

C 언어 EXPRESS(개정3판)



제 9장 함수와 변수



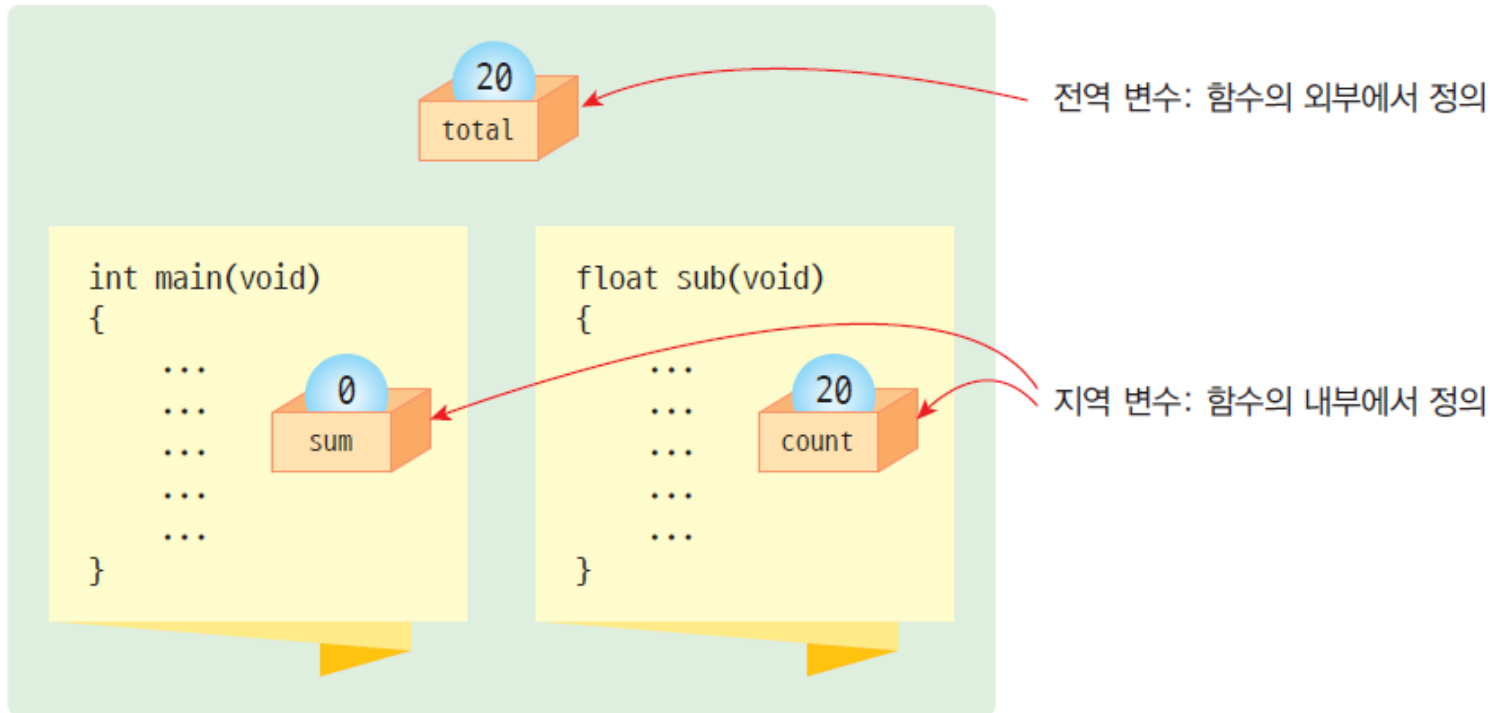
이번 장에서 학습할 내용



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



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

지역 변수는 선언된
블록을 떠나면 안됩니다.

y가 선언된 블록을 벗어나서
사용하였으므로 오류!





지역 변수 선언 위치

- 최신 버전의 C에서는 블록 안의 어떤 위치에서도 선언 가능!!

```
while(flag!=0) {
```

```
...
```

```
int x = get_integer();
```

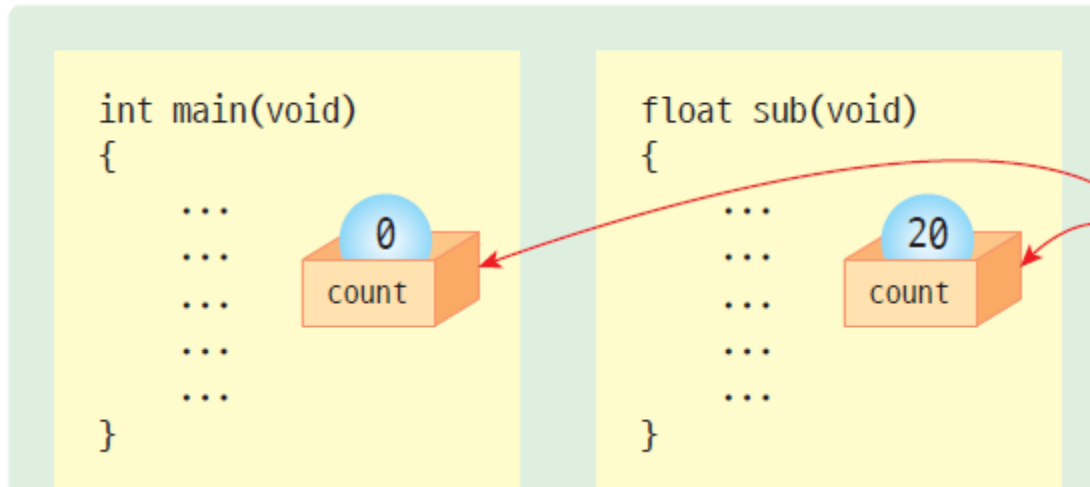
```
...
```

```
}
```

블록의 중간에서도 얼마든지
지역 변수를 선언할 수 있다.



이름이 같은 지역 변수

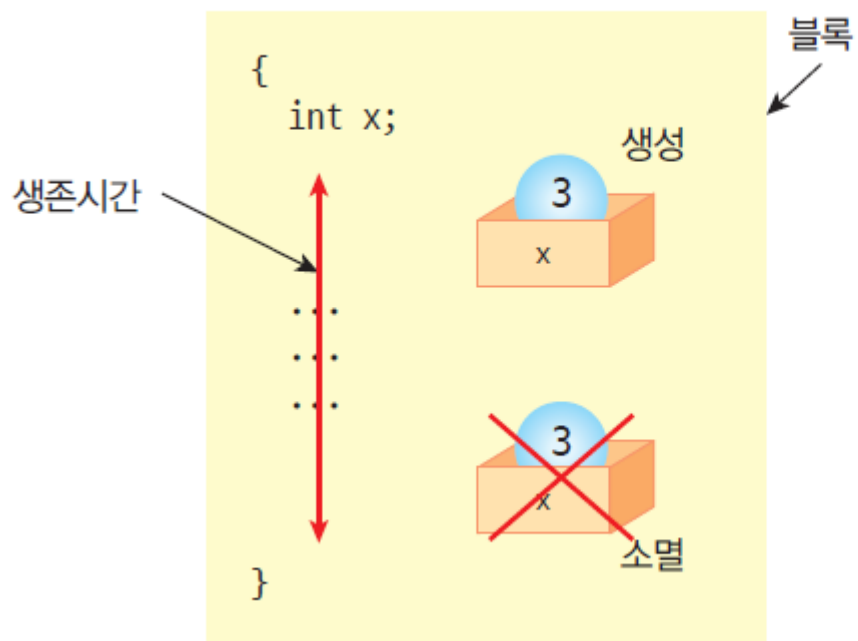


블록만 다르면
이름은 같아도 됩니다.





지역 변수의 생존 기간



지역 변수는 선언된 블록이 끝나면 자동으로 소멸됩니다.





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)
```

```
{
```

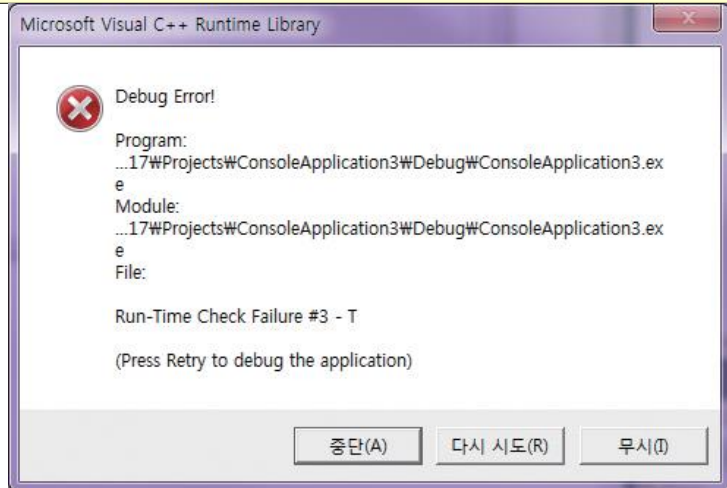
```
    int temp;
```

```
    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 변수의 초기값과 생성 기간

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

```
int A;
```

```
int B;
```

```
int add()
```

```
{
```

```
    return A + B;
```

```
}
```

```
int main()
```

```
{
```

```
    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 초기화

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





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;
```

```
}
```

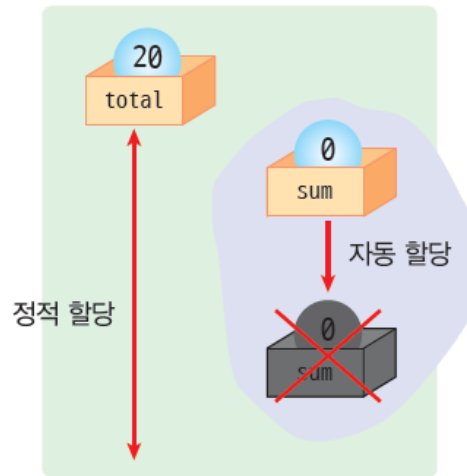
전역 변수와 지역 변수가 동일한 이름으로 선언된다.

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 `auto`가 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();
```

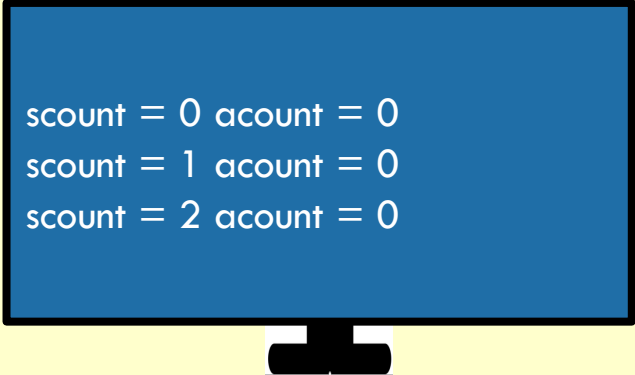
```
    sub();
```

```
    sub();
```

```
    return 0;
```

```
}
```

← 정적 지역 변수로서 static을 붙이면 지역 변수가 정적 변수로 된다.



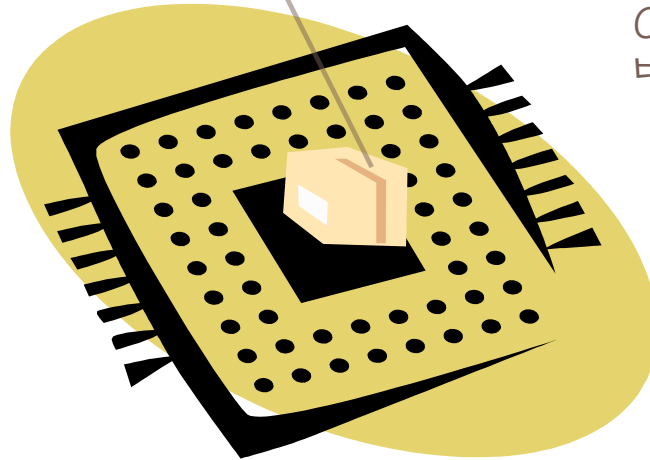
```
scount = 0 acount = 0  
scount = 1 acount = 0  
scount = 2 acount = 0
```



저장 유형 지정자 register

- 레지스터(register)에 변수를 저장.

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



CPU안의 레지스터에
변수가 저장됨



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
=====
```



```
#define _CRT_SECURE_NO_WARNINGS
#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);
}
```

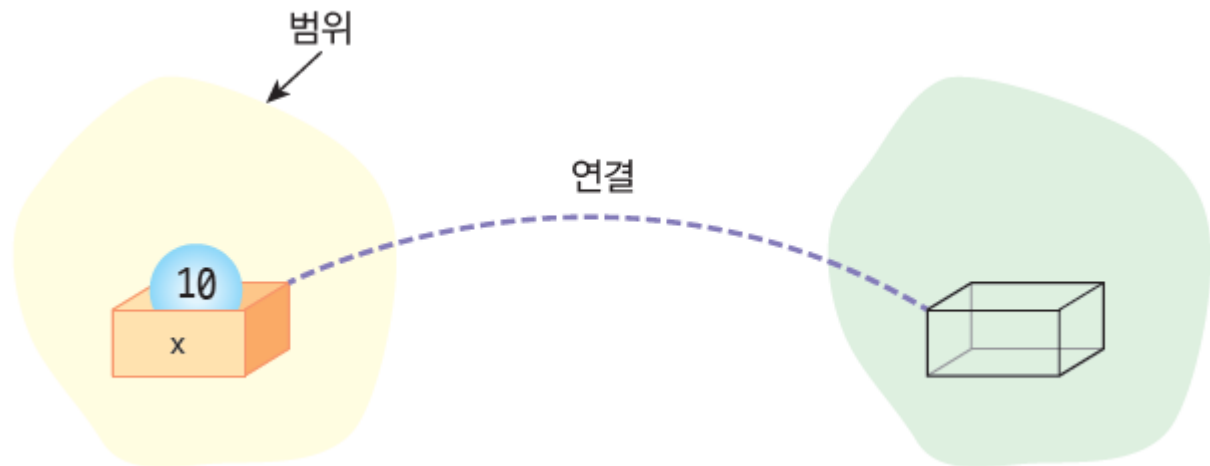


```
int main(void) {  
    printf("=====\n");  
    printf("입금 \t출금\t 잔고\n");  
    printf("=====\n");  
    save(10000);  
    save(50000);  
    save(-10000);  
    save(30000);  
    printf("=====\n");  
    return 0;  
}
```




연결

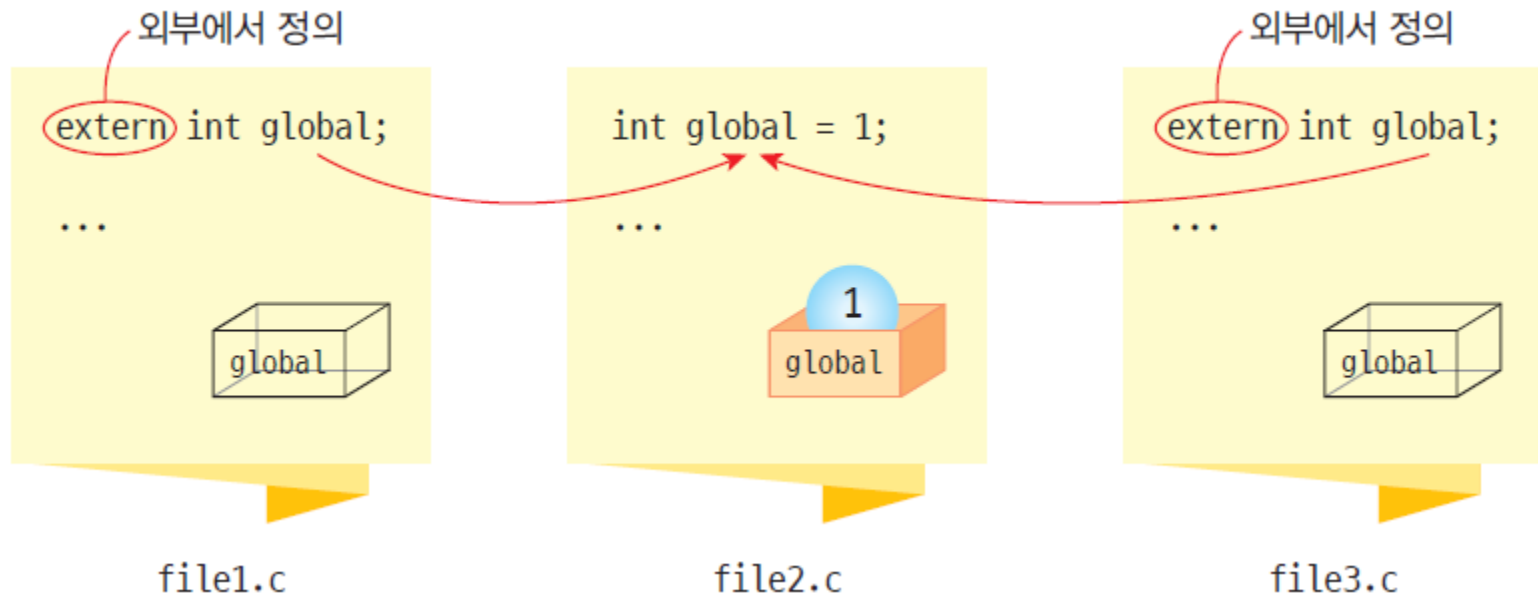
- *연결(linkage)*: 다른 범위에 속하는 변수들을 서로 연결하는 것
 - 외부 연결
 - 내부 연결
 - 무연결
- 전역 변수만이 연결을 가질 수 있다.





외부 연결

- 전역 변수를 선언하고
- 다른 파일에서 **extern**을 이용하여서 연결
- 초기화는 변수가 정의된 파일에서만 가능





연결 예제

- static이 붙으면 하나의 소스파일에서만 사용가능하다

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

```
int all_files;
```

```
static int this_file;
```

```
extern void sub();
```

```
int main(void)
```

```
{
```

```
    sub();
```

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

```
    return 0;
```

```
}
```

linkage1.c

```
extern int all_files;
```

```
// extern int this_file;
```

```
void sub(void)
```

```
{
```

```
    all_files = 10;
```

```
}
```

linkage2.c



함수의 static

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

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

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

```
int main(void)
```

```
{
```

```
    f2();
```

```
    return 0;
```

```
}
```

main.c

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

```
static void f1()
```

```
{
```

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

```
}
```

```
void f2()
```

```
{
```

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

```
}
```

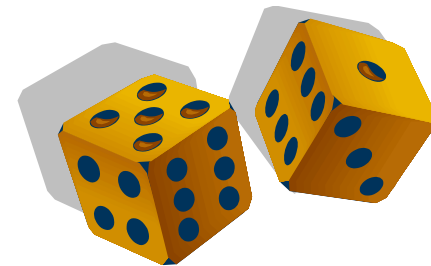
sub.c



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

- 0부터 10사이의 난수를 몇 개 생성해보자.

0 9 6 7 0 9 6 7 0 9



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

    return 0;
}
```




순환(recursion)이란?

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



팩토리얼 구하기

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

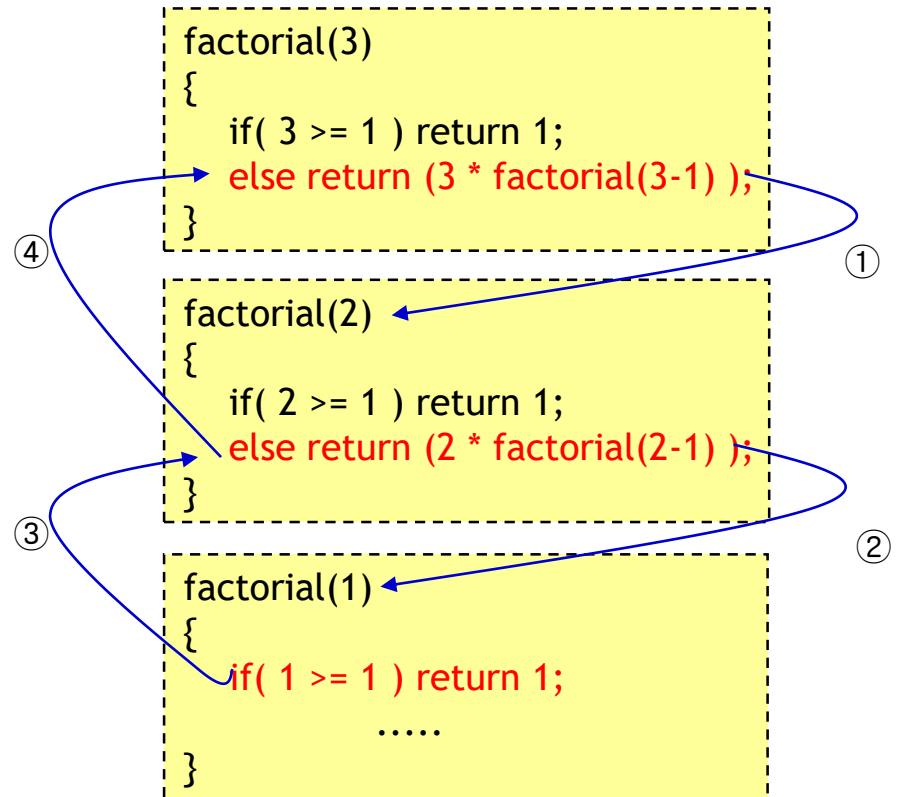
```
int factorial(int n)
{
    if( n <= 1 ) return(1);
    else return (n * factorial(n-1) );
}
```



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.



문제

자연수 N 을 입력받아 재귀함수를 이용하여 N 부터 1까지 차례대로 출력하는 프로그램을 작성하십시오.
 N 은 50이하의 자연수이다.

입력 예

5

출력 예

5 4 3 2 1



문제

100 이하의 자연수 N 을 입력받아 재귀함수를 이용하여 1부터 N 까지의 합을 구하는 프로그램을 작성하시오.

입력 예

100

출력 예

5050



문제

9자리 이하의 자연수를 입력받아 재귀함수를 이용하여 각 자리 숫자의 제곱의 합을 출력하는 프로그램을 작성하시오.

(`main()` 함수에 변수 하나, 재귀함수에 매개변수 하나만을 사용할 수 있다.)

입력 예

12345

출력 예

55



2진수 형식으로 출력하기

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

$$\begin{aligned} 7 &= 2 * 3 + 1 \\ 3 &= 2 * 1 + 1 \\ 1 &= 2 * 0 + 1 \end{aligned}$$



나머지를 역순으로 읽으면
111이 됩니다.



2진수 형식으로 출력하기

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

void print_binary(int x);

int main(void)
{
    print_binary(9);
}

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));
}

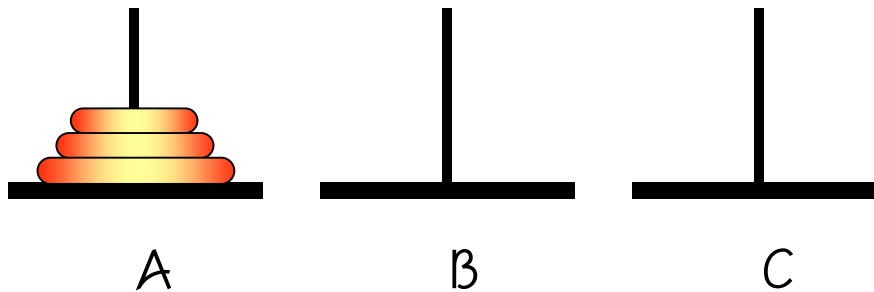
// x는 y보다 커야 한다.
int gcd(int x, int y)
{
    if (y == 0)
        return x;
    else
        return gcd(y, x % y);
}
```

10



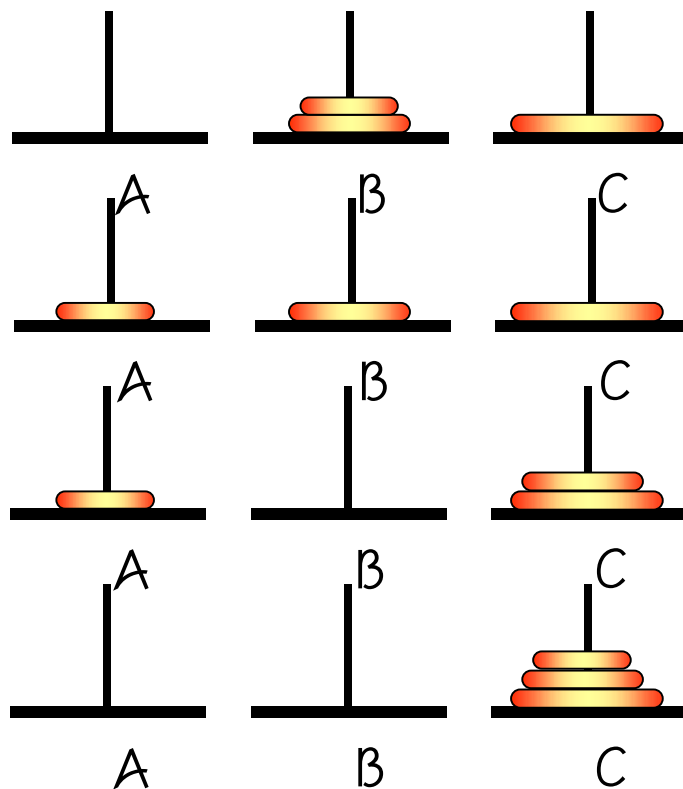
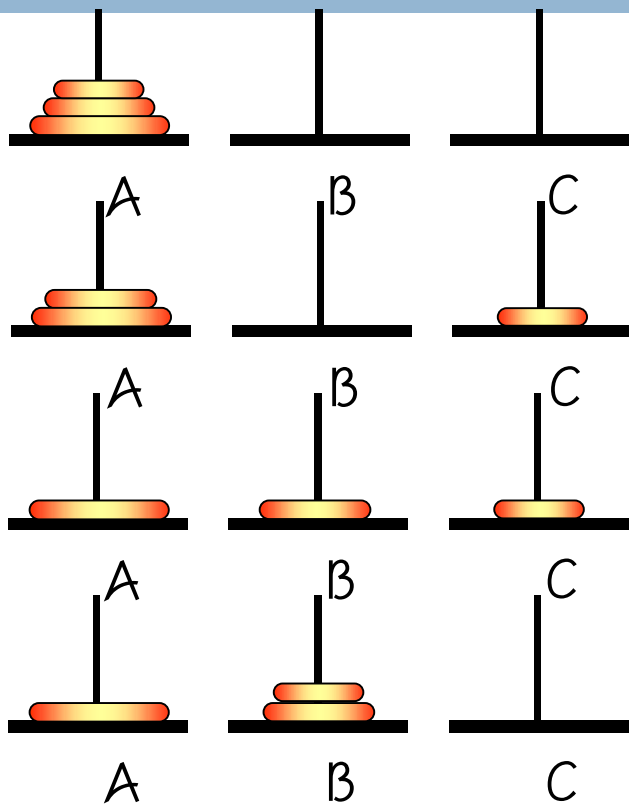
하노이 탑 문제

- 문제는 막대 **A**에 쌓여있는 원판 3개를 막대 **C**로 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.
 - 중간의 막대를 임시적으로 이용할 수 있으나 앞의 조건들을 지켜야 한다.





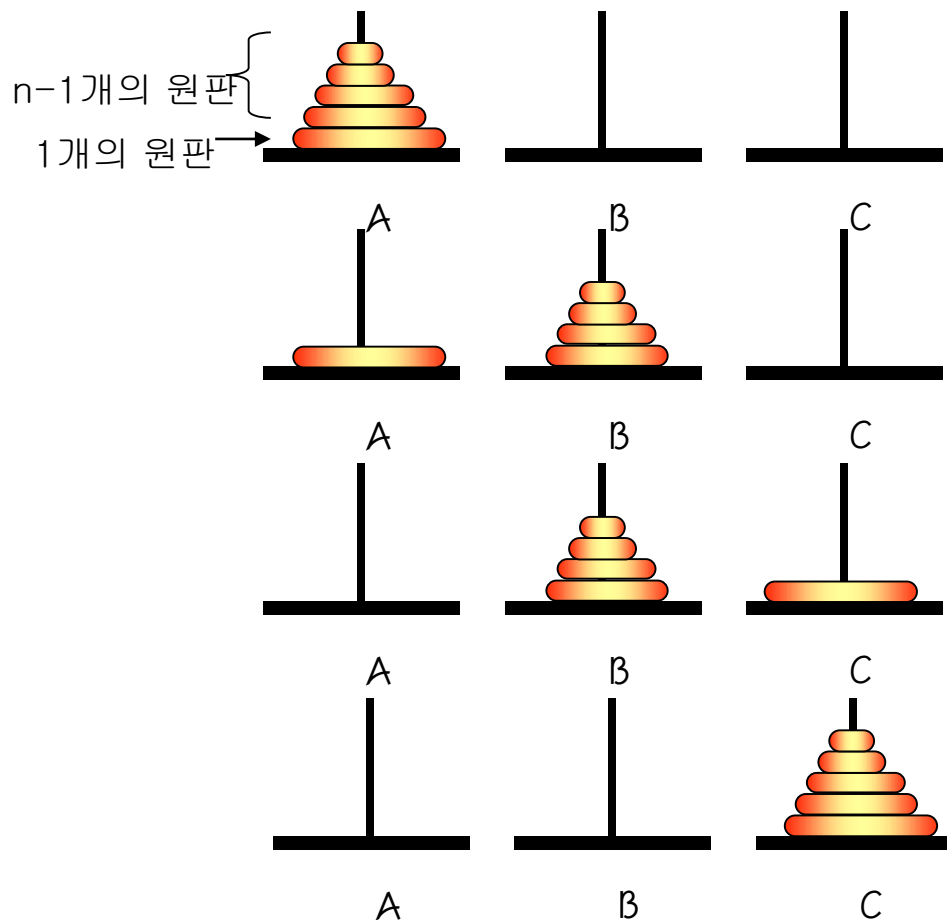
3개의 원판이 경우의 해답





n 개의 원판이 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. }
```



하노이탑 프로그램

```
#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);
    }
}
```




하노이탑 실행 결과

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.