

Lab 2

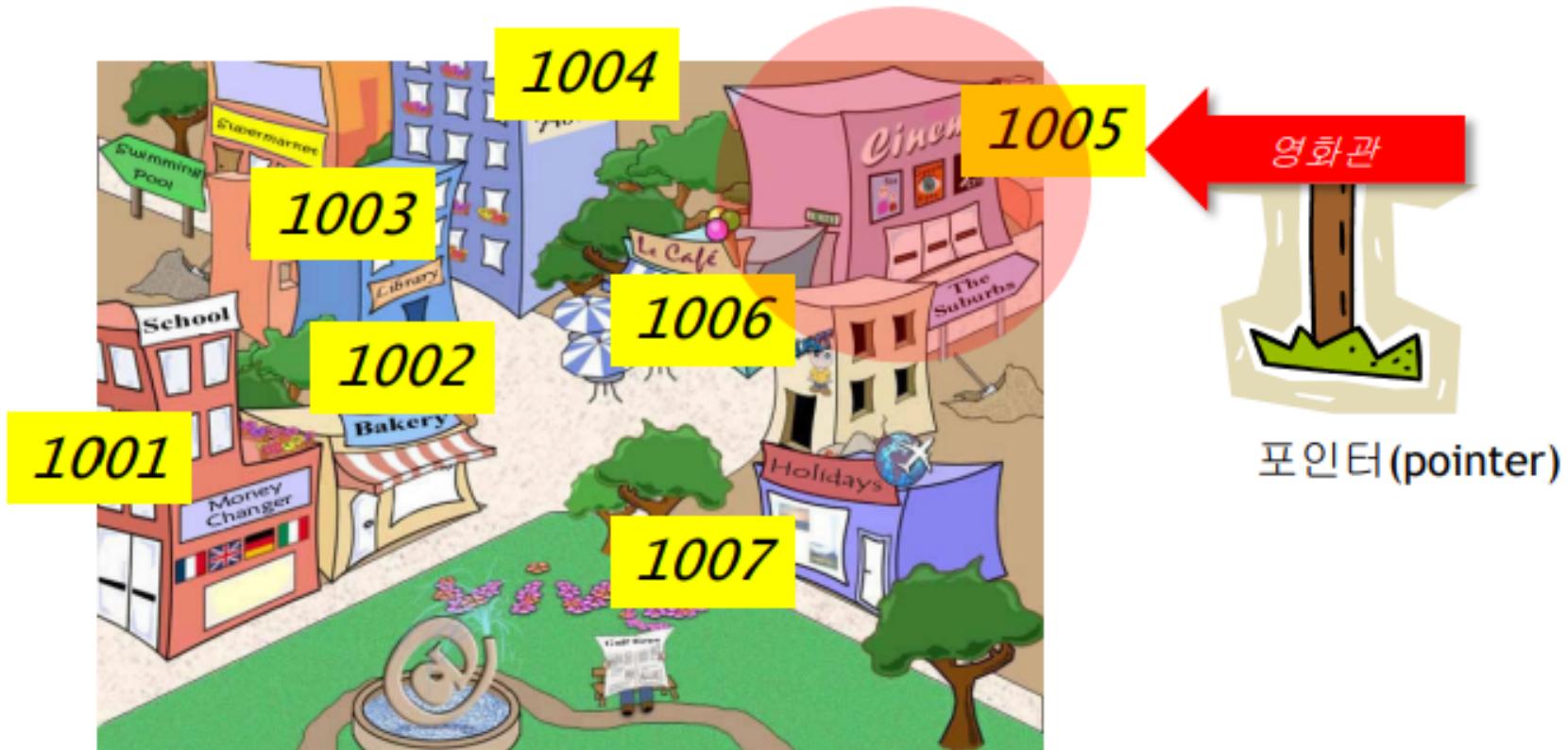
C Programming Review

DCSLAB 안병철 / 2023-03-22

- pointer
- struct
- typedef
- double pointer
- Array of Pointers
- Pointer to an Array
- Function Pointer
- Array of Function Pointers
- const and void pointer

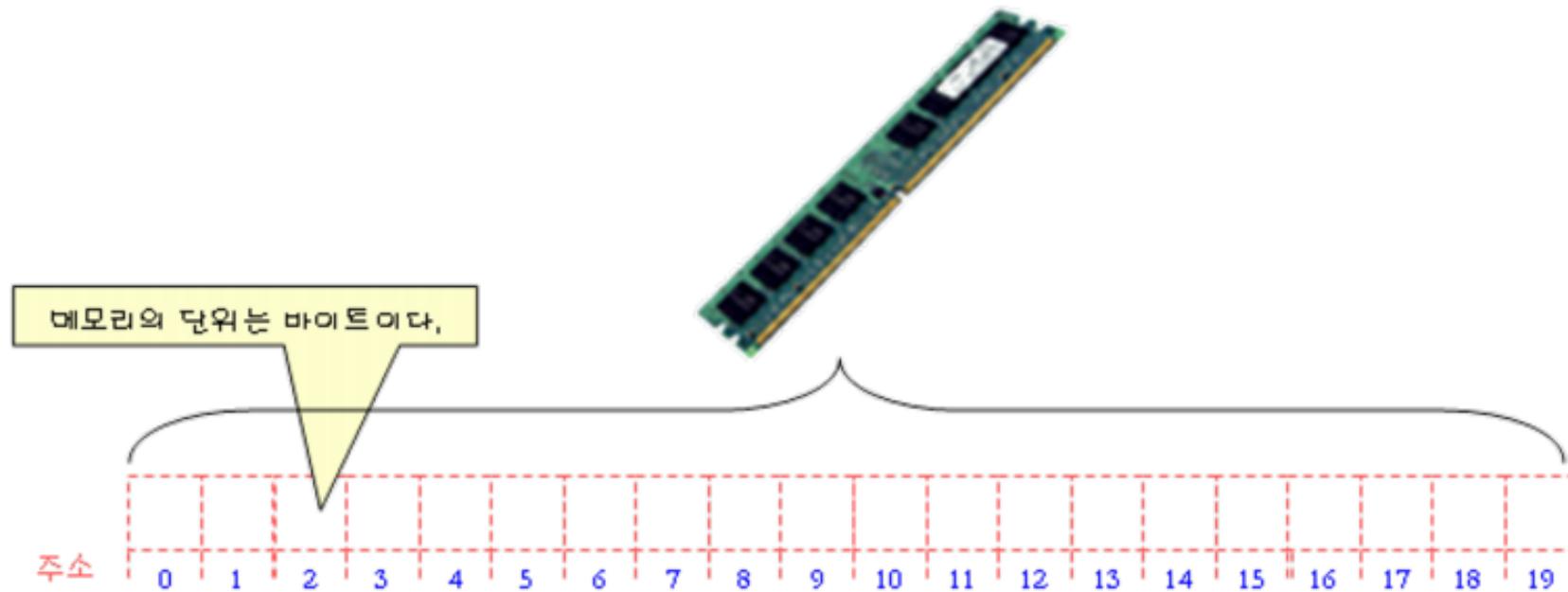
포인터

- 포인터(pointer): 주소를 가지고 있는 변수



메모리의 구조

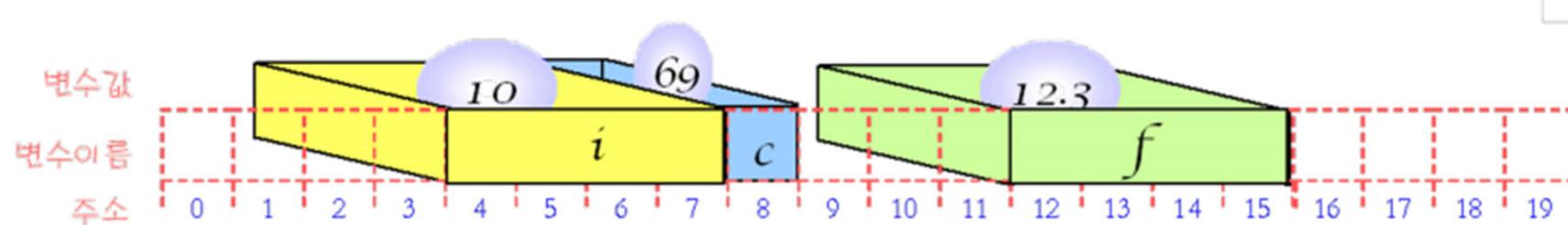
- 변수는 메모리에 저장된다.
- 메모리는 바이트 단위로 액세스된다.
 - 첫번째 바이트의 주소는 0, 두번째 바이트는 1, ...



변수와 메모리

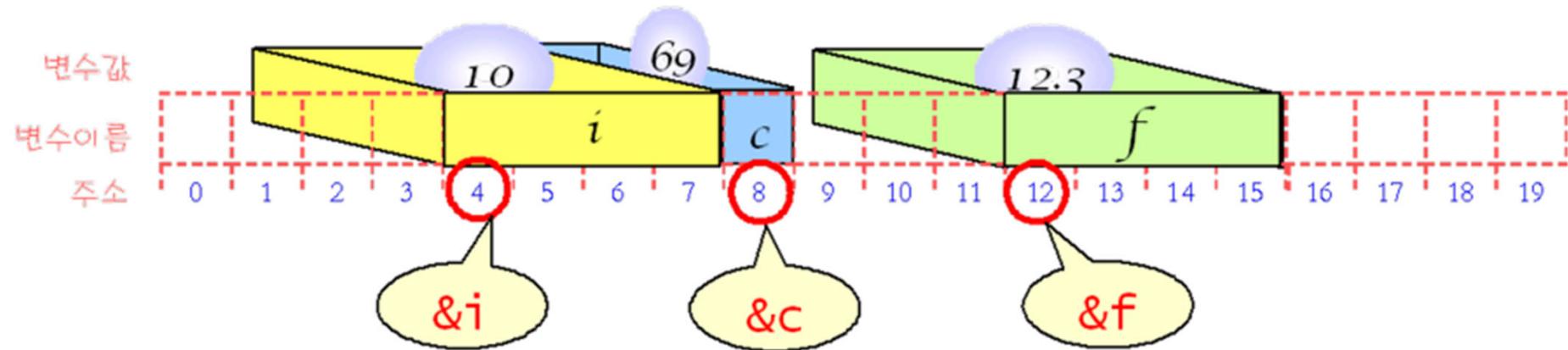
- 변수의 크기에 따라서 차지하는 메모리 공간이 달라진다.
- `char`형 변수: 1바이트, `int`형 변수: 4바이트, ...

```
int main(void)
{
    int i = 10;
    char c = 69;
    float f = 12.3;
}
```



변수의 주소

- 변수의 주소를 계산하는 연산자: &
- 변수 i의 주소: &i



변수의 주소

```
int main(void)
{
    int i = 10;
    char c = 69;
    float f = 12.3;

    printf("i의 주소: %u\n", &i);           // 변수 i의 주소 출력
    printf("c의 주소: %u\n", &c);           // 변수 c의 주소 출력
    printf("f의 주소: %u\n", &f);           // 변수 f의 주소 출력
    return 0;
}
```

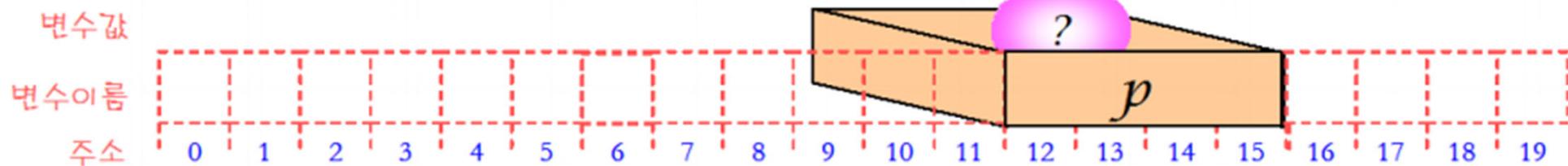
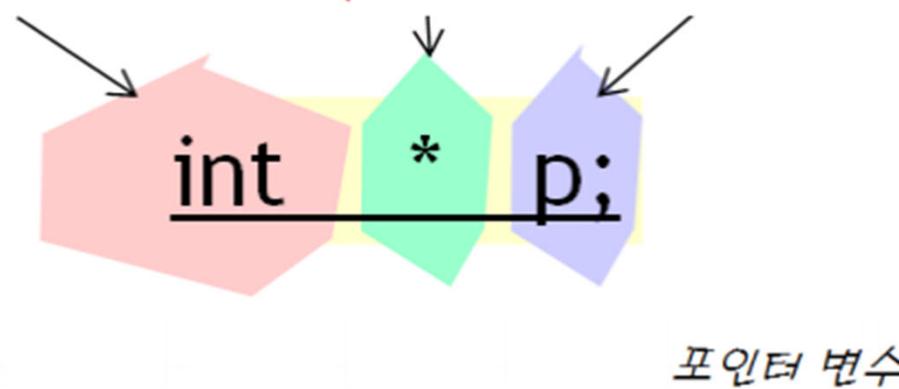
```
i의 주소: 1245024
c의 주소: 1245015
f의 주소: 1245000
```

포인터의 선언

- 포인터: 변수의 주소를 가지고 있는 변수

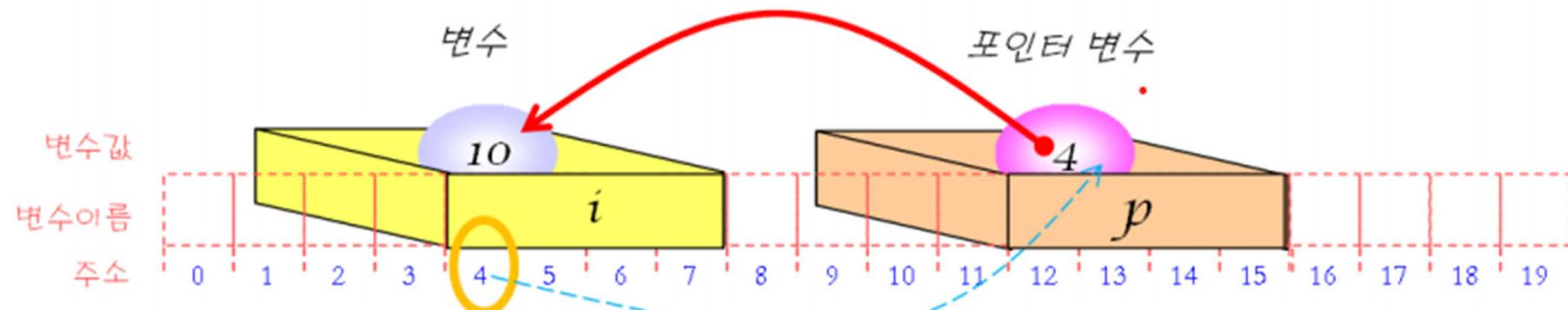
*p가 가리키는 내용은
정수가 된다.

*가 우선순위가 높아서
p는 포인터가 된다.



포인터와 변수의 연결

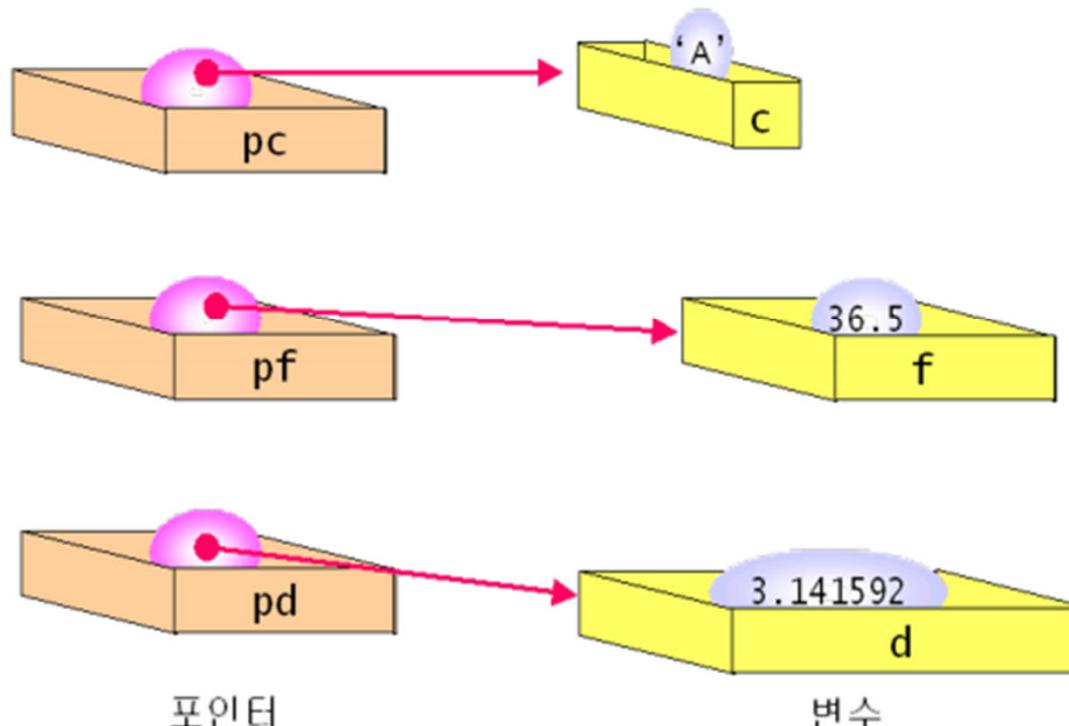
```
int i = 10;           // 정수형 변수 i 선언  
int *p;             // 포인터 변수 p 선언  
p = &i;              // 변수 i의 주소가 포인터 p로 대입
```



다양한 포인터의 선언

```
char c = 'A';           // 문자형 변수 c  
float f = 36.5;        // 실수형 변수 f  
double d = 3.141592;   // 실수형 변수 d
```

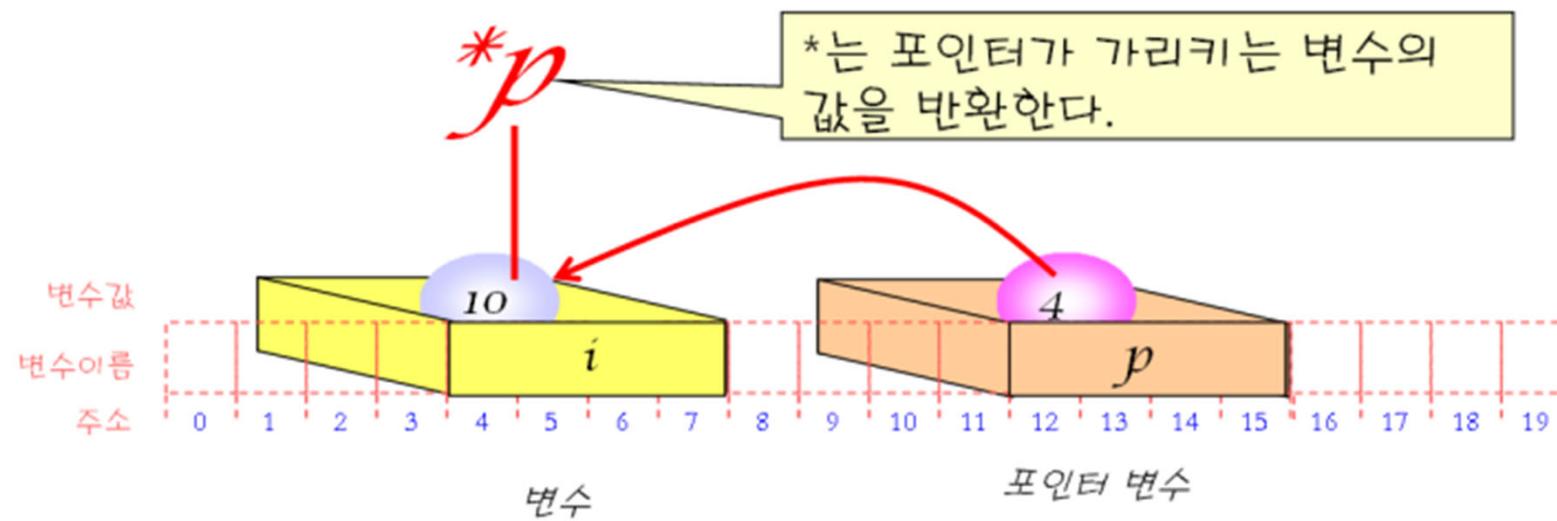
```
char *pc = &c;          // 문자를 가리키는 포인터 pc  
float *pf = &f;         // 실수를 가리키는 포인터 pf  
double *pd = &d;        // 실수를 가리키는 포인터 pd
```



간접 참조 연산자

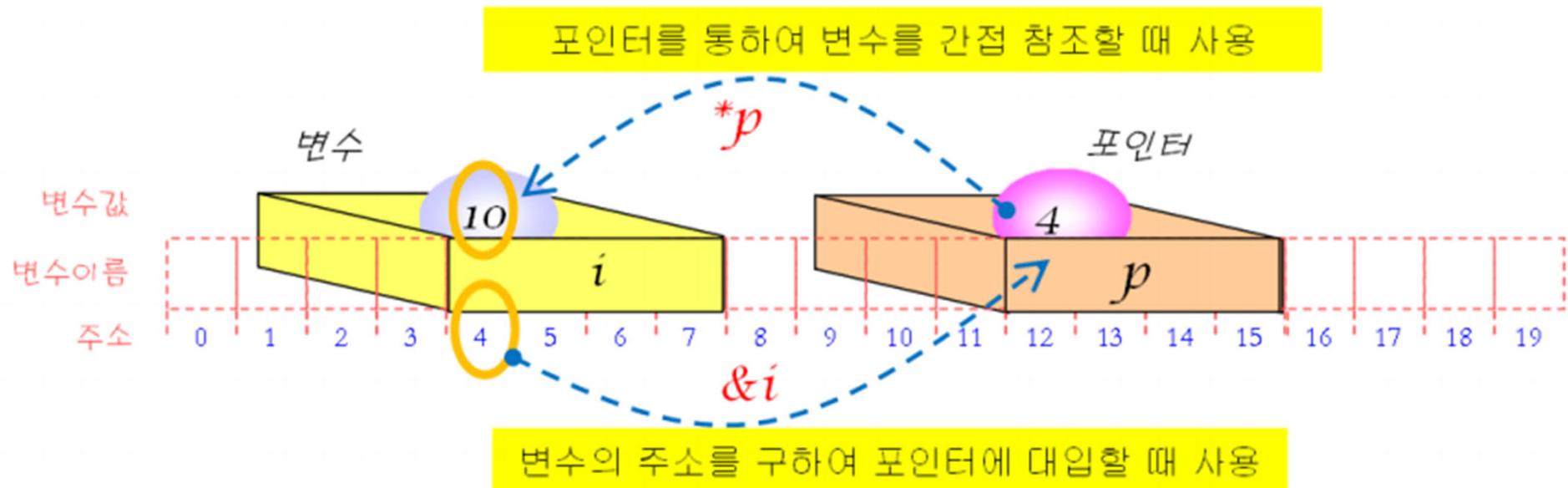
- 간접 참조 연산자 *: 포인터가 가리키는 값을 가져오는 연산자

```
int i=10;  
int *p;  
p =&i;  
printf("%d", *p);
```



& 연산자와 * 연산자

- & 연산자: 변수의 주소를 반환한다
- * 연산자: 포인터가 가리키는 곳의 내용을 반환한다.



포인터 예제 #1

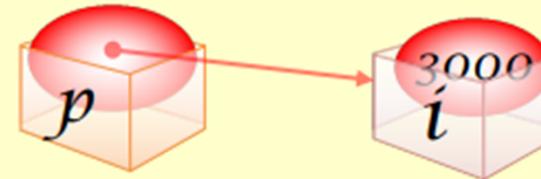
```
#include <stdio.h>

int main(void)
{
    int i = 3000;
    int *p = &i;           // 변수와 포인터 연결

    printf("&i = %u\n", &i); // 변수의 주소 출력
    printf("i = %d\n", i);  // 변수의 값 출력

    printf("*p = %d\n", *p); // 포인터를 통한 간접 참조 값 출력
    printf("p = %u\n", p);  // 포인터의 값 출력

    return 0;
}
```



```
&i = 1245024
i = 3000
*p = 3000
p = 1245024
```

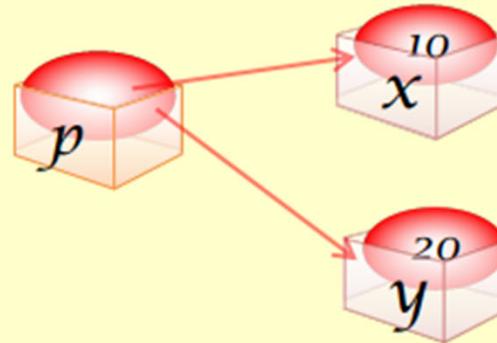
포인터 예제 #2

```
#include <stdio.h>

int main(void)
{
    int x=10, y=20;
    int *p;

    p = &x;
    printf("p = %d\n", p);
    printf("*p = %d\n\n", *p);

    p = &y;
    printf("p = %d\n", p);
    printf("*p = %d\n", *p);
    return 0;
}
```



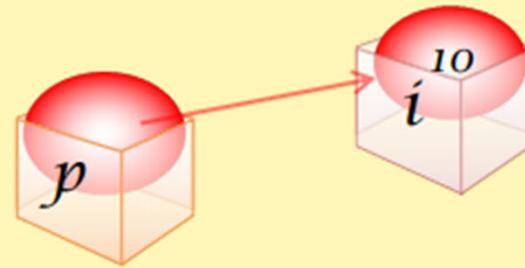
```
p = 1245052
*p = 10
p = 1245048
*p = 20
```

포인터 예제 #3

```
#include <stdio.h>
int main(void)
{
    int i=10;
    int *p;

    p = &i;
    printf("i = %d\n", i);

    *p = 20;
    printf("i = %d\n", i);
    return 0;
}
```



포인터를 통하여 변수의 값을 변경한다.

```
i = 10
i = 20
```

포인터 사용시 주의점

- 초기화가 안된 포인터를 사용하면 안된다.

```
int main(void)
{
    int *p;          // 포인터 p는 초기화가 안되어 있음
    *p = 100;        // 위험한 코드
    return 0;
}
```

OS에서 기본적으로 허가되지 않는 메모리 구역에 대한
참조는 막지만 의도치 않은 데이터를 덮어씌울 수 있음!

포인터 사용시 주의점

- 포인터가 아무것도 가리키고 있지 않는 경우에는 NULL로 초기화
- NULL 포인터를 가지고 간접 참조하면 하드웨어로 감지할 수 있다.
- 포인터의 유효성 여부 판단이 쉽다.

```
int *p = NULL;
```

포인터 사용시 주의점

- 포인터의 타입과 변수의 타입은 일치하여야 한다.

```
#include <stdio.h>

int main(void)
{
    int i;
    double *pd;

    pd = &i; // 오류! double형 포인터에 int형 변수의 주소를 대입
    *pd = 36.5;

    return 0;
}
```

포인터 연산

- 가능한 연산: 증가, 감소, 덧셈, 뺄셈 연산
- 증가 연산의 경우 증가되는 값은 포인터가 가리키는 객체의 크기

포인터 타입	++연산후 증가되는값
char	1
short	2
int	4
float	4
double	8

p++

증가 연산 예제

```
#include <stdio.h>
int main(void)
{
    char *pc;
    int *pi;
    double *pd;

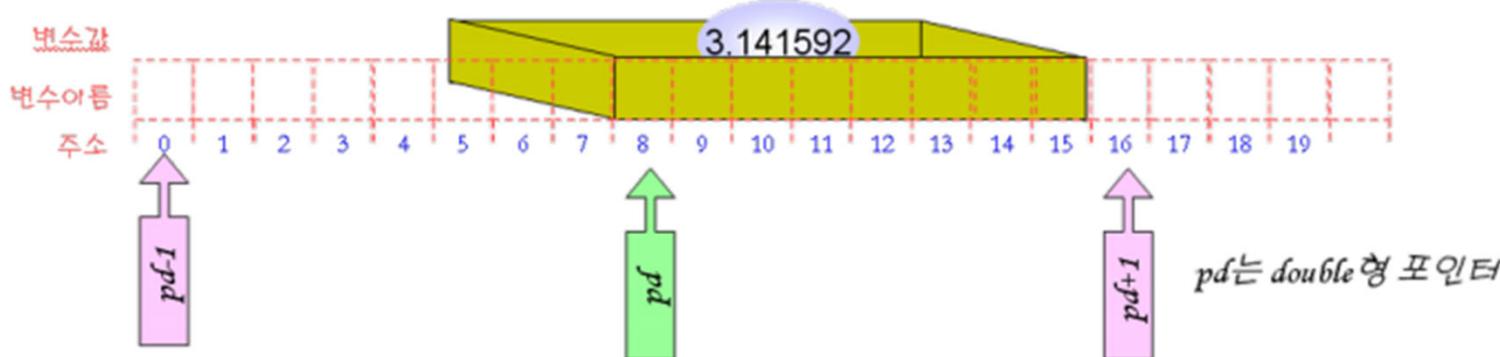
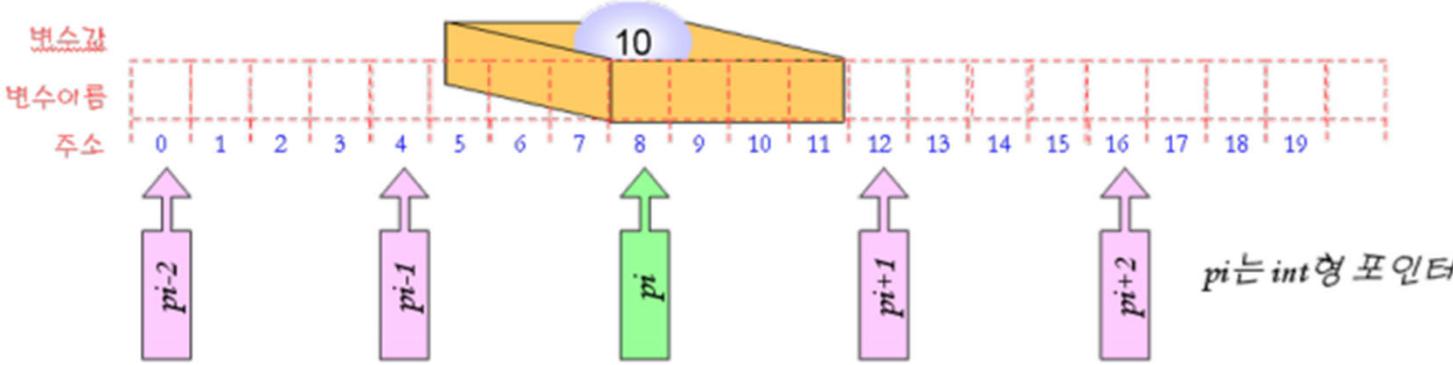
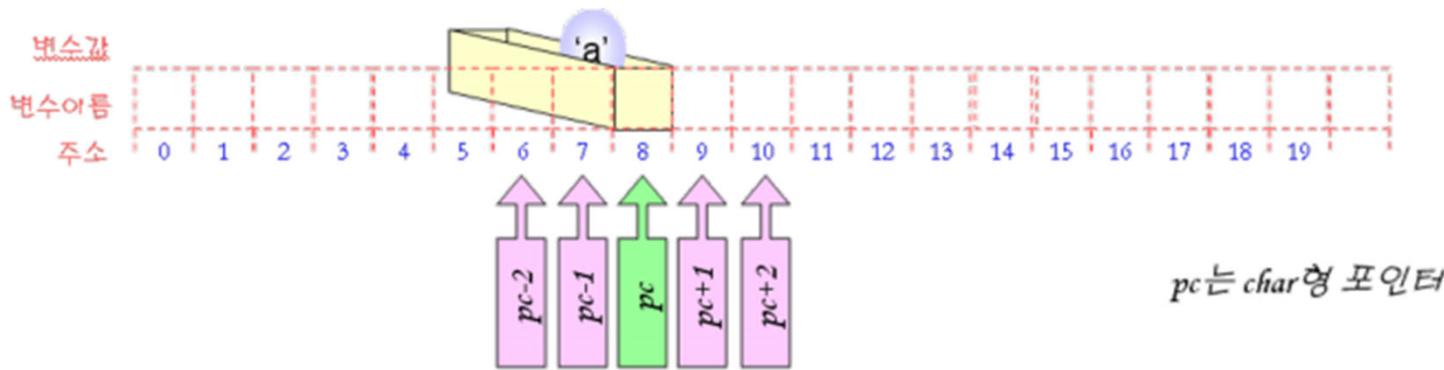
    pc = (char *)10000;
    pi = (int *)10000;
    pd = (double *)10000;
    printf("증가 전 pc = %d, pi = %d, pd = %d\n", pc, pi, pd);

    pc++;
    pi++;
    pd++;
    printf("증가 후 pc = %d, pi = %d, pd = %d\n", pc, pi, pd);
    return 0;
}
```

증가 전 pc = 10000, pi = 10000, pd = 10000

증가 후 pc = 10001, pi = 10004, pd = 10008

포인터의 증감 연산



간접 참조 연산자와 증감 연산자

- $*p++;$
 - p가 가리키는 위치에서 값을 가져온 후에 p를 증가한다.
- $(*p)++;$
 - p가 가리키는 위치의 값을 증가한다.

수식	의미
$v = *p++$	p가 가리키는 값을 v에 대입한 후에 p를 증가한다.
$v = (*p)++$	p가 가리키는 값을 v에 대입한 후에 가리키는 값을 증가한다.
$v = *++p$	p를 증가시킨 후에 p가 가리키는 값을 v에 대입한다.
$v = ++*p$	p가 가리키는 값을 가져온 후에 그 값을 증가하여 v에 대입한다.

간접 참조 연산자와 증감 연산자

```
#include <stdio.h>
```

```
int main(void)
{
```

```
    int i = 10;
    int *pi = &i;
```

```
    printf("i = %d, pi = %p\n", i, pi);
    (*pi)++;
    printf("i = %d, pi = %p\n", i, pi);
```

```
    printf("i = %d, pi = %p\n", i, pi);
    *pi++;
    printf("i = %d, pi = %p\n", i, pi);
```

```
    return 0;
}
```

pi가 가리키는 위치의 값을 증가한다.

pi가 가리키는 위치에서 값을 가져온 후에 pi를 증가한다.

```
i = 10, pi = 0012FF60
i = 11, pi = 0012FF60
i = 11, pi = 0012FF60
i = 11, pi = 0012FF64
```

포인터의 형변환

C언어에서는 꼭 필요한 경우에, 명시적으로 포인터의 타입을 변경할 수 있다.

```
double *pd = &f;  
int *pi;  
pi = (int *)pd;
```

위의 코드는 pd의 주소에 존재하는 double 값을
int형식으로 읽겠다고 선언한 것과 같다

간접 참조 연산자와 증감 연산자

```
#include <stdio.h>
int main(void)
{
    char buffer[8];
    double *pd;
    int *pi;

    pd = (double *)buffer;
    *pd = 3.14;

    printf("%f\n", *pd);
    pi = (int *)buffer;
    *pi = 123;
    *(pi+1) = 456;

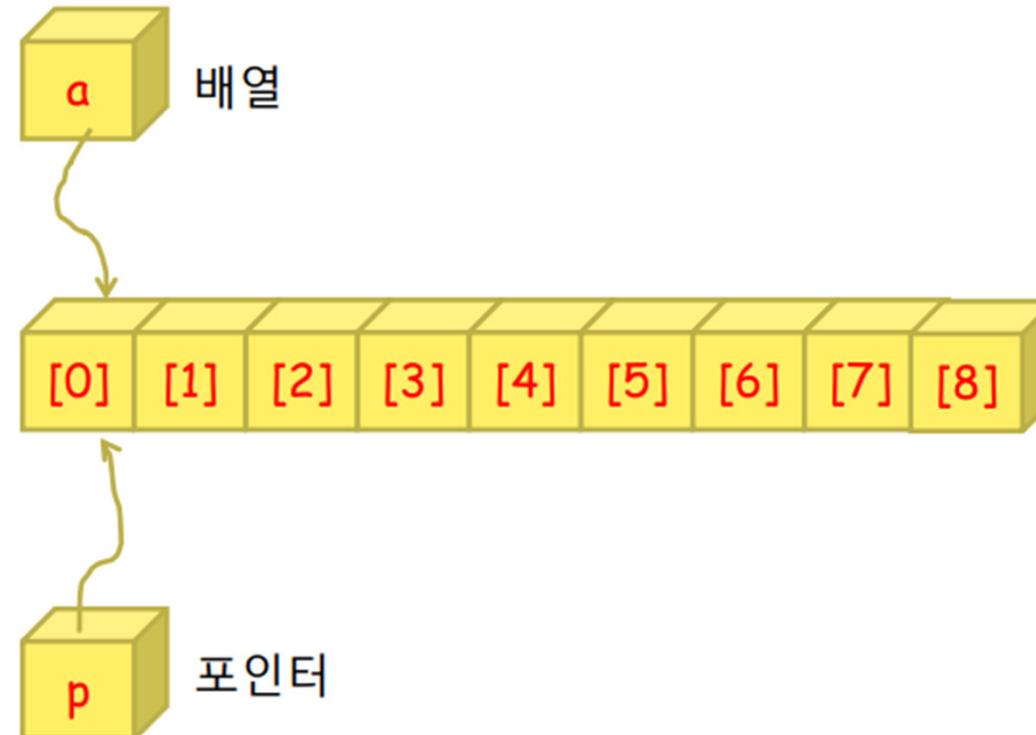
    printf("%d %d\n", *pi, *(pi+1));
    return 0;
}
```

char형 포인터를 double형 포인터로 변환, 배열의 이름은 char형 포인터이다.

char형 포인터를 int형 포인터로
변환

포인터와 배열

- 배열과 포인터는 아주 밀접한 관계를 가지고 있다.
- 배열 이름이 바로 포인터이다.
- 포인터는 배열처럼 사용이 가능하다.



포인터와 배열

// 포인터와 배열의 관계

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    int a[] = { 10, 20, 30, 40, 50 };
```

```
    printf("&a[0] = %u\n", &a[0]);
```

```
    printf("&a[1] = %u\n", &a[1]);
```

```
    printf("&a[2] = %u\n", &a[2]);
```

```
    printf("a = %u\n", a);
```

```
    return 0;
```

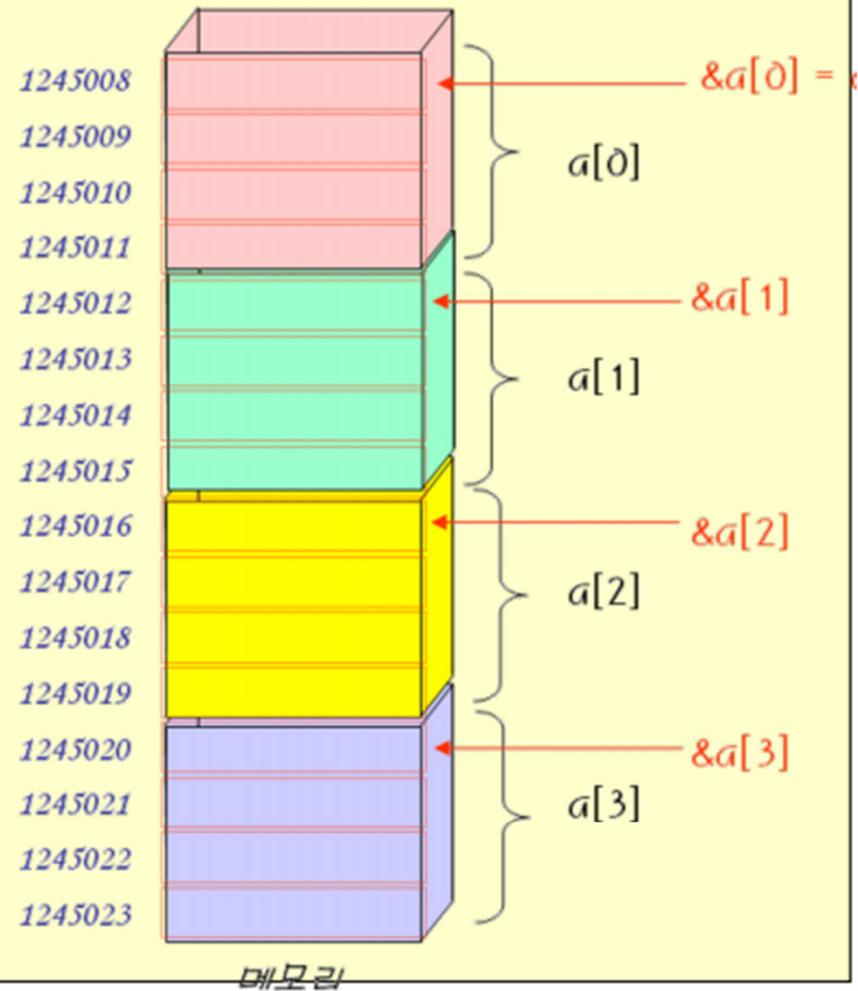
```
}
```

```
&a[0] = 1245008
```

```
&a[1] = 1245012
```

```
&a[2] = 1245016
```

```
a = 1245008
```



포인터와 배열

```
// 포인터와 배열의 관계
#include <stdio.h>
int main(void)
{
    int a[] = { 10, 20, 30, 40, 50 };

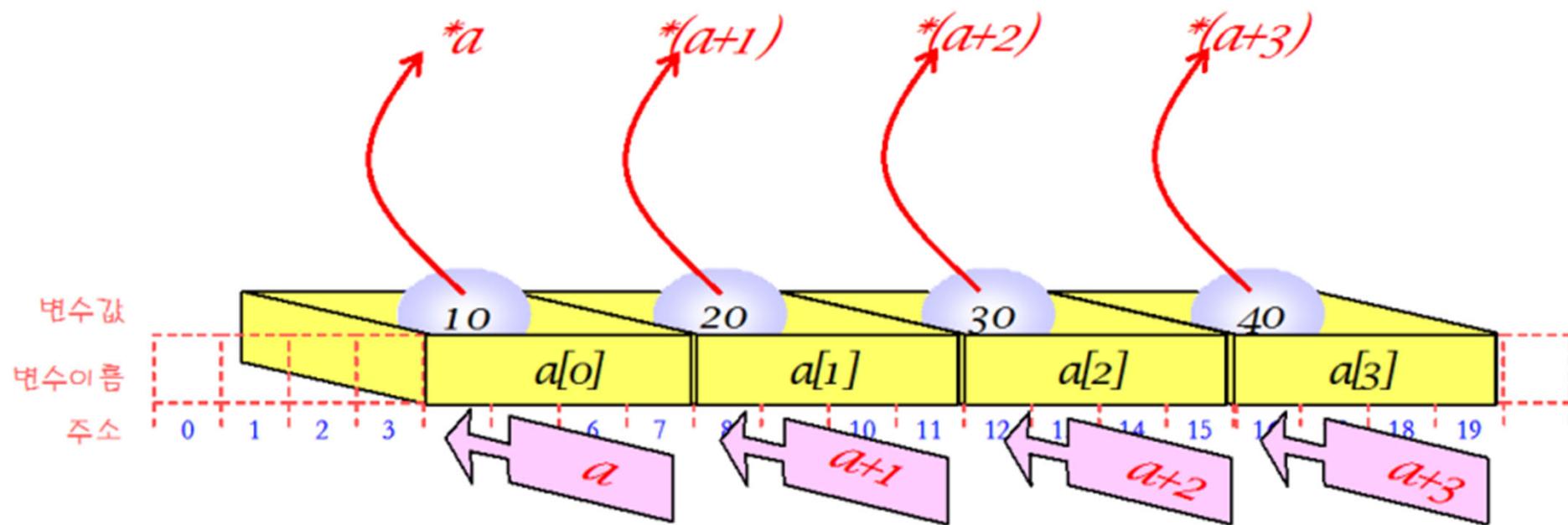
    printf("a = %u\n", a);
    printf("a + 1 = %u\n", a + 1);
    printf("*a = %d\n", *a);
    printf("* (a+1) = %d\n", *(a+1));

    return 0;
}
```

```
a = 1245008
a + 1 = 1245012
*a = 10
*(a+1) = 20
```

포인터와 배열

- 포인터는 배열처럼 사용할 수 있다.
- 인덱스 표기법을 포인터에 사용할 수 있다.



포인터를 배열처럼 사용

```
#include <stdio.h>
int main(void)
{
    int a[] = { 10, 20, 30, 40, 50 };
    int *p;

    p = a;
    printf("a[0]=%d a[1]=%d a[2]=%d \n", a[0], a[1], a[2]);
    printf("p[0]=%d p[1]=%d p[2]=%d \n\n", p[0], p[1], p[2]);

    p[0] = 60;
    p[1] = 70;
    p[2] = 80;

    printf("a[0]=%d a[1]=%d a[2]=%d \n", a[0], a[1], a[2]);
    printf("p[0]=%d p[1]=%d p[2]=%d \n", p[0], p[1], p[2]);
    return 0;
}
```

배열은 결국 포인터로
구현된다는 것을 알 수
있다.

포인터를 통하여 배열
원소를 변경할 수 있다.

a[0]=10 a[1]=20 a[2]=30
p[0]=10 p[1]=20 p[2]=30

a[0]=60 a[1]=70 a[2]=80
p[0]=60 p[1]=70 p[2]=80

포인터를 사용한 방법의 장점

- 포인터가 인덱스 표기법보다 빠르다.
 - Why?: 인덱스를 주소로 변환할 필요가 없다.

```
int get_sum1(int a[], int n)
{
    int i;
    int sum = 0;

    for(i = 0; i < n; i++)
        sum += a[i];
    return sum;
}
```

```
int get_sum2(int a[], int n)
{
    int i, sum =0;
    int *p;

    p = a;
    for(i = 0; i < n; i++)
        sum += *p++;
    return sum;
}
```

배열의 원소를 역순으로 출력

```
#include <stdio.h>
void print_reverse(int a[], int n);

int main(void)
{
    int a[] = { 10, 20, 30, 40, 50 };

    print_reverse(a, 5);
    return 0;
}

void print_reverse(int a[], int n)
{
    int *p = a + n - 1;                      // 마지막 노드를 가리킨다.

    while(p >= a)                          // 첫번째 노드까지 반복
        printf("%d\n", *p--);              // p가 가리키는 위치를 출력하고 감소
}
```

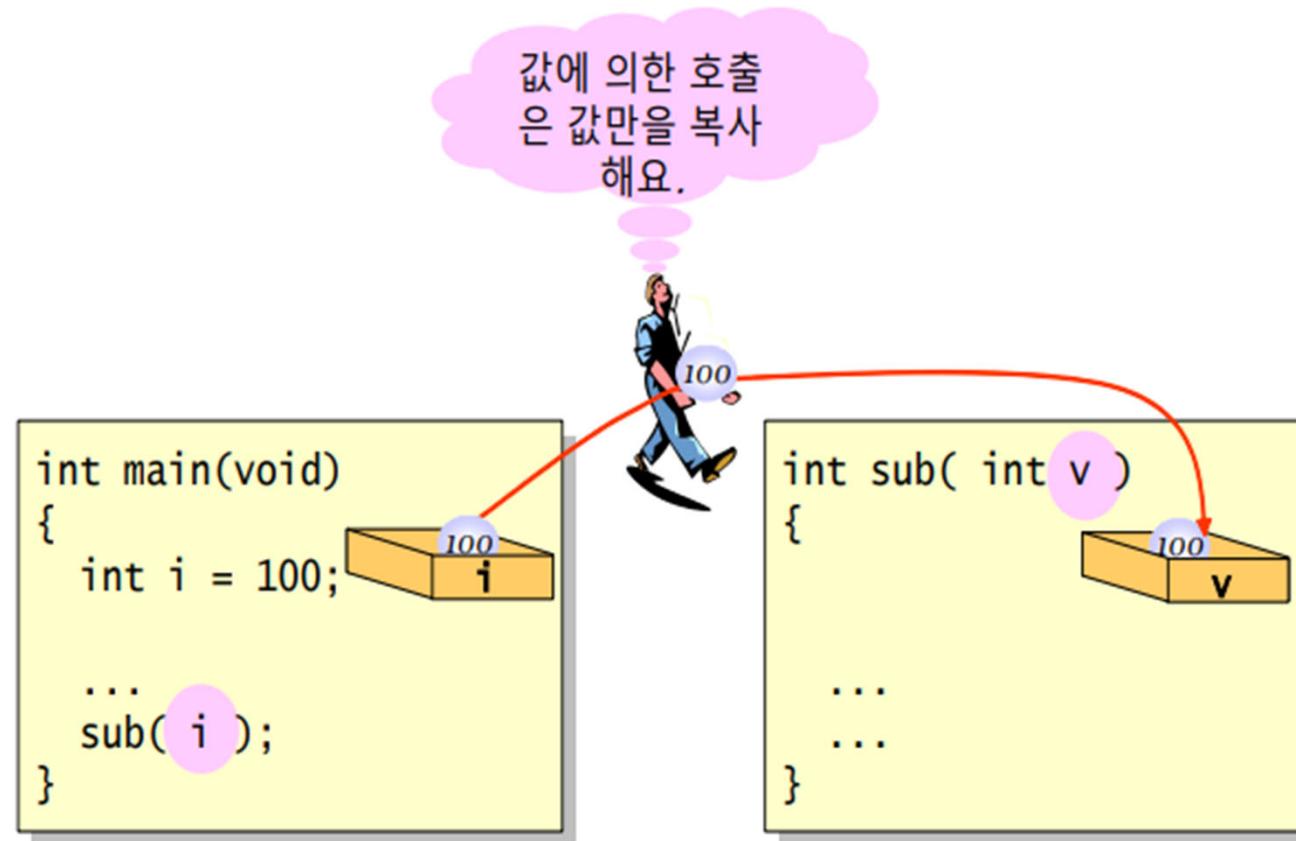
50
40
30
20
10

인수 전달 방법

- 함수 호출 시에 인수 전달 방법
 - 값에 의한 호출(call by value)
 - C에서 기본적인 방법
 - 참조에 의한 호출(call by reference)
 - C에서는 포인터를 이용하여 흉내 낼 수 있다.

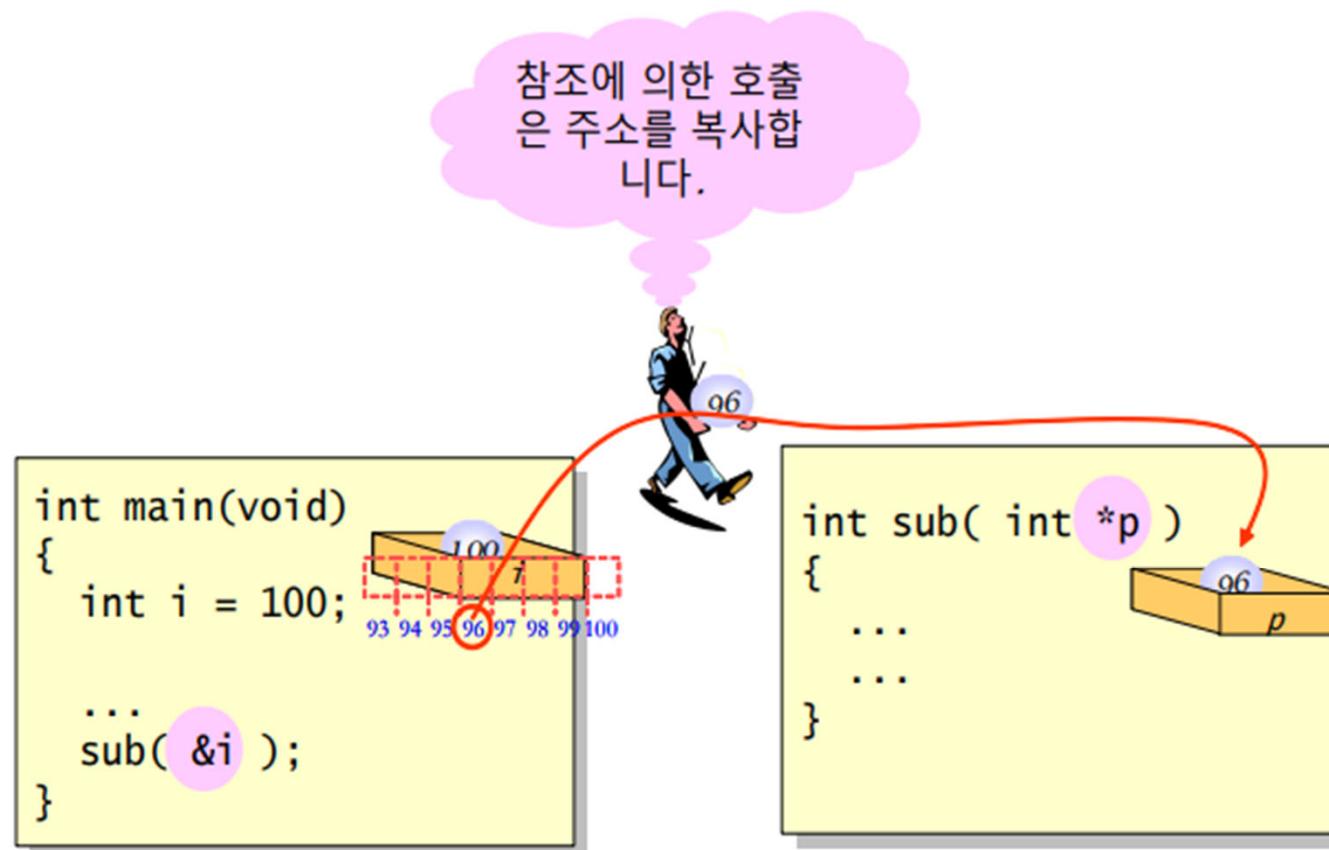
값에 의한 호출

- 함수 호출시에 변수의 값을 함수에 전달



참조에 의한 호출

- 함수 호출시에 변수의 주소를 함수의 매개 변수로 전달



swap() 함수 (call by value)

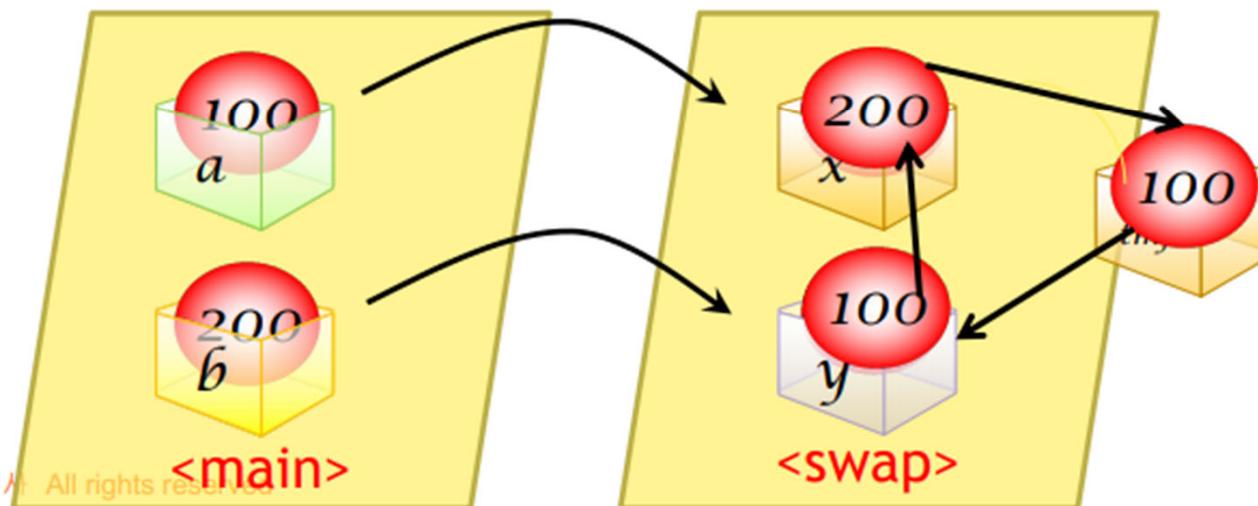
- 변수 2개의 값을 바꾸는 작업을 함수로 작성

```
#include <stdio.h>
void swap(int x, int y);
int main(void)
{
    int a = 100, b = 200;
    swap(a, b);
    return 0;
}
```

```
void swap(int x, int y)
{
    int tmp;

    tmp = x;
    x = y;
    y = tmp;
}
```

함수 호출시에 값만 복사된다.



© 2012 생능출판사 All rights reserved

swap() 함수 (call by reference)

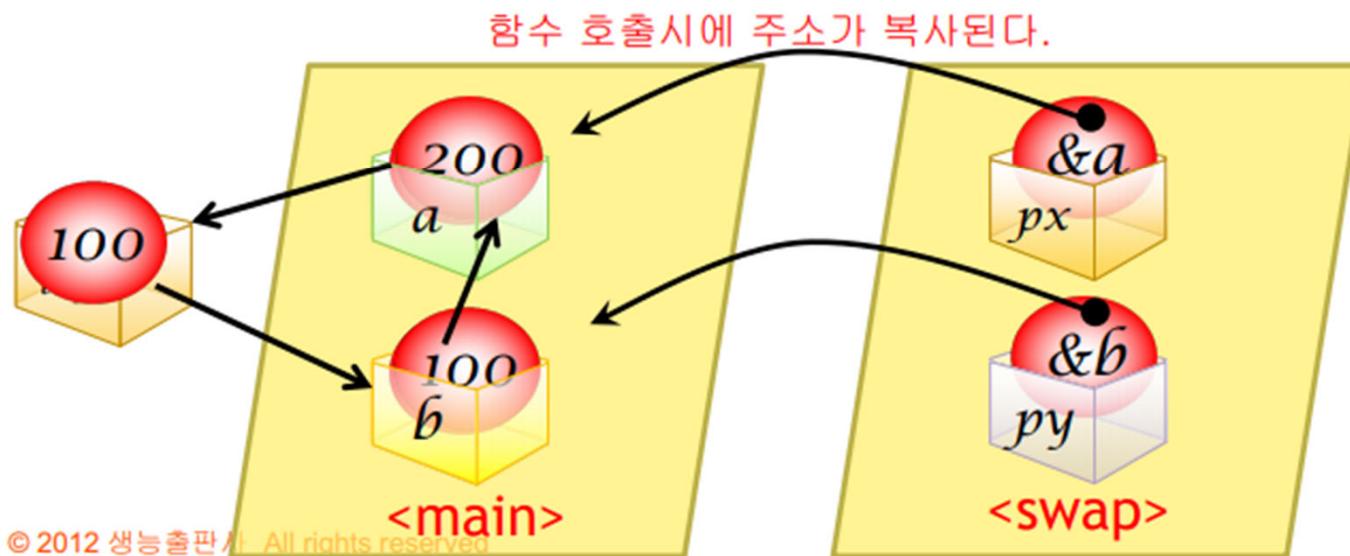
- 포인터를 이용

```
#include <stdio.h>
void swap(int x, int y);
int main(void)
{
    int a = 100, b = 200;
    swap(&a, &b);

    return 0;
}
```

```
void swap(int *px, int *py)
{
    int tmp;

    tmp = *px;
    *px = *py;
    *py = tmp;
}
```



배열 매개 변수

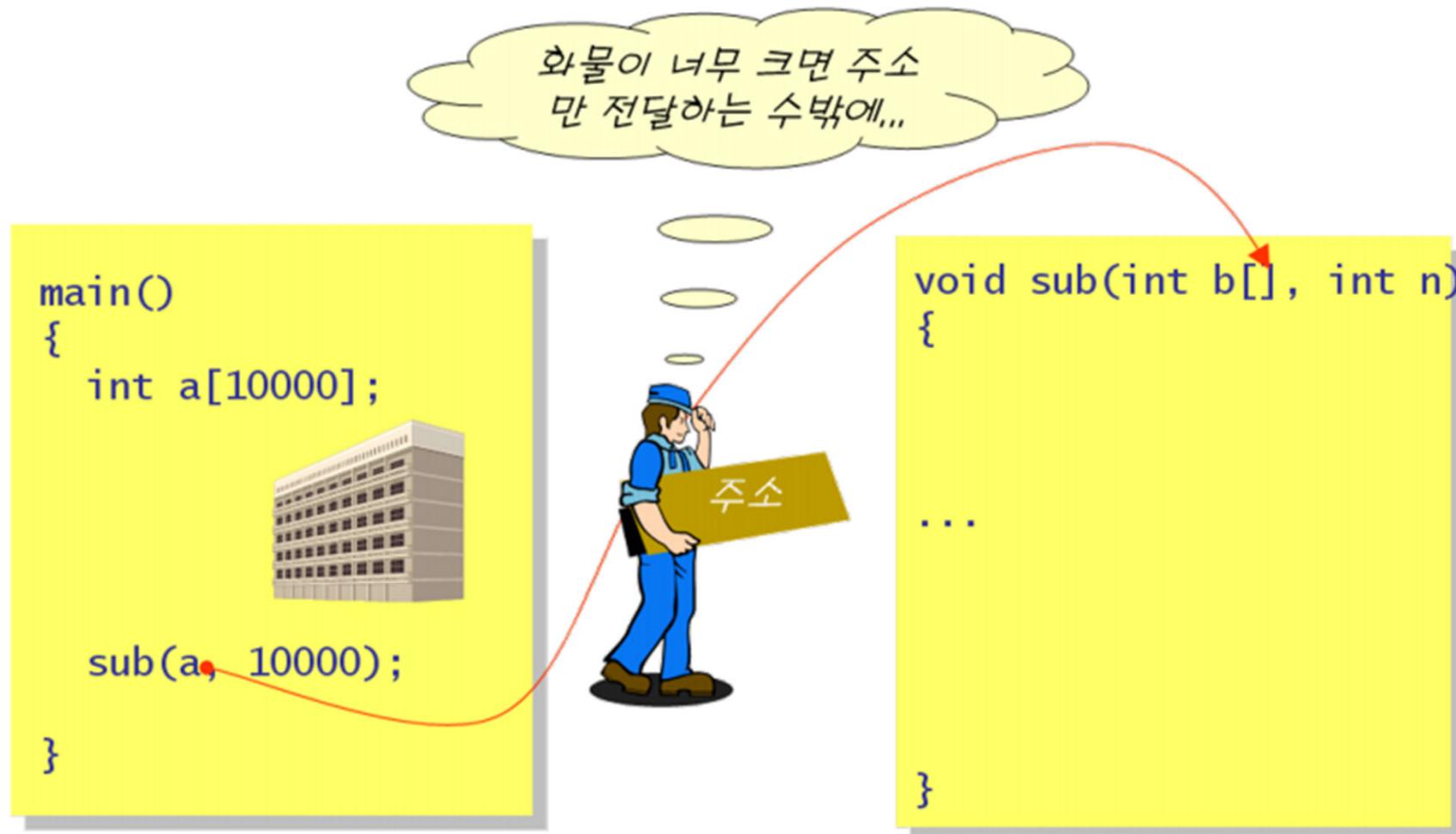
- 일반 매개 변수 vs 배열 매개 변수

```
// 매개 변수 x에 기억 장소가 할당  
void sub(int x)  
{  
    ...  
}
```

```
// b에 기억 장소가 할당되지 않는다.  
void sub( int b[] )  
{  
    ...  
}
```

- Why? -> 배열을 함수로 복사하려면 많은 시간 소모

배열 매개 변수



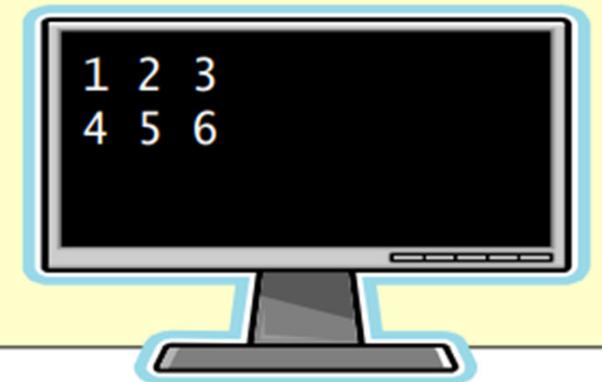
예제

```
#include <stdio.h>
void sub(int b[], int n);

int main(void)
{
    int a[3] = { 1,2,3 };

    printf("%d %d %d\n", a[0], a[1], a[2]);
    sub(a, 3);
    printf("%d %d %d\n", a[0], a[1], a[2]);
    return 0;
}

void sub(int b[], int n)
{
    b[0] = 4;
    b[1] = 5;
    b[2] = 6;
}
```



© 2012 생능출판사 All rights reserved

포인터를 반환할 때 주의점

- 함수가 종료되더라도 남아 있는 변수의 주소를 반환하여야 한다.
- 지역 변수의 주소를 반환하면, 함수가 종료되면 사라지기 때문에 오류

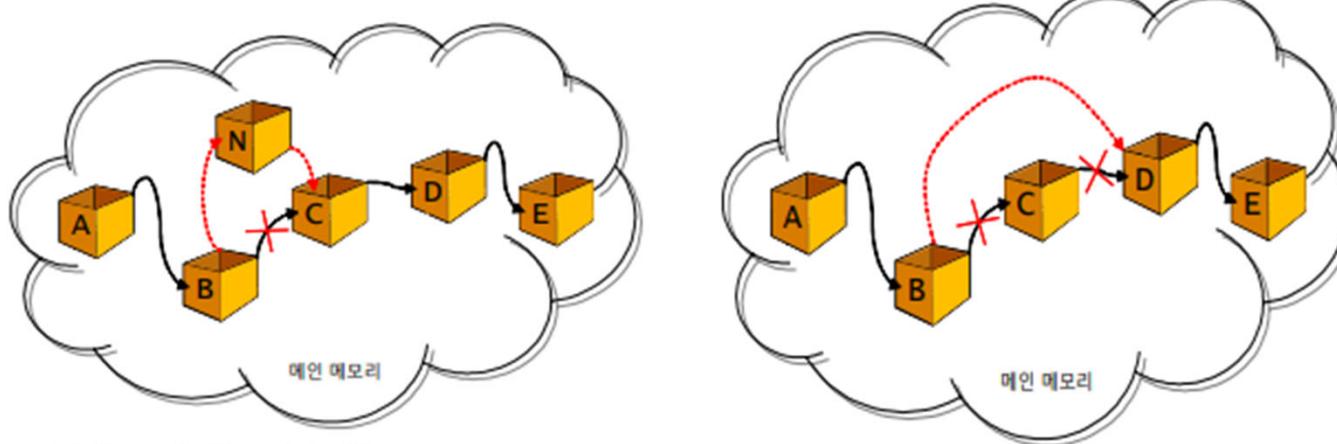
```
int *add(int x, int y)
{
    int result;
    result = x + y;
    return &result;
}
```

지역변수 result는
함수가 종료되면 소멸되므로 그
주소를 반환하면 안된다!



포인터 사용의 장점

- 연결 리스트나 이진 트리 등의 향상된 자료 구조를 만들 수 있다.

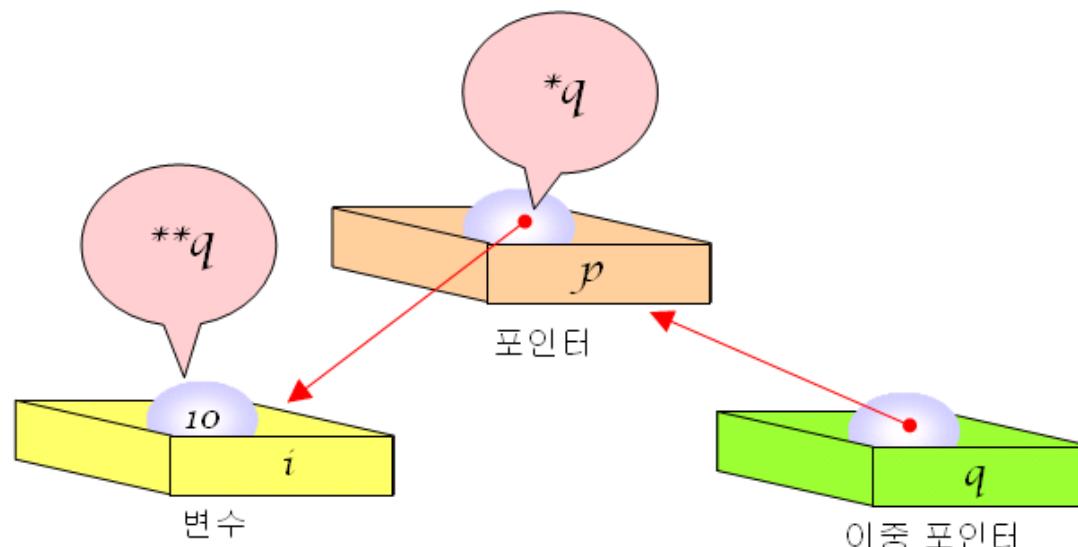


- 참조에 의한 호출
 - 포인터를 매개 변수로 이용하여 함수 외부의 변수의 값을 변경할 수 있다.
- 동적 메모리 할당
 - 17장에서 다룬다.

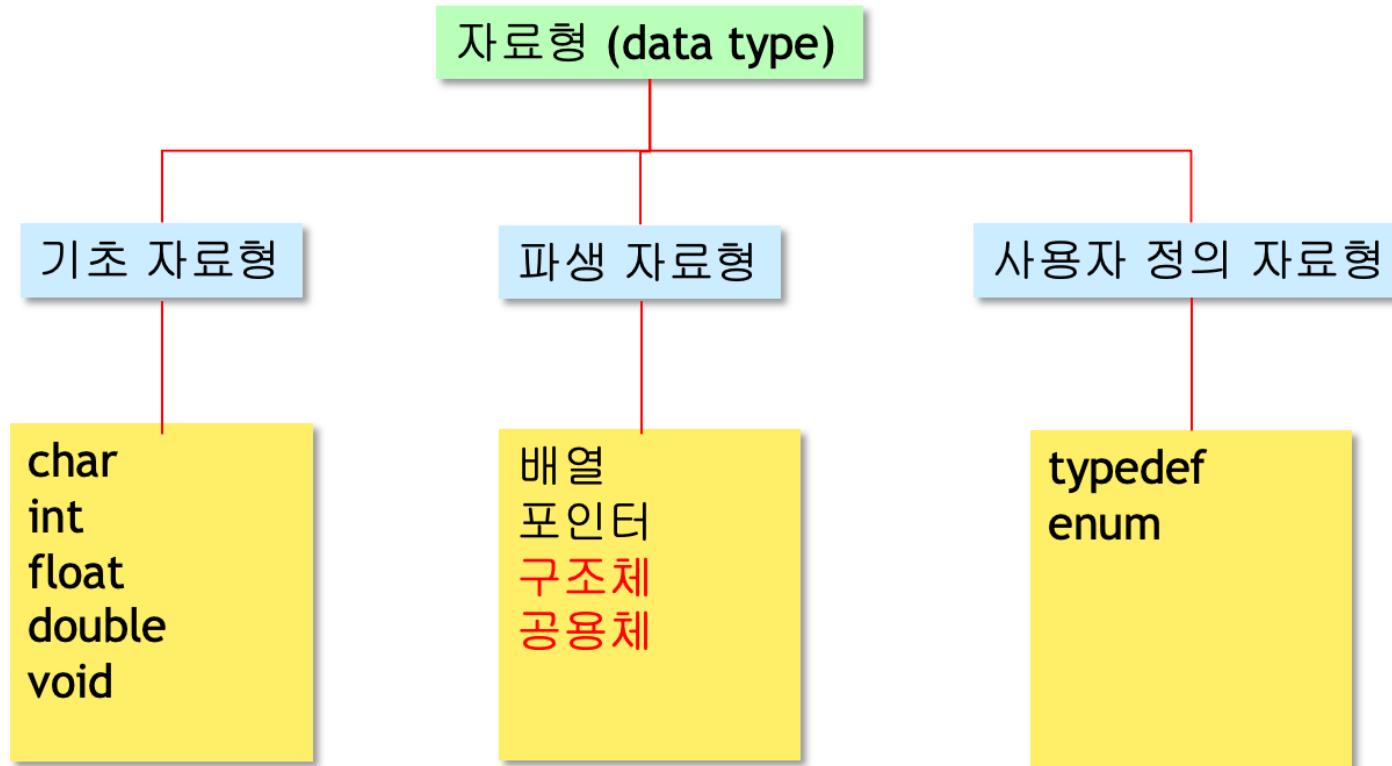
이중 포인터

이중 포인터(double pointer) : 포인터를 가리키는 포인터

```
int i = 10;           // i는 int형 변수  
int *p = &i;         // p는 i를 가리키는 포인터  
int **q = &p;        // q는 포인터 p를 가리키는 이중 포인터
```

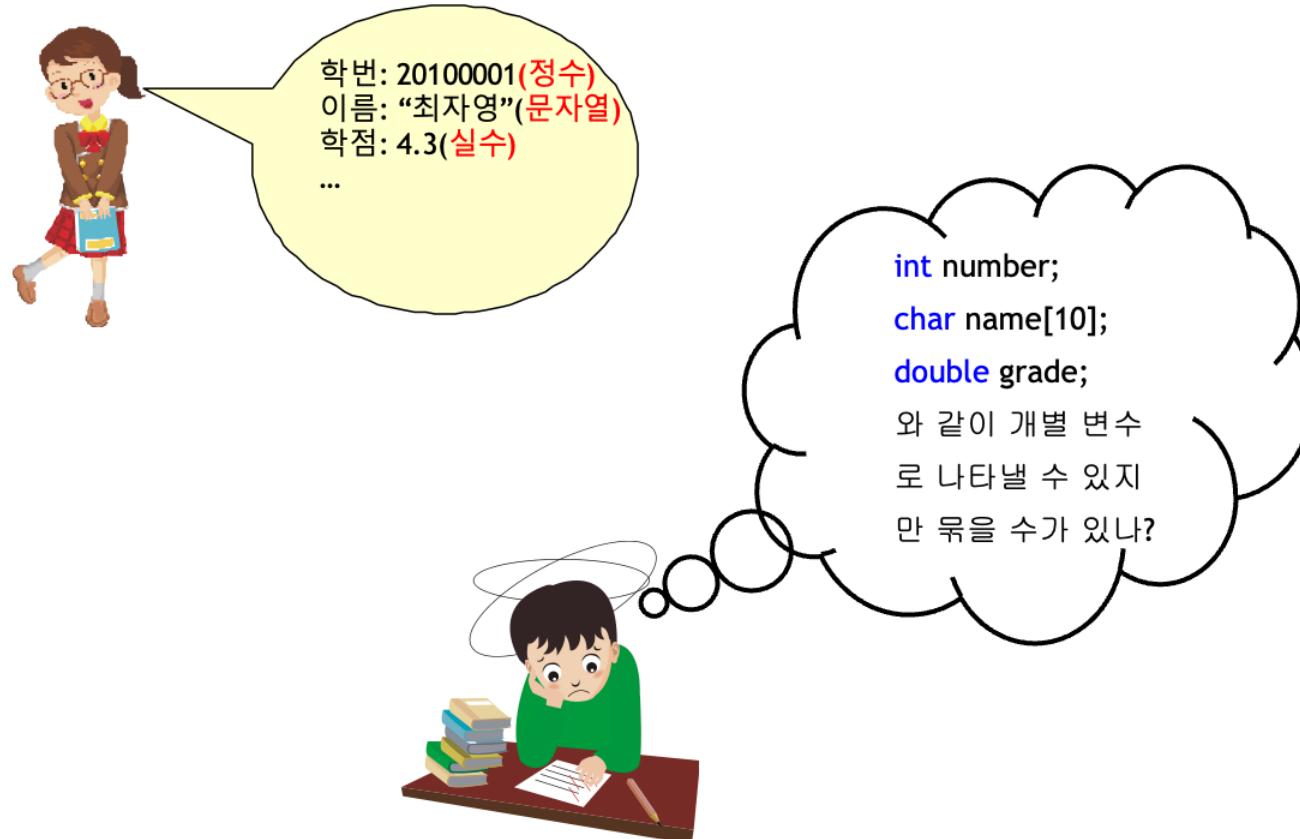


자료형의 분류



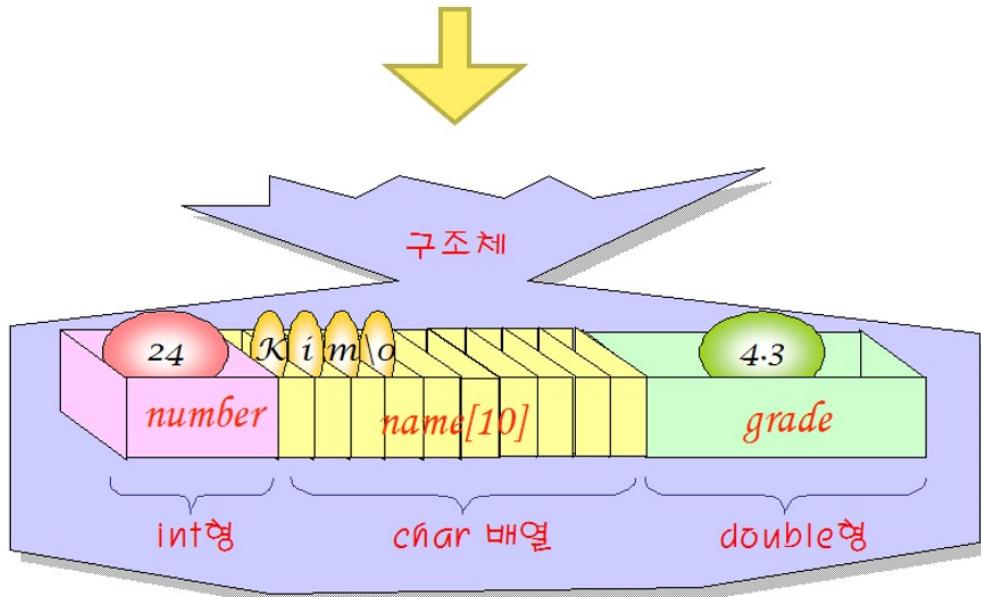
구조체의 필요성

- 학생에 대한 데이터를 하나로 모으려면?



구조체의 필요성 cont'd

```
int number;  
char name[10];  
double grade;
```

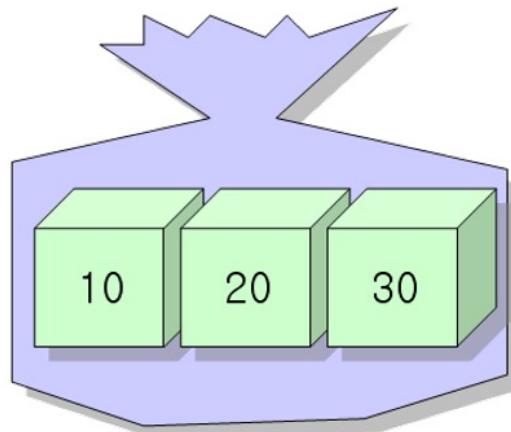


구조체를 사용하면 변수들을 하나로 묶을 수 있습니다.

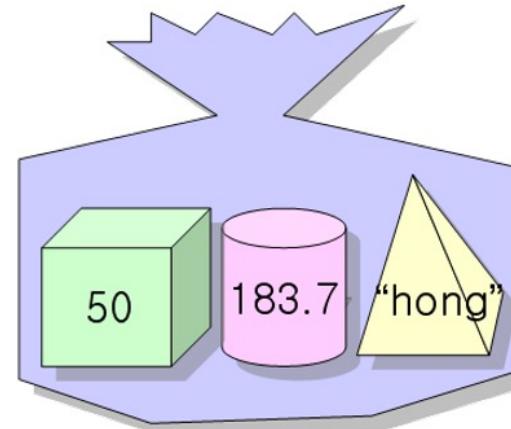


구조체 vs 배열

- 구조체 vs 배열



같은 타입의 집합



다른 타입의 집합

구조체의 선언

- 구조체 선언은 변수 선언은 아님

구조체를 정의하는 것은 와플이나 봉어빵을 만드는 틀을 정의하는 것과 같다.

와플이나 봉어빵을 실제로 만들릭 위해서는 구조체 변수를 선언하여야 한다.



구조체



구조체 변수

구조체 선언의 예시

```
// x값과 y값으로 이루어지는 화면의 좌표
struct point {
    int x;           // x 좌표
    int y;           // y 좌표
};
```

```
// 복소수
struct complex {
    double real;    // 실수부
    double imag;   // 허수부
};
```

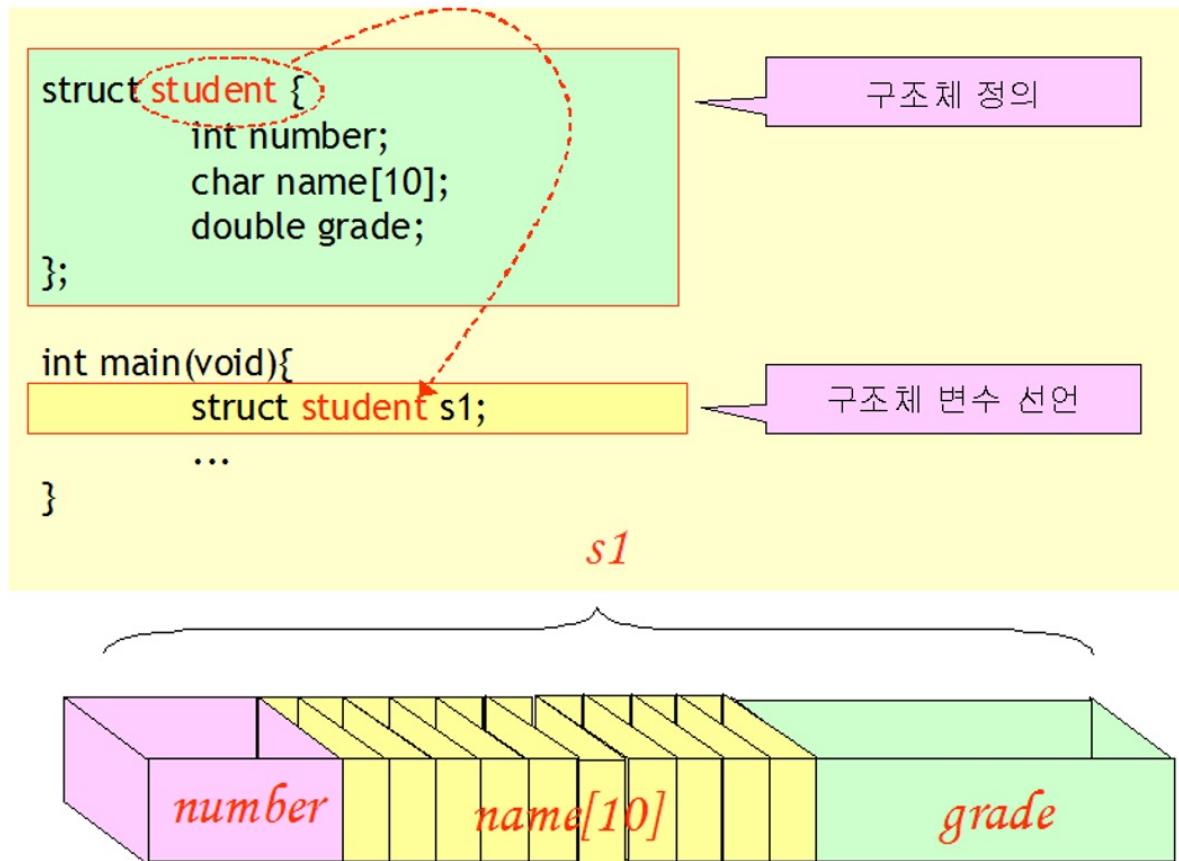
```
// 날짜
struct date {
    int month;
    int day;
    int year;
};
```

```
// 사각형
struct rect {
    int x;
    int y;
    int width;
    int grade;
};
```

```
// 직원
struct employee {
    char name[20]; // 이름
    int age;        // 나이
    int gender;     // 성별
    int salary;     // 월급
};
```

구조체 변수 선언

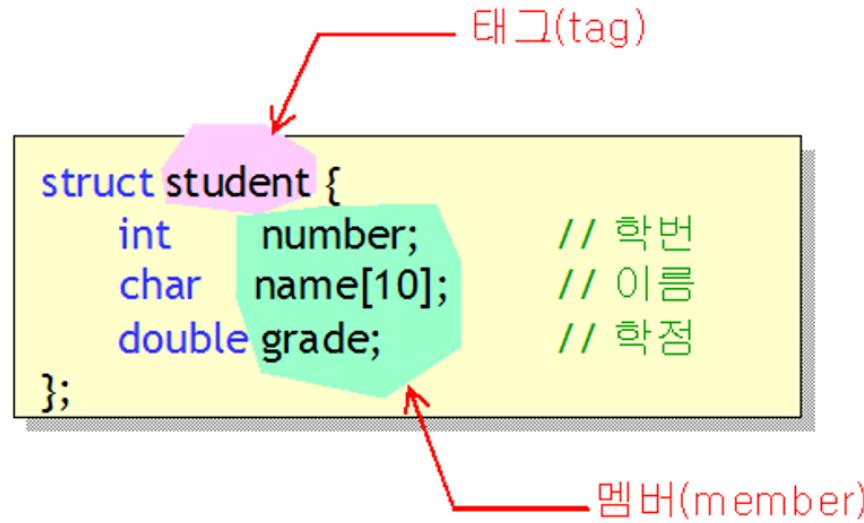
- 구조체 정의와 구조체 변수 선언은 다르다.



구조체의 선언 cont'd

- 구조체 선언 형식

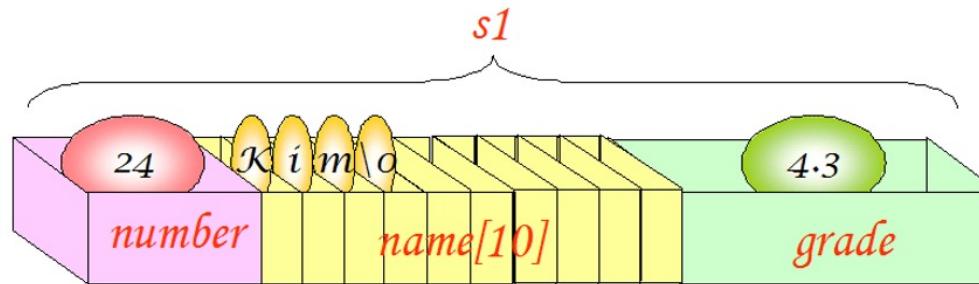
```
struct 태그 {  
    자료형      멤버1;  
    자료형      멤버2;  
    ...  
};
```



구조체 변수 초기화

- 중괄호를 이용하여 초기값을 나열한다.

```
struct student {  
    int number;  
    char name[10];  
    double grade;  
};  
struct student s1 = { 24, "Kim", 4.3 };
```



구조체 멤버 참조

- 구조체 멤버를 참조하려면 다음과 같이 .연산자를 사용한다.

```
s1.number = 26;           // 정수 멤버  
strcpy(s1.name, "Kim");  // 문자열 멤버  
s1.grade = 4.3;          // 실수 멤버
```



구조체 예제 (1)



```
...
struct student {
    int number;
    char name[10];
    double grade;
};
```

구조체 선언

```
int main(void)
```

```
{
```

```
    struct student s;
```

구조체 변수 선언

```
s.number = 20070001;
strcpy(s.name,"홍길동");
s.grade = 4.3;
```

구조체 멤버 참조

```
printf("학번: %d\n", s.number);
printf("이름: %s\n", s.name);
printf("학점: %f\n", s.grade);
return 0;
}
```



```
학번: 20070001
이름: 홍길동
학점: 4.300000
```

구조체 예제 (2)

```
struct student {  
    int number;  
    char name[10];  
    double grade;  
};
```

구조체 선언



```
학번을 입력하시오: 20070001  
이름을 입력하시오: 홍길동  
학점을 입력하시오(실수): 4.3  
학번: 20070001  
이름: 홍길동  
학점: 4.300000
```

```
int main(void)
```

```
{
```

```
    struct student s;
```

구조체 변수 선언

```
    printf("학번을 입력하시오: ");
```

구조체 멤버의 주소 전달

```
    scanf("%d", &s.number);
```

```
    printf("이름을 입력하시오: ");
```

```
    scanf("%s", s.name);
```

```
    printf("학점을 입력하시오(실수): ");
```

```
    scanf("%lf", &s.grade);
```

```
    printf("학번: %d\n", s.number);
```

```
    printf("이름: %s\n", s.name);
```

```
    printf("학점: %f\n", s.grade);
```

```
    return 0;
```

```
}
```

다른 구조체를 멤버로 가지는 구조체

```
struct date { // 구조체 선언
    int year;
    int month;
    int day;
};
```

```
struct student { // 구조체 선언
    int number;
    char name[10];
    • struct date dob; // 구조체 안에 구조체 포함
    double grade;
};
struct student s1; // 구조체 변수 선언
```

```
s1.dob.year = 1983; // 멤버 참조
s1.dob.month = 03;
s1.dob.day = 29;
```

구조체 예제 (3)

```
#include <stdio.h>

struct point {
    int x;
    int y;
};

struct rect {
    struct point p1;
    struct point p2;
};

int main(void)
{
    struct rect r;
    int w, h, area, peri;
```



구조체 예제 (3) cont'd

```
printf("왼쪽 상단의 좌표를 입력하시오: ");
scanf("%d %d", &r.p1.x, &r.p1.y);

printf("오른쪽 상단의 좌표를 입력하시오: ");
scanf("%d %d", &r.p2.x, &r.p2.y);

w = r.p2.x - r.p1.x;
h = r.p2.y - r.p1.y;

area = w * h;
peri = 2 * w + 2 * h;
printf("면적은 %d이고 둘레는 %d입니다.\n", area, peri);

return 0;
}
```



왼쪽 상단의 좌표를 입력하시오: 11
오른쪽 상단의 좌표를 입력하시오: 6 6
면적은 25이고 둘레는 20입니다.



구조체 변수의 대입과 비교

- 같은 구조체 변수끼리 대입은 가능하지만 비교는 불가능하다.

```
struct point {
    int x;
    int y;
};

int main(void)
{
    struct point p1 = {10, 20};
    struct point p2 = {30, 40};

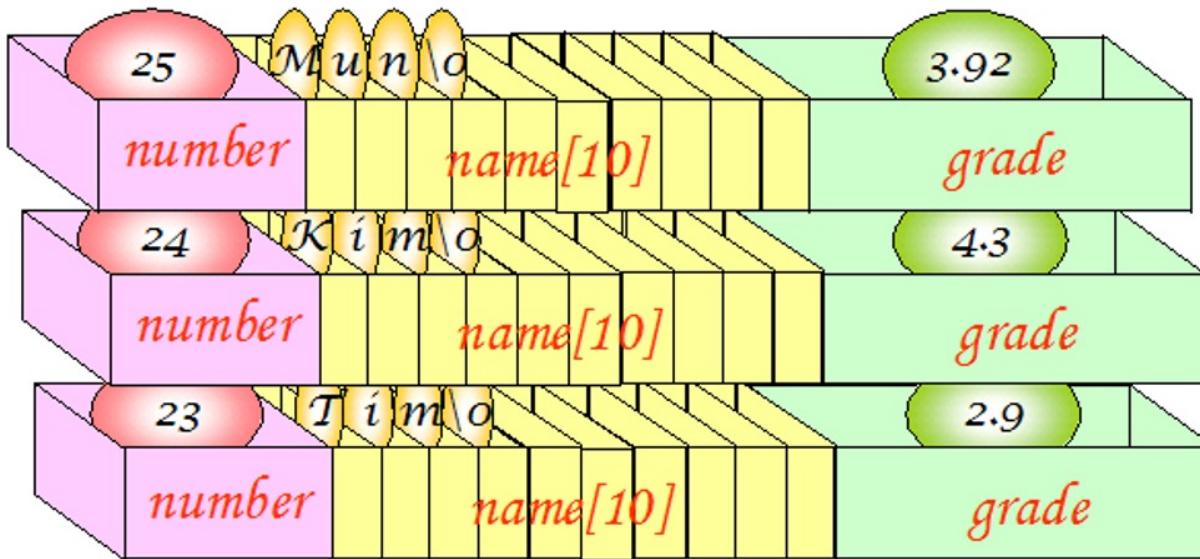
    p2 = p1;                                // 대입 가능

    if( p1 == p2 )                          // 비교 -> 컴파일 오류!!
        printf("p1와 p2이 같습니다.")

    if( (p1.x == p2.x) && (p1.y == p2.y) )
        printf("p1와 p2이 같습니다.")
}
```

구조체 배열

- 구조체를 여러 개 모은 것



구조체 배열의 선언

- 구조체 배열의 선언

```
struct student {
    int number;
    char name[20];
    double grade;
};

int main(void)
{
    struct student list[100]; // 구조체의 배열 선언

    list[2].number = 27;
    strcpy(list[2].name, "홍길동");
    list[2].grade = 178.0;
}
```

구조체 배열의 초기화

- 구조체 배열의 초기화

```
struct student list[3] = {  
    { 1, "Park", 172.8 },  
    { 2, "Kim", 179.2 },  
    { 3, "Lee", 180.3 }  
};
```

구조체 배열 예제

```
#define SIZE 3

struct student {
    int number;
    char name[20];
    double grade;
};

int main(void)
{
    struct student list[SIZE];
    int i;

    for(i = 0; i < SIZE; i++)
    {
        printf("학번을 입력하시오: ");
        scanf("%d", &list[i].number);
        printf("이름을 입력하시오: ");
        scanf("%s", list[i].name);
        printf("학점을 입력하시오(실수): ");
        scanf("%lf", &list[i].grade);
    }

    for(i = 0; i < SIZE; i++)
        printf("학번: %d, 이름: %s, 학점: %f\n", list[i].number, list[i].name, list[i].grade);
    return 0;
}
```



학번을 입력하시오: 20070001
이름을 입력하시오: 홍길동
학점을 입력하시오(실수): 4.3
학번을 입력하시오: 20070002
이름을 입력하시오: 김유신
학점을 입력하시오(실수): 3.92
학번을 입력하시오: 20070003
이름을 입력하시오: 이성계
학점을 입력하시오(실수): 2.87
학번: 20070001, 이름: 홍길동, 학점: 4.300000
학번: 20070002, 이름: 김유신, 학점: 3.920000
학번: 20070003, 이름: 이성계, 학점: 2.870000

구조체와 포인터



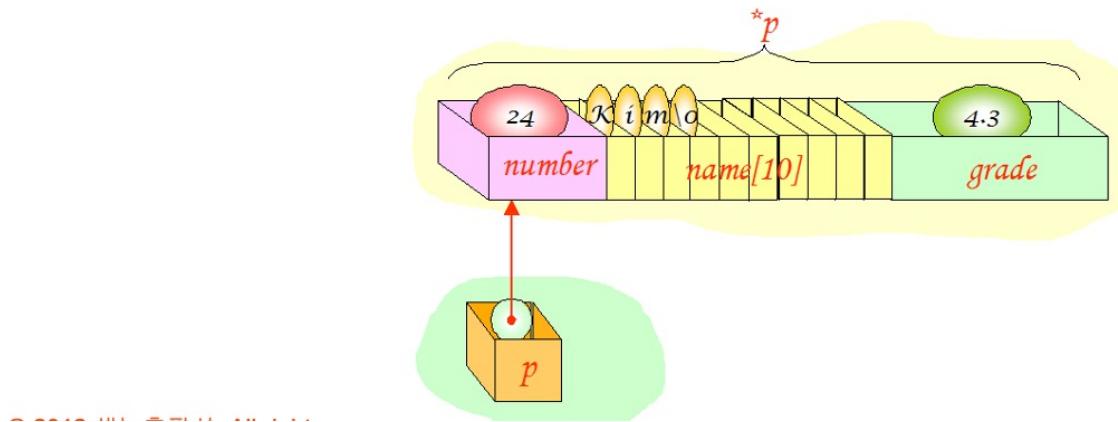
- 구조체를 가리키는 포인터
- 포인터를 멤버로 가지는 구조체

순서로 살펴봅시다.

구조체를 가리키는 포인터

- 구조체를 가리키는 포인터

```
struct student *p;  
  
struct student s = { 20070001, "홍길동", 4.3 };  
struct student *p;  
  
p = &s;  
  
printf("학번=%d 이름=%s 학점=%f \n", s.number, s.name, s.grade);  
printf("학번=%d 이름=%s 학점=%f \n", (*p).number,(*p).name,(*p).grade);
```



-> 연산자

- 연산자는 구조체 포인터로 구조체 멤버를 참조할 때 사용

```
struct student *p;
```

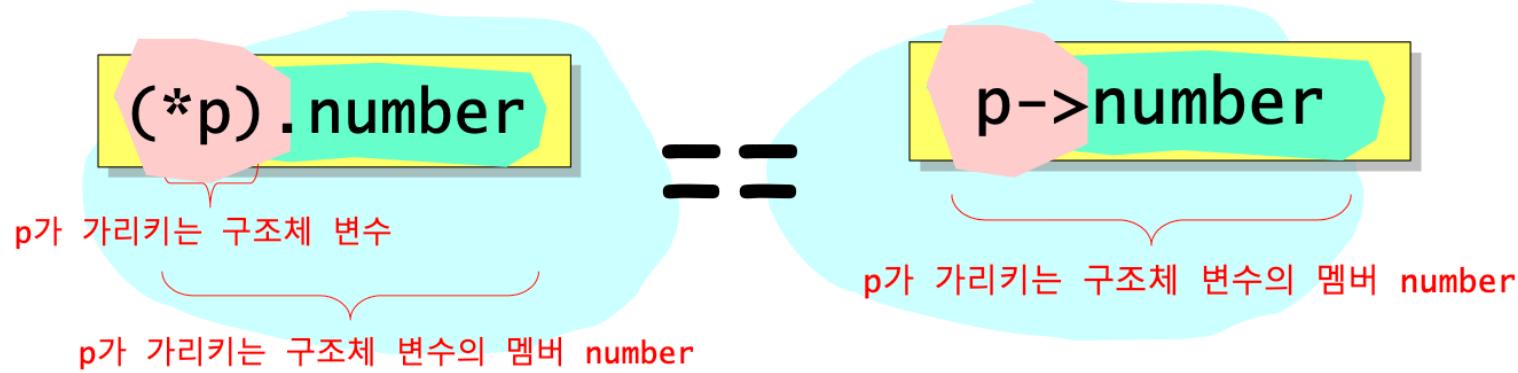
```
struct student s = { 20070001, "홍길동", 180.2 };
struct student *p;
```

```
p = &s;
```

```
printf("학번=%d 이름=%s 키=%f \n", s.number, s.name, s.grade);
printf("학번=%d 이름=%s 키=%f \n", (*p).number,(*p).name,(*p).grade);
```

```
printf("학번=%d 이름=%s 키=%f \n", p->number, p->name, p->grade);
```

-> 연산자 cont'd



구조체 포인터 예제

// 포인터를 통한 구조체 참조

```
#include <stdio.h>
```

```
struct student {  
    int number;  
    char name[20];  
    double grade;  
};
```

```
int main(void)
```

```
{
```

```
    struct student s = { 20070001, "홍길동", 4.3};  
    struct student *p;
```

```
    p = &s;
```

```
    printf("학번=%d 이름=%s 키=%f \n", s.number, s.name, s.grade);
```

```
    printf("학번=%d 이름=%s 키=%f \n", (*p).number,(*p).name,(*p).grade);
```

```
    printf("학번=%d 이름=%s 키=%f \n", p->number, p->name, p->grade);
```

```
    return 0;
```

```
}
```



```
학번=20070001 이름=홍길동 학점=4.300000
```

```
학번=20070001 이름=홍길동 학점=4.300000
```

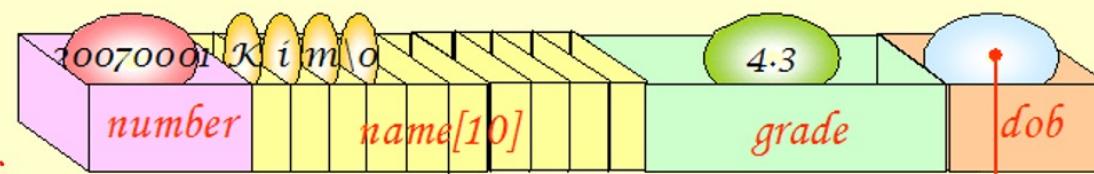
```
학번=20070001 이름=홍길동 학점=4.300000
```

포인터를 멤버로 가지는 구조체

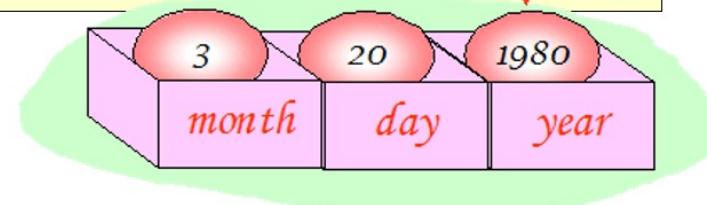
```
struct date {  
    int month;  
    int day;  
    int year;  
};
```

```
struct student {  
    int number;  
    char name[20];  
    double grade;  
    struct date *dob;  
};
```

구조체 s



구조체 d



포인터를 멤버로 가지는 구조체 cont'd

```
int main(void)
{
    struct date d = { 3, 20, 1980 };
    struct student s = { 20070001, "Kim", 4.3 };

    s.dob = &d;

    printf("학번: %d\n", s.number);
    printf("이름: %s\n", s.name);
    printf("학점: %f\n", s.grade);
    printf("생년월일: %d년 %d월 %d일\n", s.dob->year, s.dob->month, s.dob->day
    );
    return 0;
}
```



학번: 20070001
이름: Kim
학점: 4.300000
생년월일: 1980년 3월 20일

구조체와 함수

구조체와 함수

구조체를 함수의 인수로 전달하는 경우

- 구조체의 **복사본**이 함수로 전달되게 된다.
- 만약 구조체의 크기가 크면 그만큼 시간과 메모리가 소요된다.

```
int equal(struct student s1, struct student s2)
{
    if( strcmp(s1.name, s2.name) == 0 )
        return 1;
    else
        return 0;
}
```

구조체와 함수 cont'd

- 구조체의 포인터를 함수의 인수로 전달하는 경우
 - 시간과 공간을 절약할 수 있다.
 - 원본 훼손의 가능성이 있다.

```
int equal(struct student const *p1, struct student const *p2)
{
    if( strcmp(p1->name, p2->name) == 0 )
        return 1;
    else
        return 0;
}
```

포인터를 통한 구조체
의 변경을 막는다.

구조체와 함수 cont'd

- 함수에서 구조체를 반환 할 경우, 그 복사본이 반환된다.

```
struct student make_student(void)
{
    struct student s;

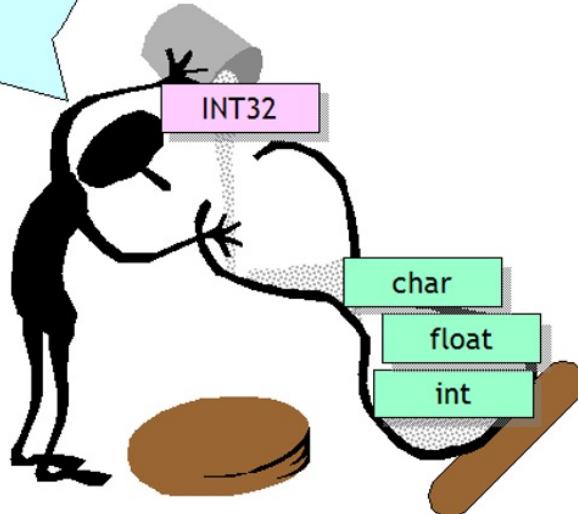
    printf("나이:");
    scanf("%d", &s.age);
    printf("이름:");
    scanf("%s", s.name);
    printf("키:");
    scanf("%f", &s.grade);

    return s;
}
```

구조체 s의 복사본
이 반환된다.

typedef

typedef은 기본 자료형에 새로운 자료형을 추가하는 것입니다.



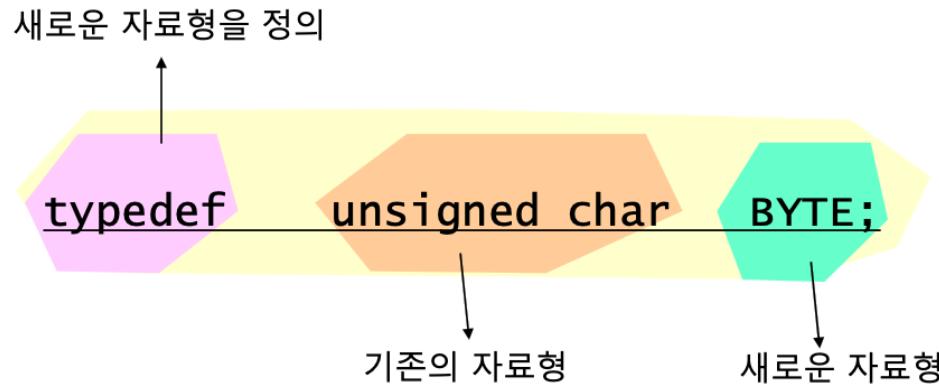
지금부터 *INT32*이라는 새로운 타입을 사용할 수 있음을 알린다.



typedef

- `typedef`은 새로운 자료형(`type`)을 정의(`define`)
- C의 기본 자료형을 확장시키는 역할

```
typedef     old_type     new_type;
```



typedef 예시 (1)

```
typedef unsigned char BYTE;
BYTE index;           // unsigned int index;와 같다.

typedef int INT32;
typedef unsigned int UINT32;

INT32 i;             // int i;와 같다.
UINT32 k;            // unsigned int k;와 같다.
```

구조체를 새로운 타입으로 정의하기

- 구조체로 새로운 타입을 정의할 수 있다.

```
struct point {  
    int x;  
    int y;  
};  
typedef struct point POINT;  
POINT a, b;
```

typedef 예시 (2)

```
#include <stdio.h>

typedef struct point {
    int x;
    int y;
} POINT;

POINT translate(POINT p, POINT delta);

int main(void)
{
    POINT p = { 2, 3 };
    POINT delta = { 10, 10 };
    POINT result;

    result = translate(p, delta);
    printf("새로운 점의 좌표는(%d, %d)입니다.\n", result.x, result.y);

    return 0;
}
```

typedef 예시 (2) cont'd

```
POINT translate(POINT p, POINT delta)
{
    POINT new_p;

    new_p.x = p.x + delta.x;
    new_p.y = p.y + delta.y;

    return new_p;
}
```



새로운 점의 좌표는 (12, 13)입니다.

typedef 와 #define 의 비교

- 이식성을 높여준다.
 - 코드를 컴퓨터 하드웨어에 독립적으로 만들 수 있다
 - (예) int형은 2바이트이기도 하고 4바이트, int형 대신에 **typedef**을 이용한 **INT32**나 **INT16**을 사용하게 되면 확실하게 2바이트인지 4바이트인지를 지정할 수 있다.
- **#define**을 이용해도 **typedef**과 비슷한 효과를 낼 수 있다. 즉 다음과 같이 **INT32**를 정의할 수 있다.
 - **#define UINT32 unsigned int**
 - **typedef float VECTOR[2]; // #define**으로는 불가능하다.
- 문서화의 역할도 한다.
 - **typedef**을 사용하게 되면 주석을 붙이는 것과 같은 효과

이중 포인터

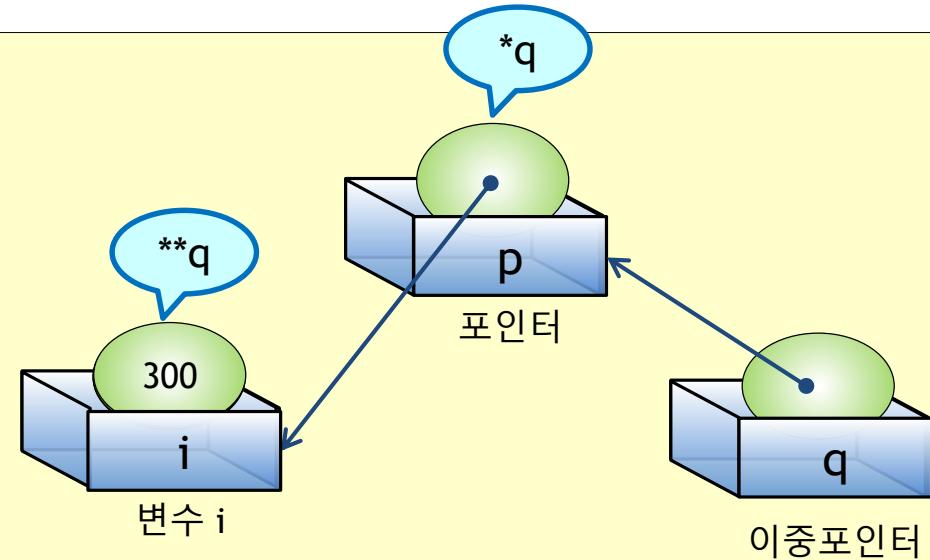
이중 포인터의 해석



이중 포인터

```
// 이중 포인터 프로그램  
#include <stdio.h>
```

```
int main(void)  
{  
    int i = 100;  
    int *p = &i;  
    int **q = &p;  
  
    *p = 200;  
    printf("i=%d  *p=%d  **q=%d \n", i, *p, **q);  
  
    **q = 300;  
    printf("i=%d  *p=%d  **q=%d \n", i, *p, **q);  
  
    return 0;  
}
```



```
*p = 200;  
printf("i=%d  *p=%d  **q=%d \n", i, *p, *q);  
  
**q = 300;  
printf("i=%d  *p=%d  **q=%d \n", i, *p, **q);
```

```
Arthur@DESKTOP-0JL7578 MINGW64 /c/vscodepractice  
$ ./hello.exe  
i=200  *p=200  *q=6422036  
i=300  *p=300  *q=6422036
```

i=200 *p=200 **q=200
i=300 *p=300 **q=300

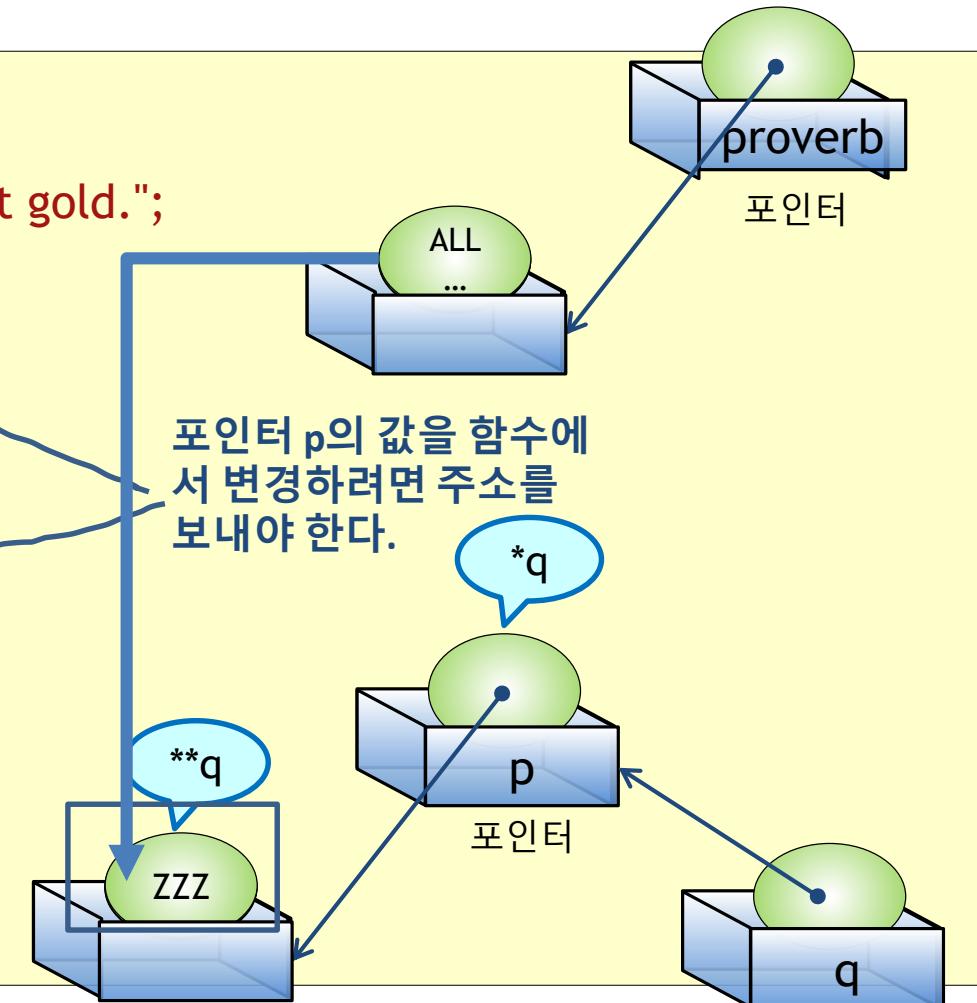


예제 #2

```
#include <stdio.h>
void set_pointer(char **q);
char *proverb="All that glisters is not gold.";
```

```
int main(void)
{
    char *p="zzz";
    set_pointer(&p);
    printf("%s \n", p);
    return 0;
}
```

```
void set_pointer(char **q)
{
    *q = proverb;
}
```



All that glisters is not gold.



중간 점검

- double형 포인터를 가리키는 이중 포인터 dp를 선언하여 보자.
- char c; char *p; char **dp; p = &c; dp = &p;와 같이 정의되었을 때
**dp은 무엇을 가리키는가?

```
1 #include <stdio.h>
2
3 int main() {
4     // double형 포인터를 가리키는 이중 포인터 dp를 선언하여 보자.
5     double **dp;
6
7     // 다음과 같이 정의 된 변수는 각각 무엇을 의미하는가?
8     char c;           // char형 변수 c
9     char *p;          // char형을 가르키는 포인터 p
10    char **dp;         // char형의 포인터를 가르키는 이중포인터 dp
11    p = &c;            // p에는 c의 주소를 대입한다.
12    dp = &p;           // dp에는 p의 주소를 대입한다.
13
14    return 0;
15 }
```

Colored by Color Scripter 



DCSLAB

포인터 배열

포인터 배열(*array of pointers*): 포인터를 모아서 배열로 만든 것

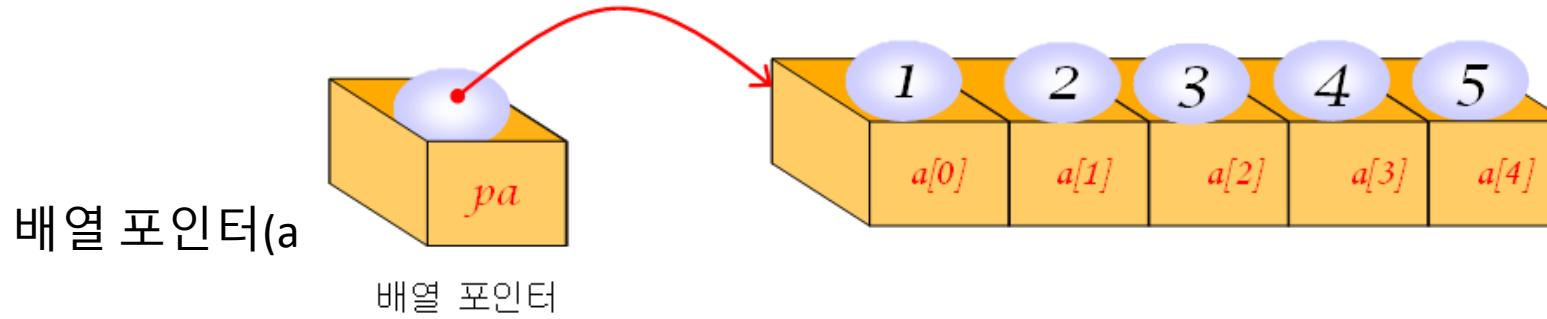
① [] 연산자가 * 연산자보다 우선 순위가 높으므로 ap는 먼저 배열이 된다.

```
int * ap [10];
```

② 어떤 배열이냐 하면 int *(포인터)들의 배열이 된다.



배열 포인터



① 괄호가 있으므로 *pa*는
먼저 포인터가 된다.

*int (*pa)[10];*

② 어떤 포인터나 하면 *int [10]*
을 가리키는 포인터가 된다.



예제

```
#include <stdio.h>

int main(void)
{
    int a[5] = { 1, 2, 3, 4, 5 };
    int (*pa)[5];
    int i;

    pa = &a;
    for(i=0 ; i<5 ; i++)
        printf("%d \n", (*pa)[i]);
    return 0;
}
```

배열 포인터

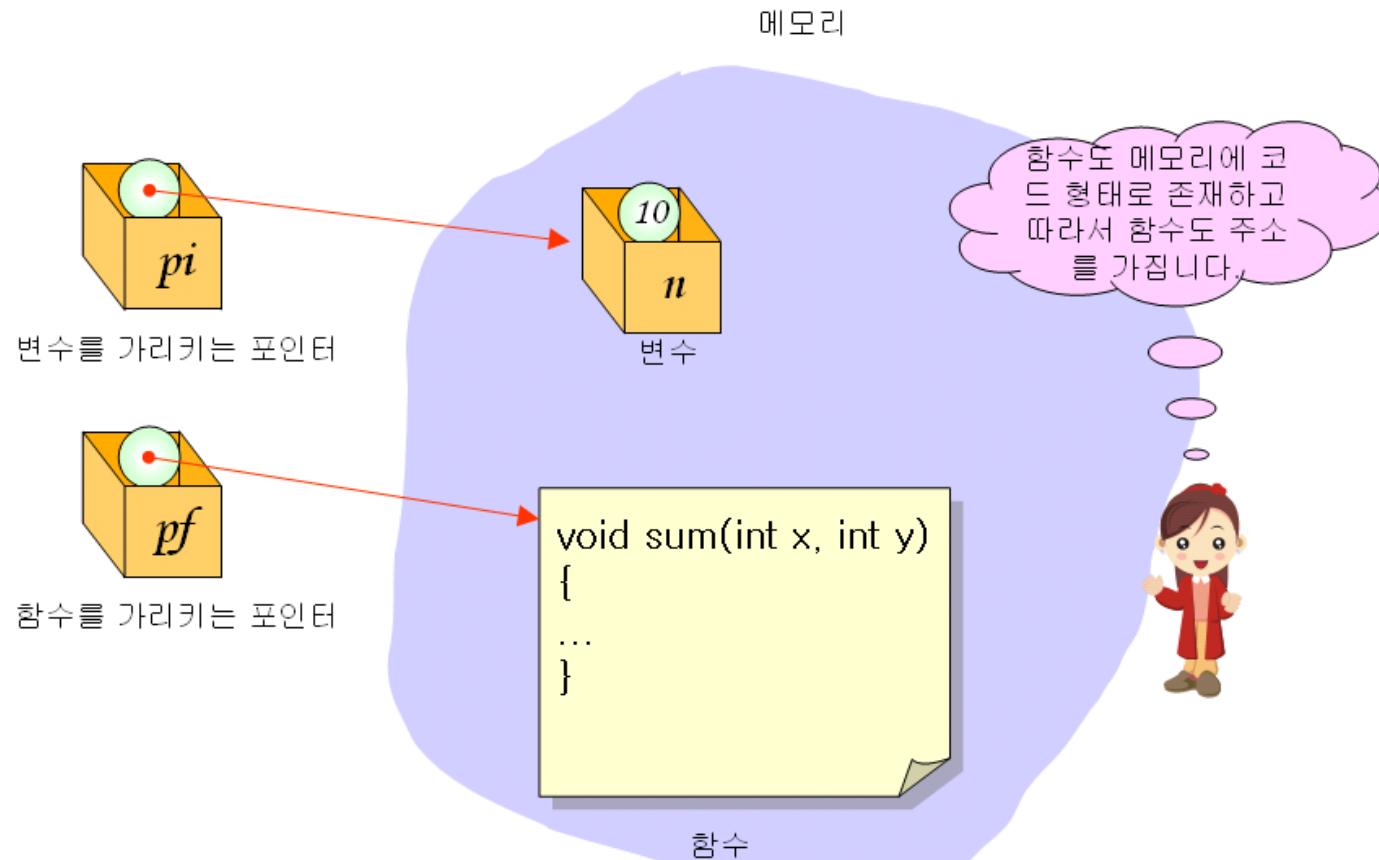


1
2
3
4
5

함수 포인터

함수 포인터(function pointer): 함수를 가리키는 포인터

```
int (*pf)(int, int);
```



함수 포인터의 해석

- ① 괄호에 의하여 () 연산자보다 * 연산자가 먼저 적용되어서 pf는 포인터가 된다.

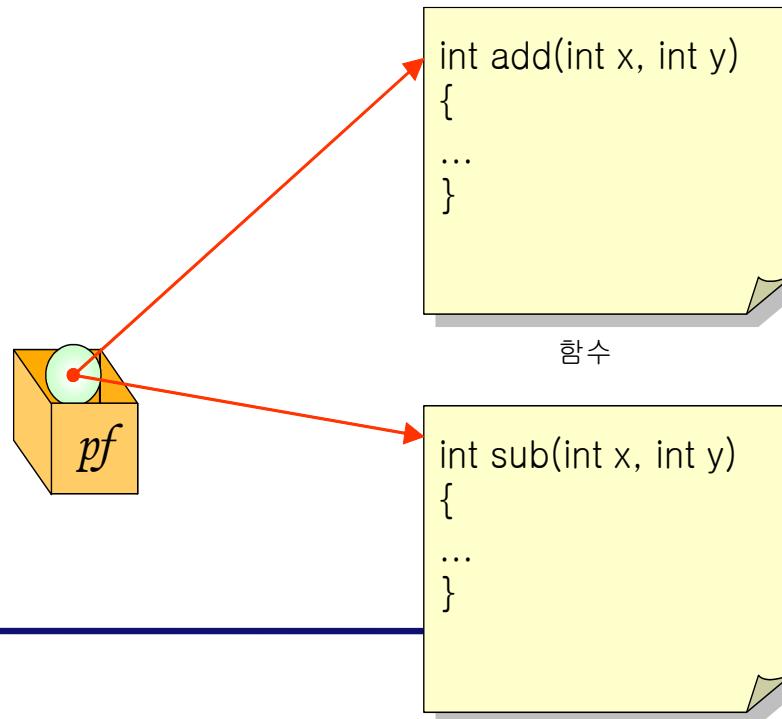
```
int (*pf) (int, int);
```

- ② 어떤 포인터냐 하면 int pf (int, int) 함수를 가리키는 포인터가 된다.



함수 포인터의 사용

```
int sub(int, int);           // 함수 원형 정의  
int (*pf)(int, int);        // 함수 포인터 정의  
...  
pf = sub;                   // 함수의 이름을 함수 포인터에 대입  
result = pf( 10, 20);       // 함수 포인터를 통하여 함수 호출
```



fp1.c

```
// 함수 포인터
#include <stdio.h>

// 함수 원형 정의
int add(int, int);
int sub(int, int);

int main(void)
{
    int result;
    int (*pf)(int, int); // 함수 포인터 정의

    pf = add; // 함수 포인터에 함수 add()의 주소 대입
    result = pf(10, 20); // 함수 포인터를 통한 함수 add() 호출
    printf("10+20은 %d\n", result);

    pf = sub; // 함수 포인터에 함수 sub()의 주소 대입
    result = pf(10, 20); // 함수 포인터를 통한 함수 sub() 호출
    printf("10-20은 %d\n", result);

    return 0;
}
```



fp1.c

```
int add(int x, int y)
{
    return x+y;
}

int sub(int x, int y)
{
    return x-y;
}
```



10+20은 30
10-20은 -10



함수 포인터의 배열

```
int (*pf[5]) (int, int);
```

① [] 연산자가 * 연산자보다 우선 순위가 높으므로 pf는 먼저 배열이 된다.

```
int (*pf[5]) (int, int);
```

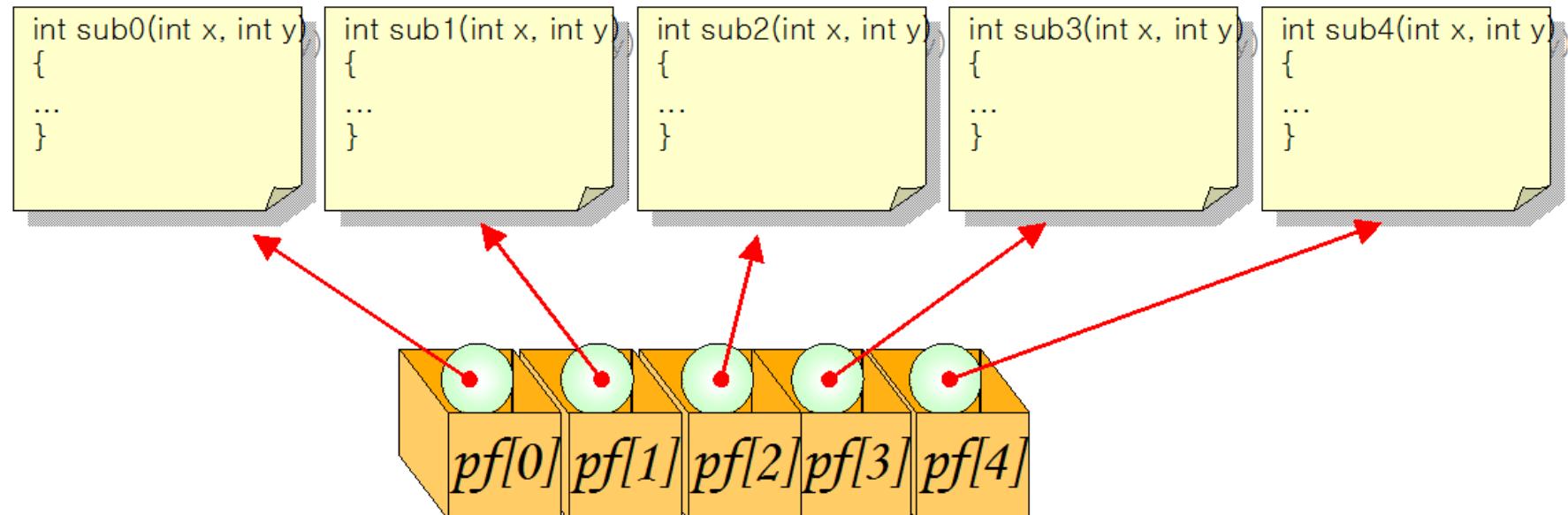
② 어떤 배열이냐 하면 i포인터들의 배열이 된다.

③ 어떤 포인터냐 하면 함수를 가리키는 포인터가 된다.



함수 포인터의 배열

```
int (*pf[5]) (int, int);
```



함수 포인터 배열

```
// 함수 포인터 배열
#include <stdio.h>

// 함수 원형 정의
void menu(void);
int add(int x, int y);
int sub(int x, int y);
int mul(int x, int y);
int div(int x, int y);

void menu(void)
{
    printf("=====\\n");
    printf("0. 덧셈\\n");
    printf("1. 뺄셈\\n");
    printf("2. 곱셈\\n");
    printf("3. 나눗셈\\n");
    printf("4. 종료\\n");
    printf("=====\\n");
}
```



함수 포인터 배열

```
int main(void)
{
    int choice, result, x, y;
    // 함수 포인터 배열을 선언하고 초기화한다.
    int (*pf[4])(int, int) = { add, sub, mul, div };

    while(1)
    {
        menu();
        printf("메뉴를 선택하시오:");
        scanf("%d", &choice);

        if( choice < 0 || choice >=4 )
            break;
        printf("2개의 정수를 입력하시오:");
        scanf("%d %d", &x, &y);

        result = pf[choice](x, y);      // 함수 포인터를 이용한 함수 호출
        printf("연산 결과 = %d\n",result);
    }
    return 0;
}
```

함수 포인터 배열
선언



DCSLAB

함수 포인터 배열

```
int add(int x, int y)
{
    return x + y;
}

int sub(int x, int y)
{
    return x - y;
}

int mul(int x, int y)
{
    return x * y;
}

int div(int x, int y)
{
    return x / y;
}
```



=====
0. 덧셈
1. 뺄셈
2. 곱셈
3. 나눗셈
4. 종료
=====

메뉴를 선택하시오:2
2개의 정수를 입력하시오:10 20
연산 결과 = 200
=====

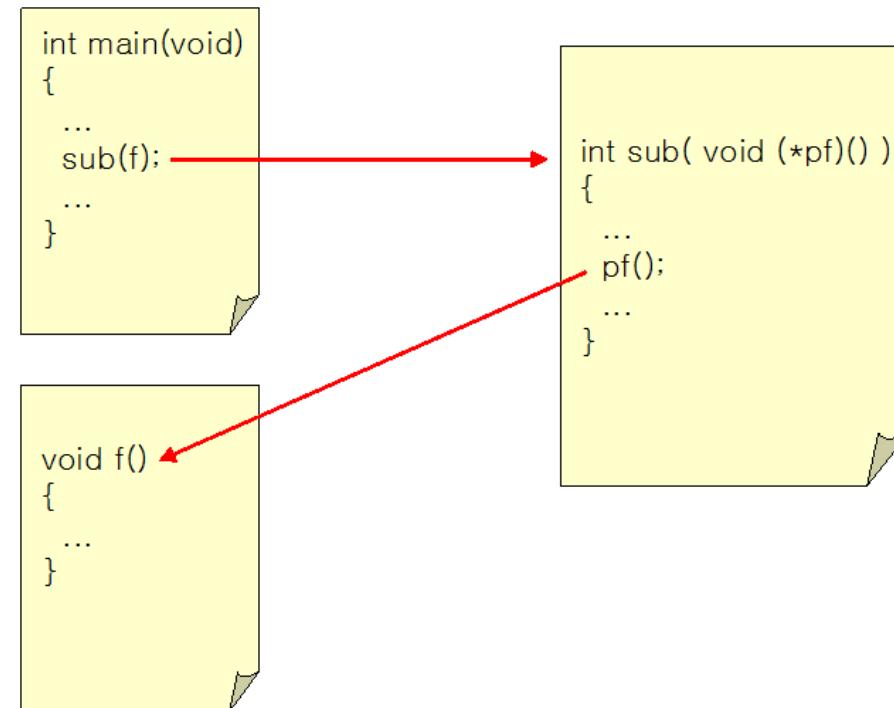
0. 덧셈
1. 뺄셈
2. 곱셈
3. 나눗셈
4. 종료
=====

메뉴를 선택하시오:

함수 인수로서의 함수 포인터

함수 포인터.

특정 함수를
호출하게 할
수 있어요.



DCSLAB

예제

다음과 같은 수식을 계산하는 프로그램을 작성하여 보자.

$$\sum_{1}^n (f^2(k) + f(k) + 1)$$

여기서 $f(k)$ 는 다음과 같은 함수들이 될 수 있다.

$$f(k) = \frac{1}{k} \text{ 또는 } f(k) = \cos(k)$$

예제

```
#include <stdio.h>
#include <math.h>

double f1(double k);
double f2(double k);
double formula(double (*pf)(double), int n);

int main(void)
{
    printf("%f\n", formula(f1, 10));
    printf("%f\n", formula(f2, 10));
}

double formula(double (*pf)(double), int n)
{
    int i;
    double sum = 0.0;

    for(i = 1; i < n; i++)
        sum += pf(i) * pf(i) + pf(i) + 1;

    return sum;
}
```

$$\sum_{1}^n (f^2(k) + f(k) + 1)$$



예제

```
double f1(double k)
{
    return 1.0 / k;
}

double f2(double k)
{
    return cos(k);
}
```



13.368736
12.716152



DCSLAB

중간 점검

int 값을 반환하고 double 값을 인수로 받는 함수의 포인터 pf를 선언하여 보자.
1번의 함수 포인터를 통하여 3.0을 인수로 하여 함수를 호출하는 문장을 작성하라.

```
1 #include <stdio.h>
2
3 int target(double, double);
4
5 int main() {
6     // int 값을 반환하고 double 값을 인수로 받는 함수의 포인터 pf를 선언하여 보자.
7     int(*pf)(double, double) = target; // pf = target;
8     // 위의 함수 포인터를 통하여 3.0을 인수로 하여 함수를 호출하는 문장을 작성하라.
9     printf("%d\n", pf(3.0, 3.0));
10 }
11
12 int target(double a, double b) {
13     return a + b;
14 }
```

Colored by Color Scripter 



const 포인터

- const를 붙이는 위치에 따라서 의미가 달라진다.

p가 가리키는 내용이 변경되지 않음을 나타낸다.

*const char *p;*

포인터 p가 변경되지 않음을 나타낸다.

*char * const p;*



예제

```
Welcome C test.c

C test.c ▶ main(void)
1 #include <stdio.h>
2 int main(void)
3 {
4     char s[] = "Barking dogs seldom bite.";
5     char t[] = "A bad workman blames his tools";
6     const char * p=s;
7     char * const q=s;
8
9     //p[3] = 'a';
10    printf("%s \n", p);
11    p = t;
12    printf("%s \n", p);
13    q[3] = 'a';
14    printf("%s \n", q);
15
16    //q = t;
17
18    return 0;
19 }
20

PS C:\Users\arthur\Desktop\쉽게 풀어쓴 c언어 express> ./test
Barking dogs seldom bite.
A bad workman blames his tools
Baraing dogs seldom bite.
PS C:\Users\arthur\Desktop\쉽게 풀어쓴 c언어 express>
```



void 포인터

- 순수하게 메모리의 주소만 가지고 있는 포인터
- 가리키는 대상물은 아직 정해지지 않음
(예) `void *vp;`
- 다음과 같은 연산은 모두 오류이다.

```
*vp;          // 오류
*(int *)vp;    // void형 포인터를 int형 포인터로 변환한다.
vp++;         // 오류
vp--;         // 오류
```



vp.c

C test.c ↗ ...

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     int a[] = { 10, 20, 30, 40, 50 };
6     void *vp;
7     printf("a[2] : %d\n", a[2]);
8     vp = a; // 가능
9     vp = &a[2]; // 가능
10    printf("a[2] : %d\n", a[2]);
11    /*vp = 35; // 오류
12    vp++; // 오류
13
14    *(int *)vp = 35; // int 포인터로 cast
15    printf("vp pointer value %d \n", *(int *)vp);
16    printf("vp pointer change int value a[2] : %d\n", a[2]);
17
18
19    return 0;
20 }
21
```



DCSLAB

중간 점검

void형 포인터 vp를 int형 포인터 ip로 형변환하는 문장을 작성하라.

```
C test.c ▶ ⌂ main()
1  #include <stdio.h>
2
3  int main() {
4
5      int target = 5;
6      void *vp = &target;
7
8      // void형 포인터 vp를 int형 포인터 ip로 형변환하는 문장을 작성하라.
9      printf("%d\n", *(int *)vp);
10
11     return 0;
12 }
13
```



QnA