§14. C知识补充

1. 位运算

1.1. 位运算的基本概念

1.1.1. 字节和位

字节：byte，计算机中数据表示的基本单位

位　：bit，　计算机中数据表示的最小单位

1 byte = 8 bit

1.1.2. 位运算

以bit为单位进行数据的运算

1.1.3. 位运算的基本方法

再次强调：
有符号数：符号位是最高位(0/1)
无符号数：符号位是0

★ 按位进行（只有0、1）

★ 要求运算数据长度相等，若不等，则右对齐，按符号位补齐左边

char a=0x37;	0000 0000 0011 0111
short b=0x1234;	0001 0010 0011 0100
char a=0xA7;	1111 1111 1010 0111
short b=0x8341;	1000 0011 0100 0001
unsigned char a=0xA7;	0000 0000 1010 0111
short b=0x8341;	1000 0011 0100 0001

★ 数在计算机内是用补码表示的

§14. C知识补充

1. 位运算

1.2. 常用的位运算

1.2.1. 与(&)

运算规则：遇0得0

例：char a=3,b=5; 求a&b

```
  0000 0011
& 0000 0101
  0000 0001              a&b=1
```

例：char a=0x87; short b=0x9c52; 求a&b

```
  1111 1111 1000 0111
& 1001 1100 0101 0010
  1001 1100 0000 0010    a&b=0x9c02 (-25598)
```

例：unsigned char a=0x87; short b=0x9c52;求a&b

```
  0000 0000 1000 0111
& 1001 1100 0101 0010
  0000 0000 0000 0010    a&b=0x2
```

例：char a=0xb6,b=0xc2; 求a&b

```
  1011 0110
& 1100 0010
  1000 0010              a&b=0x82  (-126)
```

```
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
    char a1 = 3, b1 = 5;
    cout << "a=" << (int)a1 << " b=" << (int)b1 << " a&b=" << (a1&b1) << endl;

    char a2 = 0x87;
    short b2 = 0x9c52;
    cout << "a=0x" << hex << (int)a2 << " b=0x" << b2  << " a&b=0x" << (a2&b2)
         << " a&b=0x" << short(a2&b2) << " " << dec << " a&b=" << (a2&b2) << endl;

    unsigned char a3 = 0x87;
    short b3 = 0x9c52;
    cout << "a=0x" << hex << (int)a3 << " b=0x" << b3 << " a&b=0x" << (a3&b3) << endl;

    char a4 = 0xb6, b4 = 0xc2;
    cout << "a=0x" << hex << (int)a4 << " b=0x" << (int)b4 << " a&b=0x" << (a4&b4)
         << " " << dec << "a&b=" << (a4&b4) << endl;

    return 0;
}
```

Microsoft Visual Studio 调试控制台　读懂运行结果!!!

```
a=3 b=5 a&b=1
a=0xffffff87 b=0x9c52 a&b=0xffff9c02 a&b=0x9c02  a&b=-25598
a=0x87 b=0x9c52 a&b=0x2
a=0xffffffb6 b=0xffffffc2 a&b=0xffffff82 a&b=-126
```

§14. C知识补充

1. 位运算

1.2. 常用的位运算

1.2.1. 与(&)

运算规则：遇0得0

应用：

★ 清零

例：char a=0xb6;现要求将该数清零，则：

1011 0110

& 0?00 ?00? 要清零数为1的位，本数对应位为0

0000 0000

a&0x0 a&0x1 a&0x8 a&0x9

a&0x40 a&0x41 a&0x48 a&0x49

★ 取指定位

例：char a=0xb6;现要求只保留低4位，

而高4位清0，则：

1011 0110

& 0000 1111 要保留的位，本数对应位为1

0000 0110

```
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
    /* &的应用：清0 */
    char a1=0xb6;
    cout << "char a=" << hex << (int)a1 << endl
         << "     a&0x0 =" << dec << (a1&0x0)  << endl
         << "     a&0x1 =" << dec << (a1&0x1)  << endl
         << "     a&0x8 =" << dec << (a1&0x8)  << endl
         << "     a&0x9 =" << dec << (a1&0x9)  << endl
         << "     a&0x40=" << dec << (a1&0x40) << endl
         << "     a&0x41=" << dec << (a1&0x41) << endl
         << "     a&0x48=" << dec << (a1&0x48) << endl
         << "     a&0x49=" << dec << (a1&0x49) << endl;

    /* &的应用：取指定位 */
    char a2=0xb6;
    cout << "char a=0x" << hex << (int)a2
         << " a&0x0F="  << dec << (a2&0x0F)
         << endl;

    return 0;
}
```

读懂运行结果!!!

§14. C知识补充

1. 位运算

1.2. 常用的位运算

1.2.2. 或(|)

运算规则：遇1得1

例：char a=3,b=5; 求a|b

```
  0000 0011
| 0000 0101
  0000 0111     a|b=7
```

例：char a=3; short b=5;求a|b

```
  0000 0000 0000 0011
| 0000 0000 0000 0101
  0000 0000 0000 0111     a|b=7
```

例：char a=0xb6,b=0xc2; 则a|b

```
  1011 0110
| 1100 0010
  1111 0110     a|b=0xF6
```

有符号10进制：-10

应用：★ 设定某些位为1

例：char a=0xb6;要求1,4位设为1，其它不变

```
  1011 0110
| 0000 1001     要设置的位，本数对应位为1
  1011 1111     (0xBF)
```

```
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
    char a1=3, b1=5;
    cout << "a=" << (int)a1 << " b=" << (int)b1 << " a|b=" << (a1|b1) << endl;

    char a2=3;
    short b2=5;
    cout << "a=" << (int)a2 << " b=" << b2 << " a|b=" << (a2|b2) << endl;

    char a3=0xb6, b3=0xc2;
    cout << "a=" << hex << (int)a3 << " b=" << (int)b3;
    cout << " a|b=0x" << hex << (a3|b3) << " " << dec << (a3|b3) << endl;

    /* |的应用，将1、4 bit位设为1，其它不变 */
    char a4=0xb6;
    cout << "a=" << hex << (int)a4 << " a|0x9=0x" << (a4|0x9) << endl;

    return 0;
}
```

读懂运行结果!!!

Microsoft Visual Studio 调试控制台

```
a=3 b=5 a|b=7
a=3 b=5 a|b=7
a=ffffffb6 b=fffffffc2 a|b=0xfffffff6 -10
a=ffffffb6 a|0x9=0xffffffbf
```

§14. C知识补充

1. 位运算

1.2. 常用的位运算

1.2.3. 异或(^)

运算规则：相同为0，不同为1

例：char a=3,b=5; 求a^b

```
  0000 0011
^ 0000 0101
  0000 0110     a^b=6
```

例：char a=3; short b=5;求a^b

```
  0000 0000 0000 0011
^ 0000 0000 0000 0101
  0000 0000 0000 0110     a^b=6
```

例：char a=0xb6,b=0xc2; 则a^b

```
  1011 0110
^ 1100 0010
  0111 0100     a^b=0x74
```

有符号10进制：116

```
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
    char a1=3, b1=5;
    cout << "a=" << (int)a1 << " b=" << (int)b1 << " a^b=" << (a1^b1) << endl;

    char a2=3;
    short b2=5;
    cout << "a=" << (int)a2 << " b=" << b2 << " a^b=" << (a2^b2) << endl;

    char a3=0xb6, b3=0xc2;
    cout << "a=" << hex << (int)a3 << " b=" << (int)b3;
    cout << " a^b=0x" << hex << (a3^b3) << " " << dec << (a3^b3) << endl;

    return 0;
}
```

读懂运行结果!!!

```
Microsoft Visual Studio 调试控制台
a=3 b=5 a^b=6
a=3 b=5 a^b=6
a=ffffffb6 b=ffffffc2 a^b=0x74 116
```

§14. C知识补充

1. 位运算

1.2. 常用的位运算

1.2.3. 异或(^)

运算规则：相同为0，不同为1

应用：

★ 特定位翻转（0，1互换）

例：char a=0xb6; 高4位翻转，低4位不变

```
  1011 0110
^ 1111 0000     要翻转的位，本数对应位为1
  0100 0110
```

★ 两数交换

例：char a=0xb6,b=0xc2; 要求a，b互换

三步：a=a^b　　b=b^a　　a=a^b

```
(1) a=1011 0110
    b=1100 0010
    a=0111 0100     a=a^b=0x74
(2) b=1100 0010
    a=0111 0100
    b=1011 0110     b=b^a=0xb6
(3) a=0111 0100
    b=1011 0110
    a=1100 0010     a=a^b=0xc2
```

```
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
    /* ^的应用：特定位翻转 */
    char a1=0xb6;
    cout << "a=" << hex << (int)a1 << " a^0xF0=0x" << (a1^(char)0xF0) << endl;

    /* ^的应用：两数交换 */
    char a2=0xb6, b2=0xc2;
    cout << "a=" <<hex<<(int)a2<<" b=" <<(int)b2<< endl;
    a2 = a2^b2;
    b2 = b2^a2;
    a2 = a2^b2;
    cout << "a=" <<hex<<(int)a2<<" b=" <<(int)b2<< endl;

    return 0;
}
```

读懂运行结果!!!

```
Microsoft Visual Studio 调试控制台
a=ffffffb6 a^0xF0=0x46
a=ffffffb6 b=ffffffc2
a=ffffffc2 b=ffffffb6
```

§14. C知识补充

1. 位运算

1.2. 常用的位运算

1.2.3. 异或(^)

运算规则：相同为0，不同为1

应用：

★ 简单密码传送

甲：持有secret_key
乙：持有secret_key
第三方：无法知道secret_key

甲：要发送的情报
encrypt(msg, secret_key, encryped_msg);
得到的 encryped_msg 用文件/明码等各种形式传输

乙：收到公共方式传输得到的 encryped_msg 后
decrypted(encryped_msg, secret_key, decryped_msg);
得到decryped_msg

第三方：收到 encryped_msg 后，看不懂

```
#include <iostream>
using namespace std;
void encrypt(const char* msg, const char* secret_key, char *encryped_msg)
{
    const char* p1 = msg, * p2 = secret_key;
    char* p3 = encryped_msg;
    /* 加密 */
    for (; *p1; p1++, p2++, p3++)
        *p3 = *p1 ^ *p2;
    *p3 = 0;
}
void decrypted(const char* encryped_msg, const char* secret_key, char* decryped_msg)
{
    const char* p1 = encryped_msg, * p2 = secret_key;
    char* p3 = decryped_msg;
    /* 解密(与解密操作完全一致) */
    for (; *p1; p1++, p2++, p3++)
        *p3 = *p1 ^ *p2;
    *p3 = 0;
}
int main()
{   const char* msg = "This is my student";
    const char* secret_key = "周伯通黄药师郭靖黄蓉";
    char encryped_msg[80], decryped_msg[80];

    cout << "原始信息：" << msg << endl;
    encrypt(msg, secret_key, encryped_msg);
    cout << "加密后的信息：" << encryped_msg << endl; //这个信息允许公共传播
    decrypted(encryped_msg, secret_key, decryped_msg);
    cout << "解密后的信息：" << decryped_msg << endl;

    return 0;
}
```

Microsoft Visual Studio 调试控制台

```
原始信息：This is my student
加密后的信息：偞圯砐孺啃曒蛯谳膂
解密后的信息：This is my student
```

原始信息、密钥串、加密信息，任意两个可以还原出第三个，因此要注意保护密钥串

§14. C知识补充

1. 位运算

1.2. 常用的位运算

1.2.4. 取反(~)

运算规则：0/1互反

例：char a=0x5c; 求~a

a=0101 1100

~a=1010 0011　　~a=0xa3

有符号10进制：-93

```
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
    char a=0x5c;
    cout << "a=" << hex << (int)a
         << " ~a=0x" << (~a) << " " << dec << (~a) << endl;

    return 0;
}
```

读懂运行结果!!!

```
Microsoft Visual Studio 调试控制台
a=5c ~a=0xffffffa3 -93
```

§14. C知识补充

1. 位运算

1.2. 常用的位运算

1.2.5. 左移(<<)

运算规则：左移数据，右补0

例：char a=0x12;

a=0001 0010

0010 0100 a<<1=0x24

0100 1000 a<<2=0x48

1001 0000 a<<3=0x90

0x12 = 18

0x24 = 36

0x48 = 72

0x90 = -112

无符号:144

例：int b=0x12;

a<<1=0x24

a<<2=0x48

a<<3=0x90

0x24 = 36

0x48 = 72

0x90 = 144

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    /* char型 */
    char a=0x12;
    cout << "a=0x" << hex << int(a) << " " << dec << int(a) << endl;
    cout << "a<<1=0x" << hex << (int)(char)(a<<1) << " "
         << dec << (int)(char)(a<<1) << endl;
    cout << "a<<2=0x" << hex << (int)(char)(a<<2) << " "
         << dec << (int)(char)(a<<2) << endl;
    cout << "a<<3=0x" << hex << (int)(char)(a<<3) << " "
         << dec << (int)(char)(a<<3) << endl;
    cout << endl;

    /* 直接是int型的情况 */
    int b=0x12;
    cout << "b=0x" << hex << b << " " << dec << b << endl;
    cout << "b<<1=0x" << hex << (b<<1) << " " << dec << (b<<1) << endl;
    cout << "b<<2=0x" << hex << (b<<2) << " " << dec << (b<<2) << endl;
    cout << "b<<3=0x" << hex << (b<<3) << " " << dec << (b<<3) << endl;

    return 0;
}
```

为什么是(int)(char)(a<<1)?
先 a<<1
转为 char，此时若有溢出，则会丢弃
再转 int， 以int方式输出

读懂运行结果!!!

Microsoft Visual Studio 调试控制台

```
a=0x12 18
a<<1=0x24 36
a<<2=0x48 72
a<<3=0xffffff90 -112

b=0x12 18
b<<1=0x24 36
b<<2=0x48 72
b<<3=0x90 144
```

§14. C知识补充

1. 位运算

1.2. 常用的位运算

1.2.5. 左移(<<)

运算规则：左移数据，右补0

例：char a=0x12；求a<<3

a=0001 0010

1001 0000 a<<3=0x90 有符号 -112

无符号144

★ 在不溢出(1不被舍去)的情况下，左移n位等于乘2的n次方(当做无符号数理解)

例：char a=0x12；求a<<4

a=0001 0010

1 0010 0000 a<<4=0x20 0x12=18 0x20=32

32+256(2^8)=288=18*16(2^4)

例：char a=0x9c；求a<<2

a=1001 1100

10 0111 0000 a<<2=0x70 0x9c=156 0x70=112

112+512(2^9)=624=156*4(2^2)

例：char a=0xc2；求a<<2

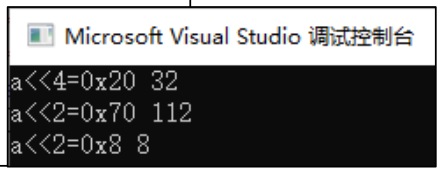
a=1100 0010

11 0000 1000 a<<2=0x8 0xc2=194 0x8=8

8+512(2^9)+256(2^8)=776=194*4(2^2)

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    char a1=0x12;
    cout << "a<<4=0x" << hex << (int)(char)(a1<<4) <<" "
         << dec << (int)(char)(a1<<4) << endl;

    char a2=0x9c;
    cout << "a<<2=0x" << hex << (int)(char)(a2<<2) <<" "
         << dec << (int)(char)(a2<<2) << endl;

    char a3=0xc2;
    cout << "a<<2=0x" << hex << (int)(char)(a3<<2) <<" "
         << dec << (int)(char)(a3<<2) << endl;
    return 0;
}
```

读懂运行结果!!!



§14. C知识补充

1. 位运算

1.2. 常用的位运算

1.2.6. 右移(>>)

运算规则：右移数据，左补0（逻辑右移）

右移数据，左补符号位（算术右移）<= C/C++的位运算时算术右移

★ 算术右移，无符号数仍补0

例：char a=0x18;

a=0001 1000

0000 1100 a>>1=0xc

0000 0110 a>>2=0x6

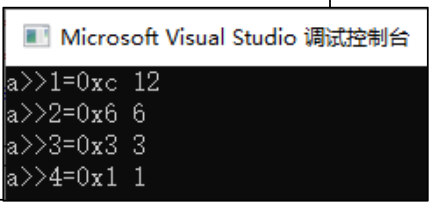
0000 0011 a>>3=0x3

0000 0001 a>>4=0x1

0x18	=	24
0xc	=	12
0x6	=	6
0x3	=	3
0x1	=	1

```
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
    char a=0x18;
    cout << "a>>1=0x" << hex << (int)(a>>1)
         << " " << dec << (int)(a>>1) << endl;
    cout << "a>>2=0x" << hex << (int)(a>>2)
         << " " << dec << (int)(a>>2) << endl;
    cout << "a>>3=0x" << hex << (int)(a>>3)
         << " " << dec << (int)(a>>3) << endl;
    cout << "a>>4=0x" << hex << (int)(a>>4)
         << " " << dec << (int)(a>>4) << endl;

    return 0;
}
```

读懂运行结果!!!



§14. C知识补充

1. 位运算

1.2. 常用的位运算

1.2.6. 右移(>>)

运算规则：右移数据，左补0（逻辑右移）

右移数据，左补符号位（算术右移）<= C/C++的位运算时算术右移

★ 算术右移，无符号数仍补0

★ 在不溢出(1不被舍去)的情况下，右移n位等于除2的n次方

(当作有符号数理解)

例：char a=0x84；求a>>1

a=1000 0100

1100 0010 a>>1=0xc2

最高位为1，若作为符号位，则表示负数

0x84 = -124
无符号：132
0xc2 = -62
无符号：194

a=1000 0100
-) 1
1000 0011
0111 1100

补码 => 原码
(1)减1
(2)取反
(3)绝对值
|a|=124
|a>>1|=62

a=1100 0010
-) 1
1100 0001
0011 1110

```
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
    char a=0x84; //有符号数补1 !!!
    cout << "a=0x" << hex << int(a) << " " << dec << int(a) << endl;
    cout << "a>>1=0x" << hex << (int)(a>>1) << " " << dec << (int)(a>>1) << endl;
    cout << endl;

    unsigned char b=0x84; //无符号数补0 !!!
    cout << "b=0x" << hex << int(b) << " " << dec << int(b) << endl;
    cout << "b>>1=0x" << hex << (int)(b>>1)<<" "   << dec << (int)(b>>1) << endl;

    return 0;
}
```

读懂运行结果!!!

```
Microsoft Visual Studio 调试控制台
a=0xffffff84 -124
a>>1=0xffffffc2 -62

b=0x84 132
b>>1=0x42 66
```

§14. C知识补充

1. 位运算

1.2. 常用的位运算

1.2.6. 右移(>>)

例：char a=0x18;

```
a=0001 1000              (24)
  0000 1100  a>>1=0xc    (12)
  0000 0110  a>>2=0x6    (6)
  0000 0011  a>>3=0x3    (3)
  0000 0001  a>>4=0x1    (1) 溢出舍去了1
  0000 0000  a>>5=0x0    (0) 再次溢出舍去1
  0000 0000  a>>6=0x0    (0) >>6以上都是0
```

例：char a=0x84;

```
a=1000 0100              (-124)
  1100 0010  a>>1=0xc2   (-62)
  1110 0001  a>>2=0xe1   (-31)
  1111 0000  a>>3=0xf0   (-16) 溢出舍去了1
  1111 1000  a>>4=0xf8   (-8)
  1111 1100  a>>5=0xfc   (-4)
  1111 1110  a>>6=0xfe   (-2)
  1111 1111  a>>7=0xff   (-1)
  1111 1111  a>>8=0xff   (-1) >>8以上都是-1
```

读懂运行结果!!!

```
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
    char a=0x18;
    int i;

    for(i=1; i<=6; i++) {
        a = a>>1;
        cout << "a>>" << i << "=0x" << hex << int(a) << " " << dec << int(a) << endl;
        }
    return 0;
}
```

Microsoft Visual Studio 调试控制台

```
a>>1=0xc 12
a>>2=0x6 6
a>>3=0x3 3
a>>4=0x1 1
a>>5=0x0 0
a>>6=0x0 0
```

读懂运行结果!!!

```
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
    char a=0x84;
    int i;

    for(i=1; i<=8; i++) {
        a = a>>1;
        cout << "a>>" << i << "=0x" << hex << int(a) << " " << dec << int(a) << endl;
        }
    return 0;
}
```

Microsoft Visual Studio 调试控制台

```
a>>1=0xffffffc2 -62
a>>2=0xfffffffe1 -31
a>>3=0xfffffff0 -16
a>>4=0xfffffff8 -8
a>>5=0xfffffffc -4
a>>6=0xfffffffe -2
a>>7=0xffffffff -1
a>>8=0xffffffff -1
```

§14. C知识补充

1. 位运算

1.3. 复合位运算符

&=　|=　^=　<<=　>>=

★ 将上例中 a = a>>1;

改为 a >>= 1; 结果相同

```
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
    char a=0x18;
    int i;

    for(i=1; i<=6; i++) {
        a >>= 1;
        cout << "a>>" << i << "=0x" << hex << int(a) << " " << dec << int(a) << endl;
        }
    return 0;
}
```

读懂运行结果!!!

```
Microsoft Visual Studio 调试控制台
a>>1=0xc 12
a>>2=0x6 6
a>>3=0x3 3
a>>4=0x1 1
a>>5=0x0 0
a>>6=0x0 0
```

```
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
    char a=0x84;
    int i;

    for(i=1; i<=8; i++) {
        a >>= 1;
        cout << "a>>" << i << "=0x" << hex << int(a) << " " << dec << int(a) << endl;
        }
    return 0;
}
```

读懂运行结果!!!

```
Microsoft Visual Studio 调试控制台
a>>1=0xffffffc2 -62
a>>2=0xffffffe1 -31
a>>3=0xfffffff0 -16
a>>4=0xfffffff8 -8
a>>5=0xfffffffc -4
a>>6=0xfffffffe -2
a>>7=0xffffffff -1
a>>8=0xffffffff -1
```

§14.C知识补充

2.带参数的main函数

2.1.引入

可执行文件运行时，目前只能简单的运行，如果能加上参数，则使用中可以更灵活

例1：两数交换(常规方法)

```
#include <iostream>
using namespace std;

int main()
{
    int a, b, t;
    cout << "请输入两个整数" << endl;
    cin >> a >> b;
    cout << "交换前：a=" << a << "  b=" << b << endl;
    t = a;
    a = b;
    b = t;
    cout << "交换后：a=" << a << "  b=" << b << endl;

    return 0;
}
```

§14.C知识补充

2.带参数的main函数

2.1.引入

可执行文件运行时，目前只能简单的运行，如果能加上参数，则使用中可以更灵活

2.2.带参数的main函数的定义形式

```
int main(int argc, char **argv)
int main(int argc, char *argv[])
```

两者均可

★ 参数解释

argc：参数的个数，若不带参数，则为1(自身)

argv：参数的内容，用指针数组表示，每个元素是一个字符串(char *)，最后一个是 NULL

- argv数组共有argc+1个元素，下标[0]~[argc]（例如：argc为3，则argv[0]是自身，argv[3]是NULL）
- 参数名argc/argv可变，类型不能变（例如：int ac, char **av）
- VS系列可以 long ac,unsigned char **av, gcc系列不可以，因此不建议其它类型

§14.C知识补充

2.带参数的main函数

2.3.使用

例2：两数交换(main函数带参数方法)

```
#include <iostream>
#include <cstdlib>  //atoi函数用到
using namespace std;
int main(int argc, char *argv[])
{
    int a, b, t;

    cout << "argc="   << argc    << endl;
    cout << "argv[0]="<< argv[0] << endl;
    a = atoi(argv[1]); //atoi是将字符串转为整数的函数
    b = atoi(argv[2]);

    cout << "交换前:a=" << a << "  b=" << b << endl;
    t = a;
    a = b;
    b = t;
    cout << "交换后:a=" << a << "  b=" << b << endl;

    return 0;
}
```

假设编译后形成形成demo.exe

1、集成环境运行　　(出错,为什么？)

2、命令行运行

demo	(出错,为什么？)
demo 10	(出错,为什么？)
demo 10 15	(正确)
demo 10 15 20	(正确)

§14.C知识补充

2.带参数的main函数

2.3.使用

例3：两数交换(main函数带参数方法 - 改进)

```
#include <iostream>
#include <cstdlib>  //atoi函数用到
using namespace std;
int main(int argc, char *argv[])
{
    int a, b, t;
    if (argc<3) { /* 参数不足3个则出现提示 */
        cout << "请带两个整数作为参数"<< endl;
        return -1;
        }
    for (t=0; t<argc; t++)    /* 打印所有的参数值 */
         cout << "argv[" << t << "]=" << argv[t] << endl;
    a = atoi(argv[1]); //atoi是将字符串转为整数的函数
    b = atoi(argv[2]);
    cout << "交换前：a=" << a << "  b=" << b << endl;
    t = a;
    a = b;
    b = t;
    cout << "交换后：a=" << a << "  b=" << b << endl;
    return 0;
}
```

假设编译后形成形成
demo.exe
1、集成环境运行
2、命令行运行
    demo
    demo 10
    demo 10 15
    demo 10 15 20

§14.C知识补充

2.带参数的main函数

2.4.综合应用

例4：作业相似度检查程序的参数设计

(1) 学生的匹配

要求能在两个特定的学生之间检查

某个特定学生和全体学生之间检查

全体学生之间相互检查

(2) 文件的匹配

要求既可以是单文件，也可以全部文件

(3) 相似度设置

要求值在60-100间浮动

(4) 输出方式

可选文件/屏幕

假设Linux下编译后形成形成 check，下列方式都正确

```
./check 2159999 2159998 12-b2.cpp 80
./check 2159999 2159998 all       80
./check 2159999 all     12-b2.cpp 75 result.txt
./check 2159999 all     all       85
./check all     all     all       85 final.txt
```

★ 具体通过作业方式来理解实现过程

§14.C知识补充

2.带参数的main函数

2.5.参数个数不固定的带参main函数

例5：在Windows的命令行下输入 ping，可以看到ping 命令的很多选项，下列命令都是正确的

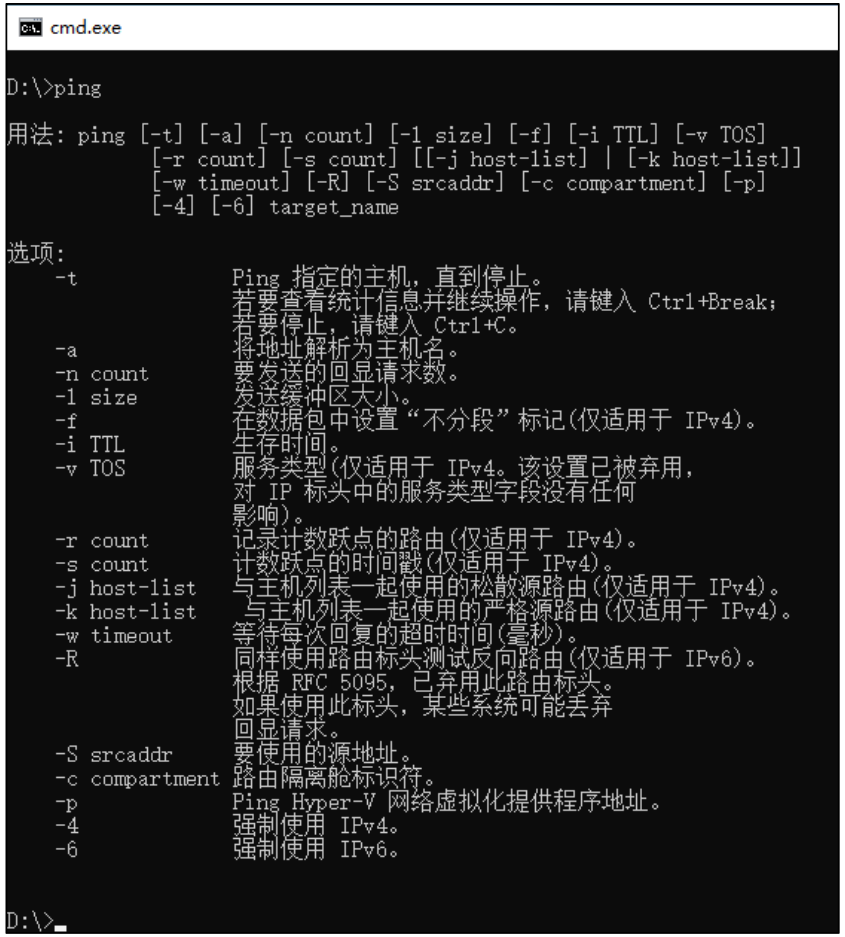
```
ping 10.10.108.117
ping -t 10.10.108.117
ping -n 10 10.10.108.117
ping -n 10 -l 50000 192.168.80.230
ping -t -l 50000 192.168.80.230
ping -l 50000 -t 192.168.80.230
```

★ 参数个数不固定，且部分参数要2个一组

★ 参数出现顺序任意

★ 具体通过作业方式来理解实现过程

思考：如果输入ping后用人机交互形式，该如何做？从用户操作方便性角度而言，可行吗？



§14. C知识补充

2. 带参数的main函数

2.6. 带参数的main函数的扩展形式(仅了解)

形式： int main(int argc, char **argv, char **env)

或： char *env[]

参数解释：
- argc：同前
- argv：同前
- env ：操作系统的环境变量，用指针数组来表示，每个元素是一个字符串(char *)，最后一个元素是NULL

使用：需要判断/取操作系统的某些设置时才用到

例6：取操作系统的环境变量(在Windows/Linux下分别运行)

```
#include <iostream>
using namespace std;

int main(int argc, char **argv, char **env)
{
    int i;
    for (i=0; env[i]; i++)
        cout<< "env[" << i << "]=" << env[i] <<endl;

    return 0;
}
```

拓展问题：如何在Windows/Linux下增加一个环境变量？

§14.C知识补充

2.带参数的main函数

2.7.带参数main函数的作用

	带参main函数方式	运行时键盘交互方式
运行方法	运行命令后直接跟各参数，不再进行人机交互	运行命令后进入人机交互
是否需要人机交互	不需要人机交互	需要人机交互 (可用输入重定向方式取消人机交互，但不方便)
适用程序	1、守护进程(开机自启动) 2、后台运行程序 3、类似ping的不定参数形式命令，必须用此形式	前端程序

§14. C知识补充

3. 共用体

例： 定义一个用于一卡通管理系统的结构，要求包含卡号、余额、消费限额、消费密码等公共信息，
此外，若持卡人是学生，要包含学号、姓名、专业等学生特有的信息，
若持卡人是教师，则包含工号、姓名、职称等教师特有的信息

```
struct student {
    定义学生信息;
};
struct teacher {
    定义教师信息;
};
struct ykt {
    公共信息;
    student sinfo;
    teacher tinto;
};
int main()
{
    ykt y1;//定义变量
}
```

```
对y1的成员的访问：

int main()
{
    ykt y1;
    ...;
    y1.卡号
    y1.sinfo.学号
    y1.tinfo.工号
    ...;
    return 0;
}
```

缺陷：无论持卡人何种身份sinfo和tinfo中必然有一个是不需要填写任何信息的，从而导致存储空间的浪费

解决：能否使sinfo/tinfo共用一段空间，当持卡人是学生时，这段空间按student方式访问，
当持卡人是教师时按teacher方式访问

=>(共用体)

§14. C知识补充

3. 共用体

```
union 共用体名 {
    共用体成员1（类型名  成员名）
    ...
    共用体成员n（类型名  成员名）
};
```

```
union data {
    short a;
    long  b;
    char  c;
};
```

★ 所有成员从同一内存开始，共用体的大小为其中占用空间最大的成员的大小

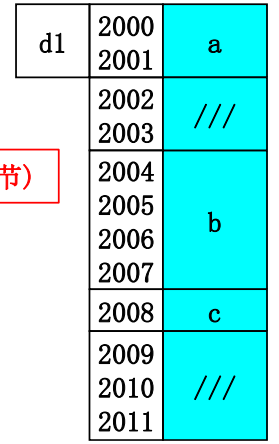
★ 给一个共用体成员赋值后，会覆盖其它成员的值，因此只有最后一次存放的成员是有效的

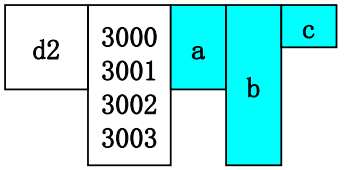
★ 其它所有定义、使用方法同结构体

```
#include <iostream>
using namespace std;

struct data1 {
    short a;
    long  b;
    char  c;
};

union data2 {
    short a;
    long  b;
    char  c;
};

int main()
{
    data1 d1;
    data2 d2;

    cout << sizeof(d1) << ' ' << sizeof(d2) << endl;
    cout << &d1.a << ' ' << &d1.b << ' ' << (void *)&d1.c << endl;
    cout << &d2.a << ' ' << &d2.b << ' ' << (void *)&d2.c << endl;

    return 0;
}
```

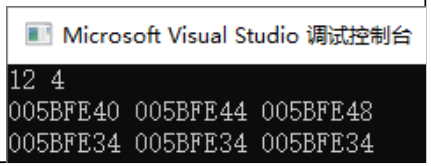
12:所有成员所占空间之和(含填充字节)

4 :所有成员中最大成员所占空间





12 4
地址：X X+4 X+8
地址：Y Y Y



§14. C知识补充

3. 共用体

★ 所有成员从同一内存开始，共用体的大小为其中占用空间最大的成员的大小

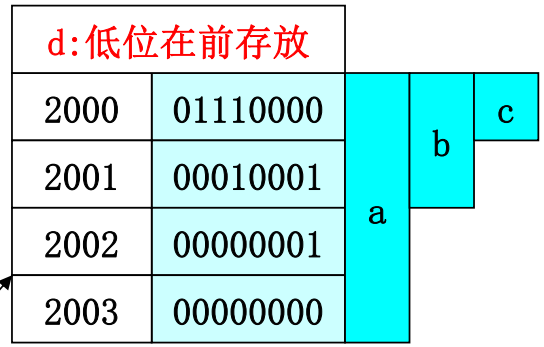
```
#include <iostream>
using namespace std;

union data {
   int   a;
   short b;
   char  c;
};

int main()
{
   union data d;
   d.a=70000;
   cout << d.a << ' ' << d.b << ' ' << d.c << endl;
   d.b=7000;
   cout << d.a << ' ' << d.b << ' ' << d.c << endl;
   d.c='A';
   cout << d.a << ' ' << d.b << ' ' << d.c << endl;
   return 0;
}
```

70000 4464 p

72536 7000 X

72513 6977 A

70000 = 00000000 00000001 00010001 01110000



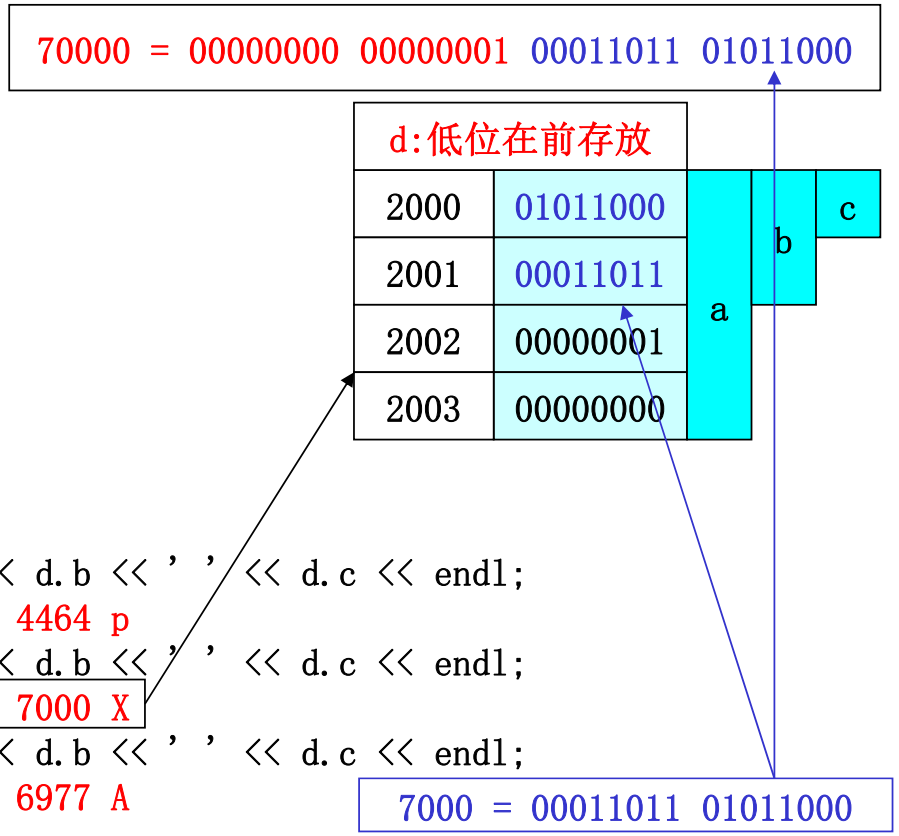
§14. C知识补充

3. 共用体

★ 所有成员从同一内存开始，共用体的大小为其中占用空间最大的成员的大小

```
#include <iostream>
using namespace std;

union data {
   int   a;
   short b;
   char  c;
};

int main()
{
   union data d;
   d.a=70000;
   cout << d.a << ' ' << d.b << ' ' << d.c << endl;
   d.b=7000;
   cout << d.a << ' ' << d.b << ' ' << d.c << endl;
   d.c='A';
   cout << d.a << ' ' << d.b << ' ' << d.c << endl;
   return 0;
}
```

```
70000 4464 p
72536 7000 X
72513 6977 A
```



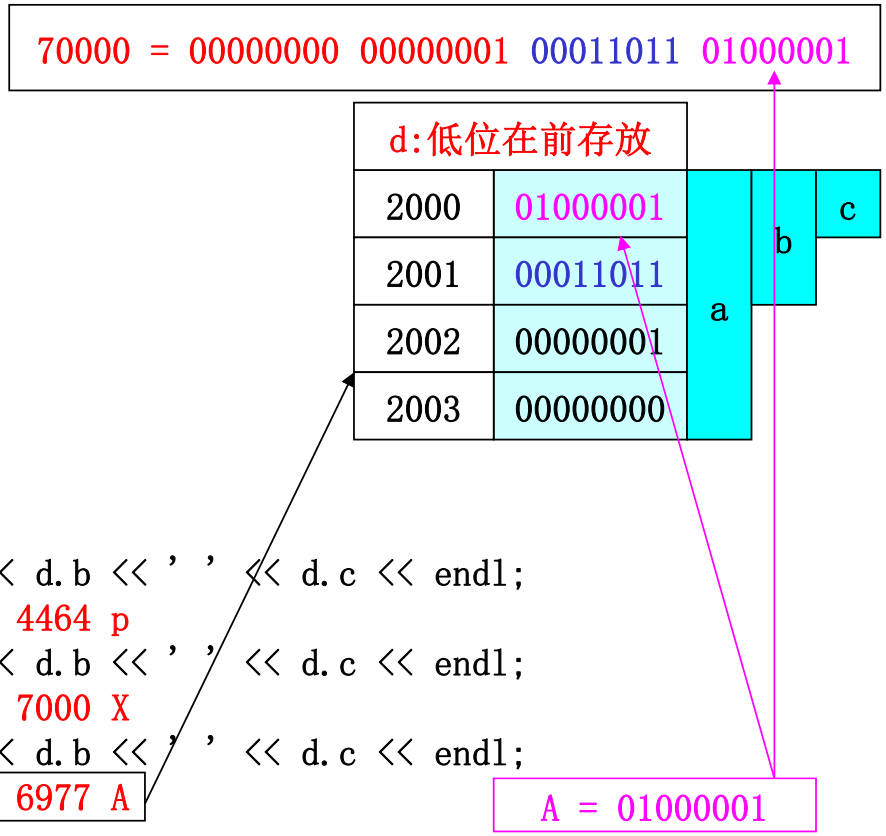
§14. C知识补充

3. 共用体

★ 所有成员从同一内存开始，共用体的大小为其中占用空间最大的成员的大小

```
#include <iostream>
using namespace std;

union data {
   int   a;
   short b;
   char  c;
};

int main()
{
   union data d;
   d.a=70000;
   cout << d.a << ' ' << d.b << ' ' << d.c << endl;
   d.b=7000;
   cout << d.a << ' ' << d.b << ' ' << d.c << endl;
   d.c='A';
   cout << d.a << ' ' << d.b << ' ' << d.c << endl;
   return 0;
}
```

```
70000 4464 p
72536 7000 X
72513 6977 A
```



§14. C知识补充

3. 共用体

★ 所有成员从同一内存开始，共用体的大小为其中占用空间最大的成员的大小

```
#include <iostream>
using namespace std;

union data {
   int   a;
   short b;
   char  c;
};

int main()
{
   union data d;
   d.c='A';
   cout << d.a << ' ' << d.b << ' ' << d.c << endl;
   d.b=7000;
   cout << d.a << ' ' << d.b << ' ' << d.c << endl;
   d.a=70000;
   cout << d.a << ' ' << d.b << ' ' << d.c << endl;
   return 0;
}
```

不确定 不确定 A

不确定 7000 X

70000 4464 p

d:低位在前存放	
2000	01000001
2001	???
2002	???
2003	???

d:低位在前存放	
2000	01011000
2001	00011011
2002	???
2003	???

d:低位在前存放	
2000	01110000
2001	00010001
2002	00000001
2003	00000000

§14. C知识补充

3. 共用体

★ 所有成员从同一内存开始，共用体的大小为其中占用空间最大的成员的大小

```
#include <iostream>
using namespace std;

union data {
    int   a;
    short b;
    char  c;
};

int main()
{
    union data d;
    d.a=70000;
    cout << d.a << ' ' << d.b << ' ' << d.c << endl;
    d.b=7000;
    cout << d.a << ' ' << d.b << ' ' << d.c << endl;
    d.c='A';
    cout << d.a << ' ' << d.b << ' ' << d.c << endl;

    return 0;
}
```

70000 = 00000000 00000001 00010001 01110000

d:低位在前存放

2000	01110000
2001	00010001
2002	00000001
2003	00000000

a, b, c

d:高位在前存放

2000	00000000
2001	00000001
2002	00010001
2003	01110000

a, b, c

不同的CPU(和操作系统、编译器无关)有不同的字节序类型，这些字节序是指整数在内存中保存的顺序，称为主机序，常见的有两种：

★ Little endian：将低序字节存储在起始地址，地址低位存储值的低位，地址高位存储值的高位(小头序/小字序)

★ Big endian ：将高序字节存储在起始地址，地址低位存储值的高位，地址高位存储值的低位(大头序/大字序)

思考：大字序系统中，本题的运行结果？

(注：本题无法通过Intel/AMD等小字序系统运行测试程序得到答案，需要手动计算)

§14. C知识补充

3. 共用体

例： 定义一个用于一卡通管理系统的结构，要求包含卡号、余额、消费限额、消费密码等公共信息，
此外，若持卡人是学生，要包含学号、姓名、专业等学生特有的信息，
若持卡人是教师，则包含工号、姓名、职称等教师特有的信息

```
struct student {
    定义学生信息;
};

struct teacher {
    定义教师信息;
};

struct ykt {
    公共信息;
    student sinfo;
    teacher tinto;
};

int main()
{
    ykt y1;//定义变量
}
```

空间浪费

```
struct student {
    定义学生信息;
};
struct teacher {
    定义教师信息;
};
union owner {
    student s;
    teacher t;
};
struct ykt {
    公共信息;
    char type; //持卡人类别
    owner info;
};
int main()
{
    ykt y1;//定义变量
}
```

此处保证s/t共用一段空间

```
int main()
{
    ykt y1;//定义变量
    ...;
    y1.卡号;
    if (y1.type=='s'){
        y1.info.s.学号;
        }
    else {
        y1.info.t.工号;
        }
    ...;

    return 0;
}
```

§14. C知识补充

4. 条件编译

4.1. 问题的提出

例1：上学期作业

不同编译器函数不同

§.基础知识题 - 字符数组的输入与输出

4. 多个字符串的输入

★ 从键盘输入含空格字符串的方法(不同编译器不同)

- VS : 有gets_s，无gets， 有fgets
- Dev C++ : 有gets， 无gets_s，有fgets
- Linux : 无gets， 无gets_s，有fgets
- fgets函数的原型定义为：

  fgets(字符数组名，最大长度，stdin);

  但与gets/gets_s的表现有不同，请自行观察

★ scanf/cin通过某些高级设置方式还是可以输入含空格的字符串的，本课程不再讨论

例2：斗地主发牌

不同编译器下的花色打印

	编译器VS	编译器Dev	编译器Linux
位运算模拟多开关	Y	Y	Y
斗地主发牌	Y	Y	Y

问题：如何让一段存在差异的代码在多编译器下均通过？

§14. C知识补充

4. 条件编译

4.1. 问题的提出

例3：

输入成绩，根据分数
打印及格或不及格

```
if (...) {
    ...
    }
else {
    ...
    }
```

某个游戏软件，适应不同
大小的屏幕

```
if (屏幕是1920*1080) {
    ...
    }
else {
    ...
    }
```

输入成绩，根据分数
打印分数等级

```
if (...) {
    ...
    }
else if (...) {
    ...
    }
else {
   ...
   }
```

可能else if
会重复多次

某个游戏软件，适应不同
大小的屏幕

```
if (屏幕是1920*1080) {
    ...
    }
else if (屏幕是1280*720){
    ...
    }
else {
    ...
    }
```

可能else if
会重复多次

左右的区别：

输入成绩，根据分数打印分数等级 对于if-else形式的双分支(包括if-elseif-else形式的多分支)， 每次执行时只能选择其中的某一个分支执行但多次执行时可能每个分支都可能会被执行到， 因此每个分支都是有意义的	某个游戏软件，适应不同大小的屏幕 对于if-else形式的双分支(包括if-elseif-else形式的多分支)， 对于某种型号的设备，多次执行的都是其中的一个固定分支， 因此分支中的其他部分对某种型号设备的可执行文件而言是无意义的

因为多种型号设备的公用代码很多(非显示部分)，因此希望维护一套源程序代码

既希望维护一套源代码，
又希望可执行文件中
仅包含有效部分(节约空间)

§14.C知识补充

4.条件编译

4.2.问题的引入

引入 ：某些程序在调试、兼容性、平台移植等情况下希望通过简单地设置参数就能生成不同的软件

方法1：把所有可能用到的代码都写进源程序文件中，再全部编译到可执行文件中，执行时根据相应的条件选择不同的代码执行
（分支语句方式，源程序统一，可执行代码大）

方法2：把所有可能用到的代码都写进源程序文件中，在编译之前根据需要选择待编译的代码段，再进行编译，可执行文件中只包含需要的代码段
（条件编译方式，源程序统一，可执行代码小）

§14. C知识补充

4. 条件编译

4.3. 条件编译的三种形式

#ifdef 标识符 　程序段1 #else 　程序段2 #endif	#ifndef 标识符 　程序段1 #else 　程序段2 #endif	#if 表达式 　程序段1 #else 　程序段2 #endif
若定义了标识符 则编译程序段1 否则编译程序段2	若未定义标识符 则编译程序段1 否则编译程序段2	若表达式为真 则编译程序段1 否则编译程序段2

★ 该过程在编译阶段完成，最终形成的可执行文件中只包含两个程序段中的某一个

- # xxx 是C/C++约定的编译预处理指令(例：#define/#include/...)

★ 预编译指令中的表达式与C语言本身的表达式基本一致，逻辑运算、算术运算等均可用于预编译指令

★ 修改预编译条件后，每次都要重新编译链接才能生成新的可执行文件

- 修改条件编译的方法一般是手动修改，再次编译
- 更进一步的方法：1、将条件编译开关放在设置或makefile文件中，通过修改makefile来编译
  2、利用标识不同编译器的预置宏定义来区分不同编译器

★ 形式如if-elseif的多分支条件编译请自行学习

★ 不可能通过执行程序时输入某值或根据程序执行时是否满足某条件的方式来选择，因为是"编译预处理"

程序如下，假设两次的输入为7，-3，写出程序的运行结果

```
#define PROGRAM
main()
{  int t;
   cin >> t;
#ifdef PROGRAM
   if (t>=0)
      cout << "t是非负整数" << endl;
#else
   if (t<0)
      cout << "t是负整数" << endl;
#endif
   cout << "End." << endl;
   return 0;
}
```

注：#ifdef 只判断是否定义，不判断定义值的T/F

未被编译进可执行文件中

输入为7时：
t是非负整数
End

输入为-3时：
End

程序如下，假设两次的输入为7，-3，写出程序的运行结果

```
#define PROGRAM
main()
{  int t;
   cin >> t;
#ifndef PROGRAM
   if (t>=0)
      cout << "t是非负整数" << endl;
#else
   if (t<0)
      cout << "t是负整数" << endl;
#endif
   cout << "End." << endl;
   return 0;
}
```

注：#ifndef 只判断是否定义，不判断定义值的T/F

未被编译进可执行文件中

输入为7时：
End

输入为-3时：
t是负整数
End

程序如下，假设两次的输入为7，-3，写出程序的运行结果

```
#define PROGRAM 0
main()
{  int t;
   cin >> t;
#if PROGRAM
   if (t>=0)
      cout << "t是非负整数" << endl;
#else
   if (t<0)
      cout << "t是负整数" << endl;
#endif
   cout << "End." << endl;
   return 0;
}
```

注：#if 要判断定义值的T/F，未定义则按F处理

未被编译进可执行文件中

输入为7时：
End

输入为-3时：
t是负整数
End

程序如下，假设两次的输入为7，-3，写出程序的运行结果

```
#define PROGRAM 1
main()
{  int t;
   cin >> t;
#if PROGRAM
   if (t>=0)
      cout << "t是非负整数" << endl;
#else
   if (t<0)
      cout << "t是负整数" << endl;
#endif
   cout << "End." << endl;
   return 0;
}
```

注：#if 要判断定义值的T/F，未定义则按F处理

未被编译进可执行文件中

输入为7时：
t是非负整数
End

输入为-3时：
End

§14.C知识补充

4.条件编译

最初的问题：如何让下面程序在多编译器中均能通过

提示：去找各编译器的预置标识

```
int main()
{
    char a[10];
    #if ***   //如果是Dev
       gets(a);
    #elif *** //如果是VS
       gets_s(a);
    #elif *** //如果是Linux
       fgets(a, 10, stdin);
    #endif
    cout << a << endl;
    return 0;
}
```

```
#if ***   //如果是VS
   gets_s(a);
#elif *** //如果是Linux
   fgets(a, 10, stdin);
#elif *** //如果是Dev
   gets_s(a);
#endif
```

§14. C++知识补充

5. 类的共用数据的保护

5.1. 引入及含义

一个数据可以通过不同的方式进行共享访问，因此可能导致数据因为误操作而改变，为了达到既能共享，又不会因误操作而改变，引入共用数据保护的概念

```
void fun(int *p)
{   *p=10;
}
int main()
{   int k=15;
    fun(&k);
}
```

```
void fun(int &p)
{   p=10;
}
int main()
{   int k=15;
    fun(k);
}
```

通过指针/引用，在fun中改变了main的局部变量k的值

§2. 基础知识

2.7. 变量

2.7.6. 常变量

含义：在程序执行过程中值不能改变的变量

形式：

```
const 数据类型 变量名=初值;
数据类型 const 变量名=初值;
const int a=10;
const double pi=3.14159;
```

★ 常变量必须在定义时赋初值，且在执行过程中不能再次赋值，否则编译错误

§12. 指针进阶

9. const指针

9.1. 共用数据的保护

9.2. 指向常量的指针变量

形式：const 数据类型 *指针变量名;

数据类型 const *指针变量名;

9.3. 常指针

形式：数据类型 *const 指针变量名;

9.4. 指向常量的常指针(同时满足9.2+9.3)

形式：const 数据类型 *const 指针变量名;

	能否指向不同变量	能否通过指针变量修改变量值	是否需要定义时初始化
指向常量的指针变量	✓	×	×
常指针	×	✓	✓
指向常量的常指针	×	×	✓

§14. C++知识补充

5. 类的共用数据的保护

5.2. 常对象

常对象：

const 类名 对象名(初始化实参表)

或 类名 const 对象名(初始化实参表)

```
const Time t1(15);
Time const t2(16, 30, 0);
```

```
T1始终是15:00:00
T2始终是16:30:00
```

★ 与常变量相同，在整个程序的执行过程中值不可再变化

★ 与常变量相同，必须在定义时进行初始化

★ 不能调用普通成员函数(即使不改变数据成员的值)

```cpp
#include <iostream>
using namespace std;
class Time {
public:
    int hour, minute, sec;
    Time(int h = 0, int m = 0, int s = 0)
    {  hour = h; minute = m; sec = s;  }
    void display()
    {  cout << hour << minute << sec;  }
};
int main()
{
    const Time t1(15);
    Time const t2(16, 30, 0);
    t1.minute = 12; //编译报错
    t2.sec = 27;    //编译报错
    t1.display();   //编译报错(虽然不改)
}
```

```
demo.cpp(15,7): error C3892: “t1”: 不能给常量赋值
demo.cpp(16,7): error C3892: “t2”: 不能给常量赋值
demo.cpp(17,16): error C2662: “void Time::display(void)”: 不能将“this”指针从“const Time”转换为“Time &”
demo.cpp(17,5): message : 转换丢失限定符
demo.cpp(8,10): message : 参见“Time::display”的声明
```

§14. C++知识补充

5. 类的共用数据的保护

5.3. 常对象成员

常对象成员：

常对象中的所有数据成员在程序执行过程中值均不可变，如果只需要限制部分成员的值在执行过程中不可变，则需要引入常对象成员的概念

- 常数据成员：该数据成员的值在执行中不可变
- 常成员函数：该函数只能引用成员的值，不能修改

常数据成员：

```
class 类名 {
    const 数据类型 数据成员名
 或 数据类型 const 数据成员名
};

class Time {
    private:
        const int hour;
        int const minute;
        int sec;
};
```

常成员函数：

```
class 类名 {
    返回类型 成员函数名(形参表) const;
};


class Time {
    public:
    void display() const;
};
```

§14. C++知识补充

5. 类的共用数据的保护

5.3. 常对象成员

使用：

★ 常数据成员要在构造函数中初始化，使用中值不变，在构造函数中进行初始化时，必须用参数初始化表

```
#define <iostream>
using namespace std;

class Time {
  public:
    const int hour;
    int minute, sec;
    Time(int h=0, int m=0, int s=0)
    {
        hour   = h;
        minute = m;
        sec    = s;
    }
};
int main()
{
   Time t1;
   t1.hour = 10; //错误
}
```

错误

```
demo.cpp(7,9): error C2789: “Time::hour”: 必须初始化常量限定类型的对象
demo.cpp(6): message : 参见“Time::hour”的声明
demo.cpp(7,9): error C2789: “Time::minute”: 必须初始化常量限定类型的对象
demo.cpp(6): message : 参见“Time::minute”的声明
demo.cpp(7,9): error C2789: “Time::sec”: 必须初始化常量限定类型的对象
demo.cpp(6): message : 参见“Time::sec”的声明
demo.cpp(9,9): error C2166: 左值指定 const 对象
demo.cpp(10,9): error C2166: 左值指定 const 对象
demo.cpp(11,9): error C2166: 左值指定 const 对象
demo.cpp(17,7): error C3892: “t1”: 不能给常量赋值
```

```
#define <iostream>
using namespace std;

class Time {
  public:
    const int hour;
    int minute, sec;
    Time(int h=0, int m=0, int s=0) : hour(h)
    {
        minute = m;
        sec    = s;
    }
};
int main()
{
   Time t1;
   t1.hour = 10; //错误
}
```

正确

```
demo.cpp(17,7): error C3892: “t1”: 不能给常量赋值
```

§14. C++知识补充

5. 类的共用数据的保护

5.3. 常对象成员

使用：

★ 常数据成员要在构造函数中初始化，使用中值不变，在构造函数中进行初始化时，必须用参数初始化表

★ 常成员函数只能引用类的数据成员(无论是否常数据成员)的值，而不能修改数据成员的值

普通成员函数

```
#include <iostream>
using namespace std;

class Time {
  public:
    int hour, minute, sec;
    void set(int h=0,int m=0,int s=0)
    {  hour   = h;
       minute = m;
       sec    = s;
    }
};
int main()
{
   Time t1;
   t1.set(14,15,23);
}
```

正确

常成员函数

```
#include <iostream>
using namespace std;

class Time {
  public:
    int hour, minute, sec;
    void set(int h=0,int m=0,int s=0) const
    {  hour   = h;
       minute = m;
       sec    = s;
    }
};
int main()
{
   Time t1;
   t1.set(14,15,23);
}
```

错误

```
demo.cpp(8,9): error C3490: 由于正在通过常里对象访问“hour”，因此无法对其进行修改
demo.cpp(9,9): error C3490: 由于正在通过常里对象访问“minute”，因此无法对其进行修改
demo.cpp(10,9): error C3490: 由于正在通过常里对象访问“sec”，因此无法对其进行修改
```

§14. C++知识补充

5. 类的共用数据的保护

5.3. 常对象成员

使用：

★ 常成员函数写成下面形式，编译不报错但不起作用

const 返回类型 成员函数名(形参表)

或 返回类型 const 成员函数名(形参表)

```
#include <iostream>
using namespace std;

class Time {
  public:
    int hour, minute, sec;
    const void set(int h, int m,int s)
    {   hour   = h;
        minute = m;
        sec    = s;
    }
};

int main()
{  Time t1;
   t1.set(14,15,23);
}
```

赋值正确，说明set不是常成员函数

```
#include <iostream>
using namespace std;

class Time {
  public:
    int hour, minute, sec;
    void const set(int h, int m, int s)
    {   hour   = h;
        minute = m;
        sec    = s;
    }
};

int main()
{  Time t1;
   t1.set(14,15,23);
}
```

赋值正确，说明set不是常成员函数

§14. C++知识补充

5. 类的共用数据的保护

5.3. 常对象成员

使用：

★ 常成员函数可以调用本类的另一个常成员函数，但不能调用本类的非常成员函数(即使该非常成员函数不修改数据成员的值)

```
#include <iostream>
using namespace std;

class Time {
  public:
    int hour, minute, sec;
    void display()
    {   cout << hour << endl;
    }
    void fun() const
    {   display();
    }
};
int main()
{
   Time t1;
   t1.fun();
}
```

错误

```
demo.cpp(9,1): error C2662: “void Time::display(void)”: 不能将“this”指针从“const Time”转换为“Time &”
demo.cpp(9,9): message : 转换丢失限定符
demo.cpp(6,10): message : 参见“Time::display”的声明
```

```
#include <iostream>
using namespace std;

class Time {
  public:
    int hour, minute, sec;
    void display() const
    {   cout << hour << endl;
    }
    void fun() const
    {   display();
    }
};
int main()
{
   Time t1;
   t1.fun();
}
```

正确

§14. C++知识补充

5. 类的共用数据的保护

5.3. 常对象成员

使用：

★ 若希望常成员函数能强制修改数据成员，则要将数据成员定义为mutable

(适用场景:一个复杂大类，希望绝大多数成员函数不修改该值)

```
#include <iostream>
using namespace std;

class Time {
    public:
    int hour, minute, sec;
    void set(int h=0, int m=0, int s=0) const
    {   hour   = h;
        minute = m;
        sec    = s;
    }
};
int main()
{  Time t1;
   t1.set(14,15,23);
}
```

错误

```
demo.cpp(8,9): error C3490: 由于正在通过常里对象访问“hour”，因此无法对其进行修改
demo.cpp(9,9): error C3490: 由于正在通过常里对象访问“minute”，因此无法对其进行修改
demo.cpp(10,9): error C3490: 由于正在通过常里对象访问“sec”，因此无法对其进行修改
```

```
#include <iostream>
using namespace std;
class Time {
    public:
        mutable int hour, minute, sec;
        void set(int h=0, int m=0, int s=0) const
        {   hour   = h;
            minute = m;
            sec    = s;
        }
};
int main()
{  Time t1;
   t1.set(14,15,23);
}
```

正确

```
#include <iostream>
using namespace std;
class Time {
    public:
        int mutable hour, minute, sec;
        void set(int h=0, int m=0, int s=0) const
        {   hour   = h;
            minute = m;
            sec    = s;
        }
};
int main()
{  Time t1;
   t1.set(14,15,23);
}
```

正确

§14. C++知识补充

5. 类的共用数据的保护

5.3. 常对象成员

使用：

★ 若定义对象为常对象，则只能调用其中的常成员函数(不能修改数据成员的值)，而不能调用其中的普通成员函数(即使该成员不修改数据成员的值)

本节开始的例子

```
#include <iostream>
using namespace std;

class Time {
  public:
    int hour, minute, sec;
    Time(int h=0, int m=0, int s=0) {  hour = h; minute = m; sec = s; }
    void display() {  cout << hour << minute << sec; }
};
int main()
{   const Time t1(15);
    Time const t2(16, 30, 0);
    t1.minute = 12; //编译错误
    t2.sec = 27;    //编译错误
    t1.display();   //编译错误
}
```

```
#include <iostream>
using namespace std;

class Time {
  public:
    int hour, minute, sec;
    Time(int h=0, int m=0, int s=0) {  hour = h; minute = m; sec = s; }
    void display() const {  cout << hour << minute << sec; }
};
int main()
{   const Time t1(15);
    Time const t2(16, 30, 0);
    t1.minute = 12; //编译错误
    t2.sec = 27;    //编译错误
    t1.display();   //编译正确
}
```

§14. C++知识补充

5. 类的共用数据的保护

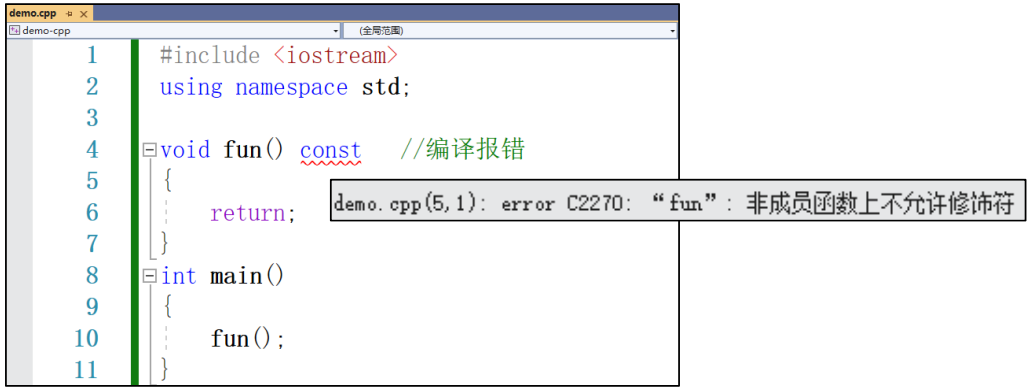
5.3. 常对象成员

使用：

★ 不能定义构造/析构函数为常成员函数(常识性问题)

★ 全局函数不能定义const

```
void fun() const //编译报错
{
    return;
}
int main()
{
    fun();
}
```



	普通数据成员	const数据成员	mutable数据成员	普通成员函数	const成员函数
普通对象 公有	读写	读	读写	可调用	可调用
const对象	读	读	读写	不可调用	可调用
普通成员函数	读写	读	读写	可调用	可调用
const成员函数	读	读	读写	不能调用	可调用

§14. C++知识补充

5. 类的共用数据的保护

5.4. 指向常对象的指针变量

⇔ 指向常量的指针变量

5.5. 指向对象的常指针

⇔ 常指针

5.6. 指向常对象的常指针

⇔ 指向常量的常指针

§12. 指针进阶

9. const指针

9.1. 共用数据的保护

9.2. 指向常量的指针变量

形式：const 数据类型 *指针变量名;

数据类型 const *指针变量名;

9.3. 常指针

形式：数据类型 *const 指针变量名;

9.4. 指向常量的常指针(同时满足9.2+9.3)

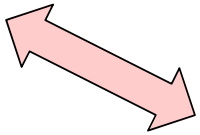
形式：const 数据类型 *const 指针变量名;

§14. C++知识补充

5. 类的共用数据的保护

5.7. 指向变量/对象的常引用

```
const int k;
int const &ref2 = k;

const Time t1;
const Time &ref1 = t1;
```



① 指向常对象的指针变量
⇔ 指向常量的指针变量

② 指向对象的常指针
⇔ 常指针

③ 指向常对象的常指针
⇔ 指向常量的常指针

★ 常引用与常指针的使用基本类似

第06模块：
★ 引用必须在声明时进行初始化，指向同类型变量，整个生存期内不能再指向其它变量
=> 普通引用已符合②且不可能是①

§14. C++知识补充

6. 类的静态成员

6.1. 引入

希望在同一个类的多个对象间实现数据共享

★ 一个对象修改，另一个对象访问得到修改后的值

★ 类似与全局变量的概念，但属于类，仅供该类的不同对象间共享数据

6.2. 静态数据成员

定义：

```
class 类名 {
    private/public:
        static 数据类型 成员名;
        ...
};
```

§14. C++知识补充

6. 类的静态成员

6.2. 静态数据成员

使用：

★ 静态数据成员不属于任何一个对象，不在对象中占用空间，
单独在静态数据区分配空间(初值为0，不随对象的释放而释放)，
一个静态数据成员只占有一个空间，所有对象均可共享访问

★ 静态数据成员同样受类的作用域限制

★ 静态数据成员必须进行初始化，初始化位置在类定义体后，
函数体外进行(此时不受类的作用域限制)
数据类型 类名::静态数据成员名=初值;

★ 虽然可以通过this指针访问，但是数据不在一起

```
#include <iostream>
using namespace std;

int a=10;
class Time {
  private:
    static int hour;
    int minute, sec;
  public:
    Time(int h=0, int m=0, int s=0) //未对hour赋值
    {  minute = m;
       sec = s;
    }
    void display() //打印三个成员的地址，目的?
    {  cout << hour << ':'<< minute << ':'<< sec << endl;
       cout << &hour << endl;
       cout << &minute << endl;
       cout << &sec << endl;
    }
};

int Time::hour = 5; //虽然是private，可以

int main()
{  Time t1;
   cout << sizeof(t1) << endl;
   cout << &a << endl;
   t1.display();
}
```

将display函数中的成员访问
均加上this->，观察结果
(例：this->hout/&this->minute)

```
8
a的地址
5:0:0
hour的地址(与a相邻，与下面两个差别很大)
minute的地址
sec的地址
```

§14. C++知识补充

6. 类的静态成员

6.2. 静态数据成员

使用：

★ 不能通过参数初始化表进行初始化，但可以通过赋值方式初始化

```
#include <iostream>
using namespace std;

class Time {
    private:
        static int hour;
        int minute, sec;
    public:
        Time(int h=0, int m=0, int s=0) : hour(h)
        {
            minute = m;  sec = s;
        }
        void display()
        {
            cout << hour << minute << sec << endl;
        }
};

int Time::hour = 5; //必须要有，否则编译错

int main()
{
    Time t1;
    cout << sizeof(Time) << endl;
    t1.display();
}
```

error C2438: “hour”: 无法通过构造函数初始化静态类数据

错误

```
#include <iostream>
using namespace std;

class Time {
    private:
        static int hour;
        int minute, sec;
    public:
        Time(int h=0, int m=0, int s=0)
        {
            hour = h;  minute = m;  sec = s;
        }
        void display()
        {
            cout << hour << minute << sec << endl;
        }
};

int Time::hour = 5; //必须要有，否则编译错

int main()
{
    Time t1;
    cout << sizeof(Time) << endl;
    t1.display();
}
```

问1：hour初始化为5，在构造函数中又被赋值为h，最终是几？

问2：哪个先执行?为什么?

正确

§14. C++知识补充

6. 类的静态成员

6.2. 静态数据成员

使用：

★ 不能通过参数初始化表进行初始化，但可以通过赋值方式初始化

★ 既可以通过类型引用，也可以通过对象名引用

★ 结论：静态数据成员不是面向对象的概念，它
破坏了数据的封装性，但方便使用，提高了运行效率

```
#include <iostream>
using namespace std;

class Time {
 public: //不符合规范，为了在main中直接访问
    static int hour;
    int minute;
    int sec;
  public:
    Time();
    Time(int h,int m,int s);
    void display();
};

int Time::hour = 10;

Time::Time()
{
    hour=0; minute=0; sec=0;
}
Time::Time(int h,int m,int s)
{
    hour=h; minute=m; sec=s;
}
void Time::display()
{   cout << hour << ":" << minute << ":" << sec << endl;
}
```

```
int main()
{
    Time t1(14,15,23), t2;
    t1.display();   0:15:23
    t2.display();   0:0:0
    t1.hour = 8;    //对象名引用
    t1.display();   8:15:23
    t2.display();   8:0:0
    Time::hour = 19;//类型引用
    t1.display();   19:15:23
    t2.display();   19:0:0

    return 0;
}
```

为什么既不是14:15:23，也不是10:15:23？

§14. C++知识补充

6. 类的静态成员

6.3. 静态成员函数

定义： class 类名 {

```
        private/public:
            static 返回类型 函数名(形参表);
            ...
    };
```

调用：

类名::成员函数名(实参表);

任意对象名.成员函数名(实参表);

使用：

★ 没有this指针，不属于某个对象

普通成员函数：

Time t1;

t1.display() ⇔ t1.display(&t1);

静态成员函数：

无

```
#include <iostream>
using namespace std;

class test {
    public:
        static void fun(int x)
        {
            cout << x << endl;
        }
};
int main()
{
    test t1;
    t1.fun(10);     //10
    test::fun(15); //15
}
```

§14. C++知识补充

6. 类的静态成员

6.3. 静态成员函数

使用：

★ 没有this指针，不属于某个对象

★ 允许体内实现或体外实现

★ 静态成员函数中可以直接访问静态数据成员

体内实现

```
#include <iostream>
using namespace std;

class test {
  private:
    static int a;
  public:
    static void fun()
    {   cout << a << endl;
    }
};
int test::a = 10;
int main()
{   test t1;
    t1.fun();   //10
    test::fun();//10
}
```

```
#include <iostream>
using namespace std;

class test {
  private:
    static int a;
  public:
    static void fun()
    {   cout << this->a << endl;
    }
};
int test::a = 10;
int main()
{   test t1;
    t1.fun();
    test::fun();
}
```

静态函数没有this指针，显式加上后会报错!!!

error C2355: “this”：只能在非静态成员函数或非静态数据成员初始值设定项的内部引用

体外实现

```
#include <iostream>
using namespace std;
class test {
    private:
        static int a;
    public:
        static void fun();
};
int test::a = 10;
void test::fun()  //此处不能static
{   cout << a << endl;
}
int main()
{   test t1;
    t1.fun();   //10
    test::fun();//10
}
```

§14. C++知识补充

6. 类的静态成员

6.3. 静态成员函数

使用：

★ 在静态成员函数中不能对非静态数据成员进行直接访问，而要通过对象参数的方式(不提倡!!!)

```
#include <iostream>
using namespace std;

class test {
   private:
      static int a;
      int b;
   public:
      static void fun()
      {
         a=10; //正确
         b=11; //错误，因为没有this指针，不知道
       }       //应该访问那个对象的b成员
};

int test::a = 5;

int main()
{
    test t1;
    t1.fun();
    test::fun();
}
```

```
demo.cpp(12,9): error C2597: 对非静态成员“test::b”的非法引用
demo.cpp(7): message : 参见“test::b”的声明
```

```
#include <iostream>
using namespace std;

class test {
   private:
      static int a;
      int b;
   public:
      static void fun(test &t)
      {
         a=10;    //正确
         t.b=11;  //正确
      }
};

int test::a = 5;

int main()
{
    test t1, t2;
    t1.fun(t1);
    test::fun(t2);
}
```

不提倡，建议静态成员函数只访问静态数据成员

§14. C++知识补充

7. 类模板

7.1. 函数模板(4.9的内容)

§4. 函数

4.9. 函数模板

函数重载的不足：对于参数个数相同，类型不同，
而实现过程完全相同的函数，
仍要分别给出各个函数的实现

问题:两段一样的代码
能否合并为一段？

```
int max(int x, int y)
{
    return x>y?x:y;
}
double max(double x, double y)
{
    return x>y?x:y;
}
```

一段代码,两个功能
1、两个int型求max
2、两个double型求max

```
#include <iostream>
using namespace std;
template <typename T> //虚类型T
T my_max(T x, T y)
{   cout << sizeof(x) << ' ';
    return x > y ? x : y;
}
int main()
{   int    a=10, b=15;
    double f1=12.34, f2=23.45;
    cout << my_max(a, b) << endl;
    cout << my_max(f1, f2) << endl;
    return 0;
}
```

```
4 15
8 23.45
```

函数模板：建立一个通用函数，其返回类型及参数类型不具体指定，用一个虚拟类型来代替，该通用函数称为函数模板，调用时再根据不同的实参类型来取代模板中的虚拟类型，从而实现不同的功能

问题1：如果传入两个unsigned int型数据，
T的类型被实例化为什么？如何证明？
问题2：如果x,y的类型不同，自行摸索转换规律

§14. C++知识补充

7. 类模板

7.1. 函数模板(4.9的内容)

§4. 函数

4.9. 函数模板

函数重载的不足：对于参数个数相同，类型不同，而实现过程完全相同的函数，仍要分别给出各个函数的实现

函数模板：建立一个通用函数，其返回类型及参数类型不具体指定，用一个虚拟类型来代替，该通用函数称为函数模板，
调用时再根据不同的实参类型来取代模板中的虚拟类型，从而实现不同的功能

使用：

★ 仅适用于参数个数相同、类型不同，实现过程完全相同的情况

★ typename可用class替代

★ 类型定义允许多个

```
template <typename T1, typename t2>
template <class T1, class t2>
```

```
#include <iostream>
using namespace std;
template <typename T1, typename T2>
char my_max(T1 x, T2 y)
{
    cout << sizeof(x) << ' ';
    cout << sizeof(y) << ' ';
    return x>y ? 'A' : 'a';
}
int main()
{
    int    a = 10, b = 15;
    double f1 = 12.34, f2 = 23.45;
    cout << my_max(a, f1) << endl;
    cout << my_max(f2, b) << endl;
    return 0;
}
```

?

问：max(a,f1)时，T1/T2类型分别是?
max(f2,b)时，T1/T2类型分别是?

§14. C++知识补充

7. 类模板

7.2. 类模板

引入：多个类，功能及实现完全相同，仅数据类型不同

```
class compare_int {
 private:
   int x, y;
 public:
   compare_int(int a, int b)
     {  x=a;   y=b;   }
   int max()
     {  return x>y?x:y; }
   int min()
     {  return x<y?x:y; }
 };
```

```
class compare_float {
 private:
   float x, y;
 public:
   compare_float(float a, float b)
     {  x=a;   y=b;   }
   float max()
     {  return x>y?x:y; }
   float min()
     {  return x<y?x:y; }
 };
```

合并为一个类型
为虚类型T的类

```
#include <iostream>
using namespace std;

template <class T>
class compare {
    private:
        T x, y;
    public:
        compare(T a, T b)
        {  x=a;   y=b;
        }
        T max()
        {  return x > y ? x : y;
         }
        T min()
        {  return x < y ? x : y;
        }
};
int main()
{    compare <int>   c1(10,15);
     compare <float> c2(10.1f, 15.2f);
     cout << c1.max() << endl;   15
     cout << c2.min() << endl;   10.1
}
```

§14. C++知识补充

7. 类模板

7.2. 类模板

使用：

★ 类模板使用的多个类需要满足下面的条件

- 数据成员个数相同，类型不同
- 成员函数个数相同，类型不同，实现过程完全相同

=> 推论：如果两个类大部分相同，可以通过定义不需要的数据成员及成员函数来达到使用类模板的目的(按实际情况决定)

```
class name_1 {
 private:
   int x, y, z;
 public:
   int f1(); //实际不需要
   void f2(int);
   void f3(int);
   void f4(int);
};
```

```
class name_1 {
 private:
   float x, y, z; //z实际不需要
 public:
   float f1();
   void f2(float);
   void f3(float);
   void f4(float);
};
```

§14. C++知识补充

7. 类模板

7.2. 类模板

使用：

★ 类模板使用的多个类需要满足下面的条件

★ 类模板可以看作是类的抽象，称为参数化的类

★ 类模板成员函数体外实现时形式有所不同

体内实现

```
#include <iostream>
using namespace std;

template <class T>
class compare {
    private:
        T x, y;
    public:
        compare(T a, T b)
             {  x=a;   y=b;   }
        T max()
             {  return x>y?x:y; }
        T min()
             {  return x<y?x:y; }
};

int main()
{   compare <int>   c1(10,15);
    compare <float> c2(10.1f, 15.2f);
    cout << c1.max() << endl;
    cout << c2.min() << endl;
}
```

体外实现

```
#include <iostream>
using namespace std;

template <class T>
class compare {
    private:
        T x, y;
    public:
        compare(T a, T b);
        T max();
        T min();
};
template <class T>  //每个体外实现的函数前都要
compare<T>::compare(T a, T b)
{   x=a;
    y=b;
}
template <class T>  //每个体外实现的函数前都要
T compare<T>::max()
{  return x>y?x:y;
}
template <class T>  //每个体外实现的函数前都要
T compare<T>::min()
{  return x<y?x:y;
}
int main()
{    compare <int>   c1(10,15);
     compare <float> c2(10.1f, 15.2f);
     cout << c1.max() << endl;
     cout << c2.min() << endl;
}
```

普通类构造函数的体外实现：

```
test::test(int x, int y)
{
    …
}
```

普通类成员函数的体外实现：

```
int test::fun(int x, int y)
{
    …
}
```

§14. C++知识补充

7. 类模板

7.2. 类模板

使用：

★ 类模板使用的多个类需要满足下面的条件

- 数据成员个数相同，类型不同
- 成员函数个数相同，类型不同，实现过程完全相同

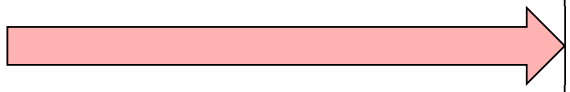
=> 推论：如果两个类大部分相同，可以通过定义不需要的数据成员及成员函数来达到使用类模板的目的(按实际情况决定)

★ 类模板可以看作是类的抽象，称为参数化的类

★ 类模板成员函数体外实现时形式有所不同

★ 类型定义允许多个

```
template <class t1, class t2>
```



```
#include <iostream>
using namespace std;

template <class t1, class t2>
class test {
  private:
    t1 x;
    t2 y;
  public:
    test(t1 a, t2 b) {
      x=a;
      y=b;
      }
};

int main()
{
    test <int,float> c1(10,15.1);
    test <int,int>   c2(10,15);
}
```

§14. C++知识补充

8. 枚举类

8.1. C方式传统枚举存在的问题

★ 不同enum的枚举常量值不能相同，但实际应用中有此需求

★ 不同enum的变量/常量不应该允许比较，但编译运行正确(无法解释语义)

★ 不同enum的变量/常量均可以直接整型输出，容易造成误解

★ C方式，enum可以直接用整型赋值，不检查范围

★ C++方式，enum不能直接整型赋值，需加强制类型转换，不检查范围

§14. C++知识补充

8. 枚举类

8.1. C方式传统枚举存在的问题

★ 不同enum的枚举常量值不能相同，但实际应用中有此需求

```
#include <iostream>
using namespace std;
enum week { sun, mon, tue, wed, thu, fri, sat };
enum candidates { zhao, qian, sun, li };
int main()
{
    cout << wed << endl;
    return 0;
}
```

```
demo.cpp(4,31): error C2365: “sun”: 重定义；以前的定义是“枚举数”
demo.cpp(3): message : 参见“sun”的声明
```

★ 不同enum的变量/常量均可以直接整型输出，容易造成误解

```
#include <iostream>
using namespace std;
enum week { sun, mon, tue };
enum candidates { zhao, qian, li };
int main()
{   cout << sun << ' ' << li << endl;
    printf("%d %d\n", sun, li);
    return 0;
}
```

```
2 0
2 0
```

★ 不同enum的变量/常量不应该允许比较，但编译运行正确(无法解释语义)

```
#include <iostream>
using namespace std;
enum week { sun, mon, tue };
enum candidates { zhao, qian, li };
int main()
{   enum week w1=mon;
    enum candidates c1=qian;
    cout << (sun < li) << ' ' << (w1==c1)<< endl;
}
```

```
1 1
```

§14. C++知识补充

8. 枚举类

8.1. C方式传统枚举存在的问题

★ C方式，enum可以直接用整型赋值，不检查范围

★ C++方式，enum不能直接整型赋值，需加强制类型转换，不检查范围

```
#include <stdio.h>

enum week { sun, mon, tue };
int main()
{
    enum week w1 = mon, w2 = 0, w3 = 4; //超定义范围
    printf("%d %d %d\n", w1, w2, w3);
    return 0;
}
```

注意：源程序必须.c形式

```
1 0 4
```

```
#include <iostream>
using namespace std;
enum week { sun, mon, tue };
int main()
{
    enum week w1 = mon, w2 = 0, w3 = 4; //超定义范围
    cout << w1 << ' ' << w2 << ' ' << w3 << endl;
    return 0;
}
```

cpp：直接整型，报错

```
demo.cpp(6,31): error C2440: “初始化”: 无法从“int”转换为“week”
demo.cpp(6,28): message : 转换为枚举类型需要显式强制转换(static_cast、C 样式强制转换或带圆括号函数样式强制转换)
demo.cpp(6,39): error C2440: “初始化”: 无法从“int”转换为“week”
demo.cpp(6,36): message : 转换为枚举类型需要显式强制转换(static_cast、C 样式强制转换或带圆括号函数样式强制转换)
```

```
#include <iostream>
using namespace std;
enum week { sun, mon, tue };
int main()
{
    enum week w1 = mon, w2 = week(0), w3 = week(4);
    cout << w1 << ' ' << w2 << ' ' << w3 << endl;
    return 0;
}
```

cpp：强转，VS报IntelliSense

C26812 枚举类型"week"未设定范围。相比于 "enum"，首选 "enum class" (Enum.3)。

```
1 0 4
```

§14. C++知识补充

8. 枚举类

8.2. 枚举类的定义与使用

定义：enum class 枚举类型 { 枚举常量1, ...,枚举常量n }

使用：类名::常量值

★ 枚举类常量为整型值，从0开始（同enum）

★ 枚举类常量不能直接输出(enum可直接输出为整型)

★ 其余使用方法基本同enum

★ 不同enum class可定义相同的枚举常量标识符(常量值可不同)

★ 不同enum class之间的常量值不允许直接比较

★ enum class的枚举变量/常量不允许直接输出，需要加强制类型转换

★ C++方式，enum不能直接整型赋值，需加强制类型转换，不检查范围(同enum)

§14. C++知识补充

8. 枚举类

8.2. 枚举类的定义与使用

★ 不同enum class可定义相同的枚举常量标识符(常量值可不同)

编译正确

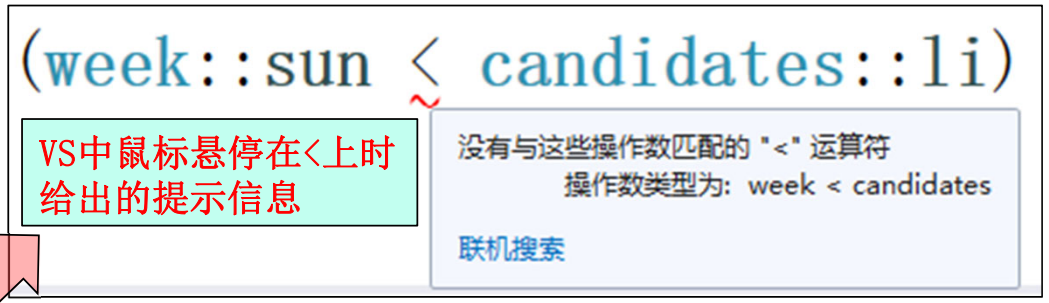
```
#include <iostream>
using namespace std;

enum class week { sun, mon, tue, wed, thu, fri, sat };
enum class candidates { zhao, qian, sun, li };
int main()
{
    return 0;
}
```

★ 不同enum class之间的常量值不允许直接比较

编译报错

```
#include <iostream>
using namespace std;

enum class week { sun, mon, tue };
enum class candidates { zhao, qian, li };
int main()
{   cout << (week::sun < candidates::li) << endl;
    return 0;
}
```

error C2676: 二进制“<”：“week”不定义该运算符或到预定义运算符可接收的类型的转换



§14. C++知识补充

8. 枚举类

8.2. 枚举类的定义与使用

★ enum class的枚举变量/常量不允许直接输出，需要加强制类型转换

```
#include <iostream>
using namespace std;
enum class week { sun, mon, tue, wed, thu, fri, sat };
int main()
{
    cout << wed << endl;
    printf("%d\n", mon);
    return 0;
}
```

enum class：直接写常量值，报未声明错

```
demo.cpp(6,13): error C2065: “wed”: 未声明的标识符
demo.cpp(7,20): error C2065: “mon”: 未声明的标识符
```

```
#include <iostream>
using namespace std;

enum class week { sun, mon, tue, wed, thu, fri, sat };
int main()
{
    cout << week::wed << endl;
    return 0;
}
```

enum class：写类名::常量值 方式
cout直接输出报错

```
demo.cpp(6,23): error C2679: 二元“<<”: 没有找到接受“week”类型的右操作数的运算符(或没有可接受的转换)
```

```
#include <iostream>
using namespace std;

enum class week { sun, mon, tue, wed, thu, fri, sat };
int main()
{   cout << int(week::end) << endl;
    printf("%d\n", week::mon);
    return 0;
}
```

enum class：写类名::常量值 方式
cout需强转后输出整型
printf直接输出整型值

```
3
1
```

§14. C++知识补充

8. 枚举类

8.2. 枚举类的定义与使用

★ C++方式，enum不能直接整型赋值，需加强制类型转换，不检查范围(同enum)

enum class无IntelliSense

```
#include <iostream>
using namespace std;
enum class week { sun, mon, tue };
int main()
{
    week w1 = week::mon, w2 = week(0), w3 = week(4);
    printf("%d %d %d\n", w1, w2, w3);
    return 0;
}
```

```
1 0 4
```

§14. C++知识补充

8. 枚举类

8.3. enum与enum class的比较

★ 不同enum的枚举常量值不能相同，但实际应用中有此需求

=> enum class允许相同

★ 不同enum的变量/常量不应该允许比较，但编译运行正确(无法解释语义)

=> enum class不允许比较

★ 不同enum的变量/常量均可以直接整型输出，容易造成误解

=> enum class的cout不允许直接输出(加强制类型转换)，printf允许

★ C方式，enum可以直接用整型赋值，不检查范围

=> enum class不支持C方式

★ C++方式，enum不能直接整型赋值，需加强制类型转换，不检查范围

=> enum class同，且VS下无 IntelliSense