Course organization

- Course introduction (Week 1)
  - Code editor: Emacs
- Part I: Introduction to C programming language (Week 1 - 12)
  - Chapter 1: Overall Introduction (Week 1-4)
    - C
    - Unix/Linux
  - Chapter 2: Types, operators and expressions (Week 4)
  - Chapter 3: Control flow (Week 5, 6)
  - Chapter 4: Functions and program structure (Week 6- 7)
  - **Chapter 5: Pointers and arrays (Week 8-9)**
  - Chapter 6: Structures (Week 10)
  - Chapter 7: Input and Output (Week 11)
- Part II: Skills others than programming languages (Week 12- 14)
  - Debugging tools（Week 12-13）
  - Keeping projects documented and manageable （Week 14）
  - Source code managing （Week 14）
- Part III: Reports from the battle field (student forum) (Week 15 – 16)

Chapter 6 Structures

Chaochun Wei

Shanghai Jiao Tong University

Spring 2014

Contents

- 6.1 Basic of structures
- 6.2 Structures and Functions
- 6.3 arrays of Structures
- 6.4 Pointers to Structures
- 6.5 Self-referential structures
- 6.6 Table lookup
- 6.7 Typedef
- 6.8 Unions
- 6.9 Bit-fields

6.1 Basics of structures

- Structure: a collection of one or more variables grouped under a single name
    - Variables (members) can be different types
    - Examples:

```
struct point {
        int x;
        int y;
};
```

```
struct Employee {
        char *Name;
        char *Address;
        char *ID;
         int    Salary;
         ….
};
```

- Keyword: ***struct***

6.1 Basic of structures

- **A struct declaration defines a type.**

  e.g.: `struct point {int x; int y} x, y, z;`

- Access a member of a structure: *structure-name.member*

  e.g.:
  ```
  struct point pt; pt = {1, 100};
  printf("%d, %d", pt.x, pt.y);
  ```

- *A Structure of structures*
  - *E.g.:*

  ```
  struct rect {
          struct point pt1;
          struct point pt2;
  };
  ```

6.2 Structures and Functions

Operations of structures

- Copy
- Assign
- &
- Access to its members (`. or ->`)
  - ***st.member***
  - Pointer version: ***pt->member***

Precedence of operations

- *`. and -> have top precedence`*
  - ***E.g.,***

    ++p -> len

    increases len, not p.

See more details in hands-on experiment 6.2

6.2 Structures and Functions

- Pass structure to functions by passing
  - members separately
  - a structure
  - a pointer to a structure

See more details in hands-on experiment 6.2

6.2 Structures and Functions

Pointers to structures

```
struct  point *pp;
pp = &origin;
printf(“origin is (%d, %d)\n”, ((*pp).x, (*pp).y);
/* the same as */
printf(“origin is (%d, %d)\n”, (pp->x, pp->y);
```

See more details in hands-on experiment 6.2

6.3 Array of structures

- Array of structures

```
/* Array of points */

struct point {
  int x;
  int y;
};

struct point pts[5];
```

- Function sizeof ( )
  - Sizeof object
  - sizeof(type_name)

returns the size of object and the type type_name

6.4 Pointers to structures

- Similar to simple types
- The size of a structure is not the sum of its members’
  - exmple
    - Struct{

      char c;

      int I;

      };

    // size of this structure may be not 5 bytes, but 8 bytes.

More details can be found in hands-on experiment 6.2 and 6.5

6.5 Self-referential structures

Recursive declaration of a structure

- E.g.,

```
struct tnode {
    char *word;       /* point to the text */
    int count;        /* number of occurrences */
    struct tnode *left;  /* left child */
    struct tnode *right; /* right child */
};
```

More details can be found in hands-on experiment 6.5

6.7 Typedef

- Creating new data type names
  - E.g1:

```
typedef int length;
length len, maxline;
length *lengths[];
```

  - E.g 2:

```
typedef struct tnode{
  char *word;
  int count;
 struct tnode *left;
  struct tnode *right;
} Treenode;
```

6.8 Union

- A variable holds (at different times) objects of different types and sizes
  - The compiler keeps track of size and types
  - A way to manipulate different types of data in a single area of storage
  - Big enough to hold the “widest” member
  - E.g.

```
union u_tag {
     int    ival;
     float  fval;
     char   *sval;
} u;
```

6.9 Bit-fields

- Pack multiple objects into a single machine word
  - Storage efficient
  - External-imposed data format
  - E.g.,

```
Struct {
   unsigned int is_keyword: 1;
   unsigned int is_extern:  1;
   unsigned int is_static:  1;
} flags;
```