Day19 A

Young W. Lim

2017-12-05 Tue

Outline

1. Based on

2. Unions, Bitwise Operations, Enumerate
   - Unions
   - Bitwise Operators
   - Enumerations

Based on

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

The size of a union type

- keyword: union
- shares the same storage space (memory location)
- the size of a union type is the largest size of its members
- the members of a union can be of any data type
- only one member of a union can be referenced at a time

Operations over a union variables

- assigning a union variable to another of the same type
- taking the address (&) of a union variable
- accessing union members (. and ->)

Initializing a union variables

- can be initialized in a declaration
  - with the value of the same type
  - as the first union member

Bits of integral operands

- a bit : 0 or 1
- a byte : 8 bits
  - 00000000 ~ 11111111
  - 256 patterns
- to manipulate the bits of the integral operands
  - `char` type
  - `short` type
  - `int` type
  - `long` type
  - `signed`
  - `unsigned`

Bitwise Operators

- bitwise operator
  - bitwise AND (&)
  - bitwise OR (|)
  - bitwise Exclusive OR XOR (^)
  - bitwise NOT (~)
  - left shift («)
  - right shfit (»)

Bitwise Assignment Operators

- bitwise assignment operator s
  - `a &= b` (`a = a & b`)
  - `a |= b` (`a = a | b`)
  - `a ^= b` (`a = a ^ b`)
  - `a ~= b` (`a = a ~ b`)
  - `a <<= b` (`a = a << b`)
  - `a >>= b` (`a = a >> b`)

Mask Operations

- `A = abcdefgh`
- `A` : 8-bit byte
- `a, b, ..., h` : bit variable
- `A & 00001111 = 0000efgh` (mask abcd, extract efgh)
- `A | 00001111 = abcd1111` (extract abcd, mask efgh)
- `A ^ 00001111 = abcd~e~f~g~h` (extract abcd, toggle efgh)

Bitwise AND Operator

```
        i   = 0x87654321 = 1000_0111_0110_0101_0100_0011_0010_0001
        m1  = 0xF0F0F0F0 = 1111_0000_1111_0000_1111_0000_1111_0000
        m2  = 0x0F0F0F0F = 0000_1111_0000_1111_0000_1111_0000_1111
        m3  = 0xFFFF0000 = 1111_1111_1111_1111_0000_0000_0000_0000
        m4  = 0x0000FFFF = 0000_0000_0000_0000_1111_1111_1111_1111

-------------------------------------------------------------------
        i   = 0x87654321 = 1000_0111_0110_0101_0100_0011_0010_0001
        m1  = 0xF0F0F0F0 = 1111_0000_1111_0000_1111_0000_1111_0000
    i & m1  = 0x80604020 = 1000_0000_0110_0000_0100_0000_0010_0000

        i   = 0x87654321 = 1000_0111_0110_0101_0100_0011_0010_0001
        m2  = 0x0F0F0F0F = 0000_1111_0000_1111_0000_1111_0000_1111
    i & m2  = 0x07050301 = 0000_0111_0000_0101_0000_0011_0000_0001

        i   = 0x87654321 = 1000_0111_0110_0101_0100_0011_0010_0001
        m3  = 0xFFFF0000 = 1111_1111_1111_1111_0000_0000_0000_0000
    i & m3  = 0x87650000 = 1000_0111_0110_0101_0000_0000_0000_0000

        i   = 0x87654321 = 1000_0111_0110_0101_0100_0011_0010_0001
        m4  = 0x0000FFFF = 0000_0000_0000_0000_1111_1111_1111_1111
    i & m4  = 0x00004321 = 0000_0000_0000_0000_0100_0011_0010_0001
```

Bitwise OR Operator

```
        i  = 0x87654321 = 1000_0111_0110_0101_0100_0011_0010_0001
        m1 = 0xF0F0F0F0 = 1111_0000_1111_0000_1111_0000_1111_0000
        m2 = 0x0F0F0F0F = 0000_1111_0000_1111_0000_1111_0000_1111
        m3 = 0xFFFF0000 = 1111_1111_1111_1111_0000_0000_0000_0000
        m4 = 0x0000FFFF = 0000_0000_0000_0000_1111_1111_1111_1111

-------------------------------------------------------------------
        i  = 0x87654321 = 1000_0111_0110_0101_0100_0011_0010_0001
        m1 = 0xF0F0F0F0 = 1111_0000_1111_0000_1111_0000_1111_0000
    i | m1 = 0xF7F5F3F1 = 1111_0111_1111_0101_1111_0011_1111_0001

        i  = 0x87654321 = 1000_0111_0110_0101_0100_0011_0010_0001
        m2 = 0x0F0F0F0F = 0000_1111_0000_1111_0000_1111_0000_1111
    i | m2 = 0x8F6F4F2F = 1000_1111_0110_1111_0100_1111_0010_1111

        i  = 0x87654321 = 1000_0111_0110_0101_0100_0011_0010_0001
        m3 = 0xFFFF0000 = 1111_1111_1111_1111_0000_0000_0000_0000
    i | m3 = 0xFFFF4321 = 1111_1111_1111_1111_0100_0011_0010_0001

        i  = 0x87654321 = 1000_0111_0110_0101_0100_0011_0010_0001
        m4 = 0x0000FFFF = 0000_0000_0000_0000_1111_1111_1111_1111
    i | m4 = 0x8765FFFF = 1000_0111_0110_0101_1111_1111_1111_1111
```

Bitwise XOR Operator

```
        i   = 0x87654321 = 1000_0111_0110_0101_0100_0011_0010_0001
        m1  = 0xF0F0F0F0 = 1111_0000_1111_0000_1111_0000_1111_0000
        m2  = 0x0F0F0F0F = 0000_1111_0000_1111_0000_1111_0000_1111
        m3  = 0xFFFF0000 = 1111_1111_1111_1111_0000_0000_0000_0000
        m4  = 0x0000FFFF = 0000_0000_0000_0000_1111_1111_1111_1111

-------------------------------------------------------------------
        i   = 0x87654321 = 1000_0111_0110_0101_0100_0011_0010_0001
        m1  = 0xF0F0F0F0 = 1111_0000_1111_0000_1111_0000_1111_0000
    i ^ m1  = 0x7795B3D1 = 0111_0111_1001_0101_1011_0011_1101_0001

        i   = 0x87654321 = 1000_0111_0110_0101_0100_0011_0010_0001
        m2  = 0x0F0F0F0F = 0000_1111_0000_1111_0000_1111_0000_1111
    i ^ m2  = 0x886A4C2E = 1000_1000_0110_1010_0100_1100_0010_1110

        i   = 0x87654321 = 1000_0111_0110_0101_0100_0011_0010_0001
        m3  = 0xFFFF0000 = 1111_1111_1111_1111_0000_0000_0000_0000
    i ^ m3  = 0x789A4321 = 0111_1000_1001_1010_0100_0011_0010_0001

        i   = 0x87654321 = 1000_0111_0110_0101_0100_0011_0010_0001
        m4  = 0x0000FFFF = 0000_0000_0000_0000_1111_1111_1111_1111
    i ^ m4  = 0x8765BCDE = 1000_0111_0110_0101_1011_1100_1101_1110
```

Shift Operator

- << (left shift operator)
  - `a << b` : shift `a` by to the left by `b` bits
  - the old msb is shifted off and is lost
  - the new lsb is filled with 0
- >> (right shift operator)
  - `a >> b` : shift `a` by to the right by `b` bits
  - the new msb is filled with 0
  - the old lsb is shifted off and is lost

Enumeration

- an `enum` defines a set of integer constants represented by identifiers
- Values in an `enum`
  - start with 0, unless specified otherwise,
  - incremented by 1

Enumeration constrains

- the identifiers in an `enum` must be unique
- the value of an `enum` constant can be set explicitly

via assignment in the `enum` definition

- multiple members of an enumeration
  can have the same constant value

Enumeration examples

```
#include <stdio.h>

enum aaa { XAA,   XBB,   XCC };
enum bbb { YAA=3, YBB,   YCC };
enum ccc { ZAA=3, ZBB=5, ZCC=-9 };

int main(void) {
  enum aaa X;
  enum bbb Y;
  enum ccc Z;

  X = XAA; printf("X= %d \n", X);
  X = XBB; printf("X= %d \n", X);
  X = XCC; printf("X= %d \n", X);

  Y = YAA; printf("Y= %d \n", Y);
  Y = YBB; printf("Y= %d \n", Y);
  Y = YCC; printf("Y= %d \n", Y);

  Z = ZAA; printf("Z= %d \n", Z);
  Z = ZBB; printf("Z= %d \n", Z);
  Z = ZCC; printf("Z= %d \n", Z);

}
```

```
X= 0
X= 1
X= 2
Y= 3
Y= 4
Y= 5
Z= 3
Z= 5
Z= -9
```