Lecture 23: Struct

Class page: https://github.com/tsung-wei-huang/cs1410-40

Dr. Tsung-Wei Huang
Department of Electrical and Computer Engineering
University of Utah, Salt Lake City, UT

Announcement

- ❑ **Final Exam starts on 11/30 and ends on 23:59 PM 12/6**
  - ❑ Take-home exam, same as midterm
  - ❑ Cover all topics
    - 50% concept questions
    - 50% programming questions
  - ❑ Free to discuss with your friends and use internet resources
  - ❑ Never copy solutions
- ❑ **We do not have any more labs**
- ❑ **We will still have lectures on 11/30 and 12/2**

Struct

- ❑ **Structures are aggregate data types—that is, they can be built using elements of several types including other `structs`.**
- ❑ **Example**

```
struct card {
   char *face;
   char *suit;
 };
```

  - ❑ `struct` introduces the definition for structure `card`
  - ❑ `card` is the structure tag and is used to declare variables of the structure type
  - ❑ `card` contains two members of type `char *`
    - These members are `face` and `suit`

Struct

- ❑ **Another Example**

```
struct employee{
    char firstName[20];
    char lastName[20];
    int age;
    char gender;
    double hourlySalary;
};
```

- ❑ **Members of the same structure must have unique names, but two different structures may contain members of the same name without conflict.**
- ❑ **Each structure definition must end with a semicolon.**

Struct

- A structure definition does not reserve any space in memory; rather, it creates a new data type that is used to declare structure variables.
- Structure variables are declared like variables of other types.
- Variables of a given structure type can also be declared by placing a comma-separated list of the variable names between the closing brace of the structure definition and the semicolon that ends the structure definition.
- The structure name is optional.
- If a structure definition does not contain a structure name, variables of the structure type may be declared only between the closing right brace of the structure definition and the semicolon that terminates the structure definition.

Struct

- **Valid operations**
  - Assigning a structure to a structure of the same type
  - Taking the address (&) of a structure
  - Accessing the members of a structure
  - Using the `sizeof` operator to determine the size of a structure

Operators for Struct

- Comparing structures is a syntax error
- Structure members are not necessarily stored in consecutive bytes of memory.
- Sometimes there are “holes” in a structure, because some computers store specific data types only on certain memory boundaries for performance reasons, such as half-word, word or double-word boundaries.
- A word is a standard memory unit used to store data in a computer—usually two bytes or four bytes and typically four bytes on today’s popular 32-bit systems.

Accessing Struct Members

- **Accessing structure members**
  - Dot operator (`.`) used with structure variables

    ```
    struct card myCard;
    cout << myCard.suit;
    ```

  - Arrow operator (**->**) used with pointers to structure variables

    ```
    struct card *myCardPtr = &myCard;
    cout<< myCardPtr->suit;
    ```

  - `myCardPtr->suit` is equivalent to

    ```
    ( *myCardPtr ).suit
    ```

Passing Struct to Functions

- ❑ There are two ways to pass the information in structures to functions.
- ❑ You can either pass the entire structure or pass the individual members of a structure.
- ❑ By default, structures are passed by value.
- ❑ Structures and their members can also be passed by reference by passing either references or pointers.
- ❑ To pass a structure by reference, pass the address of the structure object or a reference to the structure object.

Union

- **`union`**
  - Memory that contains a variety of objects over time
  - Only contains one data member at a time
  - Members of a `union` share space
  - Conserves storage
- **`union` definitions**
  - Same as struct

```
union Number {
  int x;
  float y;
};
union Number value;
```

Union

```
/* Fig. 10.5: fig10_05.c
   An example of a union */
#include <stdio.h>

/* number union definition */
union number {
   int x;     /* define int x */
   double y; /* define double y */
}; /* end union number */

 int main()
 {
    union number value; /* define union value */

    value.x = 100; /* put an integer into the union */
    printf( "%s\n%s\n%s%d\n%s%f\n\n",
           "Put a value in the integer member",
           "and print both members.",
           "int:   ", value.x,
           "double:\n", value.y );
```

Union

```
   value.y = 100.0; /* put a double into the same union */
   printf( "%s\n%s\n%s%d\n%s%f\n",
          "Put a value in the floating member",
          "and print both members.",
          "int:   ", value.x,
          "double:\n", value.y );

   return 0; /* indicates successful termination */

} /* end main */
```

Union

```
Put a value in the integer member
and print both members.
int:   100
double:
-92559592117433136000000000000000000000000000000000000000000000.000000

Put a value in the floating member
and print both members.
int:   0
double:
100.000000
```

Summary

- Struct
- Union