位运算

- `&`按位与
- `|`按位或
- `~`按位取反
- `^`按位异或
- `<<`左移
- `>>`右移
    - 逻辑右移：高位补0
    - 算术右移：负数高位补1，整数高位补0，即补最高位

基本推导公式

> a & 0 = 0
> a & 1 = a
> a | 0 = a
> a & 0 = 0
> a & ~a = 0
> a | ~a = 1
> a ^ 0 = a
> a ^ 1 = ~a 可以做到局部求反

```
// 不使用明显的分支实现abs
int abs(int n)
{
    int x = n >> 31;
    n = n ^ x;
    return n - x;
}

// 不使用明显分支实现三目运算 x >= 0 ? 59 : 85
x = x >> 31;
x = x & (85 - 59);
x = x + 59;

// 不使用明显分支实现三目运算 x < 0 ? 20 : 67
x = x >> 31;
x = x & (20 - 67);
x = x + 67;
```

位段

位段是一个`int`或者`unsigned int`类型变量中的一组相邻的位，位段通过一个结构声明来建立，该结构声明为每一个字段提供标签，并确定该字段的宽度

```
struct bit_field {
    unsigned int a : 1;  // 占1bit
    int b : 2;           // 占2bit
    int c : 0;           // 占位符，表示该空间被占用完，下一个字段要从下一个空间开始
    int d : 3;           // 占3bit
} bf;
// 通过下面来赋值
bf.a = 1;
bf.b = 2;
bf.d = 7;
```

> 如果声明的总位数超过了类型的大小，则会用到下一个类型存储的位置
> 一个字段不允许跨越两个类型之间的边界，编译器会自动移动跨界的字段，来保持对齐，可以利用指定位段为0来"填充"这段区域

位段参与运算时会自动扩展为4字节（根据所声明的类型）

文件

文本文件与二进制文件

文本文件与二进制文件没有本质区别，只是在人为识别上加以区分。类**Unix**系统上对于两种文件并不区分，而在**windows**上文本文件中有些控制字符在与二进制文件中有所不同

> `\n`在文本文件中以`\r\n`的形式表现
> 在文本文件中，字符`\0x1A`作为终止输入，即`EOF`

FILE结构

```
struct _iobuf {
        char *_ptr;         // 输入的下一个位置
        char *_base;        // 当前缓冲区的位置
        int   _cnt;         // 缓冲区剩余大小
        int   _flag;        // 文件标志
        int   _file;
        int   _charbuf;
        int   _bufsiz;      // 缓冲区大小
        char *_tmpfname;    // 临时文件名
};
```