Lab 4: Structure and dynamic structure

Programming Lab (Using C)

LIU, Yannan

ynliu@cse.cuhk.edu

Outline

- Introduction to structure
- Structure declaration
- Accessing structure members
- Structure Assignment
- Miscellaneous topics
  - Point, array, and Nested structure
- Dynamic structure

Introduction to Structure

- A *structure* is a collection of related storage elements, possibly of different types, under a single name.

```
struct student {
        char student_id[9];
        char name[26];
        double gpa;
};
```

```
struct teacher {
        char teacher_id[9];
        char name[26];
        int salary;
};
```

- To use a structure
  - Structure type declaration
    - e.g. Student or teacher
  - Structure variable declaration
    - e.g. student Tom, student Jerry, Teacher Green

Structure type declaration

- A *Structure type* can be defined using the keyword **`struct`**:

```
/* define a NEW type for storing data LATER */
struct student {
    char    id[9];
    char    name[26];
    double  gpa;
};
```

- This defines a new data **TYPE** called **`struct student`**, which consists of three *related* ***members***, **`id`**, **`name`** and **`gpa`**.

- However, NO variable storage has YET been allocated.

- This is just the *design* of the new type.

More Structure Examples

- 2D Coordinates

```
struct coord2D {
   double  x;
   double  y;
};
```

- Employee record

```
struct employee {
   char    name[50];
   double  salary[12];
   double  MPF_contrib;
};
```

- Quadratic form Ax^2+Bx+C

```
struct quad_form {
   double  A, B, C;
};
```

- Polynomial $Ax^n+Bx^{n-1}+\ldots+E$

```
struct polynomial {
   unsigned degree;
   double   coeff[5];
};
```

Structure variable declaration

```
struct student {
        char student_id[9];
        char name[26];
        double gpa;
};
```

Structure Type Declaration

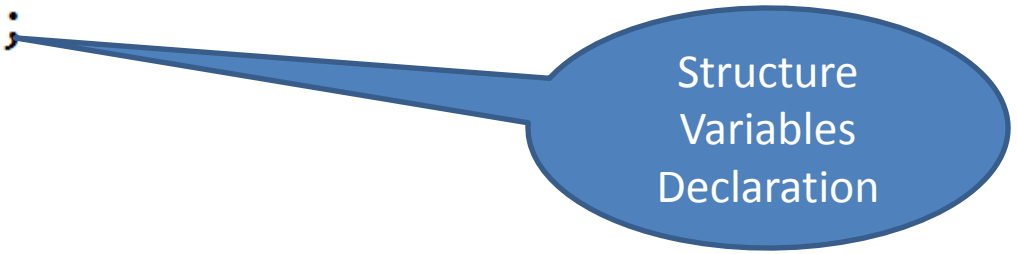
```
struct student Tom, Jerry;
```

Structure Variables Declaration

Structure variable declaration Cont'd

- Defining structure with "**typedef**"

```
typedef struct {
        char teacher_id[9];
        char name[26];
        int salary;
} teacher_t;

teacher_t Green;
```



Structure variable declaration Initializtion

- **`struct`** variables can be initialized as follows:

```
struct date {
  int day, month, year;
};

struct date   today = {25, 12, 1997};
```

Same as
```
today.day = 25,
today.month = 12,
today.year = 1997;
```

- The initial values should be constant values or constant expressions. I.e., no variables should be involved in the ***initializer*** expressions.

```
int i = 1997;

struct date   today = {25, 12, i }; // this is WRONG!
```

- If fewer initializers are listed than the number of members in the **`struct`**, values of the *remaining members are undefined*.

```
struct date   today = {25, 12}; // today.year is undefined
```

Same as
```
today.day = 25,   today.month = 12,   today.year = ?;
```

Structure variable declaration Initialization Cont'd

```
struct employee {
   char     name[50];
   double   salary[12];
   double   MPF_contrib;
};

struct employee peter = {
   "Peter Pan",
   { 9500,  9500, 10000, 11000, 12000, 12050,
    12100, 13000, 13000, 13000, 14000, 14000},
   0.05
};
```

Accessing structure members

- The ***Member*** Operator:
  - Members of a **`struct`** variable can be *accessed* using the member operator,

    i.e. the DOT .

  - For example,

```
variable.member = 999;
```

Accessing structure members Cont'd

```
1  #include <stdio.h>
2
3  struct date {
4    int day;
5    int month;                          // Define the type struct date
6    int year;
7  };
8
9  int main(void)
10 {
11   struct date   today;          // today is a variable of type struct date
12   today.day   = 25;
13   today.month = 12;                   // Use of the member operator .
14   today.year  = 1997;
15   if (today.day == 25   &&   today.month == 12)
16     printf("Merry Christmas!\n");
17   return 0;
18 }
```

Accessing structure members Cont'd

```
Your surname? Collins
Your forename? Phil
Dear P. Collins
I am CSE Admin.
```

```
1  #include <stdio.h>
2  typedef{
3    char surname[31];
4    char forename[31];
5    int  age;
6  } person_t;
7  int main(void) {
8    person_t admin= { "CSE", "Admin", 50 };
9    person_t user;
10   printf("Your surname? ");
11   gets(user.surname);
12   printf("Your forename? ");
13   gets(user.forename);
14   printf("Dear %c. %s\n", user.forename[0], user.surname);
15   printf("I am %s %s.\n", admin.surname, admin.forename);
16   return 0;
17 }
18
```

// Define the type struct person

Structure Assignments

- To assign the value of a structure variable to another structure variable of the same type, the assignment operator '=' can be applied to structure variables.

```c
#include <stdio.h>

struct student {
        char student_id[9];
        char name[26];
        double gpa;
};

int main()
{
    struct student Tom = {"11445","Tom",3.8};
    struct student Jerry;

    //Structure Variable Assignment
    Jerry = Tom;

    return 0;
}
```

Structure array

- A structure may occur as an array element.

```c
#include <stdio.h>
struct student {
  char id[9];
  char name[30];
  int  mark;
};

int main(void)
{
  struct student   student_list[100];

  student_list[99].mark = 98;

  return 0;
}
```

Nested Structure

```
1  #include <stdio.h>
2  #include <string.h>
3  struct date {
4    int     day, month, year;
5  };
6  struct book {
7    char            author[30], title[50], publisher[30];
8    int             edition;
9    struct date     date_of_pub;
10 };
11 int main(void) {
12   struct book    booklist[100];
13   strcpy(booklist[10].author,    "Al Kelley, Ira Pohl");
14   strcpy(booklist[10].title,     "C By Dissection");
15   strcpy(booklist[10].publisher, "Addison-Wesley");
16   booklist[10].edition = 4;
17   booklist[10].date_of_pub.day   = 1;
18   booklist[10].date_of_pub.month = 10;
19   booklist[10].date_of_pub.year  = 2000;
20   return 0;
21 }
```

Pointer to Structures

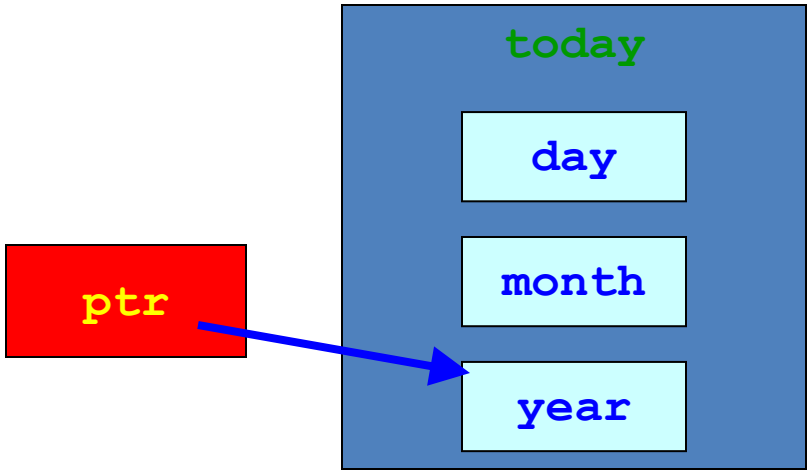
- Revision:

```
int i;
int *ptr;

ptr = &i;      // the address of i

*ptr = 1999;  // i.e. i = 1999
```

- Does the same apply to the ***members*** of a structure?

Pointer to members in Structure

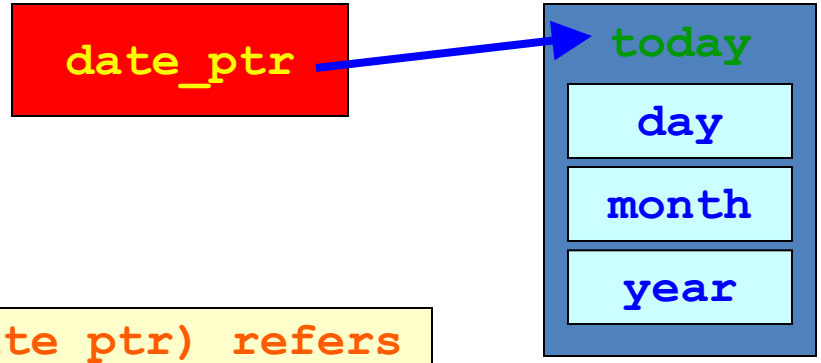
```
struct date {
  int     day, month, year;
};

struct date   today;

int *ptr;

ptr = &today.year;  // the address of today.year

*ptr = 1999;        // i.e. today.year = 1999
```



- A step further, can we have a ***Pointer to a Structure***?

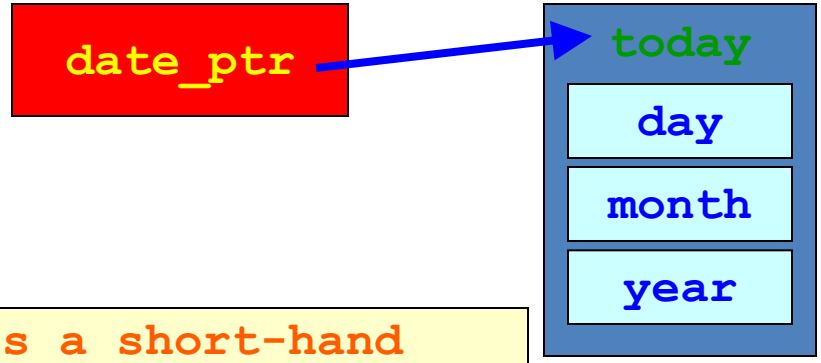
Pointer to Structures

```
1  #include <stdio.h>
2
3  struct date {
4    int day, month, year;                    // Define the type struct date
5  };
6
7  int main(void)
8  {
9    struct date   today;
10   struct date  *date_ptr;
11   date_ptr     = &today;
12   (*date_ptr).day   = 25;   // (*date_ptr) refers
13   (*date_ptr).month = 12;   //    today
14   (*date_ptr).year  = 1997;
15   if (today.day == 25   &&   today.month == 12)
16     printf("Merry Christmas!\n");
17   return 0;
18 }
```



Pointer to Structures

```
1  #include <stdio.h>
2
3  struct date {
4    int day, month, year;                    // Define the type struct date
5  };
6
7  int main(void)
8  {
9    struct date   today;
10   struct date  *date_ptr;
11   date_ptr    = &today;
12   date_ptr->day   = 25;
13   date_ptr->month = 12;
14   date_ptr->year  = 1997;
15   if (today.day == 25   &&   today.month == 12)
16     printf("Merry Christmas!\n");
17   return 0;
18 }
```




```
// -> is a short-hand
      struct_ptr->member
<-> (*struct_ptr).member
```

Pointer to Structures

- Note on the use of the operator **->**
  - This is a short-hand notation.
  - Usage examples:

```
struct_ptr->member = ...;

... = struct_ptr->member;
```

  - **struct_ptr** must be a pointer referring to a proper structure entity, i.e. storing the address of a structure.

  - **member** should be defined in the structure definition.

Practices for structure

- Define a structure type, which can record student ID, student name, and student age.
- Get the information for five students from console.

ID	Name	Age
11345	Tim	18
60765	John	17
19146	Jerry	20
20984	Lucy	22
57862	William	19

Practices for structure Cont'd

- After collecting all these information,
  - Practice 1
    - Print out the student ID for the each student, whose age is larger than 19.
  - Practice 2
    - Print out the student name for the each student, whose name's second letter is 'e'.
  - Practice 3
    - Print out the student's name, whose age is the largest.

Dynamic Memory Manipulations

- Dynamic Memory Allocation: **`malloc()`**

From the C reference manual:

```
/* header file stdlib.h declares the function */
#include <stdlib.h>

void * malloc(size_t size);
```

- **`size_t`** has been type-defined to **`unsigned int`**.
- size is the number of bytes required in the allocation.
- **`malloc()`** returns a pointer to a block of memory of **`size`** bytes.
- The function call returns **`NULL`** when the allocation failed.

Dynamic Memory Manipulations

- Dynamic Memory De-Allocation: **`free()`**

From the C reference manual:

```
/* header file stdlib.h declares the function */
#include <stdlib.h>

void free(void * ptr);
```

- **`void *`** is a *generic* pointer type, i.e. pointer of **ANY** type.
- ptr is a pointer to a block of memory *previously allocated* by **`malloc()`**.
- It releases the block of memory pointed to by **`ptr`**.
- The function call returns **`nothing`**.

Dynamic Memory for Structures

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  typedef struct {double x, double y} Coordinates;
4                                          // Define a structure type
5  int main(void) {
6     Coordinates point1 = {3.4, -5.9}, *ptr;
7                                // Declare a structure point1 and a pointer ptr
8     ptr = &point1;
9     ptr->x = 3.458,    ptr->y = -5.967;       // Modify point1 via ptr
10
11    ptr = malloc (sizeof(Coordinates));      // Create another structure
12    if ( ptr == NULL) return 0;
13
14    ptr->x = 47.57,    ptr->y = 23.45;
15
16    free(ptr);                                          // Free it after use
17    return 0;
18 }
```

Practice for dynamic structure

- Requirements:
  - Implement the same functions in above practices
  - This time, you should enable the user to specify the student number by himself during run time, instead of a constant ‘5’ fixed in code.

Code Template

- We provide code template for all the practices. You can download it from e-learning system, and just need to complete the missing parts(marked by “/*missing*/”).
- We will post the complete correct code in next week.