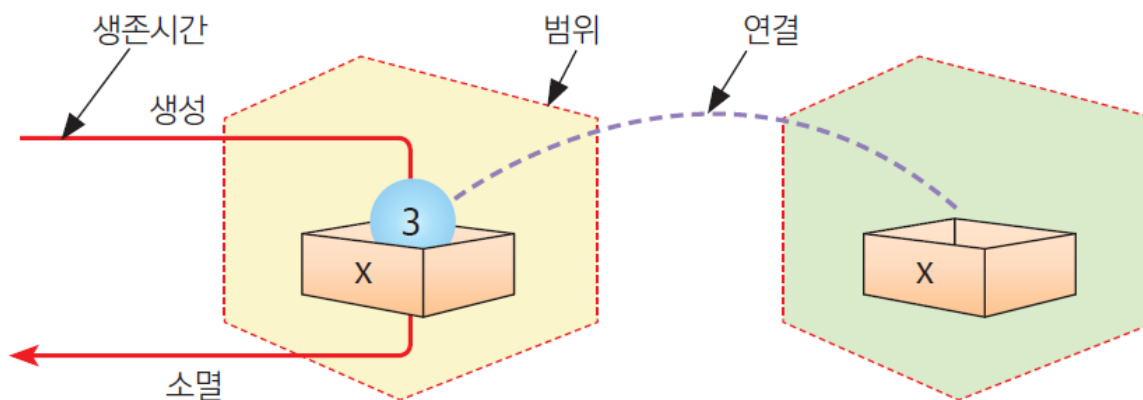
- 반복의 개념 이해
- 변수의 속성
- 전역, 지역 변수
- 자동 변수와 정적 변수
- 재귀 호출

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

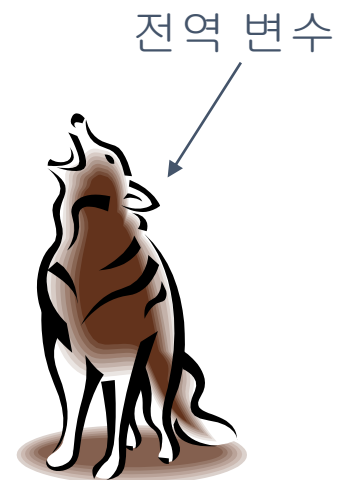
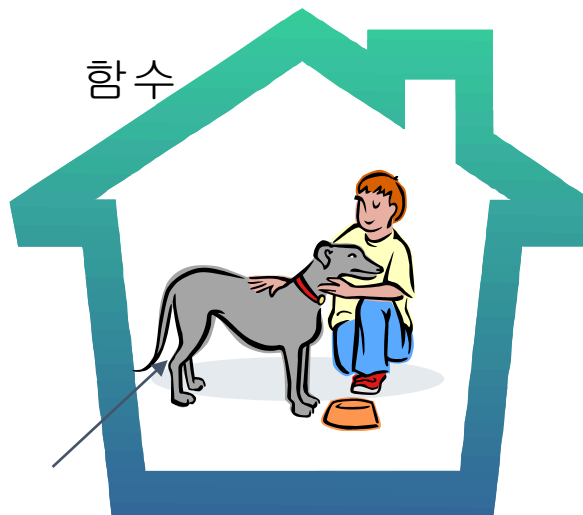
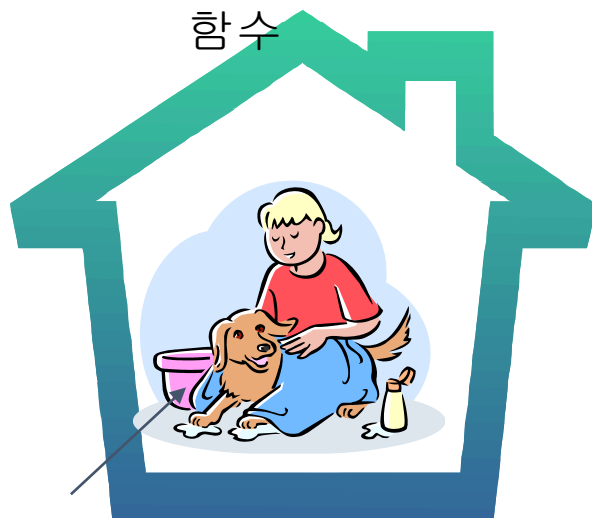
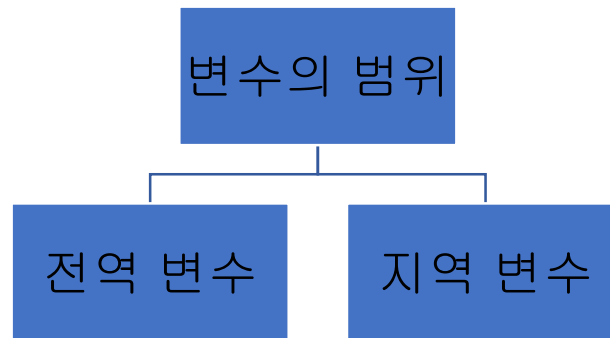


# 변수의 속성

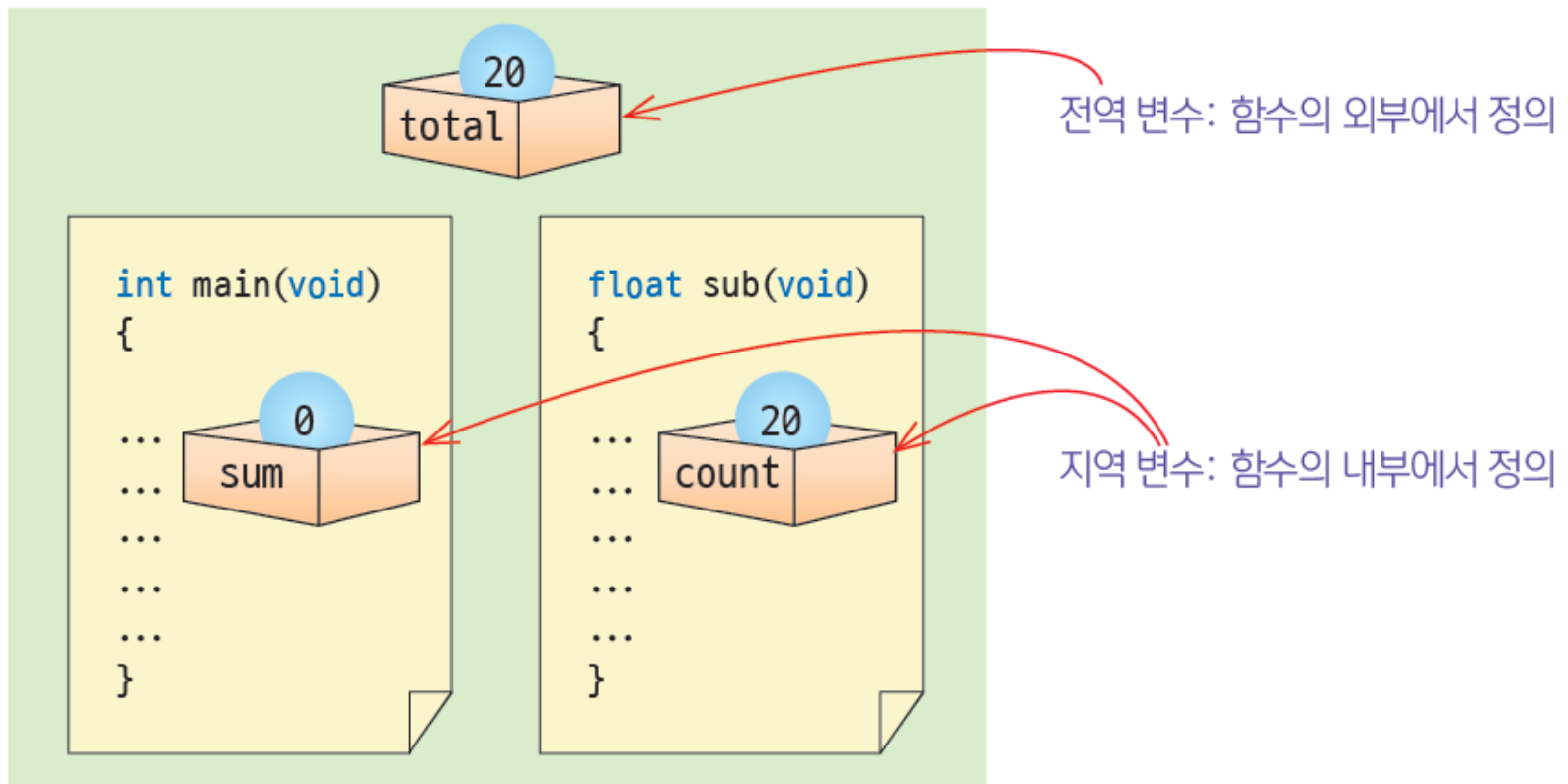
- 변수의 속성 : 이름, 타입, 크기, 값 + 범위, 생존 시간, 연결
  - 범위(scope) : 변수가 사용 가능한 범위, 가시성
  - 생존 시간(lifetime) : 메모리에 존재하는 시간
  - 연결(linkage) : 다른 영역에 있는 변수와의 연결 상태



# 변수의 범위

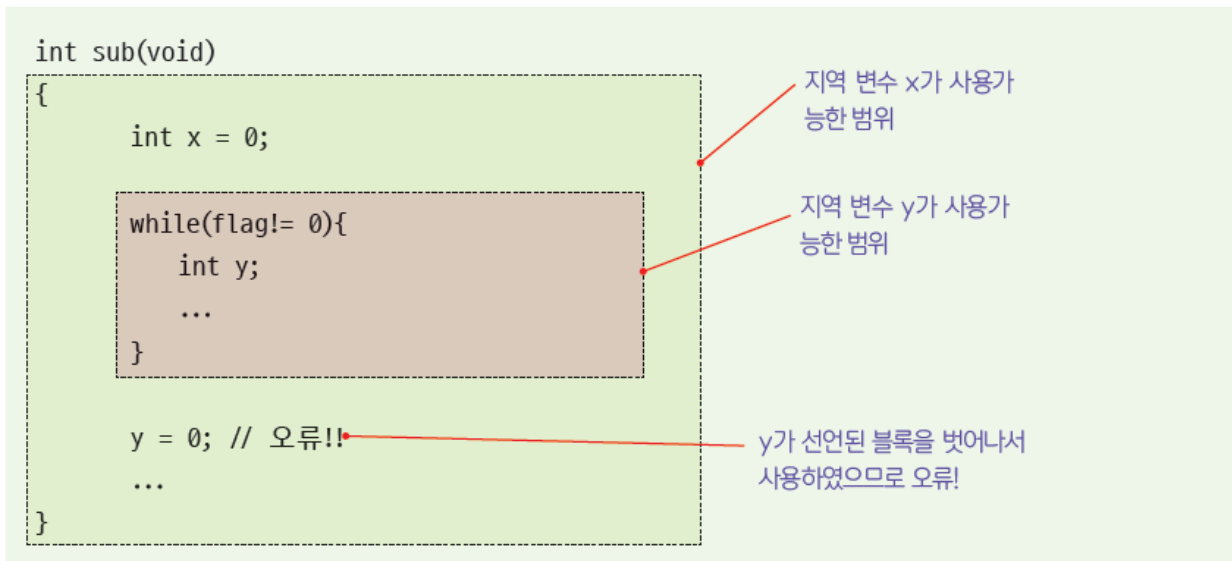


# 전역 변수와 지역 변수



# 지역 변수

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



지역 변수는 선언된 블록을 떠나면 안됩니다.



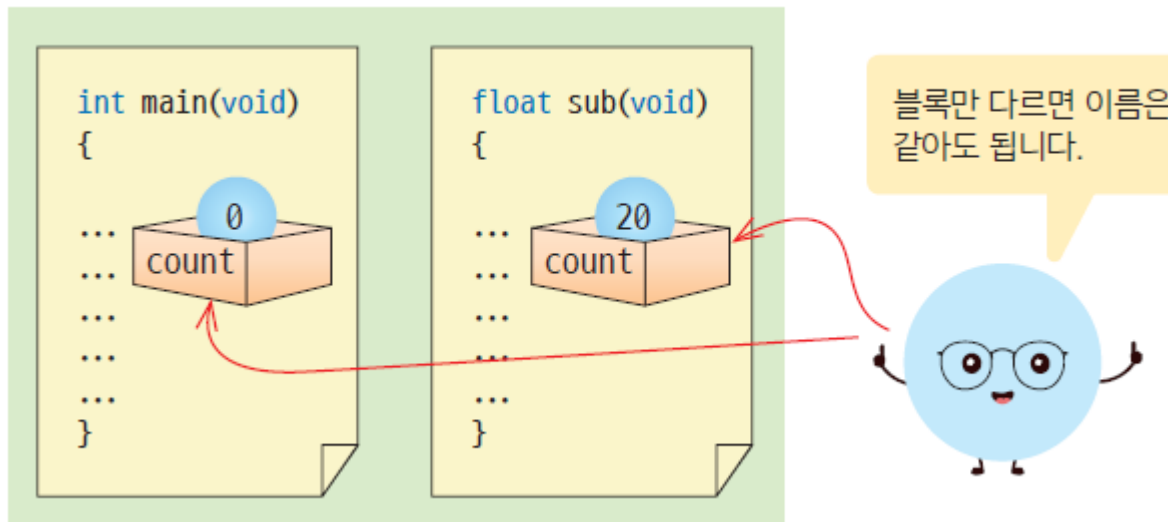
# 지역 변수 선언 위치

- 최신 버전의 C에서는 블록 안의 어떤 위치에서도 선언 가능!!

```
while(1) {  
    ...  
    ...  
    int sum = 0;  
    ...  
}
```

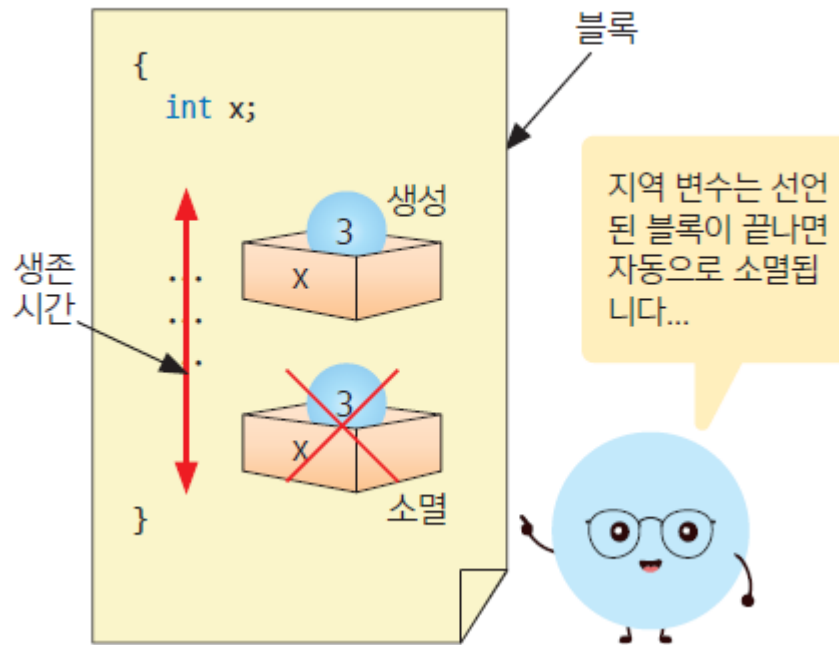
블록의 중간에서도 얼마든지 지역 변수  
를 선언할 수 있다.

# 이름이 같은 지역 변수





# 지역 변수의 생존 기간



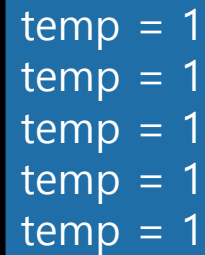
# 지역 변수 예제

```
#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.

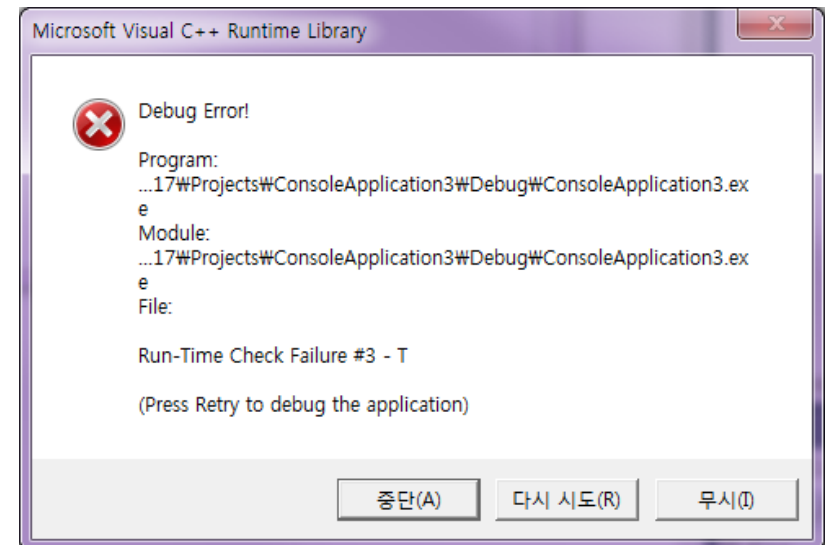


```
temp = 1
temp = 1
temp = 1
temp = 1
temp = 1
```

# 지역 변수의 초기값

```
#include <stdio.h>
int main(void)
{
    int temp;
    printf("temp = %d\n", temp);
    return 0;
}
```

초기화 되지  
않았으므로 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++;
```

```
}
```

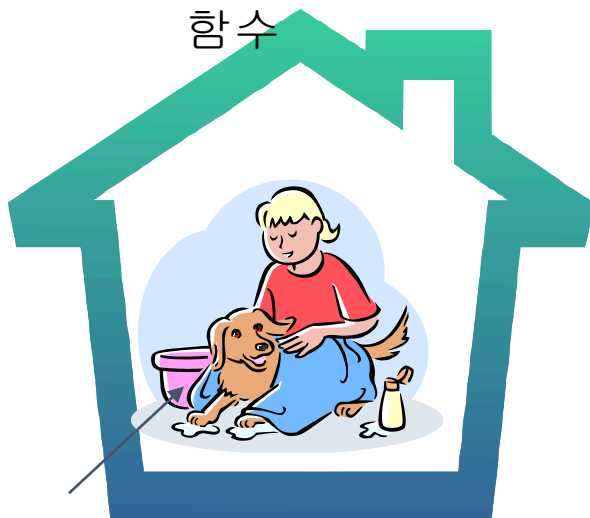
값에 의한 호출  
(call by value)

매개 변수도 일종의  
지역 변수임

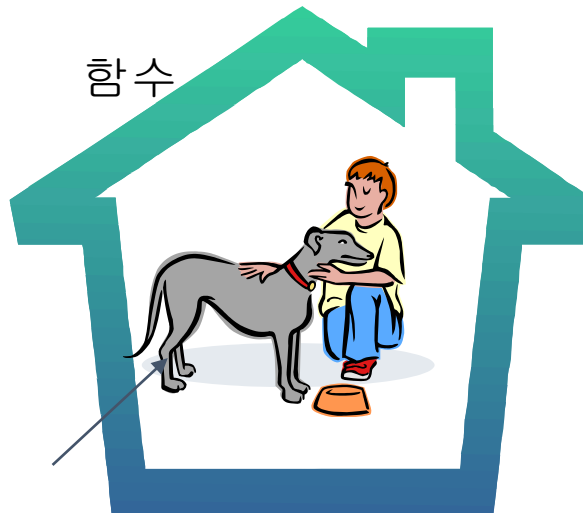
함수 호출전 i=10  
함수 호출후 i=10

# 전역 변수

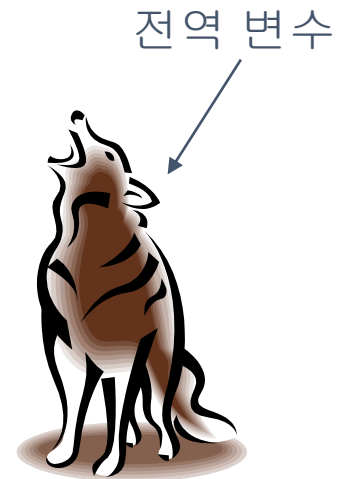
- 전역 변수(global variable)는 함수 외부에서 선언되는 변수이다.
- 전역 변수의 범위는 소스 파일 전체이다.



지역 변수



지역 변수



# 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  
범위

# 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



# 전역 변수의 사용

```
#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("*");  
}
```

출력은  
어떻게  
될까요?

\*\*\*\*\*

# 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;
```

```
}
```

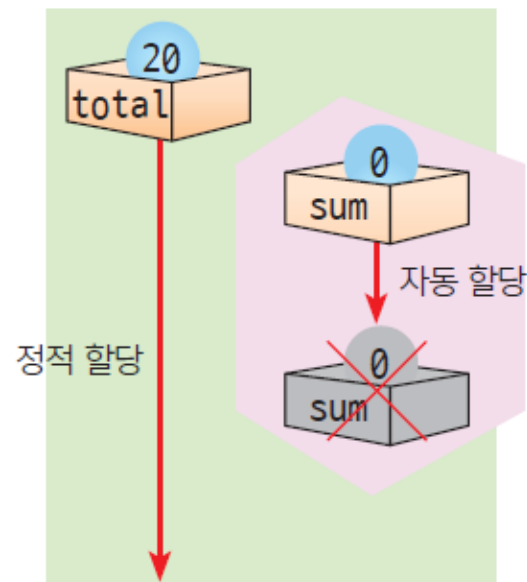
전역 변수와 지역 변수가  
동일한 이름으로 선언된다.

sum = 0

# 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 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() {
```

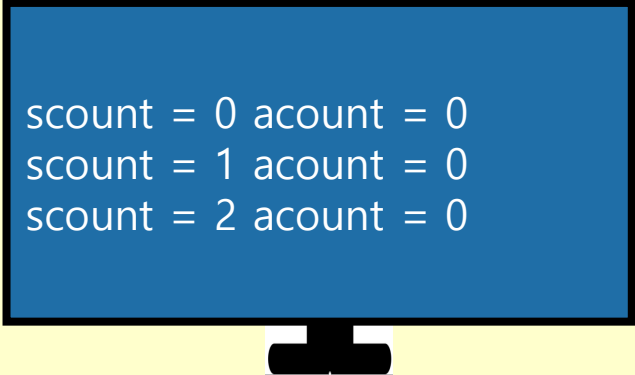
```
    static int scount = 0;  
    int acount = 0;  
    printf("scount = %d\t", scount);
```

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

```
}
```

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

정적 지역 변수로서 static을 붙이면  
지역 변수가 정적 변수로 된다.



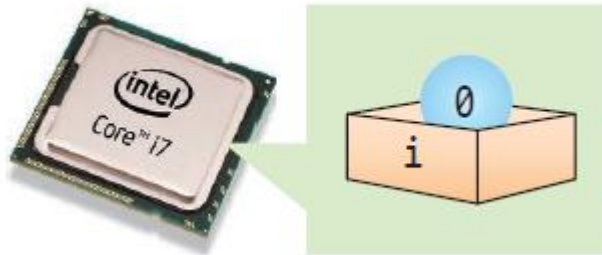
```
scount = 0 acount = 0  
scount = 1 acount = 0  
scount = 2 acount = 0
```

# 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
```

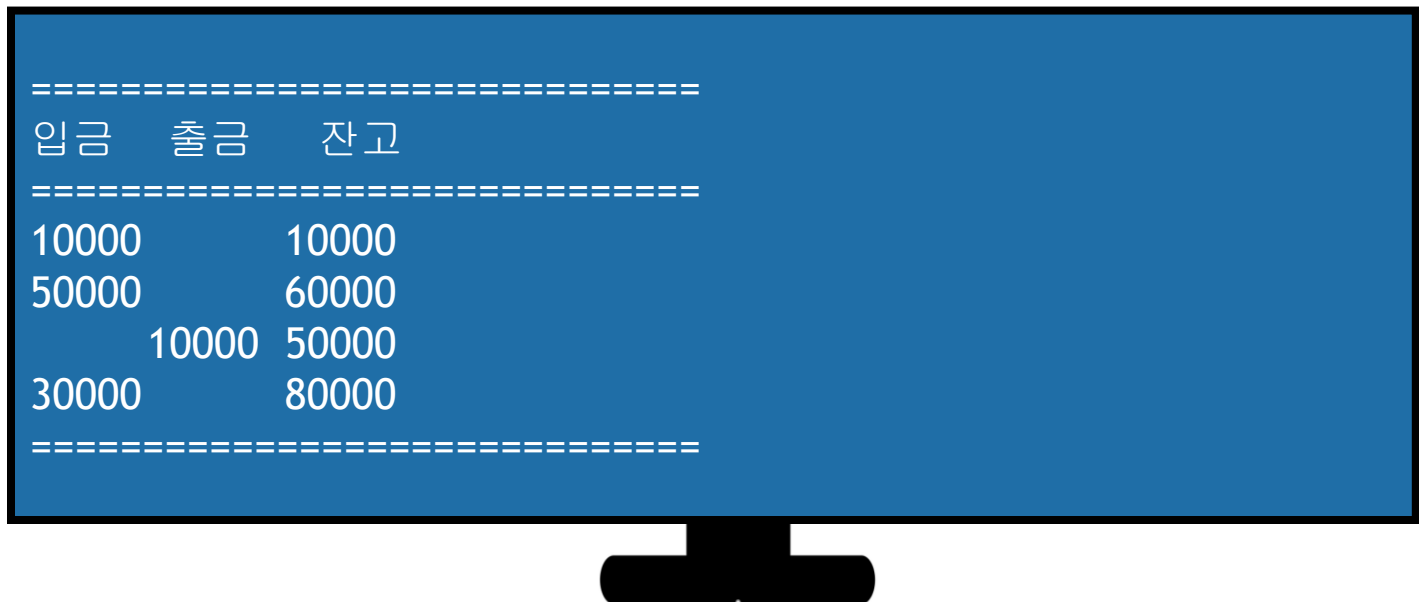
```
void wait(void) {  
    io_port = 0;  
    while (io_port != 255)  
        ;  
}
```

volatile로 지정하면 컴파일러는 최적화를 중지하게 됩니다.



# 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.



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);

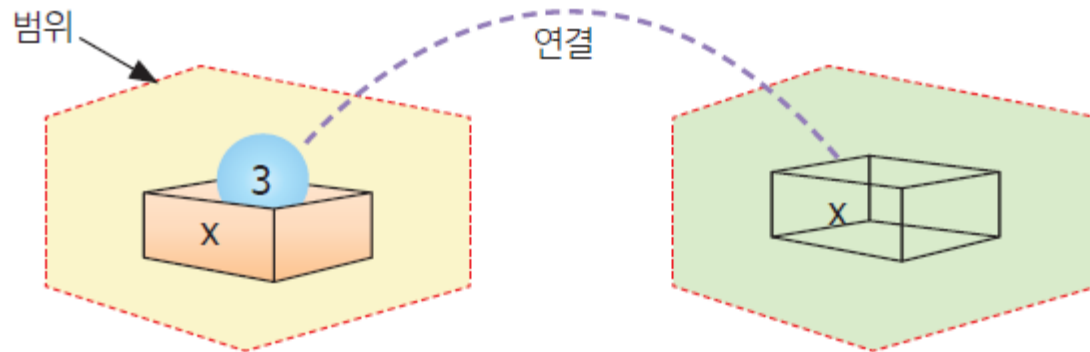
    balance += amount;
    printf("%d \n", balance);
}
```

소 스

```
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;  
}
```

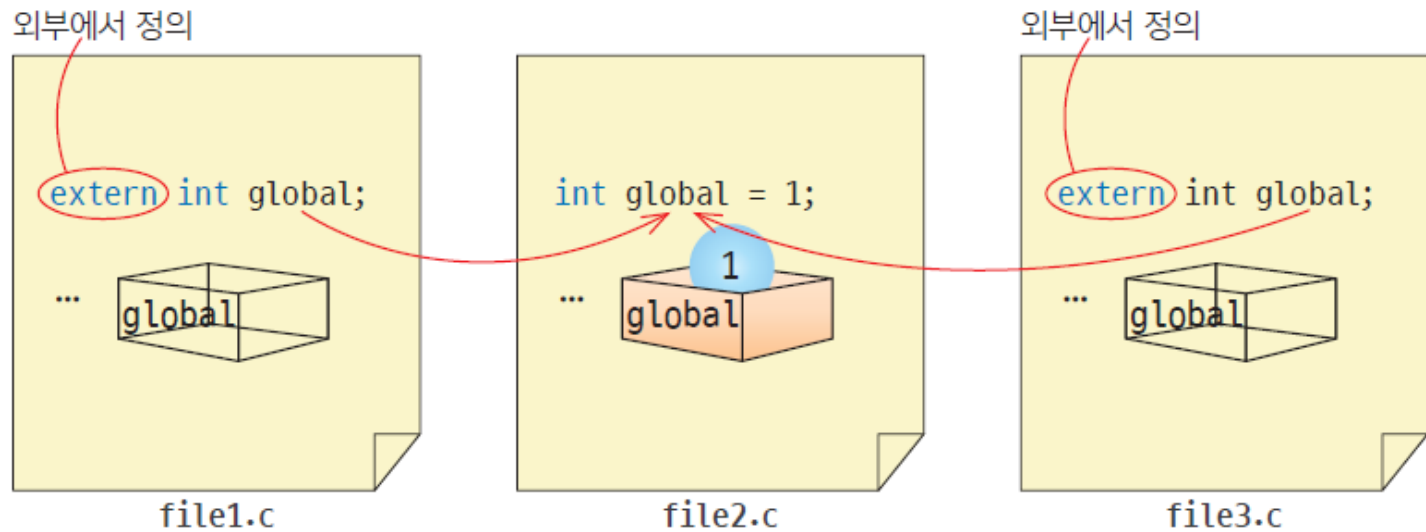
# 연결

- *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

#include <stdio.h>

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

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

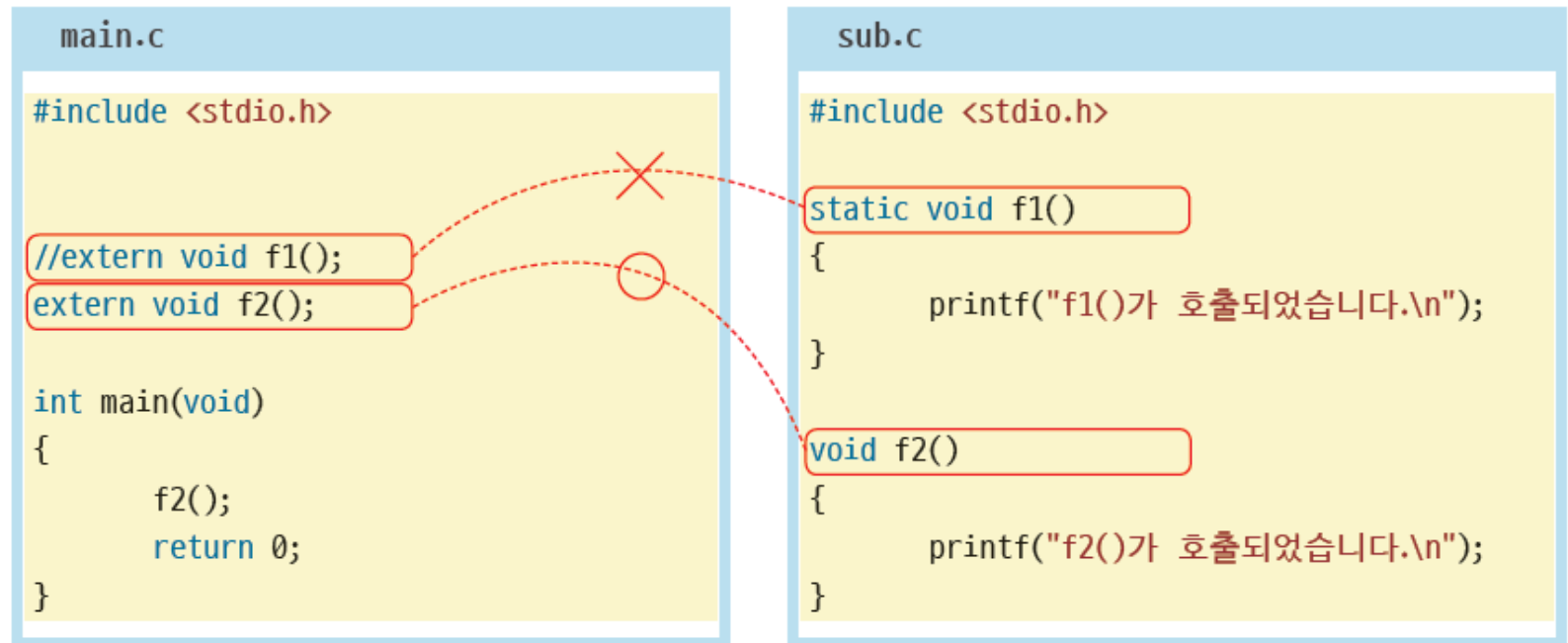
```
linkage2.c

extern int all_files;

// extern int this_file;

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

# 함수 앞의 static



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

# 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 )
```

```
{
```

```
    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 );
    for( ; num > 0; num-- )
        answer += va_arg( argptr, int );
    va_end( argptr );
    return( answer );
```

```
}
```

합은 10입니다.

매개 변수의 개수

# 순환(recursion)이란?

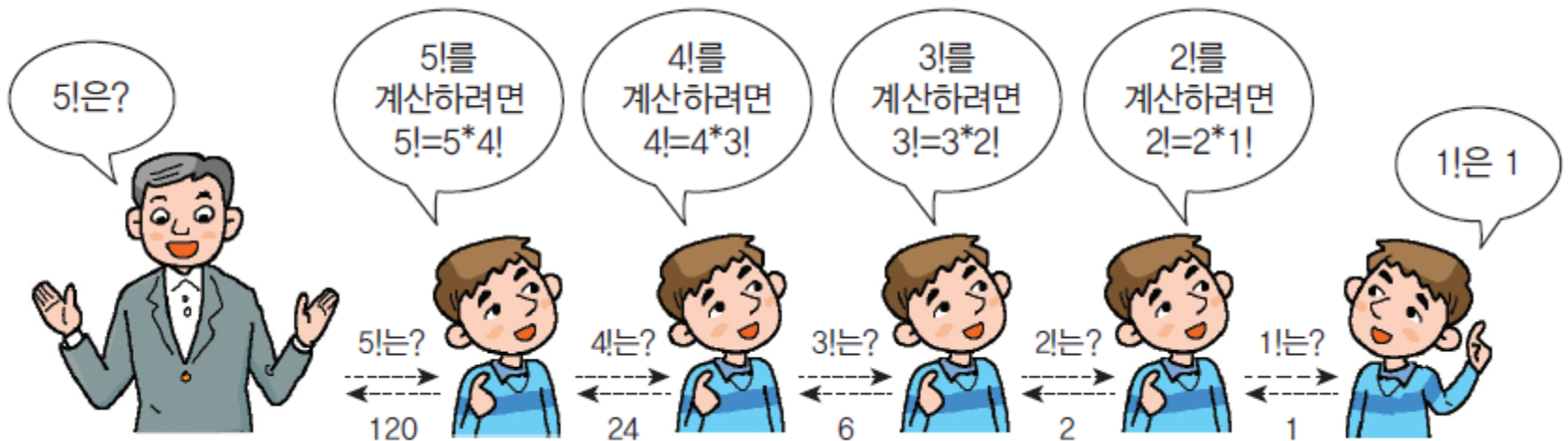
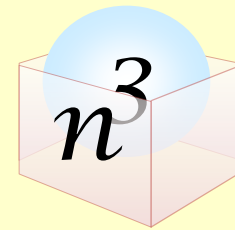
- 함수는 자기 자신을 호출할 수도 있다. 이것을 순환(recursion)라고 부른다.

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

# 팩토리얼 구하기

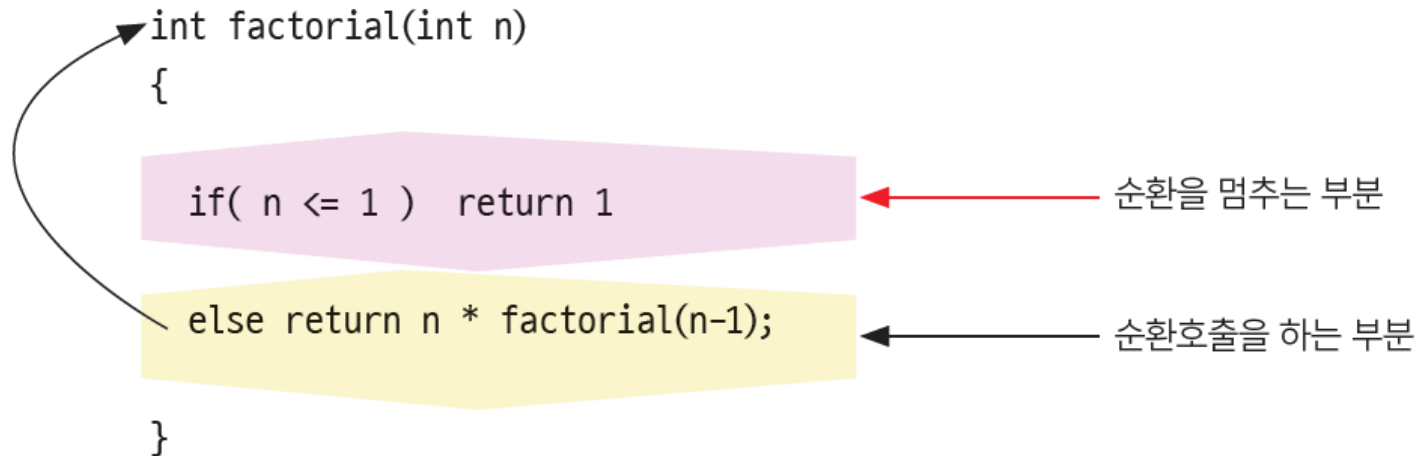
- 팩토리얼 프로그래밍:  $(n-1)!$  팩토리얼을 현재 작성중인 함수를 다시 호출하여 계산(순환 호출)

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



# 순환 함수의 구조

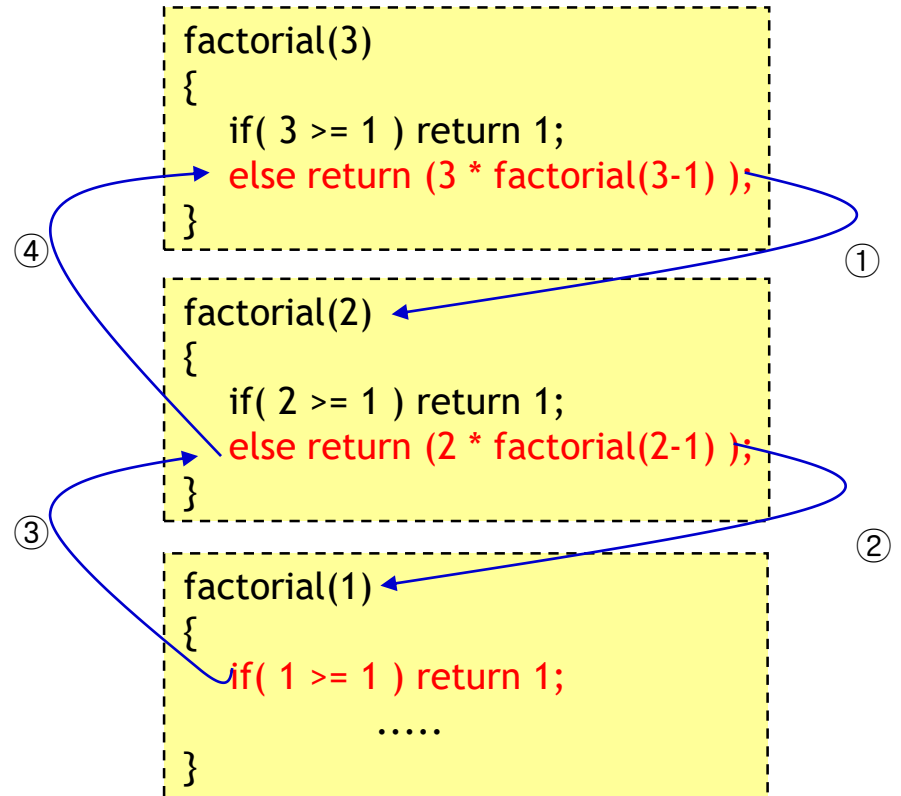
- 순환 알고리즘은 그림 9-9와 같이 자기 자신을 순환적으로 호출하는 부분과 순환 호출을 멈추는 부분으로 구성되어 있다.



# 팩토리얼 구하기

- 팩토리얼의 호출 순서

factorial(3) = 3 \* factorial(2)  
= 3 \* 2 \* 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.

Diagram illustrating the recursive steps for calculating 7 using the formula  $n = 2 * k + 1$ :

- Step 1:  $7 = 2 * 3 + 1$
- Step 2:  $3 = 2 * 1 + 1$
- Step 3:  $1 = 2 * 0 + 1$

Red arrows and circles highlight the sequence of values for  $k$ : 3, 1, 0.

나머지를 역순으로 읽  
으면 1110이 됩니다.



# 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

# Q & A

