2010-1학기 프로그래밍입문(1)

chapter 06-5 참고자료

변수의 영역과 데이터의 전달

박 종 혁

Tel: 970-6702
Email: jhpark1@snut.ac.kr

출처: 뇌를 자극하는 C프로그래밍, 한빛미디어

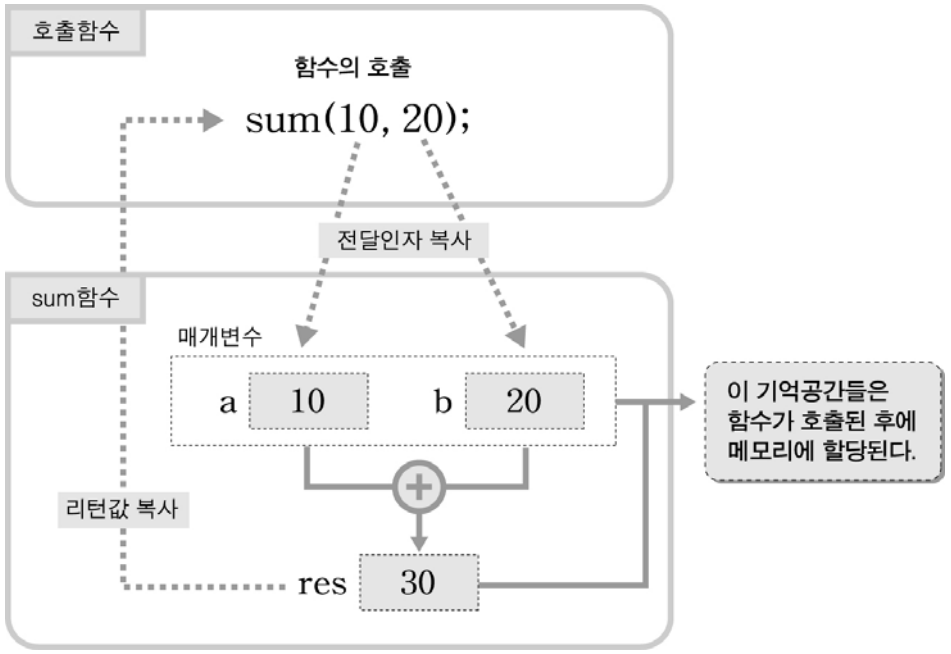
❑ 자동변수

- 지금까지 하나의 함수 안에서 선언한 변수는 자동변수이다.
- 사용범위는 하나의 함수 내부이다.
- 생존기간은 함수가 호출되어 실행되는 동안이다.
- 메모리에서의 위치는 스택영역이다.
- 자동초기화 되지 않으므로 쓰레기값이 존재한다.

▶ 자동변수의 생존기간

- 자동변수는 함수가 호출되어 변수 선언문이 실행될 때 메모리에 기억공간이 할당된다.
  - 세 개의 자동변수를 사용하는 함수의 예(매개변수도 자동변수이다).

```
int sum(int a, int b)
{
    int res;

    res=a+b;
    return res;
}
```

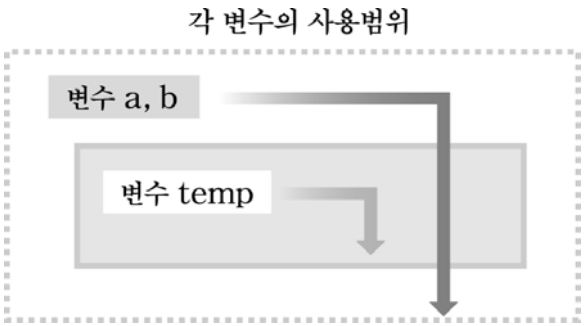


- 자동변수는 함수가 리턴되고 나면 더 이상 사용되지 않는 기억공간이므로 리턴될 때 메모리에서 사라진다.

▶ 기억공간의 할당과 회수

- 자동변수의 할당과 회수는 하나의 함수 내에서도 발생한다.
  - 함수 내에서 새로운 블록을 열고 변수를 선언하면 그 블록이 끝날 때 변수는 사라진다. 결국 변수의 사용범위는 블록 내부이다.

```
#include <stdio.h>

int main()
{
    int a=10, b=20;     // 메인함수 블록에 선언된 변수
    printf("바꾸기 전 a : %d, b: %d₩n", a, b);
    {                   // 새로운 블록의 시작
        int temp;       // 안쪽 블록에 선언된 변수
        temp=a;
        a=b;
        b=temp;
    }                   // 블록이 끝나면서 temp변수는 사라진다.
    printf("바꾼 후 a : %d, b : %d₩n", a, b);
    return 0;
}
```



▶ 중첩된 블록에서 같은 이름의 변수를 사용할 경우

- 중첩된 블록에서 같은 이름의 변수를 선언하면 가장 가까운 블록에 선언된 변수에 우선권이 있다.

```
#include <stdio.h>

int main()
{
    int val=10;
    {
        int val=20;
        {
            val++;
        }
        printf("val : %d₩n", val);
    }
    printf("val : %d₩n", val);
    return 0;
}
```

①

②

각 변수의 사용범위는 변수가 선언된 블록의 끝까지 이다.

가장 가까이 선언된 2번 변수를 사용한다.

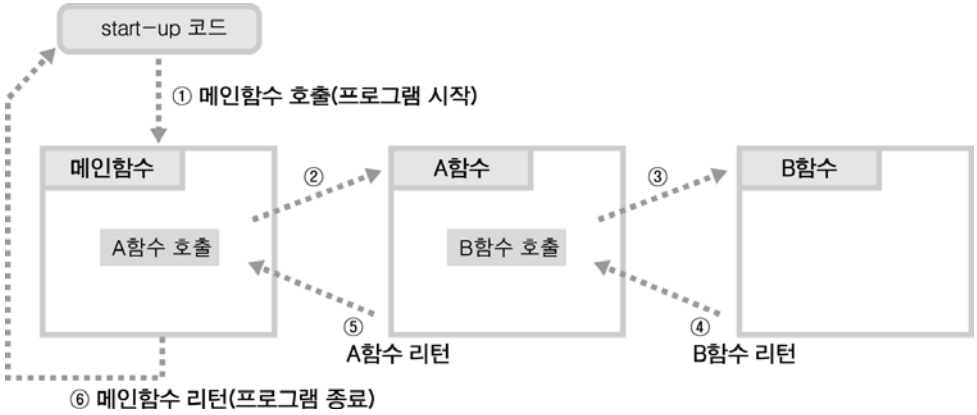
출력 결과

```
val : 21    // ② 번 출력
val : 10    // ① 번 출력
```

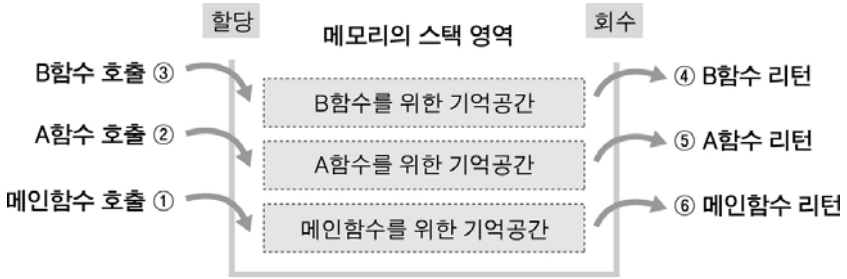
▶ 자동변수가 메모리에 할당되는 방법

- 자동변수는 함수의 호출 순서에 따라 할당되고 리턴될 때 회수된다. 따라서 가장 나중에 할당된 자동변수가 가장 먼저 사라진다.

  – 메인함수와 A, B 로 구성된 함수들의 호출과 리턴 예



- 자동변수는 함수의 호출과 리턴 규칙에 맞게 메모리의 스택(stack)영역에 할당된다.



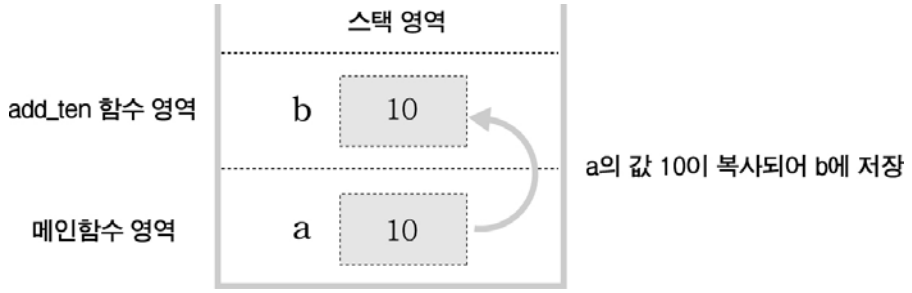
❑ 함수들 간의 데이터 전달 방법

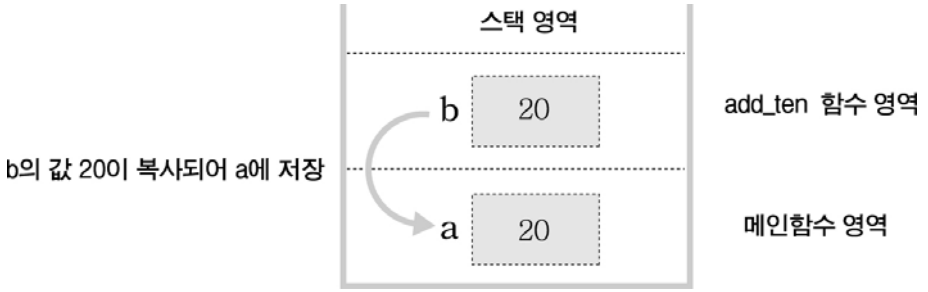
- 자동변수는 사용범위가 하나의 함수로 제한되기 때문에 함수들 간의 데이터 공유 방법이 필요하다.
- 값을 복사해서 넘겨주는 방법(call by value)
- 포인터를 사용하는 방법(call by pointer)

▶ 값을 복사해서 넘겨준다(call by value)

- 일반적인 함수의 호출 방법으로 호출함수의 전달인자가 피호출함수의 매개변수에 복사된다. 피호출함수는 리턴할 때 리턴값을 복사하여 호출함수로 전달한다.

```
#include <stdio.h>
int add_ten(int);
int main()
{
    int a=10;
    a=add_ten(a);
    printf("a : %d₩n", a);
    return 0;
}

int add_ten(int b)
{
    b=b+10;
    return b;
}
```

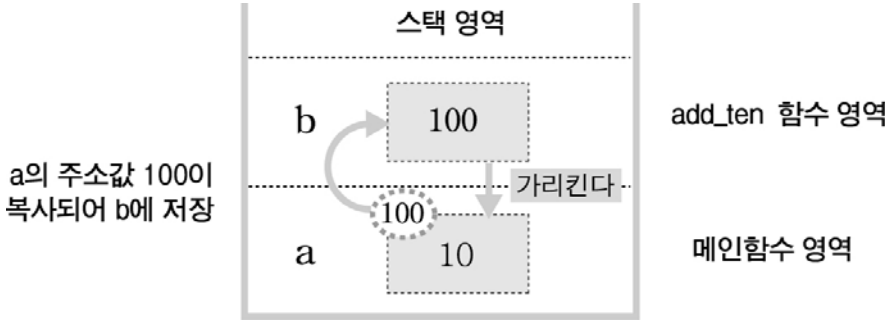


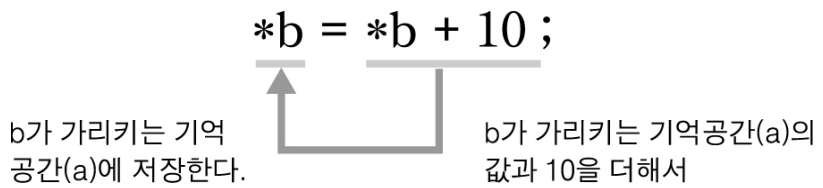


▶ 포인터를 사용한다(call by pointer)

- 호출함수에서 변수의 포인터를 전달인자로 주고 피호출함수에서는 이 포인터를 받아 호출함수의 변수를 참조하는 방식이다.

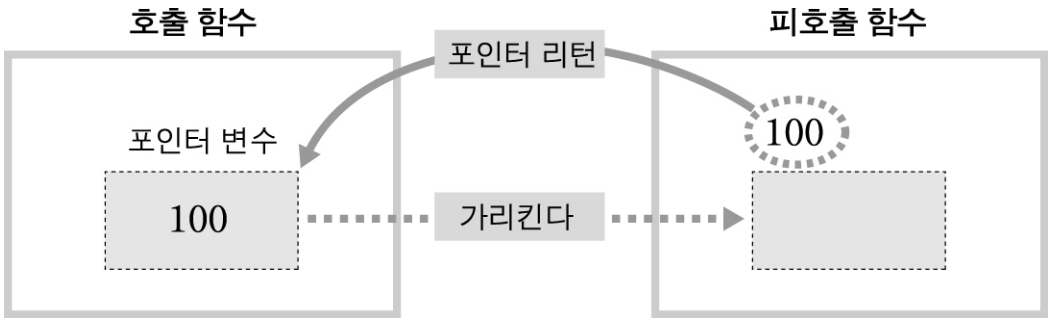
```
#include <stdio.h>
int add_ten(int *);
int main()
{
    int a=10;
    a=add_ten(&a);
    printf("a : %d₩n", a);
    return 0;
}

void add_ten(int *b)
{
    *b=*b+10;
}
```





▶ 포인터를 리턴하는 함수

- 피호출함수에서 포인터를 리턴하여 호출함수가 피호출함수의 기억공간을 참조할 수 있도록 할 수 있다.



- 포인터를 리턴하는 함수는 리턴값의 형태가 포인터형이 된다.
  - int형 변수의 포이터를 리턴하는 경우

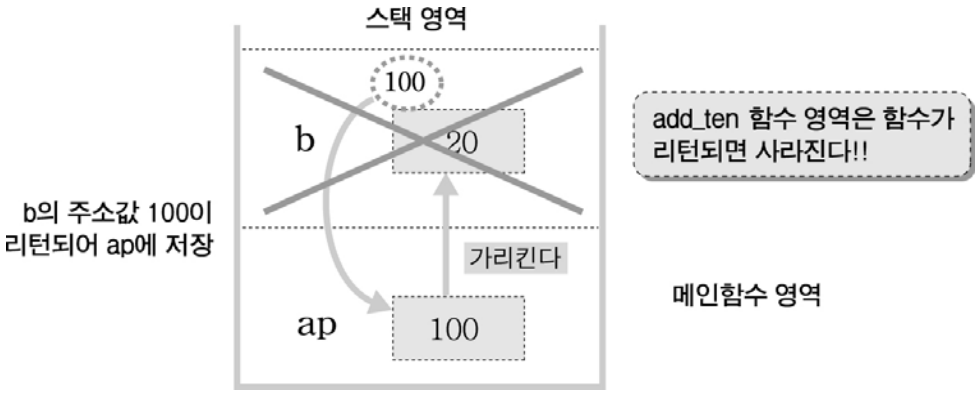
```
int * add_ten(int);
```

int형 기억공간의 포인터를 리턴한다!

▶ 포인터를 리턴하는 함수

- 자동변수의 포인터를 리턴하여 호출함수에서 다시 참조하는 것은 위험하다.

```
#include <stdio.h>
int *add_ten(int);
int main()
{
    int a=10;
    int *ap;
    ap=add_ten(a);
    printf("a : %d₩n", *ap);    // 포인터 변수 ap로 add_ten함수의 변수를 참조한다.
    return 0;
}

int *add_ten(int b)
{
    b=b+10;
    return &b;     // add_ten함수의 자동변수 b의 포인터를 리턴한다.
}
```



▶ 포인터를 리턴하는 함수

- 포인터를 리턴하는 경우는 함수가 리턴된 후에도 그 기억공간이 계속 유지되는 경우만 가능하다.
  - 호출함수로부터 포인터를 받아서 다시 리턴하는 경우(문자열 처리 함수들의 예)

```
char *strcpy(char *A, char *B);     // B의 문자열을 A에 복사하고 A를 리턴한다.
char *strcat(char *A, char *B);     // B의 문자열을 A에 붙인 후에 A를 리턴한다.
char *gets(char *A);                // A에 문자열을 입력하고 A를 리턴한다.
```

- 포인터를 리턴하면 좀더 다양한 방식으로 프로그램을 작성할 수 있다.
  - 두 문자열을 붙인 후에 그 결과를 바로 확인하는 예

```
#include <stdio.h>
#include <string.h>
int main()
{
    char src[80]="빈대";
    printf("결과 : %s₩n", strcat(src, "떡"));
    return 0;
}
```

❑ 정적변수(static variable)

- 정적변수는 함수가 리턴된 후에도 기억공간이 존재한다.
- 정적 변수는 자료형 앞에 static 예약어를 사용하여 선언한다.

```
static int sum;
```
→ 정적 변수의 선언

- 정적 변수는 기억공간의 할당과 초기화가 함수의 호출과 무관하다.

```
#include <stdio.h>
void increase();
int main()
{
    int i;
    for(i=0; i<5; i++){
        increase();
    }
    return 0;
}
```

```
void increase()
{
    static int sum=0;
    sum++;
    printf("sum : %d₩n", sum);
}
```

출력 결과

```
sum : 1
sum : 2
sum : 3
sum : 4
sum : 5
```

▶ 정적변수는 포인터를 리턴할 수 있다.

- 정적변수는 함수가 리턴된 후에도 그 기억공간이 유지되기 때문에 그 포인터를 리턴하여 호출하는 함수에서 참조할 수 있다.

```
#include <stdio.h>
int *increase();
int main()
{
    int i;
    int *sp;
    for(i=0; i<5; i++){
        sp=increase();
        printf("sum : %d₩n", *sp);
    }
    return 0;
}
```

```
int *increase()
{
    static int sum=0;
    sum++;
    return &sum;
}
```

출력 결과

```
sum : 1
sum : 2
sum : 3
sum : 4
sum : 5
```

❑ 외부변수(extern variable)

- 외부변수는 하나의 함수에 속해 있지 않으므로 여러 함수에서 자유롭게 사용할 수 있다.
- 외부 변수는 변수를 함수 밖에 선언한다.
- 외부변수를 사용하여 두 변수의 값을 바꾸는 예

```
#include <stdio.h>
void exchange();
int a, b;        // 변수를 함수 밖에 선언한다.
int main()
{
    printf("정수값 두 개를 입력하세요 : ");
    scanf("%d%d", &a, &b);
    exchange();      // 전달인자 없이 호출한다.
    printf("a : %d, b : %d₩n", a, b);
    return 0;
}
```

```
void exchange()
{
    int temp;
    temp=a;     // 외부변수 a, b를
    a=b;        // 자신의 변수인 것처럼
    b=temp;     // 사용한다.
}
```

- 외부변수도 기억공간의 생존기간이 함수의 호출여부와 무관하다.

▶ 외부변수를 사용할 때 주의할 점

- 외부변수는 여러 함수가 공유하므로 데이터의 안전성을 보장 받을 수 없다. 따라서 외부변수의 값에 변화가 생긴다면 다른 함수에 미치는 영향도 살펴야 한다.
- 외부변수와 같은 이름의 변수를 함수 내에서 선언하는 경우는 함수 내에서 선언된 변수를 우선적으로 참조한다.

exchange함수 내에 a, b가 선언되어 있다면 내부에 선언된 변수의 값을 바꾼다.