추가자료
5장. 함수와 변수

박 종 혁 교수

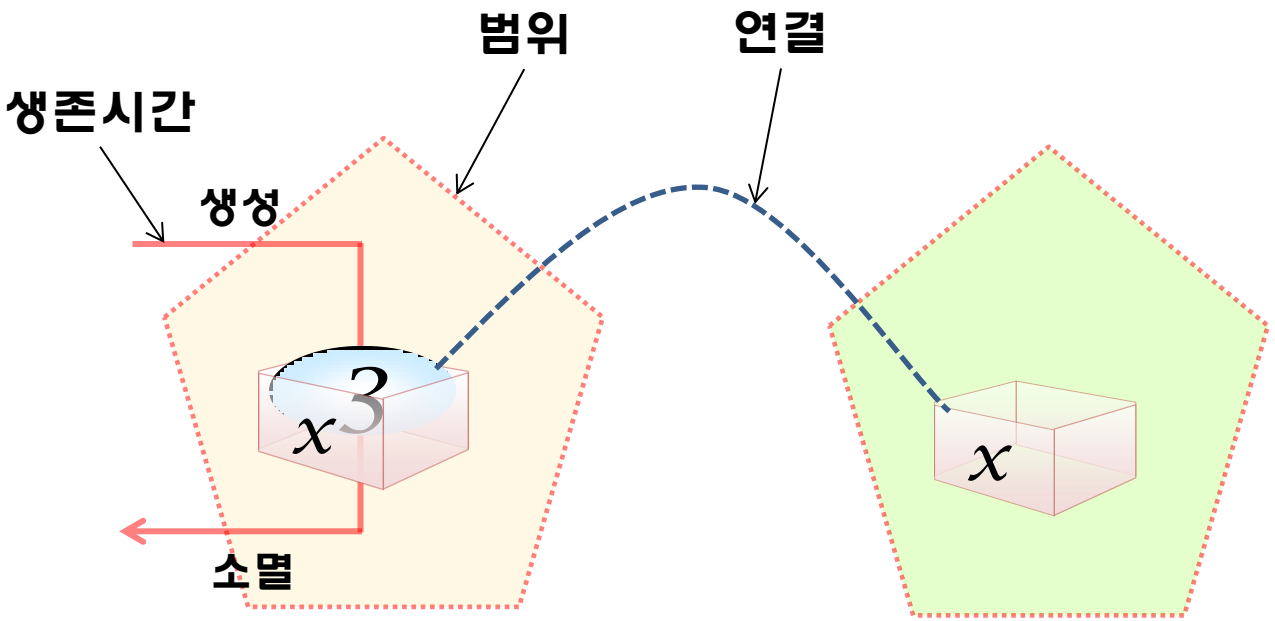
UCS Lab

Tel: 970-6702

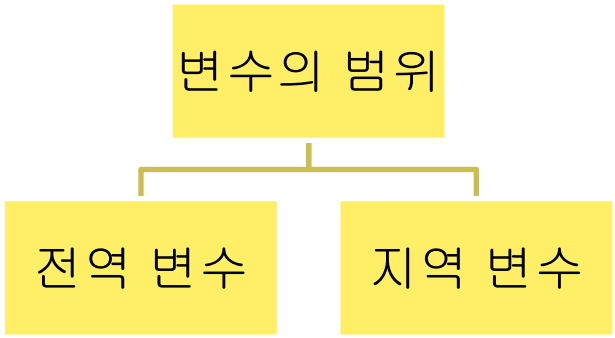
Email: jhpark1@seoultech.ac.kr

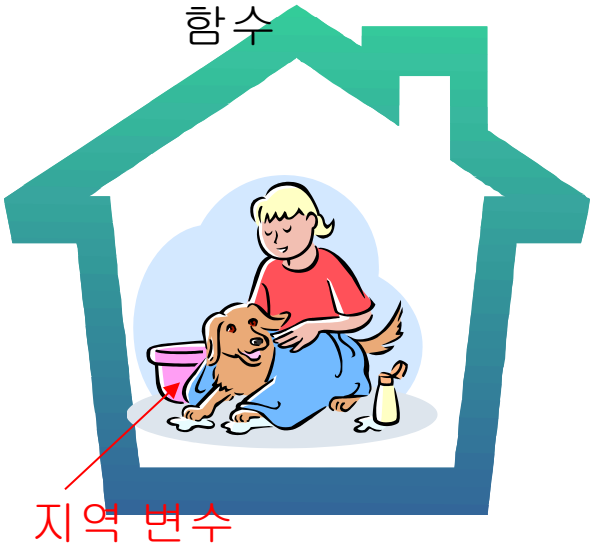
변수의 속성

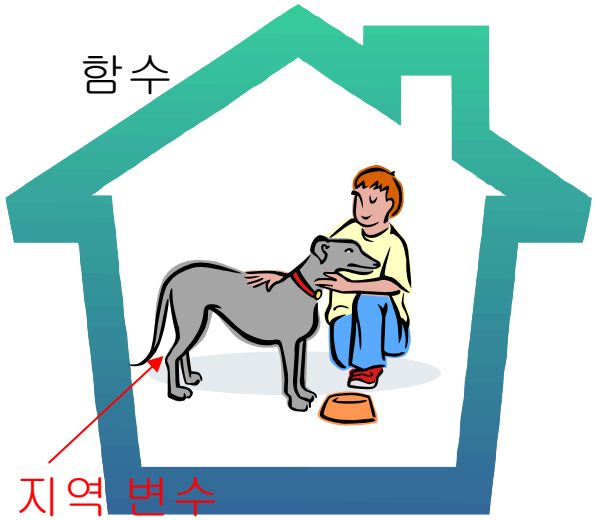
- 변수의 속성 : 이름, 타입, 크기, 값 + 범위, 생존 시간, 연결
  - 범위(scope) : 변수가 사용 가능한 범위, 가시성
  - 생존 시간(lifetime) : 메모리에 존재하는 시간
  - 연결(linkage) : 다른 영역에 있는 변수와의 연결 상태

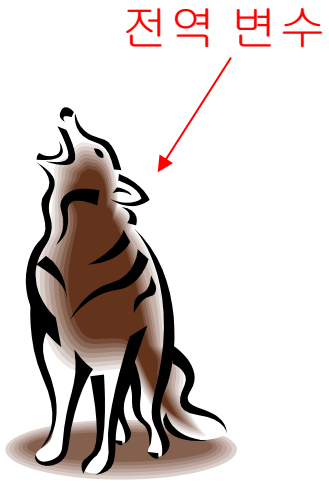


변수의 범위

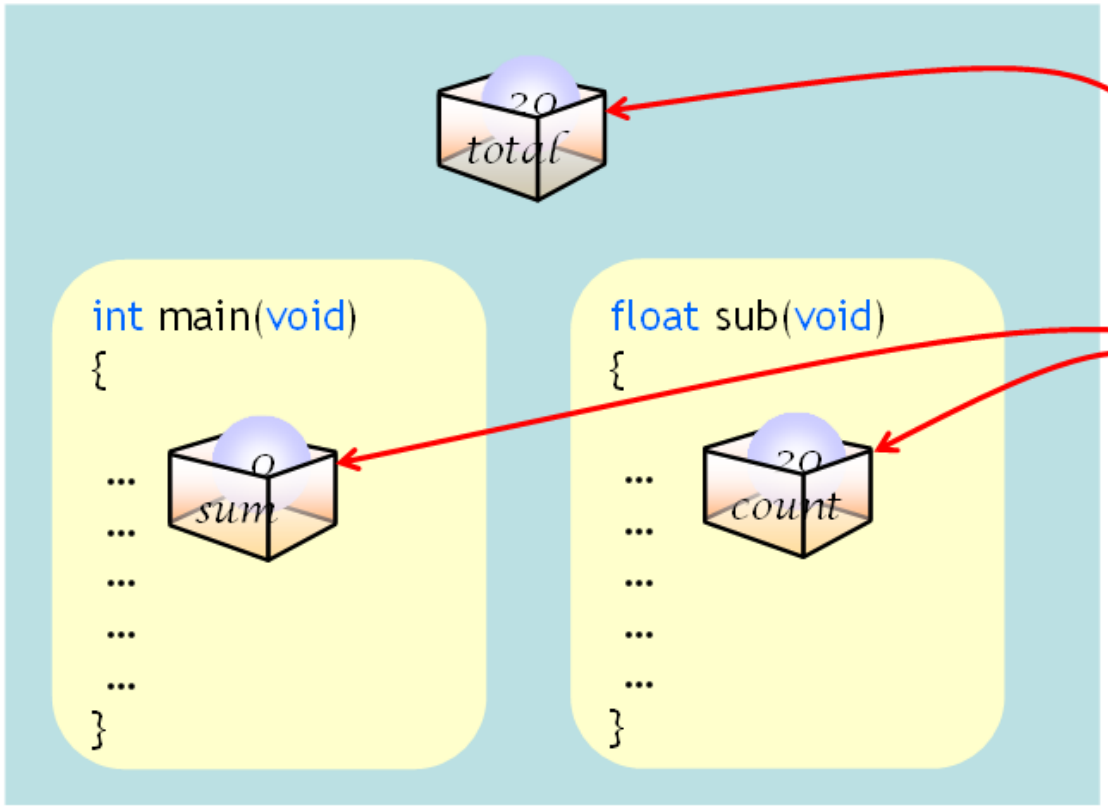








전역 변수와 지역 변수

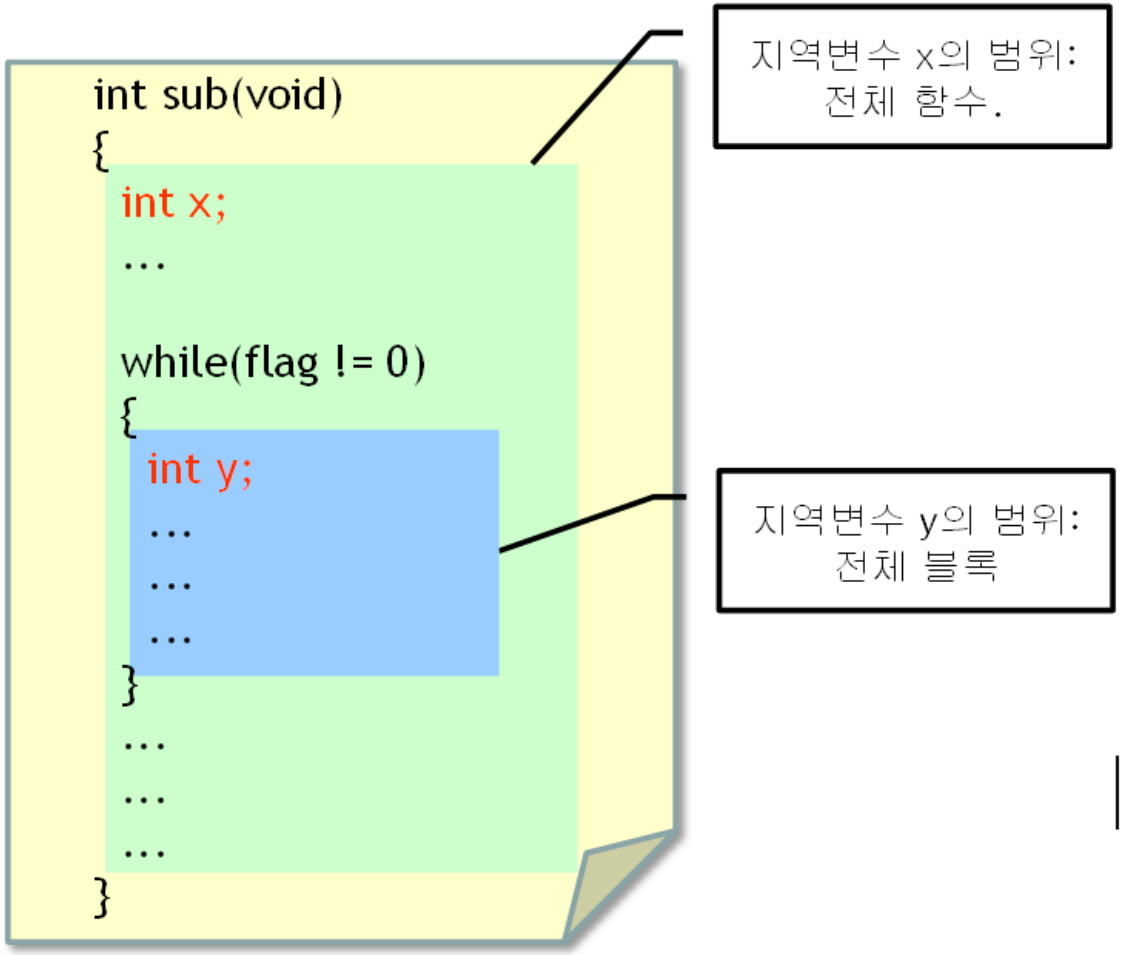


전역 변수: 함수의 외부에서 정의

지역 변수: 함수의 내부에서 정의

지역 변수

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



지역 변수 선언 위치

블록 첫 부분에서 정의

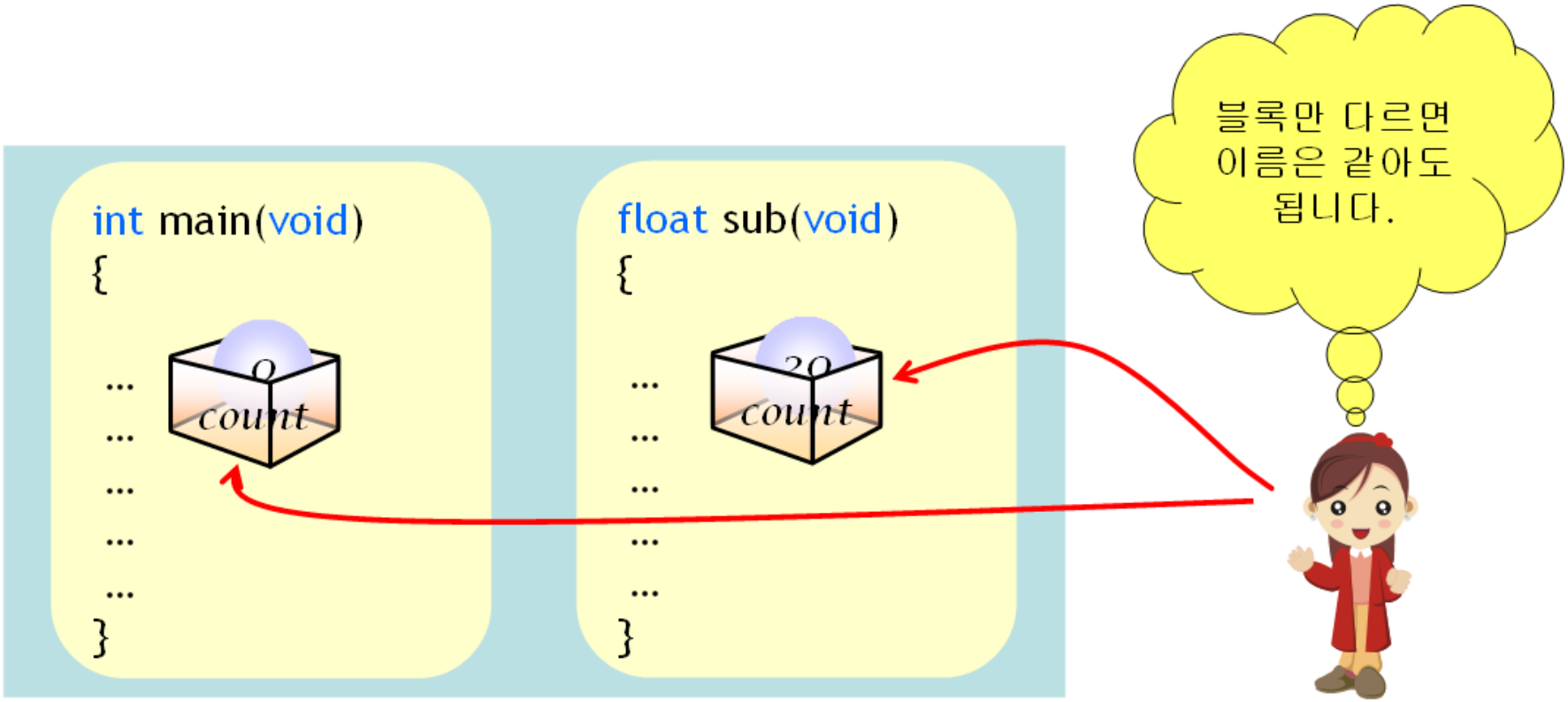
```
int sub(void)
{
    int x;    // ①
    ...
    x = 100;
    int y;    // ②
}
```

변수를 함수의 첫 부분에 선언하지 않으셨군요. 컴파일 오류입니다.

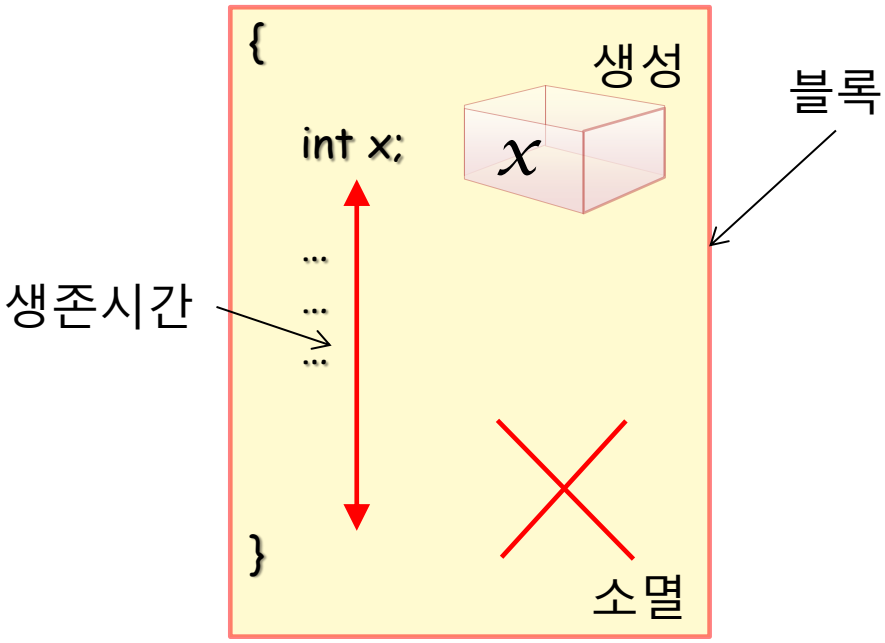
지역 변수의 범위

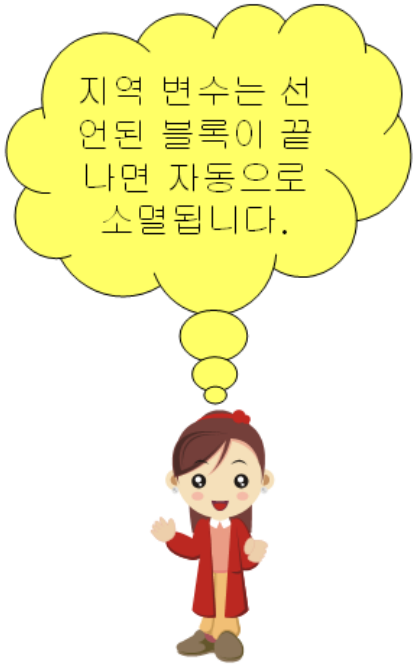
```
void sub1(void)
{
        {
                int y;
                ...
        }
        y = 4;
}
```

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

```
#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

```
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.

```
int x = 123;

void sub1()
{
        x = 456;
}

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

전역 변수의 초기값과 생존 기간

```c
#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();
        }
        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;
}
```

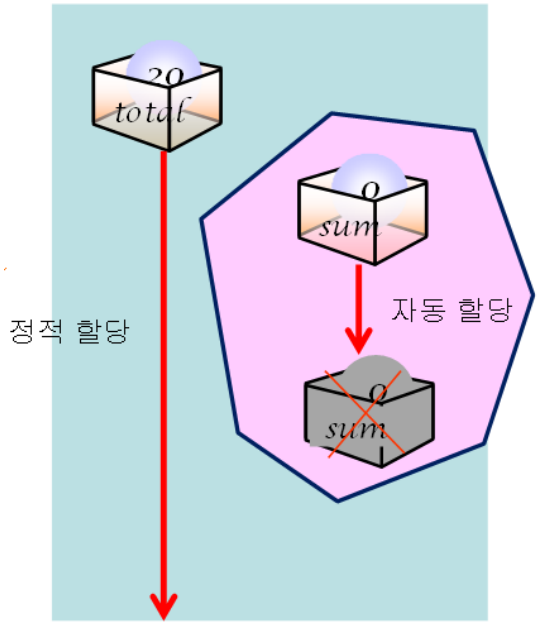
전역 변수와 지역 변수가
동일한 이름으로 선언된다.

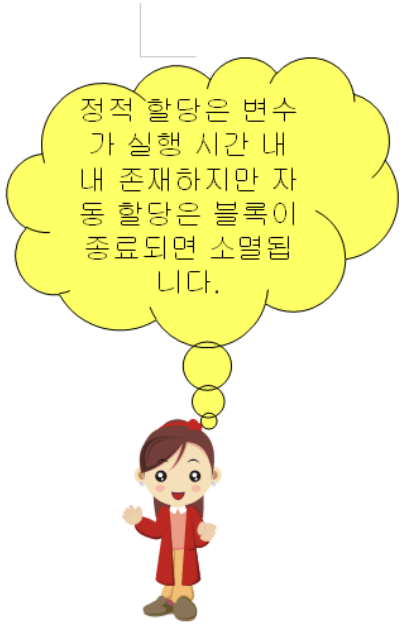
```
sum = 0
```

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 auto

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

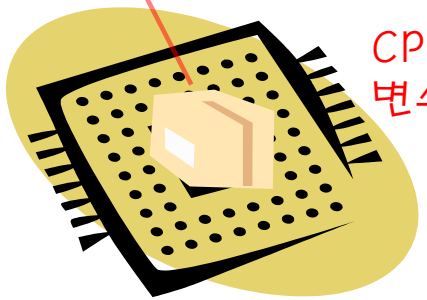
```
int main(void)
{
  auto int sum = 0;
  int i = 0;
  ...
  ...
}
```

전부 자동 변수로서 함수가 시작되면 생성되고 끝나면 소멸된다.

저장 유형 지정자 register

- 레지스터(register)에 변수를 저장.

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



CPU안의 레지스터에
변수가 저장됨

알고리즘

- *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.*

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);
        }
        if( id == super_id && password == super_password )
                return SUCCESS;
        else
                return FAIL;
}
```

정적 지역 변수

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
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

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

저장 유형 정리

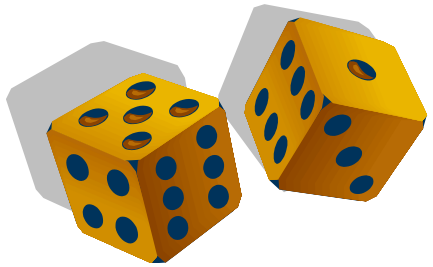
- 일반적으로는 *자동 저장 유형* 사용 권장
- 자주 사용되는 변수는 *레지스터 유형*
- 변수의 값이 함수 호출이 끝나도 그 값을 유지하여야 할 필요가 있다면 *지역 정적*
- 만약 많은 함수에서 공유되어야 하는 변수라면 *외부 참조 변수*

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
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

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



예 제

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

순환(recursion)이란?

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

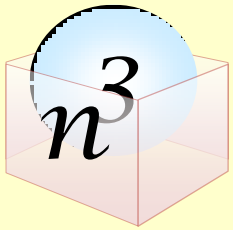
- 팩토리얼의 정의

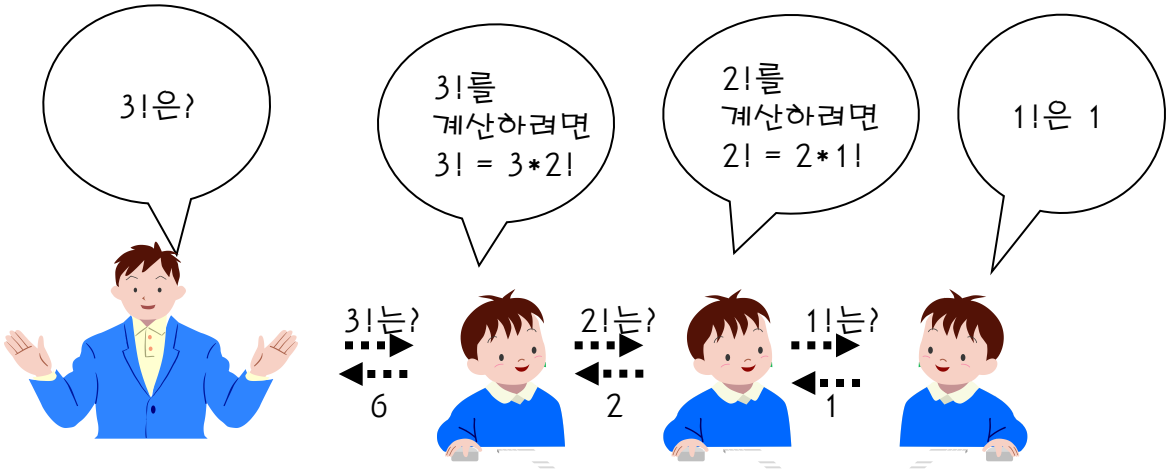
$$n! = \begin{cases} 1 & n = 1 \\ n*(n-1)! & n \geq 2 \end{cases}$$

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) );
}
```

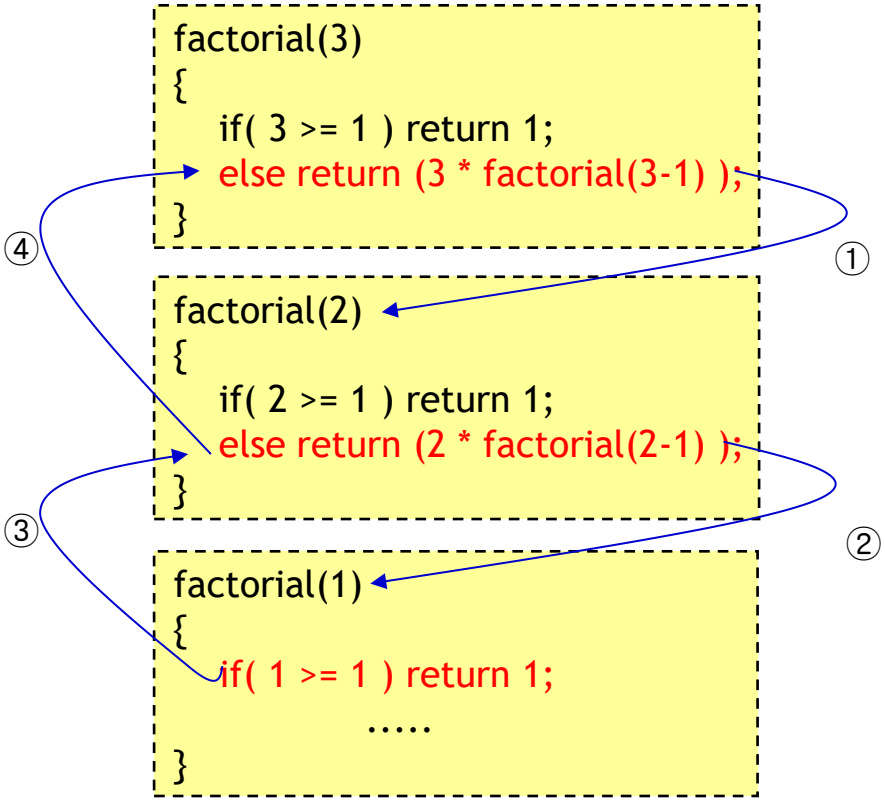




팩토리얼 구하기

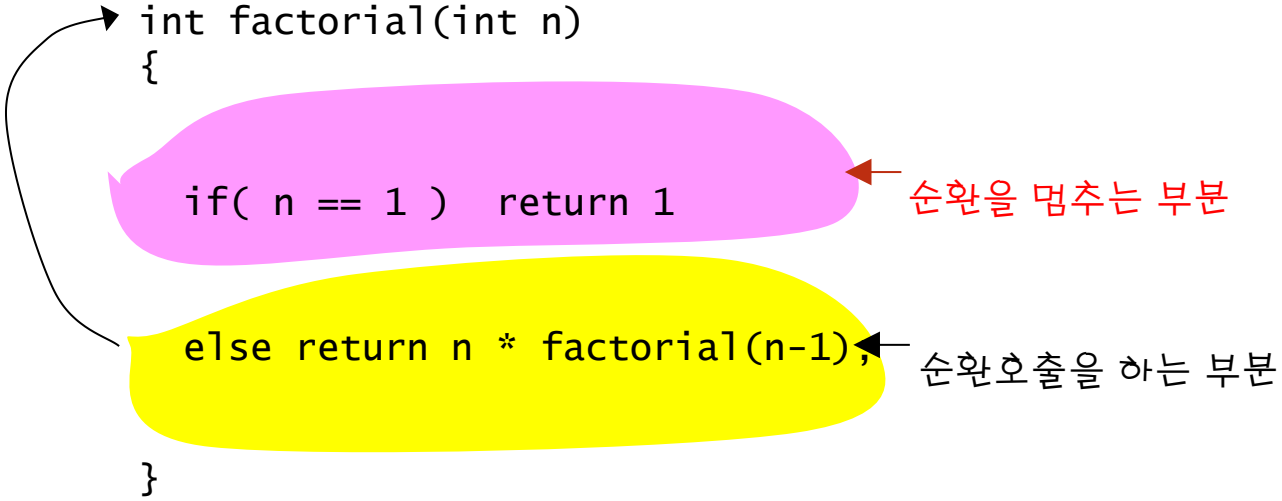
- 팩토리얼의 호출 순서

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,...

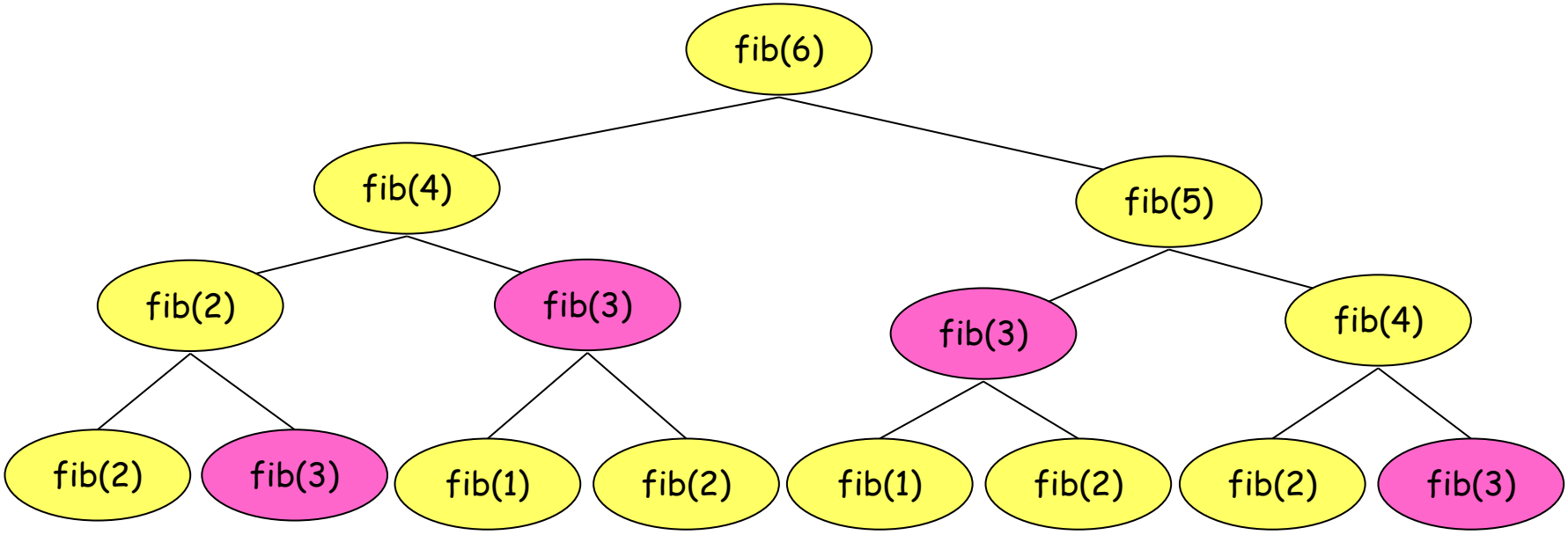
$$fib(n)\begin{cases} 0 & n=0 \\ 1 & n=1 \\ fib(n-2)+fib(n-1) & otherwise \end{cases}$$

- 순환적인 구현

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

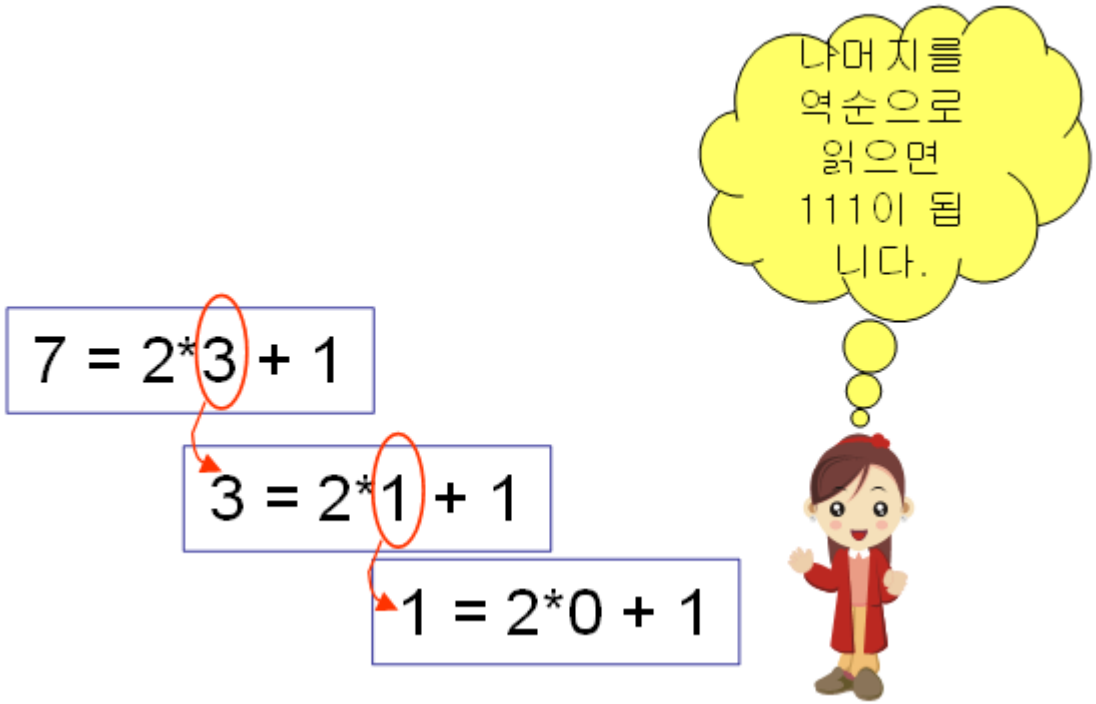
피보나치 수열의 계산

- 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



이진수 출력하기

- 정수를 이진수로 출력하는 프로그램 작성
- 순환 알고리즘으로 가능



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
        }
}
```

Reference

- C언어EXPRESS, 천인국, 생능출판사, 2016.01