例：程序unsign\_sign

unsigned char a = 128;

char b = a;

short c = b;

printf("%d\n", (unsigned short)a);

printf("%d\n", c);

printf("%d\n", (unsigned short)b);

输出为：128

-128

65408

原因是b的值为-128（128的原码和补码是1000 0000，转换为char型时，1000 0000是-128的补码）.unsigned char与char的字节长度是相同的，在相互赋值过程中，其内存中的补码未发生改变。

将char转换为short时，其字节长度发生改变，但数值未发生变化，在16位中，-128的原码为1000 0000 1000 0000，在计算机中补码为：原码除符号位以外取反加1，就为1111 1111 1000 0000，（计算机存储的为补码）强制转换为无符号数时为65408.

例：程序unsign\_sign1

int a = -1;

unsigned int b = 0;

if (a < b)

{

printf("a < b\n");

}

else if (a == b)

{

printf("a = b \n");

}

else

{

printf("a > b \n");

}

输出结果为：a > b

原因：当有符号数与无符号数进行比较运算时，C/C++会将有符号数当成无符号数来处理，-1的二进制补码为32个1，当成无符号数来处理时显然比0大。

例：程序unsign\_sign1

unsigned int g = 0;

int h = 0;

if (h < (g - 1))

{

printf("%u\n", (g - 1));

printf("h < (g - 1)\n");

}

输出结果为

4294967295

h < (g - 1)

原因：无符号数0的二进制补码为32个0，而-1的二进制补码为32个1，从而(g - 1)得到的二进制补码为32个1，按照无符号数来处理这个二进制补码，得到的值就是4294947295，显然0小于这个数。

例：程序unsign\_sign1

int c = 2147483647;

// 产生了溢出，d = 2147483647

// -2147483648的补码为1000 0000 0000 0000 0000 0000 0000 0000

// -1的补码为1111 1111 1111 1111 1111 1111 1111 1111

// 相加后溢出为0111 1111 1111 1111 1111 1111 1111 1111

// 为2147483647

int d = -2147483648 - 1;

if (c < d)

{

*printf*("c < d\n");

}

else if (c == d)

{

*printf*("d = %d\n", d);

*printf*("c = d \n");

}

else

{

*printf*("c > d \n");

}

输出为：

d = 2147483647

c = d

无符号数和有符号数的四则运算：

例：程序unsign\_sign2

unsigned int k = 1;

int m = -9;

short int m1 = -1;

unsigned short int n = 65535;

int s = 0;

// k为无符号数，m+8的结果-1的补码当成无符号数处理

k = m + 8;

*printf*("%u\n", k); // 4294967295

// m1为有符号数，n-9的结果65526的补码当成有符号数处理

m1 = n - 9;

*printf*("%d\n", m1); // -10

// 发生溢出，得到的是-2的补码

// 1111 1111 1111 1111 1111 1111 1111 1111 +

// 1111 1111 1111 1111 1111 1111 1111 1111

// = 1111 1111 1111 1111 1111 1111 1111 1110

s = k + k;

*printf*("%d\n", s); // -2

无符号数和有符号数的移位：（只有整数才能移位）

例：程序unsign\_sign3

对于无符号数，左移1位相当于乘以2，右移相当于除以2，注意：左移有可能会有溢出发生。

unsigned char b = 54;

unsigned char c = 254;

b = b << 1;

// 254:1111 1110

// 左移1位变成：1111 1100

c = c << 1; // 252

*printf*("b = %u\n", b);

*printf*("c = %u\n", c);

输出为：

b = 108

c = 252

有符号数的移位：

如果为正数，右移相当于除以2，左移则情况比较复杂，有可能改变符号位：

char m = 2;

char n = 65;

m = m >> 1; // 1

// n : 0100 0001

// 左移1位：1000 0010

n = n << 1;

*printf*("n = %d\n", n); // -126

// 左移1位：0000 0100

n = n << 1;

*printf*("n = %d\n", n); // 4

如果为负数，左移与正数的左移有些类似，有可能会改变符号位，对于右移，使用符号位来做为补充位，如果负数一直右移，最终会变为-1.

char x = -102;

char y = -126;

x = x >> 1;

*printf*("x = %d\n", x); // -51

x = x >> 10; // -1

*printf*("x = %d\n", x);

y = y << 1; // 4

*printf*("y = %d\n", y);

输出为：

x = -51

x = -1

y = 4