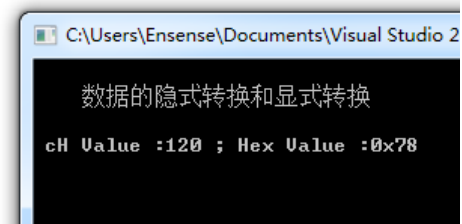


C 语言中数据的隐式转换和显式转换

1. 不同数据类型间的赋值

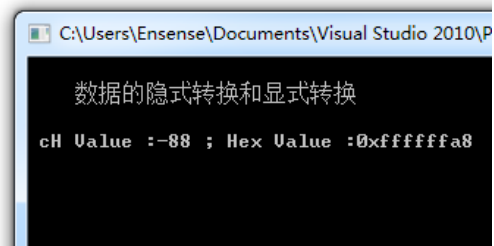
1.1 int 转换成 char

```
1 int _tmain(int argc, _TCHAR* argv[])
2 {
3     cout<<endl;
4     cout<<" 数据的隐式转换和显式转换"<<endl;
5
6     int nOrg = 0x12345678;
7
8     char cH = nOrg ;
9     printf("\r\n cH Value :%d ; Hex Value :0x%x", cH, cH);
10 }
```



Int 转换成 char 时，直接获取 int 的低 8bit 值。 0x12345678 直接获取低位 0x78 。

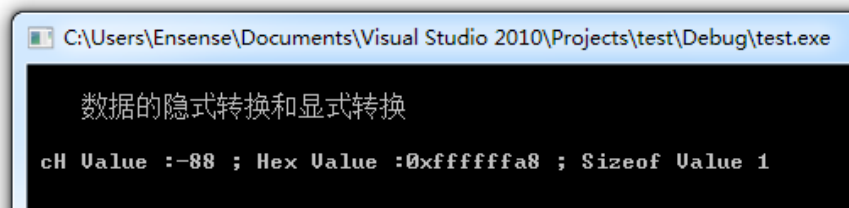
```
1 int _tmain(int argc, _TCHAR* argv[])
2 {
3     cout<<endl;
4     cout<<" 数据的隐式转换和显式转换"<<endl;
5
6     int nOrg = 0x123456A8;
7
8     char cH = nOrg ;
9     printf("\r\n cH Value :%d ; Hex Value :0x%x", cH, cH);
10 }
```



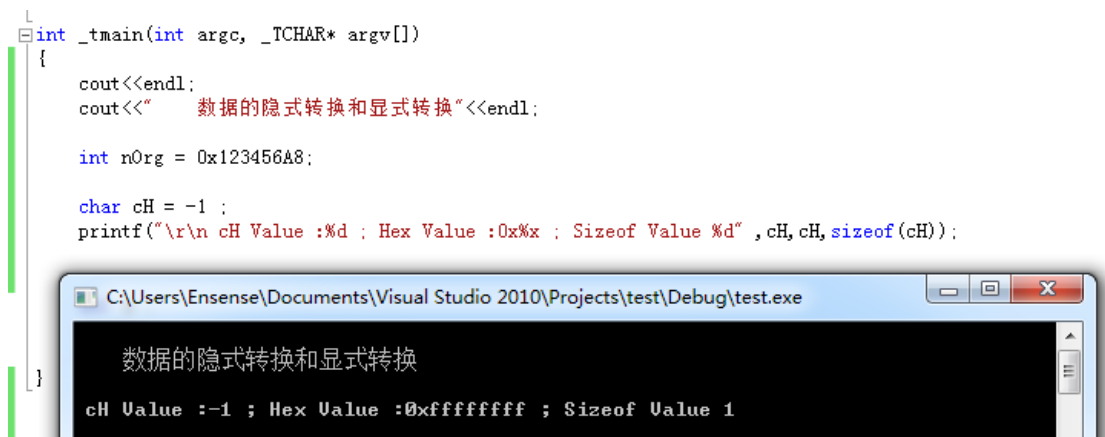
Int 转换成 char，低 8bit 的最高位直接作为符号位。 0x123456A8 直接获取低位 0xA8 。直接作为负数处理。

Printf(); 在打印时会自动把负数扩充为 32bit。

```
12 int _tmain(int argc, _TCHAR* argv[])
13 {
14     cout<<endl;
15     cout<<" 数据的隐式转换和显式转换"<<endl;
16
17     int nOrg = 0x123456A8;
18
19     char cH = (char)nOrg ;
20     printf("\r\n cH Value :%d ; Hex Value :0x%x ; Sizeof Value %d", cH, cH, sizeof(cH));
21
22
23     int a ;
24     cin>>a ;
25     return 0;
26 }
```



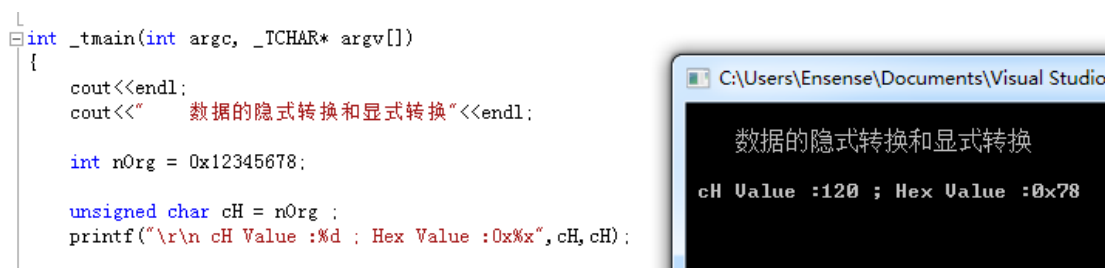
调用 sizeof 计算出 cH 的长度是 1 个字节。可以佐证上面的 0xffffffa8 是 printf();自动扩充的。



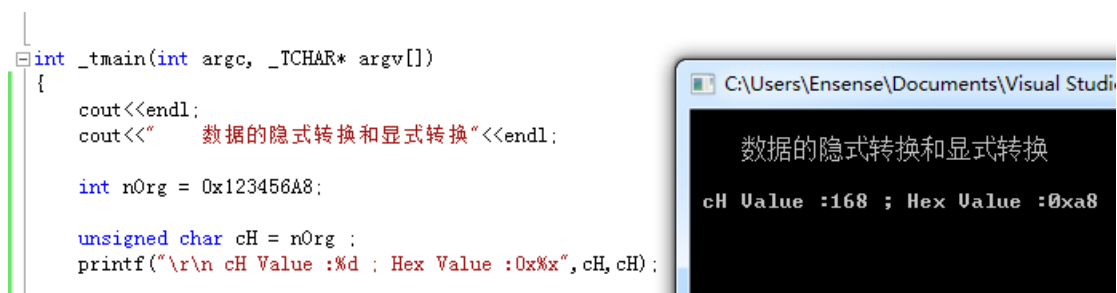
直接把 cH 赋值为 -1，printf();打印出来的也是 0xffffffff，更是证明了这里是 printf();问题。不去多考虑了。

PS:应该是 printf()把所有数据都扩充到 32bit，正数的高位是 0，所以没显示，负数是 1，就都打印出来了，感兴趣的自行研究，我对 printf()知之甚少，不做深究。

1.2 int 转换成 unsigned char



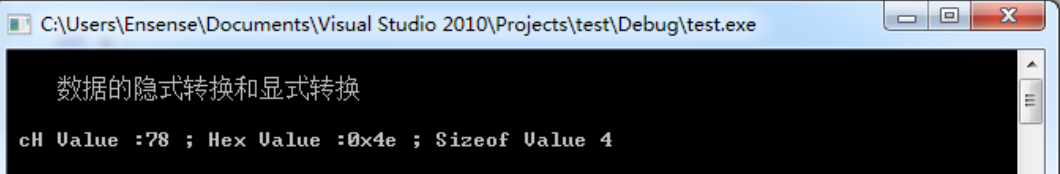
Int 转换成 unsigned char 时，直接获取 int 的低 8bit 值。0x12345678 直接获取低位 0x78。



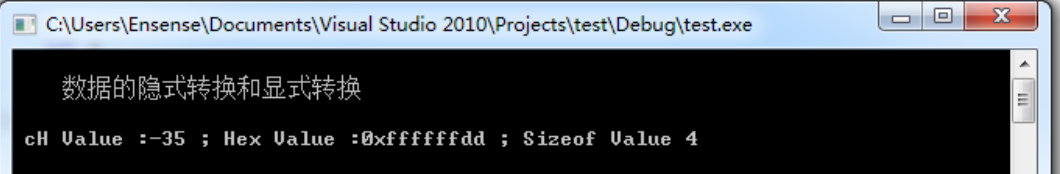
Int 转换成 unsigned char 时，直接获取 int 的低 8bit 值。0x12345678 直接获取低位 0x78。
unsigned char 型没有符号位，都是正数。

1.3 char 转换成 int

```
2 |
3 | int _tmain(int argc, _TCHAR* argv[])
4 | {
5 |     cout<<endl;
6 |     cout<<"    数据的隐式转换和显式转换"<<endl;
7 |
8 |     char cH = 78 ;
9 |
10 |    int nOrg = cH;
11 |
12 |    printf("\r\n cH Value :%d ; Hex Value :0x%x ; Sizeof Value %d",nOrg,nOrg,sizeof(nOrg));
13 |
14 |
15 |
16 |
17 |
18 |
19 |
20 |
```



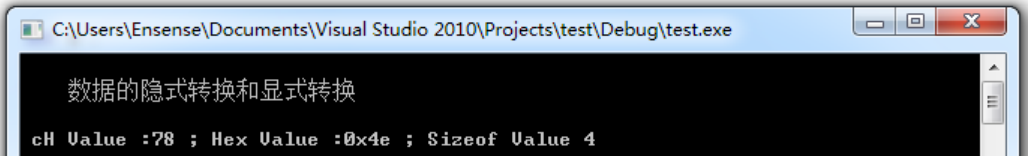
```
2 |
3 | int _tmain(int argc, _TCHAR* argv[])
4 | {
5 |     cout<<endl;
6 |     cout<<"    数据的隐式转换和显式转换"<<endl;
7 |
8 |     char cH = -35 ;
9 |
10 |    int nOrg = cH;
11 |
12 |    printf("\r\n cH Value :%d ; Hex Value :0x%x ; Sizeof Value %d",nOrg,nOrg,sizeof(nOrg));
13 |
14 |
15 |
16 |
17 |
18 |
19 |
20 |
```



char 转换成 int 时，直接在高位补充符号位。数据大小本身不变。

1.4 char 转换成 unsigned int

```
2 |
3 | int _tmain(int argc, _TCHAR* argv[])
4 | {
5 |     cout<<endl;
6 |     cout<<"    数据的隐式转换和显式转换"<<endl;
7 |
8 |     char cH = 78 ;
9 |
10 |    unsigned int nOrg = cH;
11 |
12 |    printf("\r\n cH Value :%d ; Hex Value :0x%x ; Sizeof Value %d",nOrg,nOrg,sizeof(nOrg));
13 |
14 |
15 |
16 |
17 |
18 |
19 |
20 |
```

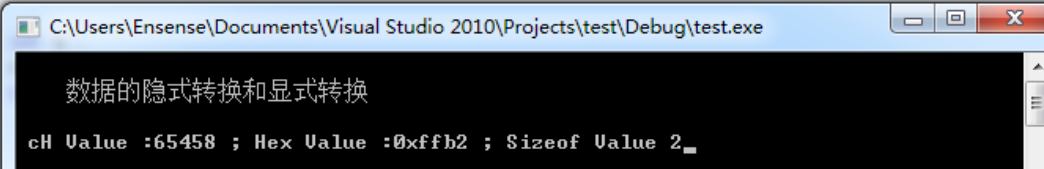


```
int _tmain(int argc, _TCHAR* argv[])
{
    cout<<endl;
    cout<<"    数据的隐式转换和显式转换"<<endl;

    char cH = -78 ;

    unsigned short nOrg = cH;

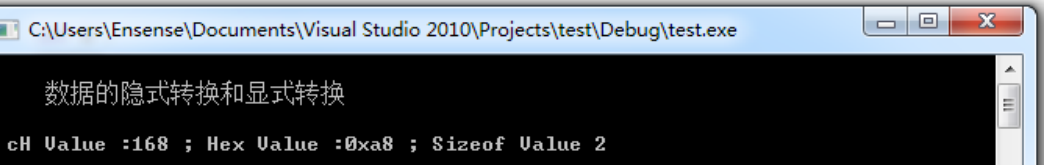
    printf("\r\n cH Value :%d ; Hex Value :0x%x ; Sizeof Value %d", nOrg, nOrg, sizeof(nOrg));
}
```



负数这里我改成 unsigned short 了，用来避免 printf();溢出什么的。但是从结果还是能看出来：

char 在扩充成 unsigned int 时，高位补充的是符号位。

```
2 |
3 | int _tmain(int argc, _TCHAR* argv[])
4 | {
5 |     cout<<endl;
6 |     cout<<"    数据的隐式转换和显式转换"<<endl;
7 |
8 |     unsigned char cH = 0xA8 ;
9 |
10 |    unsigned short nOrg = cH;
11 |
12 |    printf("\r\n cH Value :%d ; Hex Value :0x%x ; Sizeof Value %d", nOrg, nOrg, sizeof(nOrg));
13 |
14 |
15 | }
```



unsigned char 扩充成 int 时，高位补充的是 0 ；

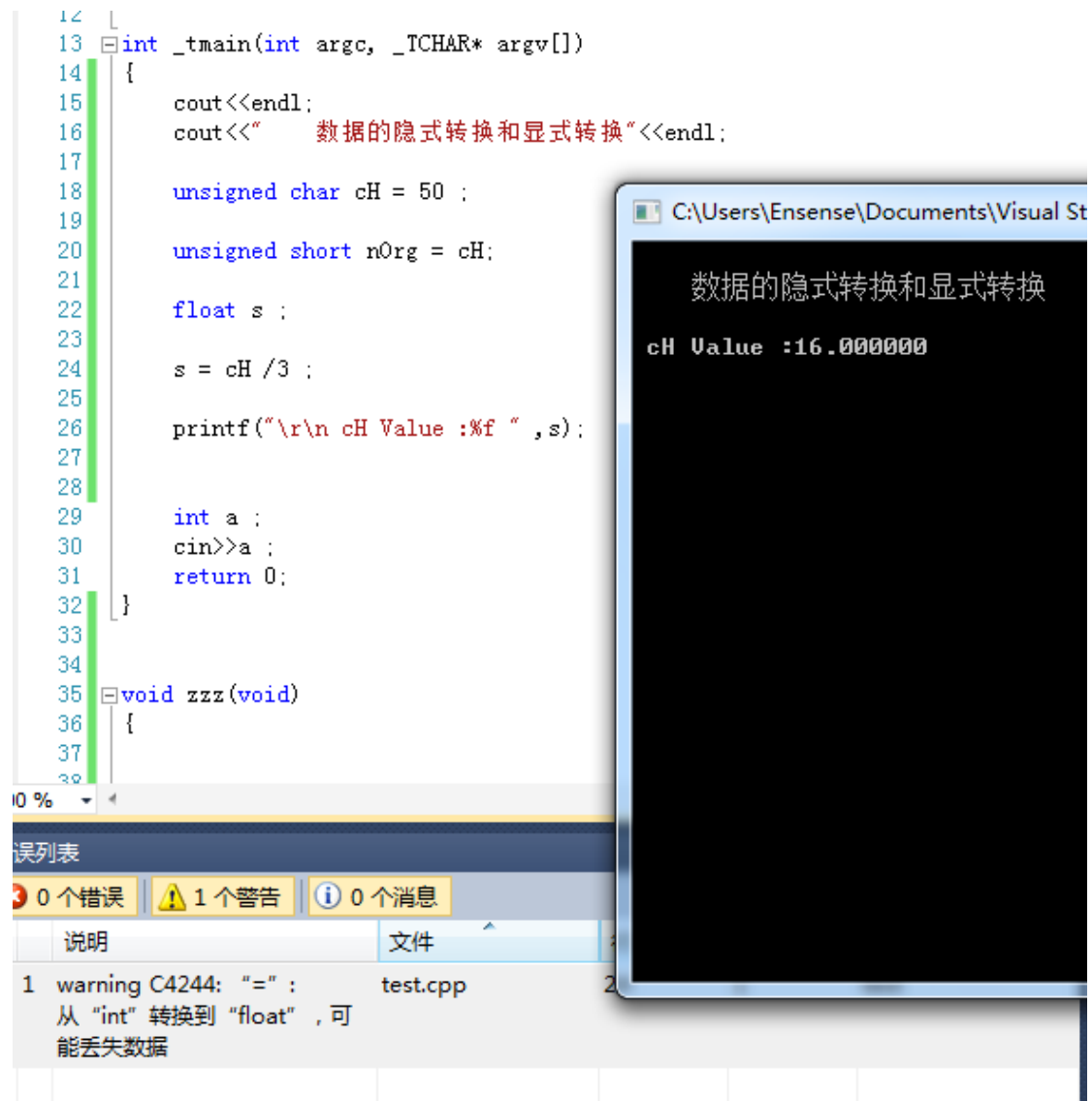
1.5 总结

对于多字节的数转换成少字节的数时，直接取多字节数的低位相应字节。如果转换后的数据是有符号的，最高位作为符号位。

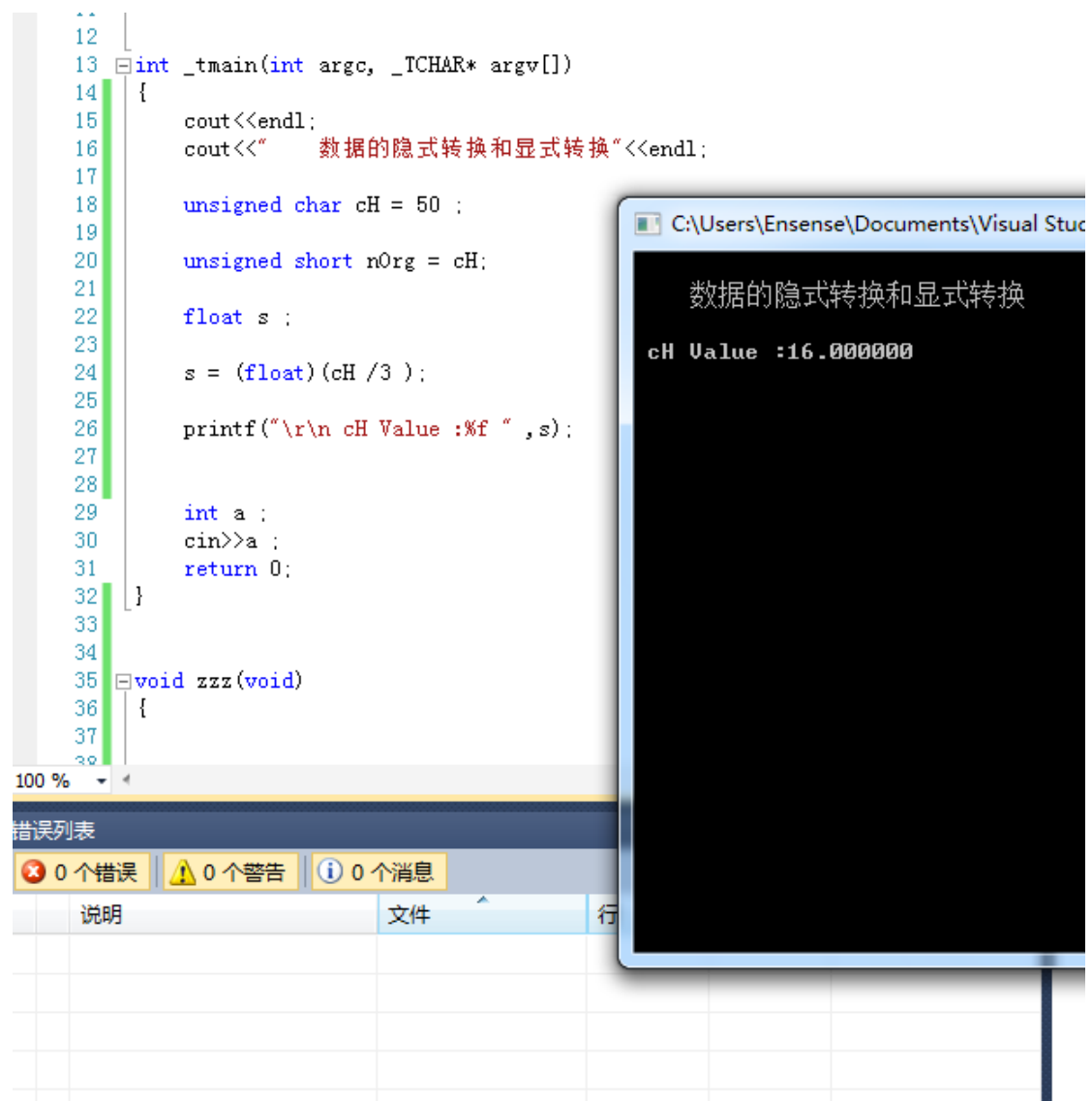
对于少字节的数扩充到多字节的数。如果少字节的数有符号，则扩充后的数高位扩充符号位，如果少字节的数无符号，则高位字节扩充 0。

2. 计算中的数据转换

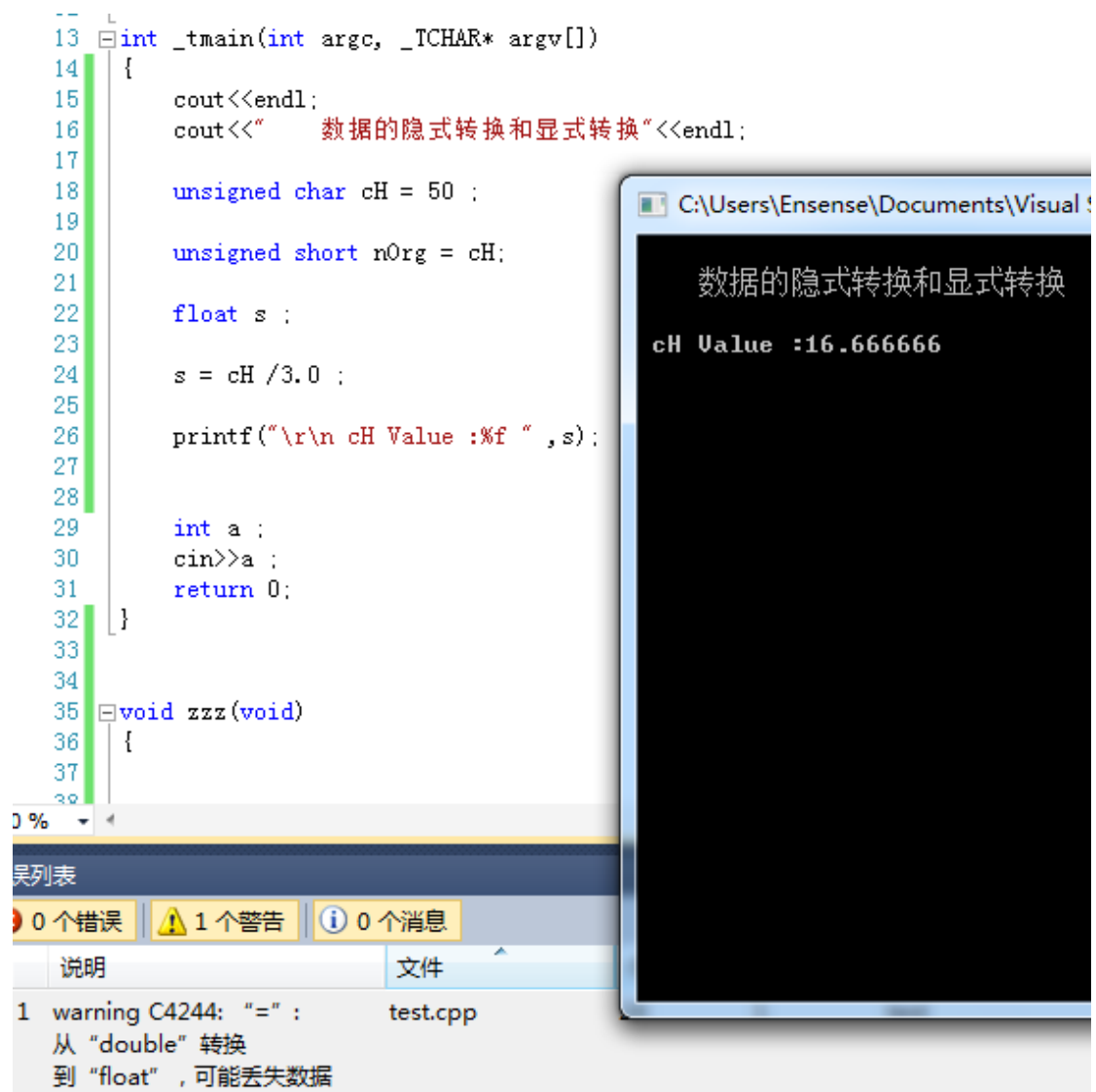
2.1 先看现象



C 语言在只有整形数据参与计算时，默认把数据扩充到 int，然后计算，计算结果默认为 int。所有 $50/3 = 16$ 。16 再转换成 float 型，是 16.000000。出现计算错误和数据精度丢失警告。



强行把计算结果显式的转换成 float 型，虽然警告消失了，但在转换发生在计算之后，计算结果已经是 16 了，再转换还是 16.000000 。



C 语言中直接定义的小数默认保存为 `double`。有 `double` 型数据参与计算时，所有数据先扩充为 `double` 型，再计算结果。计算结果保存为 `double` 型。`Double` 型数据可以保存小数，所以结果精度未丢失，但是 `double` 和 `float` 类型不符，会警告。

```
12 |
13 | int _tmain(int argc, _TCHAR* argv[])
14 | {
15 |     cout<<endl;
16 |     cout<<"    数据的隐式转换和显式转换"<<endl;
17 |
18 |     unsigned char cH = 50 ;
19 |
20 |     unsigned short nOrg = cH;
21 |
22 |     float s ;
23 |
24 |     s = cH / (float)3 ;
25 |
26 |     printf("\r\n cH Value :%f ",s);
27 |
28 |
29 |     int a ;
30 |     cin>>a ;
31 |     return 0;
32 | }
33 |
34 |
35 | void zzz(void)
36 | {
37 |
38 | }
```

100 %

错误列表

0 个错误 0 个警告 0 个消息

说明	文件
----	----

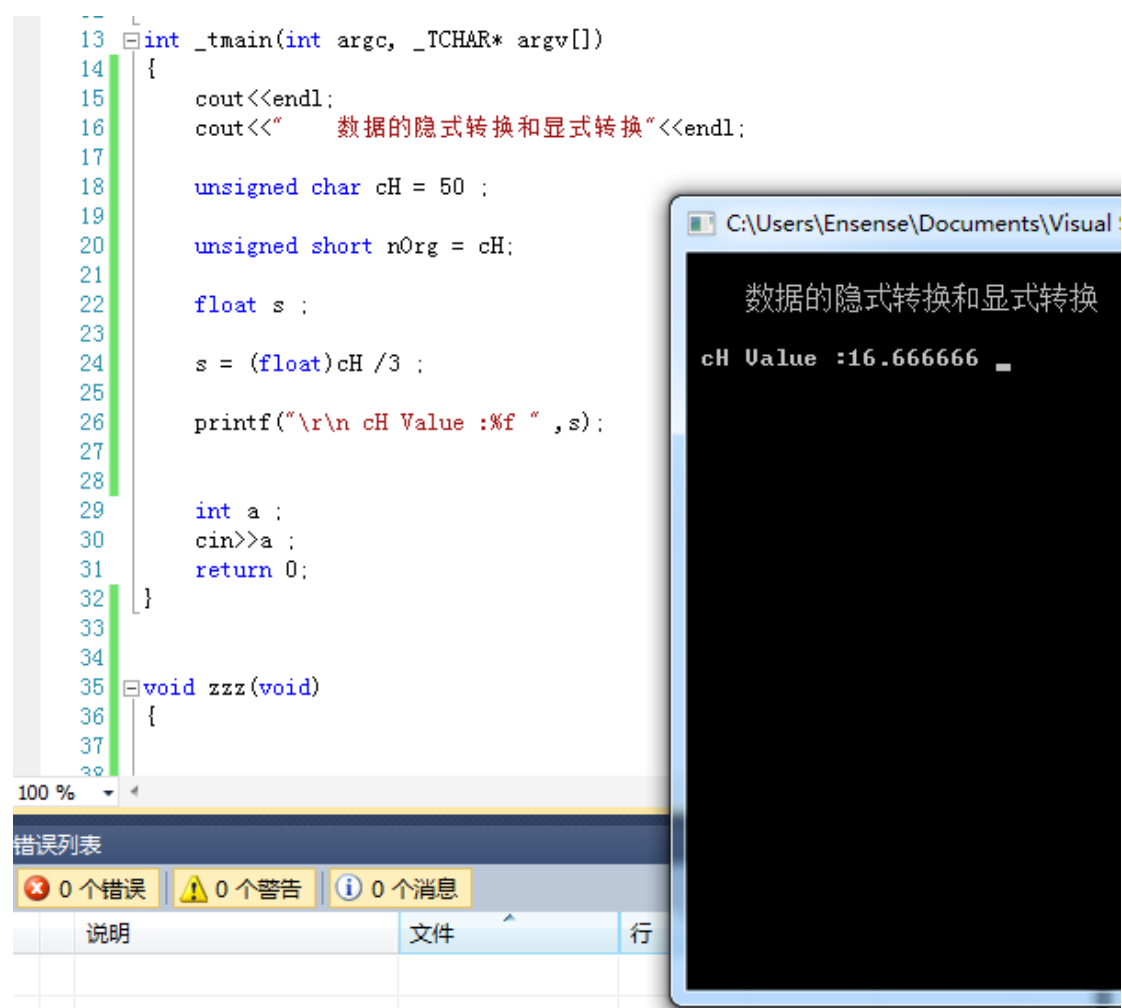
C:\Users\Ensense\Documents\Visual Stud

数据的隐式转换和显式转换

cH Value :16.666666

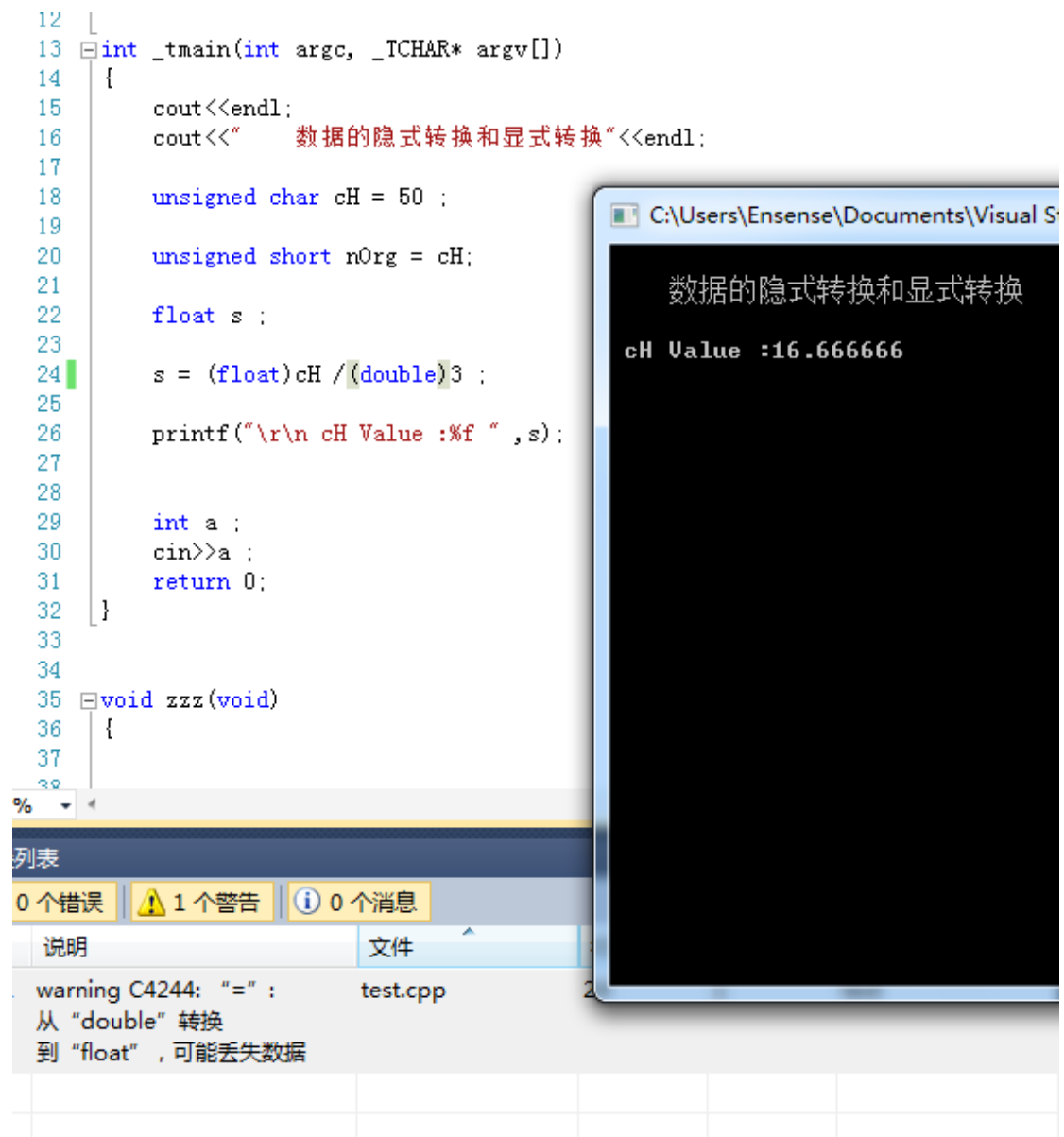
当计算中有 float 型数据参与时，会自动先把所有数据扩充到 float 型，在计算结果，结果保存为 float 型。

上图显式的把 3 转换成 float 型。最后结果正确，无警告。



当计算中有 float 型数据参与时，会自动先把所有数据扩充到 float 型，在计算结果，结果保存为 float 型。

上图显式的把 cH 转换成 float 型。最后结果正确，无警告。



当同时有 float 和 double 型数据参与计算时，数据会自动扩充到 double 再计算。结果为 double 型。所以有警告。

2.2 总结

C 语言在做计算时，对于所有整形数（int 和 char）都扩充到 int 型然后计算，此举是为了防止数据溢出。

在 C 语言中直接定义的小数数据类型（比如 3.0）是 double 型。

有 double 型参与计算时，会把其他数也扩充为 double。

没有 double 类型，有 float 型数据参与计算时，会把所有数据扩充为 float。

