Day01.C

Unions, Enumerations Bitwise Operators

Young W. Lim

December 9, 2017

0.1 Bitwise Operators

```
::::::::::::::
h2.c
::::::::::::::
#include <stdio.h>

int main(void) {
  int i  = 0x87654321;
  int m1 = 0xF0F0F0F0;
  int m2 = 0x0F0F0F0F;
  int m3 = 0xFFFF0000;
  int m4 = 0x0000FFFF;


  printf("\nBitwise AND --------------------------------\n");
  printf("i = %#010x \n", i);
  printf("m1= %#010x  i & m1= %#010x \n", m1, i & m1);
  printf("m2= %#010x  i & m2= %#010x \n", m2, i & m2);
  printf("m3= %#010x  i & m3= %#010x \n", m3, i & m3);
  printf("m4= %#010x  i & m4= %#010x \n", m4, i & m4);

  printf("\nBitwise OR  --------------------------------\n");
  printf("i = %#010x \n", i);
  printf("m1= %#010x  i | m1= %#010x \n", m1, i | m1);
  printf("m2= %#010x  i | m2= %#010x \n", m2, i | m2);
  printf("m3= %#010x  i | m3= %#010x \n", m3, i | m3);
  printf("m4= %#010x  i | m4= %#010x \n", m4, i | m4);

  printf("\nBitwise XOR --------------------------------\n");
  printf("i = %#010x \n", i);
  printf("m1= %#010x  i ^ m1= %#010x \n", m1, i ^ m1);
  printf("m2= %#010x  i ^ m2= %#010x \n", m2, i ^ m2);
  printf("m3= %#010x  i ^ m3= %#010x \n", m3, i ^ m3);
  printf("m4= %#010x  i ^ m4= %#010x \n", m4, i ^ m4);

}

::::::::::::::
h2.out
::::::::::::::

Bitwise AND --------------------------------
i = 0x87654321
m1= 0xf0f0f0f0  i & m1= 0x80604020
m2= 0x0f0f0f0f  i & m2= 0x07050301
m3= 0xffff0000  i & m3= 0x87650000
m4= 0x0000ffff  i & m4= 0x00004321

Bitwise OR  --------------------------------
```

```
i = 0x87654321
m1= 0xf0f0f0f0  i | m1= 0xf7f5f3f1
m2= 0x0f0f0f0f  i | m2= 0x8f6f4f2f
m3= 0xffff0000  i | m3= 0xffff4321
m4= 0x0000ffff  i | m4= 0x8765ffff

Bitwise XOR -------------------------------
i = 0x87654321
m1= 0xf0f0f0f0  i ^ m1= 0x7795b3d1
m2= 0x0f0f0f0f  i ^ m2= 0x886a4c2e
m3= 0xffff0000  i ^ m3= 0x789a4321
m4= 0x0000ffff  i ^ m4= 0x8765bcde



        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

-----------------------------------------------------------------
        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

-----------------------------------------------------------------
        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
```

0.2 Bit Field and Union

```
::::::::::::::
h1.c
::::::::::::::
#include <stdio.h>


struct aaa {
  unsigned char a:4;
  unsigned char b:8;
  unsigned char c:4;
  unsigned      d:16;
} ;


struct bbb {
  unsigned short a:4;
  unsigned short b:8;
  unsigned short c:4;
  unsigned short d:16;
} ;

struct ccc {
```

```
  unsigned      a:4;
  unsigned      b:8;
  unsigned      c:4;
  unsigned      d:16;
} ;

union ddd {
  int   a;
  short b0, b1;
  char  c0, c1, c2, c3;
};

union eee {
  int   a;
  short b[2];
  char  c[4];
};


int main(void) {
  struct aaa A;
  struct bbb B;
  struct ccc C;
  union  ddd D;
  union  eee E;

  A.a = B.a = C.a = 0xf;
  A.b = B.b = C.b = 0xab;
  A.c = B.c = C.c = 0xc;
  A.d = B.d = C.d = 0x1234;


  printf("------------------\n");
  printf("A.a= %hhx \n", A.a);
  printf("A.b= %hhx \n", A.b);
  printf("A.c= %hhx \n", A.c);
  printf("A.d= %hx  \n", A.d);

  printf("------------------\n");
  printf("B.a= %hhx \n", B.a);
  printf("B.b= %hhx \n", B.b);
  printf("B.c= %hhx \n", B.c);
  printf("B.d= %hx  \n", B.d);

  printf("------------------\n");
  printf("C.a= %hhx \n", B.a);
  printf("C.b= %hhx \n", B.b);
  printf("C.c= %hhx \n", B.c);
  printf("C.d= %hx  \n", B.d);
```

```
  printf("------------------\n");
  printf("sizeof(A)= %ld \n", sizeof(A));
  printf("sizeof(B)= %ld \n", sizeof(B));
  printf("sizeof(C)= %ld \n", sizeof(C));

  D.a = E.a = 0x12345678;

  printf("------------------\n");
  printf("D.b0= %hx \n", D.b0);
  printf("D.b1= %hx \n", D.b1);

  printf("------------------\n");
  printf("D.c0= %hx \n", D.c0);
  printf("D.c1= %hx \n", D.c1);
  printf("D.c2= %hx \n", D.c2);
  printf("D.c3= %hx \n", D.c3);

  printf("------------------\n");
  printf("E.b[0]= %hx \n", E.b[0]);
  printf("E.b[1]= %hx \n", E.b[1]);

  printf("------------------\n");
  printf("E.c[0]= %hx \n", E.c[0]);
  printf("E.c[1]= %hx \n", E.c[1]);
  printf("E.c[2]= %hx \n", E.c[2]);
  printf("E.c[3]= %hx \n", E.c[3]);

  printf("------------------\n");
  printf("sizeof(D)= %ld \n", sizeof(D));
  printf("sizeof(E)= %ld \n", sizeof(E));

}

::::::::::::::
h1.out
::::::::::::::
------------------
A.a= f
A.b= ab
A.c= c
A.d= 1234
------------------
B.a= f
B.b= ab
B.c= c
B.d= 1234
------------------
C.a= f
C.b= ab
C.c= c
```

```
C.d= 1234
------------------
sizeof(A)= 8
sizeof(B)= 4
sizeof(C)= 4
------------------
D.b0= 5678
D.b1= 5678
------------------
D.c0= 78
D.c1= 78
D.c2= 78
D.c3= 78
------------------
E.b[0]= 5678
E.b[1]= 1234
------------------
E.c[0]= 78
E.c[1]= 56
E.c[2]= 34
E.c[3]= 12
------------------
sizeof(D)= 4
sizeof(E)= 4
```

Bit Field

- bit field sizes and the sizes of the structures

type	a	b	c	d	sizeof
`struct aaa A;`	4	8	4	16	8
`struct bbb B;`	4	8	4	16	4
`struct ccc C;`	4	8	4	16	4

- the byte sizes

`unsigned char`	1
`unsigned short`	2
`unsigned int`	4

- `struct aaa A;` : 64 bits (= 8 bytes)

```
1st unsigned char (8) :  4 bits for padding,  4 bits for a,
2nd unsigned char (8) :                       8 bits for b
3rd unsigned char (8) :  4 bits for padding,  4 bits for c,
4th unsigned char (8) :  8 bits for padding
1st unsigned int (32) : 16 bits for padding, 16bits for d,
```

- `struct bbb B;` : 32 bits (= 4 bytes)

```
1st unsigned short (16) :  4 bits for a
                           8 bits for b
                           4 bits for c
2nd unsigned short (16) : 16 bits for d
```

- `struct ccc C;` : 32 bits (= 4 bytes)

```
1st unsigned int (32) :  4 bits for a
                         8 bits for b
                         4 bits for c
                        16 bits for d
```

- `h`

the length modifier

- from the linux man page
- integer conversion : `d, i, o, u, x` conversion.
- hh (1-byte length)

  A following integer conversion corresponds to a signed char or unsigned char argument
- h (2-byte length)

  A following integer conversion corresponds to a short int or unsigned short int argument

0.3 Macros

```
::::::::::::::
h2.c
::::::::::::::
#include <stdio.h>

// macros from
// https://stackoverflow.com/questions/1044654/bitfield-manipulation-in-c

#define SET_BIT(val, bitIndex)     val |=  (1 << bitIndex)
#define CLEAR_BIT(val, bitIndex)   val &= ~(1 << bitIndex)
#define TOGGLE_BIT(val, bitIndex)  val ^=  (1 << bitIndex)
#define IS_BIT_SET(val, bitIndex) (val &   (1 << bitIndex))


int main(void) {
  unsigned char a = 0xf0;

  SET_BIT(a, 0);
  printf("a= 0x%02hhx \n", a);

  SET_BIT(a, 1);
```

```
  printf("a= 0x%02hhx \n", a);

  CLEAR_BIT(a, 7);
  printf("a= 0x%02hhx \n", a);

  CLEAR_BIT(a, 6);
  printf("a= 0x%02hhx \n", a);

  TOGGLE_BIT(a, 5);
  printf("a= 0x%0hhx \n", a);

  TOGGLE_BIT(a, 4);
  printf("a= 0x%02hx \n", a);

  printf("IS_BIT_SET(a,0)= %d \n", IS_BIT_SET(a,0));

}

:::::::::::::::
h2.out
:::::::::::::::
a= 0xf1
a= 0xf3
a= 0x73
a= 0x33
a= 0x13
a= 0x03
IS_BIT_SET(a,0)= 1
```

SET_BIT

SET_BIT	$b_7b_6b_5b_4b_3b_2b_1b_0$	0	$b_7b_6b_5b_4b_3b_2b_1 1$
SET_BIT	$b_7b_6b_5b_4b_3b_2b_1b_0$	1	$b_7b_6b_5b_4b_3b_2 1 b_0$
SET_BIT	$b_7b_6b_5b_4b_3b_2b_1b_0$	2	$b_7b_6b_5b_4b_3 1 b_1b_0$
SET_BIT	$b_7b_6b_5b_4b_3b_2b_1b_0$	3	$b_7b_6b_5b_4 1 b_2b_1b_0$
SET_BIT	$b_7b_6b_5b_4b_3b_2b_1b_0$	4	$b_7b_6b_5 1 b_3b_2b_1b_0$
SET_BIT	$b_7b_6b_5b_4b_3b_2b_1b_0$	5	$b_7b_6 1 b_4b_3b_2b_1b_0$
SET_BIT	$b_7b_6b_5b_4b_3b_2b_1b_0$	6	$b_7 1 b_5b_4b_3b_2b_1b_0$
SET_BIT	$b_7b_6b_5b_4b_3b_2b_1b_0$	7	$1 b_6b_5b_4b_3b_2b_1b_0$

CLEAR_BIT

CLEAR_BIT	$b_7b_6b_5b_4b_3b_2b_1b_0$	0	$b_7b_6b_5b_4b_3b_2b_1 0$
CLEAR_BIT	$b_7b_6b_5b_4b_3b_2b_1b_0$	1	$b_7b_6b_5b_4b_3b_2 0 b_0$
CLEAR_BIT	$b_7b_6b_5b_4b_3b_2b_1b_0$	2	$b_7b_6b_5b_4b_3 0 b_1b_0$
CLEAR_BIT	$b_7b_6b_5b_4b_3b_2b_1b_0$	3	$b_7b_6b_5b_4 0 b_2b_1b_0$
CLEAR_BIT	$b_7b_6b_5b_4b_3b_2b_1b_0$	4	$b_7b_6b_5 0 b_3b_2b_1b_0$
CLEAR_BIT	$b_7b_6b_5b_4b_3b_2b_1b_0$	5	$b_7b_6 0 b_4b_3b_2b_1b_0$
CLEAR_BIT	$b_7b_6b_5b_4b_3b_2b_1b_0$	6	$b_7 0 b_5b_4b_3b_2b_1b_0$
CLEAR_BIT	$b_7b_6b_5b_4b_3b_2b_1b_0$	7	$0 b_6b_5b_4b_3b_2b_1b_0$

TOGGLE_BIT

TOGGLE_BIT	$b_7b_6b_5b_4b_3b_2b_1b_0$	0	$b_7b_6b_5b_4b_3b_2b_1\bar{b}_0$
TOGGLE_BIT	$b_7b_6b_5b_4b_3b_2b_1b_0$	1	$b_7b_6b_5b_4b_3b_2\bar{b}_1b_0$
TOGGLE_BIT	$b_7b_6b_5b_4b_3b_2b_1b_0$	2	$b_7b_6b_5b_4b_3\bar{b}_2b_1b_0$
TOGGLE_BIT	$b_7b_6b_5b_4b_3b_2b_1b_0$	3	$b_7b_6b_5b_4\bar{b}_3b_2b_1b_0$
TOGGLE_BIT	$b_7b_6b_5b_4b_3b_2b_1b_0$	4	$b_7b_6b_5\bar{b}_4b_3b_2b_1b_0$
TOGGLE_BIT	$b_7b_6b_5b_4b_3b_2b_1b_0$	5	$b_7b_6\bar{b}_5b_4b_3b_2b_1b_0$
TOGGLE_BIT	$b_7b_6b_5b_4b_3b_2b_1b_0$	6	$b_7\bar{b}_6b_5b_4b_3b_2b_1b_0$
TOGGLE_BIT	$b_7b_6b_5b_4b_3b_2b_1b_0$	7	$\bar{b}_7b_6b_5b_4b_3b_2b_1b_0$

IS_BIT_SET

IS_BIT_SET	$b_7b_6b_5b_4b_3b_2b_1b_0$	0	$0000000b_0$
IS_BIT_SET	$b_7b_6b_5b_4b_3b_2b_1b_0$	1	$000000b_10$
IS_BIT_SET	$b_7b_6b_5b_4b_3b_2b_1b_0$	2	$00000b_200$
IS_BIT_SET	$b_7b_6b_5b_4b_3b_2b_1b_0$	3	$0000b_3000$
IS_BIT_SET	$b_7b_6b_5b_4b_3b_2b_1b_0$	4	$000b_40000$
IS_BIT_SET	$b_7b_6b_5b_4b_3b_2b_1b_0$	5	$00b_500000$
IS_BIT_SET	$b_7b_6b_5b_4b_3b_2b_1b_0$	6	$0b_6000000$
IS_BIT_SET	$b_7b_6b_5b_4b_3b_2b_1b_0$	7	$b_70000000$

printf format string

- prepend `0x` to a hexadecimal number
- 0 flag : zero padding where padding is possible
- 2 : minimum field width

function-like macros

- from gcc documentation
- if you put spaces between the macro name and the parentheses in the macro definition
- no more a function-like macro
- it defines an object-like macro
- a macro beginning with a pair of parentheses.
- ```
  #define lang_init ()    c_init()
  lang_init()
       ==> () c_init()()
  ```
  ```