# Chapter 7. Structures

Byoung-Tak Zhang
TA: Hanock Kwak

Biointelligence Laboratory
School of Computer Science and Engineering
Seoul National Univertisy

http://bi.snu.ac.kr

# Structures

- Array
  - A derived type used to represent homogeneous data
- Structure
  - provides a means to aggregate variables of different types

```
struct card {
    int pips; /* 1,2,3,…,13 */
    char suit; /* 'c', 'd', 'h', and 's' */
};
```

- ✓ This declaration creates the derived data type **struct card**.
- ✓ A user-defined type
- ✓ Just a template, no storage allocated

# Structure Variable Declarations

```
struct card {
    int pips;
    char suit;
};
struct card c1, c2;
```

```
struct card {
    int pips;
    char suit;
} c1, c2;
```

```
typedef struct {
    int pips;
    char suit;
} card;
card c1,c2;
```

```
struct card {
    int pips;
    char suit;
};

typedef struct card card;
card c1, c2;
```

# Structures

- The member access operator **.** (dot)
  - ***structure_variable.member_name***
    - **c1.pips = 3;**
    - **c1.suit = 's';**
- Structure assignment
  - **c1 = c2;**

```
typedef struct {
    int pips;
    char suit;
} card;
card deck[52];
```

✓ The identifier **deck** is declared to be an array of **card**.

# Structures

- Within a given structure, member names must be unique.
- Members in different structures can have the same name.

```
struct fruit {
    char *name;
    int calories;
};

struct vegetable {
    char *name;
    int calories;
};
```

# Structures

- If a tag name is not supplied, then the structure type cannot be used in later declarations.

```
struct {
    int day, month, year;
    char day_name[4]; /* Mon, Tue, Wed, etc. */
    char month_name[4]; /* Jan, Feb, Mar, etc. */
} yesterday, today, tomorrow;
```

*vs.*

```
struct date {
    int day, month, year;
    char day_name[4]; /* Mon, Tue, Wed, etc. */
    char month_name[4]; /* Jan, Feb, Mar, etc. */
};
struct date yesterday, today, tomorrow;
```

# Structures

- When using **typedef** to name a structure type, the tag name may be unimportant.

```
typedef struct {
    float re;
    float im;
} complex;

complex a, b, c[100];
```

# Accessing Members of a Structure

[class_info.h]

```
#define CLASS_SIZE 100

struct student{
    char *last_name;
    int student_id;
    char grade;
};
```

[grade.c]

```
#include "class_info.h"

int main()
{
    struct student tmp,
    class[CLASS_SIZE];
    tmp.grade = 'A';
    tmp.last_name = "Casanova";
    tmp.student_id = 910017;
    …
}
```

# Accessing Members of a Structure

[check.c]

```
/* Count the failing grades. */
#include "class_info.h"

int fail(struct student class[])
{
    int i,cnt = 0;
    for (i=0; i<CLASS_SIZE; i++)
        cnt += (class[i].grade == 'F');
}
```

# Accessing Members of a Structure

- The member access operator **->**
  - access the structure members via a pointer

***pointer_to_structure -> member_name***
***⇔ (*pointer_to_structure).member_name***

# Accessing Members of a Structure

[complex.h]

```
struct complex{
    double re;
    double im;
};
typedef struct complex complex;
```

[2_add.c]

```
#include "complex.h"

void add(complex *a, complex *b,
complex *c) /* a = b+c */
{
    a->re = b->re + c->re;
    a->im = b->im + c->im;
}
```

# Accessing Members of a Structure

| Declarations and Initializations | | |
|---|---|---|
| **struct student tmp, *p = &tmp;**<br>**tmp.grade = 'A';**<br>**tmp.last_name = "Casanova";**<br>**tmp.student_id = 910017;** | | |
| Expression | Equivalent expression | Value |
| tmp.grade | p->grade | A |
| tmp.last_name | p->last_name | Casanova |
| (*p).student_id | tmp.student_id | 910017 |
| *p->last_name + 1 | (*(p->last_name)) + 1 | D |
| *(p->last_name + 2) | (p->last_name)[2] | s |

# Using Structures with Functions

- When a structure is passed as an argument to a function, it is passed by a **value**
  - A local copy is made to use in the body of the function.
  - If a structure member is an array, the array gets copied as well.
  - **relatively inefficient !!**

# Using Structures with Functions

```
struct dept {
    char dept_name[25];
    int dept_no;
}

typedef struct {
    char name[25];
    int employee_id;
    struct dept department;
    double salary;
    ….
} employee;
```

# Using Structures with Functions

```
employee update(employee r)
{
    ….
    printf("Department number: ");
    scanf("%d", &n);
    r.department.dept_no = n;
    ….
    return e;
}



employee e;
e = update(e);
```

```
void update(employee_data*p)
{
    ….
    printf("Department number: ");
    scanf("%d", &n);
    p->department.dept_no= n;
    ….
}



employee e;
update(&e);
```

# Initialization of Structures

```
card c = {13, 'h'}; /* the king of hearts */

complex a[3][3] = {
    {{1.0, -0.1}, {2.0, 0.2}, {3.0, 0.3}},
    {{4.0, -0.4}, {5.0, 0.5}, {6.0, 0.6}},
}; /* a[2][] is assigned zeros */

struct home_address {
    char *street;
    char *city_and_state;
    long zip_code;
} address = {"87 West Street", "Aspen, Colorado", 80526};

struct home_address previous_address = {0};
```

# Unions

- A union, like a structure, is a derived type.
- Unions follow the same syntax as structures, but each member has **shared memory**.

```
union what {
    int i;
    float f;
    char c[4];
};
...

printf("%ld", sizeof(union what)); // 4
```

# Unions

- The unions are used to conserve storage by allowing the same space in memory to be used for a variety of types.

```
struct student {
    char name[32];
    int student_num;
    int grade;
};

struct professor {
    char name[32];
    int salary;
    int room_no;
};
```

```
union user {
    struct student stdu;
    struct professor prof;
};
```