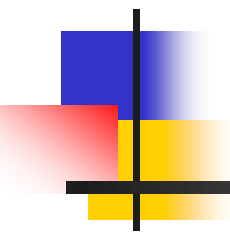


C Programming (CSE2035)

(Chap11. Derived types-enumerated, structure, and union) 2



Sungwon Jung, Ph.D.

Bigdata Processing & DB LAB
Dept. of Computer Science and Engineering
Sogang University
Seoul, Korea
Tel: +82-2-705-8930
Email : jungsung@sogang.ac.kr

Accessing structures

- 구조체도 하나의 타입이므로 포인터 변수가 존재한다.
다음의 구조체 **student**에 대해서

```
typedef struct{  
    int    id;  
    char   name[26];  
    float  gradePoints;  
}student;  
student  Park;
```

- **student**에 대한 포인터 선언 후

```
student* parkPtr;
```

- 선언된 구조체 변수 **Park**을 포인팅 하도록 연산

```
parkPtr = &Park;
```

- 이후 ***parkPtr**은 **Park**과 동일하게 사용될 수 있다.



Accessing structures

- 이후에 포인터를 통해 구조체의 멤버를 접근하기 위해서 다음과 같이 사용한다.

<code>(*parkPtr).id;</code>	<code>(*parkPtr).name ;</code>	<code>(*parkPtr).gradePoints;</code>
-----------------------------	--------------------------------	--------------------------------------

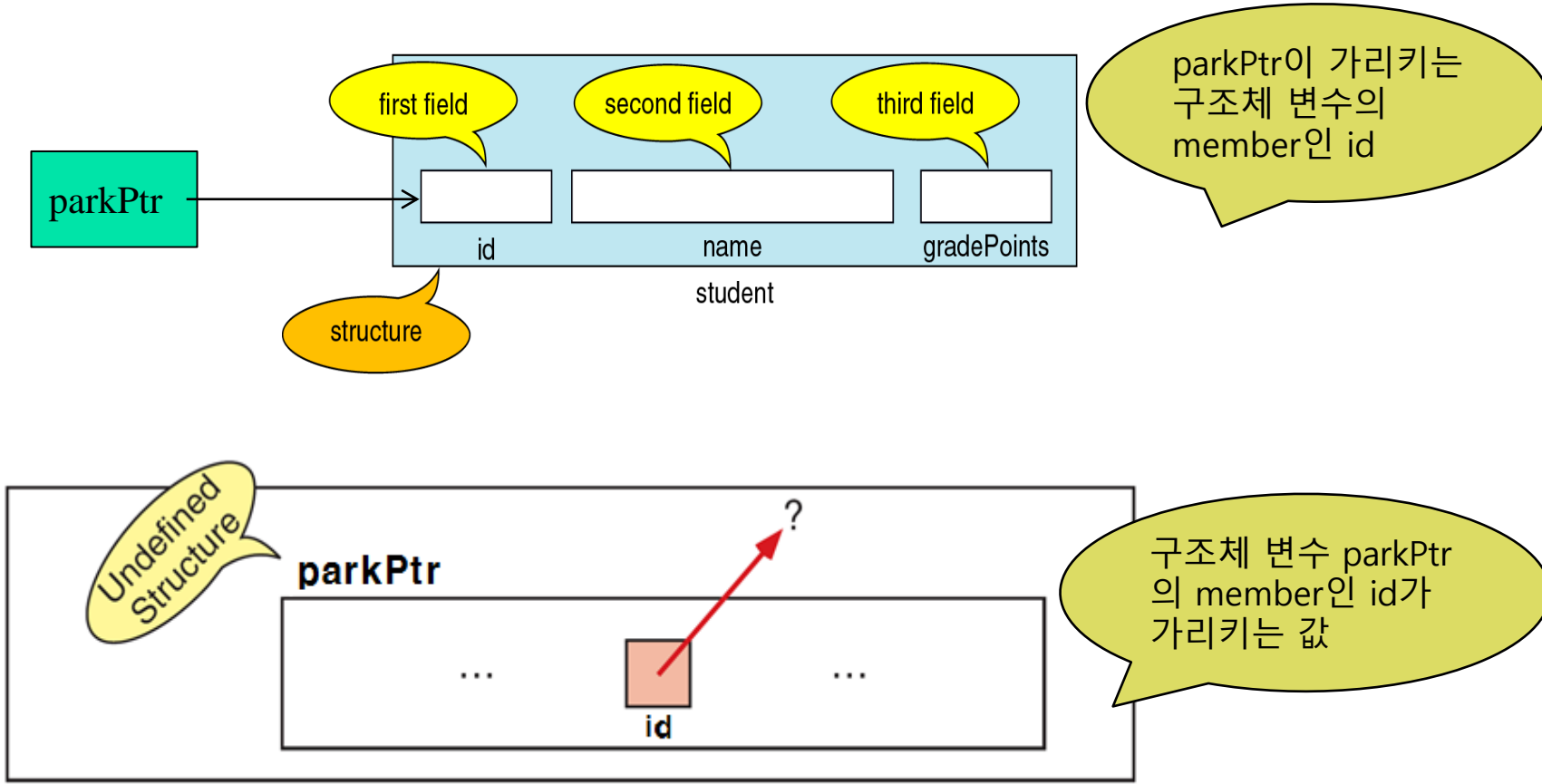
이는 다음과 같다.

<code>park.id;</code>	<code>park.name ;</code>	<code>park.gradePoints;</code>
-----------------------	--------------------------	--------------------------------

- 구조체 멤버에 대한 포인터 연산자 사용시 다음을 주의한다.
 - 멤버연산자(.) 의 우선순위가 참조연산자(*) 보다 높기 때문에 `*parkPtr.id`는 `(*parkPtr).id`와 다르다.
 - 즉, `*parkPtr.id`는 `*(parkPtr.id)`와 동일한 결과를 가지게 된다. 결과적으로, 이 표현은 `(parkPtr.id)`가 가리키는 곳의 값(*)을 의미하게 된다.

Accessing structures

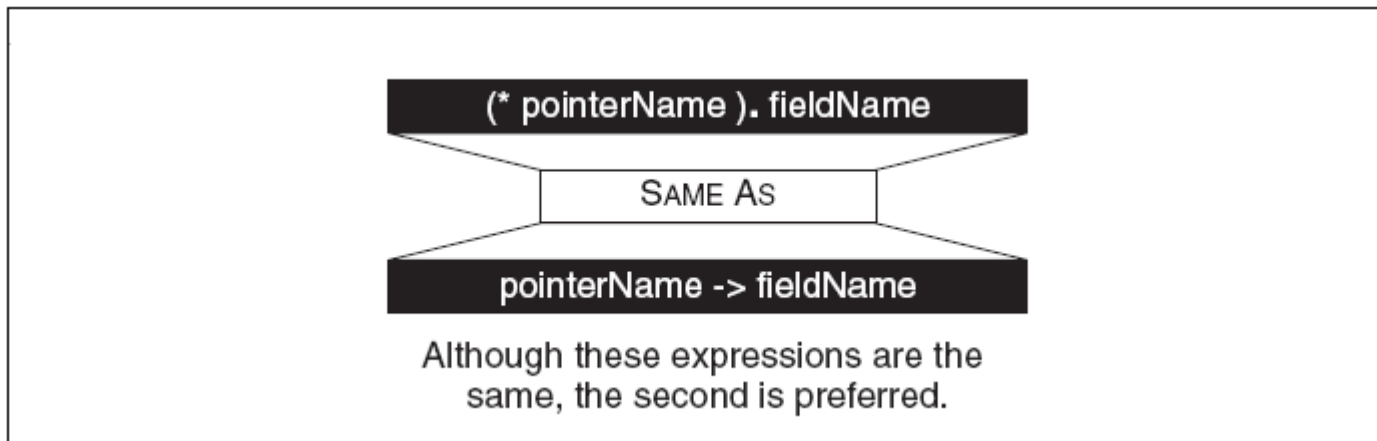
- 다음 그림은 (*parkPtr).id 와 *parkPtr.id의 차이를 보여준다.



Accessing structures

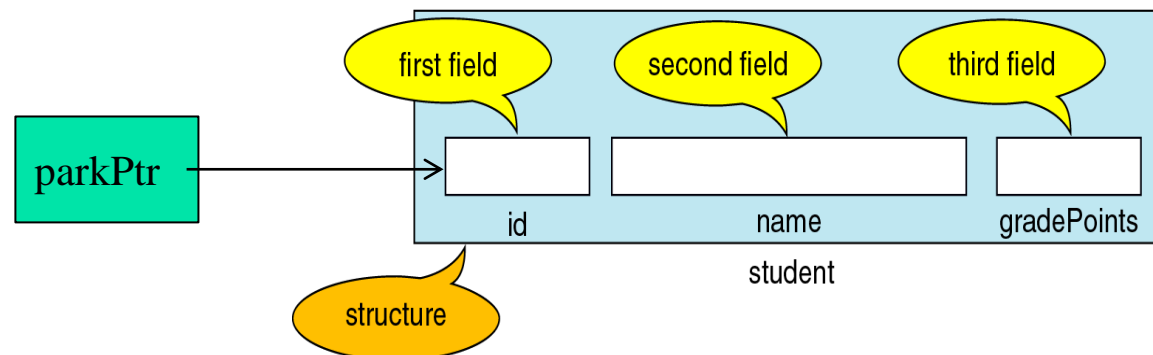
■ Selection Operator (->)

- 앞에서 본 것처럼 구조체 포인터를 이용하여 구조체 멤버에 접근할 때는 괄호 연산자를 사용해야 하는 불편함이 있다.
 - ex. (*parkPtr).id
- Selection Operator(->)는 이런 경우에 사용할 수 있는 연산자이다. 예를 들면, 위의 구문은 다음과 같이 바꿀 수 있다.
 - parkPtr->id
- selection operator는 구조체 포인터를 이용해서 멤버에 접근하고자 할 때 사용한다.



Accessing structures

- 앞에서 살펴본 예를 Selection Operator를 이용하면 다음과 같다.



`(*parkPtr).id;` `(*parkPtr).name ;` `(*parkPtr).gradePoints;`



`parkPtr->id;` `parkPtr->name ;` `parkPtr->gradePoints;`

- 즉, 멤버 변수 `id`에 접근하는 방법은 3가지가 있다.

`park.id;` `(*parkPtr).id;` `parkPtr->id`


Structure Pointers

- 구조체도 기존에 배운 타입처럼 malloc을 통해 할당 받을 수 있다. 즉, 다음과 같이 하나의 공간을 할당 받아 사용 가능하다.

```
student* parkPtr = (student*) malloc (sizeof(student));    // 할당

parkPtr->id = 3;
strcpy( parkPtr->name, "Yeouido Park");
parkPtr->gradePoints = 97.46;
```

- 이 경우에 member id에 접근할 수 있는 방법은 다음의 2가지 이다.

 parkPtr->id;	(*parkPtr).id;	parkPtr->id
--	----------------	-------------

Example

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 typedef struct{
6     int id;
7     char name[26];
8     double gradePoints;
9 }Student;
10
11 void main(){
12     Student Park;
13     Student *pPark, *pKim;
14
15     pPark = &Park;
16     pKim = (Student*) malloc (sizeof(Student));
17
18     Park.id = 20141234; Park.gradePoints = 2.3;
19     strcpy( Park.name, "Su Park" );
20
21     printf("%s(%d) : %.2lf\n", pPark->name, pPark->id, pPark->gradePoints);
22
23     pPark->id = 20151234; pPark->gradePoints = 3.3;
24     strcpy( pPark->name, "Soo Park" );
25
26     printf("%s(%d) : %.2lf\n", (*pPark).name, (*pPark).id, (*pPark).gradePoints);
27
28     pKim->id = 20161234; pKim->gradePoints = 4.3;
29     strcpy( pKim->name, "Jenny Kim" );
30
31     printf("%s(%d) : %.2lf\n", pKim->name, pKim->id, pKim->gradePoints);
32 }

```

구조체 정의

Dot notation을 이용하여 삽입

Selection notation을 이용해서 data 삽입

malloc으로 할당된 변수에 data 삽입

```

./a.out
Su Park(20141234) : 2.30
Soo Park(20151234) : 3.30
Jenny Kim(20161234) : 4.30

```


Complex structures

- 구조체도 하나의 타입이므로 구조체의 멤버로 어느 특정한 구조체를 가질 수 있다. 이러한 것을 **Nested Structure**라고 한다.
- 가령, 병원에서 환자의 정보를 기록하는 프로그램을 작성한다면 각 환자의 생년월일, 입원날짜, 퇴원날짜 같이 기본 타입이 아닌 멤버를 가질 수 있다. 이러한 날짜를 나타내는 구조체를 선언한 뒤, 이 구조체를 다른 구조체의 변수로 사용하면 된다.

```
typedef struct{
    int year;
    int month;
    int day;
}DATE;

typedef struct{
    char name[26];
    int age;
    DATE birthday;
    DATE enterDate;
    DATE leaveDate;
}PATIENT;

PATIENT Kim;
```

사용하고자 하는 struct
위에 선언해야 한다.

Complex structures

- 아래와 같이 구조체 내부에서 멤버로써 구조체를 선언 하는 것도 가능하다. 즉 구조체 내부에서 새로운 구조체를 선언하는 것도, 구조체 밖에서 선언한 구조체를 멤버로 쓰는 것도 모두 가능하다.
- 선언된 **nested structure**의 **member**는 **dot notation**을 반복하는 것으로 접근이 가능하다.

```
typedef struct {
    struct {
        int year;
        int month;
        int day;
    } date;
    struct {
        int hour;
        int min;
        int sec;
    } time;
} STAMP;

STAMP start;
```

→ 구조체 변수 `start`의 member인 `year`에 접근하기 위해서는 `start.date.year` 와 같이 반복해서 notation을 사용

→ 구조체 변수 `start`의 member인 `min`에 접근하기 위해서는 `start.time.min` 와 같이 반복해서 notation을 사용

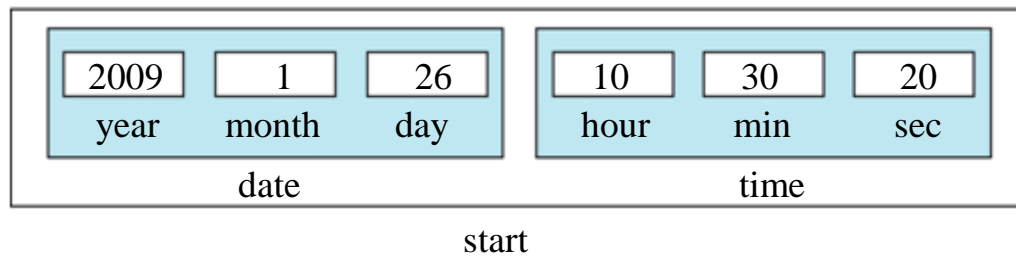
Complex structures

- Nested structure 또한 structure처럼 초기화가 가능하다.

```
typedef struct {  
    struct {  
        int year;  
        int month;  
        int day;  
    } date;  
    struct {  
        int hour;  
        int min;  
        int sec;  
    } time;  
} STAMP;
```

- 위와 같은 구조체의 경우 아래와 같이 변수 `start`를 초기화 할 수 있다.

STAMP start = {{2009, 1, 26}, {10, 30, 20}};





Array of structure

- 구조체 또한 **type**의 일종이므로 구조체 변수 또한 배열로 선언하고 사용할 수 있다.
- 예를 들어 앞에서 정의한 **student** 구조체를 가지고 살펴보자. 만약 관리해야 하는 학생의 수가 **50명**이라면 어떻게 할 것인가?

```
struct student{  
    int    id;  
    char   name[26];  
    double gradePoints;  
};  
struct student std1;  
struct student std2;  
...  
struct student std50;
```



```
struct student{  
    int    id;  
    char   name[26];  
    double gradePoints;  
};  
struct student std[50];
```

Array of structure

```
struct student{  
    int    id;  
    char   name[26];  
    double gradePoints;  
};  
struct student std[50];
```

- 위와 같이 정의된 구조체에 접근하는 방법은 아래와 같다.

```
std[0].id;           //첫 번째 학생의 ID  
std[0].name;         //첫 번째 학생의 이름  
std[0].gradePoints;  //첫 번째 학생의 성적  
...  
std[49].id;          //마지막 학생의 ID
```

- 또한 이전에 배운 포인터와 배열의 관계는 구조체의 배열에도 그대로 적용된다.

```
struct student *pStd;           //포인터 변수를 선언  
pStd = std;                     //포인터 변수는 배열의 시작을 가리킴  
std[4].id = 20071234;           //5번째 학생의 id가 20071234  
printf("%d\n", (pStd+4)->id);  //20071234가 출력된다.
```

Array of structure

```
struct student{  
    int    id;  
    char   name[26];  
    double gradePoints;  
};  
struct student *list;
```

- 또한, 구조체도 동적으로 공간을 할당 받아 사용할 수 있다.
앞선 예제를 동적 할당을 통해 구현하면 다음과 같다.

```
int i, n=50;  
list = (struct student*)malloc(sizeof(struct student) * n); // 50개의 공간을 동적 할당.  
  
for(i=0; i<n; i++)  
    list[i].id = 20071230 + i; // Dot Notation을 사용  
  
printf("%d\n", (list+4)->id); // 20071234가 출력된다.  
// Selection Notation을 사용
```