More detailed

C 程 序 设 计

C Programming

15

位处理

理论课程

厦門大學

XIAMEN UNIVERSITY

信息学院 黄 炜

(特色化示范性软件学院) 博士,副教授

School of Informatics Wei Huang

知识框架

- 位操作符
  - 按位与、按位或、按位反、按位异或
  - 移位：左移、右移
  - 应用：打开、关闭、置反、获取位
- 位字段和联合

内容纲要

值和内存表示

- 声明赋值之后的内存表示

```
int i = 1931505526;
```

  - 内存表示（字节大端序为例）

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
3000	54	68	65	20	71	75	69	63	6B	20	62	72	6F	77	6E	20
3010	66	6F	78	20	6A	75	6D	70	73	20	6F	76	65	72	20	61
3020	20	6C	61	7A	79	20	64	6F	67	2E	0D	0A	5A	D4	1F	33

73	20	6F	76
3018	3019	301A	301B

十六进制表示

  - 位表示（位大端序为例）

0	1	1	1	0	0	1	1	0	0	1	0	0	0	0	0
3018								3019							

0	1	1	0	1	1	1	1	0	1	1	1	0	1	1	0
301A								301B							

二进制表示

- 值、输出和表示形式的关系

位和字节

- 1个字节由8位组成
- 位与具体进制有关
  - 数位：十进制位、八进制位、十六进制位、二进制位
  - 一般编程中，位（bit）指的是二进制位。
- 基本单位
  - 内存中存储信息通常以字节为基本单位
  - 网络中传送信息通常以位为基本单位
- C语言支持在位的层面进行操作

位的相关操作

- 位的相关操作
  - 非取反，与取假，或取真，异或求异，同或求同

操作	符号	A=0, B=0	A=0, B=1	A=1, B=0	A=1, B=1
非	$1-x$	NOT 0 → 1	NOT 1 → 0		
与	$x \cdot y$	0 AND 0 → 0	0 AND 1 → 0	1 AND 0 → 0	1 AND 1 → 1
或	$x+y$	0 OR 0 → 0	0 OR 1 → 1	1 OR 0 → 1	1 OR 1 → 1
异或	$x-y$	0 XOR 0 → 0	0 XOR 1 → 1	1 XOR 0 → 1	1 XOR 1 → 0
同或	$1-(x-y)$	0 XNOR 0 → 1	0 XNOR 1 → 0	1 XNOR 0 → 0	1 XNOR 1 → 1
或非	$1-(x+y)$	0 NOR 0 → 1	0 NOR 1 → 0	1 NOR 0 → 0	1 NOR 1 → 0
与非	$1-x \cdot y$	0 NAND 0 → 1	0 NAND 1 → 1	1 NAND 0 → 1	1 NAND 1 → 0

内容纲要

1	位的概念
2	位运算符
3	位字段和联合

按位取反、与、或和异或

- 位逻辑运算符在二进制层面按位操作
  - 操作数应为整型，不允许浮点型、指针、结构体等

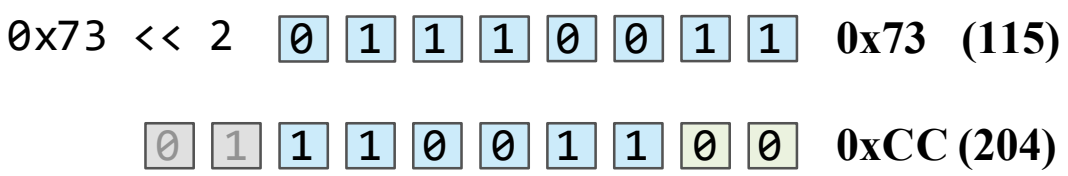
操作	按位反（~）	按位与（&）	按位或（|）	按位异或（^）
示例	~ 0xA9 = 0x56	0xA9 & 0x6C = 0x28	0xA9 | 0x6C = 0xED	0xA9 ^ 0x6C = 0xC5
运算过程	~ 1010 1001 --- 0101 0110	1010 1001 & 0110 1100 --- 0010 1000	1010 1001 | 0110 1100 --- 1110 1101	1010 1001 ^ 0110 1100 --- 1100 0101

- 位操作的结果类型
  - 位操作结果类型一般为整型。
  - 位操作数为长整型的，结果为长整型。

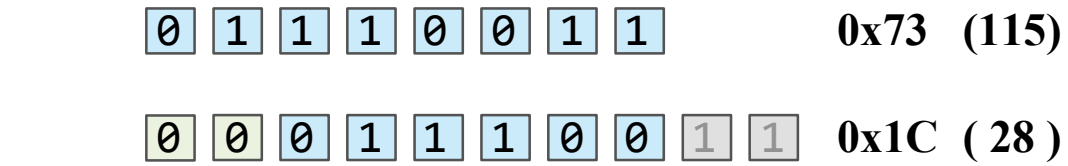
位的左移和右移

- 位的左右移操作符
  - 两侧操作数应为整型，不允许浮点型、指针、结构体等
  - 右侧操作数应为非负整数，否则结果不可控

- 左移（m << n）
  - 将m的每位向左移动n位，右侧补0，左侧截断



- 右移（m >> n）
  - 将m的每位向右移动n位，右侧截断，左侧补位特殊



位操作的类型限定

- 右移（m >> n）的特殊性
  - 左侧操作符为无符号整数或有符号正数，左侧补0。
  - 左侧操作符为有符号的负数，结果未定义。
    - 部分系统补0，部分系统补1
- 编程建议
  - 应避免书写结果未定义的代码
  - 使用结果未定义的代码时，应避免读取结果未定义的部分

位运算与赋值符号结合

- 与赋值符号结合的运算
  - 自与（&=）
  - 自或（|=）
  - 自异或（^=）
  - 自左移（<<=）
  - 自右移（>>=）

```c
/* binbit.c -- using bit operations to display binary */
#include <stdio.h>
#include <limits.h>  // for CHAR_BIT, # of bits per char
char * itobs(int, char *);
void show_bstr(const char *);
int main(void) {
    char bin_str[CHAR_BIT * sizeof(int) + 1];
    int number;
    puts("Enter integers and see them in binary.");
    puts("Non-numeric input terminates program.");
    while (scanf("%d", &number) == 1) {
        itobs(number,bin_str);
        printf("%d is ", number);
        show_bstr(bin_str);
        putchar('\n');
    }
    puts("Bye!");
    return 0;
}
```

```c
char * itobs(int n, char * ps) {
    int i;
    const static int size = CHAR_BIT * sizeof(int);
    for (i = size - 1; i >= 0; i--, n >>= 1)
        ps[i] = (01 & n) + '0';
    ps[size] = '\0';
    return ps;
}
/* show binary string in blocks of 4 */
void show_bstr(const char * str) {
    int i = 0;
    while (str[i]) {  /* not the null character */
        putchar(str[i]);
        if(++i % 4 == 0 && str[i])
            putchar(' ');
    }
}
```

```
Non-numeric input terminates program.
95↲
95 is 0000 0000 0000 0000 0000 0000 0101 1111
-1↲
-1 is 1111 1111 1111 1111 1111 1111 1111 1111
0↲
0 is 0000 0000 0000 0000 0000 0000 0000 0000
q↲
Bye!
```

位运算符的应用

- 掩码
  - 指对目标字段进行位操作的一串二进制代码。
- 掩码的设置
  - 一般将关注的位置1，其他位置0

操作	操作示例	说明
获取位	(flags & MASK) == MASK	获取特定位是否都为1
	(flags & MASK) == 0	获取特定位是否都为0
打开位	flags |= MASK	将特定位全部置为1
关闭位	flags &= ~MASK	将特定位全部置为0
转置位	flags ^= MASK	将特定位全部置反

```c
/* invert4.c -- using bit operations to display binary */
#include <stdio.h>
#include <limits.h>
char * itobs(int, char *);
void show_bstr(const char *);
int invert_end(int num, int bits);
int main(void) {
    char bin_str[CHAR_BIT * sizeof(int) + 1];
    int number;
    puts("Enter integers and see them in binary.");
    puts("Non-numeric input terminates program.");
    while (scanf("%d", &number) == 1) {
        itobs(number,bin_str);
        printf("%d is\n", number);
        show_bstr(bin_str);
        putchar('\n');
        number = invert_end(number, 4);
        printf("Inverting the last 4 bits gives\n");
        show_bstr(itobs(number,bin_str));
        putchar('\n');
    }
```

```
    puts("Bye!");
    return 0;
}
char * itobs(int n, char * ps) {
    int i;
    const static int size = CHAR_BIT * sizeof(int);
    for (i = size - 1; i >= 0; i--, n >>= 1)
        ps[i] = (01 & n) + '0';
    ps[size] = '\0';
    return ps;
}
/* show binary string in blocks of 4 */
void show_bstr(const char * str) {
    int i = 0;
    while (str[i]) { /* not the null character */
        putchar(str[i]);
        if(++i % 4 == 0 && str[i])
            putchar(' ');
    }
}
```

```
int invert_end(int num, int bits)
{
    int mask = 0;
    int bitval = 1;

    while (bits-- > 0)
    {
        mask |= bitval;
        bitval <<= 1;
    }

    return num ^ mask;
}
```

```
Enter integers and see them in binary.
Non-numeric input terminates program.
95↲
95 is
0000 0000 0000 0000 0000 0000 0101 1111
Inverting the last 4 bits gives
0000 0000 0000 0000 0000 0000 0101 0000
168654341↲
168654341 is
0000 1010 0000 1101 0111 0110 0000 0101
Inverting the last 4 bits gives
0000 1010 0000 1101 0111 0110 0000 1010
1↲
1 is
0000 0000 0000 0000 0000 0000 0000 0001
Inverting the last 4 bits gives
0000 0000 0000 0000 0000 0000 0000 1110
q↲
Bye!
```

运算符的优先级

序号	符号	说明	序号	符号	说明
1 →	后缀++ --	后缀增减量	4	+ -	算术运算：加减
	( )	函数调用	5	<< >>	位运算符：左右移
	[ ]	数组下标	6	< > <= >=	关系运算符：大小
	.	结构体联合体成员	7	== !=	关系运算符：相等
	->	结构/联合指针成员	8	&	位运算符：与
	(type){list}	复合文字	9	^	位运算符：异或
2 ←	前缀++ --	前缀增减量	10	|	位运算符：或
	+ -	正负号	11	&&	逻辑运算符：与
	!	逻辑运算符：非	12	||	逻辑运算符：或
	~	位运算符：非	13	?:	三元条件运算符
	(type)	强制类型转换	14 ←	=	赋值
	*	间接寻址		+= -= *= /= %=	自增、自减、自乘、自除、自模
	&	取址		<<= >>= &= ^= |=	自左右移、自与、自异或、自或
	sizeof	存储空间			
3	* / %	算术运算：乘除模	15	,	逗号表达式

运算符的优先级

- 一元＜二元（除赋值和逗号）＜三元＜赋值＜逗号
  - 一元（部分）、三元、赋值为自右向左结合
    - 前缀、正负、非、类型转换、指针、空间
  - 一元（部分）、二元（除赋值外）为自左向右
    - 后缀、指向（函数、数组元素、成员）、复合文字
- 二元：算术＜移位＜关系＜位（除移位外）＜逻辑
  - 算术：乘除模＜加减
  - 关系：大小＜相等
  - 位：与（相当于乘）＜异或（相当于减）＜或（相当于加）
  - 逻辑：与（相当于乘）＜或（相当于加）

内容纲要

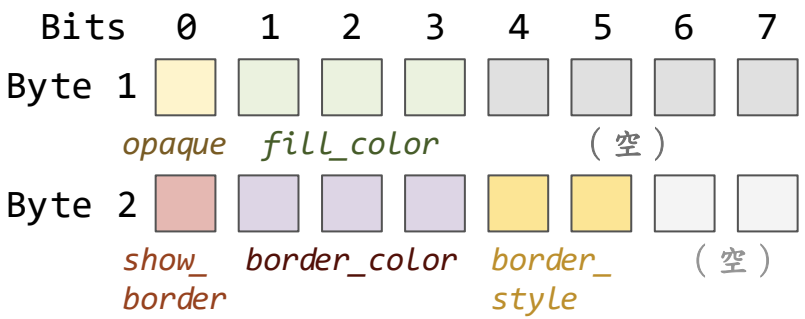
1	位的概念
2	位操作符
3	位字段和联合

位字段

- 位字段是有符号或无符号整型中一组相邻的位

```
struct box_props {
    bool opaque                 : 1;  // or unsigned int (pre C99)
    unsigned int fill_color     : 3;
    unsigned int                : 4;
    bool show_border            : 1;  // or unsigned int (pre C99)
    unsigned int border_color   : 3;
    unsigned int border_style   : 2;
    unsigned int                : 2;
};
```

- 在内存里如何存储？
  - 自低向高、不跨边界
  - 尽量无缝连接、允许无变量名



```c
/* fields.c -- define and use fields */
#include <stdio.h>
#include <stdbool.h>   //C99, defines bool, true, false
/* line styles      */
#define SOLID   0
#define DOTTED  1
#define DASHED  2
/* primary colors   */
#define BLUE    4
#define GREEN   2
#define RED     1
/* mixed colors     */
#define BLACK   0
#define YELLOW  (RED | GREEN)
#define MAGENTA (RED | BLUE)
#define CYAN    (GREEN | BLUE)
#define WHITE   (RED | GREEN | BLUE)
const char * colors[8] = {"black", "red", "green", "yellow",
    "blue", "magenta", "cyan", "white"};
void show_settings(const struct box_props * pb);
```

```c
struct box_props {
    bool opaque                 : 1;   // or unsigned int (pre C99)
    unsigned int fill_color     : 3;
    unsigned int                : 4;
    bool show_border            : 1;   // or unsigned int (pre C99)
    unsigned int border_color   : 3;
    unsigned int border_style   : 2;
    unsigned int                : 2;
};
int main(void) { /* create and initialize box_props structure */
    struct box_props box = {true, YELLOW , true, GREEN, DASHED};
    printf("Original box settings:\n");
    show_settings(&box);
    box.opaque = false;
    box.fill_color = WHITE;
    box.border_color = MAGENTA;
    box.border_style = SOLID;
    printf("\nModified box settings:\n");
    show_settings(&box);
    return 0;
}
```

```
void show_settings(const struct box_props * pb) {
    printf("Box is %s.\n",pb->opaque == true ? "opaque": "transparent");
    printf("The fill color is %s.\n", colors[pb->fill_color]);
    printf("Border %s.\n",pb->show_border == true ? "shown" : "not shown");
    printf("The border color is %s.\n", colors[pb->border_color]);
    printf ("The border style is ");
    switch(pb->border_style) {
        case SOLID  : printf("solid.\n"); break;
        case DOTTED : printf("dotted.\n"); break;
        case DASHED : printf("d
        default     : printf("u
    }
}
```

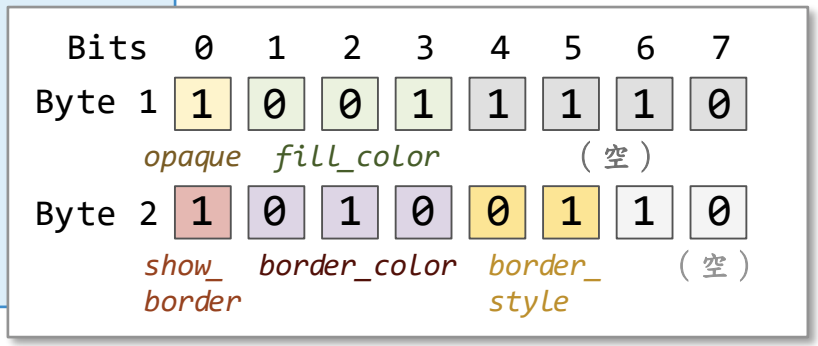
```
Original box settings:
Box is opaque.
The fill color is yellow.
Border shown.
The border color is green.
The border style is dashed.

Modified box settings:
Box is transparent.
The fill color is white.
Border shown.
The border color is magenta.
The border style is solid.
```

位字段与联合

- 位字段的结构体可以和联合体结合

```
struct box_props {
    bool opaque                 : 1;
    unsigned int fill_color     : 3;
    unsigned int                : 4;
    bool show_border            : 1;
    unsigned int border_color   : 3;
    unsigned int border_style   : 2;
    unsigned int                : 2;
};
/* look at data as struct or as unsigned short */
union Views {
    struct box_props st;
    unsigned short us;
};
v.us = 25977;
```

域	值
v.us	0x6579
v.st.opaque	1
v.st.fill_color	4
v.st.show_border	1
v.st.border_color	2
v.st.border_style	2



```c
/* dualview.c -- bit fields and bitwise operators */
#include <stdio.h>
#include <stdbool.h>
#include <limits.h>
/* BIT-FIELD CONSTANTS */
/* line styles      */
#define SOLID   0
#define DOTTED  1
#define DASHED  2
/* primary colors   */
#define BLUE    4
#define GREEN   2
#define RED     1
/* mixed colors     */
#define BLACK   0
#define YELLOW  (RED | GREEN)
#define MAGENTA (RED | BLUE)
#define CYAN    (GREEN | BLUE)
#define WHITE   (RED | GREEN | BLUE)
```

```
/* BITWISE CONSTANTS   */
#define OPAQUE           0x1
#define FILL_BLUE        0x8
#define FILL_GREEN       0x4
#define FILL_RED         0x2
#define FILL_MASK        0xE
#define BORDER           0x100
#define BORDER_BLUE      0x800
#define BORDER_GREEN     0x400
#define BORDER_RED       0x200
#define BORDER_MASK      0xE00
#define B_SOLID          0
#define B_DOTTED         0x1000
#define B_DASHED         0x2000
#define STYLE_MASK       0x3000

const char * colors[8] = {"black", "red", "green", "yellow",
    "blue", "magenta", "cyan", "white"};
```

```
struct box_props {
    bool opaque                    : 1;
    unsigned int fill_color        : 3;
    unsigned int                   : 4;
    bool show_border               : 1;
    unsigned int border_color      : 3;
    unsigned int border_style      : 2;
    unsigned int                   : 2;
};
union Views {    /* look at data as struct or as unsigned short */
    struct box_props st_view;
    unsigned short us_view;
};
void show_settings(const struct box_props * pb);
void show_settings1(unsigned short);
char * itobs(int n, char * ps);
int main(void) {
    /* create Views object, initialize struct box view */
    union Views box = {{true, YELLOW , true, GREEN, DASHED}};
    char bin_str[8 * sizeof(unsigned int) + 1];
```

```
    printf("Original box settings:\n");
    show_settings(&box.st_view);
    printf("\nBox settings using unsigned int view:\n");
    show_settings1(box.us_view);
    printf("bits are %s\n", itobs(box.us_view,bin_str));
    box.us_view &= ~FILL_MASK;          /* clear fill bits */
    box.us_view |= (FILL_BLUE | FILL_GREEN); /* reset fill */
    box.us_view ^= OPAQUE;               /* toggle opacity */
    box.us_view |= BORDER_RED;           /* wrong approach */
    box.us_view &= ~STYLE_MASK;        /* clear style bits */
    box.us_view |= B_DOTTED;         /* set style to dotted */
    printf("\nModified box settings:\n");
    show_settings(&box.st_view);
    printf("\nBox settings using unsigned int view:\n");
    show_settings1(box.us_view);
    printf("bits are %s\n", itobs(box.us_view,bin_str));
    return 0;
}
```

```
void show_settings(const struct box_props * pb) {
    printf("Box is %s.\n", pb->opaque == true ? "opaque":
"transparent");
    printf("The fill color is %s.\n", colors[pb->fill_color]);
    printf("Border %s.\n", pb->show_border == true ? "shown" :
"not shown");
    printf("The border color is %s.\n", colors[pb->border_color]);
    printf("The border style is ");
    switch(pb->border_style) {
        case SOLID  : printf("solid.\n"); break;
        case DOTTED : printf("dotted.\n"); break;
        case DASHED : printf("dashed.\n"); break;
        default     : printf("unknown type.\n");
    }
}
void show_settings1(unsigned short us) {
    printf("box is %s.\n",
           (us & OPAQUE) == OPAQUE? "opaque": "transparent");
    printf("The fill color is %s.\n", colors[(us >> 1) & 07]);
```

```
    printf("Border %s.\n",
           (us & BORDER) == BORDER? "shown" : "not shown");
    printf ("The border style is ");
    switch(us & STYLE_MASK) {
        case B_SOLID  : printf("solid.\n"); break;
        case B_DOTTED : printf("dotted.\n"); break;
        case B_DASHED : printf("dashed.\n"); break;
        default       : printf("unknown type.\n");
    }
    printf("The border color is %s.\n", colors[(us >> 9) & 07]);
}
char * itobs(int n, char * ps) {
    int i;
    const static int size = CHAR_BIT * sizeof(int);
    for (i = size - 1; i >= 0; i--, n >>= 1)
        ps[i] = (01 & n) + '0';
    ps[size] = '\0';
    return ps;
}
```

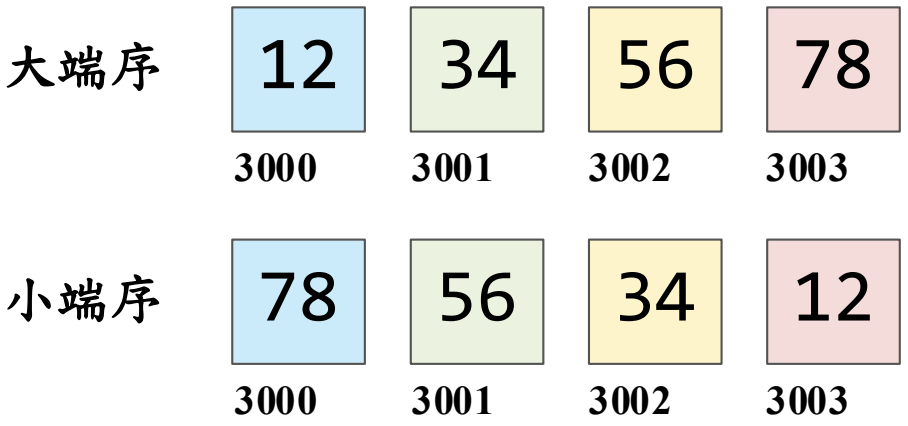
```
Original box settings:
Box is opaque.
The fill color is yellow.
Border shown.
The border color is green.
The border style is dashed.

Box settings using unsigned int view:
box is opaque.
The fill color is black.
Border not shown.
The border style is solid.
The border color is black.
bits are 00000000000000000000000000000001

Modified box settings:
Box is transparent.
The fill color is yellow.
Border shown.
The border color is green.
The border style is dashed.

Box settings using unsigned int view:
box is transparent.
The fill color is cyan.
Border not shown.
The border style is dotted.
The border color is red.
bits are 00000000000000000001001000001100
```

位字段与联合

- 位字段和按位视图的区别
  - 按位视图需要记住位置信息
  - 位字段位置对应关系是依赖于实现的

端序

- 端（尾）
  - 内存，低地址为端；计算机网络，先发送为端
- 大数
  - 数字存储按字节或按位划分，更大的部分为大数
- 端序
  - 大端序：大数的存在尾部
  - 小端序：小数的存在尾部



```c
// align.c -- using _Alignof and _Alignas  (C11)
#include <stdio.h>
int main(void) {
    double dx;
    char ca;
    char cx;
    double dz;
    char cb;
    char _Alignas(double)  cz;
    printf("char alignment:   %zd\n", _Alignof(char));
    printf("double alignment: %zd\n", _Alignof(double));
    printf("&dx: %p\n", &dx);
    printf("&ca: %p\n", &ca);
    printf("&cx: %p\n", &cx);
    printf("&dz: %p\n", &dz);
    printf("&cb: %p\n", &cb);
    printf("&cz: %p\n", &cz);
    return 0;
}
```

```
char alignment:   1
double alignment: 8
&dx: 0028FEE8
&ca: 0028FEE7
&cx: 0028FEE6
&dz: 0028FED8
&cb: 0028FED7
&cz: 0028FED0
```

C 程 序 设 计

C Programming

15

谢谢观看

理论课程

厦門大學

XIAMEN UNIVERSITY

信息学院 黄 炜

(特色化示范性软件学院) 博士,副教授

School of Informatics Wei Huang