Structures and Union
Review

• bitwise operations
  – you need them for performance in terms of space and time
  – shifts are equivalent to arithmetics

• enumeration
  – you can define a set
  – each member is represented as an integer

• preprocessor directives
  – process your program before it is compiled
Structures

Like `enum`, it may define a new type

Aggregate variables of different types

Each member of a structure can be
- array
- structure
- arrays of structures

```c
struct date { /* structure declaration with a tag name */
    int day, month, year;
    char day_name[4];    /* Mon, Tue, Wed, ... */
    char month_name[4];  /* Jan, Feb, Mar, ... */
}

struct date yesterday, today, tomorrow;
/* variable declaration of the struct date type */
```
Accessing a member

• dot ( . ) operator
  – structure_name.member_name
  – e.g) yesterday.year

• -> operator
  – pointer_to_structure->member_name
  – is same as
  – (*pointer_to_structure).member_name
struct complex {
    double re;  /* real part */
    double im;  /* imag part */
} *a, *b;  /* 2 pointer variables pointing to complex structure */

Accessing members of a structure can be done then by

    a -> re = b -> re;
    a -> im = b -> im;
### Declarations and assignments

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

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

<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>p -&gt; 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>
Using structures

• assignment works (NOT for arrays) as long as two variables are of the same structure type

• structure is more like a primitive type when used as a function parameter
  – call by value – the whole structure is copied
    • inefficient
    • this is one of reasons why you have -> operator
  – if it contains an array, the whole array is copied
• to write a function to update employee information
  1. pass a structure
  2. pass a pointer to structure (this is more efficient because ...)
employee_data update(employee_data e)
{
    printf("Input the department number: ");
    scanf("%d", &n);
    e.department.dept_no = n;
    return e;
}

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

complex a[3][3] = { /* 2 dimensional array of stru */
    {{1.0, -0.1}, {2.0, 0.2}, {3.0, 0.3}}, /* a[0][] */
    {{4.0, -0.4}, {5.9, 0.8}, {6.6, 4.7}}, /* a[1][] */
}; /* a[2][] is initialized to 0 */
Initialization 2

```c
struct home address {
    char *street;
    char *city_and_state;
    long zip_code;
} address ={"285 Old Westport Road", "North Dartmouth, Massachusetts", 02747}

struct home address previous address = {0};
/* All members of a structure will be initialized to zero. It causes pointer members to be initialized with the pointer value NULL and array members (if any) to have their elements initialized to zero. */
```
unions

union int_or_float {
    int     i;
    float   f;
};

• similar to structure, but
• it defines a set of alternative values that may be stored in a shared location

• The programmer is responsible for interpreting the value correctly
Unions

• to access a union member
  
• the members of a structure and or a union can be array, structure, union
#include <stdio.h>

typedef union int_or_float {
    int i;
    float f;
} number;

int main(void)
{
    number n;
    n.i = 4444;
    printf("i: %10d     f: %16.10e
", n.i, n.f);
    n.f = 4444.0;
    printf("i: %10d     f: %16.10e
", n.i, n.f);
    return 0;
}
bit field

A bit field is an `int` or `unsigned` member of a structure or a union

- bit field may be unnamed
- unnamed bit field of width 0 is for alignment of the next word

- restrictions
  - array of bit fields
  - address operator &

```c
struct floating_number {
    unsigned      sign_bit : 1,
    exponent : 8,
    significand : 23;
}
```
```c
#include <limits.h>
#include <stdio.h>

typedef struct {
    unsigned   b0 : 8, b1 : 8, b2 : 8, b3 : 8;
} word_bytes;

typedef struct {
    unsigned
        b0 : 1, b1 : 1, b2 : 1, b3 : 1, b4 : 1, b5 : 1, b6 : 1,
        b7 : 1, b8 : 1, b9 : 1, b10 : 1, b11 : 1, b12 : 1, b13 : 1,
        b14: 1, b15 : 1, b16 : 1, b17 : 1, b18 : 1, b19 : 1, b20 : 1,
        b21: 1, b22 : 1, b23 : 1, b24 : 1, b25 : 1, b26 : 1, b27 : 1,
        b28: 1, b29 : 1, b30 : 1, b31;
} word_bits;

typedef union {
    int             i;
    word_bits     bit;
    word_bytes    byte;
} word;
```