



C 프로그래밍 및 실습

C Programming

이지민



CHAP 10 구조체

구조체의 기본_구조체의 개념

- ❖ 구조체는 서로 다른 데이터형의 변수들을 하나로 묶어서 사용하는 기능이다.

```
char title[40];  
int price ;  
double rate;
```



```
struct content {  
    char title[40];  
    int price;  
    double rate;  
};
```

- ❖ C 언어에서 구조체는 사용자 정의형을 만드는 방법이다.

```
char name[20];  
char phone[20];  
int ringtone;
```

함께 사용되는
변수들을 묶어서
구조체를 정의한다.



```
태그 이름  
struct contact  
{  
    char name[20]; // 이름  
    char phone[20]; // 전화번호  
    int ringtone; // 벨 소리  
};
```

구조체의
멤버

세미콜론이
필요하다

struct contact이
새로운 데이터형이 된다.

구조체의 기본_구조체의 정의

❖ 태그 이름

- 구조체를 구별하기 위한 이름

❖ 구조체의 멤버

- 구조체를 구성하는 변수

❖ 구조체 정의의 끝을 나타내는 세미콜론(;)이 필요하다.

형식

```
struct 태그명 {  
    데이터형 멤버명;  
    데이터형 멤버명;  
    :  
};
```

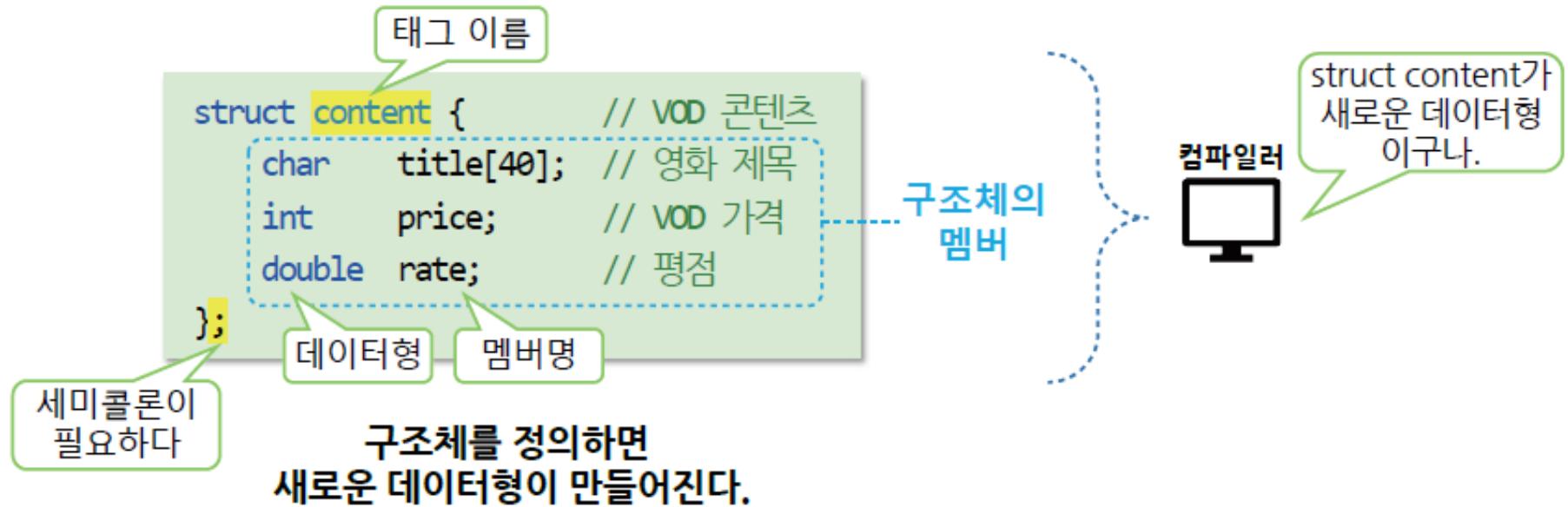
사용예

```
struct content {  
    char title[40];  
    int price;  
    double rate;  
};
```

❖ 구조체는 함수 밖에 정의하며, 소스 파일의 시작 부분에 정의하는 것이 좋다.

구조체의 기본_구조체의 정의

- ❖ 구조체를 정의하면 새로운 데이터형이 만들어진다.
 - 구조체형의 변수를 선언해야 구조체 변수가 메모리에 할당된다.



구조체의 기본_구조체의 정의

- ❖ 구조체의 바이트 크기는 멤버들의 바이트 크기를 모두 더한 것보다 크거나 같다.

```
printf("%d", sizeof(struct content));
```

content 구조체의 바이트
크기를 구한다.

- ❖ 구조체를 사용할 때는 **struct** 키워드와 태그 이름을 함께 사용해야 한다.

- 'struct content'가 데이터형 이름이 된다.
- **struct** 키워드 없이 태그 이름만 사용하면 컴파일 에러가 발생한다.



```
printf("%d", sizeof(content));
```

content가 구조체 이름
이라는 것을 알 수 없다.

구조체의 기본_구조체의 정의

❖content 구조체의 정의

```
01 #include <stdio.h>          구조체는 소스  
02                                     파일의 시작  
03                                     부분에 정의한다.  
04 struct content {                  // VOD 콘텐츠  
05     char    title[40];           // 영화 제목  
06     int     price;            // VOD 가격  
07     double   rate;            // 평점(0~10 사이의 값)  
08 };  
09  
10 int main(void)  
11 {  
12     printf("content 구조체의 크기: %d\n", sizeof(struct content));  
13 }
```

실행 결과

content 구조체의 크기: 56

멤버의 크기 합인 52보다 큰 값이다.

구조체의 바이트 크기를 구한다.

구조체의 기본_구조체 변수

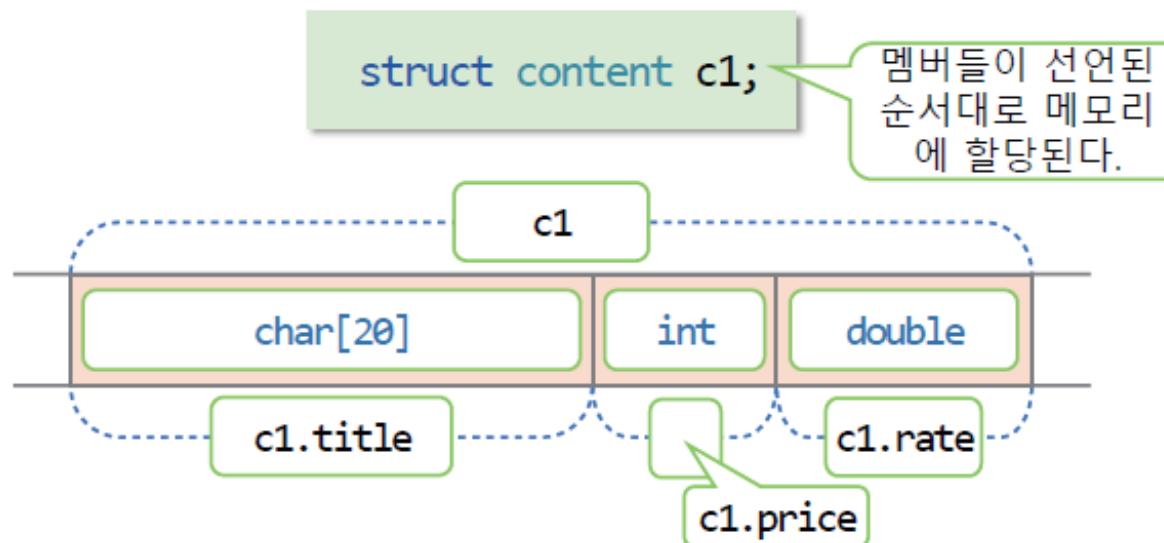
❖ 구조체 변수의 선언

형식 `struct 태그명 변수명;`

사용예

```
struct content movie;  
struct content c1, c2;
```

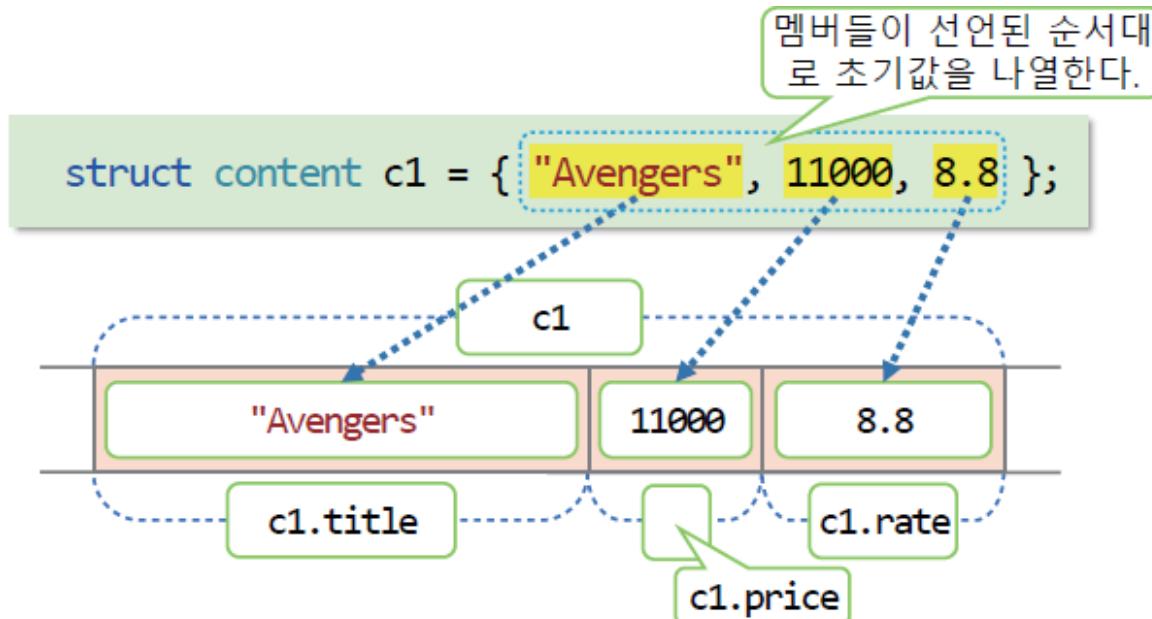
- 구조체의 멤버들이 선언된 순서대로 메모리에 할당



구조체의 기본_구조체 변수

❖ 구조체 변수의 초기화(1/2)

- { } 안에 멤버들의 초기값을 선언된 순서대로 나열



```
struct content c2 = { "Aladdin", 11000 };
```

초기값이 부족하면 나머지는 0으로 초기화

```
struct content c3 = { "Spider-Man", 5500, 8.1, 1 };
```

초기값이 멤버의 개수보다 많으면 안된다.

구조체의 기본_구조체 변수

❖ 구조체 변수의 초기화(2/2)

- 구조체를 정의하면서 구조체 변수를 함께 선언할 수 있다.

```
struct app_info {  
    char title[128];  
    int notification;  
    int version;  
} facebook;
```

구조체를 정의하면서
변수를 선언한다.

```
struct {  
    char title[128];  
    int notification;  
    int version;  
} facebook;
```

일회성으로 사용되는 구조체는
태그 이름을 생략할 수 있다.

facebook 변수 선언 후
이 구조체형을 다시 사용
할 수 없다.

구조체의 기본_구조체 변수

❖ 구조체 변수의 사용(1/2)

- 구조체의 멤버에 접근하려면 멤버 접근 연산자(.)를 이용한다.

```
c1.rate = 8.9;  
c1.price *= 0.8;  
strcat(c1.title, ": Endgame");
```

- 구조체의 멤버는 항상 구조체 변수를 통해서만 접근할 수 있다.



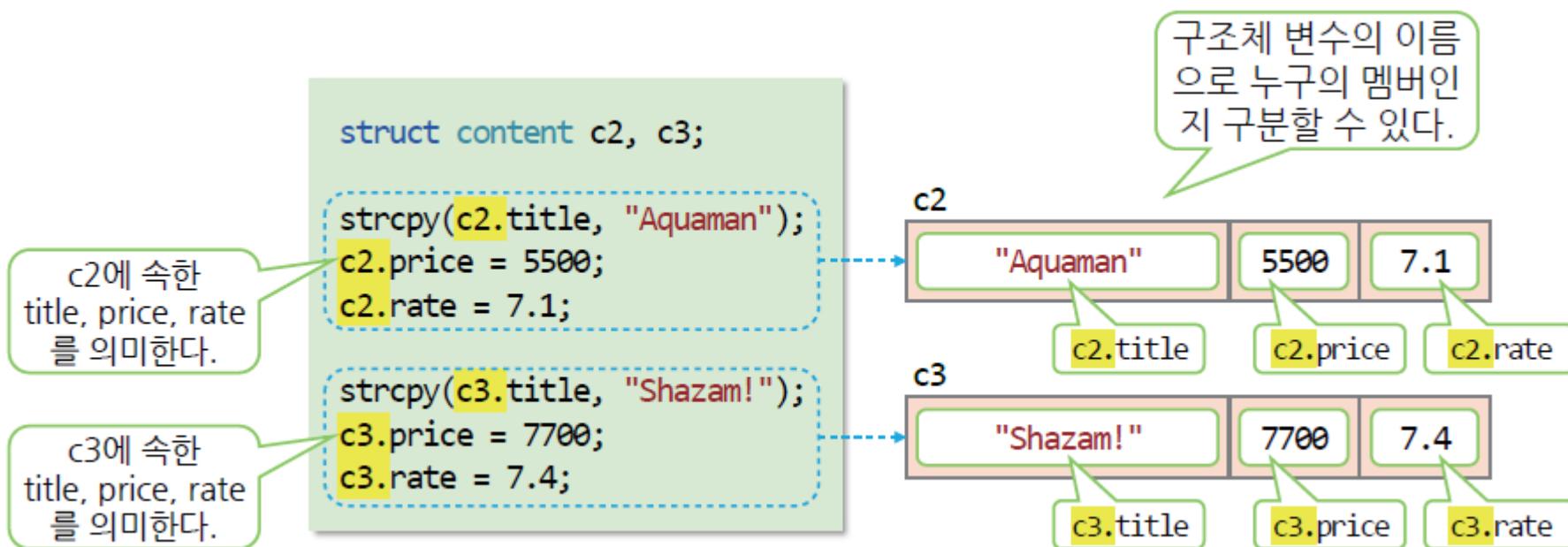
```
rate = 8.9;
```

content 구조체의 멤버가 아닌
일반 변수 rate를 의미한다.

구조체의 기본_구조체 변수

❖ 구조체 변수의 사용(2/2)

- 구조체 변수를 여러 개 선언하면, 서로 다른 메모리에 할당된다.
 - 구조체의 멤버는 구조체 변수마다 각각 할당된다.
 - 구조체 변수를 통해서 멤버에 접근하기 때문에 어떤 변수의 멤버를 사용하는지 구분할 수 있다.



구조체의 기본_구조체 변수

❖ 구조체 변수의 선언 및 사용 예

```
01 #define _CRT_SECURE_NO_WARNINGS  
02 #include <stdio.h>  
03 #include <string.h>  
04  
05 struct content {  
06     char    title[40];  
07     int     price;  
08     double   rate;  
09 };  
10  
11 int main(void)  
12 {  
13     struct content c1 = { "Avengers", 11000, 8.8 };  
14     struct content c2, c3;
```

실행 결과

```
c1 = Avengers, 11000, 8.8  
c2 = Aquaman, 5500, 7.1  
c3 = Shazam!, 7700, 7.4
```

구조체 변수의
선언 및 초기화

초기화하지 않으면
쓰레기값이 된다.

구조체의 기본_구조체 변수

❖ 구조체 변수의 선언 및 사용 예

```
16     strcpy(c2.title, "Aquaman");
17     c2.price = 5500;
18     c2.rate = 7.1;
```

구조체 변수 c2의
멤버를 변경한다.

```
19
20     strcpy(c3.title, "Shazam!");
21     c3.price = 7700;
22     c3.rate = 7.4;
```

구조체 변수 c3의
멤버를 변경한다.

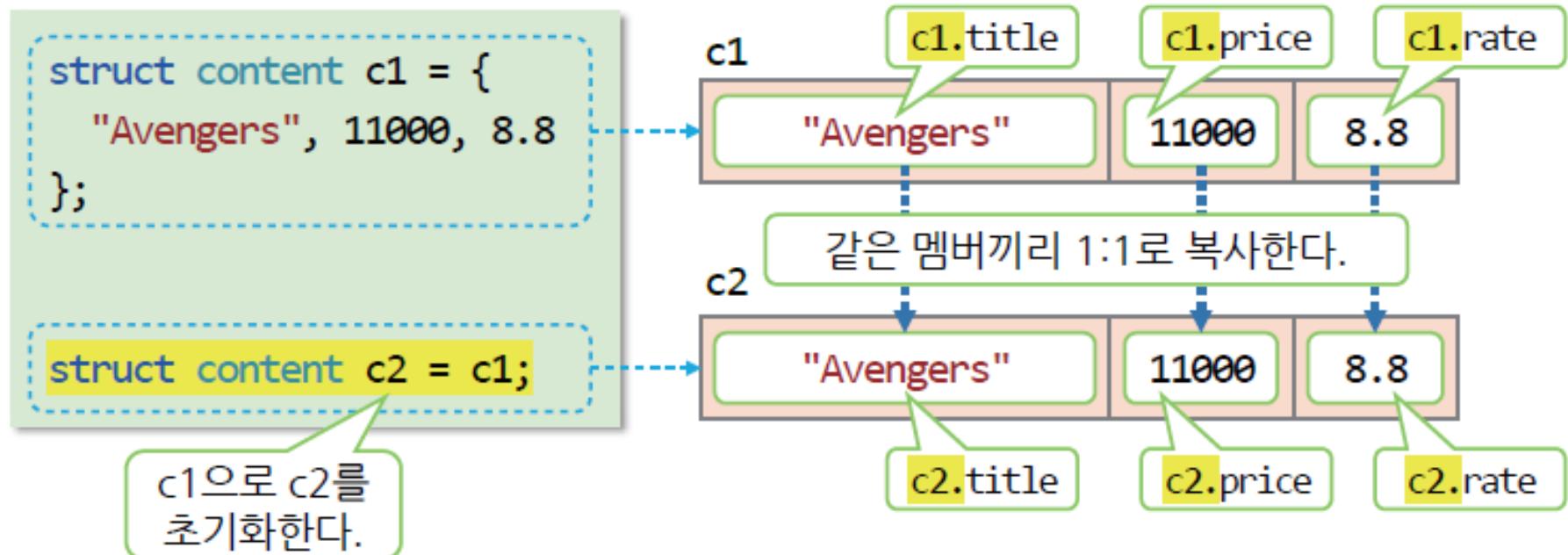
```
23
24     printf("c1 = %s, %d, %.1f\n", c1.title, c1.price, c1.rate);
25     printf("c2 = %s, %d, %.1f\n", c2.title, c2.price, c2.rate);
26     printf("c3 = %s, %d, %.1f\n", c3.title, c3.price, c3.rate);
27 }
```

구조체 변수 c1, c2, c3의
멤버를 출력한다.

구조체의 기본_구조체 변수

❖ 구조체 변수 간의 초기화

- 구조체 변수로 다른 구조체 변수를 초기화하면, 같은 멤버끼리 1:1로 복사해서 초기화한다.



구조체의 기본_구조체 변수

❖ 구조체 변수 간의 대입

- 같은 멤버끼리 1:1로 복사해서 대입한다.

```
c3 = c1;
```

c1의 모든 멤버가
c3로 대입된다.

- 구조체 변수에 {}를 이용해서 값을 직접 대입할 수는 없다.



```
c3 = { "Aquaman", 5500, 7.1 };
```

대입에는 {}를 사용할 수 없다.

- 구조체의 멤버인 배열끼리는 대입할 수 없다.



```
c3.title = c1.title;
```

구조체의 멤버인 배열끼리는
대입할 수 없다.

```
strcpy(c3.title, c1.title);
```

title이 문자 배열이므로 문자열
처리 함수로 복사한다.

구조체의 기본_구조체 변수

❖ 구조체 변수 간의 초기화와 대입(1/2)

```
01 #include <stdio.h>
02 #include <string.h>
03
04 struct content {
05     char    title[40];
06     int     price;
07     double   rate;
08 };
09
10 int main(void)
11 {
```

구조체의 정의

실행 결과

```
c1 = Avengers, 11000, 8.8
c2 = Avengers, 11000, 8.8
c3 = Avengers, 11000, 8.8
```

구조체의 기본_구조체 변수

❖ 구조체 변수 간의 초기화와 대입(2/2)

```
12     struct content c1 = { "Avengers", 11000, 8.8 };
13     struct content c2 = c1;                                같은 멤버끼리 1:1로
14     struct content c3 = { 0 };                            복사해서 초기화한다.
15
16     c3 = c1;                                         같은 멤버끼리 1:1로
17                                         복사해서 대입한다.
18
19     printf("c1 = %s, %d, %.1f\n", c1.title, c1.price, c1.rate);
20     printf("c2 = %s, %d, %.1f\n", c2.title, c2.price, c2.rate);
21     printf("c3 = %s, %d, %.1f\n", c3.title, c3.price, c3.rate);
22 }
```

구조체의 기본_구조체 변수

❖ 구조체 변수의 비교

- 구조체 변수에는 관계 연산자를 사용할 수 없다.



```
if (c1 == c2)
```

==의 피연산자로 구조체 변수를 사용할 수 없다.

```
printf("c1과 c2가 같습니다.\n");
```

- 구조체 변수의 값이 같은지 비교하려면 멤버 대 멤버로 비교해야 한다.

title 멤버는 문자열이므로
strcmp 함수를 이용해야 한다.

```
if (strcmp(c1.title, c2.title) == 0 && c1.price == c2.price && c1.rate == c2.rate)  
printf("c1과 c2의 값이 같습니다.\n");
```

구조체의 기본_구조체 변수

❖ 구조체 변수의 비교 예

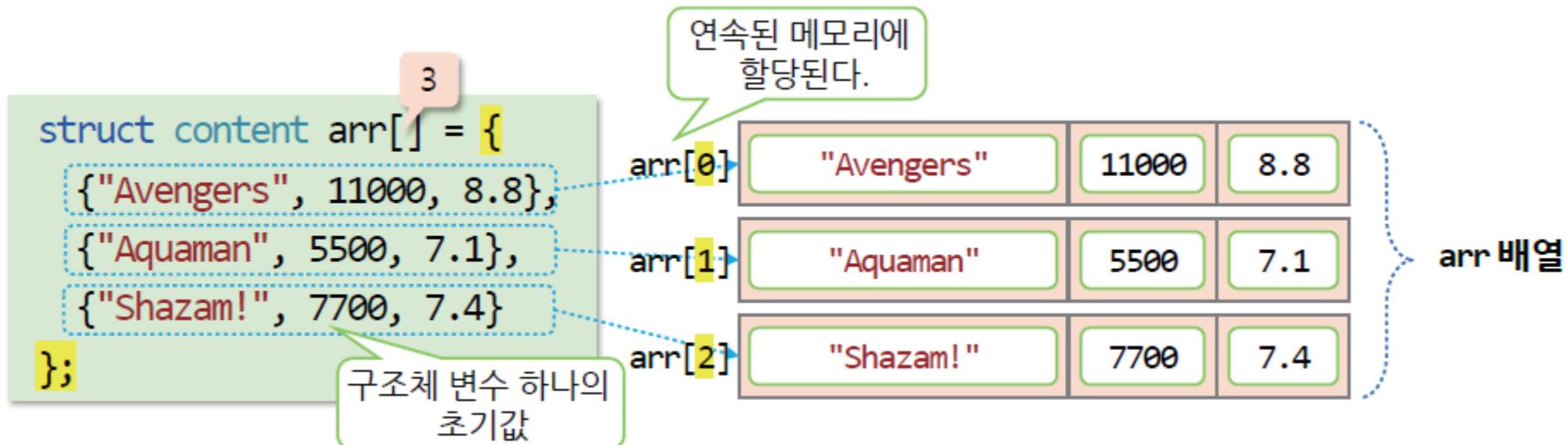
```
01 #include <stdio.h>
02 #include <string.h>
04 struct content {
05     char    title[40];
06     int     price;
07     double   rate;
08 };
10 int main(void)
11 {
12     struct content c1 = { "Avengers", 11000, 8.8 };
13     struct content c2 = c1;           // c1으로 c2를 초기화한다.
16     if (strcmp(c1.title, c2.title) == 0 && c1.price == c2.price && c1.rate == c2.rate)
17         printf("c1과 c2의 값이 같습니다.\n");
18     else
19         printf("c1과 c2의 값이 다릅니다.\n");
20 }
```

각각의 멤버가 모두 같은지 비교한다.

구조체의 활용_구조체 배열

❖ 구조체 배열 선언 및 사용(1/2)

- 구조체 배열을 선언하려면, 배열 이름을 쓰고 [] 안에 배열의 크기를 지정한다.
 - 구조체 배열의 원소들도 메모리에 연속적으로 할당된다.
- 구조체 배열을 초기화하려면 { } 안에 배열 원소의 초기값을 나열한다.



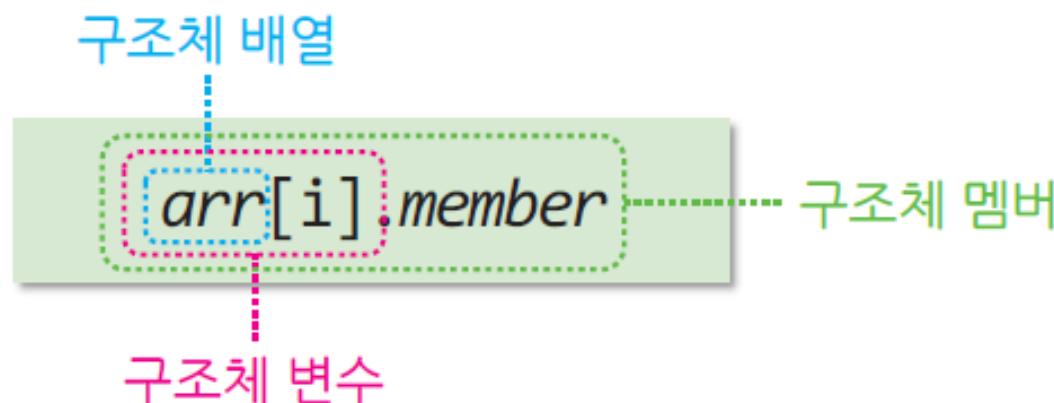
구조체의 활용_구조체 배열

❖ 구조체 배열 선언 및 사용(2/2)

- 구조체 배열의 원소에 접근하려면 인덱스를 이용한다.
 - *arr*가 구조체 배열의 이름일 때 구조체의 멤버에 접근하려면 *arr[i].member*로 접근한다.

```
for (i = 0; i < size; i++) // arr[i]는 content 구조체 변수이다.
```

```
printf("arr[%d] = %s, %d, %.1f\n", i, arr[i].title, arr[i].price, arr[i].rate);
```



구조체의 활용_구조체 배열

❖ 구조체 배열의 사용 예

```
10 int main(void)           구조체 배열의  
11 {                         선언 및 초기화  
12     struct content arr[] = {  
13         {"Avengers", 11000, 8.8},  
14         {"Aquaman", 5500, 7.1},  
15         {"Shazam!", 7700, 7.4}  
16     };  
17     int size = sizeof(arr) / sizeof(arr[0]);  
18     int i;  
19  
20     for (i = 0; i < size; i++)  
21         printf("arr[%d] = %s, %d, %.1f\n", i, arr[i].title, arr[i].price, arr[i].rate);  
22 }
```

구조체 배열의 선언 및 초기화

실행 결과

arr[0] = Avengers, 11000, 8.8
arr[1] = Aquaman, 5500, 7.1
arr[2] = Shazam!, 7700, 7.4

구조체 배열의 크기

arr[i]는 content 구조체 변수이다.

구조체의 활용_구조체 배열

❖ 구조체 배열의 탐색

▪ 콘텐츠 검색 기능

- 먼저 검색할 제목을 입력받아 제목으로 검색해서 찾은 콘텐츠의 가격과 평점을 확인하는 프로그램

```
char title[40];
printf("제목? ");
gets_s(title, sizeof(title));
```

- 구조체 배열의 원소 중에서 title과 제목이 같은 원소를 찾으면 break로 for 문을 탈출한다.

```
for (i = 0; i < size; i++)
    if (strcmp(arr[i].title, title) == 0)
        break;
```

제목이 같은 원소를 찾으면
for를 탈출한다.
(i는 찾은 원소의 인덱스)

구조체의 활용_구조체 배열

❖ 구조체 배열의 검색(1/2)

```
10 int main(void)
11 {
12     struct content arr[] = {
13         {"Avengers", 11000, 8.8}, {"Aquaman", 5500, 7.1}, {"Shazam!", 7700, 7.4},
14         {"X-Men", 3300, 8.0}, {"Us", 8800, 7.1}, {"Inception", 2200, 8.7}
15     };
16     int size = sizeof(arr) / sizeof(arr[0]); ----- 초기화된 구조체 배열
17     int i;
18     char title[40];
19
20     printf("제목? ");
21     gets_s(title, sizeof(title)); ----- 검색할 제목 문자열을
                                            입력받는다.
```

구조체의 활용_구조체 배열

❖ 구조체 배열의 검색(2/2)

```
23     for (i = 0; i < size; i++)  
24         if (strcmp(arr[i].title, title) == 0)  
25             break;  
26  
27     if (i == size)  
28         printf("해당 콘텐츠를 찾을 수 없습니다.\n");  
29     else  
30         printf("%s: 가격=%d, 평점=%.1f\n", arr[i].title, arr[i].price, arr[i].rate);  
31 }
```

검색 성공 시 i가
찾은 원소의 인덱스

실행 결과

제목? Aquaman

Aquaman: 가격=5500, 평점=7.1

실행 결과

제목? Captain Marvel

해당 콘텐츠를 찾을 수 없습니다.

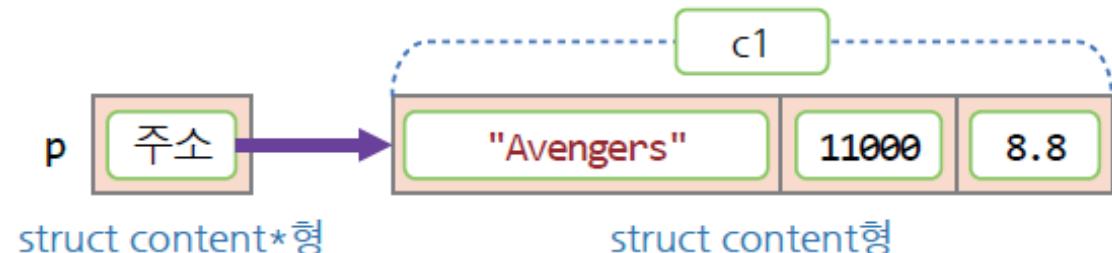
구조체의 활용_구조체 포인터

❖ 구조체 포인터 개념(1/2)

- 구조체 포인터는 구조체 변수의 주소를 저장하는 포인터

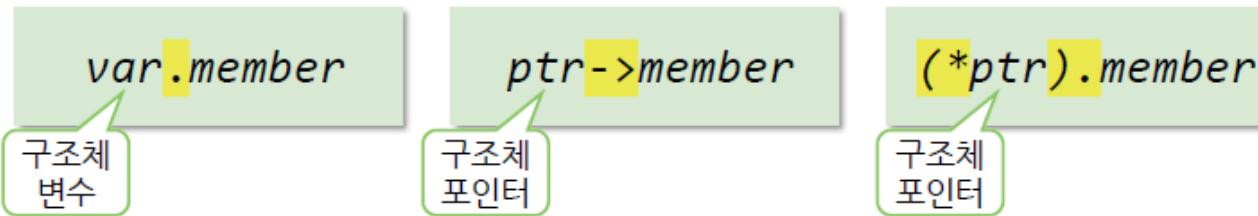
```
struct content c1 = {  
    "Avengers", 11000, 8.8  
};  
struct content *p = &c1;
```

p는 content 구조체
변수를 가리킨다.



▪ 간접 멤버 접근 연산자

- 구조체 포인터 `ptr`이 가리키는 구조체 변수의 멤버에 접근하려면, `ptr->member` 또는 `(*ptr).member`을 사용한다.



구조체의 활용_구조체 포인터

❖ 구조체 포인터 개념(2/2)

- 구조체 변수에 읽기 전용으로 접근하게 하려면 **const** 포인터로 선언

```
const struct content* ptr = &c1;
```

content 구조체 변수에 대한
읽기 전용 포인터

- const** 포인터로는 구조체 변수의 값을 읽어볼 수만 있고 변경할 수는 없다.



```
ptr->price = 9900;
```

ptr이 가리키는 구조체의
멤버를 변경할 수 없다.

구조체의 활용_구조체 포인터

❖ 구조체 포인터 예

```
11 int main(void)
12 {
13     struct content c1 = { "Avengers", 11000, 8.8 };
14     struct content *p = &c1;          구조체 변수를 가리키는
15
16     p->price *= 0.8;              포인터 선언
17     p->rate = 8.9;
18     strcat(p->title, ": Endgame");
19
20     printf("%s, %d, %.1f\n", p->title, p->price, p->rate);
21 }
```

p가 가리키는 구조체의
멤버를 변경한다.

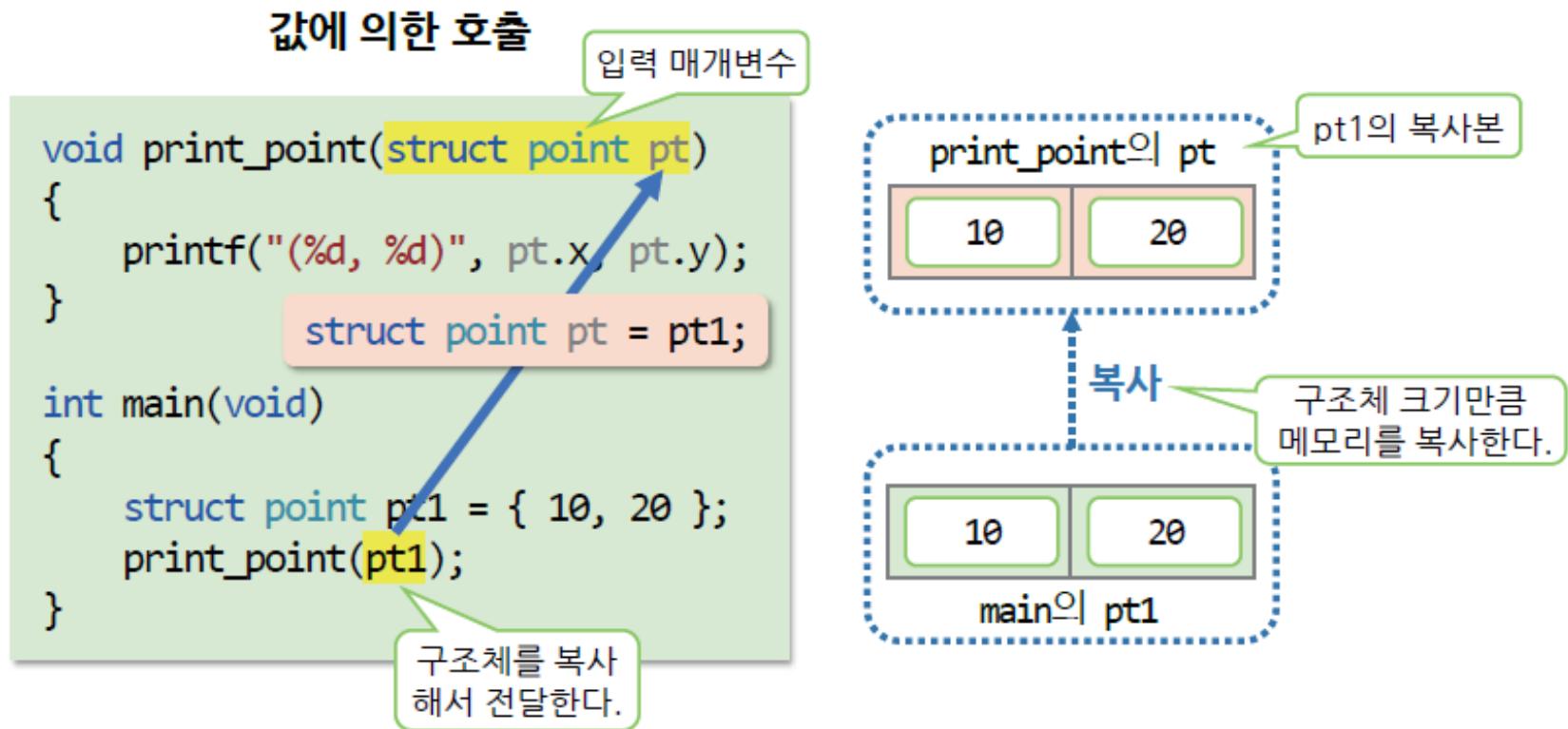
실행 결과

Avengers: Endgame, 8800, 8.9

구조체의 활용_함수의 인자로 구조체 전달

❖ 값에 의한 호출(1/2)

- 구조체를 값에 의한 호출로 전달하면 인자를 매개변수로 복사해서 전달한다. → 구조체 복사



구조체의 활용_함수의 인자로 구조체 전달

❖ 값에 의한 호출(2/2)

- 구조체의 복사는 메모리를 복사하는 데 시간이 걸리므로 공간적·시간적 성능 저하를 유발한다.
- 기본형에 비해 크기가 큰 구조체는 복사하는 대신 구조체의 주소를 전달하는 것이 좋다. → 참조에 의한 호출
 - point처럼 크기가 작은 구조체는 값에 의한 호출을 사용해도 성능 저하가 거의 없다.
 - content처럼 크기가 큰 구조체는 참조에 의한 호출로 전달한다.

구조체의 활용_함수의 인자로 구조체 전달

❖ 값에 의한 호출로 구조체 전달하기 예(1/2)

```
01 #include <stdio.h>
02
03 struct point {
04     int x, y;
05 };
06 void print_point(struct point pt);
07
08 int main(void)
09 {
10     struct point arr[] = {
11         {10, 20}, {35, 41}, {12, 63}, {72, 55}, {92, 86}, {4, 27}
12     };
13     int size = sizeof(arr) / sizeof(arr[0]);
14     int i;
```

구조체 정의가 함수 선언보다 앞에 와야 한다.

구조체를 매개변수로 갖는 함수의 선언

구조체의 활용_함수의 인자로 구조체 전달

❖ 값에 의한 호출로 구조체 전달하기 예(2/2)

```
15  
16     for (i = 0; i < size; i++) {  
17         print_point(arr[i]);  
18     }  
19     printf("\n");  
20 }  
21  
22 void print_point(struct point pt)  
23 {  
24     printf("(%.d, %.d) ", pt.x, pt.y);  
25 }
```

arr[i]를 함수의 매개변수 pt로 복사해서 전달한다.

pt는 입력 매개변수이므로 값에 의한 호출로 전달

실행 결과

(10, 20) (35, 41) (12, 63) (72, 55) (92, 86) (4, 27)

구조체의 활용_함수의 인자로 구조체 전달

❖ 참조에 의한 호출(1/2)

- 구조체를 복사하지 않고 전달하려면 포인터로 전달

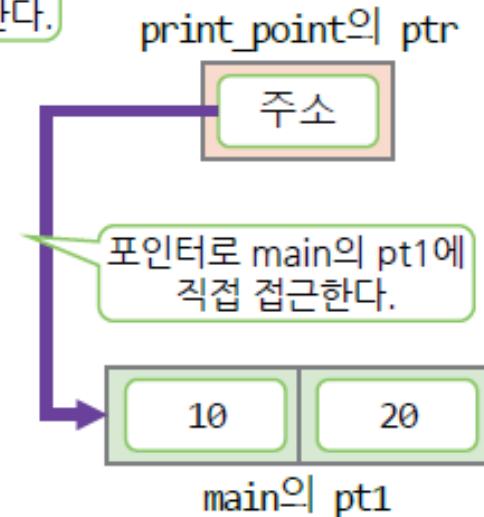
참조에 의한 호출

```
void print_point(struct point *ptr)
{
    printf("(%d, %d)", ptr->x, ptr->y);
}
struct point &ptr = &pt1;

int main(void)
{
    struct point pt1 = { 10, 20 };
    print_point(&pt1);
}
```

구조체를 복사하지 않고 주소만 전달한다.

구조체 포인터로 멤버에 접근한다.



구조체의 활용_함수의 인자로 구조체 전달

❖ 참조에 의한 호출(2/2)

- 구조체가 입력 매개변수일 때는 **const** 포인터로 전달하는 것이 좋다.
 - 함수 안에서 **const** 포인터가 가리키는 구조체 변수를 변경할 수 없다.

```
void print_point(const struct point* ptr)
{
    NO     ptr->x = 100;
    printf("(%.d, %.d) ", ptr->x, ptr->y);
}
```

- 구조체가 함수의 출력 매개변수이거나 입출력 매개변수일 때는 일반 포인터로 전달한다.

구조체의 활용_함수의 인자로 구조체 전달

❖ 참조에 의한 호출로 구조체 전달하기 예(1/3)

```
01 #define _CRT_SECURE_NO_WARNINGS
02 #include <stdio.h>
03
04 struct point {
05     int x, y;
06 };
07 void print_point(const struct point* ptr);
08 void move_point(struct point* ptr, int offset);
09
10 int main(void)
11 {
12     struct point arr[] = {
13         {10, 20}, {35, 41}, {12, 63}, {72, 55}, {92, 86}, {4, 27}
14     };
15     int size = sizeof(arr) / sizeof(arr[0]);
16     int i, offset;
```

입력 매개변수

구조체를 매개변수로 갖는 함수의 선언

출력 매개변수

구조체의 활용_함수의 인자로 구조체 전달

❖ 참조에 의한 호출로 구조체 전달하기 예(2/3)

```
18     for (i = 0; i < size; i++) {  
19         print_point(&arr[i]);  
20     }  
21     printf("\n");  
22  
23     printf("이동할 오프셋? ");  
24     scanf("%d", &offset);
```

arr[i]의 주소를 매개변수 ptr로 전달한다.
구조체를 복사하지 않고 전달하므로 효율적이다.

```
25     for (i = 0; i < size; i++) {  
26         move_point(&arr[i], offset);  
27         print_point(&arr[i]);  
28     }  
29 }  
30 }
```

arr[i]를 offset만큼 이동 한다. → arr[i] 변경

구조체의 활용_함수의 인자로 구조체 전달

❖ 참조에 의한 호출로 구조체 전달하기 예(3/3)

```
31 void print_point(const struct point* ptr) ----- ptr은 입력 매개변수
32 {
33     printf("(%d, %d) ", ptr->x, ptr->y);
34 }
35
36 void move_point(struct point* ptr, int offset)
37 {
38     ptr->x = ptr->x + offset; ----- ptr은 출력 매개변수
39     ptr->y = ptr->y + offset;
40 }
```

ptr은 출력 매개변수

ptr이 가리키는 구조체의
멤버를 변경한다.

실행 결과

(10, 20) (35, 41) (12, 63) (72, 55) (92, 86) (4, 27)

이동할 오프셋? 5

(15, 25) (40, 46) (17, 68) (77, 60) (97, 91) (9, 32)

구조체의 활용_함수의 인자로 구조체 전달

❖ 구조체는 참조에 의한 호출로 전달하는 것이 좋다.

- ① 함수의 매개변수는 구조체 포인터형으로 선언한다.

```
void print_content(struct content* ptr);
```

- ② 구조체 변수가 입력 매개변수일 때는 `const`를 지정한다.

```
void print_content(const struct content* ptr);
```

- ③ 구조체를 매개변수로 갖는 함수를 호출할 때는 구조체 변수의 주소를 인자로 전달한다.

```
struct content c1 = { "Avengers", 11000, 8.8 };
print_content(&c1);
```

구조체의 활용_함수의 인자로 구조체 전달

- ❖ 구조체는 참조에 의한 호출로 전달하는 것이 좋다.
④ 함수를 정의할 때는 구조체 포인터로 구조체의 멤버에 접근한다.

```
void print_content(const struct content* ptr)
{
    printf("%s, %d, %.1f\n", ptr->title, ptr->price, ptr->rate);
}
```

구조체의 활용_구조체의 멤버로 다른 구조체 사용

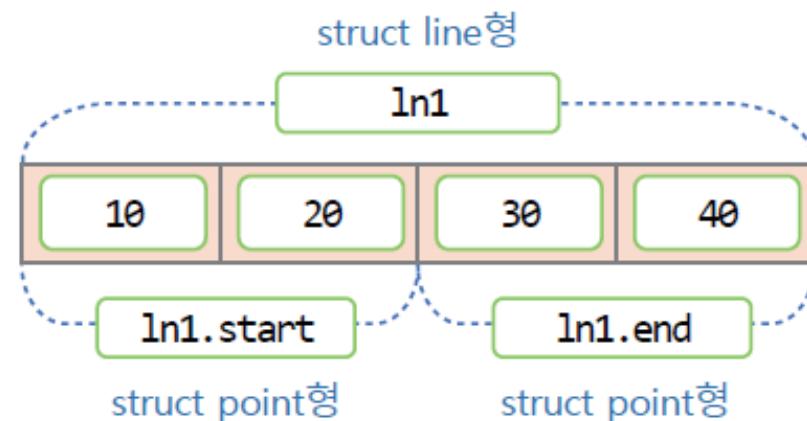
- ❖ 구조체 안에 다른 구조체 변수를 멤버로 포함할 수 있다.

```
struct line {  
    struct point start, end;  
};
```

line 구조체 정의 앞에 point 구조체가 정의되어 있어야 한다.

- ❖ line 구조체 변수를 선언하면 point 구조체 변수인 start와 end가 메모리에 할당된다.

```
struct line ln1 = {  
    {10, 20}, {30, 40}  
};
```



구조체의 활용_구조체의 멤버로 다른 구조체 사용

- ❖ line 구조체의 멤버에 접근하려면 멤버 접근 연산자 .를 이용 한다.

```
print_point(&ln1.start);
```

In1의 멤버인 start를 point 구조체 변수로 사용할 수 있다.

- ❖ 멤버의 멤버에 접근할 때 멤버 접근 연산자를 연속해서 사용 할 수 있다.

```
ln1.start.x = 100;
```

여러 번 연속해서 멤버에 접근할 수 있다.

```
double get_length(const struct line* ptr)
{
    int dx = ptr->end.x - ptr->start.x;
    int dy = ptr->end.y - ptr->start.y;
    return sqrt(dx * dx + dy * dy);
}
```

구조체의 활용_구조체의 멤버로 다른 구조체 사용

❖ line 구조체의 정의 및 사용(1/3)

```
01 #include <stdio.h>
02 #include <math.h>
03
04 struct point {
05     int x, y;
06 };
07 struct line {
08     struct point start, end;
09 };
10 void print_point(const struct point* ptr);
11 double get_length(const struct line* ptr);
12
```

point가 line보다 앞에 정의되어야 한다.

구조체의 활용_구조체의 멤버로 다른 구조체 사용

❖ line 구조체의 정의 및 사용(2/3)

```
13 int main(void)
14 {
15     struct line ln1 = {
16         {10, 20}, {30, 40}
17     };
18     printf("직선 정보: ");
19     print_point(&ln1.start);
20     print_point(&ln1.end);
21     printf("\n길이: %f\n", get_length(&ln1));
22 }
```

ln1의 start, end 멤버를 point 구조체 변수로 사용할 수 있다.

구조체의 활용_구조체의 멤버로 다른 구조체 사용

❖ line 구조체의 정의 및 사용(3/3)

```
24 void print_point(const struct point* ptr)
25 {
26     printf("(%d, %d) ", ptr->x, ptr->y);
27 }
28
29 double get_length(const struct line* ptr) ----- ptr은 입력 매개변수
30 {
31     int dx = ptr->end.x - ptr->start.x;
32     int dy = ptr->end.y - ptr->start.y;
33     return sqrt(dx * dx + dy * dy);
34 }
```

ptr은 입력 매개변수

여러 번 연속해서 멤버에 접근할 수 있다.

실행 결과

직선 정보: (10, 20) (30, 40)
길이: 28.284271

구조체의 활용_구조체의 멤버로 다른 구조체 사용

❖ line 구조체의 정의 및 사용(3/3)

```
24 void print_point(const struct point* ptr)
25 {
26     printf("(%d, %d) ", ptr->x, ptr->y);
27 }
28
29 double get_length(const struct line* ptr) ----- ptr은 입력 매개변수
30 {
31     int dx = ptr->end.x - ptr->start.x;
32     int dy = ptr->end.y - ptr->start.y;
33     return sqrt(dx * dx + dy * dy);
34 }
```

ptr은 입력 매개변수

여러 번 연속해서 멤버에 접근할 수 있다.

실행 결과

직선 정보: (10, 20) (30, 40)
길이: 28.284271

열거체와 공용체_열거체

❖ 열거체

- 정수형의 일종으로 열거형이라고도 한다.
- 정수형 변수가 특정 값들 중 한 가지 값을 가질 때 특정 값을 **열거 상수**로 정의할 수 있다.
- 열거체는 이런 열거 상수들로 정의되는 데이터형이다.
- 열거체도 일종의 사용자 정의형이다.

열거체와 공용체_열거체

❖ 열거체의 정의(1/2)

형식 enum 태그명 {열거상수1, 열거상수2, …};

사용예

```
enum color {red, green, blue};  
enum direction {  
    north, south, east, west  
};
```

- C 컴파일러는 열거체를 int형으로 처리하고, 열거 상수를 정수형 상수로 정의한다.

```
enum direction { north, south, east, west };
```

- 열거 상수를 추가로 정의하려면, { }의 끝부분에 이름만 추가하면 된다.

```
enum direction { north, south, east, west, northeast, northwest };
```

열거체와 공용체_열거체

❖ 열거체 정의(2/2)

- 열거 상수를 특정값으로 정의하려면, 열거 상수 이름 다음에 =을 쓰고 열거 상수의 값을 써준다.

```
enum direction { north = 0x00, south = 0x01, east = 0x10, west = 0x11 };
```

- 열거 상수 중 일부에만 값을 지정하면, 나머지 열거 상수는 지정된 값보다 1씩 커지는 정수값으로 자동으로 설정된다.

```
enum direction { north = 1, south, east, west };
```

south, east, west는
2, 3, 4로 정의된다.

열거체와 공용체_열거체

❖ 매크로 상수와 열거 상수

int 변수와 매크로 상수를 사용하는 경우

```
#define NORTH  
#define SOUTH  
#define EAST  
#define WEST  
#define NORTHEAST  
  
int main(void)  
{  
    int d1 = NORTH;  
    d1 = EAST;  
    :  
}
```

int형 변수를 사용한다.

0
1
2
3
4

값을 직접 정의 해야 한다.
겹치지 않는 값을 직접 지정해야 한다.

열거체와 열거 상수를 사용하는 경우

```
enum direction {  
    north, south, east, west,  
    northeast  
};  
  
int main(void)  
{  
    enum direction d1 = north;  
    d1 = east;  
    :  
}
```

자동으로 값이 할당된다.
열거 상수의 이름만 추가하면 된다.
열거체 변수를 사용한다.

열거체와 공용체_열거체

❖ 열거체의 사용

- 열거체도 'enum 태그명'으로 사용해야 한다.
- 열거체 변수는 int형으로 처리되므로 열거체 변수의 값을 출력하면 정수로 출력된다.

열거 상수 east의
정수값이 출력된다.

```
enum direction d1 = south;  
d1 = east;  
printf("%d", d1);
```

열거체 변수를 열거
상수로 초기화하거나
대입한다.

- 열거체의 태그명을 생략하고 열거 상수만 정의할 수 있다.

```
enum { north, south, east, west };
```

열거체와 공용체_열거체

❖ 열거체의 정의 및 사용 예(1/2)

```
01 #include <stdio.h>
02
03 enum direction { north, south, east, west };
04
05 int main(void)
06 {
07     enum direction moves[] = {
08         north, north, east, south, south, west,
09     };
10     int size = sizeof(moves) / sizeof(moves[0]);
11     int i;
12
13     printf("이동 순서: ");
14     for (i = 0; i < size; i++) {
15         switch (moves[i]) {
```

열거체와 열거 상수
정의

열거체 배열

moves[i] 열거체 변수의
값에 따라 처리

열거체와 공용체_열거체

❖ 열거체의 정의 및 사용 예(2/2)

```
16     case north:  
17         printf("북 ");  
18         break;  
19     case south:  
20         printf("남 ");  
21         break;  
22     case east:  
23         printf("동 ");  
24         break;  
25     case west:  
26         printf("서 ");  
27         break;  
28     }  
29 }  
30 printf("\n");  
31 }
```

열거 상수도 정수형
상수이므로 case에
사용할 수 있다.

실행 결과

이동 순서: 북 북 동 남 남 서

열거체와 공용체_공용체

❖ 공용체

- 여러 멤버들이 메모리를 공유해서 사용하는 기능
 - 한 멤버의 값을 변경하면 다른 멤버들의 값이 함께 변경된다.
- 공용체의 멤버들은 모두 같은 주소에 할당된다.

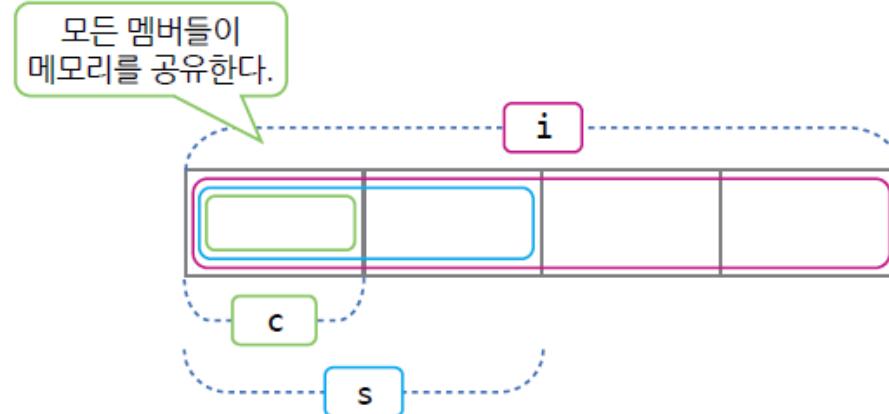
형식

```
union 태그명 {  
    데이터형 멤버명;  
    데이터형 멤버명;  
    ...  
};
```

사용예

```
union color_t {  
    unsigned int dword;  
    unsigned char rgb[4];  
};
```

```
union test {  
    int     i;  
    char   c;  
    short  s;  
};  
union test t1;
```



열거체와 공용체_공용체

❖ 공용체

- 공용체의 크기는 멤버 중 가장 큰 멤버의 크기와 같다.

```
printf("%d", sizeof(union test));
```

가장 큰 멤버 i의
크기와 같다.

- 공용체를 초기화할 때는 {} 안에 첫 번째 멤버의 초기값만 지정

```
union test t1 = { 0x12345678 };
```

t1.i를 초기화한다.

- 공용체의 멤버에 접근할 때도 멤버 접근 연산자인 .를 사용한다. 공용체 포인터로 멤버에 접근할 때는 간접 멤버 접근 연산자인 ->를 사용한다.

열거체와 공용체_공용체

❖ test 공용체의 정의 및 사용 예(1/2)

```
01 #include <stdio.h>
02
03 union test {
04     int     i;
05     char    c;
06     short   s;
07 };
08
09 int main(void)
10 {
11     union test t1 = { 0x12345678 };
12 }
```

공용체의 정의

t1의 모든 멤버가 같은 주
소에 할당된다.
초기화할 때 첫번째 멤버
의 초기값을 지정한다.

열거체와 공용체_공용체

❖ test 공용체의 정의 및 사용 예(2/2)

```
13 printf("t1.i의 주소 = %p\n", &t1.i);
14 printf("t1.c의 주소 = %p\n", &t1.c);
15 printf("t1.s의 주소 = %p\n", &t1.s);
16
17 printf("sizeof(union test) = %d\n", sizeof(union test));
18
19 printf("t1.i = %x\n", t1.i);
20 printf("t1.c = %x\n", t1.c);
21 printf("t1.s = %x\n", t1.s);
22 }
```

가장 큰 멤버의 크기

멤버들의 주소가
모두 같다.

실행 결과

```
t1.i의 주소 = 00F2FE88
t1.c의 주소 = 00F2FE88
t1.s의 주소 = 00F2FE88
sizeof(union test) = 4
t1.i = 12345678
t1.c = 78
t1.s = 5678
```

열거체와 공용체_공용체

❖ **typedef(1/2)**

- **typedef**를 이용하면 기존의 데이터형에 대한 별명을 만들 수 있다.

형식

typedef 기존데이터형 새이름;

사용예

```
typedef struct point point_t;  
typedef unsigned int uint_t;
```

- 구조체를 정의할 때 **typedef**를 이용해서 'struct 태그명' 대신 사용될 이름을 준비해두면 사용하기 편하다.

```
struct point {  
    int x, y;  
};  
typedef struct point point_t;
```

point_t는 struct point에
대한 별명

열거체와 공용체_공용체

❖ **typedef(2/2)**

- 구조체를 정의하면서 **typedef**를 함께 정의할 수도 있다.

```
typedef struct point {  
    int x, y;  
} point_t;
```

- **typedef**를 정의한 다음에도 기존의 데이터형을 그대로 사용할 수 있다.

```
point_t pt1 = { 10, 20 };  
struct point *ptr = &pt1;
```

point_t와 struct point
둘 다 사용할 수 있다.

열거체와 공용체_공용체

❖ **typedef**의 정의 및 사용

```
01 #include <stdio.h>
02
03 typedef struct point {
04     int x, y;
05 } point_t;
06 void print_point(const point_t* ptr);
07
08 int main(void)
09 {
10     point_t pt1 = { 10, 20 };
11     print_point(&pt1);
12 }
13
14 void print_point(const point_t* ptr)
15 {
16     printf("%d, %d) ", ptr->x, ptr->y);
17 }
```

실행 결과

(10, 20)

struct point 대신
point_t형을 사용
할 수 있다.