프로그래밍 기초

10장. 변수 유효 범위

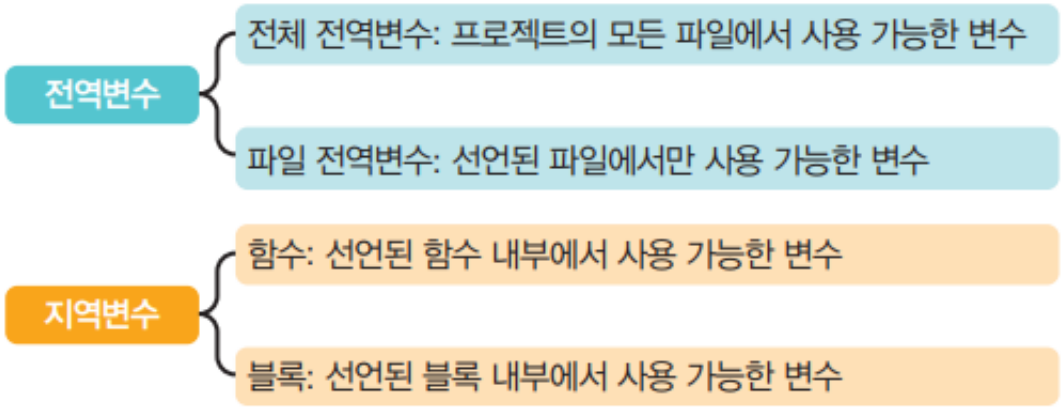
학습목표

- **변수 유효범위를 이해하고 설명할 수 있다.**
  - 지역 변수, 자동변수를 선언하고 사용
  - 전역 변수를 선언하고 사용
  - 키워드 extern의 필요성과 사용 방법

- **정적 변수와 레지스터 변수를 이해하고 설명할 수 있다.**
  - 기억 부류 auto, static, register, extern
  - 지역 정적 변수와 전역 정적 변수의 필요성과 사용

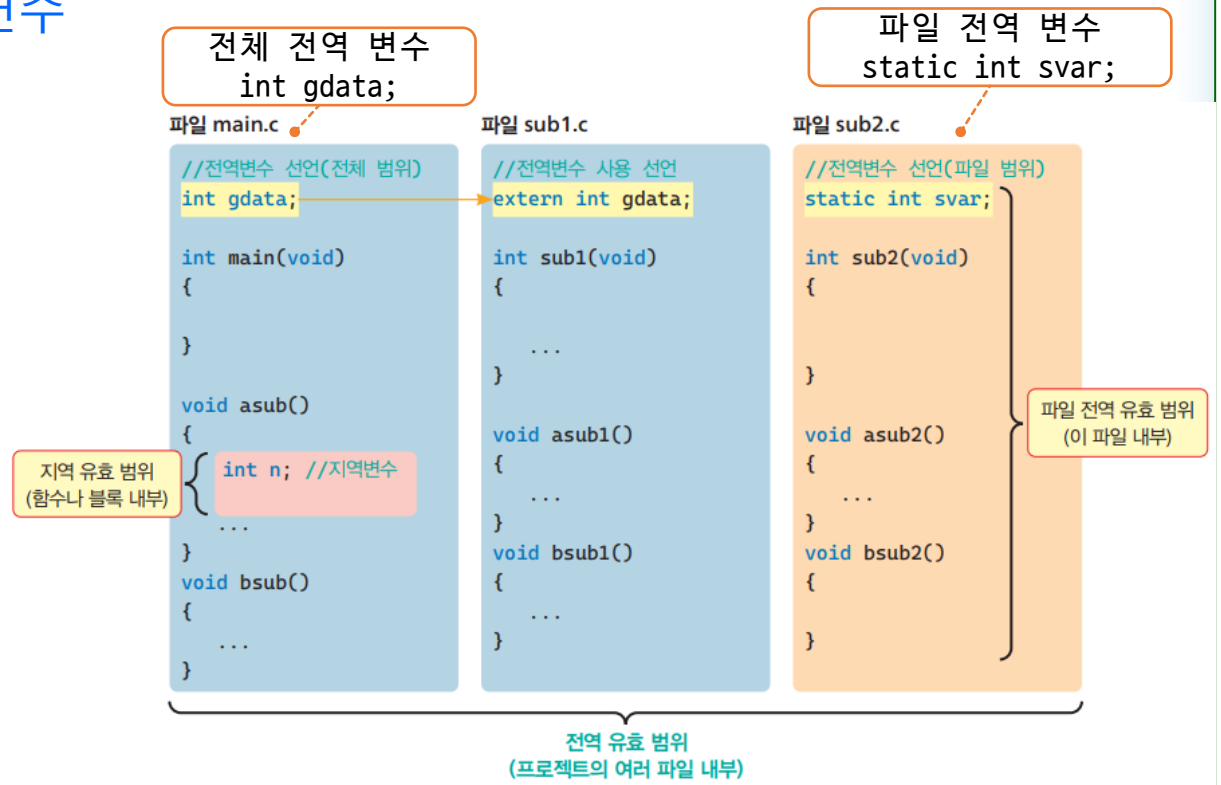
변수 범위와 지역 변수

- **변수의 유효 범위(scope)**
  - 선언된 변수가 사용될 수 있는 범위
  - 지역 유효 범위(local scope)와 전역 유효 범위(global scope)로 나눔
- **지역 유효 범위**
  - 함수 또는 블록 내부에서 선언된 변수는 해당 함수나 블록에서만 사용 가능
- **전역 유효 범위**
  - 프로젝트나 파일에서 사용 가능한 범위
  - 하나의 파일에서만 변수의 참조가 가능할 수 있으며
  - 또는 프로젝트를 구성하는 모든 파일에서 변수의 참조가 가능



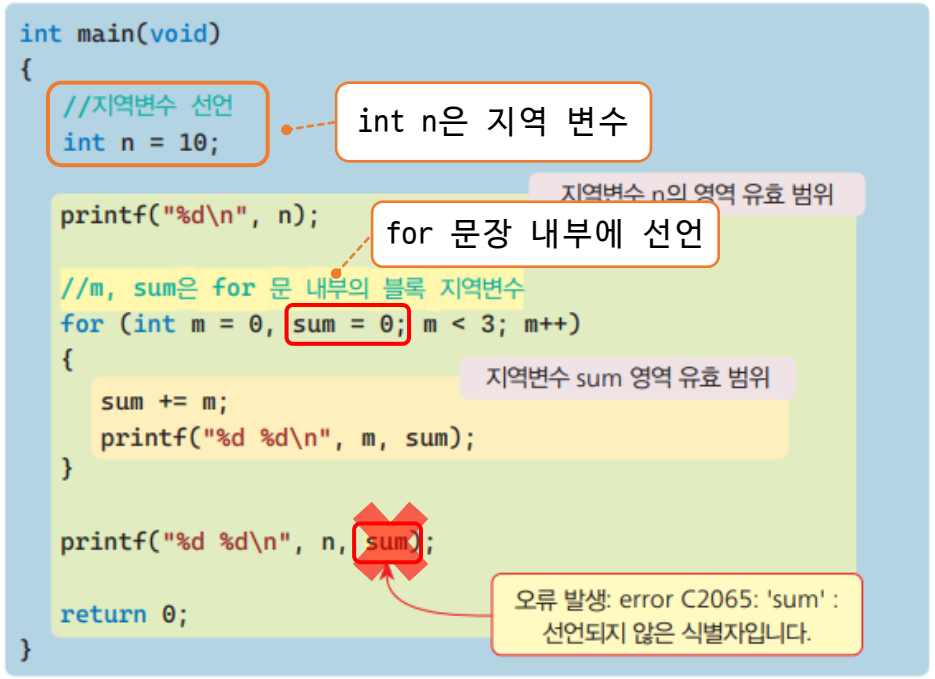
지역과 전역

- **프로젝트**
  - 파일 main.c, sub1.c와 sub2.c, 3개의 소스 파일로 구성
- **지역 변수**
  - 파일 main.c의 함수 asub()에서 선언된 변수 n은 함수 내부에서만 사용
  - 모든 함수 내부에서 선언된 변수는 모두 지역 변수
- **전체 전역 변수**
  - 파일 main.c의 상단에 선언된 int gdata는 프로젝트 전체 파일에서 사용
  - gdata를 파일 sub1.c에서 사용하려면
    - ➢ 먼저 extern int gdata로 선언이 필요
- **파일 전역 변수**
  - 파일 sub2.c의 상단에 선언된 svar는 파일 sub2.c에서만 사용
  - static int svar
    - ➢ static 전역 변수는 선언된 파일에서만 사용 가능



지역 변수(local variable)

- **지역 변수**
  - 함수 또는 블록 내부에서 선언된 변수
  - 선언 문장 이후에 함수나 블록의 내부에서만 사용이 가능
    - ➢ **함수의 매개변수**도 함수 전체에서 사용 가능한 **지역 변수**
  - 선언 후 초기화하지 않으면 쓰레기 값이 저장되므로 주의
  - 지역 변수는 스택(stack) 메모리 영역에 할당
    - ➢ 함수나 블록이 종료되는 순간 메모리에서 자동으로 제거
      - **이러한 이유에서 지역 변수는 자동변수(automatic variable)**
    - ➢ 지역 변수 선언에서 자료형 앞에 키워드 auto가 사용 가능
      - **키워드 auto는 생략 가능**



지역 변수 선언과 사용 (실습)

<01localvar.c>

```c
#include <stdio.h>
void sub(int param); // 함수 원형

int main(void)
{
    int n = 10; // 지역 변수 선언
    printf("%d\n", n);

    for (int m = 0, sum = 0; m < 3; m++)
    {
        sum += m;
        printf("\t%d %d\n", m, sum);
    }
    printf("%d\n", n); // n 참조 가능
    printf("%d %d\n", m, sum); // m, sum 참조 불가능
    sub(20); // 함수호출

    return 0;
}

void sub(int param) // 매개변수인 param도 지역 변수로 사용
{
    int local = 100; // 지역 변수 local
    printf("\t%d %d\n", param, local); // param과 local 참조 가능
    printf("%d\n", n); //n 참조 불가능
}
```

m, sum은 for문 내부에 선언된 지역 변수

```
01localvar.c: In function 'main':
01localvar.c:18:27: error: 'm' undeclared (first use in this function)
   18 |         printf("%d %d\n", m, sum); // m, sum 참조 불가능
      |                           ^
01localvar.c:18:27: note: each undeclared identifier is reported only once for each function it appear
s in
01localvar.c:18:30: error: 'sum' undeclared (first use in this function); did you mean 'sub'?
   18 |         printf("%d %d\n", m, sum); // m, sum 참조 불가능
      |                              ^~~
      |                              sub
```

```
01localvar.c: In function 'sub':
01localvar.c:29:24: error: 'n' undeclared (first use in this function)
   29 |         printf("%d\n", n);                              // n 참조 불가능
      |                        ^
```

전역 변수(global variable)

- **전역 변수(global variable)**
  - 함수 외부에 선언된 변수
  - 프로젝트의 모든 함수나 블록에서 참조 가능
  - 선언되면 자동으로 자료형에 맞는 0으로 초기값이 지정
    - 정수형은 0, 문자형은 null 문자인 '\0', 실수형은 0.0, 포인터형은 NULL값이 저장

- **함수나 블록에서 전역 변수와 같은 이름으로 지역 변수 선언 가능**
  - 함수 내부나 블록에 선언된 지역 변수로 인식(절대 사용 금지)

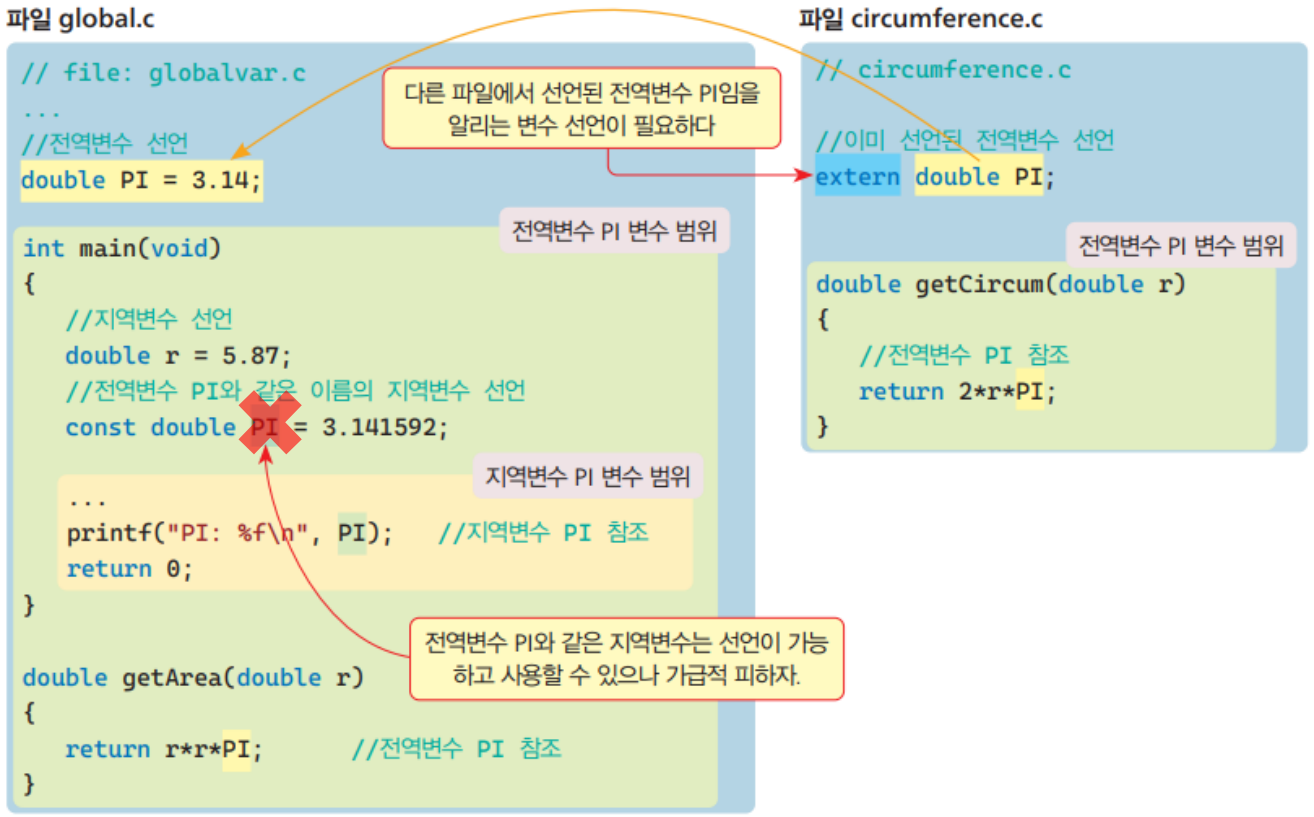
    - 동일한 이름의 전역 변수는 참조 불가능: 이런 변수는 사용하지 않도록 해야 됨

- **전역 변수는 동일 프로젝트의 다른 파일에서 참조 가능**
  - 다른 파일에 선언된 전역 변수를 참조
    - 키워드 extern을 사용: 다른 파일에 선언된 전역 변수임을 선언

- **전역 변수 사용 단점**
  - 프로그램 어느 부분에서 수정이 되었는지 알기 어려움
  - 전역 변수는 가능한 제한적으로 사용하는 것이 바람직함

전역 변수 참조

- **프로젝트의 다른 파일에서도 참조가 가능**
  - 키워드 extern을 사용하여 이미 다른 파일에서 선언된 전역 변수임을 선언
    - ➢ extern 참조선언 구문에서 자료형 double 등은 생략 가능
    - ➢ 이미 존재하는 전역 변수의 유효 범위를 확장



전역 변수와 지역 변수의 선언과 사용

<02-1global.c>

```c
#include <stdio.h>

double getArea(double);
extern double getCircum(double);

double PI = 3.14;
int gi;

int main(void)
{
    double r = 5.87;
    //전역 변수 PI와 같은 이름의 지역 변수 선언
    const double PI = 3.141592;

    printf("면적: %.2f\n", getArea(r));
    printf("둘레1: %.2f\n", 2 * PI * r);
    printf("둘레2: %.2f\n", getCircum(r));
    printf("PI: %f\n", PI); //지역 변수 PI 참조
    printf("gi: %d\n", gi); //전역 변수 gi 기본 값

    return 0;
}

double getArea(double r)
{
    return r * r * PI; //전역 변수 PI 참조
}
```

전역 변수(gi)는 초기값을 지정하지 않으면 자동으로 기본값인 0이 저장

지역 변수 PI 선언
- 전역 변수 PI는 참조 불가능

<02-2circumference.c>

```c
//이미 외부에서 선언된 전역 변수임을 알리는 선언
extern double PI;

double getCircum(double r)
{
    return 2 * r * PI; //전역 변수 PI 참조
}
```

컴파일

```
$ gcc 02-1global.c 02-2circumference.c -o circle
$ ./circle
```

실행 결과

```
면적: 108.19
둘레1: 36.88
둘레2: 36.86
PI: 3.141592
gi: 0
```

LAB: 1에서 100 사이의 난수 알아 맞히기를 두 파일로 구현

<lab1-1numguess.c>

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

#define MAX 100

int guess; //정답인 전역 변수 선언
extern int test_num(int input);

int main(void)
{
    int input; //지역 변수 선언

    srand(time(NULL));
    guess = rand() % MAX + 1; //정답 지정

    printf("1에서 %d 사이에서 한 정수가 결정되었습니다.\n", MAX);
    printf("이 정수는 무엇일까요? 입력해 보세요. : ");
    while(1) {
        scanf("%d", &input);
        int result = test_num(input);
        if(result == 0)
        {
            printf("정답입니다.\n");
            break;
        }
        else if(result == -1)
            printf("입력한 수보다 작습니다. 다시 입력하세요. : ");
        else if(result == 1)
            printf("입력한 수보다 큽니다. 다시 입력하세요. : ");
    }
    printf("프로그램 종료\n");
    return 0;
}
```

<lab1-2testnum.c>

```c
extern int guess; //전역 변수 선언
int test_num(int input)
{
    int result = 0;
    if (input > guess)
        result = -1;
    else if (input < guess)
        result = 1;
    else
        result = 0;
    return result;
}
```

컴파일 및 실행

```
% gcc lab1-1numguess.c lab1-2testnum.c -o lab1
% ./lab1
1에서 100 사이에서 한 정수가 결정되었습니다.
입력한 수보다 큽니다. 다시 입력하세요. : 37
입력한 수보다 큽니다. 다시 입력하세요. : 38
정답입니다.
프로그램 종료
```

변수의 4가지 기억 부류

- **변수 4가지 기억 부류(storage class)**
  - auto, register, static, extern
    - 할당되는 메모리 영역이 결정되고 메모리의 할당과 제거 시기가 결정
  - 기억 부류는 키워드 auto, register, static, extern에 의해 구분
    - 자동변수인 auto는 일반 지역 변수로 생략 가능
  - auto와 register
    - 지역 변수에만 이용이 가능
  - static
    - 지역과 전역 모든 변수에 이용 가능
  - extern
    - 전역 변수에만 사용이 가능
    - 컴파일러에게 변수가 이미 어딘가 (주로 다른 파일)에 존재하고 이제 사용하겠다는 것을 알리는 구문에 사용되는 키워드
    - extern이 선언되는 위치에 따라 이 변수의 사용의 범위는 전역 또는 지역으로 한정
  - 기억 부류 auto, register, static
    - 새로운 변수의 선언에 사용되는 키워드

변수의 4가지 기억 부류

- **사용 구문**
  - 변수 선언 문장에서 자료형 앞에 하나의 키워드를 넣는 방식
  - 키워드 extern을 제외하고 나머지 3개의 기억 부류는 변수선언에서 초기값을 저장 가능
    - ➢ extern 으로 변수 선언 시 초기값 지정하지 않음
  - 키워드 auto는 지역 변수 선언에 사용되며 생략 가능
  - 함수에 선언된 모든 변수가 auto가 생략된 자동 변수(지역 변수)

기억 부류 변수 선언

```
기억 부류 자료형 변수이름;
기억 부류 자료형 변수이름 = 초기값;
```

```
auto int n;
register double yield;
static double data = 5.85;
int age;
extern int input;
extern int input = 4;
```

초기화는 오류 발생

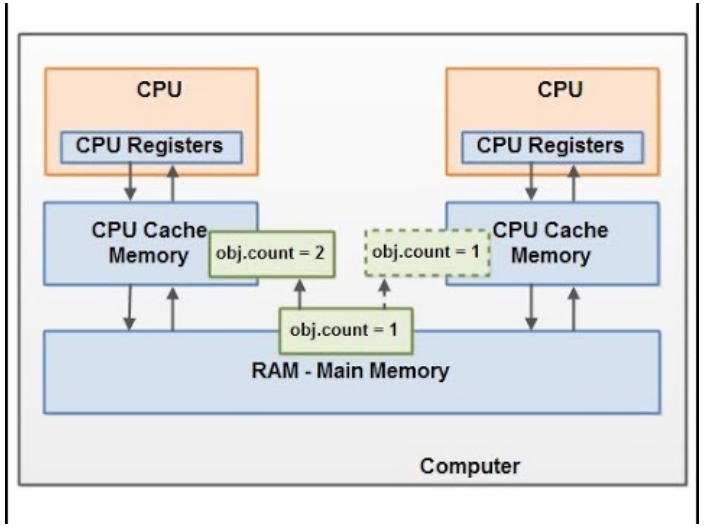
표 10-1 기억 부류 종류와 유효 범위

기억 부류 종류	전역	지역
auto	X	O
register	X	O
static	O	O
extern	O	X

레지스터(register) 변수

- **레지스터(register) 변수**
  - 변수의 저장 공간이 일반 메모리가 아니라 CPU 내부의 레지스터(register)에 할당되는 변수
  - 키워드 register를 자료형 앞에 넣어 선언: register int count;
    - ➢ 지역 변수에만 이용이 가능
  - 지역 변수로서 함수나 블록이 시작되면서 CPU의 내부 레지스터에 값이 저장
    - ➢ 함수나 블록을 빠져나오면서 소멸되는 특성
  - CPU 내부에 있는 기억 장소이므로 **일반 메모리보다 빠르게 참조 가능**
  - 일반 메모리에 할당되는 변수가 아니므로 주소연산자 &를 사용할 수 없음
    - ➢ 주소연산자 &를 사용하면 문법오류가 발생

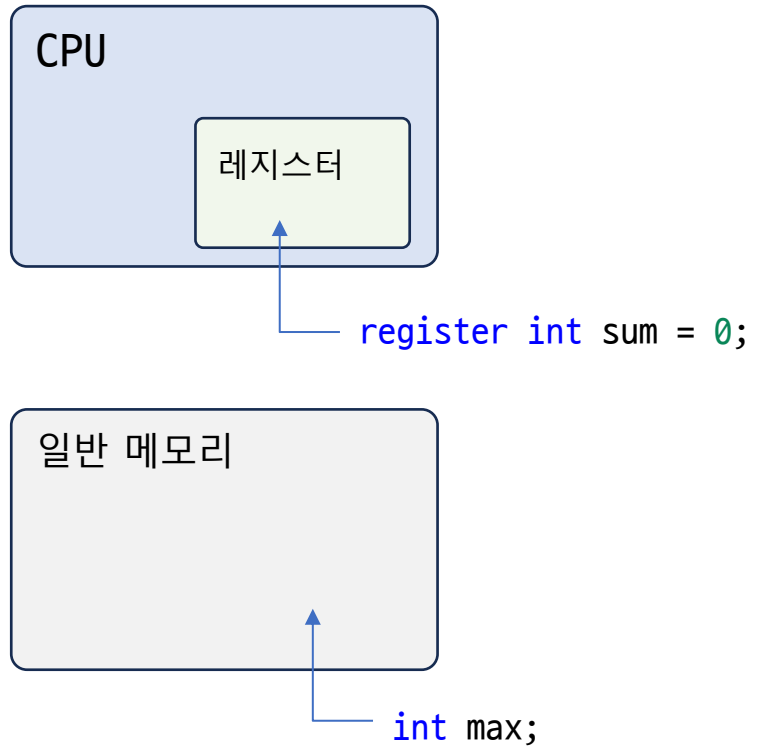
- **레지스터가 모자라면 일반 지역 변수로 할당**
  - 레지스터 변수는 처리 속도를 증가시키려는 변수에 이용
  - 특히 반복문의 횟수를 제어하는 제어 변수에 이용하면 효과적



레지스터 변수의 선언과 사용

<03regvar.c>

```c
#include <stdio.h>

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

    //메모리에 저장되는 일반 지역 변수 선언
    int max;
    printf("양의 정수 입력 >> ");
    scanf("%d", &max);

    //레지스터 블록 지역 변수 선언
    for (register int count = 1; count <= max; count++)
        sum += count;

    printf("합: %d\n", sum);

    return 0;
}
```



실행 결과

```
양의 정수 입력 >> 10
합: 55
```

정적 변수(static variable)

- **정적(static) 변수**
  - 변수 선언에서 자료형 앞에 키워드 static을 선언
- **정적 변수 종류**
  - 정적 지역 변수(static local variable)
  - 정적 전역 변수(static global variable)
- **정적 변수 특징**
  - 프로그램이 시작되면 메모리에 할당되고, 프로그램이 종료되면 메모리에서 제거
    - ➢ 지속적으로 저장 값을 유지하거나 수정 가능한 특성을 가짐
  - 초기값
    - ➢ 초기화는 단 한번만 수행
    - ➢ 초기값을 지정하지 않으면 자동으로 자료형에 따라 0이나 '\0' 또는 NULL값이 저장
      - – 한번 초기화된 정적 변수는 프로그램 실행 중간에 더 이상 초기화되지 않는 특성
      - – 주의할 점은 초기화는 상수로만 가능

```
static int svar = 1; //정적 변수
```

정적 변수 및 지역 변수 동작 비교

<04static_local.c>

```c
#include <stdio.h>

void increment(void);

int main(void)
{
    for (int count = 0; count < 3; count++)
        increment(); // 3번 함수호출
}

void increment(void)
{
    static int sindex = 1;
    int aindex = 1;

    printf("정적 지역 변수 sindex: %2d,\t", sindex++);
    printf("지역 변수 aindex: %2d\n", aindex++);
}
```

static int sindex: 정적 지역 변수
- increment() 함수가 호출될 때 1회만 초기화됨
- 전역 변수처럼 동작: 값이 계속 증가

int aindex: 지역 변수
- increment() 함수가 호출될 때마다 새롭게 메모리에 할당
- aindex의 값은 항상 1

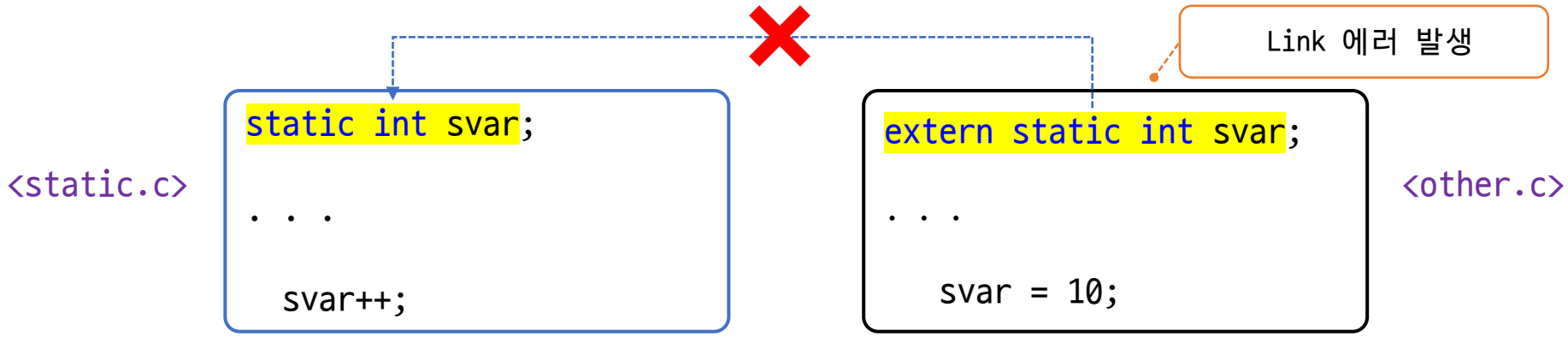
실행 결과

```
정적 지역 변수 sindex:  1,      지역 변수 aindex:  1
정적 지역 변수 sindex:  2,      지역 변수 aindex:  1
정적 지역 변수 sindex:  3,      지역 변수 aindex:  1
```

정적 지역 변수와 정적 전역 변수

- **정적 지역 변수**
  - 함수나 블록에서 static으로 선언되는 변수
  - 선언된 블록 내부에서만 참조 가능
  - 변수 유효 범위(scope)는 지역 변수와 같음
    - ➢ 함수나 블록을 종료해도 메모리에서 제거되지 않고 계속 메모리에 유지 관리
    - ➢ 할당된 저장 공간은 프로그램이 종료되어야 메모리에서 제거

- **정적 전역 변수**
  - 선언된 파일 내부에서만 참조가 가능
  - extern에 의해 다른 파일에서 참조가 불가능



정적 지역 변수와 지역 변수의 차이

<04slocal.c>

```c
#include <stdio.h>

void reset(void); //함수원형
void count(void); //함수원형

int main(void)
{
    //자동 지역 변수
    for (int i = 1; i <= 5; i++)
    {
        reset();
        count();
    }
}
```

```c
void reset(void) //매번 1을 지정
{
    int n = 1;  // 지역 변수
    printf("자동 지역 변수 n: %2d\n", n);
    n++;
}

void count(void) //이전 값을 계속 누적
{
    static int s = 10; //정적 지역 변수
    printf("\t정적 지역 변수 s: %2d\n", s);
    s++;
}
```

실행 결과

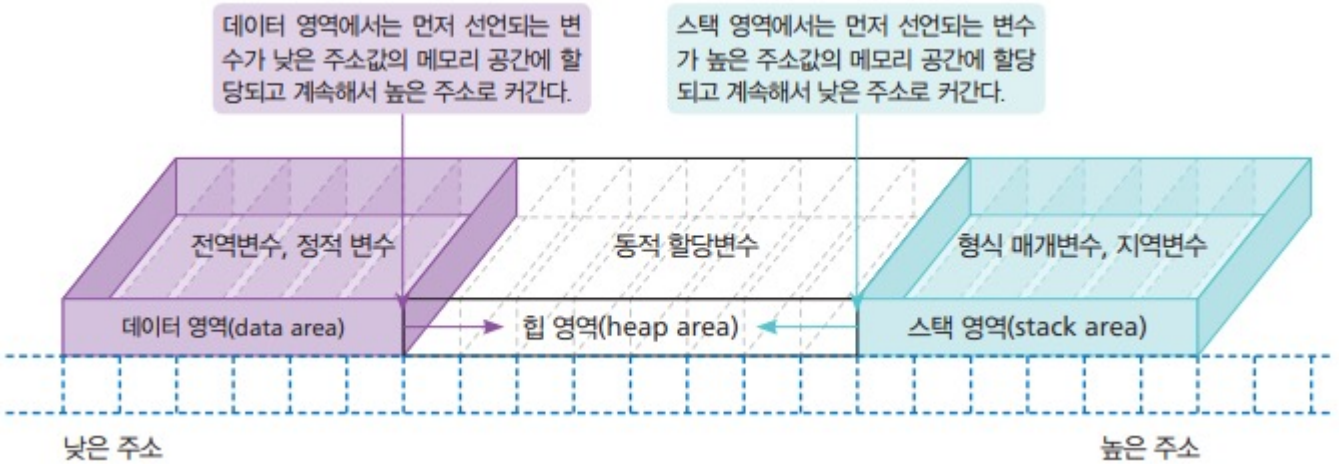
```
자동 지역 변수 n:  1
        정적 지역 변수 s: 10
자동 지역 변수 n:  1
        정적 지역 변수 s: 11
자동 지역 변수 n:  1
        정적 지역 변수 s: 12
자동 지역 변수 n:  1
        정적 지역 변수 s: 13
자동 지역 변수 n:  1
        정적 지역 변수 s: 14
```

데이터, 스택, 힙 영역

- **메인 메모리의 영역은 프로그램 실행 과정에서**
  - 데이터(data) 영역, 힙(heap) 영역, 스택(stack) 영역 세 부분으로 구분
- **기억 부류**
  - 변수의 유효 범위(scope)와 생존기간(lifetime)을 결정
  - 변수의 저장공간의 위치가 데이터(data) 영역, 힙(heap) 영역, 스택(stack) 영역인지 결정
  - 초기 값도 결정

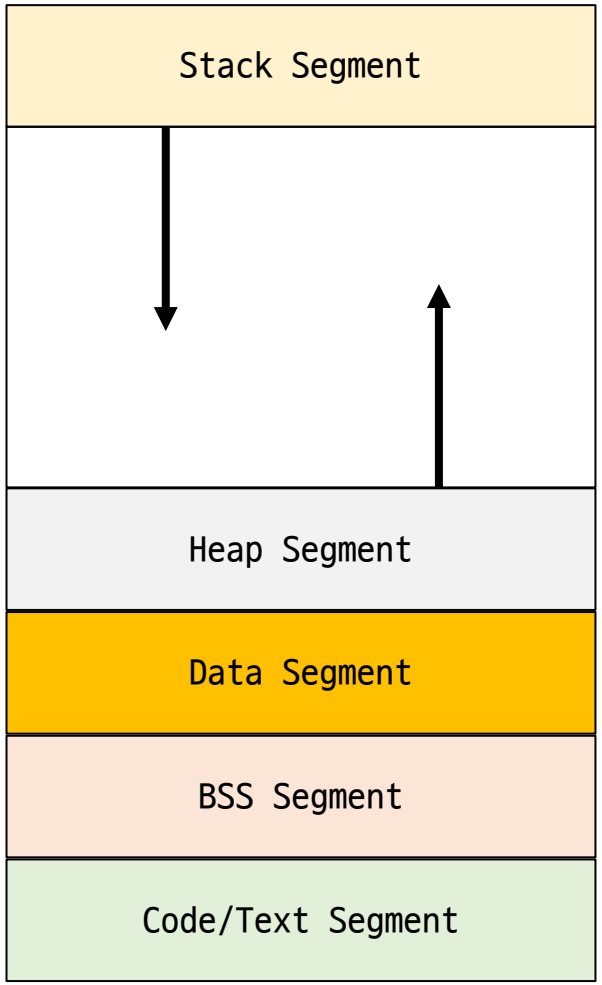
메모리 영역

- **데이터 영역**
  - 전역 변수와 정적변수가 할당되는 저장공간
- **힙 영역**
  - 동적 할당(dynamic allocation)되는 변수가 할당되는 저장공간
- **스택 영역**
  - 함수 호출에 의한 형식 매개변수 그리고 함수 내부의 지역 변수가 할당되는 저장공간
    - ➢ 함수 호출과 종료에 따라 메모리가 할당되었다가 다시 제거되는 작업이 반복



Memory 구조

- **스택 세그먼트 (Stack Segment)**
  - 지역 변수와 함수 파라미터, 리턴값 등
- **힙 세그먼트(Heap Segment)**
  - 동적 할당한 메모리 저장 (malloc)
  - 운영체제에 따라 스택 세그먼트와 힙 세그먼트의 위치는 달라짐
- **데이터 세그먼트 (Data Segment)**
  - 초기화된 전역 변수와 초기화된 정적 변수 저장
- **BSS (Block started symbol Segment)**
  - 초기화 되지 않은 전역 변수와 초기화 되지 않은 정적 변수
  - 프로그램 시작 시점에 0 으로 자동 초기화
- **코드/텍스트 세그먼트 (Code/Text Segment)**
  - 프로그램 기계어 명령 (코드만 저장됨)
  - 읽기만 가능하고 쓰기는 금지



메모리 저장 예제

<memory_usage.c>

```c
/**
* Memory 사용 예제
* - Stack, Heap, Data, BSS, Text Segment 사용
*/
#include <stdio.h>
#include <stdlib.h>

char init[] = "C Programming";
int num_init[100];

int main(int argc, char *argv[])
{
    float f;
    int value;

    char *p = "Hello World";

    char *cp = (char *)malloc(1000);
    printf("%s\n", p);

    free(cp);
    return 0;
}
```

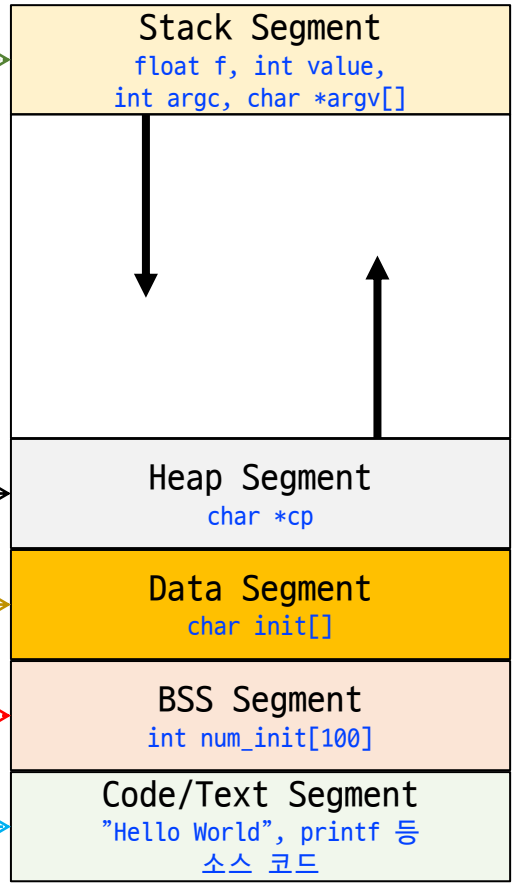
초기화된 전역 변수

초기화 되지 않은 전역 변수

지역 변수

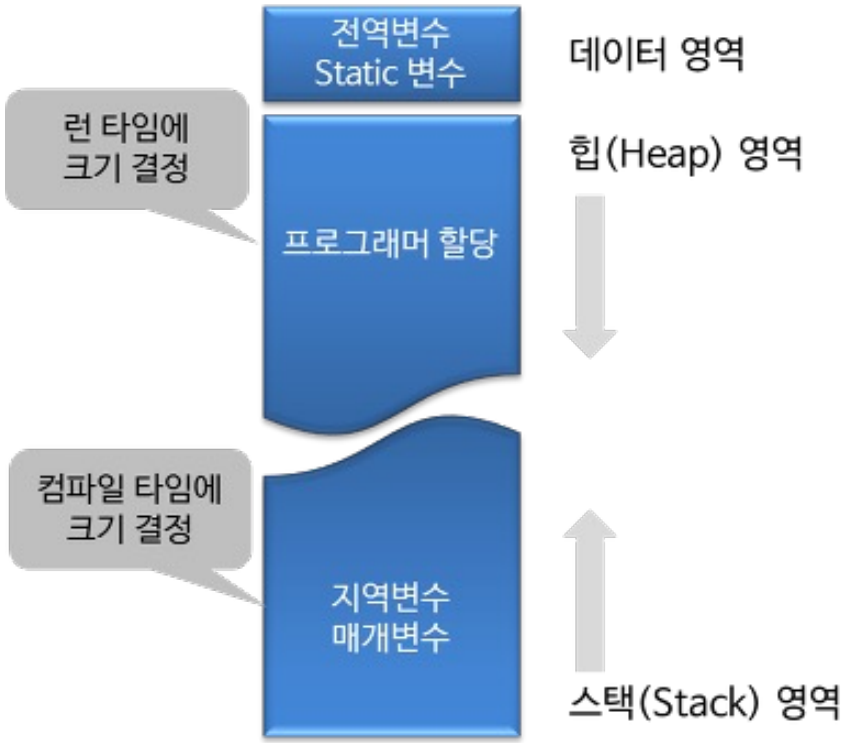
포인터 변수 초기화

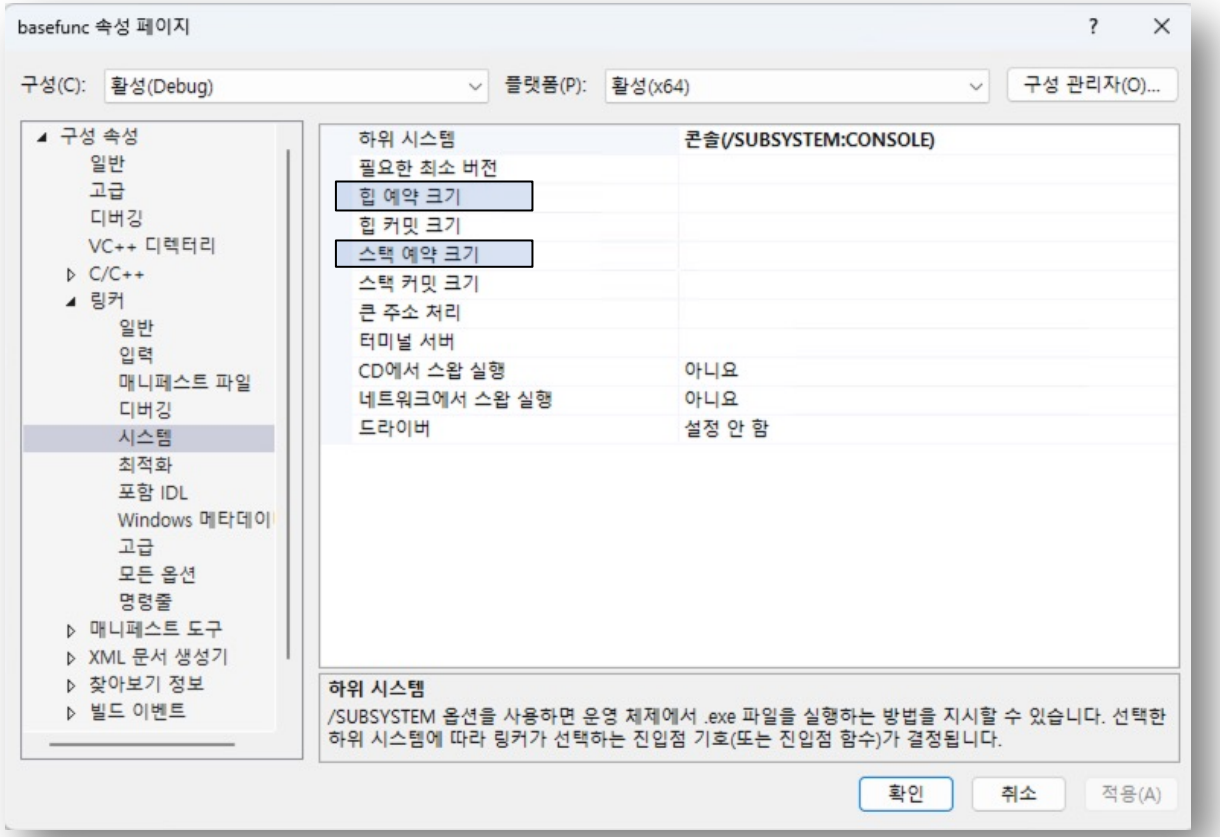
동적 할당



Visual Studio에서 Stack과 Heap 크기

- Stack과 Heap 크기 설정 가능
  - 힙 예약 크기: 기본 1MB
  - 스택 예약 크기: 기본 1MB





변수 활용 기준

- **변수 활용 기준**
  - 실행 속도를 개선하고자 하는 경우
    - ➢ 제한적으로 특수한 지역 변수인 **레지스터 변수**를 이용
  - 함수나 블록 내부에서 함수나 블록이 종료되더라도 계속적으로 값을 저장할 경우
    - ➢ **정적 지역 변수**를 이용
  - 해당 파일 내부에서만 변수를 공유하는 경우
    - ➢ **정적 전역 변수**를 이용
  - 프로그램의 모든 영역에서 값을 공유하는 경우
    - ➢ **전역 변수**를 이용
    - ➢ 가능하면 전역 변수의 사용을 줄이는 것이 프로그램의 이해를 높일 수 있음

선언 위치	상세 종류	키워드		유효 범위	기억 장소	생존 기간
전역	전역변수	참조선언	extern	프로그램 전역	메모리 (데이터 영역)	프로그램 실행 시간
	정적 전역변수	static		파일 내부		
지역	정적 지역변수	static		함수나 블록 내부		
	레지스터 변수	register			레지스터	함수 또는 블록 실행 시간
	자동 지역변수	auto(생략가능)			메모리 (스택 영역)	

변수의 종류에 따른 생존 기간과 유효 범위

- **전역 변수와 정적 변수**
  - 모두 생존 기간이 프로그램 시작 시에 생성되어 프로그램 종료 시에 제거

- **자동 지역 변수와 레지스터 변수**
  - 함수가 시작되는 시점에서 생성되어 함수가 종료되는 시점에서 제거

구분	종류	메모리 할당 시기	동일 파일 외부 함수에서의 이용	다른 파일 외부 함수에서의 이용	메모리 제거 시기
전역	전역변수	프로그램 시작	O	O	프로그램 종료
	정적 전역변수	프로그램 시작	O	X	프로그램 종료
지역	정적 지역변수	프로그램 시작	X	X	프로그램 종료
	레지스터 변수	함수(블록) 시작	X	X	함수(블록) 종료
	자동 지역변수	함수(블록) 시작	X	X	함수(블록) 종료

- **변수의 종류에 따른 초기 값**

구분	종류	자동 저장되는 기본 초기값	초기값 저장
전역	전역변수	자료형에 따라 0이나 '\0' 또는 NULL 값이 저장	프로그램 시작 시
	정적 전역변수		
지역	정적 지역변수		
	레지스터 변수	쓰레기 값이 저장	함수나 블록이 실행될 때마다
	자동 지역변수		

전역 변수와 지역 변수의 선언과 참조

<06-1sclass.c>

```c
#include <stdio.h>

void in(void);
void out(void);

int g = 10; // 전역 변수
static int sg = 20; // 정적 전역 변수

int main(void)
{
    int a = 100; /* main() 함수의 자동 지역 변수 */

    printf("%d %d %d\n", g, sg, a);
    in(); out();
    in(); out();
    in(); out();
    printf("%d %d %d\n", g, sg, a);

    return 0;
}
```

실행 결과

```
10 20 100
        11 21 2 1
                        12
        13 22 2 2
                        14
        15 23 2 3
                        16
16 23 100
```

```c
void in(void)
{
    int fa = 1;

    static int fs; // in() 함수의 정적 지역 변수

    printf("\t%d %d %d %d\n", ++g, ++sg, ++fa, ++fs);
}
```

<06-2out.c>

```c
#include <stdio.h>

extern int g, sg;

void out()
{
    printf("\t\t\t%d\n", ++g);

    //외부 파일에 선언된 정적 전역 변수이므로 실행 시 오류
    //printf("%d\n", ++sg);
}
```

컴파일

```
gcc 06-1sclass.c 06-2out.c -o 06-1sclass
```

Lab 은행계좌의 입출금 구현

- **전역 변수 total**
  - 전역 변수 total에는 초기 금액과 계좌 잔고가 저장

- **두 함수 save()와 withdraw()의 정적 지역 변수 amount를 사용**
  - 매개변수 금액의 입출금을 구현
  - 정적 지역 변수 amount를 사용하여 총입금액과 총출금액을 관리하여 출력

실행 결과

```
입금액   출금액    총입금액    총출금액      잔고
=============================================
                                      10000
  50000              50000            60000
         30000                30000    30000
  60000             110000            90000
         20000                50000    70000
=============================================
```

Lab 은행계좌의 입출금 구현

```c
#include <stdio.h>

int total = 10000; //전역 변수

void save(int); //입금 함수원형

void withdraw(int); //출금 함수원형

int main(void)
{
    printf(" 입금액 출금액 총입금액 총출금액 잔고\n");
    printf("=============================================\n");
    printf("%46d\n", total);
    save(50000);
    withdraw(30000);
    save(60000);
    withdraw(20000);
    printf("=============================================\n");

    return 0;
}
```

```c
//입금액을 매개변수로 사용
void save(int money)
{
    //총입금액이 저장되는 정적 지역 변수
    static int amount;
    total += money;
    amount += money;
    printf("%7d %17d %20d\n", money, amount, total);
}

//출금액을 매개변수로 사용
void withdraw(int money)
{
    //총 출금액이 저장되는 정적 지역 변수
    static int amount;
    total -= money;
    amount += money;
    printf("%15d %20d %9d\n", money, amount, total);
}
```

실행 결과

```
 입금액  출금액    총입금액    총출금액      잔고
=============================================
                                        10000
  50000                 50000             60000
         30000                    30000     30000
  60000                110000             90000
         20000                    50000     70000
=============================================
```

Questions?