

# 9장 사용자 정의형

김명호

# 내용

- 구조체
- 구조체 멤버 접근 연산자 .
- 구조체 포인터
- 구조체 멤버 접근 연산자 ->
- 구조체와 함수
- 공용체
- 공용체와 구조체
- 열거형

# 사용자 정의형

- 기본 자료형 이외에 사용자가 직접 선언하는 형
  - 배열
  - 구조체
  - 공용체
  - 열거형

# 구조체

- 서로 다른 형의 변수들을 하나로 묶어 주는 방법 제공
  - 이질적인 데이터 집합을 하나의 단위로 취급할 수 있게 함
- (예) 성적 처리 프로그램
  - 학생 이름과 그 학생 점수는 한 쌍으로 다루는 것이 좋음

# 구조체

- 학생 별 점수를 위한 구조체 선언

```
struct name_grade {  
    char   name[8];  
    int    grade;  
};
```

- struct : 구조체 선언을 위한 키워드
- name\_grade : 구조체 태그 이름
- name, grade : 구조체 멤버

# 구조체

- 구조체 선언

```
struct name_grade {  
    char   name[8];  
    int    grade;  
};
```

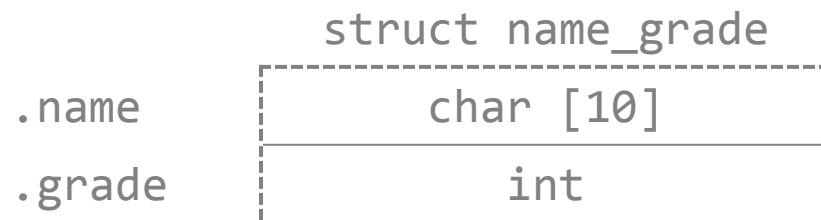
- 이 선언은 메모리 할당을 받는 변수를 선언한 것이 아니라 struct name\_grade **형**을 선언한 것임
- struct name\_grade는 자료형이 오는 자리에 사용할 수 있음

# 구조체

- 구조체 선언

```
struct name_grade {  
    char   name[8];  
    int    grade;  
};
```

- struct name\_grade 형은 크기가 12 바이트임



# 구조체 변수 선언

## • struct name\_grade 형 변수 선언

```
struct name_grade {
    char  name[8];
    int   grade;
};
```

```
struct name_grade  st_g1, st_g2;
```

- struct name\_grade가 자료형이기 때문에 이 코드는 st\_g1과 st\_g2라는 변수를 선언하는 선언문임
- 컴파일러는 st\_g1과 st\_g2 변수에 메모리 할당을 함

|        |          |
|--------|----------|
|        | st_g1    |
| .name  | char [8] |
| .grade | int      |

|        |          |
|--------|----------|
|        | st_g2    |
| .name  | char [8] |
| .grade | int      |



## 구조체 멤버 접근 연산자 .

- `st_g1`과 `st_g2`는 구조체 변수로 일반 변수처럼 다룰 수 없음

```
st_g1 = 10;    // 오류
```

- 같은 형의 구조체 변수간에는 배정이 가능함

```
st_g1 = st_g2;
```

- `st_g1`과 `st_g2`의 멤버는 일반 변수처럼 다룰 수 있음
- 구조체 변수의 멤버는 . 연산자로 접근할 수 있음
  - `st_g1.name` : `char [8]` 형 변수, `char` 형 배열
  - `st_g1.grade` : `int` 형 변수

# 구조체 멤버 접근 연산자 .

- 예제

```
strcpy(st_g1.name, "이순신");  
st_g1.grade = 98;
```

|        |       |
|--------|-------|
|        | st_g1 |
| .name  | “이순신” |
| .grade | 98    |

# 구조체 사용

## 프로그램 9.1

```
#include <stdio.h>
#include <string.h>
int main(void){
    struct name_grade{
        char name[8];
        int grade;
    };
    struct name_grade st_g1, st_g2;

    strcpy(st_g1.name, "이순신");
    st_g1.grade = 98;
    strcpy(st_g2.name, st_g1.name);
    st_g2.grade = st_g1.grade;
    printf("%s의 점수는 %d점 입니다.\n", st_g2.name, st_g2.grade);
    return 0;
}
```

# 프로그램 결과

이순신의 점수는 98점 입니다.

# 구조체 사용

## 프로그램 9.2

```
#include <stdio.h>
#include <string.h>
struct name_grade{
    char name[8];
    int grade;
};
int main(void){
    struct name_grade st_g1, st_g2;

    strcpy(st_g1.name, "이순신");
    st_g1.grade = 98;
    st_g2 = st_g1;

    printf("%s의 점수는 %d점 입니다.\n", st_g2.name, st_g2.grade);
    return 0;
}
```

# 다양한 구조체 변수 선언

- 구조체 형 선언과 동시에 변수 선언

```
struct name_grade {  
    char name[8];  
    int grade;  
} st_g3;  
struct name_grade st_g4;
```

# 다양한 구조체 변수 선언

- 구조체 태그가 없는 선언

```
struct {  
    char name[8];  
    int grade;  
} st_g5;  
struct {  
    char name[8];  
    int grade;  
} st_g6;
```

– st\_g5와 st\_g6는 다른 형

st\_g5 = st\_g6 ;    // 오류

# typedef

- 구조체 형 이름은 보통 길기 때문에 typedef를 많이 사용함
- 사용 예

```
typedef struct name_grade    name_grade;  
name_grade    st_g7;
```

```
typedef struct {  
    char    name[8];  
    int     grade;  
} name_grade;  
name_grade    st_g8, st_g9;
```



# 구조체의 초기화

- 변수 선언문에서 배열과 유사하게 초기화 함

```
struct name_grade st_g10 = {"둘리", 89};
```

```
// 멤버 순서 대로 초기화 됨
```

```
// st_g10.name = "둘리", st_g10.grade = 89
```

```
struct name_grade st_g11 = {"둘리"};
```

```
// grade 멤버의 값은 정의되지 않음
```

# 구조체의 초기화

- C99

- . 연산자를 사용하여 멤버를 지정하여 초기화할 수 있음

```
struct name_grade st_g12 = { .grade = 100 };
```

```
// grade 멤버만 100으로 초기화
```

# 복합 리터럴

- C99

- 중괄호로 묶인 데이터 집합

```
{ "이순신", 98 }
```

- 캐스트를 사용하여 형을 지정할 수 있음

```
(struct name_grade) { "이순신", 98 };
```

- 구조체 멤버의 값을 지정하기 위해 유용

```
st_g1 = (struct name_grade) { "이순신", 98 };
```

- 복합 리터럴 내에서 멤버를 지정할 수도 있음

```
st_g1 = (struct name_grade) { .grade = 98,  
                               .name = "이순신" };
```

# 구조체 멤버

- 구조체의 멤버로 구조체가 올 수 있음

```
struct subject {
    char    name[8];
    struct  name_grade student1;
    struct  name_grade student2;
    struct  name_grade student3;
    float    avg;
};

struct subject math;

math.student1.grade = 100;
```

|           |        | math     |
|-----------|--------|----------|
| .name     |        | char [8] |
|           |        |          |
| .student1 | .name  | char [8] |
|           | .grade | Int      |
| .student2 | .name  | char [8] |
|           | .grade | Int      |
| .student3 | .name  | char [8] |
|           | .grade | int      |
| .avg      |        | float    |

# 구조체

## 프로그램 9.3

```
struct name_grade{
    char name[8];
    int  grade;
};
struct subject {
    char    name[8];
    struct  name_grade student1;
    struct  name_grade student2;
    struct  name_grade student3;
    float   avg;
};
int main(void){
    struct subject math = {"수학", {"하나", 90}, {"둘", 44}, {"셋", 76}};
    math.avg =
        (math.student1.grade + math.student2.grade + math.student3.grade) / 3.0;
    printf("%s : %d 점\n", math.student1.name, math.student1.grade);
    printf("%s : %d 점\n", math.student2.name, math.student2.grade);
    printf("%s : %d 점\n", math.student3.name, math.student3.grade);
    printf("%s 평균은 %.2f점 입니다.\n", math.name, math.avg);
}
```

## 프로그램 결과

하나 : 90 점

둘 : 44 점

셋 : 76 점

수학 평균은 70.00점 입니다.

# 구조체 포인터

- 구조체 포인터

```
struct name_grade *st_gp;  
st_gp = &st_g1;  
st_gp.grade = 88;           // 오류
```

```
strcpy((*st_gp).name, "하느님");  
(*st_gp).grade = 88;
```

```
*st_gp.grade = 88;          // 오류  
// *st_gp.grade == *(st_gp.grade)
```

# 멤버 접근 연산자 ->

- -> 연산자
    - 구조체 포인터를 통해 멤버를 접근할 때 사용
- ```
strcpy(st_gp -> name, "하니");  
st_gp -> grade = 88;
```



# 구조체 포인터

## 프로그램 9.4

```
#include <stdio.h>

typedef struct name_grade{
    char name[8];
    int  grade;
} name_grade;

int main(void){
    name_grade  st, *st_p = &st;

    st = (name_grade) {"이순신", 98};

    printf("%s의 점수는 %d점 입니다.\n",
           st_p -> name, (*st_p).grade);
    return 0;
}
```

# 구조체 배열

- 일반 배열과 같은 방법으로 선언하고 사용됨

```
struct name_grade st_a[5];
```

- 구조체 배열 초기화

```
struct name_grade st_a[5] =
    {{ "하나", 77 }, { "둘", 87 }, { "셋", 65 },
     { "넷", 90 }, { "다섯", 98 } };
```

- C99 초기화

```
struct name_grade st_a[5] = { [1] = { "둘", 87 } };
struct name_grade st_a[5] =
    { [1].grage = 87, [1].name = "둘" };
```

# 구조체 배열

## 프로그램 9.5

```
int main(void){
    int sum = 0, i;
    float avg = 0.0;
    name_grade  st_a[N] = {{"하나", 77}, {"둘", 87},
                           {"셋", 65}, {"넷", 90}, {"다섯", 98}};
    printf("   이름           점수\n");
    for (i = 0; i < N; i++) {
        printf("%-10s  %3d\n", st_a[i].name, st_a[i].grade);
        sum += st_a[i].grade;
    }
    avg = (float) sum / N;
    printf("성적 평균은 %.2f 점입니다.\n", avg);
    return 0;
}
```

## 프로그램 결과

| 이름 | 점수 |
|----|----|
| 하나 | 77 |
| 둘  | 87 |
| 셋  | 65 |
| 넷  | 90 |
| 다섯 | 98 |

성적 평균은 83.40 점입니다.

## 구조체와 함수

- 구조체는 함수의 인자로서 함수에 전달될 수 있고, 함수로부터 리턴될 수도 있음
- 함수의 인자로서 구조체가 전달될 때 구조체는 값으로 전달됨

# 구조체와 함수

## 프로그램 9.6

```
#include <stdio.h>
typedef struct grade{
    int    grade[3];
    char   p_f[3];
    int    sum;
    float  avg;
} grade;

void grade_proc(grade st){
    st.sum = st.grade[0] + st.grade[1] + st.grade[2];
    st.avg = st.sum / 3.0;
    st.p_f[0] = st.grade[0] < 60 ? 'f' : 'p';
    st.p_f[1] = st.grade[1] < 60 ? 'f' : 'p';
    st.p_f[2] = st.grade[2] < 60 ? 'f' : 'p';
}
```

# 구조체와 함수

## 프로그램 9.6

```
int main(void){
    grade st = {{0}, {0}, -1, -1.0};

    printf("성적 입력(국어, 산수, 과학) : ");
    scanf("%d%d%d", &st.grade[0], &st.grade[1], &st.grade[2]);

    grade_proc(st);                // 구조체 전달

    printf("국어   : %d (%c)\n", st.grade[0], st.p_f[0]);
    printf("산수   : %d (%c)\n", st.grade[1], st.p_f[1]);
    printf("과학   : %d (%c)\n", st.grade[2], st.p_f[2]);
    printf("총점   : %d\n", st.sum);
    printf("평균   : %.2f\n", st.avg);
    return 0;
}
```

## 프로그램 결과

성적 입력(국어, 산수, 과학) : 50 100 80

국어 : 50 ( )

산수 : 100 ( )

과학 : 80 ( )

총점 : -1

평균 : -1.00



# 구조체와 함수

## 프로그램 9.7

```
#include <stdio.h>
typedef struct grade{
    int    grade[3];
    char   p_f[3];
    int    sum;
    float  avg;
} grade;

grade grade_proc2(grade st){
    st.sum = st.grade[0] + st.grade[1] + st.grade[2];
    st.avg = st.sum / 3.0;
    st.p_f[0] = st.grade[0] < 60 ? 'f' : 'p';
    st.p_f[1] = st.grade[1] < 60 ? 'f' : 'p';
    st.p_f[2] = st.grade[2] < 60 ? 'f' : 'p';
    return st;
}
```

# 구조체와 함수

## 프로그램 9.7

```
int main(void){
    grade st = {{0}, {0}, -1, -1.0};

    printf("성적 입력(국어, 산수, 과학) : ");
    scanf("%d%d%d", &st.grade[0], &st.grade[1], &st.grade[2]);

    st = grade_proc2(st);           // 구조체를 전달하고 리턴 받음

    printf("국어   : %d (%c)\n", st.grade[0], st.p_f[0]);
    printf("산수   : %d (%c)\n", st.grade[1], st.p_f[1]);
    printf("과학   : %d (%c)\n", st.grade[2], st.p_f[2]);
    printf("총점   : %d\n", st.sum);
    printf("평균   : %.2f\n", st.avg);
    return 0;
}
```

## 프로그램 결과

성적 입력(국어, 산수, 과학) : 50 100 80

국어 : 50 (f)

산수 : 100 (p)

과학 : 80 (p)

총점 : 230

평균 : 76.67

## 구조체와 함수

- 구조체가 많은 멤버를 가지거나, 큰 배열을 멤버로 가질 경우, 함수의 인자로 구조체를 전달하는 것은 상대적으로 비효율적임
- 따라서 대부분의 응용 프로그램에서는 함수의 인자로 구조체의 주소를 사용함

# 구조체와 함수

## 프로그램 9.8

```
int grade_proc3(grade * stp){
    if (stp == NULL) {
        printf("오류 : NULL 포인터\n");
        return -1;
    }
    stp -> sum = stp -> grade[0] + stp -> grade[1] +
                stp -> grade[2];
    stp -> avg = stp -> sum / 3.0;
    stp -> p_f[0] = stp -> grade[0] < 60 ? 'f' : 'p';
    stp -> p_f[1] = stp -> grade[1] < 60 ? 'f' : 'p';
    stp -> p_f[2] = stp -> grade[2] < 60 ? 'f' : 'p';
    return 0;
}
```

# 구조체와 함수

## 프로그램 9.8

```
int main(void){
    grade st = {{0}, {0}, -1, -1.0};

    printf("성적 입력(국어, 산수, 과학) : ");
    scanf("%d%d%d", &st.grade[0], &st.grade[1], &st.grade[2]);

    if (grade_proc3(&st))                // 구조체 포인터 전달
        return 1;

    printf("국어   : %d (%c)\n", st.grade[0], st.p_f[0]);
    printf("산수   : %d (%c)\n", st.grade[1], st.p_f[1]);
    printf("과학   : %d (%c)\n", st.grade[2], st.p_f[2]);
    printf("총점   : %d\n", st.sum);
    printf("평균   : %.2f\n", st.avg);
    return 0;
}
```

# 공용체

- `union`
- 공용체는 구조체와 비슷한 구문 형식을 가지지만 각 멤버들은 같은 기억장소를 공유함
- 공용체형은 메모리의 같은 위치에 저장될 여러 값의 집합을 정의
- 저장된 값을 올바르게 해석하는 것은 프로그래머의 책임

# 공용체 선언

- 예제

```
union short_or_float {  
    short    s;  
    float    f;  
};
```

- union : 키워드
- short\_or\_float : 공용체 태그 이름
- s, f : 공용체 멤버

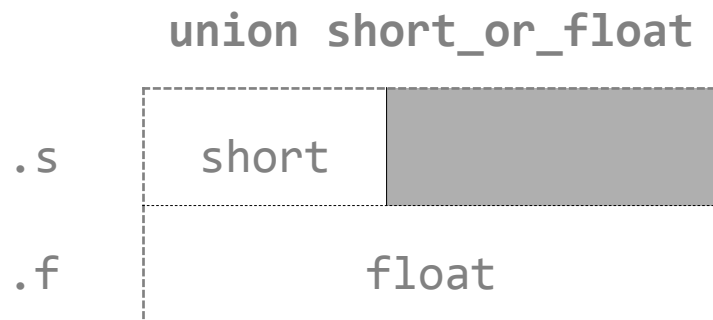


# 공용체 선언

## • 예제

```
union short_or_float {  
    short    s;  
    float    f;  
};
```

- short\_or\_float 공용체 **형**을 선언한 것임
- union short\_or\_float **모양** (4 바이트)



# 공용체 변수 선언

- 공용체 변수는 구조체 변수와 같은 방법으로 선언됨

```
union short_or_float a, b, c;
```

- 이 선언으로 a, b, c에 대한 기억장소 할당

- 공용체의 멤버 접근 방법은 구조체의 멤버 접근 방법과 동일

```
a.s = 10
```

```
b.f = 1.0
```

# 공용체

## 프로그램 9.9

```
#include <stdio.h>
typedef union short_or_float {
    short    s;
    float    f;
} number;
int main(void){
    number    n;
    n.s = 2007;
    printf("s: %10d          f: %16.10e\n", n.s, n.f);
    n.f = 2007.0;
    printf("s: %10d          f: %16.10e\n", n.s, n.f);
    return 0;
}
```

# 프로그램 결과

```
s:      2007      f: 2.8124060179e-42
s:     -8192      f: 2.00700000000e+03
```

## 구조체와 공용체

- 공용체는 구조체의 멤버로 주로 사용됨
- 구조체가 공용체를 멤버로 가질 경우 추가적인 멤버를 하나 더 정의하여 현재 공용체에 어떤 멤버의 값이 저장됐는지를 표시함

# 구조체와 공용체

## 프로그램 9.10

```
#include <stdio.h>
#define  WON  0
#define  DOLLAR 1
union won_or_dollar {
    int      won;
    float    dollar;
};
struct product {
    char      *name;
    _Bool     w_d;
    union     won_or_dollar    price;
};
```

# 구조체와 공용체

## 프로그램 9.10

```
int main(void){
    int i;
    struct product item[2];
    item[0].name = "PMP";
    item[0].price.won = 500000;
    item[0].w_d = WON;
    item[1].name = "CAMERA";
    item[1].price.dollar = 799.95;
    item[1].w_d = DOLLAR;
    for (i = 0; i < 2; i++) {
        printf("품명: %-10s", item[i].name);
        if (item[i].w_d)
            printf("가격: $%11.2f\n", item[i].price.dollar);
        else
            printf("가격: %12d 원\n", item[i].price.won);
    }
    return 0;
}
```

## 프로그램 결과

|             |         |        |   |
|-------------|---------|--------|---|
| 품명 : PMP    | 가격 :    | 500000 | 원 |
| 품명 : CAMERA | 가격 : \$ | 799.95 |   |



# 열거형

- 사람은 숫자보다는 단어에 더 익숙함
- 열거형은 제한적이지만 숫자대신 단어를 사용할 수 있게 함
- 키워드 `enum`은 열거형을 선언하는데 사용됨
- 열거형은 유한집합을 명명하고, 그 집합의 원소로서 식별자를 선언하는 수단을 제공함

# 열거형

- 예제

```
enum day {SUN, MON, TUE, WED, THU, FRI, SAT};
```

- enum : 키워드
- day : 태그 이름
- SUN, MON, ..., SAT : 열거자
- 이 선언은 enum day **형** 정의임

# 열거형

- 열거형은 열거자는 정수처럼 사용됨
- 열거자의 값은 디폴트로 첫 번째 원소는 0이고, 각 원소는 이전 원소의 값보다 하나 큰 값을 가짐

```
enum day {SUN, MON, TUE, WED, THU, FRI, SAT};  
// SUN : 0, MON : 1, TUE : 2, . . .  
printf("%d\n", SUN);           // 0 출력
```

# 열거형

- 열거자를 다른 값으로 초기화할 수 있음

- 초기화가 없는 열거자는 앞 열거자보다 1 큰 값을 가짐

```
enum month {Jan=1, Feb, Mar, Apr, May, Jun, Jul,  
            Aug, Sep, Oct, Nov, Dec};
```

```
// Jan : 1, Feb : 2, . . .
```

```
enum fruit {apple = 7, pear, orange = 3, lemon};
```

```
// apple : 7, pear : 8, orange : 3, . . .
```

```
enum veg {beet = 17, carrot = 17, corn = 17};
```

# 열거형 변수

- 이미 정의된 열거형으로 변수 선언

```
enum day d1, d2;
```

- 열거형 선언과 함께 변수 선언

```
enum day {SUN, MON, TUE, WED, THU, FRI, SAT} d1, d2;
```

- 태그 이름 생략할 수 있음

```
enum {SUN, MON, TUE, WED, THU, FRI, SAT} d1, d2;
```

- 열거형 변수는 일반 변수와 같이 사용됨

```
d1 = SUN;
```

```
if (d1 == d2)
```

```
    . . .
```

# 열거형

## 프로그램 9.11 일부

```
#include <stdio.h>
enum day {SUN, MON, TUE, WED, THU, FRI, SAT};
int main(void){
    time_t now;
    enum day today;
    now = time(NULL);                // 1970년 1월 1일부터 흐른 시간(초)
    today = (now / (60*60*24) + 4) % 7; //1970년 1월 1일 -> 목요일
    switch (today) {
        case SUN:
            printf("오늘은 일요일 입니다.\n");
            break;
        case MON:
            printf("오늘은 월요일 입니다.\n");
            break;
        . . .
    default:
        printf("time() 함수 오류입니다.\n");
    }
    return 0;
}
```

# 프로그램 결과

오늘은 수요일 입니다.