

- 6.7. 引用(C++新增)
- 6.7.1. 引用的基本概念

含义:变量的别名

声明: int a, &b=a; //a和b表示同一个变量

★ 引用不分配单独的空间(指针变量有单独的空间)

变量的定义: 分配空间 变量的声明: 不分配空间

★ 引用需在声明时进行初始化,指向同类型的变量,在整个生存期内不能再指向其它变量

int a, &c=a; //正确 int b, &c=b; //错 &c=b; //错 c已是a的别名,不能再b 无论定义/赋值均不行

```
int a, &c=a, b;

c=b/b=c ⇔ a=b/b=a

int a, &c=a, b[10];

c=b[3] ⇔ a=b[3]

都正确
```

★ 不能声明引用数组和指向引用的指针,但可声明数组的引用、数组元素的引用和指向指针的引用

int &b[3]; //错误, 不能声明引用数组

int &*p; //错误,不能定义指向引用的指针

int a[5],(&b)[5]=a; //正确, 引用指向整个数组 int a[5],&b=a[3]; //正确, 引用指向数组元素 int *a, *&b=a; //正确, 指向指针的引用



- 6.7. 引用(C++新增)
- 6.7.1. 引用的基本概念

含义:变量的别名

声明: int a, &b=a; //a和b表示同一个变量

- ★ 引用不分配单独的空间(指针变量有单独的空间)
- ★ 引用需在声明时进行初始化,指向同类型的简单变量,在整个生存期内不能再指向其它变量
- ★ 不能声明指向数组的引用、引用数组和指向引用的指针,但可声明数组元素的引用和指向指针的引用
- ★ &的理解

定义语句新变量名前: 引用声明符

int a, &b=a;

其它(定义语句已定义变量名, 执行语句): 取地址运算符

int a, *p=&a;

p=&a;

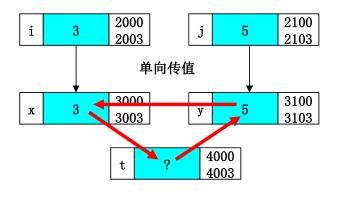
引用的简单使用: 出现在普通变量可出现的任何位置

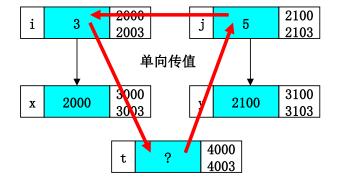
```
//例: 简单变量的引用
#include <iostream>
using namespace std;
int main()
{ int a=10, &b=a;
    a=a*a;
    cout << a << " " << b << endl; 100 100
    b=b/5;
    cout << a << " " << b << endl; 20 20
    return 0;
}
```

- 6.7. 引用(C++新增)
- 6.7.1. 引用的基本概念
- 6.7.2. 引用作函数参数

例:两数交换

```
void swap(int x, int x)
                         直接传值
{ int t;
                           错误
   t = x;
   x = y;
   y = t;
int main()
{ int i=3, j=5;
   swap(i, j);
   cout << i << " " << j << endl;
   return 0;
void swap(int *x, int *y)
                          传地址
{ int t;
                           正确
   t = *x;
   *_X = *_Y;
   *y = t:
int main()
{ int i=3, j=5;
   swap(&i, &j);
   cout << i << " " << j << endl;
   return 0;
```

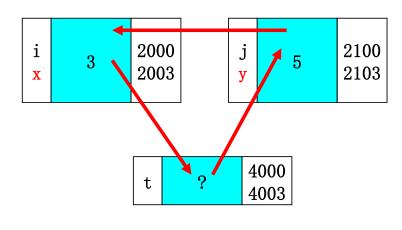








- 6.7. 引用(C++新增)
- 6.7.1. 引用的基本概念
- 6.7.2. 引用作函数参数
- 例:两数交换



- ★ 形参是引用时,不需要声明时初始化,调用时,形参不分配空间,只是当作实参的别名,因此对形参的访问就是对实参的访问
- ★ 实参虽然是变量名,但传递给形参的实际上是实参的地址,同时形参不单独分配空间,只是虚实结合(地址传递方式)



- 6.7. 引用(C++新增)
- 6.7.1. 引用的基本概念
- 6.7.2. 引用作函数参数
- ★ 实参虽然是变量名,但传递给形参的实际上是实参的地址,同时形参不单独分配空间,只是虚实结合(地址传递方式)
 - C++函数参数传递的两种方式
 - ◆ 传值:单向传值,实形参分占不同空间

```
void fun(int x)
{ …
}

形参的变化
不影响实参

int main()
{ int k = 10;
   fun(k);
}
```

```
void fun(int *x)
{ …
}

可通过形参间接
访问实参,但本质
仍是单向传值

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

◆ 传址:实形参重合,对形参的访问就是对实参的访问

```
void fun(int *x)
{ ...
}

对形参数组
的修改影响
实参数组

int main()
{ int k[10] = {...};
fun(k);
}
```

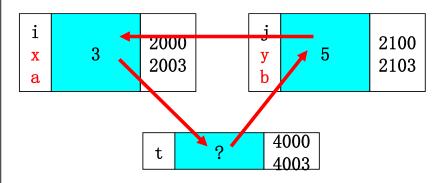
```
void fun(int &x)
{ …
}

对形参的访问
就是对实参的访问
int main()
{ int k=10;
fun(k);
}
```



- 6.7. 引用(C++新增)
- 6.7.1. 引用的基本概念
- 6.7.2. 引用作函数参数
- ★ 形参是引用时,不需要声明时初始化,调用时,形参不分配空间,只是当作实参的别名,因此对形参的访问就是对实参的访问
- ★ 实参虽然是变量名,但传递给形参的实际上是实参的地址,同时形参不单独分配空间,只是虚实结合(地址传递方式)
- ★ 引用允许传递

```
void swap1(int &a, int &b)
{    int t;
        t = a;
        a = b;
        b = t;
}
void swap(int &x, int &y)
{
    swap1(x, y);
}
int main()
{
    int i=3, j=5;
    swap(i, j);
}
```





- 6.7. 引用(C++新增)
- 6.7.2. 引用作函数参数
- ★ 形参是引用时,不需要声明时初始化,调用时,形参不分配空间,只是当作实参的别名,因此对形参的访问就是对实参的访问
- ★ 实参虽然是变量名,但传递给形参的实际上是实参的地址,同时形参不单独分配空间,只是虚实结合(地址传递方式)
- ★ 引用允许传递
- ★ 当引用做函数形参时,实参不允许是常量/表达式,否则编译错误(形参为const引用时实参可为常量/表达式)

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

void fun(int x)
{
   cout << x << endl;
}
int main()
{ int i=10;
   fun(i); //正确
   fun(15); //正确
   return 0;
}
```

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

void fun(int &x)
{
   cout << x << endl;
}
int main()
{   int i=10;
   fun(i); //正确
   fun(15); //编译错
   return 0;
}
```

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

void fun(const int &x)
{
   cout << x << endl;
}
int main()
{ int i=10;
   fun(i); //正确
   fun(15); //正确
   return 0;
}
```

error C2664: "void fun(int &)": 无法将参数 1 从"int"转换为"int &"

问:系统认为错误的原因是什么?是为了防止什么隐患出现?

答: 引用做函数形参,表示形参是可读可写的,而实参是常量/表达式,不可写

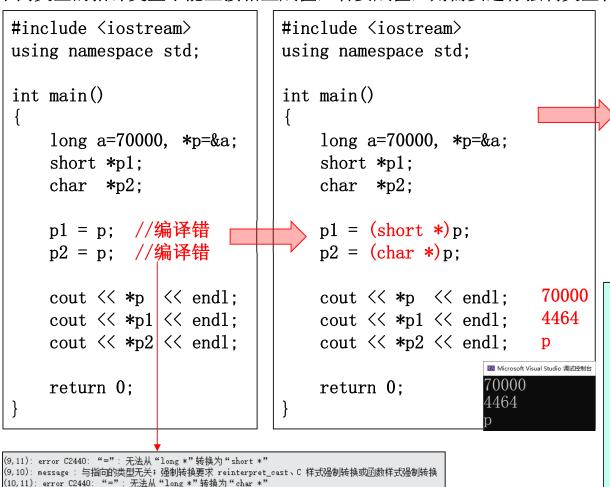
=> 形参是实参别名,但得到的权限(读/写)大于原始权限(只读)



- 6.7. 引用(C++新增)
- 6.7.3. 关于引用的特别说明
- ★ 引用在需要<mark>改变实参值</mark>的函数调用时比指针方式更容易理解,形式也更简洁,不容易出错
- ★ 引用不能完全替代指针(可以将指针理解为if-else,引用理解为switch-case)
- ★ 引用是C++新增的,纯C的编译器不支持,后续工作学习中接触的大量底层代码仍是由C编写的,此时无法使用引用 (VS/Dev都是C++编译器,兼容编译纯C,以后缀名.c/.cpp来区分如何编译)
- ★ 对于计算机底层而言, 仍需要透彻理解指针!!!

6.8. 不同基类型指针的相互赋值





(10,10): message : 与指向的类型无关;强制转换要求 reinterpret_cast、C 样式强制转换或函数样式强制转换

70000 = 00000000 00000001 00010001 01110000

 p
 2000 取2000~2003这4字节

 p1
 2000 取2000~2001这2字节

 p2
 2000 取2000这一字节

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

★ 字节序简介

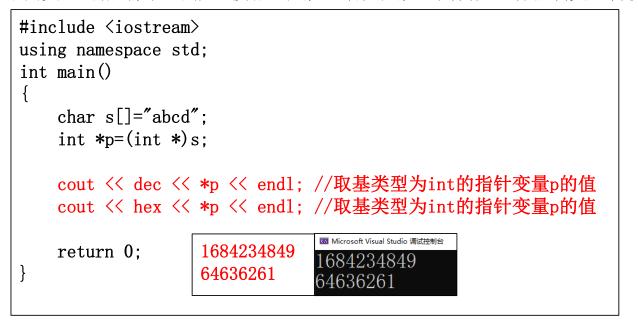
在计算机中,存储器的基本单位是字节,因此对于多字节数据类型, 会涉及到字节的顺序问题,即<mark>字节序</mark>,主要分为两种:

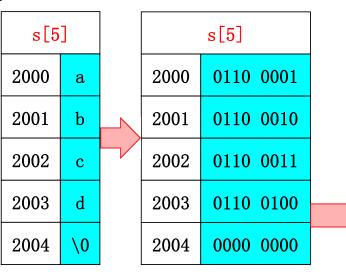
Big Endian / 大字序 / 大字节序 / 大端字节序 数据的高字节存储在低地址,低字节存储在高地址(PowerPC) Little Endian / 小字序 / 小字节序 / 小端字节序 数据的低字节存储在低地址,高字节存放在高地址(x86)

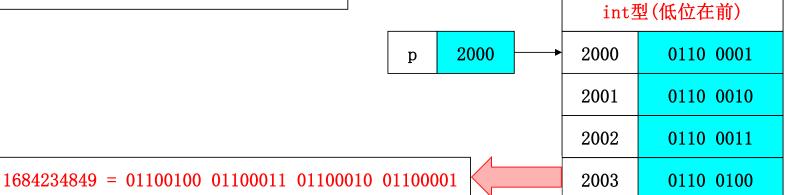
- 字节序与CPU硬件有关,与操作系统、编译器等无关
- 部分CPU为大小字序可配置
- 目前的主流选择为小字序(因为更适合内部处理)

6.8. 不同基类型指针的相互赋值

★ 不同类型的指针变量不能直接相互赋值,若要赋值,则需要进行强制类型转换







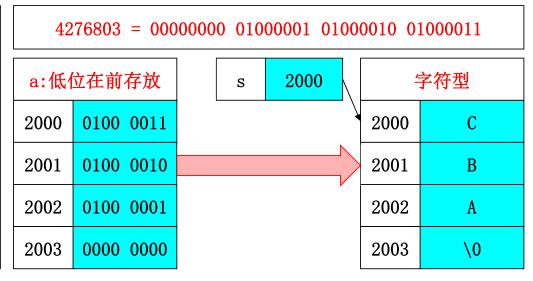


6.8. 不同基类型指针的相互赋值

★ 不同类型的指针变量不能直接相互赋值,若要赋值,则需要进行强制类型转换

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

int main()
{
    int a=4276803; //0x414243
    char *s=(char *)&a;
    cout << s << '#' << endl; //串方式输出
    return 0;
}
```



```
int main()
{
    int a= 0x41424344;
    char *s=(char *)&a;
    cout << s << '#' << endl; //串方式输出
    return 0;
}
```

6.8. 不同基类型指针的相互赋值

★ 不同类型的指针变量不能直接相互赋值,若要赋值,则需要进行强制类型转换

问题:

- 1、如何知道double型数据的存储(三段制)?
- 2、同一个数据,如何知道float和double的存储差异?
- 3、如何知道某种编译器下两个相邻的变量间隔几字节?
- 4、如何知道数组和某相邻变量间隔几字节?
- 5、如何知道某函数的原始执行代码?
- 6, ...

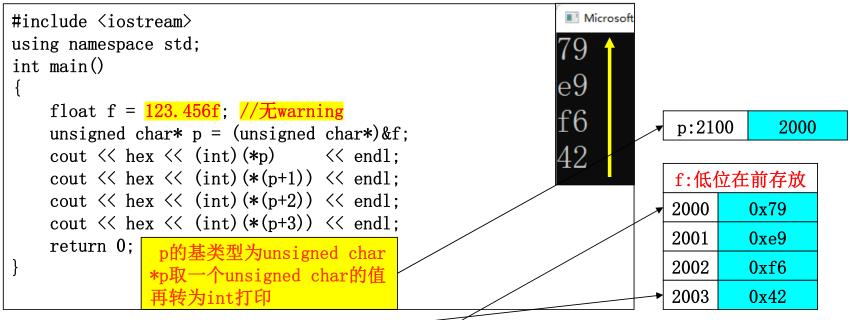


§.基础知识题 - 浮点数机内存储格式(IEEE 754)理解



第02模块的PPT作业,看懂原理

注意:除了对黄底红字的具体值进行改动外,其余部分不要做改动,也暂时不需要弄懂为什么(需要第6章的知识才能弄懂)





§.基础知识题 - 浮点数机内存储格式(IEEE 754)理解



改写

```
Microsoft
#include <iostream>
using namespace std;
int main()
                                                           e9
    float f = \frac{123.456}{}; //无warning
                                                           f6
    int *p = (int *)&f:
    cout << hex << *p << endl;
    short *q = (short *)&f;
    cout << hex << *(q+1) << endl:
                                               ■ Microsoft Visual Studio 调试控制台
    return 0:
                                              42f6e979
                                              42f6
//注: 忽略本题出现的warning
```

p:210	00	2000			
q:2200		2000			
£./Æ/	分 左	治方法			
7 1 1 1	f:低位在前存放				
2000	0x79				
2001	0xe9				
2002		0xf6			

0x42

§ 2. 基础知识

问题2: 现在看到一个以1开始的二进制整数(1***),应该是无符号的正数, 还是有符号的负数?

答:错误的问题!!!

不是看到这个二进制整数后,再去判断该数是什么类型, 而是要先确定以什么类型去看待这个数,再去确定该数的值 即:必须通过类型确定值,而不是通过值确定类型!!! 问:已知内存中某字节的值是0x42,就说明这个字节的值是char型的'B'吗?

答:错误的问题!不是看到值再去判断类型,必须确定该字节的数据类型后,才能<mark>唯一确定</mark>该字节的值(float的最高8bit / int型的最高8bit / short的高8bit / char型B)

2003

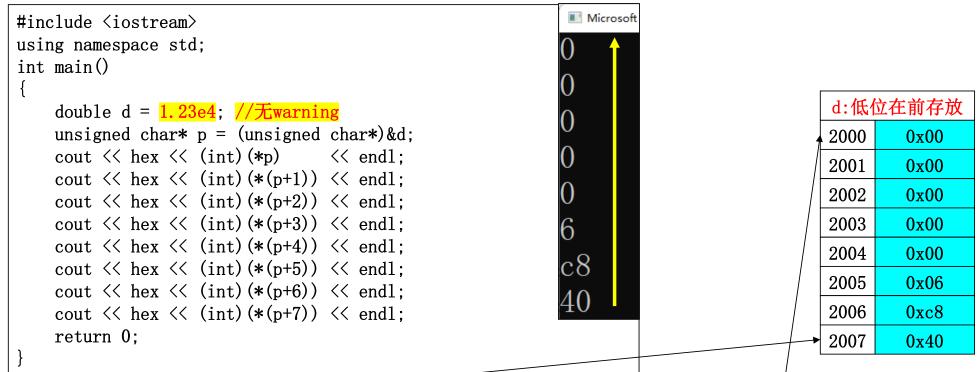
等价问题

§.基础知识题 - 浮点数机内存储格式(IEEE 754)理解



基础知识:用于看懂double型数据的内部存储格式的程序如下: 第02模块的PPT作业,看懂原理

注意:除了对黄底红字的具体值进行改动外,其余部分不要做改动,也暂时不需要弄懂为什么(需要第6章的知识才能弄懂)



11位指数

52位尾数

阶码: 100 0110 0 = 140 - 127 = 13

1. $50146484375 \times 2^{13} = 12300 = 1.23e4$

尾数: 100 0000 0011 0000 0000 0000 = 0.50146484375

■ 科学 ⑤ 1.50146484375 × 2 ^ 13 = 12,300



2002

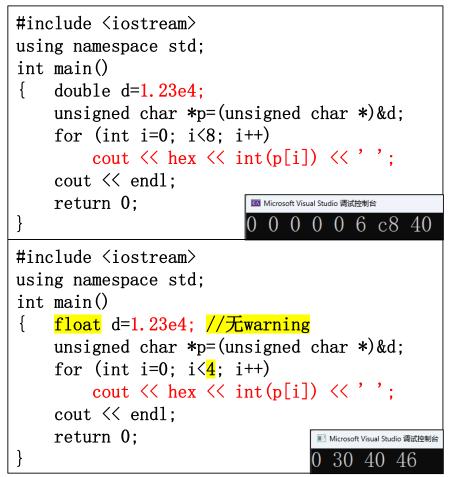
2003

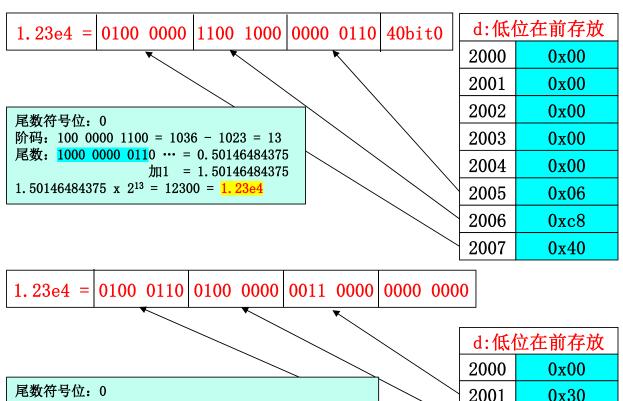
0x40

0x46

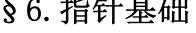
6.8. 不同基类型指针的相互赋值

★ 不同类型的指针变量不能直接相互赋值,若要赋值,则需要进行强制类型转换





b01 = 1,50146484375



- 6.8. 不同基类型指针的相互赋值
- ★ 不同类型的指针变量不能直接相互赋值,若要赋值,则需要进行强制类型转换

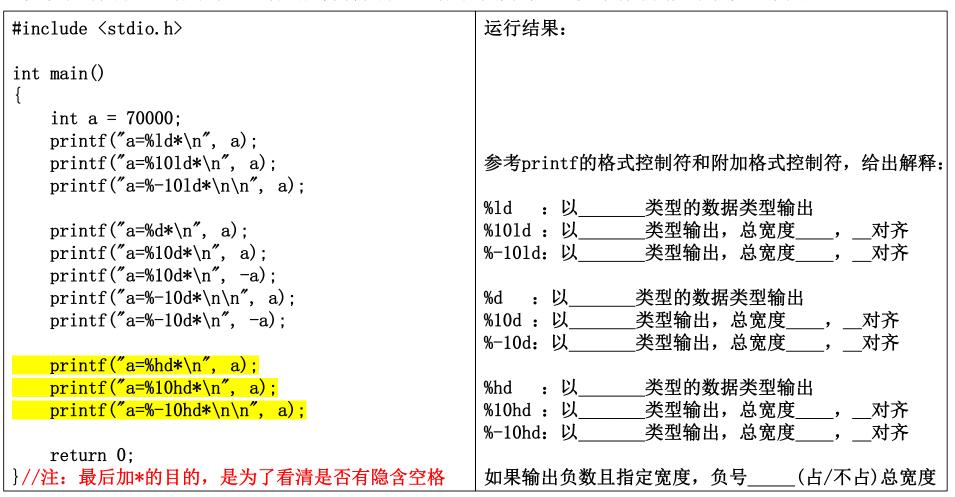
例: 验证浮点数表示是否存在误差的样例程序

```
#include <iostream>
using namespace std;
int main()
     float d1 = 1.23e4; //\pmwarning
     cout \langle\langle (d1==1.23e4) \langle\langle end1;
                                                                            #include <iostream>
                                                                            using namespace std:
     float d2 = 1.2; //有warning-
                                                                           ∃int main()
     cout \langle\langle (d2==1.2) \langle\langle end1;
                                                                                double \mathbf{d} = 1.2;
                                           ■ Microsoft Visual Studio 调试控制台
                                                                                unsigned char* p = (unsigned char*)&d;
     return 0;
                                                                                for (int i = 0; i < 8; i++)
                                                                                    cout << hex << int(p[i]) << ' ';
                                                                                cout << end1:
                                                                                                 ■ Microsoft Visual Studio 调试控制台
                                                                                return 0;
                                                                                                 33 33 33 33 33 33 f3 3f
```

产生warning的原因: 52位尾数赋给23位尾数时,丢弃的不是全0



- 1. 格式化输出函数printf的基本理解
 - E. 观察下列程序的运行结果,回答问题并将程序的运行结果截图贴上(如果有错则贴错误信息截图)







- 1. 格式化输出函数printf的基本理解
 - E. 观察下列程序的运行结果,回答问题并将程序的运行结果截图贴上(如果有错则贴错误信息截图)

```
#include <stdio.h>

int main()
{
    int a = 70000;

    printf("a=%hd*\n", a);
    printf("a=%10hd*\n", a);
    printf("a=%-10hd*\n\n", a);

    return 0;
}//注:最后加*的目的,是为了看清是否有隐含空格

| Microsoft Visual Studio 開始技术 | a=4464*
    a=4464*
    a=4464*
    a=4464*
```

70000 = 00000000 00000001 00010001 01110000

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

结论:在C方式中,如果%格式符与实际数据类型不一致, 以格式符为准



- 2. 格式化输入函数scanf的基本理解
 - F. 观察下列程序的运行结果,回答问题并将程序的运行结果截图贴上(如果有错则贴错误信息截图)

```
#define CRT SECURE NO WARNINGS
                              #define CRT SECURE NO WARNINGS
                                                             #define CRT SECURE NO WARNINGS
#include <stdio.h>
                              #include <stdio.h>
                                                             #include <stdio.h>
int main()
                              int main()
                                                             int main()
   short c;
                                                                 short c:
                                  int c:
   scanf ("%d", &c);
                                  scanf ("%hd", &c);
                                                                 scanf ("%hd", &c);
   printf("c=\%hd\n", c);
                                  printf("c=\%d\n", c);
                                                                printf("c=%hd\n", c);
   return 0;
                                  return 0;
                                                                return 0;
                                                             假设键盘输入为: 10✓
假设键盘输入为: 10✓
                              假设键盘输入为: 10✓
                                                             则输出为:
则输出为:
                              则输出为:
                                                             假设键盘输入为: 70000✓
                                                             则输出为:
结论:
1、附加格式控制符h的作用是
2、如果格式控制符的数据类型和要读取的变量类型的字节大小不一致(例: 4/2字节),则
```



2. 格式化输入函数scanf的基本理解

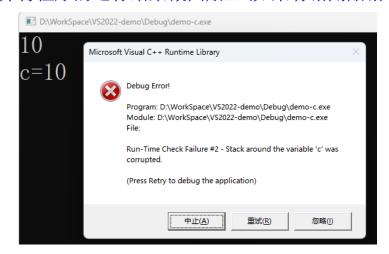
F. 观察下列程序的运行结果,回答问题并将程序的运行结果截图贴上(如果有错则贴错误信息截图)

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

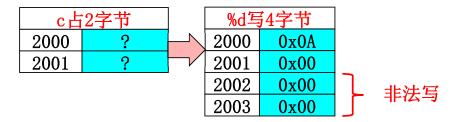
int main()
{
    short c;

    scanf("%d", &c);
    printf("c=%hd\n", c);

    return 0;
}
```

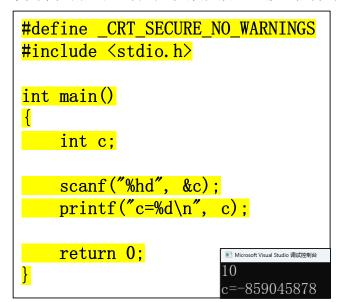


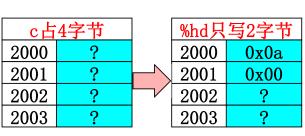
warning C4477: "scanf": 格式字符串"%d"需要类型"int *"的参数,但可变参数 1 拥有了类型"short *" message: 请考虑在格式字符串中使用"%hd"





- 2. 格式化输入函数scanf的基本理解
 - F. 观察下列程序的运行结果,回答问题并将程序的运行结果截图贴上(如果有错则贴错误信息截图)





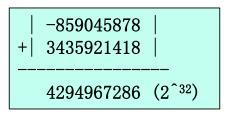
warning C4477: "scanf": 格式字符串"%hd"需要类型"short *"的参数,但可变参数 1 拥有了类型"int *"message: 请考虑在格式字符串中使用"%d"message: 请考虑在格式字符串中使用"%Id"message: 请考虑在格式字符串中使用"%I32d"

1 ON THE PROPERTY OF THE PROPE

2. 格式化输入函数scanf的基本理解

F. 观察下列程序的运行结果,回答问题并将程序的运行结果截图贴上(如果有错则贴错误信息截图)



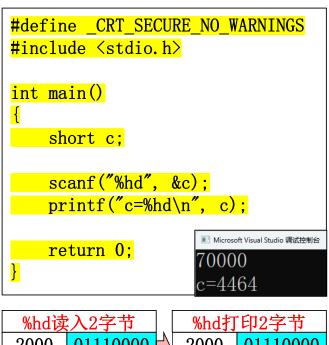


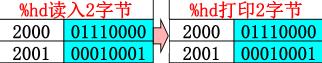
```
上一页的验证程序
#define CRT SECURE NO WARNINGS
#include <stdio.h>
int main()
    int c:
    unsigned char *p = (unsigned char *)&c;
    printf("%x %x %x %x\n", *p, *(p+1), *(p+2), *(p+3));
                                                           ■ Microsoft Visual Studio 调试控制台
   scanf ("%hd", &c);
                                                          cc cc cc cc
   printf("c=%d\n", c);
                                                          10 輸入
    printf("%x %x %x %x\n", *p, *(p+1), *(p+2), *(p+3));
                                                          c = -859045878
   return 0;
                                                          a 0 cc cc
```

c占4字节		%hd只写2字节	
2000	0xcc	2000	0x0a
2001	0xcc	2001	0x00
2002	0xcc	2002	0xcc
2003	0xcc	2003	0xcc



- 2. 格式化输入函数scanf的基本理解
 - F. 观察下列程序的运行结果,回答问题并将程序的运行结果截图贴上(如果有错则贴错误信息截图)





键盘输入的70000给c赋值时, 舍弃高16位

 $70000 = \frac{00000000 - 00000001}{00000001} 00010001 01110000$