13장 변수의 영역과 데이터 공유

한빛미디어 Hanbit Media, Inc.

13-1 변수 사용 영역

❖ 지역 변수 (1/2)

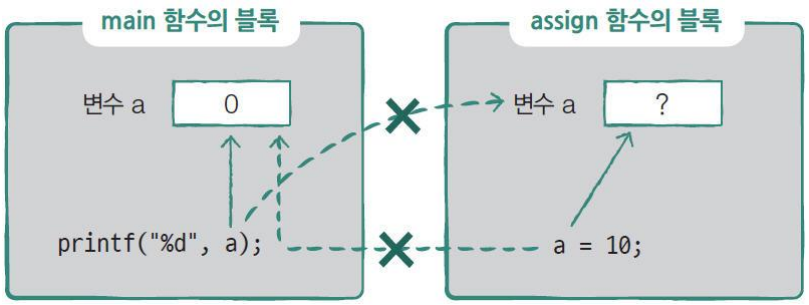
두 함수에서 같은 이름의 지역 변수를 사용한 경우 소스 코드 예제13-1.c

```c
#include <stdio.h>

void assign(void);

int main(void)
{
    auto int a = 0;

    assign();
    printf("main 함수 a : %d\n", a);

    return 0;
}
```

```c
void assign(void)
{
    int a;

    a = 10;
    printf("assign 함수 a : %d\n", a);
}
```

실행결과

```
assign 함수 a : 10
main 함수 a : 0
```



❖ 지역 변수 (2/2)

- 지역 변수는 선언된 함수(블록) 안에서만 사용할 수 있다.

  ➡ 디버깅에 유리하다.

- 선언된 블록이 끝나면 저장 공간이 메모리에서 사라진다.

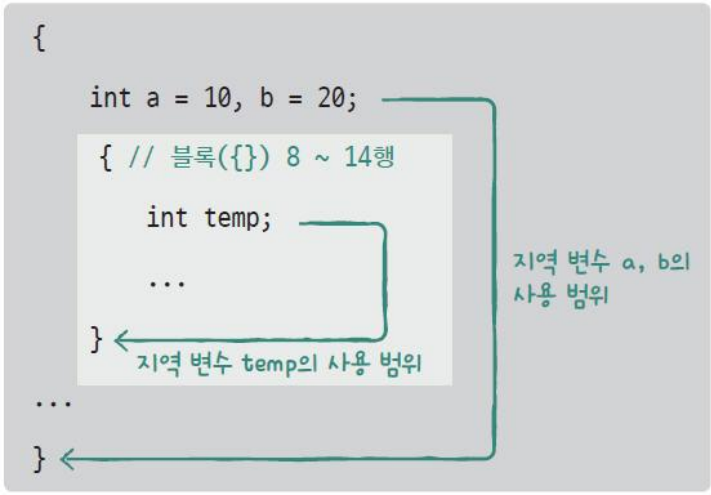
  ➡ 메모리를 효율적으로 사용한다.

13-1 변수 사용 영역

❖ 블록 안에서 사용하는 지역 변수 (1/2)

블록 안에 지역 변수를 사용하여 두 변수를 교환하는 프로그램 소스 코드 예제13-2.c

```
01  #include <stdio.h>
02
03  int main(void)
04  {
05      int a = 10, b = 20;
06
07      printf("교환 전 a와 b의 값 : %d, %d\n", a, b);
08      {                   // 블록 시작
09          int temp;       // temp 변수 선언
10
11          temp = a;
12          a = b;          // a와 b는 5행에 선언된 변수
13          b = temp;
14      }                   // 블록 끝
15      printf("교환 후 a와 b의 값 : %d, %d\n", a, b);
16
17      return 0;
18  }
```

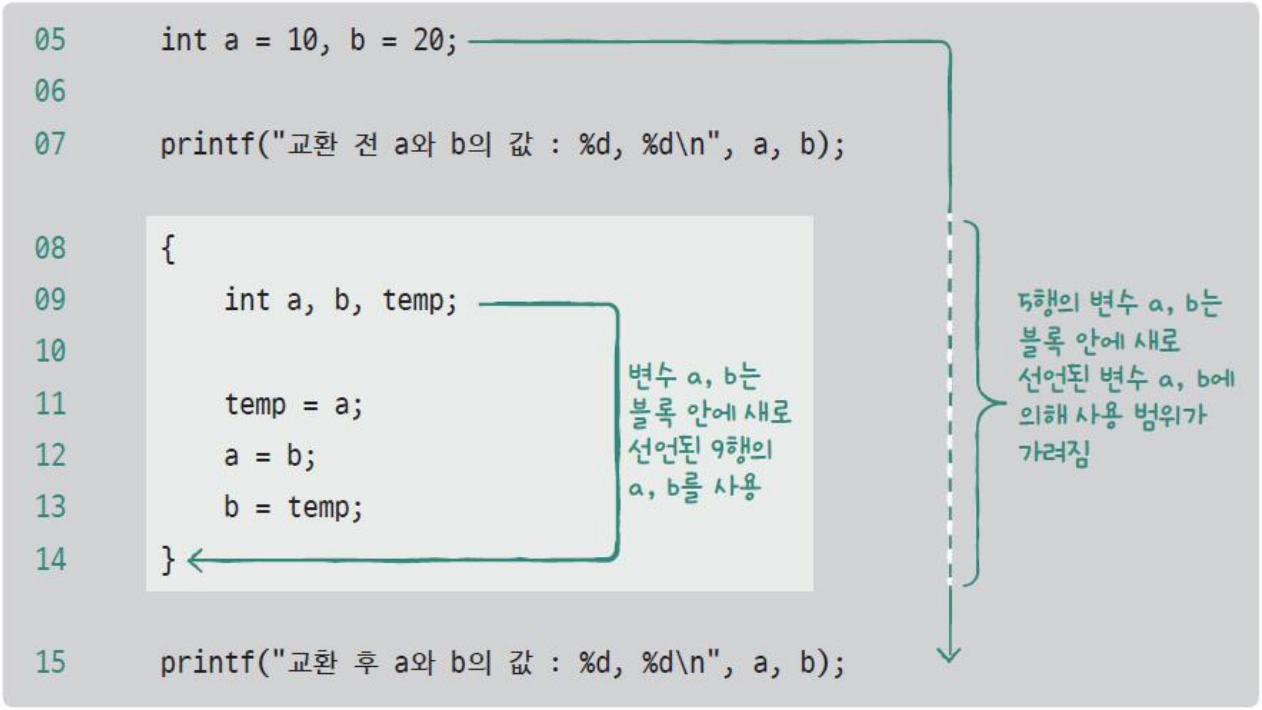


실행결과

```
교환 전 a와 b의 값 : 10, 20
교환 후 a와 b의 값 : 20, 10
```

❖ 블록 안에서 사용하는 지역 변수 (2/2)

- 같은 이름의 변수가 둘 이상이면 가장 가까운 변수를 사용한다.



13-1 변수 사용 영역

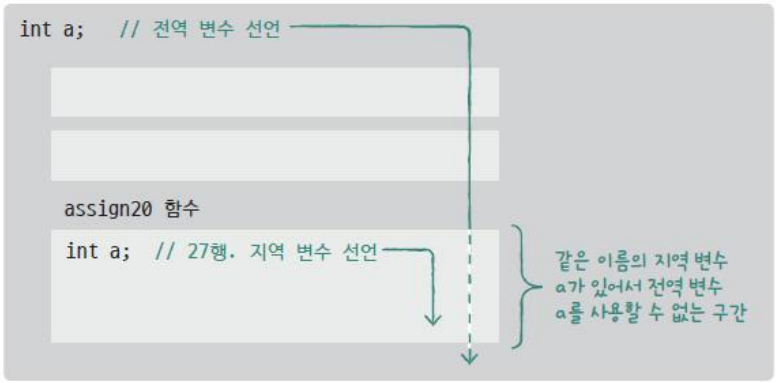
❖ 전역 변수 (1/2)

전역 변수의 사용 소스 코드 예제13-3.c

```c
#include <stdio.h>

void assign10(void);
void assign20(void);

int a;

int main(void)
{
    printf("함수 호출 전 a 값 : %d\n", a);

    assign10();
    assign20();

    printf("함수 호출 후 a 값 : %d\n", a);

    return 0;
}

void assign10(void)
{
    a = 10;
}

void assign20(void)
{
    int a;

    a = 20;
}
```

실행결과

```
함수 호출 전 a 값 : 0
함수 호출 후 a 값 : 10
```



❖ 전역 변수 (2/2)

- 이름이 바뀌면 사용 함수의 모든 이름을 찾아 바꿔야 한다.
- 값이 이상하면 접근 가능한 모든 함수를 살펴야 한다.
- 같은 이름의 지역 변수에 의해 사용 범위가 제한 된다.

13-1 변수 사용 영역

❖ 정적 지역 변수

auto 지역 변수와 static 지역 변수의 비교 소스 코드 예제13-4.c

```c
#include <stdio.h>

void auto_func(void);       // auto_func 함수 선언
void static_func(void);     // static_func 함수 선언

int main(void)
{
    int i;

    printf("일반 지역 변수(auto)를 사용한 함수...\n");
    for (i = 0; i < 3; i++)
    {
        auto_func();
    }

    printf("정적 지역 변수(static)를 사용한 함수...\n");
    for (i = 0; i < 3; i++)
    {
        static_func();
    }

    return 0;
}

void auto_func(void)
{
    auto int a = 0;

    a++;
    printf("%d\n", a);
}

void static_func(void)
{
    static int a;

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

실행결과

```
일반 지역 변수(auto)를 사용한 함수...
1
1
1
정적 지역 변수(static)를 사용한 함수...
1
2
3
```

함수 밖에 있다면?
저장 공간의 할당과 회수는 함수의 호출 및 반환과 관계가 없습니다!

13- 1 변수 사용 영역

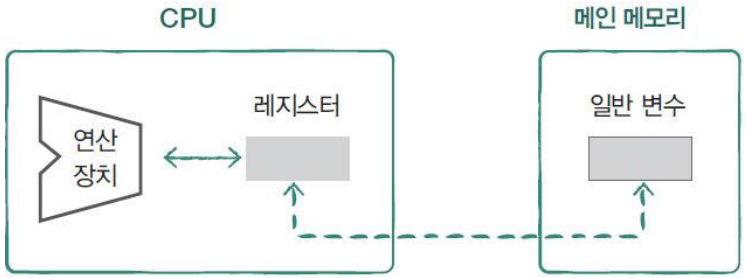
❖ 레지스터 변수 (1/2)

레지스터 변수를 반복문에 사용한 예 소스 코드 예제13-5.c

```
01  #include <stdio.h>
02
03  int main(void)
04  {
05      register int i;                 // 레지스터 변수
06      auto int sum = 0;               // auto 지역 변수
07
08      for (i = 1; i <= 10000; i++)    // 반복 과정에서 i를 계속 사용함
09      {
10          sum += i;                   // i 값을 반복하여 누적
11      }
12
13      printf("%d\n", sum);
14
15      return 0;
16  }
```

실행결과

```
50005000
```



❖ 레지스터 변수 (2/2)

- 전역 변수는 레지스터 변수로 선언할 수 없다.
- 레지스터 변수는 주소를 구할 수 없다.
- 레지스터의 사용 여부는 컴파일러가 결정한다.

마무리

키워드로 끝내는 핵심 포인트

- **지역 변수**의 사용 범위는 블록으로 제한된다.
- 지역 변수와 **전역 변수**의 사용 범위가 겹치면 지역 변수를 먼저 사용한다.
- 지역 변수에 static을 사용해서 **정적 지역 변수**로 만들면 프로그램의 시작부터 종료까지 저장 공간이 유지된다.
- **레지스터 변수**는 컴파일러가 레지스터에 생성할지 말지를 결정한다.

마무리

표로 정리하는 핵심 포인트

표 13-1 여러 가지 변수의 특징

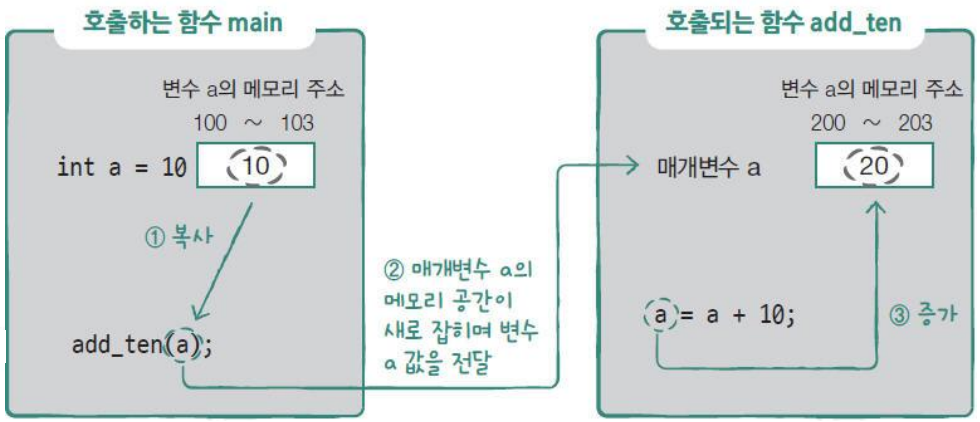
종류	지역 변수			전역 변수	
예약어	auto	register	static	없음	static
선언 위치	코드 블록 내부			함수 외부	
사용 범위	선언 ~ 선언한 블록 끝			프로그램 전체	하나의 파일 내부
메모리 존재 기간	선언 ~ 선언한 블록 끝			프로그램 시작 ~ 종료	
자동 초기화	없음			0으로 자동 초기화	
메모리 위치	스택 영역	레지스터		데이터 영역	

13- 2 함수의 데이터 공유 방법

❖ 값을 복사해서 전달하는 방법

10을 더하기 위해 값을 인수로 주는 경우 소스 코드 예제13-6.c

```
01  #include <stdio.h>
02
03  void add_ten(int a);   // 함수 선언
04
05  int main(void)
06  {
07      int a = 10;
08
09      add_ten(a); // a 값을 복사하여 전달
10      printf("a : %d\n", a);
11
12      return 0;
13  }
14
15  void add_ten(int a)          // 7행의 a와 다른 독립적인 저장 공간 할당
16  {
17      a = a + 10;              // 15행의 매개변수 a에 10을 더한다.
18  }
```



실행결과

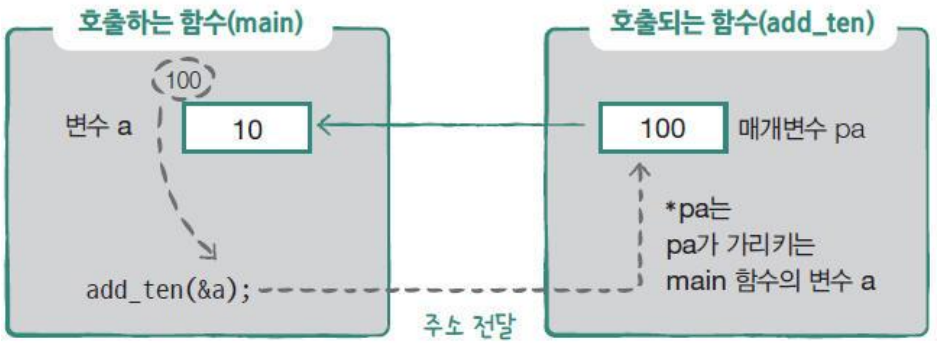
```
a : 10
```

13- 2 함수의 데이터 공유 방법

❖ 주소를 전달하는 방법

포인터를 써서 변수의 값에 10을 더하는 경우 소스 코드 예제13-7.c

```c
#include <stdio.h>

void add_ten(int *pa);
                // 매개변수로 포인터 pa 선언
int main(void)
{
    int a = 10;

    add_ten(&a);            // a의 주소를 인수로 준다.
    printf("a : %d\n", a); // 증가된 a 값 출력

    return 0;
}

void add_ten(int *pa)  // 포인터 pa가 a의 주소를 받는다.
{
    *pa = *pa + 10;    // 포인터 pa가 가리키는 변수의 값 10 증가
}
```



실행결과

```
a : 20
```

13- 2 함수의 데이터 공유 방법

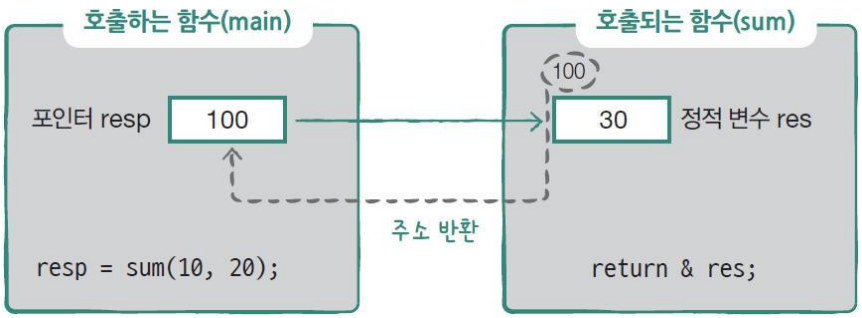
❖ 주소를 반환하는 함수

주소를 반환하여 두 정수의 합 계산 소스 코드 예제13-8.c

```
01  #include <stdio.h>
02
03  int *sum(int a, int b);
04
05  int main(void)
06  {
07      int *resp;
08
09      resp = sum(10, 20);
10      printf("두 정수의 합 : %d\n", *resp);
11
12      return 0;
13  }
14
```

실행결과

```
두 정수의 합 : 30
```




```
15  int *sum(int a, int b)
16  {
17      static int res;
18
19      res = a + b;
20
21      return &res;
22  }
```

마무리

키워드로 끝내는 핵심 포인트

- **값을 복사해서 전달**하면 호출하는 함수의 값은 바뀌지 않는다.
- 호출하는 함수의 값이 바뀌려면 **주소**를 인수로 **전달**해야 한다.
- 정적 지역 변수나 전역 변수와 같이 함수가 반환된 후에도 저장 공간이 유지되는 경우만 **주소를 반환**한다.

마무리

표로 정리하는 핵심 포인트

표 13-2 여러 가지 데이터 공유 방법

공유 방법	호출하는 함수	호출되는 함수
값을 넘겨준다.	`int a = 10;` `func(a);`	`void func(int b);`
값을 반환받는다.	`int a;` `a = func();`	`return b;`
주소를 넘겨준다.	`int a = 10;` `func(&a);`	`void func(int *p);`
주소를 반환받는다.	`int *p;` `p = func();`	`int b = 10;` `return &b;`