

09-1. 포인터의 기본 개념

C언어에서 메모리에 접근하는 가장 쉬운 방법은 변수를 사용하는 것이었다. 요 단원에서는 메모리의 주소 값을 이용하는 포인터에 대하여 살펴보자.

변수 선언으로 메모리에 공간을 확보하고, 그곳을 꺼내 쓰는 공간으로 사용하였다. 변수명은 그런 메모리 공간을 식별할 수 있는 방 이름이다.

하지만, 변수의 단점은 선언된 블록, 함수 내부로 사용이 제한되어 있다. 같은 변수명을 사용하더라도 블록이나 함수가 다르면 별도의 저장 공간을 확보하게 되어 전혀 다른 변수로 사용되는 것이다.

포인터는 사용 범위를 벗어난 경우도 데이터를 공유할 수 있는 새로운 방법이다.

1) 메모리 주소

프로그램은 사용하는 메모리의 위치를 주소 값으로 식별가능하다. 메모리의 위치를 식별하는 주소 값은 바이트 단위로 구분된다. 이 값은 0부터 시작하고 바이트 단위로 1씩 증가하므로 2 바이트 이상의 크기를 갖는 변수는 여러 개의 주소 값에 걸쳐 할당된다.

2) 주소 연산자 &

저장된 공간에 주소로 사용하는 방법을 알아보자.

먼저 '주소' 라고 하면 변수가 할당된 메모리 공간의 시작 주소를 의미한다. 시작 주소를 알면 그 위치부터 변수의 크기만큼 메모리를 사용할 수 있다. 주소는 주소 연산자 &를 사용하여 구한다.

출력 : &변수명 [주소연산자][변수명]

주소연산자 &는 단항 연산자이며 변수만을 피연산자로 사용해 시작 주소를 구한다.

- 예제

```
#include <stdio.h>

int main(void){
    int a;
    double b;
    char c;

    printf("int형 변수의 주소 : %u\n", &a);
    printf("double형 변수의 주소 : %u\n", &b);
    printf("char형 변수의 주소 : %u\n", &c);

    return 0;
}
```

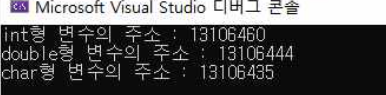


그림 1. 결과

```
#include <stdio.h>

int main(void){
    int a;
    double b;
    char c;

    printf("int형 변수의 주소 : %p\n", &a);
    printf("double형 변수의 주소 : %p\n", &b);
    printf("char형 변수의 주소 : %p\n", &c);

    return 0;
}
```

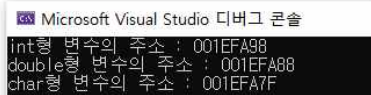
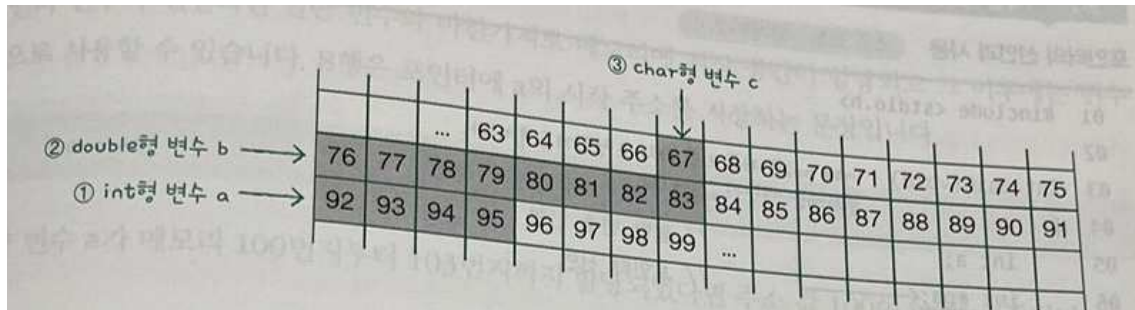


그림 2. 에러나서 수정한 이미지

컴퓨터는 프로그램 실행 후 남아 있는 메모리를 활용하므로 실행 결과가 다르게 나올 수 있다. 그래서 그림 1처럼 계속해서 다른 주소값이 나온다. 이를 실행하면 출력 변환 문자가 맞지 않아 %p로 바꾸라는 경고 메시지가 나오는데 그대로 실행시켜도 무방하다. (%p로 바꾸면 16진수로 표기한다.)

주소 연산자 &를 사용하여 변수에 할당된 메모리의 시작 주소를 확인하고 시작 주소에 변수의 크기를 더하면 변수가 메모리의 어디서부터 어디까지 할당되었는지 확인 가능하다.



3) 포인터와 간접 참조 연산자 *

변수에 할당된 메모리 주소를 활용하는 방법을 살펴보자.

메모리의 주소는 필요할 때마다 계속 주소 연산을 수행하는 것보다 한 번 구한 주소를 저장해서 사용하면 편리한데, 포인터가 변수의 메모리 주소를 저장하는 변수이다.

따라서 주소를 저장할 포인터도 변수처럼 선언해서 사용한다.

차별점이 있다면 변수 앞에 *를 붙여주면 된다.

출력 : 자료형 *변수명;

이때 자료형은 저장할 주소가 어떤 변수의 주소인지 변수의 자료형을 저장하면 된다.

-예제

```
#include <stdio.h>

int main(void){
    int a;
    int *pa;

    pa = &a;
    *pa = 10;

    printf("포인터로 a값 출력 : %d\n", *pa);
    printf("변수명으로 a값 출력 : %d\n", a);

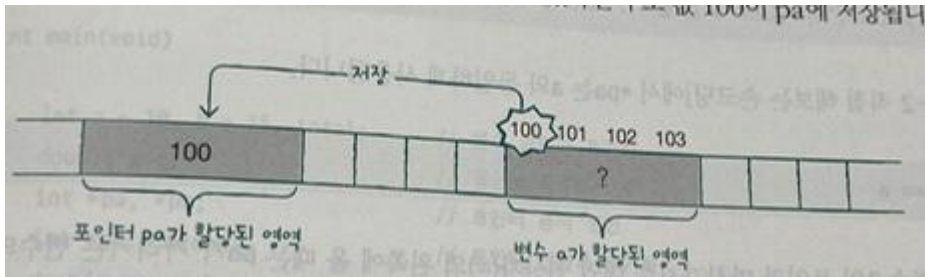
    return 0;
}
```

Microsoft Visual Studio 디버그 콘솔

```
포인터로 a값 출력 : 10
변수명으로 a값 출력 : 10
```

pa = &a; // 포인터에 a의 주소를 대입한다.

*pa= 10; // 포인터로 변수 a에 10을 대입한다.



포인터 pa는 변수 a가 메모리 어디에 할당되었는지 위치를 기억한다. 이렇게 포인터가 어떤 변수의 주소를 저장한 경우 가리킨다고 표현하고 둘의 관계를 $pa \rightarrow a$ 로 표시한다.

포인터가 어떤 변수를 가리키면 포인터로 가리키는 변수를 사용할 수 있다.

즉, 포인터 pa로 변수 a를 사용 가능하다,

포인터가 가리키는 변수를 사용할 때는 포인터에 특별한 연산자를 사용하는데,

이를 간접 참조 연산자 * 라고 하거나 포인터 연산자라고 한다.

`*pa == a`

*pa는 변수 a의 쓰임과 마찬가지로 대입 연산자의 왼쪽에 올 때 pa가 가리키는 변수의 저장 공간으로 사용되고, 오른쪽에 올때는 pa가 가리키는 변수의 값으로 사용된다.

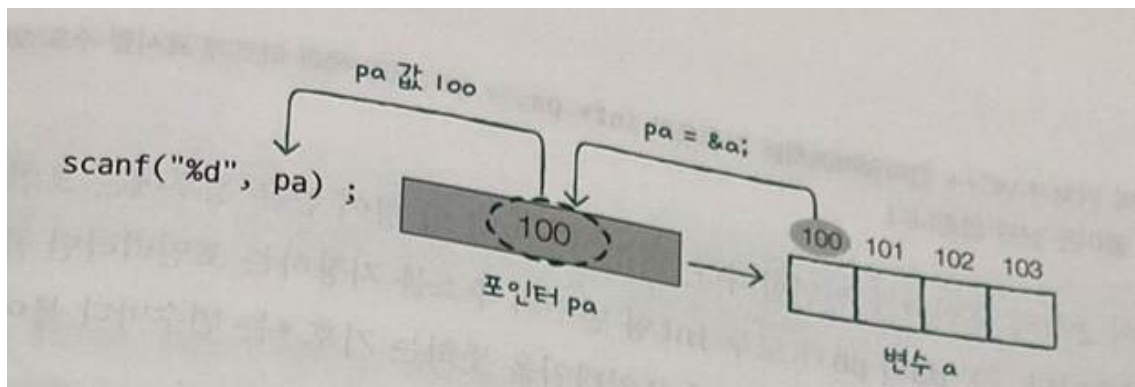
scanf는 입력하는 변수가 메모리 어디에 할당되었는지 저장 공간의 위치를 알아야 한다.

따라서 입력할 변수의 주소를 인수로 준다.

scanf를 쓰는 방법은 다음과 같다.

&a로 변수 a의 저장 공간 찾기

`scanf("%d", &a)` 또는 `scanf("%d", pa)`



4) 여러 가지 포인터 사용해보기

포인터가 어떤 변수를 가리키게 되면 그 후엔 간접 참조 연산자를 통해 가리키는 변수를 자유롭게 쓸 수 있다.

-예제


```
#include <stdio.h>

int main(void){
    int a=10, b=15, total;
    double avg;
    int* pa, * pb;
    int* pt = &total;
    double* pg = &avg;

    pa = &a;
    pb = &b;

    *pt = *pa + *pb;
    *pg = *pt / 2.0;

    printf("두 정수의 값 : &d, %d\n", *pa, *pb);
    printf("두 정수의 합 : %d\n", *pt);
    printf("두 정수의 평균 : %.1f\n", *pg);
}
```



변수를 선언, 초기화

포인터 선언, 초기화

double형 포인터 선언, 초기화를 먼저 해준다.

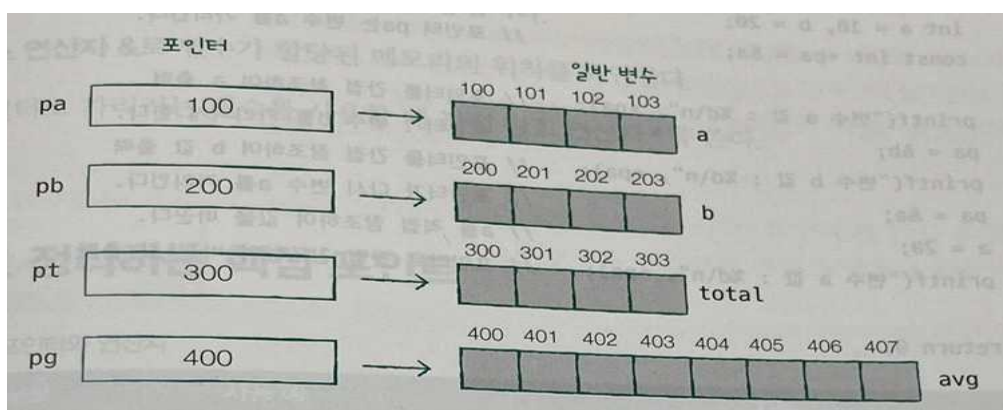
포인터 pa에 a 주소 대입, 포인터 pb에 b 주소 대입

int *pt

pt = &total 이거를 한번에

int *pt = &total로 할 수 있다.

a와 b를 더해 total에 저장하고, 평균을 구함



5) const를 사용한 포인터

const는 예약어를 포인터에 사용하면 가리키는 변수의 값을 바꿀 수 없다는 의미로, 변수에 사용 하는 것과 다른 의미를 가진다.

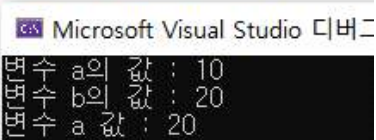
-예제

```
#include <stdio.h>

int main(void){
    int a = 10, b = 20;
    const int* pa = &a;

    printf("변수 a의 값 : %d\n", *pa);
    pa = &b;
    printf("변수 b의 값 : %d\n", *pa);
    pa = &a;
    a = 20;
    printf("변수 a 값 : %d\n", *pa);

    return 0;
}
```



6행에서 포인터 `pa`를 선언할 때 `const`로 상수화하였다. 만약 `const`가 일반 변수처럼 포인터를 고정 시키면, `pa`는 다른 변수의 주소를 저장할 수 없다. 그러나 출력 결과에서 `const`의 사용과는 무관하게 변수 `b`의 주소를 저장하고 그 값을 간접 참조하여 출력한다.

포인터에 사용된 `const`의 의미는?

`pa`가 가리키는 변수 `a`는 `pa`를 간접 참조하여 바꿀 수 없다는 것이다.

6) 마무리

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

포인터의 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.

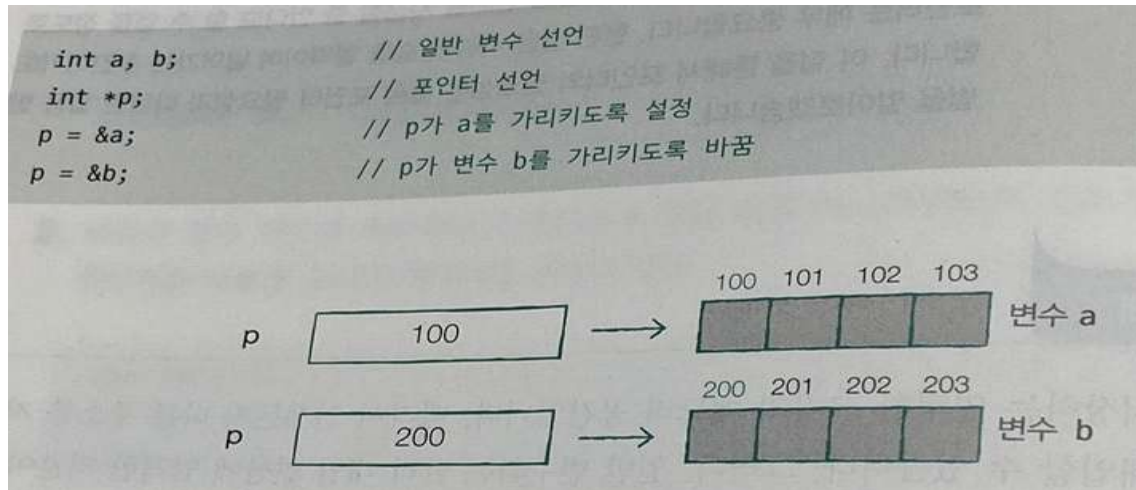
7) 확인 문제

09-2. 포인터 완전 정복을 위한 포인터 이해하기

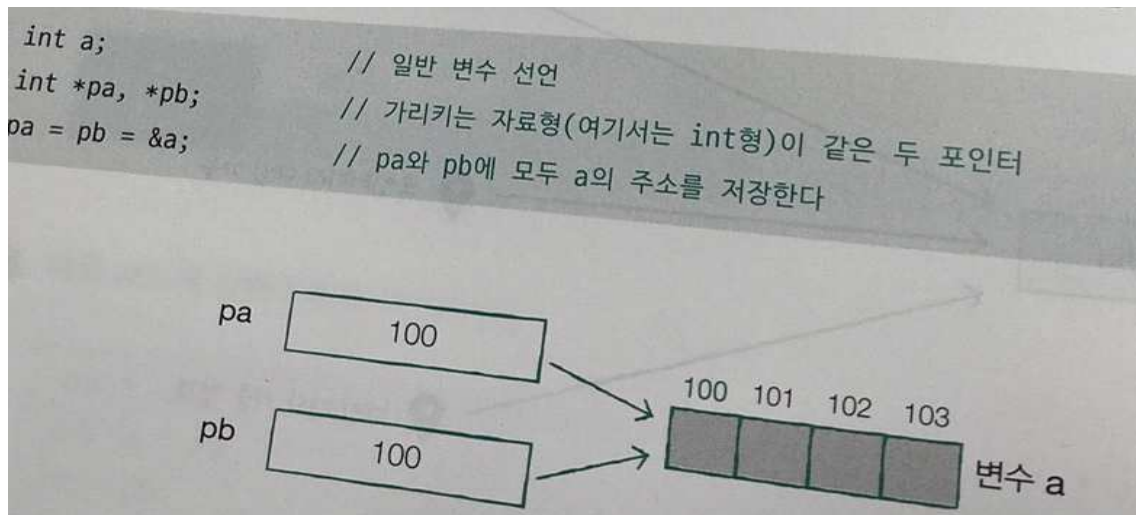
포인터 = 주소 저장, 포인터끼리 대입 가능, 대입 연산 기준 엄격
(값을 언제든지 바꿀 수 있다.)

1) 주소와 포인터의 차이

주소는 변수에 할당된 메모리 저장 공간의 시작 주소 값 자체이고, 포인터는 그 값을 저장하는 또 다른 메모리 공간이다. 따라서 특정 변수의 주소 값은 바뀌지 않지만 포인터는 다른 주소를 대입해 그 값을 바꿀 수 있다.



주소는 상수이고 포인터는 변수이다.
두 포인터가 같은 주소를 저장할 수도 있다.



2) 주소와 포인터의 크기

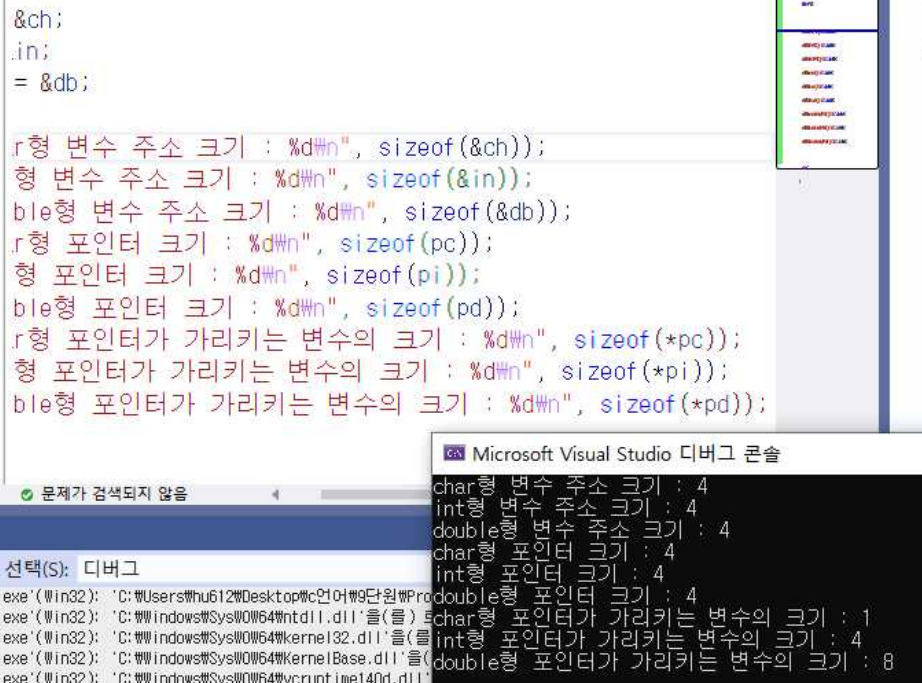
포인터도 저장 공간이기 때문에 크기가 있다. 포인터의 크기는 저장할 주소의 크기에 따라 결정되는데 크기가 클수록 더 넓은 범위의 메모리를 사용가능하다.

포인터의 크기는 컴파일러에 따라 다를 수 있으나 모든 주소와 포인터는 자료형과 상관없이 크기가 같다.

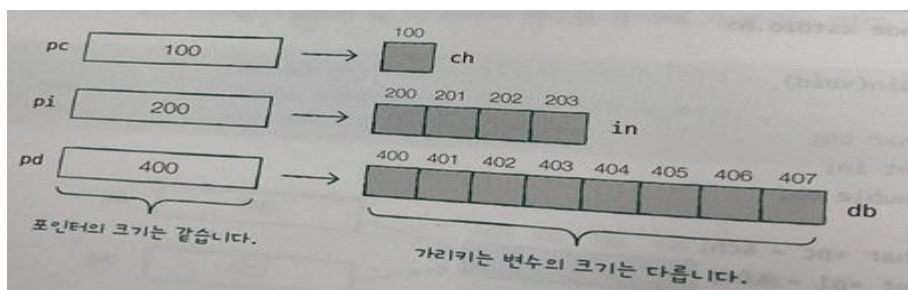
- 예시

```
char ch;
int in;
double db;

char* pc = &ch;
int* pi = &in;
double* pd = &db;
```



ch, in, db 모두 각각 변수 자체의 크기는 다르지만, 시작 주소 값의 크기는 모두 같다.
따라서 포인터가 가리키는 자료형도 상관없이 모두 크기가 같다.
그렇지만 포인터가 가리키는 변수의 크기는 실제 자료형의 크기이기 때문에 1,4,8이 출력이 된다.



3) 포인터의 대입 규칙

- 포인터는 가리키는 변수의 형태가 같을 때만 대입해야 한다.

예시)

```
#include <stdio.h>

int main(void)
{
    int a = 10;
    int* p = &a;
    double* pd;

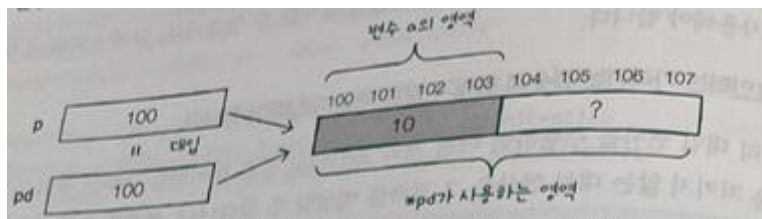
    pd = p;
    printf("%lf\n", *pd);

    return 0;
}
```

Microsoft Visual Studio 디버그 콘솔

```
-92559592117432107884277659021957555520241347761778250032873472.000000
```

pd는 double형인데 p 라는 int형을 입력해서 a에 할당된 영역 이후 할당되지 않은 영역까지 사용하게 된다.



- 형 변환을 사용한 포인터의 대입은 언제나 가능하다.
- 방금 같은 사례에서 형 변환을 사용했다면 이상한 수가 나오지 않는다.

예시)

```
double a = 3.4;
double *pd = &a;
int *pi;
pi = (int *)pd; # pd 값을 형변환해서 pi에 대입
```

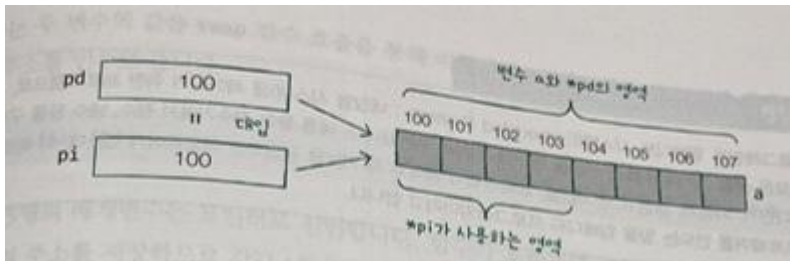
```
int main(void)
{
    double a = 3.4;
    double* pd = &a;
    int* pi;
    pi = (int*)pd;

    printf("%d\n", pi);

    return 0;
}
```

Microsoft Visual Studio 디버그 콘솔

```
17824112
```

4) 포인터를 사용하는 이유

변수를 사용하는 가장 쉬운 방법은 이름을 쓰는 것이다. 포인터를 사용하려면 추가적인 변수 선언이 필요하고 주소 연산, 간접 참조 연산 등 각종 연산을 수행해야 한다.

포인터는 주로 임베디드 프로그래밍을 할 때 메모리에 직접 접근하는 경우나 동적 할당된 메모리를 사용하는 경우 필요하다.

*임베디드 프로그래밍?

임베디드 시스템을 제어하기 위한 프로그램으로 오늘날 만드는 거의 모든 생활 기기에서 특정 기능을 제어하기 위해 구현된다. 예로 정수기에서 정수, 냉수 등을 구분해 물이 나오게 한다던지 하는 하드웨어가 있는데 하드웨어를 제어하는 소프트웨어를 만들어내는 일을 임베디드 프로그래밍이라고 한다.

- 예시(포인터를 사용한 두 변수의 값 교환)

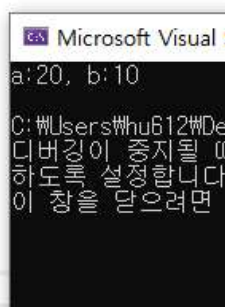
```
void swap(int* pa, int* pb);
```

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

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

    return 0;
}

void swap(int* pa, int* pb)
{
    int temp;
    temp = *pa;
    *pa = *pb;
    *pb = temp;
}
```

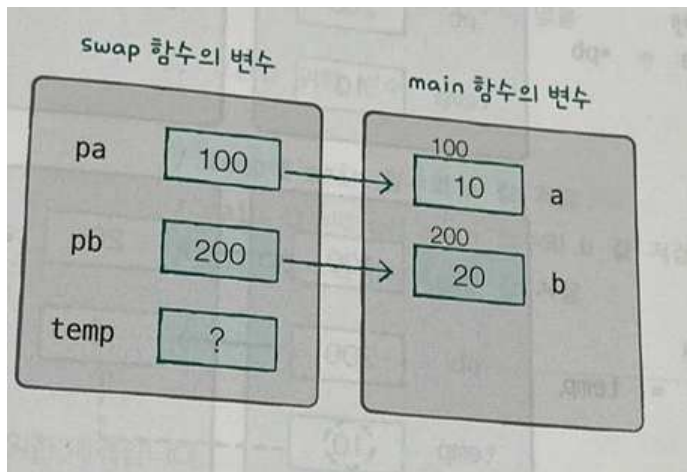


swap 함수 설정

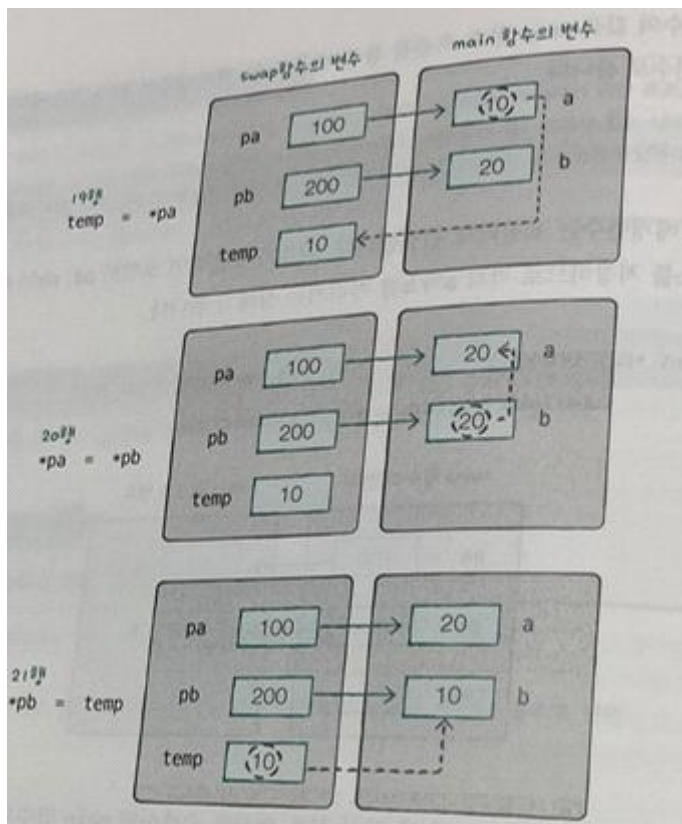
temp에 pa가 가리키는 변수 저장

pa가 가리키는 변수에 pb가 가리키는 변수 저장

pb가 가리키는 변수에 temp 저장



요런 상황!



* 그럼 포인터 없이 두 변수의 값을 바꾸는 함수는 불가능 할까?!

```

void swap(void);

int main(void)
{
    int c=10, d=20;

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

    return 0;
}

void swap(void)
{
    int temp;

    temp = c;
    c = d;
    d = temp;
}

```

2 0		
목록		
솔루션		
오류 목록		
6 오류 0 경고		
오류 목록		
코드	설명	프로젝트
E0020	식별자 "c"이(가) 정의되어 있지 않습니다.	Project1
E0020	식별자 "d"이(가) 정의되어 있지 않습니다.	Project1

함수 안에 선언된 변수명은 사용 범위가 함수 내부로 제한되어 있어서 main 함수에 있는 변수 a,b는 다른 함수인 swap 함수에서 그 이름을 사용할 수 없다. 변수가 선언된 시점부터 선언된 블록 끝까지로 제한된다는 이야기이다. 따라서 할 수 없다.

-예제(main 함수에서 a,b의 값을 swap 함수에 인수로 주는 방법이다.)

```

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;
}

```



swap 안에서 복사본으로 값을 주기 때문에 a,b값에는 변함이 없다.

swap 함수에서 바꾼 값을 main 함수로 변환하는 방법을 생각해볼 수 있는데, 함수는 오직 하나의 값만을 반환가능하며, 한 번의 함수 호출을 통해 두 변수의 값을 바꾸는 것은 불가능하다.

5) 마무리

- 주소와 포인터는 상수와 변수의 차이가 있다.
- 포인터의 크기는 주소의 크기와 같다.
- 포인터에 주소를 저장할 때는 가리키는 자료형이 같아야 한다.
- 포인터의 주요 기능 중 하나는 함수 간에 효과적으로 데이터를 공유하는 것이다.

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);

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

	$p_d = p(x)$	
--	--------------	--

6) 확인 문제