Pointers and Arrays Structures

adopted from KNK C Programming : A Modern Approach

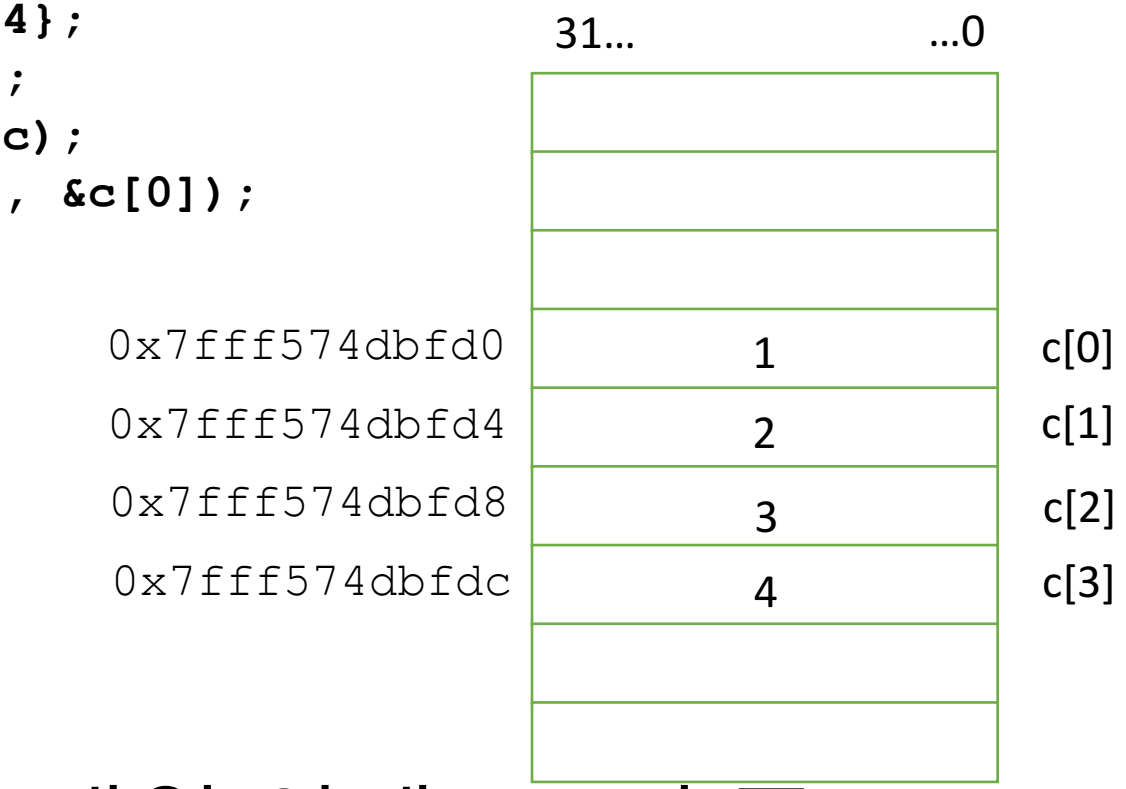
요약

Pointers and Arrays

배열의 주소

```
#include <stdio.h>
int main(){
    int c[] = {1, 2, 3, 4};
    printf("c\t%p\n", c);
    printf("&c\t%p\n", &c);
    printf("&c[0]\t%p\n", &c[0]);
    return 0;
}
```

```
$ gcc -o a.out addr.c
$ ./a.out
c       0x7fff574dbfd0
&c      0x7fff574dbfd0
&c[0]   0x7fff574dbfd0
```

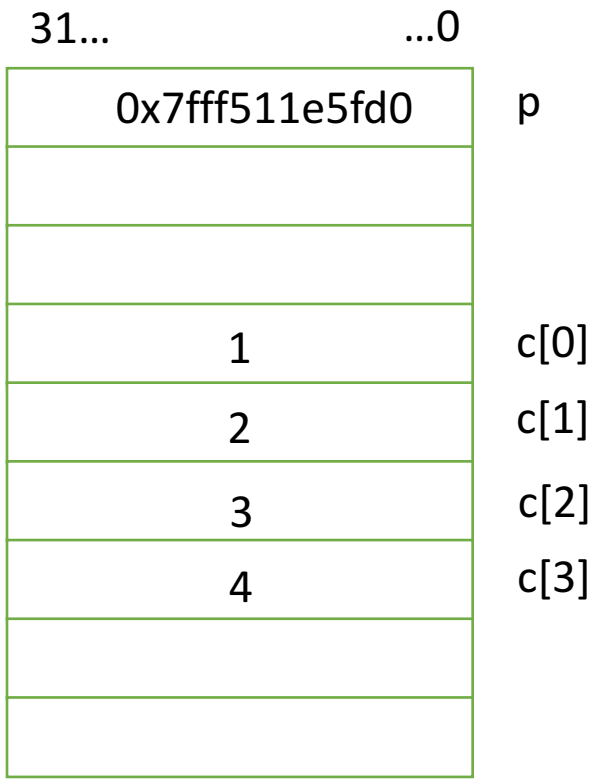


배열 시작 주소 = 배열 인덱스 0의 주소

배열의 주소를 포인터로 확인하는 방법

```
#include <stdio.h>
int main(){
    int c[] = {1, 2, 3, 4};
    int *p = c;
    printf("c\t%p\n", c);
    printf("&c\t%p\n", &c);
    printf("&c[0]\t%p\n", &c[0]);
    printf("&*p\t%p\n",&*p);
    return 0;
}
```

```
$ gcc -o a.out addr.c
$ ./a.out
c       0x7fff574dbfd0
&c      0x7fff574dbfd0
&c[0]   0x7fff574dbfd0
&*p     0x7fff58497fd0
```

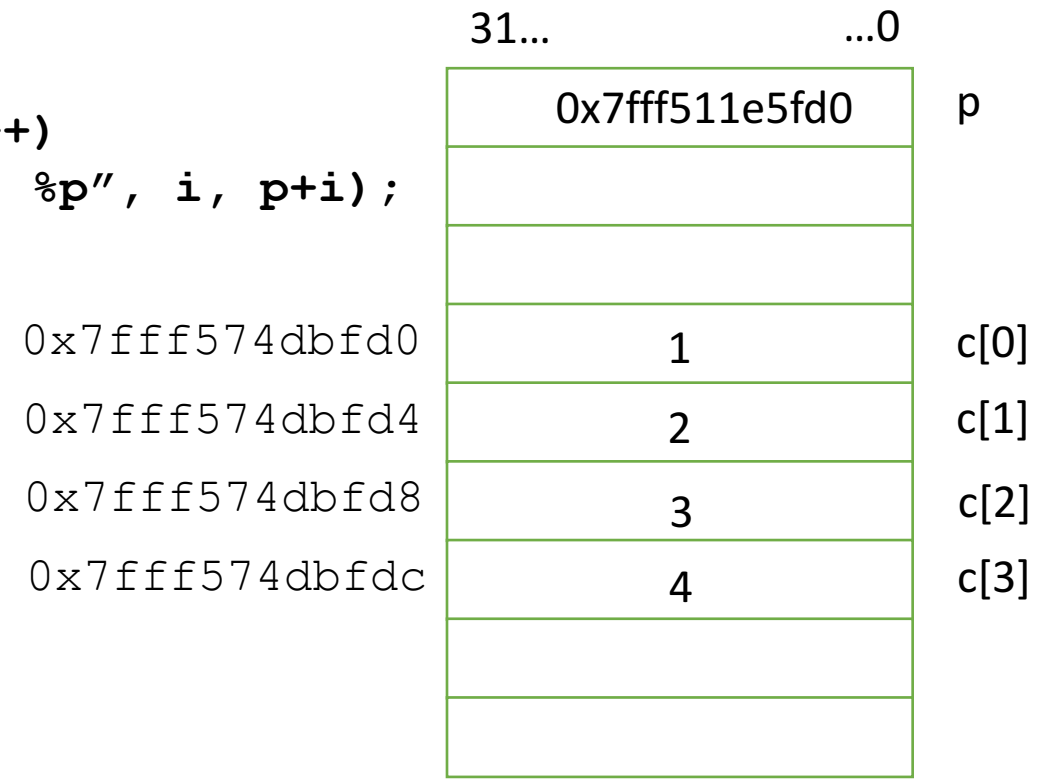


포인터로 배열 참조 가능

포인터 변수의 덧셈

```
#include <stdio.h>
int main(){
    int c[] = {1, 2, 3, 4};
    int *p = c;
    for (int i = 0; i<4; i++)
        printf("p+%d, addr: %p", i, p+i);
    return 0;
}
```

```
$ gcc -o a.out addr.c
$ ./a.out
p+0     0x7fff574dbfd0
p+1     0x7fff574dbfd4
p+2     0x7fff574dbfd8
p+3     0x7fff58497fdc
```

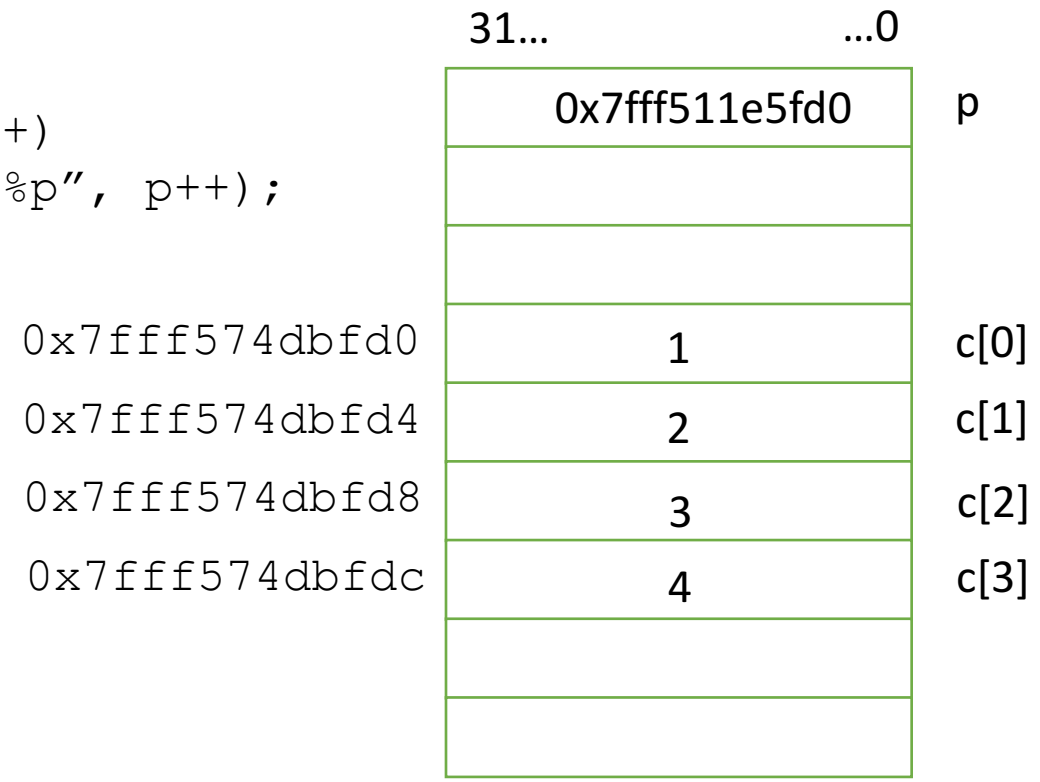


포인터 변수를 증감하면 형의 크기만큼 증감

포인터 변수의 덧셈 II

```
#include <stdio.h>
int main(){
    int c[] = {1, 2, 3, 4};
    int *p = c;
    for (int i = 0; i<4; i++)
        printf("p++, addr: %p", p++);
    return 0;
}
```

```
$ gcc -o a.out addr.c
$ ./a.out
p++     0x7fff574dbfd0
p++     0x7fff574dbfd4
p++     0x7fff574dbfd8
p++     0x7fff58497fdc
```



++, -- 증감 연산자도 형의 크기만큼 증감

Putting it together

1. 배열 시작 주소 = 배열 인덱스 0의 주소

2. 포인터로 배열 참조 가능

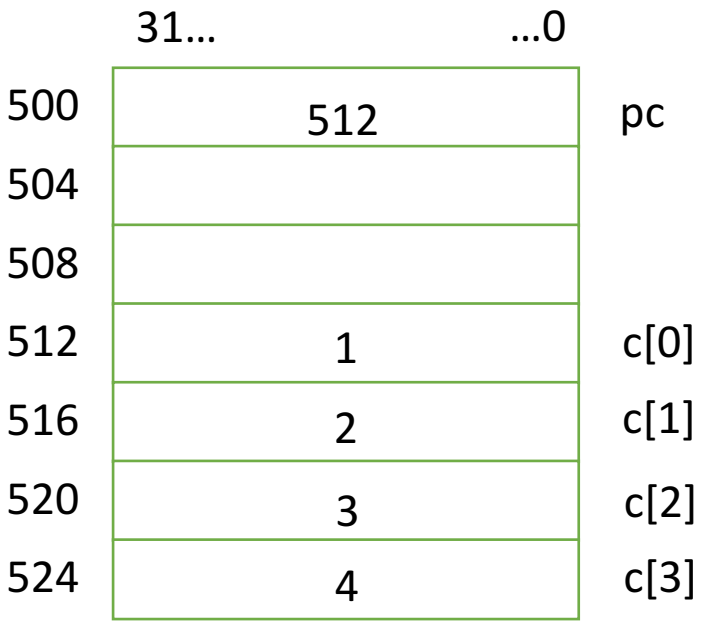
+ 3. 포인터 변수의 증감 → 형의 크기만큼 증감

포인터로 배열 인덱스 참조 가능

포인터로 배열 인덱스 지정하기 공식

시작 주소+(변수의 타입)*인덱스

```
int c[] = {1, 2, 3, 4};
int *p = c;
p+2;
```



c[2]의 주소 = 512 + 4*2 = 520

포인터로 배열 인덱스 지정하기

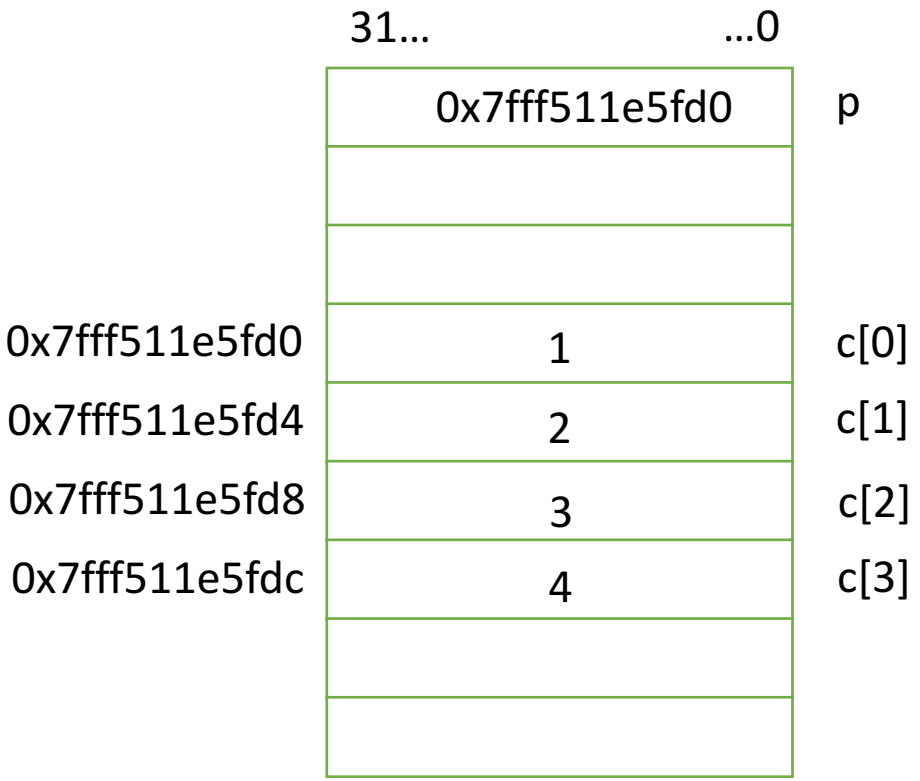
- 포인터 변수는 형의 크기 만큼 증감함

```
int c[] = {1, 2, 3, 4};
int *p = c;
```

연습 문제

*p

c 주소의 값 1을 읽음



포인터로 배열 인덱스 지정하기

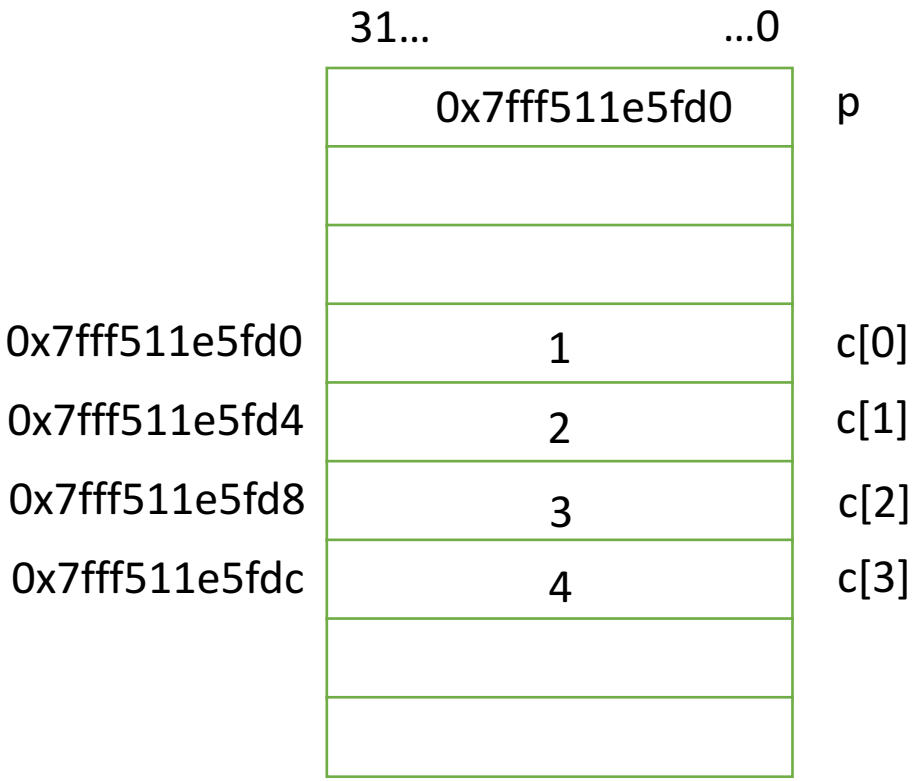
- 포인터 변수는 형의 크기 만큼 증감함

```
int c[] = {1, 2, 3, 4};
int *p = c;
```

연습 문제

*p c[0]을 읽음

*p+2 c[2]를 읽음



구조체 (structures)

꼭 알아야 할 것

- 구조체란
- 메모리에서의 표현
- 구조체 선언
- 구조체 초기화
- 구초제 값의 접근
- 구조체 활용
- 구조체로 형 선언

구조체란

- 복합 저장 공간
  - 하나 이상의 서로 다른 형의 변수를 저장 가능
  - 구조체 안에 구조체 포함 가능
  - 관련있는 데이터의 모음

모이면 풍성해지는 구조체

- 예:
  - 좌표계의 한 지점의 위치 값 (x, y, z)
  - 성적표의 과목마다 받은 성적 (국, 영, 수, …)
  - 연락처에 들어가는 정보 (이름, 전화1, 전화2, 생일, 주소, …)
  - 도서관 서지 정보 (책 제목, 페이지수, 출판사, 출판일, …)

구조체 선언: 기본 구조

```
struct {
    변수형  변수이름;
    변수형  변수이름;
    …
} 구조체명;
```

멤버 변수

좌표계의 한 지점의 위치 값 (x, y, z)

지점의 좌표

```
struct {
    int x;
    int y;
    int z;
} PointA;
```

성적표의 과목마다 받은 성적 (국, 영, 수, …)

성적표

```
struct {
    float Kor;
    float Eng;
    float Math;
} Score;
```

구조체 선언: 기본 구조

```
struct {
    변수형  변수이름;
    변수형  변수이름;
    …
} 구조체명;
```

멤버 변수

연락처에 들어가는 정보 (이름, 전화1, 전화2, 생일, 주소, …)

전화번호

```
struct {
    char Name[20];
    char Phone[13];
    char Birth[10];
    char Addr[100];
} PersonInfo;
```

도서관 서지 정보 (책 제목, 페이지수, 출판사, 출판일, …)

서지정보

```
struct {
    int Title;
    int Page;
    int Pub;
    int Year;
} Book;
```

구조체 초기화: 선언시 초기화

```
struct {
    변수형 변수이름;
    변수형 변수이름;
    …
} 구조체명 = {값1, 값2, … };
```

배열처럼 초기화

지점의 좌표

```
struct {
    int x;
    int y;
    int z;
} PointA = { 30, 40, 50 }, PointB = {50, 30, 20 };
```

구조체 값의 접근

지점의 좌표

```
struct {
    int x;
    int y;
    int z;
} PointA = { 30, 40, 50 }, PointB = {50, 30, 20 };
```

새로운 연산자 . 점

구조체의 멤버 변수 접근

```
PointA . x; PointA . y; PointA . z;
```

구조체 초기화: 선언 후 초기화

지점의 좌표

```
struct {
    int x;
    int y;
    int z;
} PointA;
```

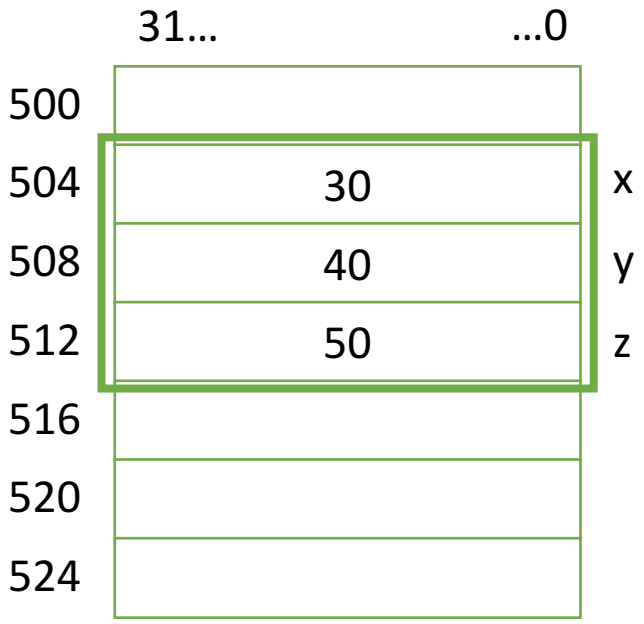
```
PointA.x = 30;
PointA.y = 40;
PointA.z = 50;
```

메모리에서의 표현

지점의 좌표

```
struct {
    int x;
    int y;
    int z;
} PointA;

PointA.x = 30;
PointA.y = 40;
PointA.z = 50;
```



구조체 활용: 함수의 인자, 매개변수

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

구조체의 선언

리턴 타입: struct point

```
struct point savePoint(int x, int y, int z)
{
  struct point p;
  p.x = x;
  p.y = y;
  p.z = z;
  return p;
}
```

구조체 변수 리턴

```
int main()
{
    struct point pointA;
    pointA = savePoint(30, 40, 50);
…
}
```

리턴 받은 정보를 PointA에 저장

구조체 형 선언

- 태그로 형 선언
- typedef로 형 선언

지점의 좌표

```
struct Point{
    int x;
    int y;
    int z;
} ;
```

구조체에 반복 사용할 수 있는 이름을 부여함

```
struct Point A = {30, 40, 50};
struct Point B = {40, 30, 20};
```

구조체 태그가 있기 때문에 구조체 재선언 없이
같은 타입의 구조체를 재활용하여 선언 할 수 있음
태그가 없으면 구조체 선언부를 매번 다시 써야 함

구조체 형 선언

- 태그로 선언하여 쓰는 방법의 단점
  - “struct 구조체 태그명”을 반복하여 써야 함
- typedef으로 선언하는 방법
  - 선언한 구조체가 새로운 형이됨

지점의 좌표

```
typedef struct {
    int x;
    int y;
    int z;
} Point;
```

typedef 새로운 형을 만들 때 쓰는 키워드

새로운 형의 이름

```
Point A = {30, 40, 50};
Point B = {40, 30, 20};
```

다른 형들처럼 변수 명 앞에 씀

구조체의 할당

지점의 좌표

```
typedef struct {
    int x;
    int y;
    int z;
} Point;

Point A = {30, 40, 50};
Point B;
B = A;
```

구조체는 = 연산자로 복사 됨
A.x에 30, A.y에 40, A.z에 50을 저장
B = A; 문장으로 통해
B.x에 30, B.y에 40, B.z에 50을 저장

단, 모든 구조체가 동일하게 동작하는 것은 아님
상세 자료 참고