C 프로그래밍

(001/002)

제7강

신 한 대 학 교
소프트웨어융합학과
교수 송 진 희

신한대학교

C 프로그래밍

제 7 강

- 지역 변수(Local Variable)
- 전역 변수(Global Variable)
- 변수의 생존 범위(Scope Rule)

신한대학교

학습 목표

- 지역 변수의 개념을 설명할 수 있고, 지역 변수를 선언· 사용할 수 있다.
- 전역 변수의 설명할 수 있고, 전역 변수를 선언· 사용할 수 있다.
- 변수의 Scope Rule을 설명할 수 있다.
- 정적 변수의 개념을 설명할 수 있고, 정적 변수를 선언· 사용할 수 있다.

6강 – 정리 요약

○함수

- 프로그램에서 반복 처리되는 기능을 독립시켜 이름(함수명)을 부여해서 사용
- 반복 처리를 위한 코드를 한 번만 기술한 후, 함수명으로 호출해서 사용 가능
- 소스 코드의 중복 기술을 방지하고, 한 번 작성된 함수를 필요할 때 마다 재사용 가능

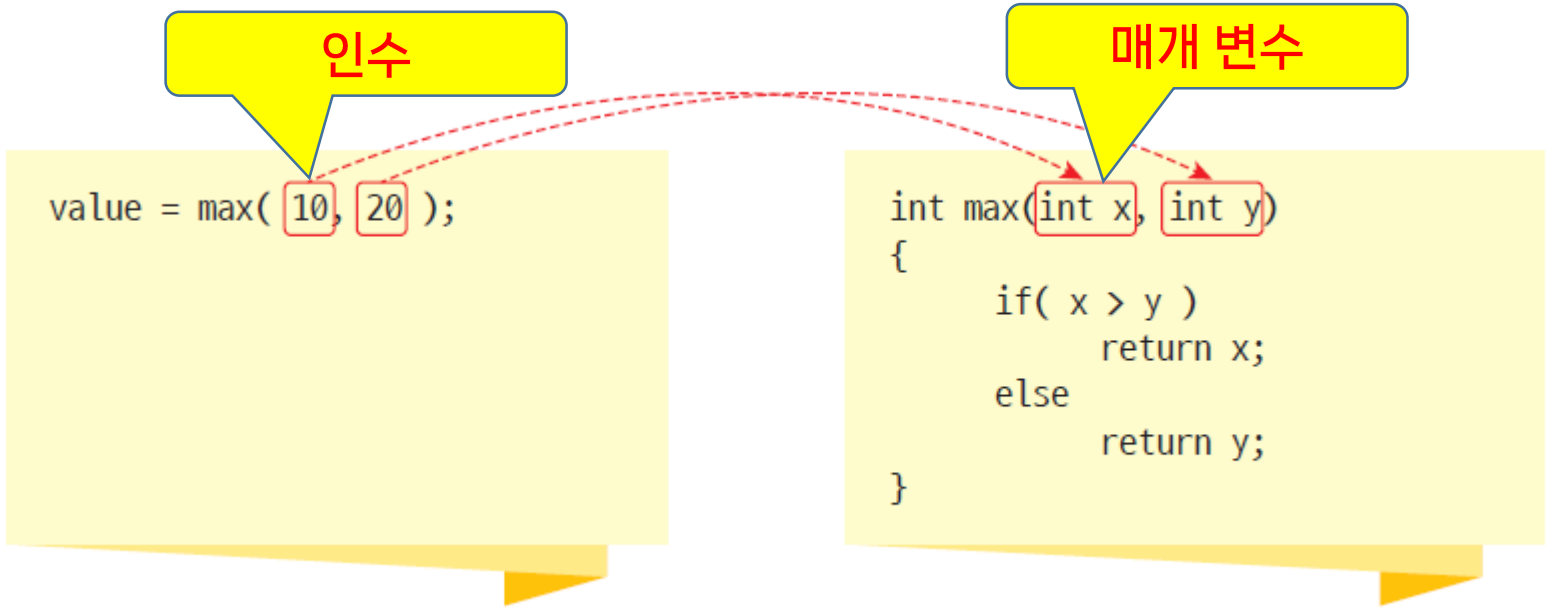
○함수의 처리 결과를 호출한 소스 프로그램으로 반환 가능

○함수 처리에 필요한 입력 자료를 전달받아 처리 가능

6강 – 정리 요약

○함수 처리에 필요한 입력 자료를 전달받아 처리 가능

- ➢실 인수 : 함수에 전달되는 실제 데이터
- ➢매개 변수 : 함수를 호출한 프로그램에서 전달한 데이터를 수신할 변수

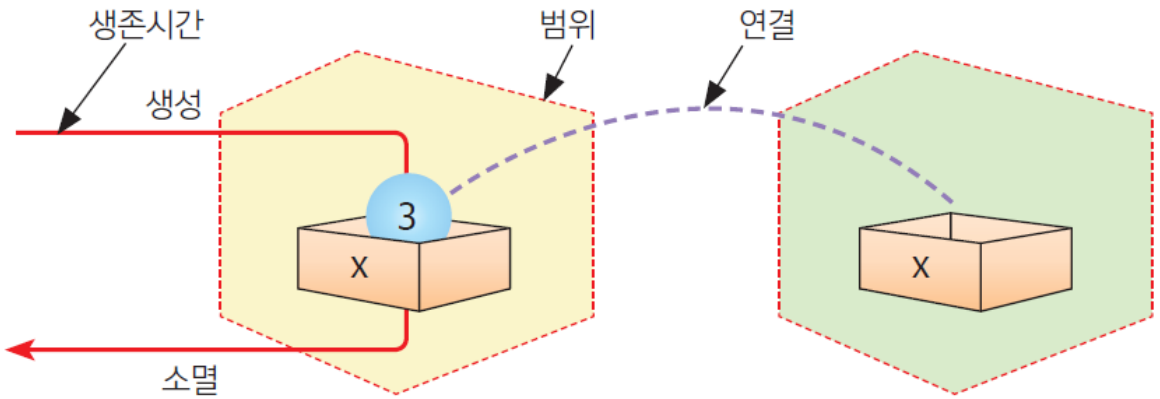


변수의 속성

○**변수의 속성 :**

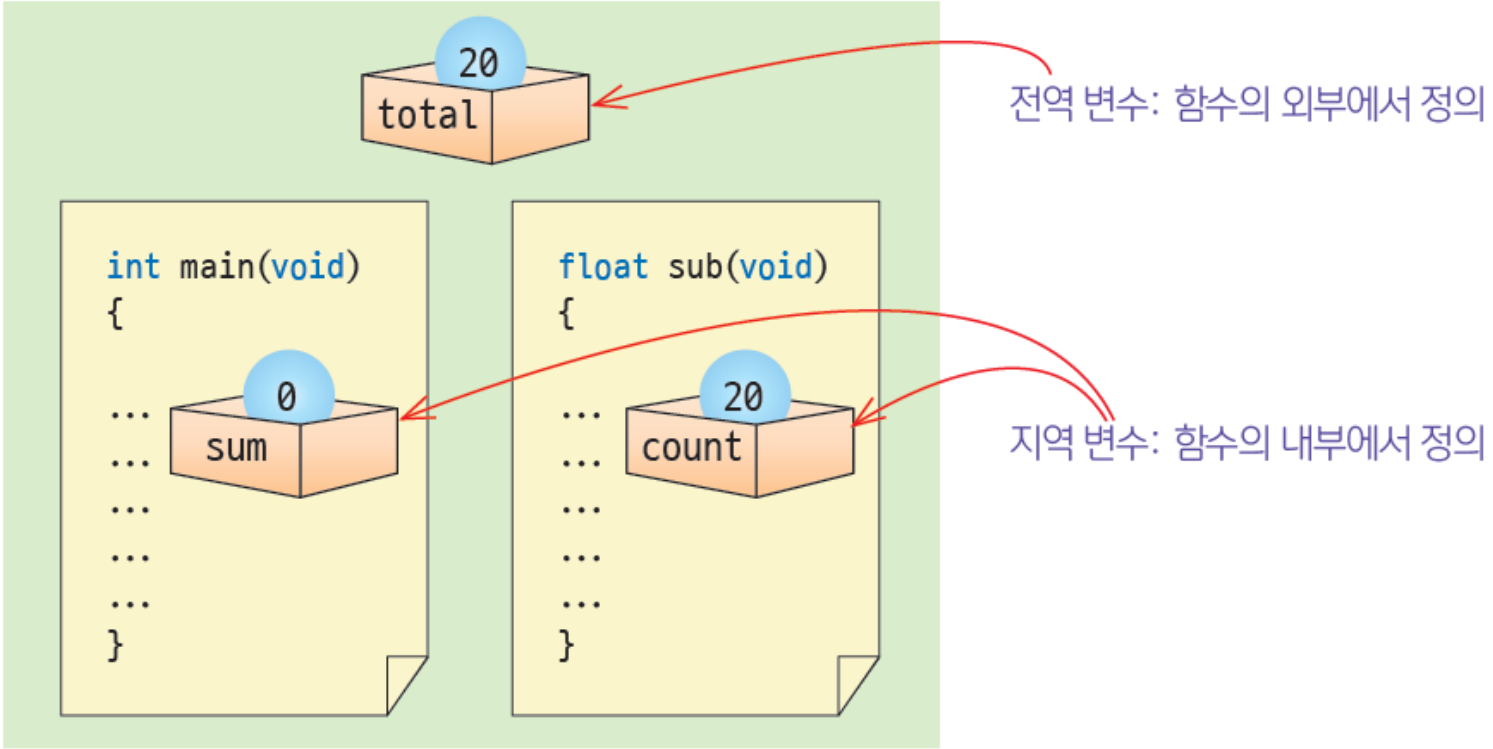
이름, 타입, 크기, 값 + 범위, 생존 시간, 연결

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



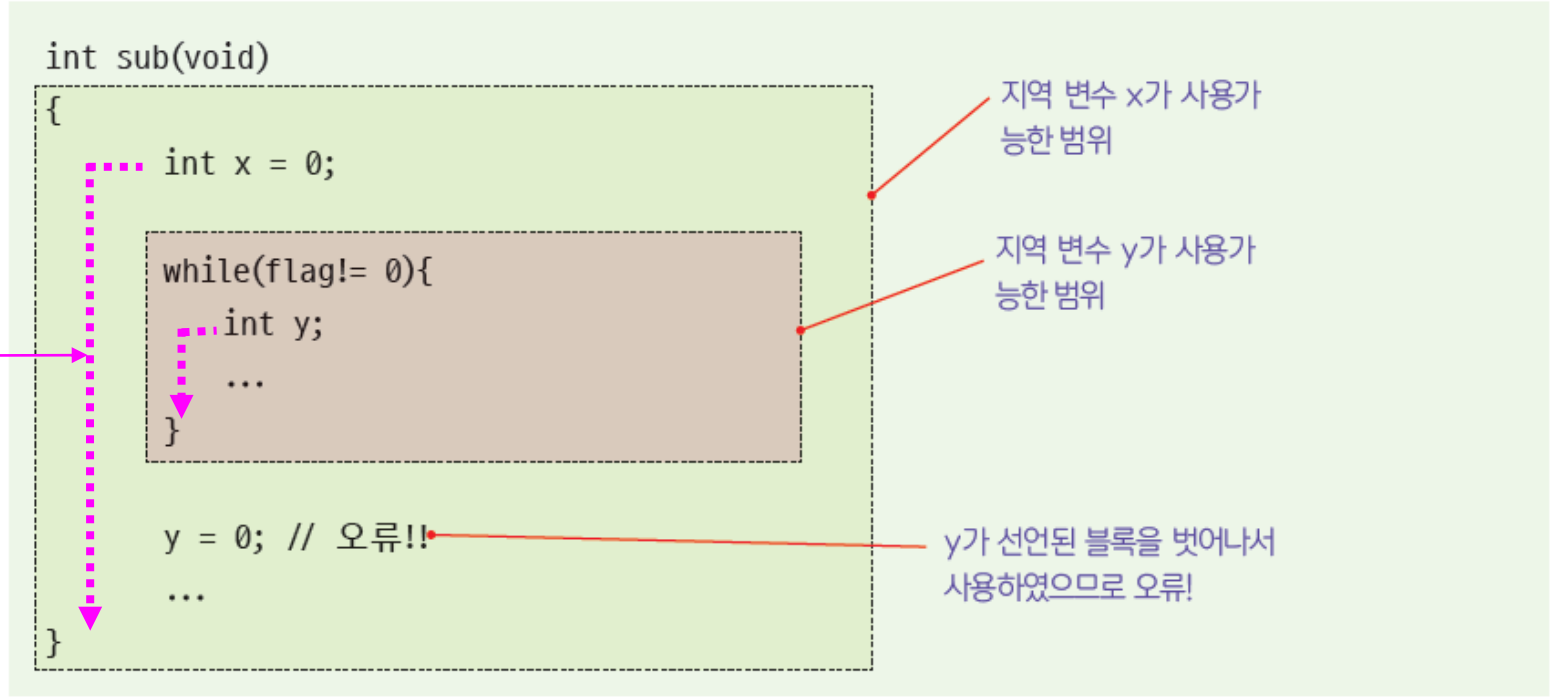
전역 변수와 지역 변수

- 전역 변수(Glabal Variable) : 소스 파일의 어느 곳에서나 사용 가능
- 지역 변수(Local Variable) : 변수가 선언된 블록 내에서만 사용 가능



지역 변수

○지역 변수(local variable) : 블록 안에 선언되는 변수



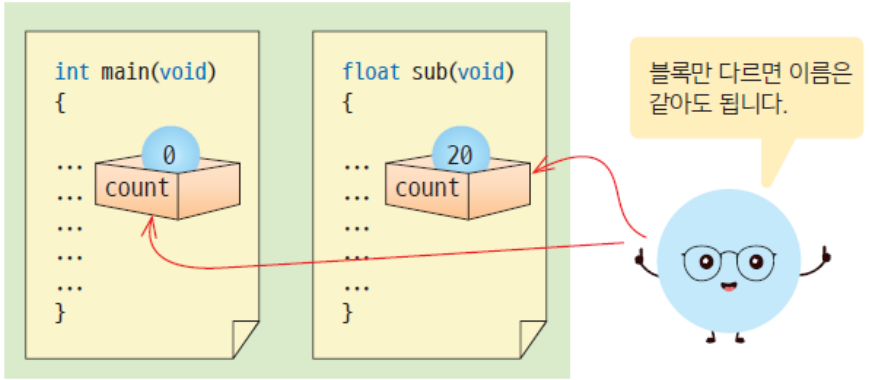
지역 변수 선언 위치

○최신 버전의 C에서는 블록 안의 어떤 위치에서도 선언 가능!!

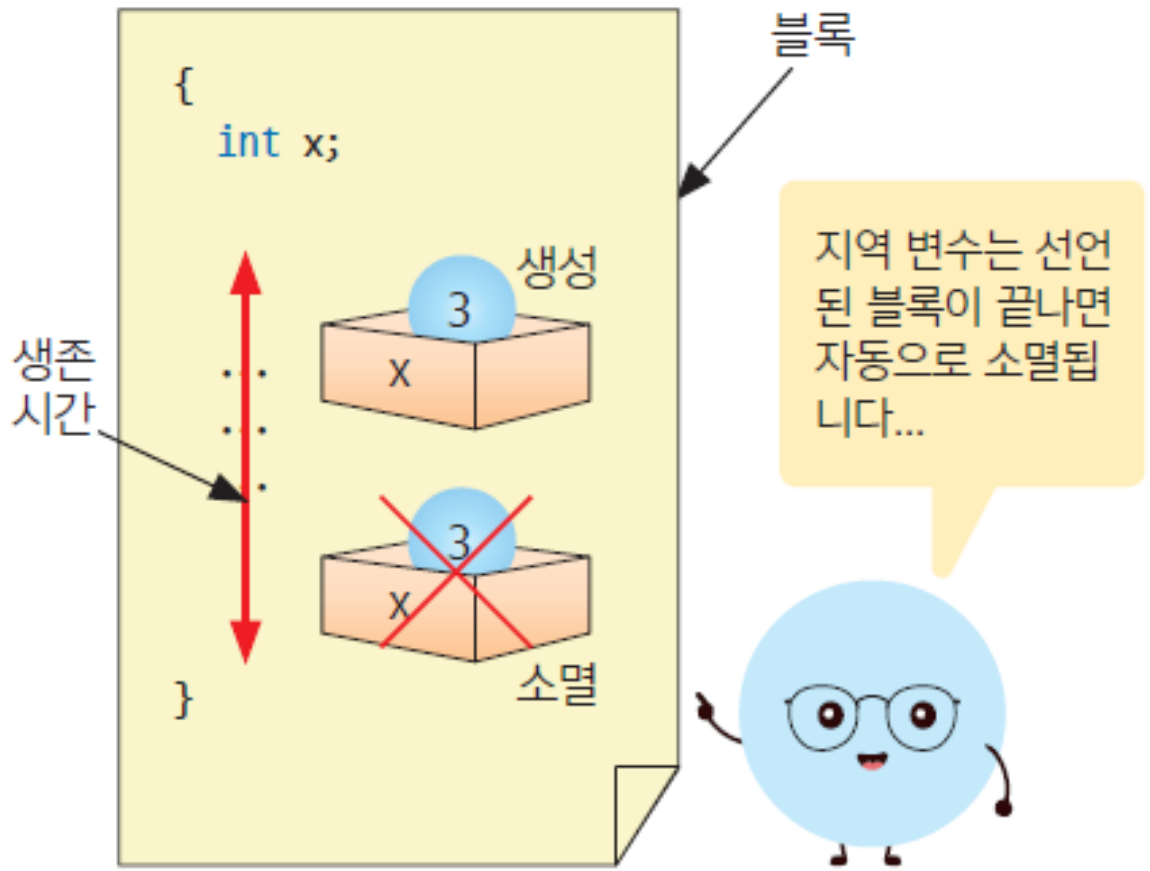
○선언된 블

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

블록의 중간에서도 얼마든지 지역 변수를 선언할 수 있다.



지역 변수의 생존 기간



지역 변수의 사용 예

```c
#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
```

ex1.c :

local.c*

Project7강(23)

```c
#include <stdio.h>

void main()
{
    int x = 5;

    while (x > 0) {
        int y = 1;

        x = x - y;
        printf("y = %d \n", y);
    }

    printf("x = %d \n", x);
    //printf("y = %d \n", y);
}
```

E0020 identifier "y" is undefined

C2065 'y': undeclared identifier

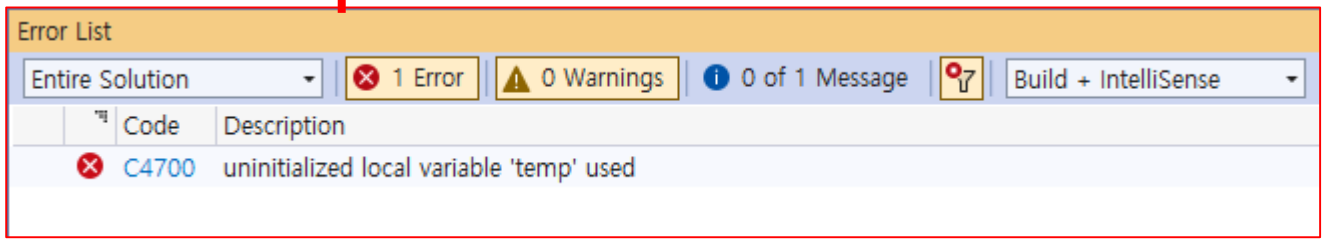
Microsoft Visual Studio Debug Console

```
y = 1
y = 1
y = 1
y = 1
y = 1
x = 0

H:₩강의록(신한)₩2021년_2학기_동영상강의₩C프로그래밍₩실습소스코드₩Pro
988) exited with code 0.
Press any key to close this window . . .
```

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 변수

- 지역 변수의 모든 특징 적용
- 지역 변수와 다른 점 : 함수 호출시의 인수 값으로 초기화

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

매개 변수도 일종의 지역 변수

함수의 매개 변수

ex2.c :

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

int main(void)
{    int i;

     i = 10;
     printf("함수 호출 전 i=%d\n", i);

     inc(i);
     printf("함수 호출 후 i = %d \n", i);
     return 0;
}

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

값에 의한 호출
(call by value)

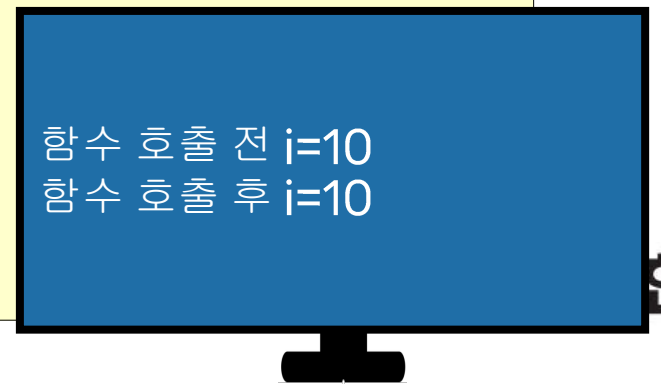
매개 변수도 일종의
지역 변수임

i : 10

변수 i의 값을 복사

counter : 10

counter : 11

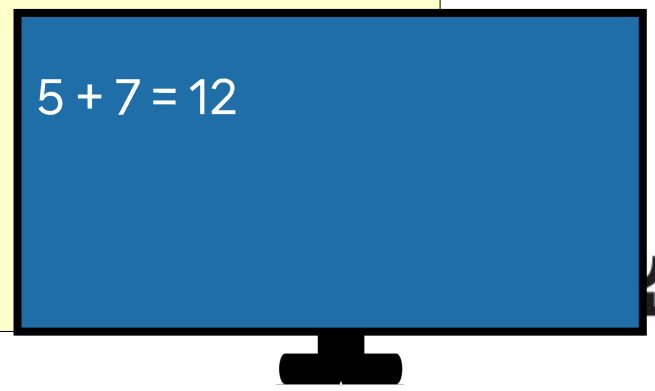


전역 변수의 초기값과 생존 기간

```
#include<stdio.h>
int a;
int b;

int add()
{
    return a + b;     //전역변수 a, b 사용
}

int main(void)
{
    int answer;

    a = 5;        //전역변수
    b = 7;        //전역변수
    answer = add();
    printf(" % d + % d = % d\n", a, b, answer);

    return 0;
}
```

전역 변수 초기값은 0

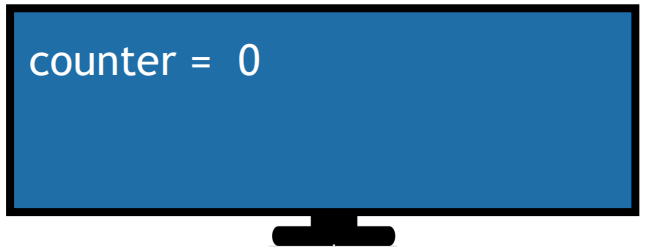
전역 변수의 범위



전역 변수의 초기값

```c
#include <stdio.h>

int counter;

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

전역 변수는 컴파일러가 프로그램 실행시에 0으로 초기화



전역 변수의 사용

○거의 모든 함수에서 사용하는 공통적인 데이터는 전역 변수로 한다.

○일부 함수들만 사용하는 데이터

- 전역 변수로 하지 말고 함수의 인수로 전달한다.

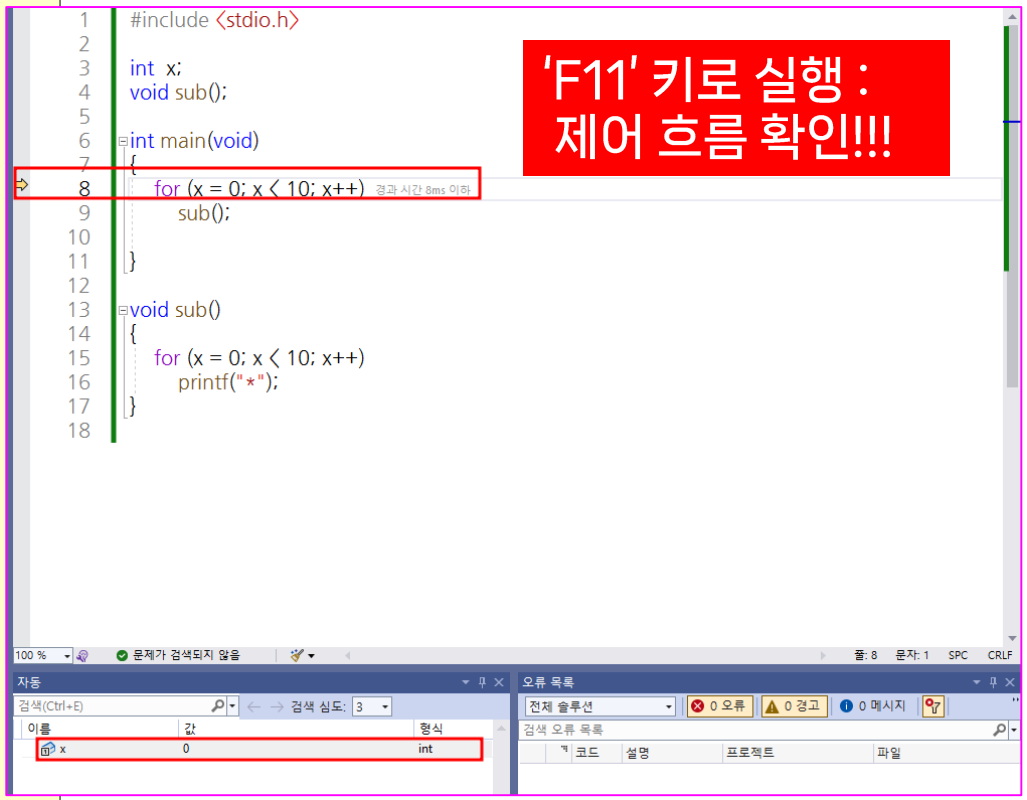
전역 변수의 사용(1)

ex3.c :

스파게티 코드
(spaghetti code)

'F11' 키로 실행 :
제어 흐름 확인!!!

```c
#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(2)

global.c | local.c

Project7강(23)

```
1  #include <stdio.h>
2
3  int x, k;
4  void sub();
5
6  int main(void)
7  {
8      for (k = 0; k < 10; k++)
9      {
10         sub();
11         printf("\n");
12
13     }
14 }
15
16 void sub()
17 {
18     for (x = 0; x < 10; x++)
19         printf("*");
20 }
21
```

Microsoft Visual Studio Debug Console

```
**********
**********
**********
**********
**********
**********
**********
**********
**********
**********
```

전역 변수(k)를 호출해서 사용

global.c | local.c

Project7강(23)

```
1  #include <stdio.h>
2
3  int x;
4  void sub();
5
6  int main(void)
7  {
8      for (int x = 0; x < 10; x++)
9      {
10         sub();
11         printf("\n");
12
13     }
14 }
15
16 void sub()
17 {
18     for (x = 0; x < 10; x++)
19         printf("*");
20 }
21
```

Microsoft Visual Studio Debug Console

```
**********
**********
**********
**********
**********
**********
**********
**********
**********
**********
```

For() 블록의 지역 변수(x)로 명시적 선언해서 사용

전역변수 x

신한대학교

같은 이름의 전역 변수와 지역 변수

전역 변수와 지역 변수가 동일한
이름으로 선언!!

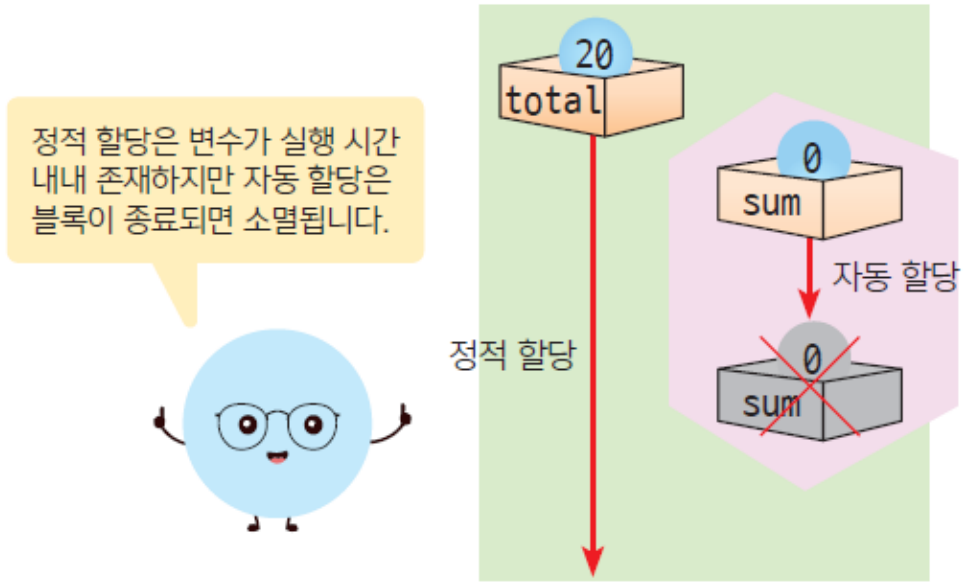
```c
#include <stdio.h>

int sum = 1;        // 전역 변수

int main(void)
{
    int sum = 0;        // 지역 변수

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

    return 0;
}
```

```
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



생존 기간

○생존 기간을 결정하는 요인
- ➢변수가 선언된 위치
- ➢저장 유형 지정자

○저장 유형 지정자
- ➢auto(자동 변수) : 함수나 블록 내에서 선언된 지역 변수
  - 변수를 선언한 위치에서 생성 / 선언된 블록을 벗어나면 자동 소멸
  - auto 예약어 생략 가능
- ➢static(정적 변수) : 키워드 static 사용해서 변수 선언
  - 프로그램이 실행될 때 생성 / 프로그램이 종료되면 소멸
- ➢register(레지스터 변수) : 데이터를 레지스터에 저장하는 변수
  - 키워드 register 사용해서 변수 선언 / 지역 변수에만 사용 가능
  - CPU를 사용하므로 처리 속도가 매우 빠름(루프 제어 변수 사용->성능 향상)
  - 주소 연산자(&) 사용 불가
- ➢extern(외부 참조 변수) : 블록에서 전역 변수에 접근
- ➢volatile(휘발 변수) : 하드웨어가 수시로 변수 값을 변경하는 경우에 사용
  - volatile 로 지정된 문장을 실행할 때마다 변수의 새로운 값을 메모리에서 읽어온다.

auto 변수

- 변수를 선언한 위치에서 자동으로 만들어지고 블록을 벗어나면 자동으로 소멸되는 저장 유형을 지정
- 지역 변수는 **auto** 생략 가능

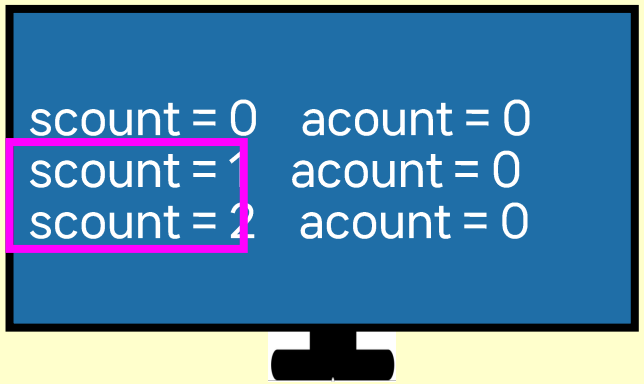
```
int main(void)
{
  auto int sum = 0;
  int i = 0;
  ...
  ...
}
```

전부 자동 변수로서 함수가 시작되면 생성되고, 끝나면 소멸된다.

static 변수

```
#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 변수**로 된다.

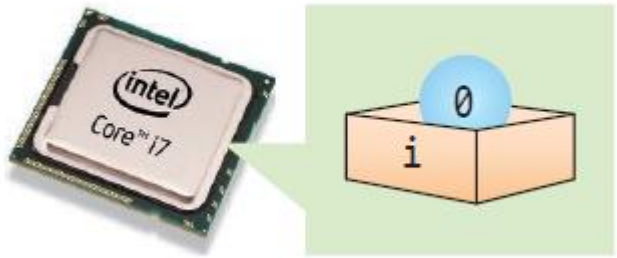


register 변수

○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 중지

예제1 : 은행 계좌 구현하기

○돈이 생기면 저금하는 사람의 저축 잔고를 확인하는 함수 save(int amount)를 작성하여 보자.

- amount : 저금할 금액
- 함수 호출 방법 : save(100) ➔ 저축액 100원
- save() 함수 : 정적 변수를 사용 ➔ 현재까지 저축된 총액을 기억
  - 함수가 호출될 때마다 총 저축액을 다음과 같이 화면에 출력한다.

소스

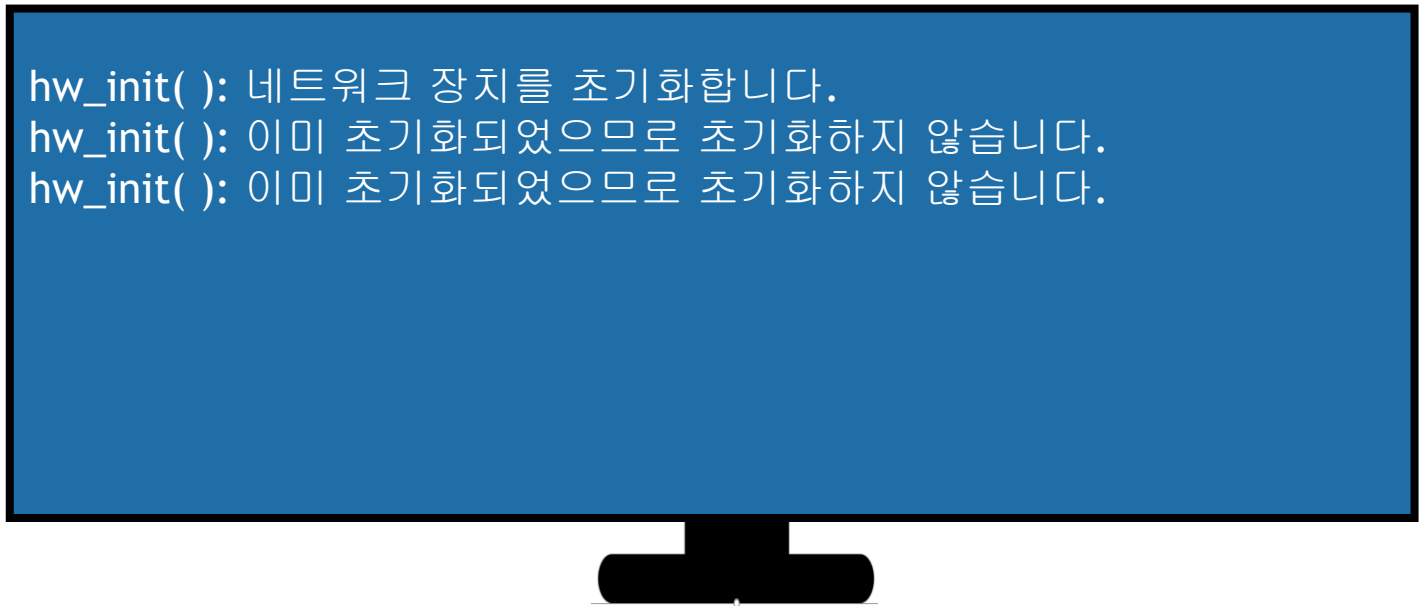
```c
#include <stdio.h>

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

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

	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;
}
```

예제2: 한번만 초기화하기

○정적 변수는 한번만 초기화하고 싶은 경우에도 사용



소스

ex4.c :

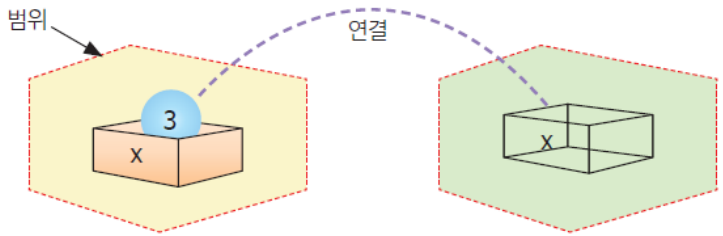
```c
#include <stdio.h>
#include <stdlib.h>

void hw_init();

int main(void)
{
        hw_init();
        hw_init();
        hw_init();

        return 0;
}

void hw_init()
{       static int inited = 0;

        if( inited == 0 ){
                printf("hw_init(): 네트워크 장치를 초기화합니다. \n");
                inited = 1;
        }
        else {
                printf("hw_init(): 이미 초기화되었으므로 초기화하지 않습니다. \n");
        }
}
```

연결

○ *연결(linkage):* **다른 범위에 속하는 변수들을 서로 연결하는 것**

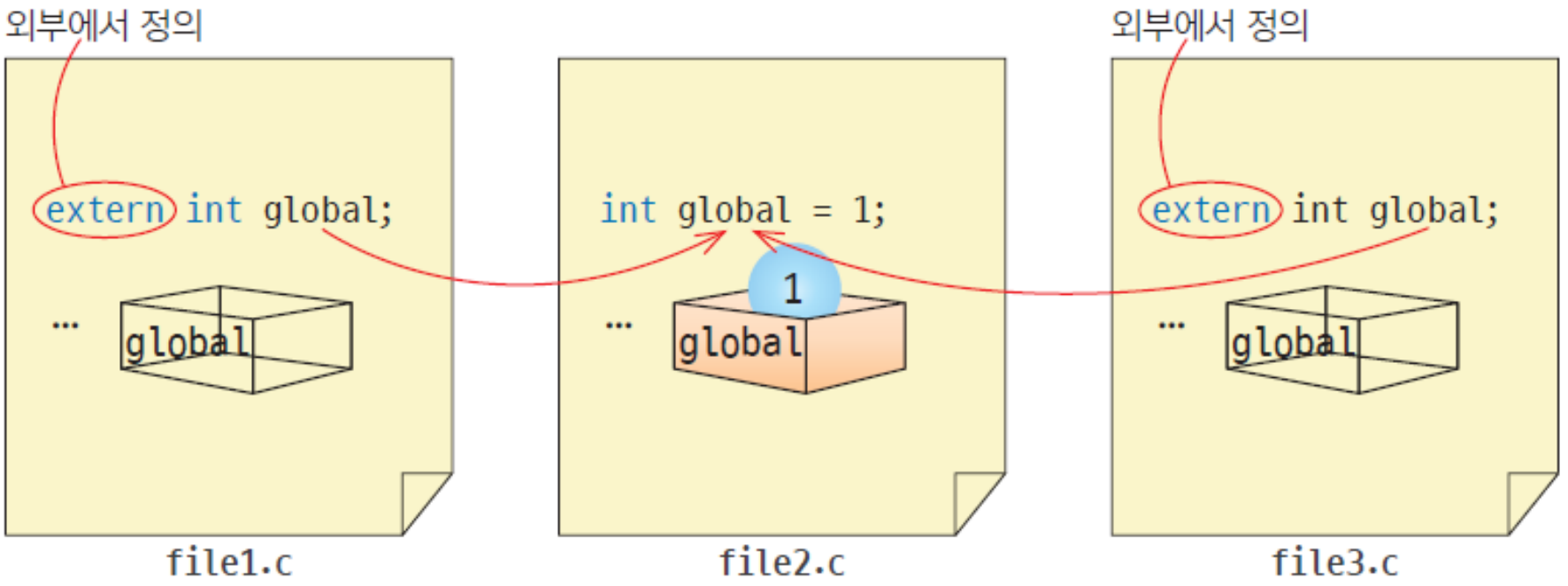
- ➢외부 연결 : 한 개 프로그램이 여러 개의 소스 파일로 구성되어 있을 때 사용
- ➢내부 연결 : static으로 정의되면 하나의 소스 파일 안에서만 사용 가능
- ➢무연결

○ **전역 변수만이 외부 연결을 가질 수 있다.**



외부 연결

○전역 변수를 extern을 이용하여서 서로 연결



연결 예제

linkage1.c

```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

```c
extern int all_files;

// extern int this_file;

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

link2.c | link1.c | global.c | local.c

Project7강(23)

```c
#include <stdio.h>

int all_files;
static int this_file = 5;
extern void sub();

int main()
{
    sub();

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

    return 0;
}
```

링킹 중단

link2.c | link1.c | global.c | local.c

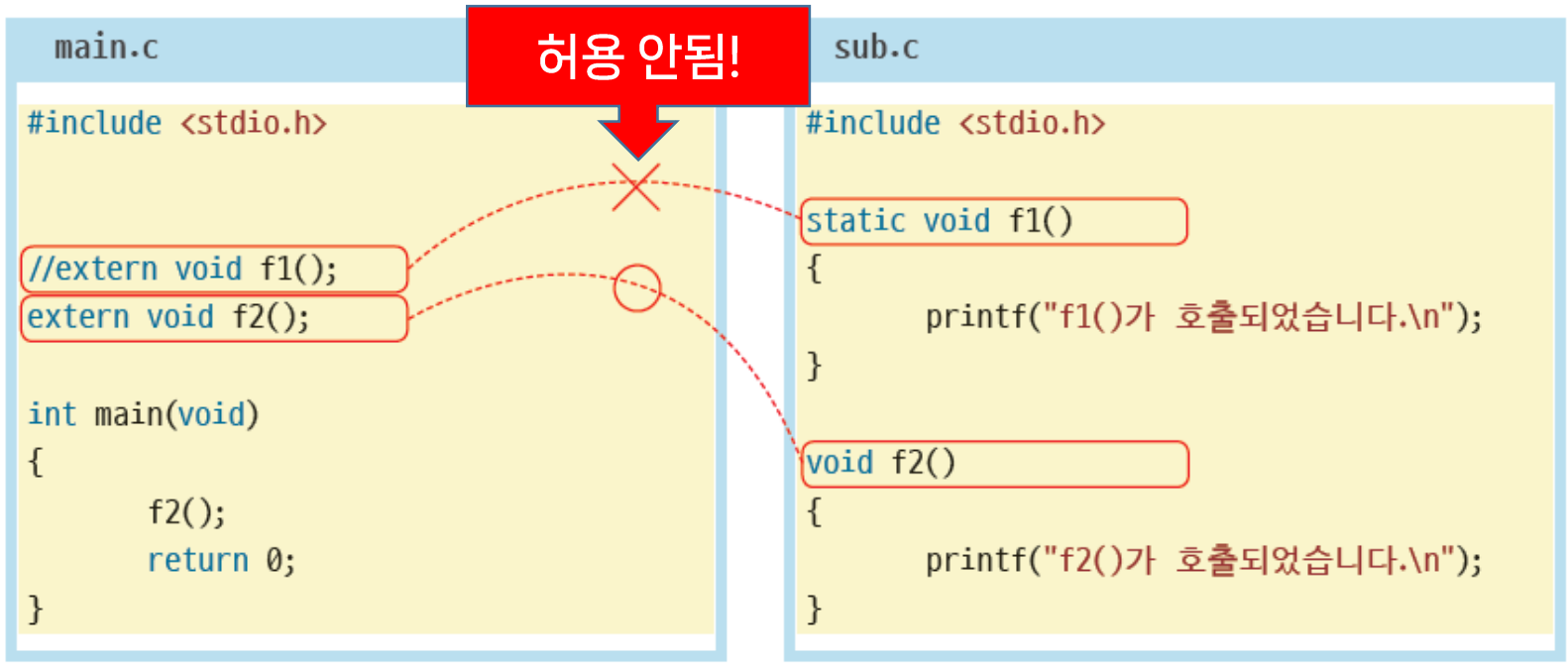
```c
extern int all_files;
extern int this_file;

void sub()
{
    all_files = 10;

    printf("%d \n", this_file);
}
```

Error List

Entire Solution | 2 Errors | 1 Warning | 0 of 1 Message | Build + Intel

Code	Description
C4013	'printf' undefined; assuming extern returning int
LNK2001	unresolved external symbol this_file
LNK1120	1 unresolved externals

함수앞의 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;
}
```

어떤 저장 유형을 사용하여 하는가?

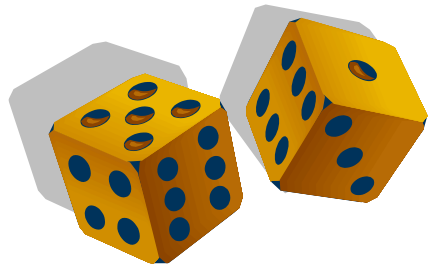
- 일반적으로는 ***자동 저장 유형*** 사용 권장
- 변수의 값이 함수 호출이 끝나도 그 값을 유지하여야 할 필요가 있다면 ***지역 정적***
- 만약 많은 함수에서 공유되어야 하는 변수라면 ***외부 참조 변수***

저장 유형	키워드	정의되는 위치	범위	생존 시간
자동	auto	함수 내부	지역	임시
레지스터	register	함수 내부	지역	임시
정적 지역	staic	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

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



실행 결과

○다음과 같은 함수를 작성하여 사용해 보자.

- 0 ~ M-1 사이의 난수를 생성하는 random_i()
- 0.0 ~ 1.0 사이의 난수를 생성하는 random_f()

○0 ~ 100사이의 난수를 몇 개 생성해보자.

예제

random1.c

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

static unsigned int seed = SEED;                    // 난수 생성 시드값(17)

// 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

```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;
}
```

```
Microsoft Visual Studio 디버그

15819 8185 15958 1163 29117 11067 18406 22707 29612 20242
H:\강의록(신한)\2025_2학기\소스파일\Project7강\x64\Debug\Pro
```

```c
for (i = 0; i < 10; i++)
    printf("%lf ", random_f());
```

```
Microsoft Visual Studio 디버그

15819 8185 15958 1163 29117 11067 18406 22707 29612 20242
0.186102 0.165044 0.077761 0.904386 0.519089 0.458296 0.119358 0.018830 0.428236 0.400555
H:\강의록(신한)\2025_2학기\소스파일\Project7강\x64\Debug\Project7강.exe(프로세스 26592개)이
```

가변 매개 변수

○매개 변수의 개수가 가변적으로 변할 수 있는 기능

- ➢함수 원형의 매개 변수 목록에 '매개변수 개수, …'으로 변수 목록 표현
- ➢ 함수 안에서 va_list 타입의 변수 선언
- ➢함수 안에서 va_start()를 호출해서 가변 매개변수 기능 시작
- ➢ va_arg()가 호출될 때마다 인수들이 하나씩 반환됨
- ➢ va_end(va_list 타입의 변수명)를 호출해서 종료

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

호출 때 마다 매개 변수의 개수가 변경될 수 있다.

가변 매개 변수

ex5.c :

```c
#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, ... )        //가변 매개변수 함수 헤더 선언
{
        int answer = 0;
        va_list argptr;

        va_start(argptr, num);      // 가변 매개변수 기능 시작

        for( ; num > 0; num-- )
            answer += va_arg(argptr, int); // 각각의 4개 인수(4, 3, 2, 1) 추출해서 합산

        va_end( argptr );

        return( answer );
}
```

매개 변수의 개수

순환(recursion)이란?

○순환(recursion) : 함수가 자기 자신을 호출하는 것

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

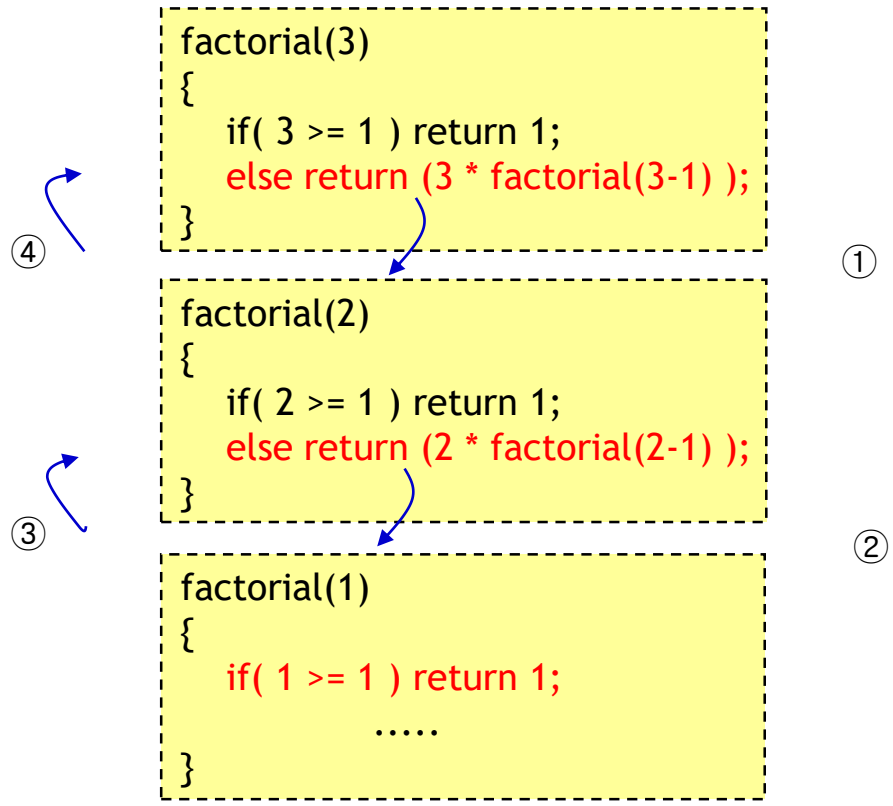
```
int factorial(int n)
{
   if( n <= 1 )
         return(1);
   else
         return (n * factorial(n-1) );
}
```

0! = 1
1! = 1 * (0!)
2! = 2 * (1!)
3! = 3 * (2!)
:

팩토리얼 구하기

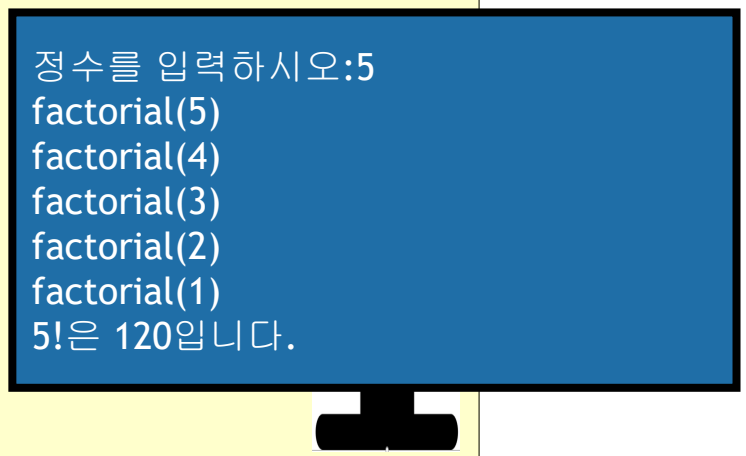
○팩토리얼의 호출 순서

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



팩토리얼 계산

```c
// 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;
}
```



최대 공약수 구하기

○최대 공약수를 구하는 방법으로 유클리드의 호제법 사용.
이 방법은 두 수 x와 y의 최대 공약수는 y와 (x % y)의 최대 공약수와 같으며 x와 0의 최대 공약수는 x라는 것이다

```
gcd(x, y) = gcd(y, x % y)
gcd(x, 0) = x
```

```c
// 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);
}
```

7강 – 정리 요약

○지역 변수 / 전역 변수 선언

- ➢ 전역 변수 : main() 함수 바깥에서 선언
- ➢ 지역 변수 : 블록 안에서 선언

○변수의 생존 기간

➢정적 할당(static allocation):

- 프로그램 실행 시간 동안 계속 유지

➢자동 할당(automatic allocation):

- 블록에 들어갈 때 생성
- 블록에서 나올 때 소멸

7강 – 정리 요약

○변수의 저장 유형

➢auto(자동 변수) : 함수나 블록 내에서 선언된 지역 변수
- 변수를 선언한 위치에서 생성 / 선언된 블록을 벗어나면 자동 소멸
- auto 예약어 생략 가능

➢static(정적 변수) : 키워드 static 사용해서 변수 선언
- 프로그램이 실행될 때 생성 / 프로그램이 종료되면 소멸

➢register(레지스터 변수) : 데이터를 레지스터에 저장하는 변수
- 키워드 register 사용해서 변수 선언 / 지역 변수에만 사용 가능
- CPU를 사용하므로 처리 속도가 매우 빠름(루프 제어 변수 사용->성능 향상)
- 주소 연산자(&) 사용 불가

➢extern(외부 참조 변수) : 블록에서 전역 변수에 접근

➢volatile(휘발 변수) : 하드웨어가 수시로 변수 값을 변경하는 경우에 사용
- volatile 로 지정된 문장을 실행할 때마다 변수의 새로운 값을 메모리에서 읽어온다.


신 한 대 학 교
SHINHAN UNIVERSITY