

# Structures and Unions

For copyright and license information,  
<http://class.icc.skku.ac.kr/~min/program/license.html>

# Structure

- Type이 다른 data들을 묶어서 사용하기 위한 수단
- Keyword "struct"를 사용하여 묶인 data에 대한 새로운 type을 만든다.
- **struct** a\_person {        /\* tag is optional \*/  
    char \*name;  
    int  date\_of\_birth[3];  
    long salary;  
} person;        /\* variable is optional \*/

# Why Structure ?

- Program 대상이 객체인 경우가 거의 대부분이다.

(예) logic gate, coil, transistor, ...


- 그 객체 들은 속성을 가지고 있다.

(예) coil 값, 감은 횟수, 통의 굵기 ...

- struct : 속성들을 모아서 하나의 데이터 덩어리로 사용하기 편리한 수단

# Another Definition of Structure

- ```
typedef struct a_person {      /* tag is optional */
    char *name;
    int   date_of_birth[3];
    long  salary;
} PERSON;
```



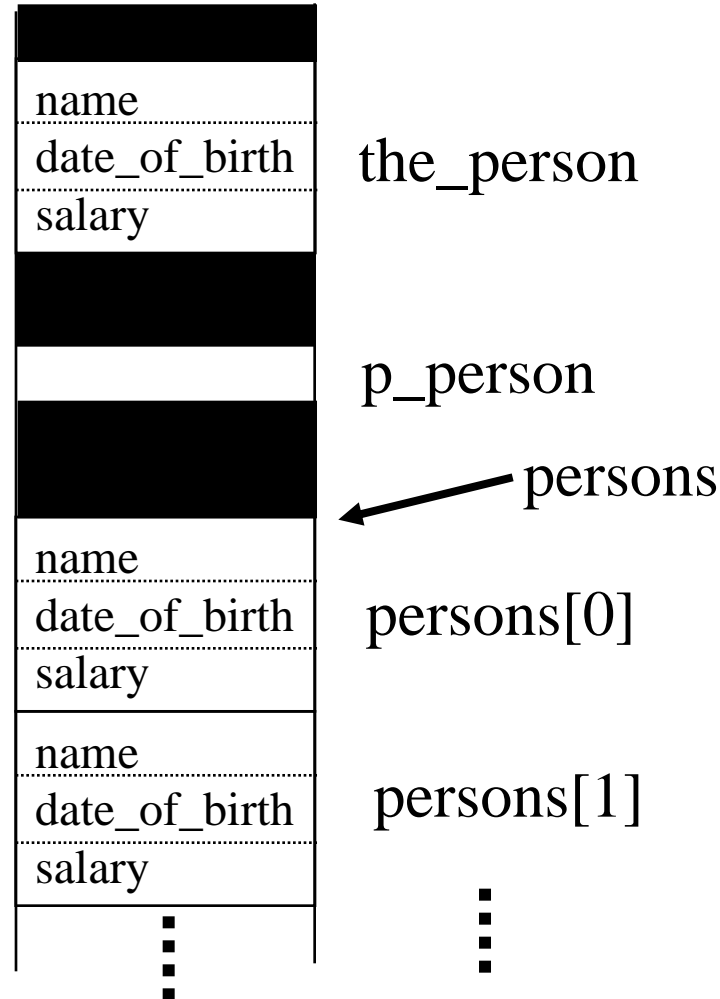
dynamic memory  
allocation을 염두에  
둔 선언

```
PERSON      the_person, persons[100], *p_person;
struct a_person the_person, persons[100], *p_person;
```

# Memory of A Structure

```
typedef struct
  a_person {
    char *name;
    int date_of_birth[3];
    long salary;
  } PERSON;
```


```
PERSON the_person;
PERSON persons[100];
PERSON *p_person;
```



# Access to members

```
PERSON the_person, *p_person;
```

malloc( )  
strcpy( )



```
the_person . name = "Your name";
```

```
the_person . date_of_birth[0] = 1980;
```

```
the_person . salary = 1800000L;
```

```
printf("name = %s\n", the_person . name);
```

```
p_person = &the_person;
```

```
printf("name = %s\n", p_person->name);
```

# Assignment on Struct Variable

```
struct a_person {      /* tag is optional */
    char *name;
    int date_of_birth[3];
    long salary;
} person1, person2;
```

malloc( ) &  
strcpy()

person1 = person2; /\* equivalent to the following \*/

**person1.name = person2.name; /\* maybe a problem \*/**

person1.date\_of\_birth[0] = person2.date\_of\_birth[0];

person1.date\_of\_birth[1] = person2.date\_of\_birth[1];

person1.date\_of\_birth[2] = person2.date\_of\_birth[2];

person1.salary = person2.salary;

# Memory Issue of struct

- We're sitting on a 32-bit Windows PC

```
typedef struct person {  
    char  gender;      // 1 byte  
    char  *name;       // 4 bytes  
    short age;         // 2 bytes  
    int   phone;       // 4 bytes  
} PERSON ;
```

- $\text{sizeof}(\text{PERSON}) = 1+4+2+4 = 11 ?$



# Can we use Pointer Arithmetic ?

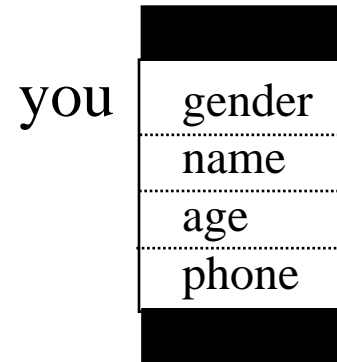
```
struct person {  
    char  gender;    // 1 byte  
    char  *name;     // 4 bytes  
    short age;       // 2 bytes  
    int   phone;     // 4 bytes
```

```
} you;
```

```
char *p = (char *)(&you);
```

// Is this the same as "&(you.gender)" ?

```
++p; // Is this same as "you.name" ?
```



# C 언어 표준이 주는 답

- Structure member의 memory 내의 순서는 structure 선언에 주어진 순서대로 따른다.
- 그러나, memory 내의 address alignment issue에 따라 padding이 들어갈 수 있다.
- 첫 번째 field의 주소는 structure의 첫 번째 주소와 같다

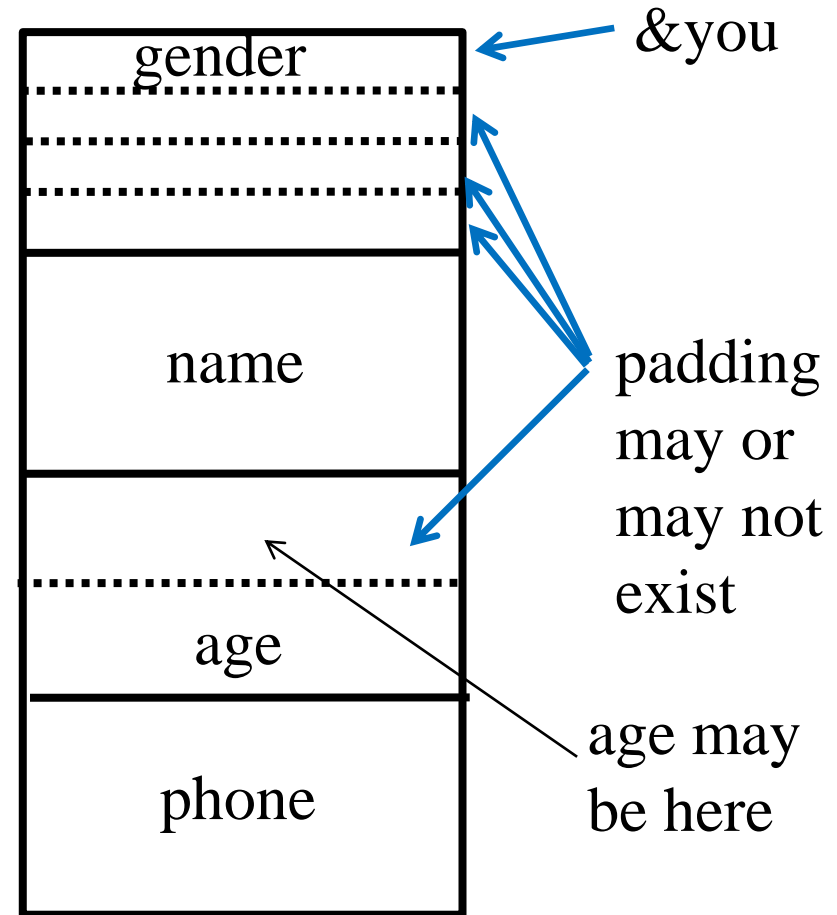
# Memory 내의 배치: 2<sup>n</sup>에 배치

```
struct person {  
    char  gender; // 1 byte  
    char  *name;  // 4 bytes  
    short age;    // 2 bytes  
    int   phone;  // 4 bytes  
} you;
```

```
char *p;  
sizeof(struct person) == 16 ?  
(p = (char *)&you) ==  
    &you.gender
```

```
++p != you.name ?
```

**DO NOT USE POINTER  
ARITHMETIC**

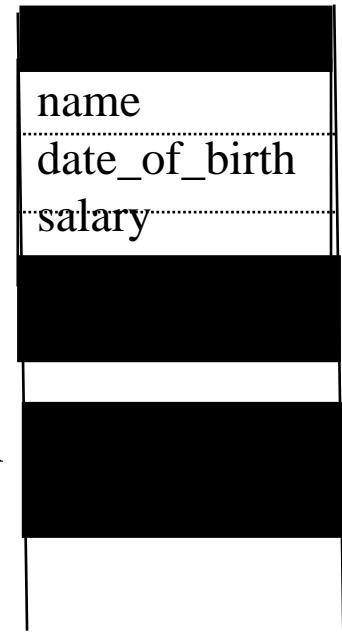


# Dynamic Memory Allocation

```
typedef struct a_person {  
    char *name;  
    int date_of_birth[3];  
    long salary;  
} PERSON;
```

```
PERSON *p_person;
```

p\_person



```
p_person = (PERSON *)malloc(sizeof(PERSON));  
p_person->name = "Min, Hyoung Bok";
```

# Dynamic Memory Allocation

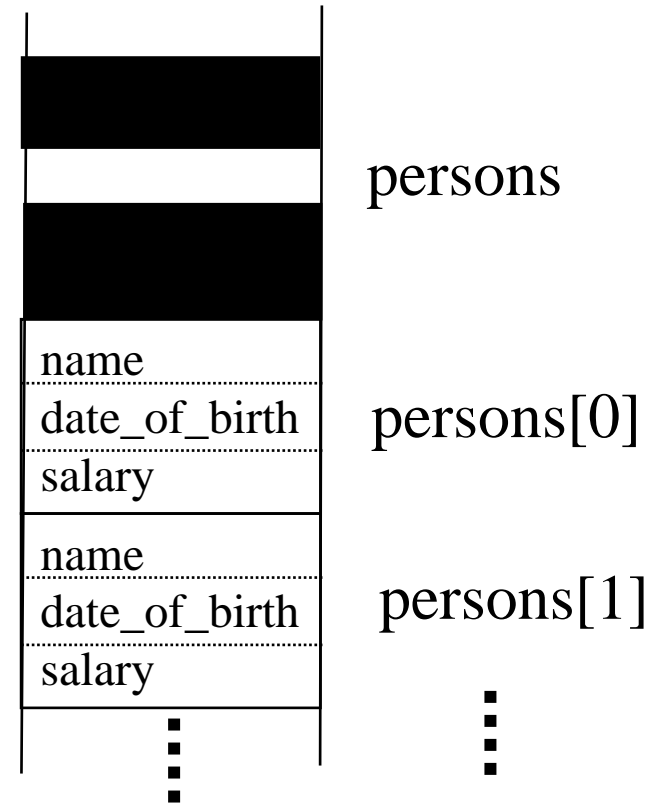
```
typedef struct a_person {  
    char *name;  
    int date_of_birth[3];  
    long salary;  
} PERSON;
```

```
#define ARR_SIZE 100
```

```
PERSON *persons;
```

```
persons = (PERSON *)malloc(  
    ARR_SIZE*sizeof(PERSON));
```

```
persons[0].name = "Min, Hyoung Bok";
```



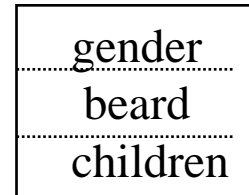
# Union

- Assign different types and names to a single memory space (shares a single memory)
- Use a keyword “union”, and the usage (declaration and member access) is the same as “struct”

```
union a_common { /* tag is optional */  
    char name[32];  
    int year;  
    long salary;  
} common_mem;    /* name of memory shared */
```

# Example of Union

- 남성은 수염의 유무에 대한 정보를, 여성의 경우, 출산 아동 수에 관한 정보를 담는다고 하자.



```
typedef struct person_info {  
    enum {FEMALE, MALE} gender;  
    enum {TRUE, FALSE} beard;  
    int children;  
} PERSON;      /* waste of memory */
```

# Save Memory by using Union

```
typedef struct person_info {  
    enum {FEMALE, MALE} gender;  
    union {  
        int children;  
        enum {TRUE, FALSE} beard;  
    } gender_specific_info;  
} PERSON;
```



```
PERSON persons[10];
```

```
persons[0].gender
```

```
persons[0].gender_specific_info.children = 3;
```

```
persons[0].gender_specific_info.beard = TRUE;
```

➤ Share the same  
memory space