C 프로그래밍 및 실습

11. 구조체

세종대학교

목차

1) 구조체 개요

- **학생들의 성적을 처리하는 프로그램을 고려해보자**
  - 정보: 학번, 학생 이름, 학생 평점, 과목 번호, 과목명
    - ✓ 학생 정보와 과목 정보로 나누어 관련 항목끼리 묶으면 편리
      - ✓ 학생: 학번, 이름, 평점
      - ✓ 과목: 번호, 이름
  - C언어에서는 이와 같은 목적을 위해 **구조체** 지원

```
/* 학생 정보 */
int student_id;
char student_name[8];
double student_grade;
```

```
struct student {
    int id;
    char name[8];
    double grade;
} st;
```

```
/* 과목 정보 */
int subject_id;
char subject_name[15];
```

```
struct subject {
    int id;
    char name[15];
} sub;
```

1) 구조체 개요

- **구조체(structure)**
  - 의미상 관계가 있는 항목을 그룹으로 **묶어** 표현한 자료형
    - ✓ 구조체는 int, char와 같이 변수의 모양을 의미
    - ✓ 차이는 int, char는 기본적으로 정해져 있는 기본 자료형이고, 구조체는 사용자가 용도에 맞게 만들어 사용하는 **사용자 자료형**
  - 구조체를 구성하는 변수들을 **멤버** 라고 부름

```
/* 학생 정보 */
int student_id;
char student_name[8];
double student_grade;
```

```
struct student {
    int id;
    char name[8];
    double grade;
} st;
```

  - 주의!! 구조체는 개념과 문법적 성질은 배열(동일한 정보의 단순 모임)과 매우 다르다.

2) 구조체의 정의, 선언, 사용

- **구조체 (자료형) 정의**
  - 자료형(변수의 **모양**)을 정의하는 것 - 변수 선언과는 다른 개념
  - struct이라는 키워드를 사용
  - 일반적인 구조체 정의 형식

```
struct 구조체자료형이름{
    멤버자료형 멤버변수;
    멤버자료형 멤버변수;
    ...
} ;     // 세미콜론
```

(예시)

```
struct student{
    int id;
    char name[8];
    double grade;
} ;
```

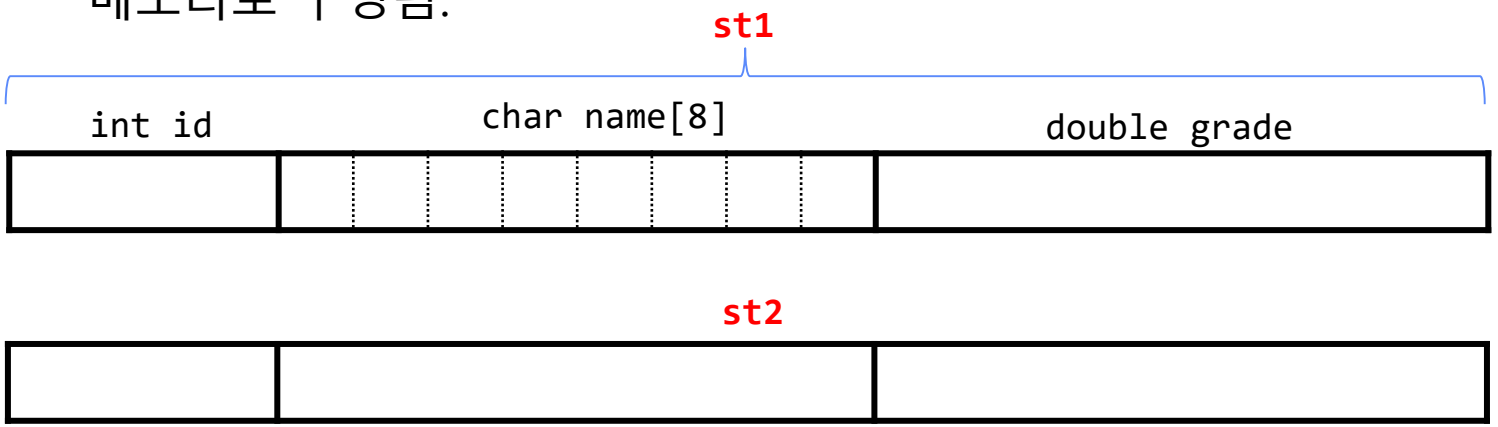
  - 정의만 해서는 메모리에 공간이 할당되지 않음

2) 구조체의 정의, 선언, 사용

- **구조체 (변수) 선언**
  - 일반적인 변수 선언과 동일한 형태
    - ✓ **자료형 변수명;**
    - ✓ 구조체에서는 'struct 구조체자료형이름'이 통째로 자료형을 나타냄

```
struct student st1, st2;   // 구조체 변수 선언
```
자료형 변수

  - 변수 선언을 해야 비로소 메모리에 공간이 할당됨
    - ✓ 구조체 변수 자체는 메모리 공간을 따로 가지지 않고, 멤버들의 메모리로 구성됨.



2) 구조체의 정의, 선언, 사용

- **구조체 변수 초기화**
  - 다른 초기화와 유사하게 중괄호 안에 멤버 변수 순서대로 초기화 값 나열

```
struct student st1 = { 10, "Tom", 3.2};
```

st1

10	"Tom"	3.2

2) 구조체의 정의, 선언, 사용

- **구조체 멤버 변수 사용**
  - 구조체 멤버 연산자 (.) 사용 : **구조체변수.멤버변수**
  - 비교) 구조체 변수는 멤버 변수 전체를 나타냄

  - 멤버 변수의 자료형에 의해 사용방법이 결정됨
    - ✓ int형 멤버 변수 id는 일반적인 int형 변수를 사용하는 것과 동일한 방식으로 사용

```
struct student st1, st2;    // 구조체 변수 선언

st1.id = 10;                // int형 변수 대입
st1.id = st1.id * 2;        // int형 변수 연산
printf("id: %d", st1.id);   // int형 변수 출력
```

2) 구조체의 정의, 선언, 사용

- **프로그램 1: 구조체를 사용하는 전형적인 프로그램**

```
struct student{      // 구조체 student 정의는
   int id;           // 함수 밖에서
   char name[8];
   double grade;
};

void main( ) {
   struct student st1 = {10, "Tom", 3.2};  // 변수 선언 및 초기화

   st1.id += 20;                  // 구조체 멤버 참조
   strcpy(st1.name, "alice");     // 주의: st1.name = "alice" (X)
   st1.name[0] = 'A';

   printf("id: %d\n", st1.id);            // 정수 출력
   printf("name: %s\n", st1.name);        // 문자열 출력
   printf("grade: %.2f\n", st1.grade);    // 부동소수 출력
}
```

실행 결과

```
id: 30
name: Alice
grade: 3.20
```

2) 구조체의 정의, 선언, 사용

- **[예제 11.1] (기본형 → 구조체로 확장) 주 메뉴 1개, 부 메뉴 3개, 음료 1개의 값을 입력 받고, 각 항목의 값과 총합을 출력하는 프로그램을 작성하시오.**

실행 예시
(붉은 색은 사용자 입력)

```
Main dish: 30
Side dish 1: 3
Side dish 2: 5
Side dish 3: 0
Beverage: 10

Total: 30 + 3 + 5 + 0 + 10 = 48
```

1. (기본형 – 구조체 사용 하지 않음) 변수 maindish, sidedish[3], beverage를 선언, 입력, 출력

2. (구조체로 확장) 주 메뉴, 부 메뉴, 음료가 하나의 런치 박스 안에 들어 있는 것으로 간주해, 구조체를 선언해서 입력 받고 출력하시오.
   - maindish, sidedish[3], beverage를 묶은 **구조체 변수 box** 선언

2) 구조체의 정의, 선언, 사용

- **구조체 정의와 선언의 다양한 형태 1**
  - 자료형 정의와 변수 선언을 **따로** (가장 일반적인 형태)

```
struct student{      // 구조체 자료형 정의 (함수 밖에서)
   int id;   char name[8];   double grade;
};

void func(){
   struct student st1;  // 지역 변수 st1 선언
}
```

2) 구조체의 정의, 선언, 사용

- **구조체 정의와 선언의 다양한 형태 2**
  - 자료형 정의와 변수 선언을 **동시에**
    - ✓ **student** : 구조체 자료형 이름, **st** : 구조체 변수 이름

```
struct student{      // 구조체 자료형 정의
   int id;   char name[8];   double grade;
} st;              // 전역 변수 st 선언

void func(){
   struct student st1;  // 지역 변수 st1 선언
}
```

- ✓ 정의, 선언과 동시에 초기화도 가능

```
struct student{      // 구조체 자료형 정의
   int id;   char name[8];   double grade;
} st = {10, "Tom", 3.2} ;  // 전역 변수 선언 및 초기화
```

2) 구조체의 정의, 선언, 사용

- **구조체 정의와 선언의 다양한 형태 3**
  - 구조체 자료형 이름 생략 가능
  - 다만, 자료형 이름이 없으므로, 다른 곳에서 선언 불가능

```
struct student{      // 구조체 자료형 정의 (이름 없음)
   int id;   char name[8];   double grade;
} st;              // 구조체 변수 st 선언 (가능)

void func(){
   struct student st1;   // 컴파일 오류
   struct    st2;        // 컴파일 오류
   ...
}
```

2) 구조체의 정의, 선언, 사용

- **구조체 정의와 선언의 다양한 형태 4**
  - 구조체 자료형을 함수 안에서 정의 하면?
    - ✓ 정의한 함수 안에서만 사용 가능

```
void func1(){
    struct student{      // 구조체 자료형 정의 (함수 안)
        int id;   char name[8];   double grade;
    };
    struct student st1;  // 구조체 변수 선언
    ...
}
void func2(){
    struct student st2;  // 컴파일 오류
    ...
}
```

2) 구조체의 정의, 선언, 사용

- **구조체에 사용 가능한 연산자**
  - 구조체는 사용자가 만든 자료형이기 때문에
    기본 자료형인 int, char, double 등에 비해
    사용 가능한 연산자가 제한적임
    - ✓ 예) 산술 연산, 비교 연산 등은 지원 안 됨
      - ✓ st1 + st2 : 구조체끼리 더하라? 의미적으로 불명확
      - ✓ st1 < st2 : 구조체끼리 비교하라? 의미적으로 불명확
  - 구조체 변수에 사용 가능한 연산자
    - ✓ 대입연산자(=), 주소연산자(&), 간접참조 연산자(*),
      sizeof 연산자 정도…

2) 구조체의 정의, 선언, 사용

- **구조체 변수의 대입 연산**
  - 모든 멤버 변수에 대해, 대입 연산이 수행됨

```
struct student st1 = { 10, "Tom", 3.2};
struct student st2;

st2 = st1;
printf("id: %d\n", st2.id);
printf("name: %s\n", st2.name);
printf("grade: %.2f\n", st2.grade);
```

실행 결과

```
id: 10
name: Tom
grade: 3.20
```

배열의 경우에도 name[0]부터 name[7]까지,
각 원소마다 단순 대입 (strcpy가 호출되는 것이 아님)

2) 구조체의 정의, 선언, 사용

- **[예제 11.2] 예제 11.1의 런치 박스 구조체 사용하여 다음 프로그램을 작성하시오.**
  - ① 두 개의 런치 박스 A와 B선언
  - ② 런치 박스 A의 각 항목의 가격을 사용자로부터 입력 받기
  - ③ 런치 박스 A의 정보를 B에 복사
  - ④ B의 주메뉴 가격을 사용자로부터 입력 받아 바꾸기
  - ⑤ A와 B의 가격 정보를 화면에 출력

2) 구조체의 정의, 선언, 사용

- **구조체에 할당되는 메모리 크기**
  - 구조체 크기는 구조체 멤버 크기의 합? 아닐 수도 있다.
    - ✓ 4 (int) + 8 (char 배열) + 8 (double) = 20 bytes 일까?

```
struct student{
   int id;   char name[8];   double grade;
};

void main( )
{  printf("%d ", sizeof( int ) );
   printf("%d ", sizeof( char[8]) );
   printf("%d ", sizeof( double ) );
   printf("%d ", sizeof( struct student ) );
}
```

결과:

```
4 8 8 24
```

→ **구조체의 크기가 필요한 경우 sizeof 연산자를 사용하자.**
(다른 자료형도 마찬가지)

목차

3) 구조체 배열

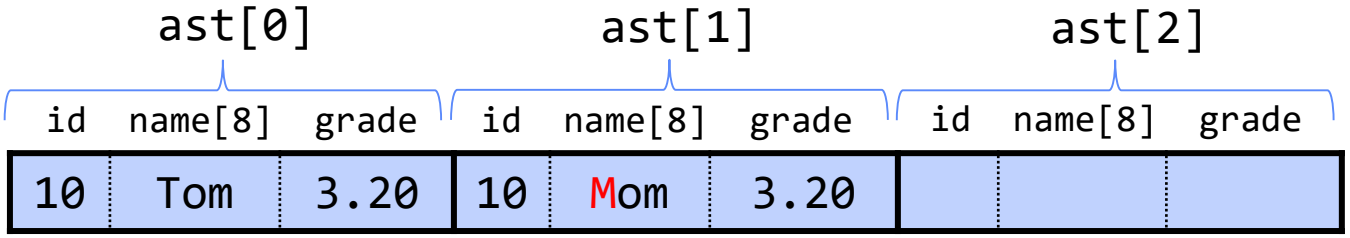
- **구조체가 배열, 포인터, 함수 등과 결합되어 확장될 수 있음**
  - 구조체 배열(3절), 구조체 포인터(4절), 구조체와 함수(5절)
  - 대부분 기존에 학습한 문법을 그대로 적용하여 확장

- **구조체 배열**
  - 구조체가 원소로 사용된 배열
    - ✓ 단, 같은 구조체끼리의 묶음만 허용
  - 선언, 접근, 초기화 등 구조체 배열에 대한 문법
    - ✓ 일반 배열과 동일

3) 구조체 배열

- **구조체 배열 선언과 접근:** [ ] 사용

```
struct student ast[3];   // 구조체 배열 선언

ast[0].id = 10;
strcpy(ast[0].name , "Tom");
ast[0].grade = 3.2;

ast[1] = ast[0];                    // 구조체 대입
ast[1].name[0] = 'M';
```



3) 구조체 배열

- **구조체 배열 초기화:** 중괄호 { } 이용

```
int i;
struct student ast[3]
    = { { 10, "Tom", 3.2},
        { 20, "Alice"} };
  // 생략된 부분은 모두 0으로 초기화

for( i= 0 ; i < 3 ; ++i )    // 반복문을 이용한 배열 출력
   printf("%d: %d, %s, %.2f\n",
           i, ast[i].id, ast[i].name, ast[i].grade);
```

실행 결과

```
0: 10, Tom, 3.20
1: 20, Alice, 0.00
2: 0, ,0.00
```

3) 구조체 배열

- **[예제 11.3] 복소수 구조체 배열**
  - 복소수를 구조체로 표현하고, 크기가 3인 복소수 구조체 배열을 선언한 후(0번, 1번 복고수는 아래와 같이 초기화), 0번과 1번 복소수의 결과를 2번 복소수에 대입하는 프로그램을 작성하시오.
    - ✓ 0번 복소수: 1.2 + 2.0i
    - ✓ 1번 복소수: -2.2 - 0.3i

```
struct complex {          // 구조체 자료형 선언
    double real, imag;
};
```

실행 결과

```
0: 1.2 + 2.0i
1: -2.2 + -0.3i
2: -1.0 + 1.7i
```

3) 구조체 배열

- [예제 11.4] 예제 11.1에서 정의한 구조체'의 '배열'을 이용하여 2개의 런치 박스의 정보를 사용자로부터 입력 받고 출력하는 프로그램을 작성하시오.

목차

4) 구조체 포인터

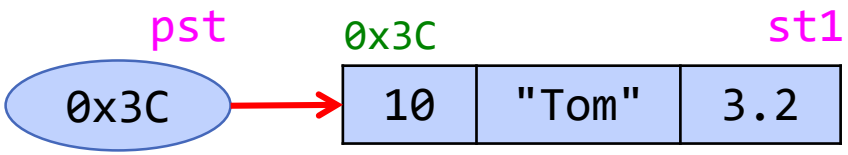
- **구조체 포인터**
  - 구조체 변수를 가리키는 포인터
  - 즉, 구조체 변수의 시작 주소가 저장

- **기본적인 사용법은 일반적인 포인터와 동일**
  - 다만, 구조체 포인터에서만 사용하는 표현법 존재

4) 구조체 포인터

- **구조체 포인터 변수 선언 및 연결**
  - 일반 포인터 선언과 동일 : * 사용
  - 주소 연산자(&): 구조체 변수의 시작 주소

```
struct student st1 = { 10, "Tom", 3.2};
struct student *pst; // 구조체 포인터 변수 선언
pst = &st1;    // 연결
```



4) 구조체 포인터

- **간접 연산자(*)를 이용한 구조체 변수 접근**

```
struct student st1 = { 10, "Tom", 3.2}, st2;
struct student *pst = &st1; // 선언 및 연결

st2 = *pst; // st2에 pst가 가리키는 구조체 변수를 대입

printf("%d, %s, %.2f\n", st1.id, st1.name, st1.grade);
printf("%d, %s, %.2f\n", st2.id, st2.name, st2.grade);
```

실행 결과

```
10,Tom,3.20
10,Tom,3.20
```

4) 구조체 포인터

- **간접 참조를 이용한 구조체 변수의 멤버 접근 (방법 1)**
  - 포인터가 가리키는 변수에 접근하기 위해 간접연산자 * 사용
  - 구조체의 멤버에 접근하기 위해 멤버 연산자 . 사용

```
struct student st1, *pst = &st1;

(*pst).id = 20;   // pst가 가리키는 구조체의
                  // 멤버 id에 20 대입

printf("id: %d\n", (*pst).id + 15 );
```

실행 결과

```
id: 35
```

  - * 보다 . 의 우선순위가 높기 때문에 괄호 반드시 필요

4) 구조체 포인터

- **간접 참조를 이용한 구조체 변수의 멤버 접근 (방법2)**
  - 구조체 포인터에서만 사용하는 전용 연산자: **->**
  - 주로 이 연산자를 사용함

```
(*pst).id = 20;   // pst가 가리키는 구조체의
                  // 멤버 id에 20 대입
```

⏸

```
pst->id = 20;     // pst가 가리키는 구조체의
                  // 멤버 id에 20 대입
```

4) 구조체 포인터

- **프로그램 1(p. 9)의 포인터 버전**
  - 간접 참조로만 구조체에 접근하도록 수정한 코드

```
struct student st1 = {10, "Tom", 3.2};  // 변수 선언 및 초기화
struct student *pst = &st1;             // 포인터 선언 및 연결

pst->id += 20;   // pst가 가리키는 구조체의 멤버 id에 접근
strcpy(pst->name, "alice");      // 멤버 name에 접근
pst->name[0] = 'A';              // 멤버 name[0]에 접근

printf("id: %d\n", pst->id);
printf("name: %s\n", pst->name);
printf("grade: %.2f\n", pst->grade);
```

4) 구조체 포인터

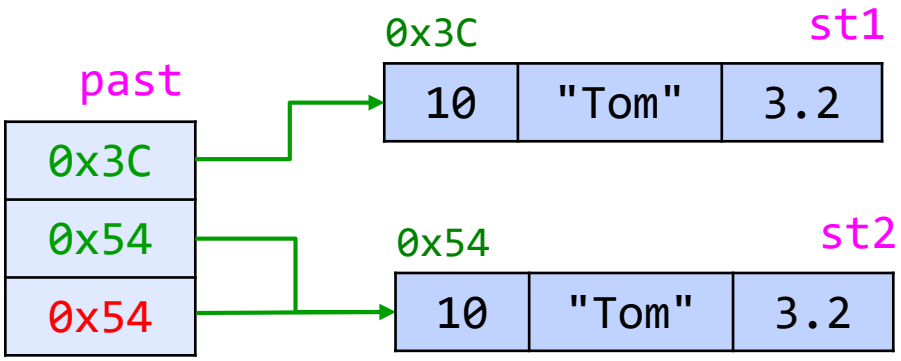
- **[예제 11.5] 예제 11.1을 구조체 포인터를 사용하여 작성하시오.**
  - 구조체 변수에 직접 접근하지 말고, 포인터를 이용하여 간접 접근할 것

```
/* 참고 */
struct lunchbox{
    int maindish
    int sidedish[3];
    ...
};

int main() {
    struct lunchbox box, *p = &box;

    scanf("%d", &p->maindish);
    scanf("%d", &p->sidedish[0]);
    ...
    printf("%d\n", p->maindish);
    printf("%d\n", p->sidedish[0]);
    ...
}
```

4) 구조체 포인터

- **구조체 포인터 배열**
  - 구조체 포인터가 원소인 배열

```
struct student st1 = { 10, "Tom", 3.2 }, st2;
struct student *past[3] = { &st1, &st2 }; // 구조체 포인터 배열

past[2] = past[1];        // 주소 값 대입: past[2]도 st2를 가리킴
*past[2] = *past[0];      // past[2]가 가리키는 구조체에
                             past[0]이 가리키는 구조체 대입
```



목차

5) 구조체와 함수

- **구조체를 함수의 인자로 사용**
  - 실인자의 값이 형식인자에 대입됨
    - ✓ 구조체 변수라고 달라지는 건 없음

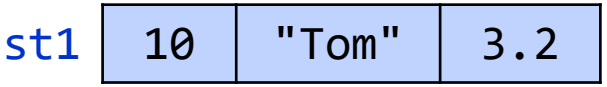
```
void print(struct student st) {
    printf("id: %d\n", st.id);
    printf("name: %s\n", st.name);
    printf("grade: %.2f\n", st.grade);
}

int main() {
    struct student st1 = {10, "Tom", 3.2};
    print(st1);
    return 0;
}
```

구조체의 값 대입

실행 결과

```
id: 10
name: Tom
grade: 3.20
```

print의 변수



main의 변수

5) 구조체와 함수

- **구조체를 함수의 반환형으로 사용**
  - 구조체 전체 값이 통째로 호출함수에게 전달됨

```
struct student init( ) {
    struct student st = { 0, "", 0};
    return st;  // 구조체 변수 st의 값 반환
}
int main() {
    struct student st1 = {10, "Tom", 3.2};

    printf("%d, %s, %.2f\n", st1.id, st1.name, st1.grade);
    st1 = init();  // init()의 반환값 대입
    printf("%d, %s, %.2f\n", st1.id, st1.name, st1.grade);
    return 0;
}
```

실행 결과

```
10, Tom, 3.20
0,,0.00
```

5) 구조체와 함수

- **[예제 11.6] 두 복소수를 입력 받아, 두 복소수의 합을 출력하는 프로그램을 작성하시오.**
  - 구조체 complex 사용 (예제 11.3, p.25 참고)
  - add_complex() 함수를 정의하여 사용할 것
    - ✓ 구조체 complex형 변수 두 개를 인자로 받아, 두 복소수의 덧셈 결과 (구조체 complex형)를 반환
  - main() 함수
    - ✓ 사용자로부터 복소수의 정보를 입력 받아 구조체 변수에 저장
    - ✓ add_complex 함수를 호출하여 덧셈 결과 얻기
    - ✓ 덧셈 결과 출력

5) 구조체와 함수

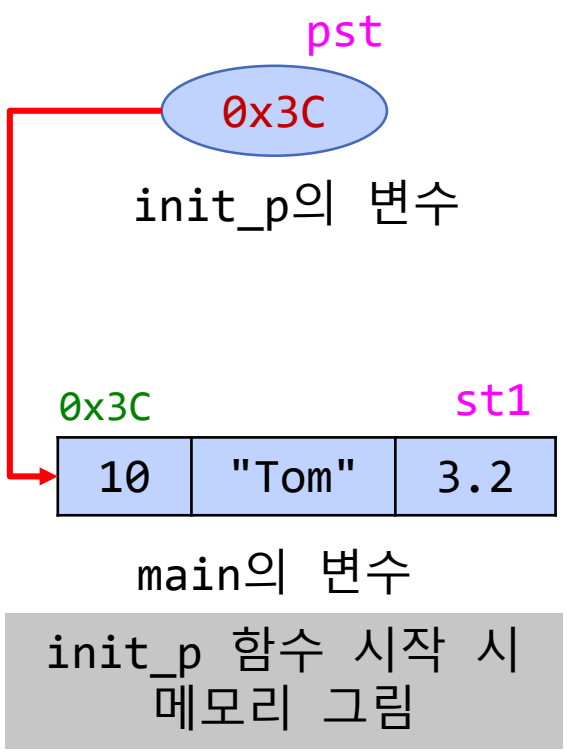
- **구조체 포인터 변수를 함수 인자로 사용**
  - 실인자의 값(주소)이 형식인자에 대입됨

```
void init_p(struct student *pst) {
    pst->id = 0;
    pst->name[0] = '\0';
    pst->grade = 0.0;
}
int main() {
    struct student st1 = {10, "Tom", 3.2};
    init_p( &st1 );
    printf("%d, %s, %.2f\n", st1.id,
            st1.name, st1.grade);
    return 0;
}
```

구조체의 주소 대입

실행 결과

```
0, , 0.00
```



init_p 함수 시작 시 메모리 그림

5) 구조체와 함수

- **구조체 주소를 반환하는 함수**
  - 일반 변수의 주소 반환과 동일

```
struct student *next_addr(struct student *pst) {
  return pst+1;
}
int main() {
   struct student st[2] = {{10, "Tom", 3.2},
                           {20, "Ann", 3.5}};
   struct student *p;

   p = next_addr(st);          // 배열의 이름(주소) 전달

   printf("%d, %s, %.2f\n", p->id, p->name, p->grade);
   return 0;
}
```

실행 결과

```
20, Ann, 3.50
```

5) 구조체와 함수

- **[예제 11.7] 두 복소수를 입력 받아, 절대값이 큰 복소수를 출력하는 프로그램을 작성하시오.**
  - 구조체 complex 사용 (예제 11.3, p.25 참고)
  - larger_complex() 함수를 정의하여 사용할 것
    - ✓ 구조체 complex형 포인터 변수 두 개를 인자로 받아, 절대값의 제곱이 큰 복소수의 포인터를 반환
    - ✓ 참고) 복소수 a+bi의 절대값의 제곱은 a^2+b^2
  - main() 함수
    - ✓ 사용자로부터 복소수의 정보를 입력 받아 구조체 변수에 저장
    - ✓ larger_complex 함수를 호출하여 절대값이 큰 복소수의 주소 얻기
    - ✓ 절대값이 큰 복소수 출력

5) 구조체와 함수

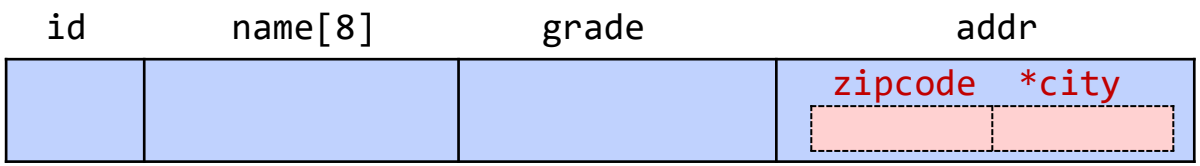
- **함수 호출 (총 정리)**
  - 함수 인자나 반환 값의 자료형에 관계없이 함수 호출 과정은 동일
  - 함수 호출 시에는 **전달된 값을 형식인자에 대입**
    - ✓ 차이가 나는 이유는 전달된 값이
      - ✓ 정수, 문자를 나타내느냐
      - ✓ 주소를 나타내느냐에 따른
      - ✓ **부수적인 효과일 뿐...**
  - 함수 종료 시에는 리턴 값이 통째로 호출함수에 전달
  - **(결론)**
    - ✓ **자료형(int, 포인터, 구조체, 배열)마다 따로 외우려고 하지 말고 원리를 이해하자!!!!**

목차

6) 중첩 구조체 및 자기참조 구조체

- **중첩 구조체(nested structure)**
  - 다른 **구조체**가 구조체의 멤버로 사용
  - 중첩 구조체 정의 예

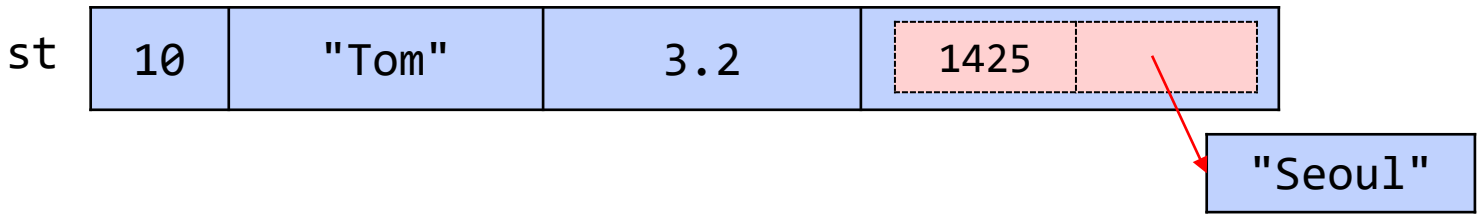
```
struct address {   // 구조체 address 정의
   int zipcode;
   char *city;
};
struct student5 {   // 구조체 student 정의
   int id;   char name[8];   double grade;
   struct address addr;  // 멤버 addr의 자료형은 struct address
};
```



6) 중첩 구조체 및 자기참조 구조체

- 중첩 구조체 사용 예

```
struct student5 st = {10, "Tom", 3.2, {1425, "Seoul"}};

st.addr.zipcode = 7189;          // 중첩 구조체 멤버 접근
st.addr.city = "Incheon";

printf("id: %d\n", st.id);
printf("name: %s\n", st.name);
printf("grade: %.2f\n", st.grade);
printf("zipcode: %d\n", st.addr.zipcode);
printf("city: %s\n", st.addr.city);
```



6) 중첩 구조체 및 자기참조 구조체

- **(주의) 자신과 동일한 구조체 자료형은 멤버로 사용될 수 없음**
  - **왜?** 자기 자신을 정의하기 위해 자기 자신이 필요 (순환 오류..)
    - ✓ 아래 구조체의 메모리 그림을 그릴 수 있는가?

```
struct student6{    // 구조체 student 정의
   int id;
   char name[8];
   double grade;
   struct student6 roommate;  // 컴파일 오류
   struct student6 friends[10]; // 컴파일 오류
};
```

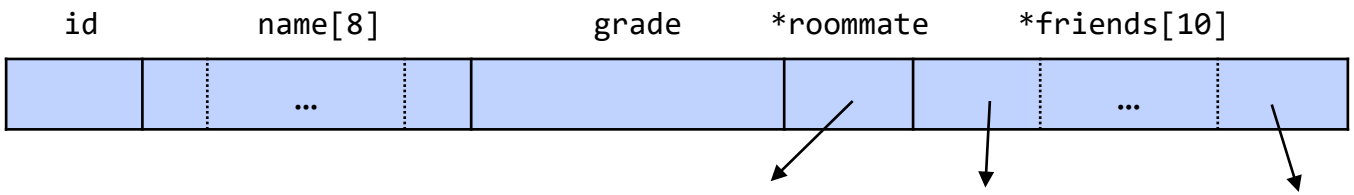
- ✓ 구조체 배열도 마찬가지

- ✓ 그런데, 구조체 포인터는? (다음 슬라이드)

6) 중첩 구조체 및 자기참조 구조체

- **자기참조 구조체**
  - 자신과 동일한 구조체의 "**포인터**"는 멤버로 가능
    - ✓ 아래에서 roommate는 포인터 변수로 주소 값을 저장하므로 struct student의 정의에 포함 가능 (메모리 그림 그릴 수 있음)

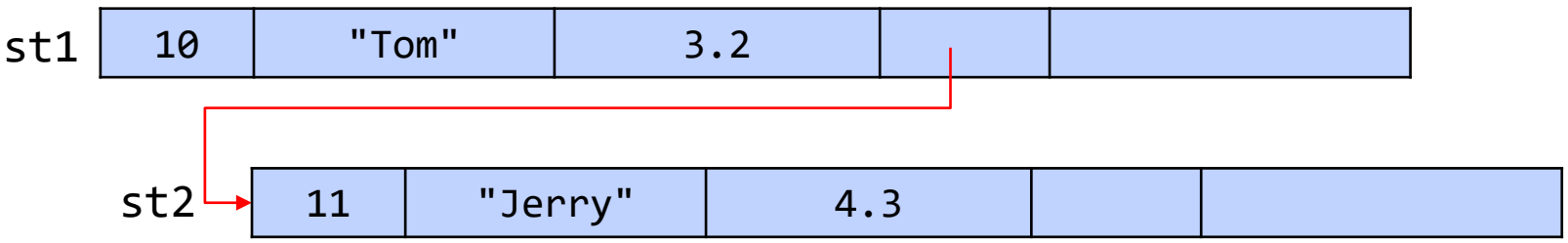
```
struct student7{    // 구조체 student 정의
    int id;
    char name[8];
    double grade;
    struct student7 *roommate;  // 문제 없음
    struct student7 *friends[10]; // 문제 없음
};
```



6) 중첩 구조체 및 자기참조 구조체

- 자기참조 구조체 사용 예

```
struct student7 st1 = {10, "Tom", 3.2};   // roommate와 friends
struct student7 st2 = {11, "Jerry", 4.3}; // NULL로 초기화

st1.roommate = &st2;     // st1의 룸메이트는 st2

printf("id: %d\n", st1.id);
printf("name: %s\n", st1.name);
printf("grade: %.2f\n", st1.grade);
printf("roommate's name: %s\n", st1.roommate->name);
```



6) 중첩 구조체 및 자기참조 구조체

- **[예제 11.8] struct student7을 사용하여 3명의 학생을 나타내는 변수 3개를 선언하고, 한 학생의 친구로 다른 두 명의 학생을 등록하라.**
  - 구조체 멤버는 적절히 초기화 or 값 대입
  - friends 멤버를 이용하여 친구의 정보 출력

실행 결과(예시)

```
000과 000는 000의 친구입니다.
```

목차

7) typedef 사용자 형정의

- **사용자 형정의**
  - typedef문을 이용하여 자료형의 이름을 새로 정의할 수 있음
  - 예

```
typedef int INT;   // INT 자료형 정의 (마지막에 세미콜론)
```

✓ int라는 자료형을 INT라는 이름으로 정의
✓ 주의!! 위 정의에서 INT는 변수 이름이 아니고 자료형 이름
✓ INT를 이용한 변수 선언

```
INT num;          // INT형 변수 num 선언
INT arr[5];       // INT형 배열 변수 arr 선언
INT *pi;          // INT형 포인터 변수 pi 선언
```

7) typedef 사용자 형정의

- 다양한 사용자 형정의의 예

```
typedef int INT;
typedef int * INTPTR;
typedef unsigned int AGE;
typedef unsigned int ID;
typedef unsigned char UCHAR;
typedef unsigned char * UCHARPTR;

AGE age1, age2;
ID id1, id2;
UCHARPTR p;      // unsigned char *p; 와 동일한 선언
```

- ✓ 위 예에서 AGE, ID 모두 unsigned int 이지만, 위와 같은 식으로 자료형에 의미 부여 가능
- ✓ 긴 이름을 간결하게 줄일 수 있음

7) typedef 사용자 형정의

- **구조체와 typedef**
  - 구조체의 경우 키워드 struct로 인해서 선언이 길어짐
  - typedef문을 이용하여 자료형의 이름을 재정의하여 프로그래밍의 간결성 확보

```
struct student{
   int id;  char name[8];  double grade;
};

typedef struct student STUDENT;  // 자료형 재정의

int main() {
   STUDENT st1 = {10, "Tom", 3.2};

   printf("id: %d\n", st1.id);        // 구조체 멤버의 값
   return 0;
}
```

7) typedef 사용자 형정의

- **구조체 재정의의 다양한 형태**
  - typedef와 구조체 정의를 하나로 묶기 (이 형태를 많이 사용)

```
typedef struct student{
   int id;   char name[8];   double grade;
} STUDENT;        // 자료형 정의
STUDENT st;       // STUDENT 형으로 변수 st 선언
```

```
typedef struct student{
   int id;   char name[8];   double grade;
} student;  // 구조체 이름과 사용자 자료형 이름이 같아도 됨
student st;       // student 형으로 변수 st 선언
```

```
typedef struct student{     // 구조체 이름 생략 가능
   int id;   char name[8];   double grade;
} student;
student st;       // student 형으로 변수 st 선언
```

7) typedef 사용자 형정의

- **주의!! type 사용자 정의와 구조체 정의 비교**
  - 모양이 유사하지만, typedef가 앞에 있느냐 없느냐에 따라 의미가 완전히 달라짐

```
typedef struct student{
   int id;   char name[8];   double grade;
} STUDENT;        //STUDENT는 자료형
```

```
struct student{
   int id;   char name[8];   double grade;
} st;             // st는 변수
```

7) typedef 사용자 형정의

- **[예제 11.9] 예제 11.1에서 정의한 구조체의 '배열'을 이용하여 2개의 런치 박스의 정보를 사용자로부터 입력 받고 출력하는 프로그램을 작성하시오. (예제 11.4와 동일한 문제)**
  - 단, "예제 11.1에서 정의한 구조체"를 typedef를 이용하여 사용자 자료형으로 정의할 것 (자료형 이름은 알아서 정하기)