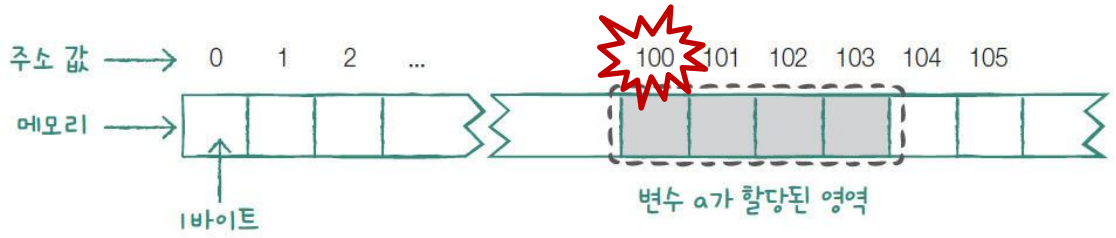
9장 포인터

09- 1 포인터의 기본 개념

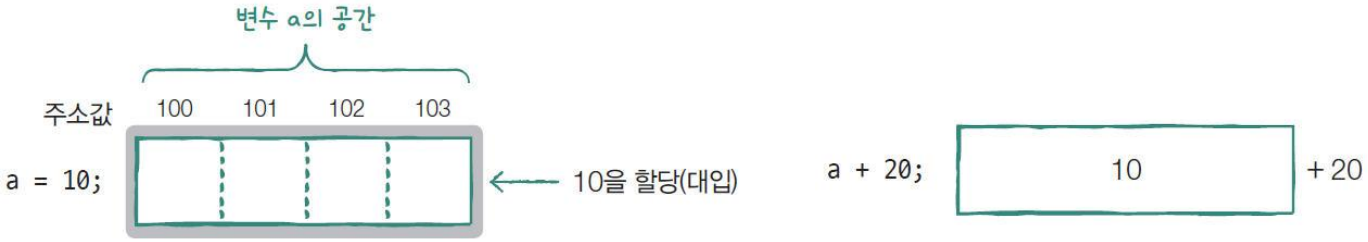
❖ 메모리의 주소

- 주소(address) 는 변수가 할당된 메모리의 시작 주소다.

```
int a;
```



- 변수의 공간(l-value)이나 값(r-value)은 이름으로 사용한다.



- 주소를 알면 주소로도 변수의 공간이나 값을 사용할 수 있다.

09- 1 포인터의 기본 개념

❖ 주소 연산자 : & (1/2)

변수의 메모리 주소 확인 소스 코드 예제9-1.c

```
01  #include <stdio.h>
02
03  int main(void)
04  {
05      int a;      // int형 변수 선언
06      double b;   // double형 변수 선언
07      char c;     // char형 변수 선언
08
09      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  }
```

실행결과

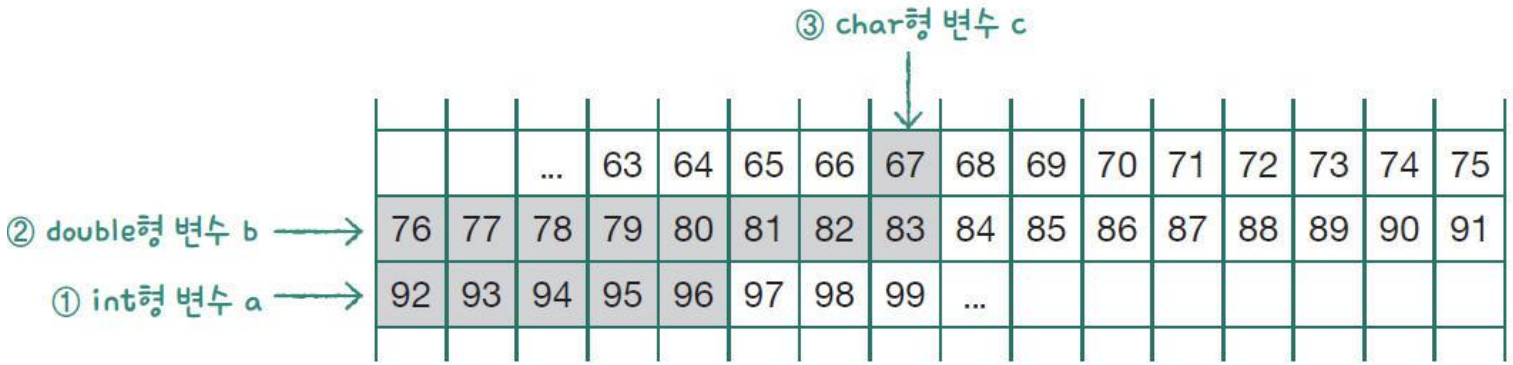
```
int형 변수의 주소 : 2750392
double형 변수의 주소 : 2750376
char형 변수의 주소 : 2750367
```

❖ 주소 연산자 : & (2/2)

- 출력 결과로 변수가 할당된 메모리 상태를 확인할 수 있다.

실행결과

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



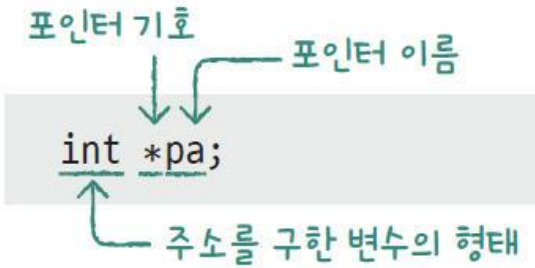
09- 1 포인터의 기본 개념

❖ 포인터와 간접 참조 연산자 : * (1/3)

포인터의 선언과 사용 소스 코드 예제9-2.c

```
01 #include <stdio.h>
02
03 int main(void)
04 {
05     int a;                  // 일반 변수 선언
06     int *pa;                // 포인터 선언
07
08     pa = &a;                // 포인터에 a의 주소 대입
09     *pa = 10;               // 포인터로 변수 a에 10 대입
10
11     printf("포인터로 a 값 출력 : %d\n", *pa);
12     printf("변수명으로 a 값 출력 : %d\n", a);     // 변수 a 값 출력
13
14     return 0;
15 }
```

int* pa;로 자동으로 위치 조정되기도 합니다.
결과에는 영향이 없습니다.



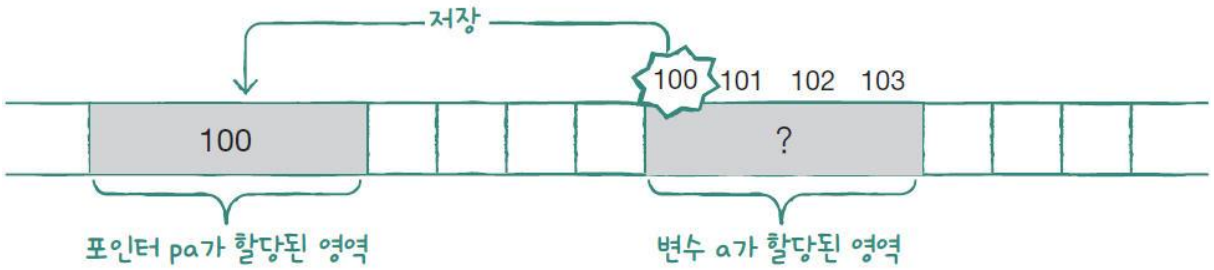
실행결과

```
포인터로 a 값 출력 : 10
변수명으로 a 값 출력 : 10
```

❖ 포인터와 간접 참조 연산자 : * (2/3)

- 포인터는 변수의 시작 주소를 저장한다.

```
pa = &a;
```

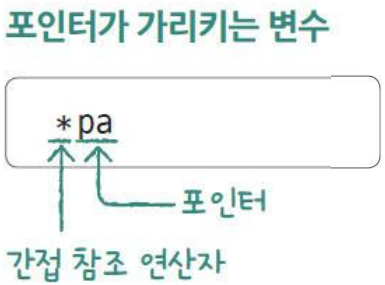


- 포인터가 변수를 '**가리킨다**' 말하고 화살표로 표현한다.

pa → a ← 포인터 pa는 변수 a를 가리킨다!

❖ 포인터와 간접 참조 연산자 : * (3/3)

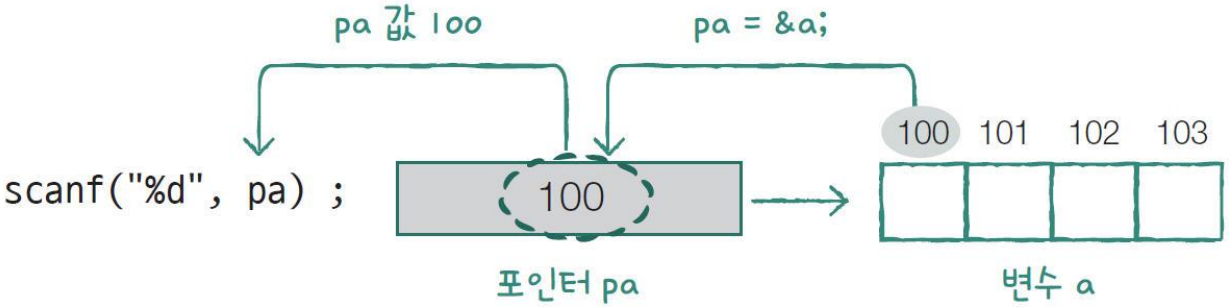
- 포인터로 가리키는 변수를 사용할 때는 *연산자를 쓴다.




```
*pa = 10;       // 9행. 포인터로 변수 a에 10 대입

printf("포인터로 a 값 출력 : %d\n", *pa);       // 11행
```

- scanf 함수로 입력할 때는 포인터만 쓸 수 있다.



❖ 여러 가지 포인터 사용해보기 (1/2)

포인터를 사용한 두 정수의 합과 평균 계산 소스 코드 예제9-3.c

```
01  #include <stdio.h>
02
03  int main(void)
04  {
05      int a = 10, b = 15, total;
06      double avg;
07      int *pa, *pb;
08      int *pt = &total;
09      double *pg = &avg;
10
11      pa = &a;
12      pb = &b;
13
14      *pt = *pa + *pb;
15      *pg = *pt / 2.0;
16
17      printf("두 정수의 값 : %d, %d\n", *pa, *pb);
18      printf("두 정수의 합 : %d\n", *pt);
19      printf("두 정수의 평균 : %.1lf\n", *pg);
20
21      return 0;
22  }
```

실행결과

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

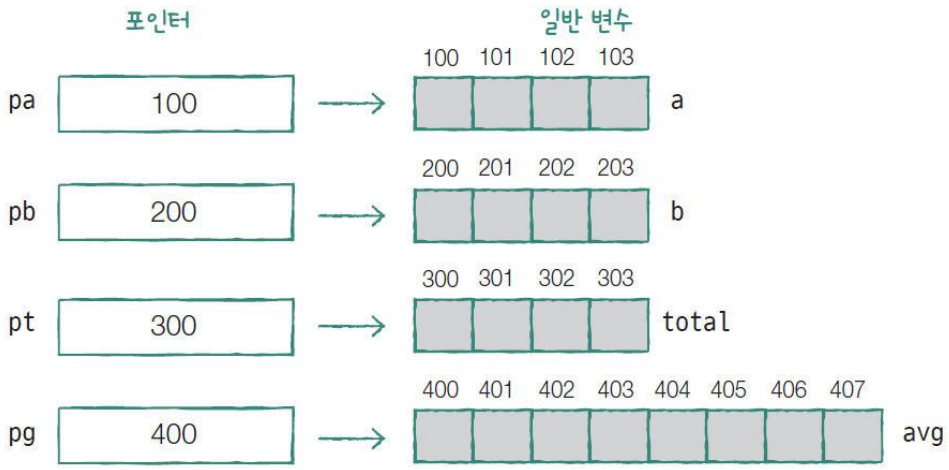
❖ 여러 가지 포인터 사용해보기 (2/2)

- 가리키는 자료형에 맞게 포인터를 선언한다.

```
double *pg = &avg;                    // 9행
```

avg가 double형 변수이므로 pg 앞에 double을 붙임

- 가리키는 자료형과 상관없이 항상 첫번째 주소만 저장한다.



09- 1 포인터의 기본 개념

❖ const를 사용한 포인터

포인터에 const 사용 소스 코드 예제9-4.c

```c
#include <stdio.h>

int main(void)
{
    int a = 10, b = 20;
    const int *pa = &a;                 // 포인터 pa는 변수 a를 가리킨다.

    printf("변수 a 값 : %d\n", *pa);    // 포인터를 간접 참조하여 a 출력
    pa = &b;                            // 포인터가 변수 b를 가리키게 한다.
    printf("변수 b 값 : %d\n", *pa);    // 포인터를 간접 참조하여 b 값 출력
    pa = &a;                            // 포인터가 다시 변수 a를 가리킨다.
    a = 20;                             // a를 직접 참조하여 값을 바꾼다.
    printf("변수 a 값 : %d\n", *pa);    // 포인터로 간접 참조하여 바뀐 값 출력

    return 0;
}
```

*pa = 20; (×)

a는 pa를 간접 참조하여 바꿀 수 없다

실행결과

```
변수 a 값 : 10
변수 b 값 : 20
변수 a 값 : 20
```

마무리

키워드로 끝내는 핵심 포인트

- **포인터**는 메모리를 사용하는 또 다른 방법이다.
- **주소 연산자 &**로 변수가 할당된 메모리의 위치를 확인한다.
- 포인터로 가리키는 변수를 사용할 때 **간접 참조 연산자 ***를 쓴다.

마무리

표로 정리하는 핵심 포인트

표 9-1 포인터와 연산자

구분	사용 예	기능
주소 연산자	`int a;` `&a;`	변수 앞에 붙여 사용하며, 변수가 할당된 메모리의 시작 주소 값을 구한다.

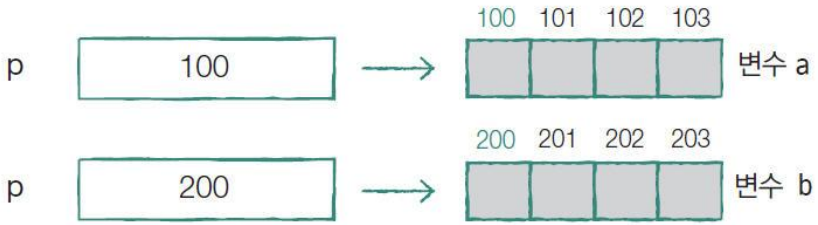
포인터	`char *pc;`
	`int *pi;`
	`double *pd;`
		시작 주소 값을 저장하는 변수며, 가리키는 자료형을 표시하여 선언한다.

간접 참조 연산자	`*pi = 10;`	포인터에 사용하며, 포인터가 가리키는 변수를 사용한다.

❖ 주소와 포인터의 차이

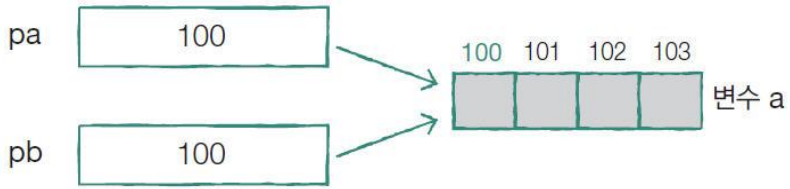
- 주소는 상수고 포인터는 변수라서 포인터의 값은 바뀔 수 있다.

```
int a, b;
int *p;
p = &a;
p = &b;
```



- 두 개 이상의 포인터가 하나의 변수를 가리킬 수 있다.

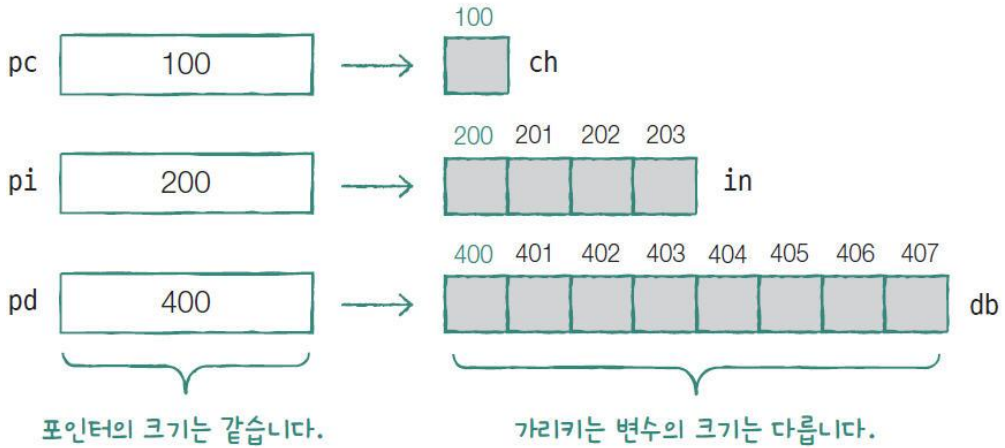
```
int a;
int *pa, *pb;
pa = pb = &a;
```



❖ 주소와 포인터의 크기 (1/2)

주소와 포인터의 크기 소스 코드 예제9-5.c

```
01  #include <stdio.h>
02
03  int main(void)
04  {
05      char ch;
06      int in;
07      double db;
08
09      char *pc = &ch;
10      int *pi = &in;
11      double *pd = &db;
12
```



❖ 주소와 포인터의 크기 (2/2)

```
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  }
```

실행결과

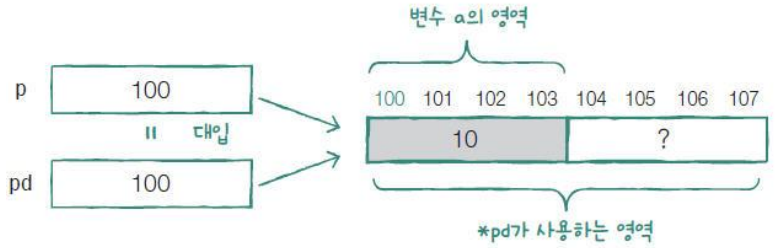
```
char형 변수의 주소 크기 : 4
int형 변수의 주소 크기 : 4
double형 변수의 주소 크기 : 4
char * 포인터의 크기 : 4
int * 포인터의 크기 : 4
double * 포인터의 크기 : 4
char * 포인터가 가리키는 변수의 크기 : 1
int * 포인터가 가리키는 변수의 크기 : 4
double * 포인터가 가리키는 변수의 크기 : 8
```

09- 2 포인터의 완전 정복을 위한 포인터 이해하기

❖ 포인터의 대입 규칙 (1/2)

허용되지 않는 포인터의 대입 소스 코드 예제9-6.c

```
01  #include <stdio.h>
02
03  int main(void)
04  {
05      int a = 10;             // 변수 선언과 초기화
06      int *p = &a;            // 포인터 선언과 동시에 a를 가리키도록 초기화
07      double *pd;             // double형 변수를 가리키는 포인터
08
09      pd = p;                 // 포인터 p 값을 포인터 pd에 대입
10      printf("%lf\n", *pd);   // pd가 가리키는 변수의 값 출력
11
12      return 0;
13  }
```



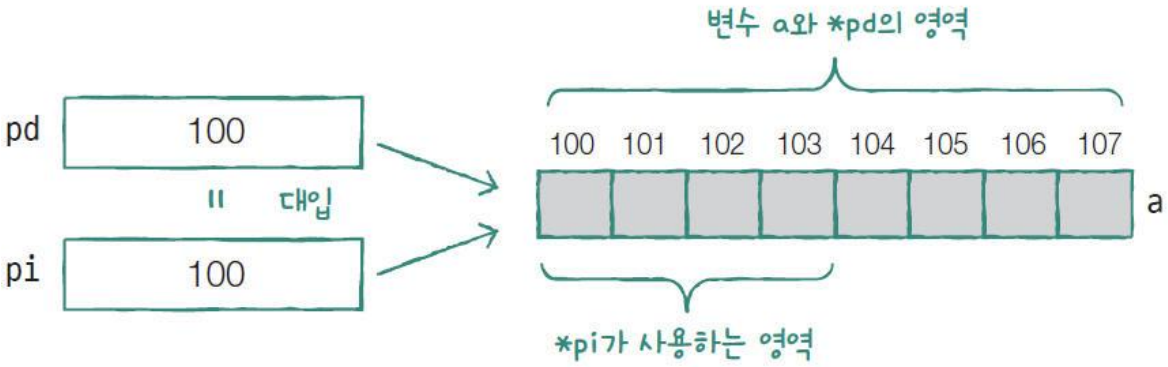
실행결과

```
-92559592117432108000000000000000000000000000000000000000000000000.000000
```

❖ 포인터의 대입 규칙 (2/2)

- 형 변환을 사용한 포인터의 대입은 언제나 가능하다.

```
double a = 3.4;        // double형 변수 선언
double *pd = &a;       // pd가 double형 변수 a를 가리키도록 초기화
int *pi;               // int형 변수를 가리키는 포인터
pi = (int *)pd;        // pd 값을 형 변환하여 pi에 대입
```

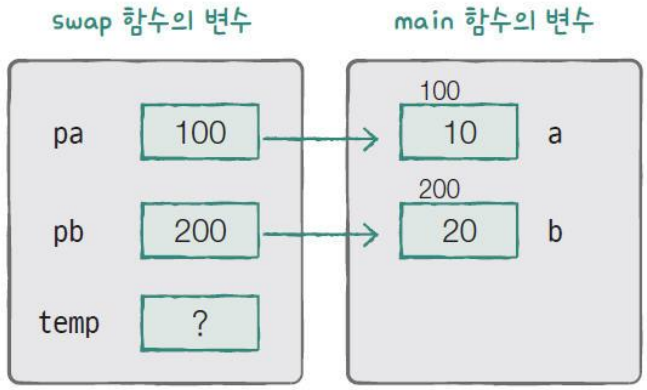


09- 2 포인터의 완전 정복을 위한 포인터 이해하기

❖ 포인터를 사용하는 이유 (1/2)

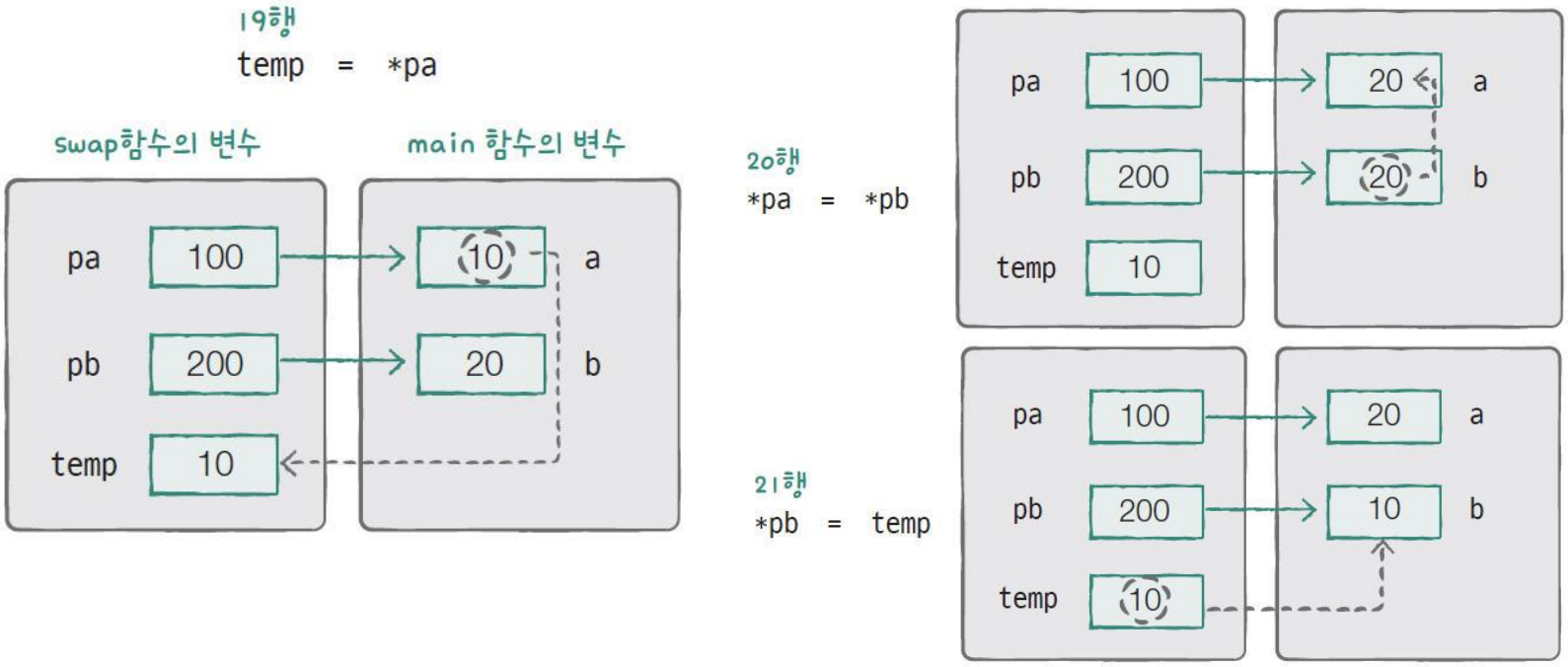
포인터를 사용한 두 변수의 값 교환 소스 코드 예제9-7.c

```
01  #include <stdio.h>
02
03  void swap(int *pa, int *pb);
04
05  int main(void)
06  {
07      int a = 10, b = 20;
08
09      swap(&a, &b);
10      printf("a:%d, b:%d\n", a, b);
11
12      return 0;
13  }
14
15  void swap(int *pa, int *pb)
16  {
17      int temp;
18
19      temp = *pa;
20      *pa = *pb;
21      *pb = temp;
22  }
```



❖ 포인터를 사용하는 이유 (2/2)

```
temp = *pa; // 19행. temp에 pa가 가리키는 변수의 값 저장
*pa = *pb;  // 20행. pa가 가리키는 변수에 pb가 가리키는 변수의 값 저장
*pb = temp; // 21행. pb가 가리키는 변수에 temp 값 저장
```



09- 2 포인터의 완전 정복을 위한 포인터 이해하기

❖ 포인터 없이 두 변수의 값을 바꾸는 함수는? (1/2)

다른 함수의 변수 사용하기 소스 코드 예제9-8.c

```
01  #include <stdio.h>
02
03  void swap(void);
04
05  int main(void)
06  {
07      int a = 10, b = 20;
08
09      swap();
10      printf("a:%d, b:%d\n", a, b);
11
12      return 0;
13  }
14
15  void swap(void)
16  {
17      int temp;
18
19      temp = a;
20      a = b;
21      b = temp;
22  }
```

에러 ×

```
error C2065: 'a' : 선언되지 않은 식별자입니다.
error C2065: 'a' : 선언되지 않은 식별자입니다.
error C2065: 'b' : 선언되지 않은 식별자입니다.
error C2065: 'b' : 선언되지 않은 식별자입니다.
```

❖ 포인터 없이 두 변수의 값을 바꾸는 함수는? (2/2)

변수의 값을 인수로 주는 경우 소스 코드 예제9-9.c

```c
#include <stdio.h>

void swap(int x, int y);

int main(void)
{
    int a = 10, b = 20;

    swap(a, b);
    printf("a:%d, b:%d\n", a, b);

    return 0;
}

void swap(int x, int y)
{
    int temp;

    temp = x;
    x = y;
    y = temp;
}
```

실행결과

```
a:10, b:20
```

마무리

키워드로 끝내는 핵심 포인트

- **주소**와 **포인터**는 상수와 변수의 차이가 있다.
- **포인터의 크기**는 주소의 크기와 같다.
- **포인터**에 주소를 저장할 때는 가리키는 **자료형**이 같아야 한다.
- **포인터의 주요 기능** 중 하나는 함수 간에 효과적으로 데이터를 공유하는 것이다.

마무리

표로 정리하는 핵심 포인트 (1/2)

표 9-2 간접 참조 연산자를 사용한 예(포인터 pa가 변수 a를 가리킬 때)

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

마무리

표로 정리하는 핵심 포인트 (2/2)

표 9-3 주소와 포인터의 특징

구분	사용 예	기능

포인터	`int a, b;`
	`int *p = &a ;`
	`p = &b;`
		포인터는 변수이므로 그 값을 다른 주소로 바꿀 수 있다.

포인터의 크기	`int *p;` `sizeof(p)`	포인터의 크기는 컴파일러에 따라 다를 수 있으며, sizeof 연산자로 확인한다.

포인터의 대입 규칙	`int *p;`
	`double *pd;`
	`pd = p; ( X )`
		포인터는 가리키는 자료형이 일치할 때만 대입한다.