Structures

- **Array**
  - A derived type used to represent homogeneous data

- **Structure**
  - provides a means to aggregate variables of different types
    
    ```c
    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
Structures

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

struct card c1, c2;

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

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

card c1, c2;
```
The member access operator .

```
structure_variable.member_name
```

```
c1.pips = 3;
c1.suit = 's';
```

Structure assignment

```
c1 = c2;
```

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

✓ The identifier `deck` is declared to be an array of `struct card`. 
Structures

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

```c
struct fruit {
    char *name;
    int   calories;
};
struct vegetable {
    char *name;
    int   calories;
};
struct fruit a;
struct vegetable b;

a.calories = 100;
b.calories = 120;
```
If a tag name is not supplied, then the structure type cannot be used in later declarations.

```c
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.

```c
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

/* 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';
}

// int fail(struct student *class)
// 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`
    
    `*pointer_to_structure.member_name` ! ERROR
    
    <=> `*(pointer_to_structure.member_name)`
## Accessing Members of a Structure

**[complex.h]**
```c
struct complex{
    double re;
    double im;
};
typedef struct complex complex;
```

**[2_add.c]**
```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

```c
struct student tmp, *p = &tmp;
tmp.grade = 'A';
tmp.last_name = "Casanova";
tmp.student_id = 910017;
```

<table>
<thead>
<tr>
<th>Expression</th>
<th>Equivalent expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>tmp.grade</td>
<td>p-&gt;grade</td>
<td>A</td>
</tr>
<tr>
<td>tmp.last_name</td>
<td>p-&gt;last_name</td>
<td>Casanova</td>
</tr>
<tr>
<td>(*p).student_id</td>
<td>tmp.student_id</td>
<td>910017</td>
</tr>
<tr>
<td>*p-&gt;last_name + 1</td>
<td>(*p-&gt;last_name) + 1</td>
<td>D</td>
</tr>
<tr>
<td>*(p-&gt;last_name + 2)</td>
<td>(p-&gt;last_name)[2]</td>
<td>s</td>
</tr>
</tbody>
</table>
### Operator Precedence and Associativity: A Final Look

<table>
<thead>
<tr>
<th>Operator</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>() [] .</td>
<td>left to right</td>
</tr>
<tr>
<td>++ (postfix)</td>
<td>right to left</td>
</tr>
<tr>
<td>+ (prefix)</td>
<td>left to right</td>
</tr>
<tr>
<td>+ (unary)</td>
<td>left to right</td>
</tr>
<tr>
<td>* / %</td>
<td>left to right</td>
</tr>
<tr>
<td>+ -</td>
<td>left to right</td>
</tr>
<tr>
<td>&lt;&lt; &gt;&gt;</td>
<td>left to right</td>
</tr>
<tr>
<td>&lt; &lt;= &gt; &gt;=</td>
<td>left to right</td>
</tr>
<tr>
<td>== !=</td>
<td>left to right</td>
</tr>
<tr>
<td>&amp;</td>
<td>left to right</td>
</tr>
<tr>
<td>^</td>
<td>left to right</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>left to right</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>:</td>
<td>right to left</td>
</tr>
<tr>
<td>= += -= *= /= etc.</td>
<td>right to left</td>
</tr>
<tr>
<td>, (comma operator)</td>
<td>left to right</td>
</tr>
</tbody>
</table>
Using Structures with Functions

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

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

typedef struct {
    char              name[25];
    int               employee_id;
    struct dept      department;
    struct home_address *a_ptr;
    double            salary;
    ....
} employee_data;
```
Using Structures with Functions

employee_data update(employee_data r)
{
    ....
    printf("Input the department number: ");
    scanf("%d", &n);
    r.department.dept_no = n;
    return e;
}

employee_data e;
e = update(e);

void update(employee_data *p)
{
    ....
    printf("Input the department number: ");
    scanf("%d", &n);
    p->department.dept_no = n;
    return e;
}

void 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 fruit frt = {"plum", 150};

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};