작성자 : 김기희 (cupid072@naver.com)

구조체

- 학습목표
  - 구조체의 개념을 설명할 수 있다.
  - 구조체 자료형을 선언할 수 있다.
  - 구조체 변수를 선언하고 올바른 구조체 변수 초기화 방법을 설명할 수 있다.
  - 구조체 멤버 정렬 제한에 대하여 설명할 수 있다.
  - 구조체 멤버를 참조할 수 있다.
  - 구조체 포인터를 통해 구조체 멤버를 참조할 수 있다.
  - 구조체 데이터를 함수에 전달하는 방법을 설명할 수 있다.
  - 자기 참조 구조체에 대하여 설명할 수 있다.
  - 비트필드에 대해 설명할 수 있다.
  - 올바른 비트필드 사용법을 설명할 수 있다.

프리젠테이션 사용 불가

구조체 개념

- 구조체란 서로 다른 데이터형의 정보를 묶어 복잡한 데이터를 간결하게 단일 변수처럼 처리할 수 있도록 해 주는 복합 데이터형이다.

- 구조체를 구성하는 각 요소를 멤버(member)라고 한다.

- 구조체 선언방법

```
struct 구조체명 {
  member 선언;
  ...
};
```

```
struct _person {
    char name[20];
    unsigned short age;
};
```

프리젠테이션 사용 불가

구조체의 특성

- 구조체명은 식별자를 생성하는 규칙에 따라 정의한다.
- 구조체 자료형의 선언은 컴파일러에게 구조체 자료형의 구조와 함께 구조체가 사용될 것임을 알리는 것이다.
- 그러므로 구조체 자료형 선언 시 멤버를 초기화 하는 것은 불가능 하다.
- 구조체 멤버는 변수, 배열, 포인터 변수, 다른 구조체 데이터형 변수로 구성할 수 있다.
- 구조체 자료형은 구조체 변수를 선언 시 실제로 메모리가 할당된다.
- 구조체의 각 멤버에 접근할 때에는 멤버 참조 연산자(.)를 사용한다.
- 구조체에 대한 대입 연산자는 구조체 데이터 복사가 수행된다.

프리젠테이션 사용 불가

구조체 변수의 선언과 초기화 #1

- 구조체 자료형의 선언

```
struct _person {
    char name[20];
    unsigned short int age;
};
```

struct _person person	
char [20]	unsigned short int
김기희	30

- 구조체 변수의 선언과 초기화

```
struct _person person = { "김기희", 30 };
```

- 구조체 변수의 초기화는 구조체 멤버의 값을 중괄호로 감싸서 기술한다.

프리젠테이션 사용 불가

구조체 멤버의 선택적 초기화

- 구조체 자료형의 선언

```
struct _person {
    char name[20];
    unsigned short int age;
    unsigned int birth;
};
```

struct _person person		
char [20]	unsigned short int	unsigned int
name	age	birth
김기희	30	19990512

- 구조체 멤버의 선택적 초기화

```
struct _person person = { .name = "김기희", .age = 30, .birth = 19990512 };
```

- 선택적 초기화시 주의사항
  - 멤버명이 지정된 초기화 리스트의 다음 항목은 지정된 멤버 다음 멤버의 값이 된다.

```
struct _person person = { .name = "김기희", 30, .birth = 19990512 };      // 멤버 age의 초기 값이 됨
struct _person person = { .age = 30, .name = "김기희", 19990512 };// 멤버 age의 초기 값이 됨
```

  - 구조체 멤버를 선택적으로 초기화 하는 경우 모든 멤버에 대한 값을 명시적으로 기술할 것을 권장

프리젠테이션 사용 불가

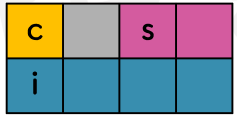
구조체 패딩

- 구조체 정렬은 구조체 멤버의 주소 값이 멤버 크기의 배수가 되도록 조정하는 것을 말한다.
- 구조체 패딩은 구조체 정렬 과정에서 발생하는 구조체 멤버 사이의 사용되지 않은 메모리 공간을 말한다.
- 구조체 정렬은 구조체를 구성하는 기본 자료형 중에서 가장 큰 타입을 기준으로 설정된다.
- 구조체 정렬을 위한 할당 크기는 컴파일러에 따라 다를 수 있다.

```
struct A {
    char c;
    int i;
    short s;
}
```


```
sizeof(struct A);       // 12byte
```

```
struct B {
    char c;
    short s;
    int i;
}
```



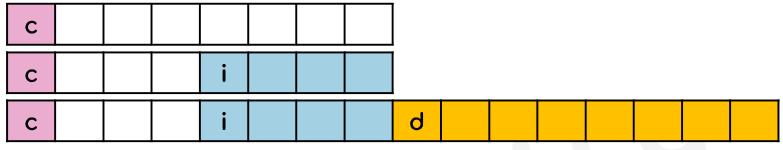

```
sizeof(struct B); // 8byte
```

프리젠테이션 사용 불가

구조체 패딩

- 다음 구조체 패딩 원리는 다음과 같다.

```
struct A {
    char c;
    int i;
    double d;
}
```



- 구조체 멤버 중 가장 큰 자료형의 크기가 기본 할당 크기가 된다.
- 할당된 공간의 시작 위치로부터 첫 번째 멤버 크기의 배수에 해당하는 위치로부터 1바이트를 멤버 c에 할당한다.
- 할당된 공간의 시작 위치를 기준으로 하여 두 번째 멤버 크기의 배수에 해당하는 위치로부터 4바이트를 멤버 i에 할당한다.
- 다음 멤버에 할당할 공간이 부족하므로 구조체 멤버 중 가장 큰 자료형의 크기 만큼 메모리 할당을 추가한다.
- 추가된 메모리 공간에 세 번째 멤버 크기의 배수에 해당하는 위치로부터 8바이트를 멤버 d에 할당한다.

무단 배포 및 상업적 사용 불가

구조체 배열

- 기본자료형과 마찬가지로 구조체 자료형 또한 배열로 구성할 수 있다.

```
struct _person {                                        // 구조체 자료형의 선언
    char name[20];
    unsigned short int age;
};

struct _person persons[5] = { {"홍길동", 20}, ..... };          // 구조체 배열의 선언 및 초기화
```

persons[0]		persons[1]		persons[2]		persons[3]		persons[4]	
name	age	name	age	name	age	name	age	name	age

```
persons[0].name;                                // 구조체 배열의 참조
```

프레젠테이션 사용 불가

구조체 포인터

- 구조체 포인터는 구조체 데이터가 저장된 메모리 번지수를 가리킨다.
- 구조체 포인터를 통해 구조체 멤버를 참조하고자 하는 경우 구조체 간접 참조 연산자(->)를 사용한다.

```
struct _person {
    char name[20];
    unsigned short age;
}

struct _person kim = { "김기희", 20 };
struct _person* ptr = &kim;

printf("이름 : %s\n", kim.name);          // 구조체 멤버 참조
printf("이름 : %u\n", (&kim)->age);       // 주소 연산을 통한 구조체 포인터 간접 참조
printf("이름 : %s\n", ptr->name);         // 구조체 포인터 간접 참조
printf("이름 : %u\n", (*ptr).age);        // 포인터 간접 참조 연산을 통한 멤버 참조
```

프리젠테이션 사용 불가

구조체 포인터의 필요성

- 구조체에 대한 대입 연산자는 구조체 데이터 복사를 수행한다.

```
struct _person kihee = { "김기희", 20 };
struct _person gildong = kihee;                    // 구조체 변수 복사 수행

strcpy(gildong.name, "고길동");
gildong.age = 30;

printf("p1.name : %s\n", kihee.name);              // 김기희
printf("p1.age : %u\n", kihee.age);                // 20

printf("p2.name : %s\n", gildong.name);            // 고길동
printf("p2.age : %u\n", gildong.age);              // 30
```

프리젠테이션 사용 불가

구조체 포인터의 필요성

- 함수 호출 시 구조체 변수를 전달 인자로 사용하는 경우 구조체의 데이터 복사가 수행된다.
- 구조체 데이터의 복사는 함수 호출 시 오버헤드를 유발한다.

```
void print(struct _person p) {
    strcpy(p.name, "홍길동");
    p.age = 30;
    printf("이름 : %s\n", p.name);          // 홍길동
    printf("연령 : %d\n", p.age);           // 30
}

struct _person kihee = { "김기희", 20 };
print(kihee);
printf("이름 : %s\n", kihee.name);          // 김기희
printf("연령 : %u\n", kihee.age);           // 20
```

※ 오버헤드 (Overhead)
함수 호출 시 발생되는 자원의 소모를 말한다.
단순히 메모리 소비만을 말하는 것이 아니라 실행 속도 저하와 같은 현상을 포함한다.

프리젠테이션 사용 불가

구조체 포인터의 필요성

- 구조체 포인터 사용 시 함수를 사용한 구조체 데이터 조작이 가능하다.

```
void print(struct _person* p) {
    strcpy(p->name, "홍길동");                    // 함수 내부에서 인수로 전달 받은 구조체 멤버를 조작할 수 있다.
    p->age = 30;
}

struct _person kihee = { "김기희", 20 };
print(&kihee);
printf("이름 : %s\n", kihee.name);                // 홍길동
printf("연령 : %u\n", kihee.age);                 // 30
```

- 함수에서 변경할 수 없도록 하고자 하는 경우 const 식별자를 사용한다.

```
void print(const struct _person* p);              // 함수 내부에서 구조체 멤버에 대한 조작 불가
void print(struct _person* const p);              // 함수 내부에서 파라미터 p에 새로운 대입 불가
void print(const struct _person* const p);        // 멤버 조작 불가 + 새 주소 대입 불가
```

프리젠테이션 사용 불가

구조체 배열을 가리키는 포인터

- 구조체 배열은 구조체 포인터를 사용하여 참조할 수 있다.

```
struct _person persons[5] = { {"홍길동", 20}, {"심청이", 16}, ..... };
struct _person* ptr = persons;

printf("이름 : %s\n", ptr[0].name);          // 포인터를 배열처럼 사용
printf("연령 : %d\n", ptr[0].age);

printf("이름 : %s\n", (*ptr).name);          // 간접 참조 연산자를 통한 멤버 참조
printf("연령 : %d\n", (*ptr).age);

printf("이름 : %s\n", ptr->name);            // 간접 멤버 참조 연산자에 의한 멤버 참조
printf("연령 : %d\n", ptr->age);
```

※ 참조연산자 ([] . ->)의 우선 순위는 간접 참조 연산자(*) 보다 우선순위가 높다.

프리젠테이션 사용 불가

구조체를 맴버로 하는 구조체

- 구조체는 다른 구조체 타입을 맴버로 가질 수 있다.

```
struct _person {
        char name[20];
        int age;
};

struct _student {
        char major[20];
        struct _person person;
};
```

struct _student st		
major	struct _person person	
	name	age
전산학과	홍길동	20

- 변수의 선언

```
struct _student st = { "전산학과", { "홍길동", 20 } };
printf("전공 : %s, 이름 : %s, 연령 : %d\n", st.major, st.person.name, st.person.age);
```

프리젠테이션 사용 설명

자기 참조 구조체

- 자기 참조 구조체란? 구조체 멤버로 자신과 같은 구조체 자료형을 가리키는 포인터를 포함하고 있는 구조체를 말한다.

- 구조체가 자신의 타입을 멤버로 가지는 경우 구조체 포인터의 형태로만 가질 수 있다.

- 주로 Linked List(링크드 리스트)의 노드를 구현할 때 사용된다.

```
struct _node {
  int data;
  struct _node* next;        // OK
}
```

```
struct _node {
int data;
    struct _node next;       // ERROR
}
```

프리젠테이션 사용 불가

비트 필드(Bit Field)

- 비트 필드는 bit 단위로 멤버를 할당하여 사용하는 복합 데이터형이다.
- 비트 필드는 구조체 정의를 사용하여 선언하며, 멤버 필드 선언 시 콜론(:)으로 구분하여 bit 폭을 지정한다.
- 비트 필드에 값을 대입할 때에는 해당 필드 멤버가 저장할 수 있는 값에 주의하여야 한다.

```
struct bit_field {
    int sa: 1;              // 부호 비트만 존재하므로 0 또는 -1 값만 저장
    int sb: 2;              // -2^(2-1) ~ 2^(2-1)-1     (-2 ~ 1)
    int sc: 4;              // -2^(4-1) ~ 2^(4-1)-1     (-8 ~ 7)
    unsigned int ua: 1;     // 0 또는 1
    unsigned int ub: 2;     // 0 ~ 2^2-1                (0 ~ 3)
    unsigned int uc: 4;     // 0 ~ 2^4-1                (0 ~ 15)
}
```

프리젠테이션 사용 불가

비트 필드 사용시 주의 사항

- 비트 필드 멤버의 주소는 구할 수 없다.
- 비트 필드 멤버에 저장 가능한 값의 범위에 주의하여야 한다.
- 비트 필드 멤버는 동일한 자료형을 사용하여 선언하도록 한다.

```
struct bit_field {
        int sa: 2;
        int sb: 2;
        unsigned int ua: 2;
        unsigned int ub: 2;
}
```

int형 크기 만큼 할당

```
struct bit_field {
        int si: 2;
        unsigned int ui: 2;
        char sc: 2;
        unsigned short us: 4;
}
```

int형 크기 * 2 만큼 할당

프리젠테이션 사용 불가

구조체 배열의 동적 할당

- 구조체 배열의 동적 할당

```
struct _person {
    char name[20];
    int age;
};
.....
struct _person* persons = (struct _person*) malloc(sizeof(struct _person) * 3);
```

STACK	HEAP					
persons	persons[0]		persons[1]		persons[2]	
0x...	name	age	name	age	name	age

구조체 포인터 배열의 동적 할당

- 구조체 배열의 동적 할당

```
typedef struct {
    char name[20];
    int age;
} person_t;
.....
struct _person** persons = (struct _person**) malloc(sizeof(struct _person*) * 3);
persons[0] = (struct _person*) malloc(sizeof(struct _person));
```

STACK	HEAP				
persons	persons[0]	persons[1]	persons[2]	0xFF86...	
0x...	0xFF86...	NULL	NULL	name	age