

Day18 A

Young W. Lim

2017-12-06 Wed

1 Based on

2 Structures

- Initializing Structures
- Accessing Structures
- Type Definition
- Passing Structures to Functions
- Bit Fields

"C How to Program", Paul Deitel and Harvey Deitel

I, the copyright holder of this work, hereby publish it under the following licenses: GNU head Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled GNU Free Documentation License.

CC BY SA This file is licensed under the Creative Commons Attribution ShareAlike 3.0 Unported License. In short: you are free to share and make derivative works of the file under the conditions that you appropriately attribute it, and that you distribute it only under a license compatible with this one.

Initializing Structures

- initialized using initializer lists
- when fewer initializers, the members with no initializers are initialized to zero or NULL
- members of global structure variables are initialized to zero or NULL unless explicitly initialized
- structure variables may be initialized by a structure variable assignment of the same type by individual member assignments

Accessing Members of Structures

- the structure member operator (`.`)
 - accesses a structure member via the structure variable name
- the structure pointer operator (`->`)
 - accesses a structure member via a pointer to the structure variable

```
struct aaa {  
    int a;  
    char b;  
};
```

```
struct aaa A;  
struct aaa *p = &A;
```

```
(*p).a // A.a  
(*p).b // A.b
```

```
p->a  
p->b
```

- a mechanism for creating synonyms for previously defined types
 - to create shorter type names
 - to increase portability
- names for structure types are often defined with typedef
- often used to create synonyms for the basic data types

Using Structures with Functions

- structure variable may be passed to functions
 - by passing individual structure members
 - by passing entire structure (passing by value, default)
 - by passing a pointer to a structure variable (passing by reference)

Passing Arrays of Structures

- arrays of structures is passed to functions
 - by reference (default)
- pass the array name by reference

Passing Structures of Arrays

- structure of an array may be used to pass the array by value
 - because structures are passed by values
 - so its member arrays are passed also by values

- can specify the number of bits where an `unsigned int` or `int` member of structure / union is stored
- better memory utilization by storing data in the minimum number of bits required

Bit Field Width

- `unsigned int / int member_name : integer_width`
- *width* ranges from 0 to the number of bits for `int`

```
#include <stdio.h>

struct aaa {
    unsigned char a:9;
    unsigned int b:7;
};

int main(void) {
    struct aaa A;

    printf("sizeof(A)= %ld \n", sizeof(A));
}

---
```

t.c:4:3: error: width of 'a' exceeds its type
 unsigned char a:9;
 ^

Unnamed Bit Fields

- unnamed bit field
 - for padding bits
- unnamed bit field with a zero width
 - for the alignment of the next bit field on a new storage unit boundary