

컴퓨터 프로그래밍2

포인터

장서윤 pineai@cnu.ac.kr

포인터

- ▶ 변수의 메모리 주소 저장하는 변수

- ▶ 변수

- ▶ 메모리 상의 저장 공간
 - ▶ 위치를 알면 사용할 수 있음

- ▶ 위치값 - 주소

- ▶ 주소 사용하기 위한 조건
 - ▶ 주소 구하는 주소 연산자
 - ▶ 주소를 담는 포인터
 - ▶ 포인터로 변수 사용시 필요한 간접참조 연산자 기능 습득

포인터와 연산자

.....

표 9-1 포인터와 연산자

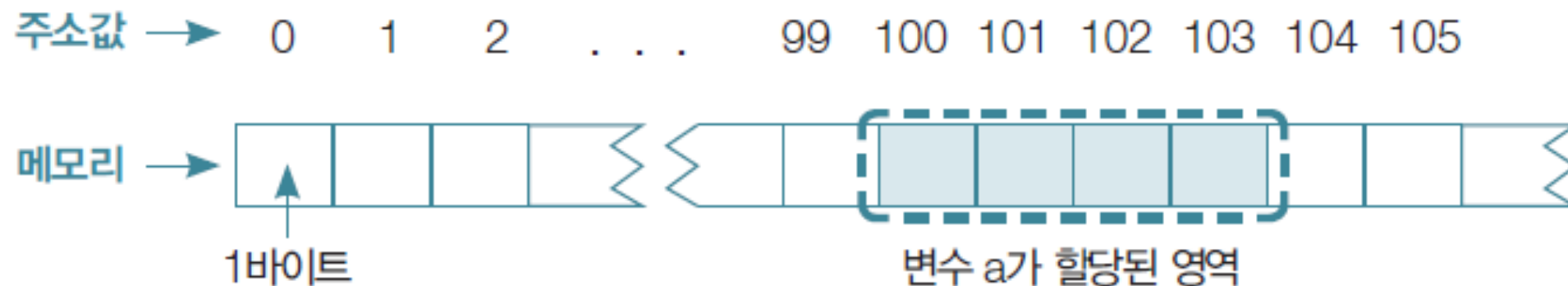
구분	사용 예	기능
주소 연산자	<code>int a;</code> <code>&a;</code>	변수 앞에 붙여 사용하며 변수가 할당된 메모리의 시작 주소값을 구한다.
포인터	<code>char *pc;</code> <code>int *pi;</code> <code>double *pd;</code>	시작 주소값을 저장하는 변수며 가리키는 자료형을 표시하여 선언한다.
간접참조 연산자	<code>*pi = 10;</code>	포인터에 사용하며 포인터가 가리키는 변수를 사용한다.

주소 연산자(&)

- ▶ 프로그램이 사용하는 메모리에는 바이트 별로 주소값 존재
 - ▶ 0부터 시작하고 바이트 단위로 1씩 증가
 - ▶ 2바이트 이상의 크기를 갖는 변수는 여러 개의 주소값에 걸쳐 할당



- ▶ ex) int형 변수 a가 메모리 100번지부터 할당
 - ▶ 100번지부터 103번지까지 4바이트에 걸쳐 할당된다는 의미



주소 연산자(&)

.....

예제 9-1 변수의 메모리 주소 확인

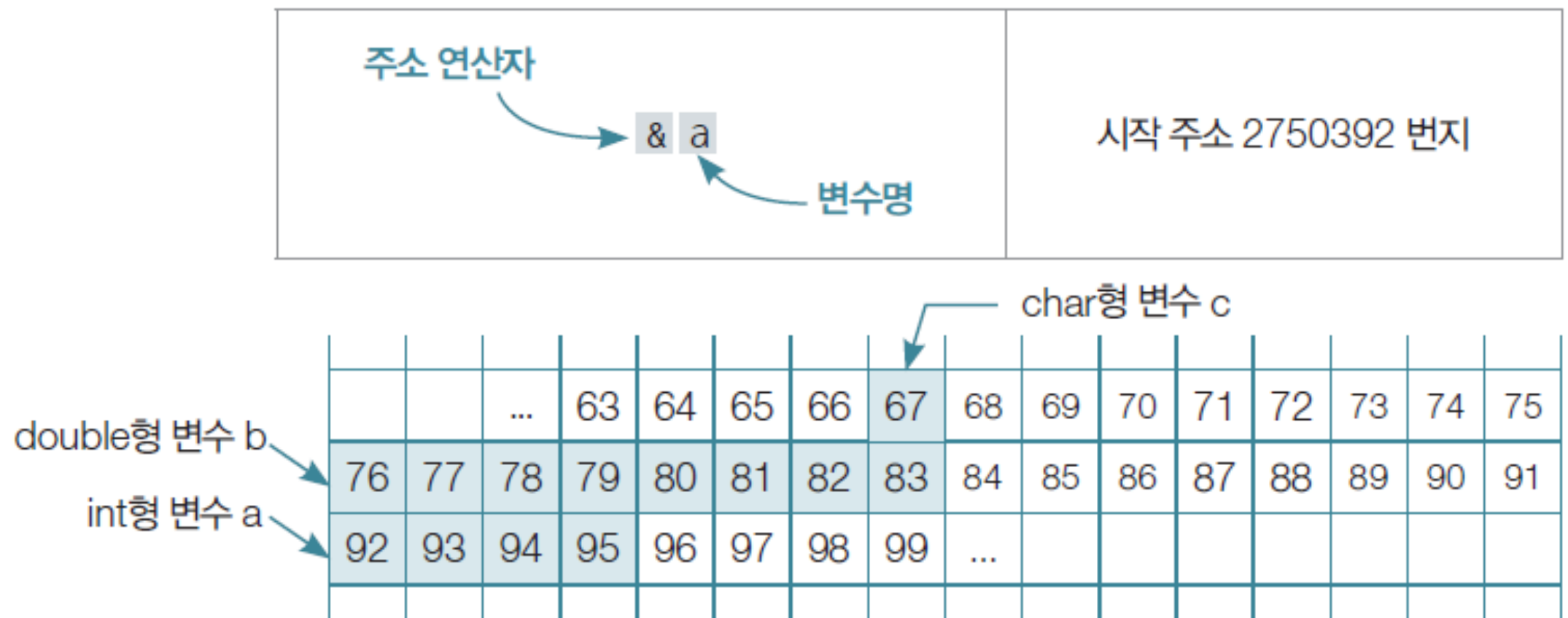
```
1. #include <stdio.h>
2.
3. int main(void)
4. {
5.     int a;           // int형 변수 선언
6.     double b;        // double형 변수 선언
7.     char c;          // char형 변수 선언
8.
9.     printf("int형 변수의 주소 : %u\n", &a); // 주소 연산자로 주소 계산
10.    printf("double형 변수의 주소 : %u\n", &b);
11.    printf("char형 변수의 주소 : %u\n", &c);
12.
13.    return 0;
14. }
```

실행 결과

int형 변수의 주소 : 2750392
double형 변수의 주소 : 2750376
char형 변수의 주소 : 2750367

주소 연산자(&)

- ▶ 변수 선언 부분
 - ▶ 변수 선언문 실행되면 각 자료형 크기만큼 메모리에 저장 공간 할당
 - ▶ 변수가 메모리 어디에 할당되었는지 궁금하다면 주소 연산자 & 사용



주소 연산자(&)

▶ 주소 출력

▶ 주소는 0부터 시작하는 양수

▶ 출력할 때 %u 변환문자 사용

▶ 또는 %x나 %p를 사용해 16진수로 출력하는 방법

▶ 주소를 계속 사용한다면?

▶ 사용할 때마다 주소 연산 수행하는 것보다 한 번 구한 주소 를 포인터에 저장

▶ 주소 저장할 포인터 선언이 우선

포인터와 간접참조 연산자(*)

예제 9-2 포인터의 선언과 사용

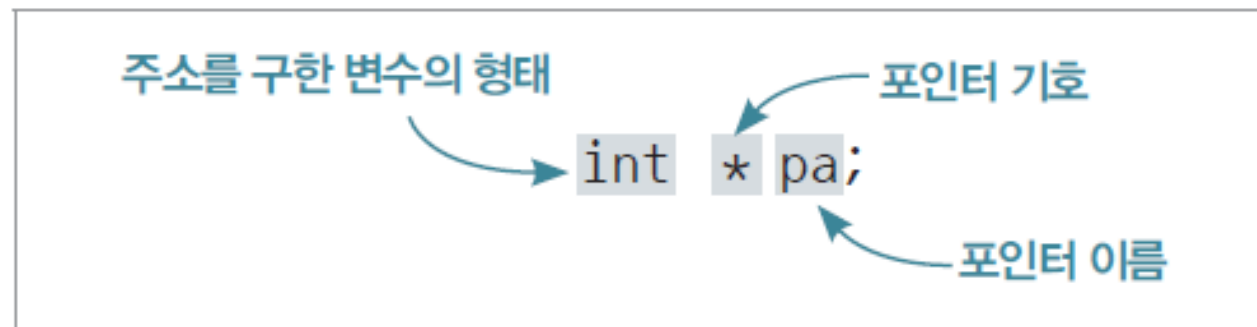
```
1. #include <stdio.h>
2.
3. int main(void)
4. {
5.     int a;                // 일반 변수 선언
6.     int *pa;              // 포인터 선언
7.
8.     pa = &a;              // 포인터에 a의 주소 대입
9.     *pa = 10;             // 포인터로 변수 a에 10 대입
10.
11.    printf("포인터로 a값 출력 : %d\n", *pa);
12.    printf("변수명으로 a값 출력 : %d\n", a); // 변수 a값 출력
13.
14.    return 0;
15. }
```

실행
결과
포인터로 a값 출력 : 10
변수명으로 a값 출력 : 10

포인터와 간접참조 연산자(*) 주소값을 통해서 사용하겠다.

➤ 포인터 선언

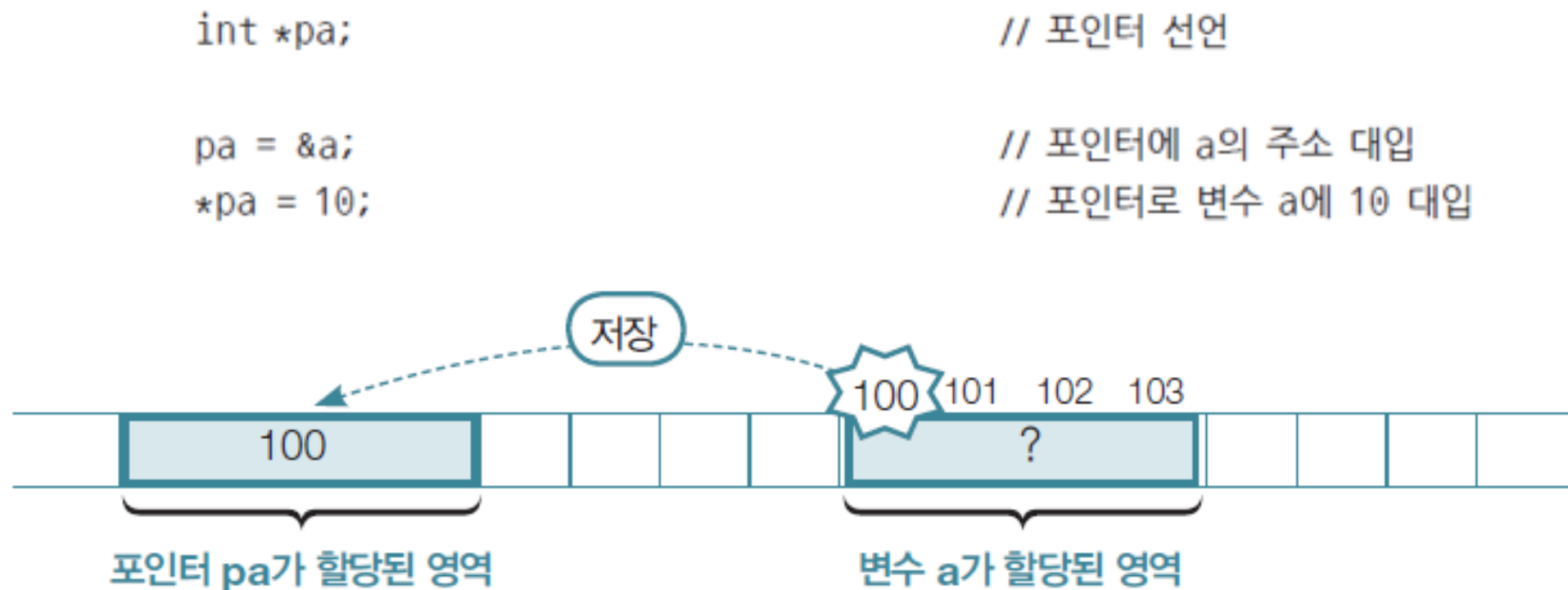
- int형 변수의 주소 저장하면 int 사용
- double형 변수의 주소 저장하면 double 사용



```
int a; int b;  
a = 10;  
* &a = 10;  
b = * &a + 20;  
printf("%d", *&a);
```

포인터와 간접참조 연산자(*)

- ▶ 포인터에 a의 시작 주소 저장하는 문장
- ▶ ex) 변수 a가 메모리 100번지부터 103번지까지 할당되었다면?
 - ▶ 주소 100번지가 pa에 저장



포인터가 주소를 저장하면 '가리킨다'고 말하고 화살표를 그려 간단히 표현합니다.

pa → a 포인터 pa는 변수 a를 가리킨다!

포인터와 간접참조 연산자(*)

- ▶ 포인터에 간접참조 연산자를 사용한 결과 = 가리키는 변수

`*pa == a` 같다

- ▶ 간접참조 연산자 사용한 예

표 9-2 간접참조 연산자를 사용한 예

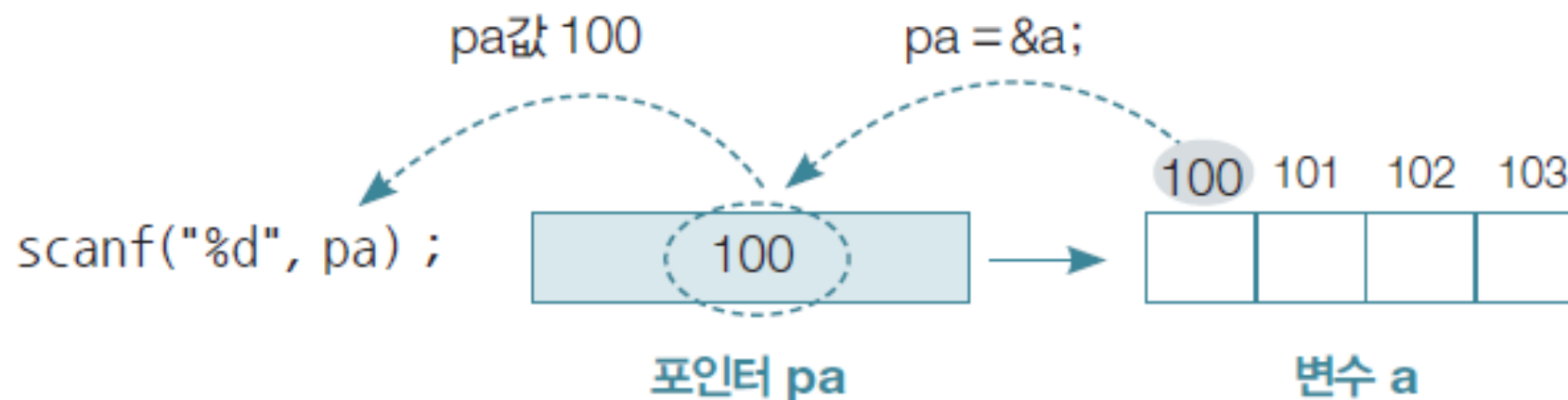
구분	변수 a 사용	포인터 pa 사용
대입 연산자 왼쪽	<code>a = 10;</code>	<code>*pa = 10;</code>
대입 연산자 오른쪽	<code>b = a;</code>	<code>b = *pa;</code>
피연산자	<code>a + 20;</code>	<code>*pa + 20;</code>
출력	<code>printf("%d", a);</code>	<code>printf("%d", *pa);</code>
입력	<code>scanf("%d", &a);</code>	<code>scanf("%d", &*pa);</code> <code>scanf("%d", pa);</code>

포인터와 간접참조 연산자(*)

- ▶ 포인터 `pa` 통해 변수 `a`에 입력할 때

<code>*pa == a</code>	① <code>pa</code> 가 가리키는 변수 <code>a</code> 를 구함
<code>&*pa == &a</code>	② 다시 변수 <code>a</code> 의 주소를 구함

```
scanf("%d", pa) ; // 포인터 pa값은 &a
```



여러 가지 포인터

예제 9-3 포인터를 사용한 두 정수의 합과 평균 계산

```
1. #include <stdio.h>
2.
3. int main(void)
4. {
5.     int a = 10, b = 15, tot;           // 변수 선언과 초기화
6.     double avg;                       // 평균을 저장할 변수
7.     int *pa, *pb;                     // 포인터 동시 선언
8.     int *pt = &tot;                   // 포인터 선언과 초기화
9.     double *pg = &avg;                // double형 포인터 선언과 초기화
10.
11.    pa = &a;                           // 포인터 pa에 변수 a의 주소 대입
12.    pb = &b;                           // 포인터 pb에 변수 b의 주소 대입
13.
14.    *pt = *pa + *pb;                   // a값과 b값을 더해 tot에 저장
15.    *pg = *pt / 2.0;                   // tot값을 2로 나눈 값을 avg에 저장
16.
17.    printf("두 정수의 값 : %d, %d\n", *pa, *pb); // a값과 b값 출력
18.    printf("두 정수의 합 : %d\n", *pt);          // tot값 출력
19.    printf("두 정수의 평균 : %.1lf\n", *pg);     // avg값 출력
20.
21.    return 0;
22. }
```

실행
결과

두 정수의 값 : 10, 15
두 정수의 합 : 25
두 정수의 평균 : 12.5



여러 가지 포인터

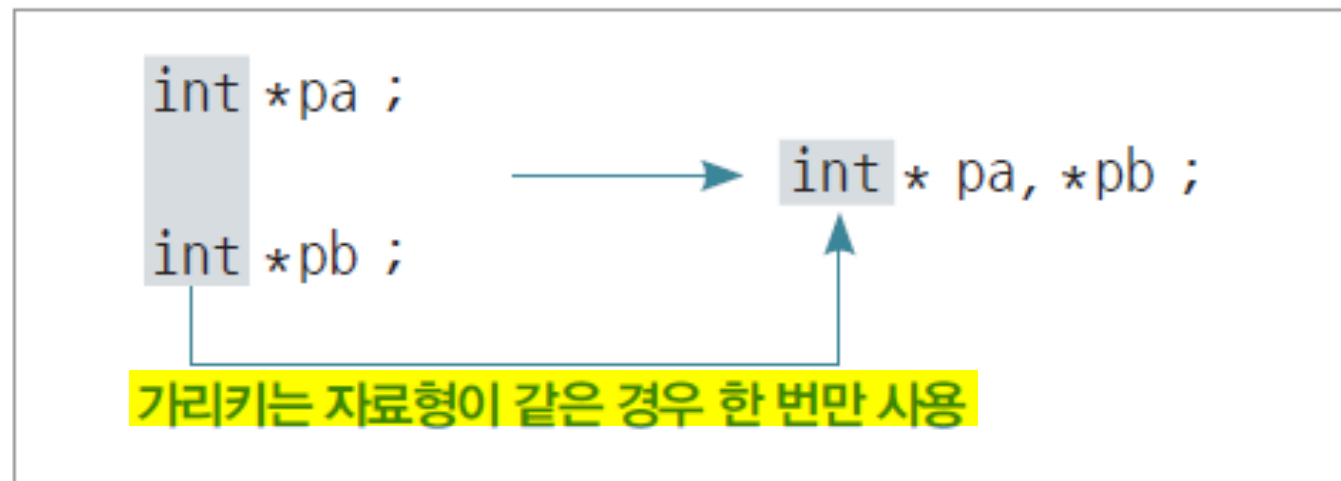
- ▶ 2개의 포인터 선언

- ▶ 가리키는 변수의 형이 같은 경우 포인터 연속 선언 가능

- ▶ ex) pa와 pb가 모두 int형 변수의 주소 저장하는 포인터라면?

- ▶ 콤마 사용하여 한 번에 선언 가능

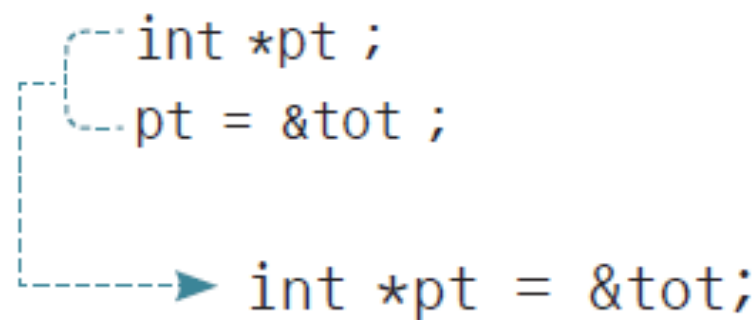
- ▶ 각 변수가 포인터임을 뜻하는 기호(*)는 변수마다 붙여야 함



```
int *pa, *pb, pc;    // pa, pb는 int형을 가리키는 포인터, pc는 int형 변수
```

여러 가지 포인터

- ▶ 포인터의 선언과 동시에 초기화하는 것도 가능
- ▶ 주소를 구할 int형 변수 tot 먼저 선언해야 함

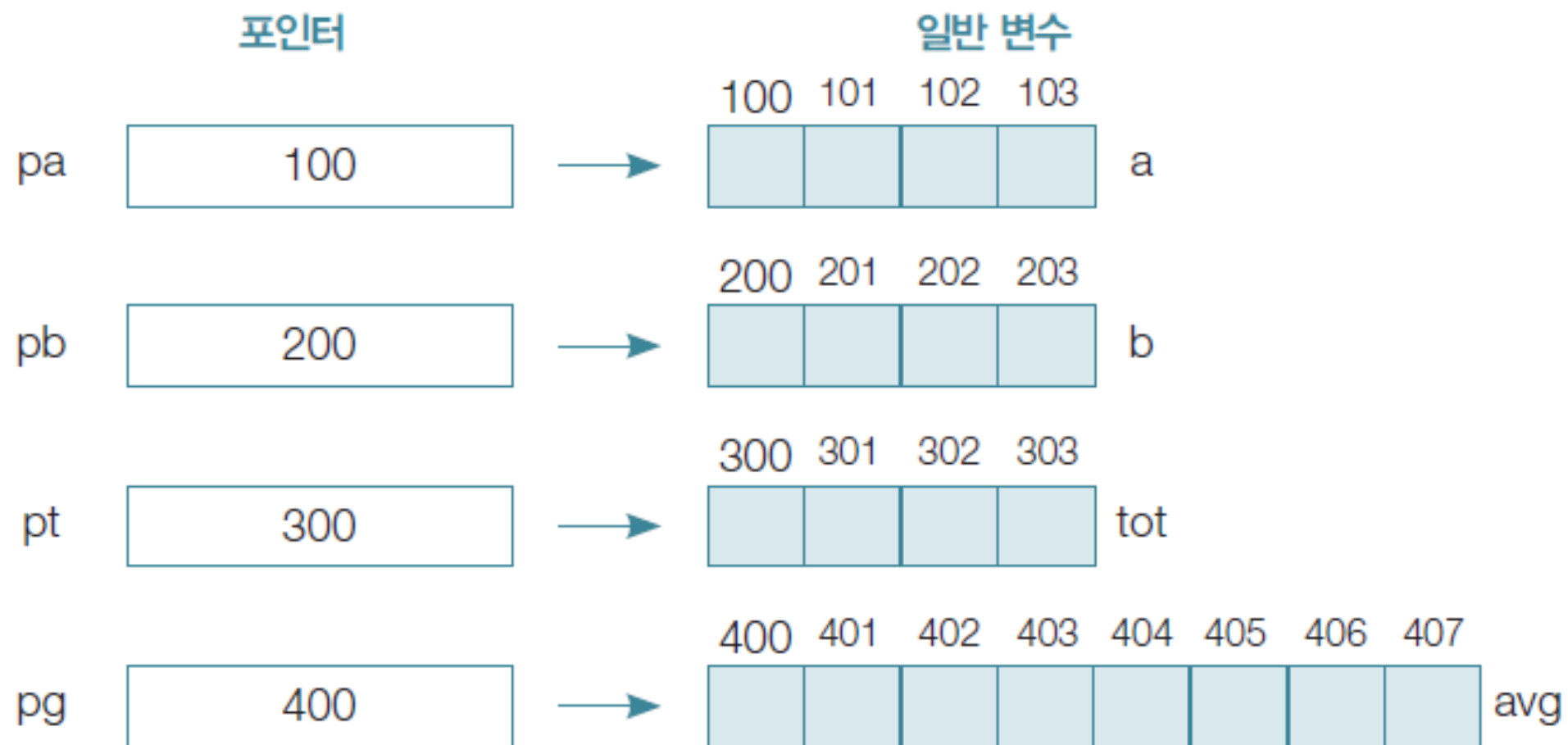


```
int *pt ;           // 포인터 선언
pt = &tot ;         // 주소를 구해 저장

int *pt = &tot;     // 포인터 선언과 동시에 주소로 초기화
```

여러 가지 포인터

- ▶ Ex) 변수 a, b, tot, avg의 메모리 시작 주소값이 각각 100, 200, 300, 400번지라면 다음과 같이 그림으로 표현



CONST를 사용한 포인터

- 포인터에 **const** 사용
 - 가리키는 변수의 값 바꿀 수 없음

예제 9-4 포인터에 const 사용

```
1. #include <stdio.h>
2.
3. int main(void)
4. {
5.     int a = 10, b = 20;
6.     const int *pa = &a;           // 포인터 pa는 변수 a를 가리킨다.
7.
8.     printf("변수 a값 : %d\n", *pa); // 포인터를 간접참조하여 a 출력
9.     pa = &b;                       // 포인터가 변수 b를 가리키게 한다.
10.    printf("변수 b값 : %d\n", *pa); // 포인터를 간접참조하여 b값 출력
11.    a = 20;                         // a를 직접 참조하여 값을 바꾼다.
12.    printf("변수 a값 : %d\n", *pa); // 포인터로 간접참조하여 바뀐 값 출력
13.
14.    return 0;
15. }
```

실행
결과

```
변수 a값 : 10
변수 b값 : 20
변수 a값 : 20
```

포인터에 관한 궁금한 이야기

❖ 포인터는 변수

- ▶ 포인터는 주소를 저장하는 메모리 공간
 - ▶ 다른 주소 저장하거나 포인터끼리 대입 가능
 - ▶ 일반 변수와 다른 특징도 있으므로 대입 연산 할 때 주의해야 함

표 9-3 주소와 포인터의 특징

구분	사용 예	기능
포인터	<pre>int a, b; int *p = &a; p = &b;</pre>	포인터는 변수이므로 그 값을 다른 주소로 바꿀 수 있다.
포인터의 크기	<pre>int *p; sizeof(p)</pre>	포인터의 크기는 컴파일러에 따라 다를 수 있으며 sizeof 연산자로 확인한다.
포인터의 대입 규칙	<pre>int *p; double *pd; pd = p; → (X)</pre>	포인터는 가리키는 자료형이 일치할 때만 대입한다.

포인터에 관한 궁금한 이야기

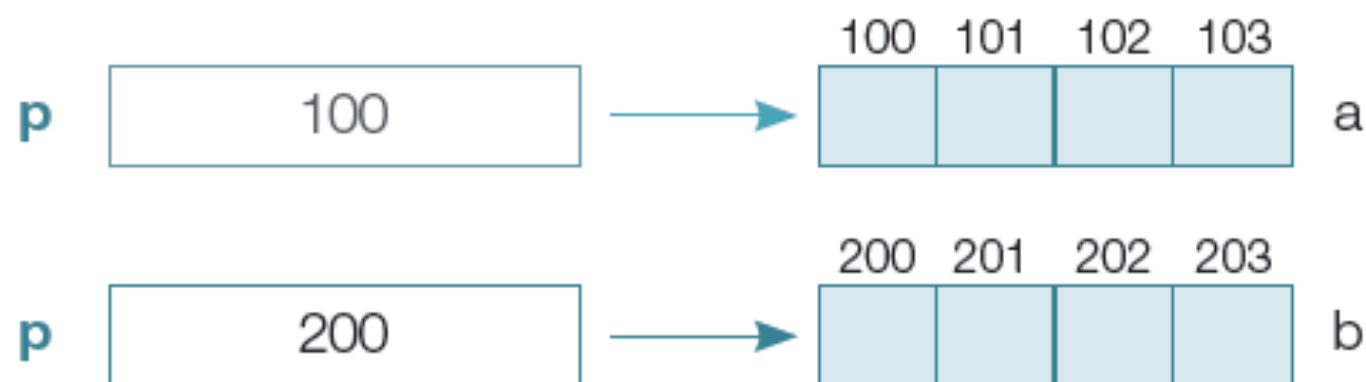
- ❖ 주소와 포인터의 차이 주소는 상수, 포인터는 변수
 - ▶ 주소는 변수에 할당된 메모리 저장 공간 시작 주소값 자체
 - ▶ 포인터는 그 값 저장하는 또 다른 메모리 공간
 - ▶ 특정 변수의 주소값은 바뀌지 않음
 - ▶ 포인터는 다른 주소 대입하여 그 값 바꿀 수 있음

포인터에 관한 궁금한 이야기

❖ 주소와 포인터의 차이

➤ 변수 a, b가 메모리에 할당된 상태 예시

```
int a, b;           // 일반 변수 선언
int *p;             // 포인터 선언
p = &a;             // p가 a를 가리키도록 설정
p = &b;             // p가 변수 b를 가리키도록 바꿈
```

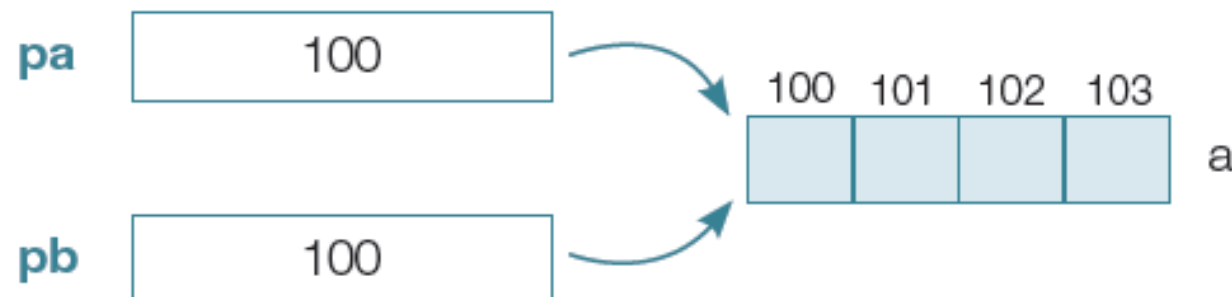


포인터에 관한 궁금한 이야기

❖ 주소와 포인터의 차이

- ▶ 두 포인터가 하나의 변수 동시에 가리키기

```
int a;           // 일반 변수 선언
int *pa, *pb;    // 가리키는 자료형이 같은 두 포인터
pa = pb = &a;    // pa와 pb에 모두 a의 주소를 저장한다.
```



- ▶ `a`값 바꾸거나 연산하는 데 `pa`와 `pb` 모두 사용 가능

```
*pa = 10;        // pa가 가리키는 변수 a에 10 대입
printf("%d", *pb); // pb가 가리키는 변수 a값 10 출력
```

예제 9-5 주소와 포인터의 크기

```
1. #include <stdio.h>
2.
3. int main(void)
4. {
5.     char ch;
6.     int in;
7.     double db;
8.
9.     char *pc = &ch;
10.    int *pi = &in;
11.    double *pd = &db;
12.
13.    printf("char형 변수의 주소 크기 : %d\n", sizeof (&ch));
14.    printf("int형 변수의 주소 크기 : %d\n", sizeof (&in));
15.    printf("double형 변수의 주소 크기 : %d\n", sizeof (&db));
16.
17.    printf("char * 포인터의 크기 : %d\n", sizeof (pc));
18.    printf("int * 포인터의 크기 : %d\n", sizeof (pi));
19.    printf("double * 포인터의 크기 : %d\n", sizeof (pd));
20.
21.    printf("char * 포인터가 가리키는 변수의 크기 : %d\n", sizeof (*pc));
22.    printf("int * 포인터가 가리키는 변수의 크기 : %d\n", sizeof (*pi));
23.    printf("double * 포인터가 가리키는 변수의 크기 : %d\n", sizeof (*pd));
24.
25.    return 0;
26. }
```

실행
결과

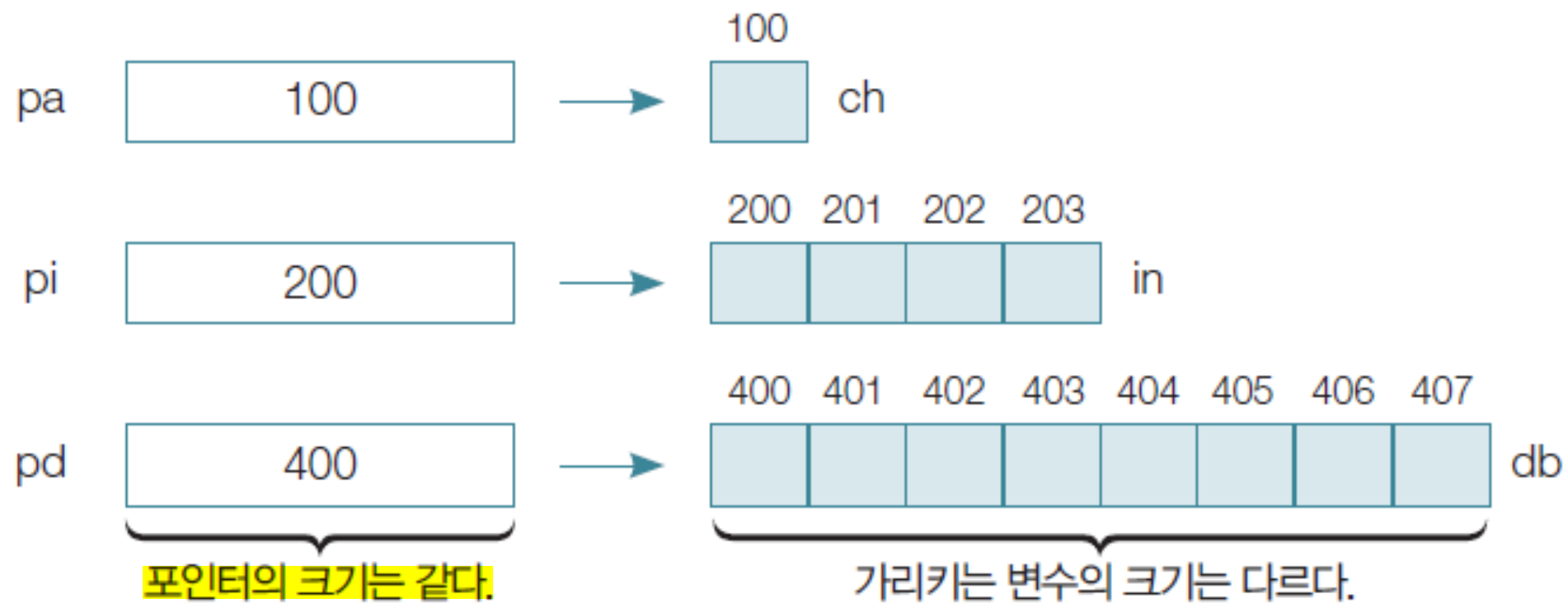
char형 변수의 주소 크기 : 4
int형 변수의 주소 크기 : 4
double형 변수의 주소 크기 : 4

char * 포인터의 크기 : 4
int * 포인터의 크기 : 4
double * 포인터의 크기 : 4

char * 포인터가 가리키는 변수의 크기 : 1
int * 포인터가 가리키는 변수의 크기 : 4
double * 포인터가 가리키는 변수의 크기 : 8

포인터에 관한 궁금한 이야기

- ▶ 포인터에 간접 참조 연산자 사용해 가리키는 변수 크기 출력 도식화



❖ 포인터의 대입 규칙

```

1. #include <stdio.h>
2.
3. int main(void)
4. {
5.     int a = 10;           // 변수 선언과 초기화
6.     int *p = &a;         // 포인터 선언과 동시에 a를 가리키도록 초기화
7.     double *pd;          // double형 변수를 가리키는 포인터
8.
9.     pd = p;              // 포인터 p값을 포인터 pd에 대입
10.    printf("%lf\n", *pd); // pd가 가리키는 변수의 값 출력
11.
12.    return 0;
13. }

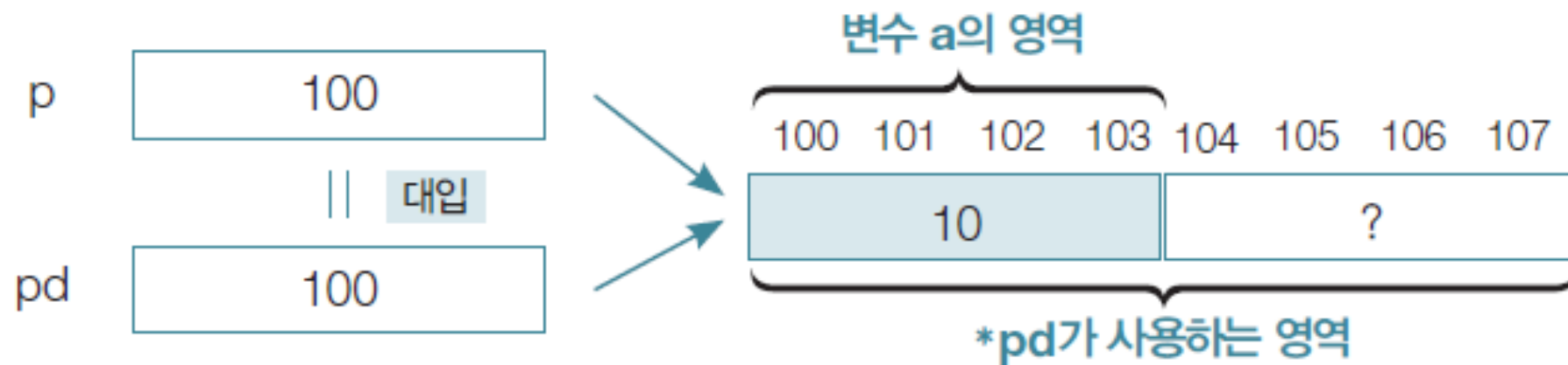
```


포인터에 관한 궁금한 이야기

.....

❖ 포인터의 대입 규칙

▶ 변수 a 메모리 100번지부터 할당 후 실행 결과



▶ 가리키는 자료형이 일치하지 않는 포인터의 대입을 시도하면 경고 메시지 출력

warning C4133: '=' : 'int *'과(와) 'double *' 사이의 형식이 호환되지 않습니다.

포인터에 관한 궁금한 이야기

❖ 포인터의 대입 규칙

- ▶ 형변환 사용한 포인터의 대입은 언제나 가능
 - ▶ 포인터가 가리키는 자료형이 다른 경우 형변환 연산자 사용하면 경고 메시지 없이 대입 가능
 - ▶ 대입한 후에 포인터를 통한 사용에 문제가 없어야 함

```
double a = 3.4;           // double형 변수 선언
double *pd = &a;          // pd가 double형 변수 a를 가리키도록 초기화
int *pi;                  // int형을 변수를 가리키는 포인터
pi = (int *)pd;           // pd값을 형변환하여 pi에 대입
```

포인터에 관한 궁금한 이야기

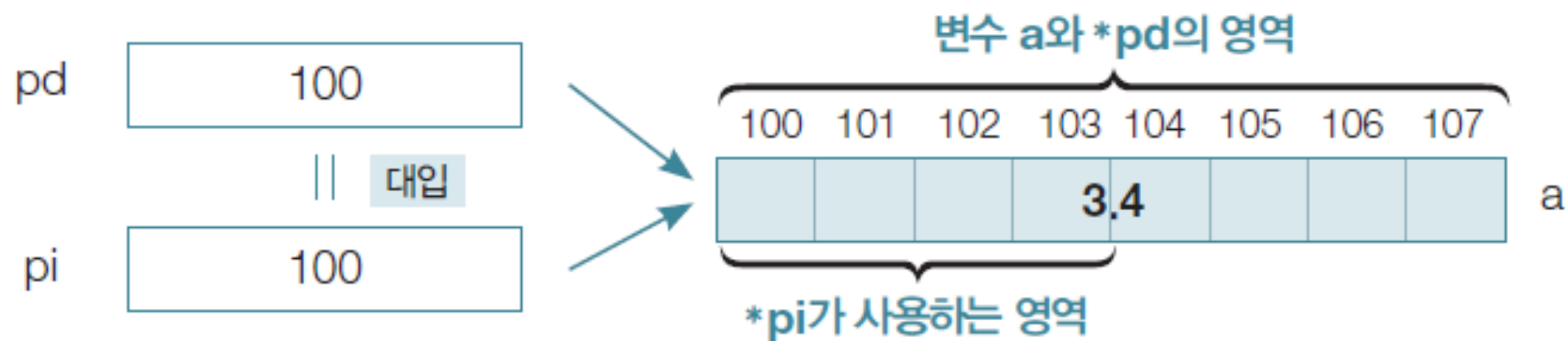
❖ 포인터의 대입 규칙

▶ pi에 간접참조 연산 수행 시

▶ 변수 a의 일부를 int형 변수처럼 사용할 수 있음

▶ 변수명 a와 포인터를 사용한 *pd와 *pi가 같은 공간 공유

▶ 어떤 경로를 통해서든 값이 변할 수 있으니 주의



포인터에 관한 궁금한 이야기

❖ 포인터의 대입 규칙

- ▶ 포인터에 100번지를 직접 대입하는 것은 가능할까?
 - ▶ 형변환 연산자 사용
 - ▶ 컴파일 경고나 에러 없이 원하는 정수값을 포인터에 대입하여 사용 가능
- ▶ Ex) 100번지~103번지까지 4바이트의 메모리 공간을 int형 변수로 쓰는 포인터 예제

```
int *p;  
p = (int *) 100;           // 100을 int형 변수의 주소로 형변환하여 p에 대입  
*p = 10;                   // 100번지부터 103번지까지 4바이트의 공간에 10 대입
```

포인터에 관한 궁금한 이야기

❖ 포인터의 대입 규칙

▶ 포인터 초기화하지 않는 건 더 위험

▶ 포인터에 간접참조 연산 수행하면 알 수 없는 곳으로 찾아가 데이터 바꿈

```
int *p;           // 초기화되지 않은 포인터
*p = 10;          // 쓰레기값 번지부터 4바이트의 공간에 10 대입
```



포인터에 관한 궁금한 이야기

❖ 포인터가 필요한 이유

- ▶ 변수 사용하는 가장 쉬운 방법은 이름을 쓰는 것
 - ▶ 포인터를 사용하려면 추가적인 변수 선언 필요
 - ▶ 각종 연산 수행해야 하므로 필요한 곳에만 사용
- ▶ 포인터가 반드시 필요한 경우
 - ▶ 임베디드 프로그래밍 할 때 메모리 직접 접근하는 경우
 - ▶ 동적 할당한 메모리 사용하는 경우

포인터에 관한 궁금한 이야기

.....
예제 9-7 포인터를 사용한 두 변수의 값 교환

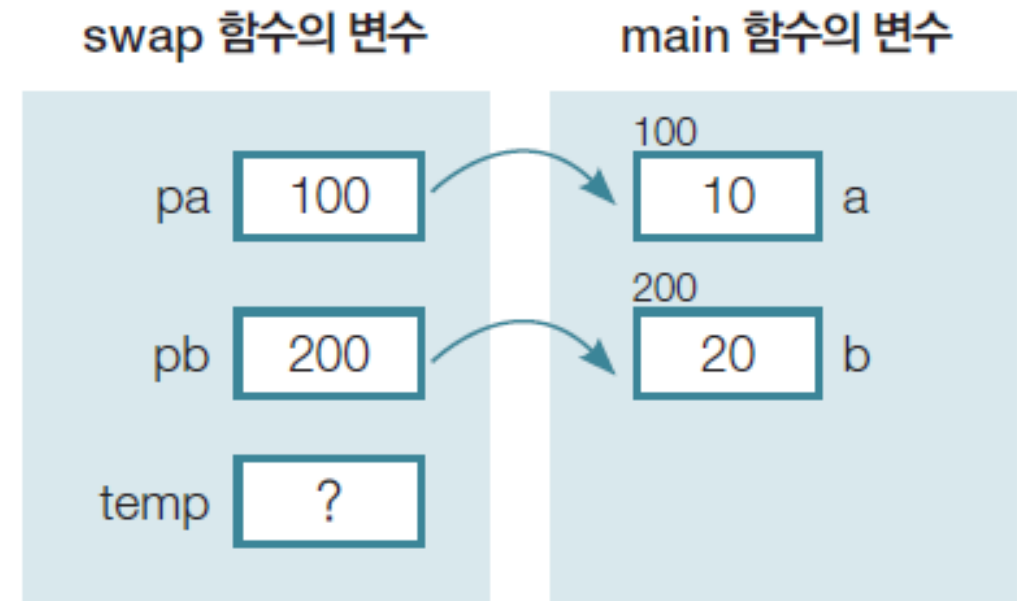
실행
결과 a:20, b:10

```
1. #include <stdio.h>
2.
3. void swap(int *pa, int *pb);      // 두 변수의 값을 바꾸는 함수의 선언
4.
5. int main(void)
6. {
7.     int a = 10, b = 20;          // 변수 선언과 초기화
8.
9.     swap(&a, &b);                 // a, b의 주소를 인수로 주고 함수 호출
10.    printf("a:%d, b:%d\n", a, b); // 변수 a, b 출력
11.
12.    return 0;
13. }
14.
15. void swap(int *pa, int *pb)      // 매개변수로 포인터 선언
16. {
17.     int temp;                    // 교환을 위한 임시 변수
18.
19.     temp = *pa;                  // temp에 pa가 가리키는 변수의 값 저장
20.     *pa = *pb;                  // pa가 가리키는 변수에 pb가 가리키는 변수의 값 저장
21.     *pb = temp;                 // pb가 가리키는 변수에 temp값 저장
22. }
```

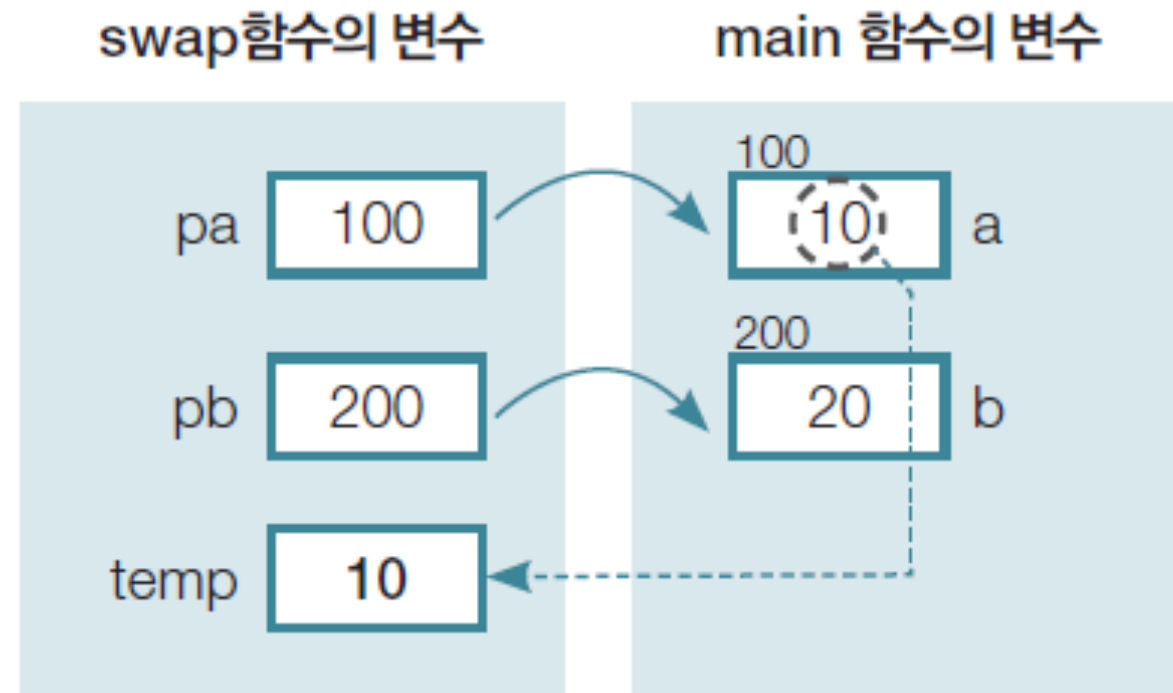
포인터에 관한 궁금한 이야기

❖ 포인터가 필요한 이유

- ▶ 함수 호출되면 포인터 pa, pb는 main 함수의 변수 a, b의 주소 저장



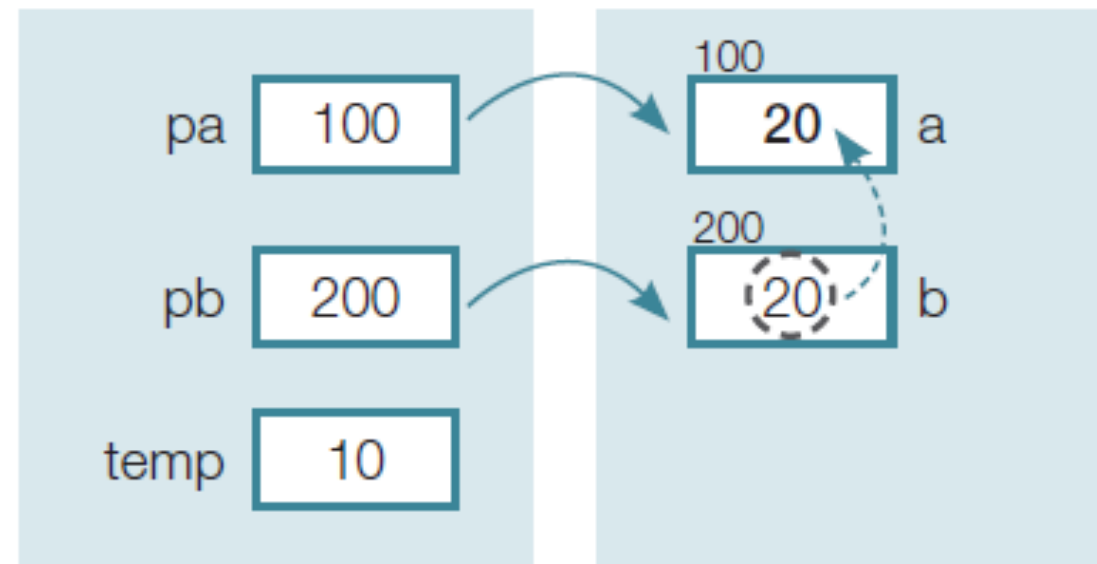
`temp = *pa`



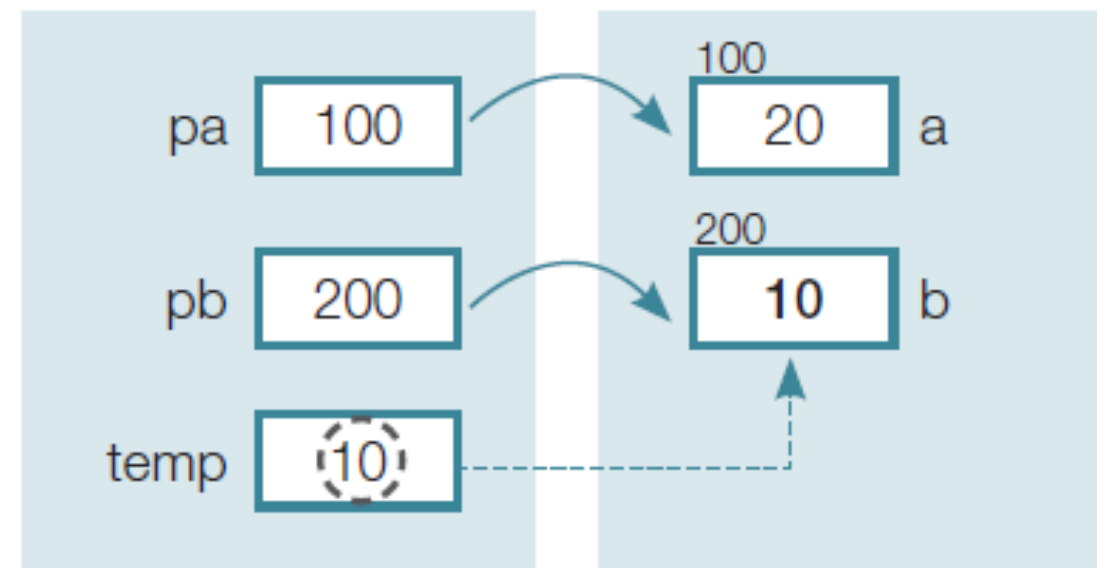
포인터에 관한 궁금한 이야기

❖ 포인터가 필요한 이유

`*pa = *pb`



`*pb = temp`



포인터에 관한 궁금한 이야기

예제 9-8 다른 함수의 변수를 이름으로 사용

```
1. #include <stdio.h>
2.
3. void swap(void);           // 두 변수의 값을 바꾸는 함수 선언
4.
5. int main(void)
6. {
7.     int a = 10, b = 20;     // 변수 선언과 초기화
8.
9.     swap();                 // 인수 없이 함수 호출
10.    printf("a:%d, b:%d\n", a, b); // 변수 a, b 출력
11.
12.    return 0;
13. }
14.
15. void swap(void)            // 인수가 없으므로 매개변수도 없음
16. {
17.     int temp;              // 교환을 위한 변수
18.
19.     temp = a;               // temp에 main 함수의 a값 저장
20.     a = b;                  // main 함수의 a에 main 함수의 b값 저장
21.     b = temp;              // main 함수의 b에 temp값 저장
22. }
```

포인터에 관한 궁금한 이야기

❖ 포인터가 필요한 이유

▶ 포인터 없이 두 변수의 값 바꾸기

▶ swap 함수에서 main 함수의 a, b를 이름으로 직접 사용하는 방법

▶ 컴파일 과정에서 에러 메시지 발생

```
error C2065: 'a' : 선언되지 않은 식별자입니다.    main.c    19
error C2065: 'a' : 선언되지 않은 식별자입니다.    main.c    20
error C2065: 'b' : 선언되지 않은 식별자입니다.    main.c    20
error C2065: 'b' : 선언되지 않은 식별자입니다.    main.c    21
```

예제 9-9 변수의 값을 인수로 주는 경우

```
1. #include <stdio.h>
2.
3. void swap(int x, int y);           // 두 변수의 값을 바꾸는 함수 선언
4.
5. int main(void)
6. {
7.     int a = 10, b = 20;           // 변수 선언과 초기화
8.
9.     swap(a, b);                   // a, b의 값을 복사해서 전달
10.    printf("a:%d, b:%d\n", a, b); // 변수 a, b 출력
11.
12.    return 0;
13. }
14.
15. void swap(int x, int y)           // 인수 a, b의 값을 x, y에 복사해서 저장
16. {
17.     int temp;                     // 교환을 위한 변수
18.
19.     temp = x;                     // temp에 x값 저장
20.     x = y;                         // x에 y값 저장
21.     y = temp;                     // y에 temp값 저장
22. }
```

.....

포인터에 관한 궁금한 이야기

❖ 포인터가 필요한 이유

- main 함수에서 a, b값을 swap 함수에 인수로 주는 방법
 - 이름이 같아도 함수가 다르면 메모리에 별도 저장 공간 확보
 - 다른 변수로 사용
- swap 함수에서 바꾼 값을 main 함수로 반환하는 방법
 - 함수는 오직 하나의 값만 반환 가능
 - 한 번의 함수호출을 통해 두 변수 값 바꾸는 것은 불가능