§6. 指针基础

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. 指针基础

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;
}
```

本例无任何实用价值
1、多定义一个名称
2、两者容易混淆

§6. 指针基础

6.7. 引用(C++新增)

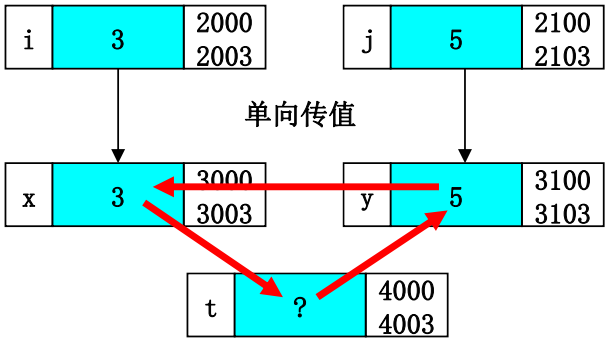
6.7.1. 引用的基本概念

6.7.2. 引用作函数参数

例：两数交换

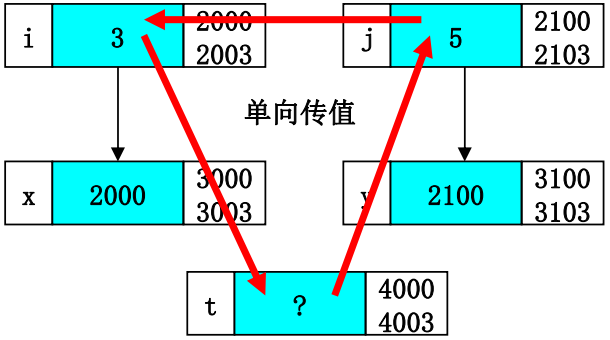
直接传值
错误

```
void swap(int x, int x)
{   int t;
    t = x;
    x = y;
    y = t;
}
int main()
{   int i=3, j=5;
    swap(i, j);
    cout << i << " " << j << endl;
    return 0;
}
```



传地址
正确

```
void swap(int *x, int *y)
{   int t;
    t  = *x;
    *x = *y;
    *y = t;
}
int main()
{   int i=3, j=5;
    swap(&i, &j);
    cout << i << " " << j << endl;
    return 0;
}
```



§6. 指针基础

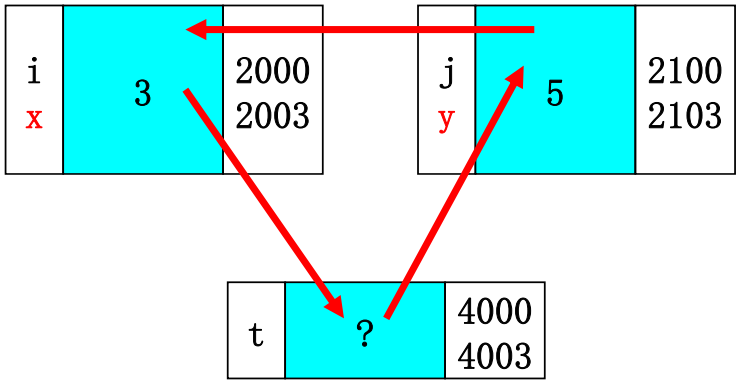
6.7. 引用(C++新增)

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 &y)   引用做形参
{   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;
}
```



★ 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. 指针基础

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)
{ …
}

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

形参的变化不影响实参

```
void fun(int *x)
{ …
}

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

可通过形参间接访问实参，但本质仍是单向传值

  - 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)
{ …
}

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

对形参数组的修改影响实参数组

```
void fun(int &x)
{ …
}

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

对形参的访问就是对实参的访问

§6. 指针基础

6.7. 引用(C++新增)

6.7.1. 引用的基本概念

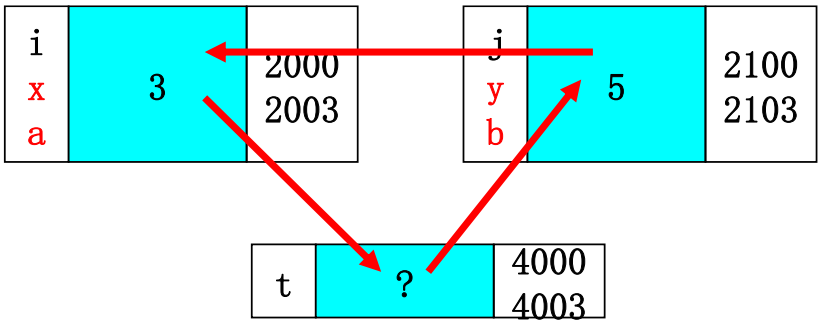
6.7.2. 引用作函数参数

★ 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. 指针基础

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;
}
```

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

```
#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;
}
```

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. 指针基础

6.7. 引用(C++新增)

6.7.3. 关于引用的特别说明

★ 引用在需要改变实参值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. 指针基础

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()
{
    long a=70000, *p=&a;
    short *p1;
    char  *p2;

    p1 = p;  //编译错
    p2 = p;  //编译错

    cout << *p  << endl;
    cout << *p1 << endl;
    cout << *p2 << endl;

    return 0;
}
```

```
(9,11): error C2440: "=": 无法从"long *"转换为"short *"
(9,10): 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
(10,11): error C2440: "=": 无法从"long *"转换为"char *"
(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
```

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

int main()
{
    long a=70000, *p=&a;
    short *p1;
    char  *p2;

    p1 = (short *)p;
    p2 = (char *)p;

    cout << *p  << endl;  70000
    cout << *p1 << endl;  4464
    cout << *p2 << endl;  p

    return 0;
}
```

Microsoft Visual Studio 调试控制台

```
70000
4464
p
```

70000 = 00000000 00000001 00010001 01110000

p	2000

p1	2000

p2	2000

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

§6. 指针基础

6.8. 不同基类型指针的相互赋值

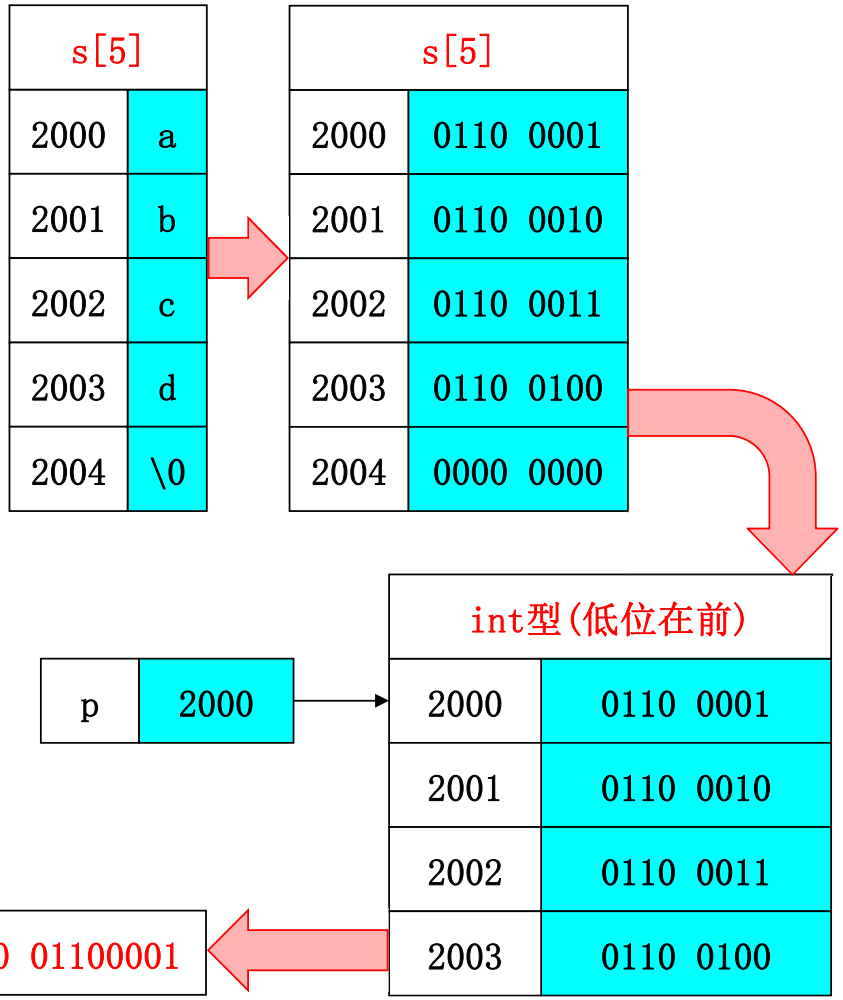
★ 不同类型的指针变量不能直接相互赋值，若要赋值，则需要进行强制类型转换

```
#include <iostream>
using namespace std;
int main()
{
    char s[]="abcd";
    int *p=(int *)s;

    cout << dec << *p << endl; //取基类型为int的指针变量p的值
    cout << hex << *p << endl; //取基类型为int的指针变量p的值

    return 0;
}
```

```
1684234849
64636261
```

Microsoft Visual Studio 调试控制台

```
1684234849
64636261
```



§6. 指针基础

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

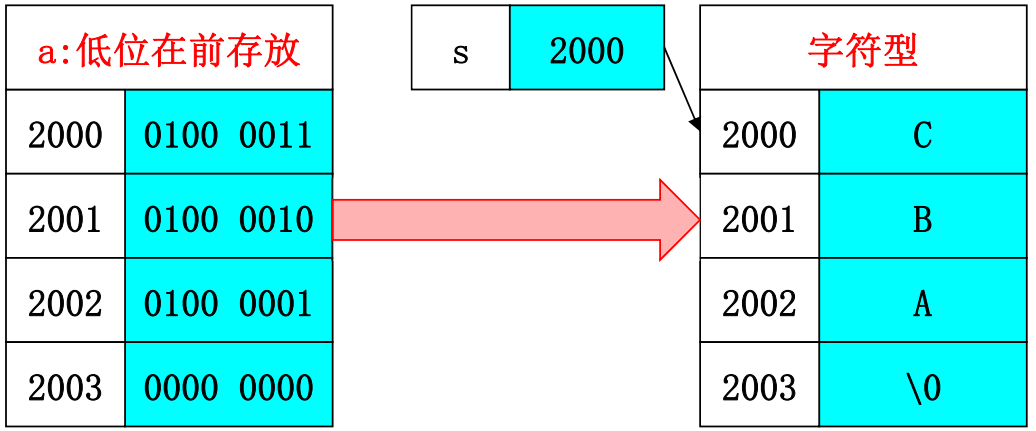
```
#include <iostream>
using namespace std;

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

    return 0;
}
```

CBA

Microsoft Visual Studio 调试控制台

CBA#

6276803 = 00000000 01000001 01000010 01000011




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

    return 0;
}
```

Microsoft Visual Studio 调试控制台

DCBA烫烫萌\?默#

为什么?

§6. 指针基础

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

基础知识：用于看懂float型数据的内部存储格式的程序如下：

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）

```
#include <iostream>
using namespace std;
int main()
{
    float f = 123.456f;
    unsigned char* p = (unsigned char*)&f;
    cout << hex << (int)(*p)     << endl;
    cout << hex << (int)(*(p+1)) << endl;
    cout << hex << (int)(*(p+2)) << endl;
    cout << hex << (int)(*(p+3)) << endl;
    return 0;
}
```

Microsoft

79
e9
f6
42

d:低位在前存放	
2000	0x79
2001	0xe9
2002	0xf6
2003	0x42

上例解读：单精度浮点数123.456，在内存中占四个字节，四个字节的值依次为0x42 0xf6 0xe9 0x79（按打印顺序逆向取）

转换为32bit则为：0100 0010 | 1111 0110 | 1110 1001 | 0111 1001

8位指数　　23位尾数

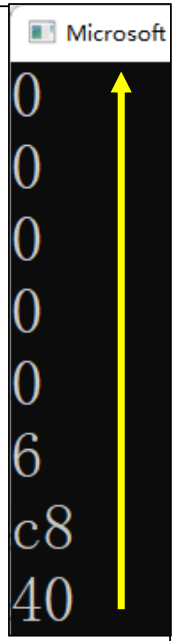
§.基础知识题 - 浮点数机内存储格式(IEEE 754)理解

第02模块的PPT作业，看懂原理

基础知识：用于看懂double型数据的内部存储格式的程序如下：

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）

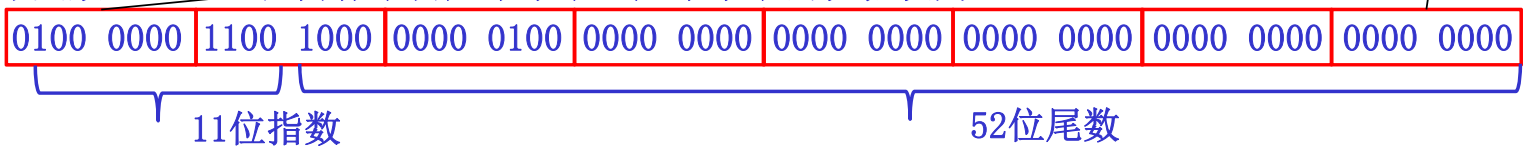
```
#include <iostream>
using namespace std;
int main()
{
    double d = 1.23e4;
    unsigned char* p = (unsigned char*)&d;
    cout << hex << (int)(*p)     << endl;
    cout << hex << (int)(*(p+1)) << endl;
    cout << hex << (int)(*(p+2)) << endl;
    cout << hex << (int)(*(p+3)) << endl;
    cout << hex << (int)(*(p+4)) << endl;
    cout << hex << (int)(*(p+5)) << endl;
    cout << hex << (int)(*(p+6)) << endl;
    cout << hex << (int)(*(p+7)) << endl;
    return 0;
}
```



d:低位在前存放	
2000	0x00
2001	0x00
2002	0x00
2003	0x00
2004	0x00
2005	0x06
2006	0xc8
2007	0x40

上例解读：双精度浮点数1.23e4，在内存中占八个字节，八个字节的值依次为0x40 0xc8 0x06 0x00 0x00 0x00 0x00 0x00(逆向)

转换为64bit则为：



§6. 指针基础

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()
{   double d=1.23e4;
    unsigned char *p=(unsigned char *)&d;
    for (int i=0; i<8; i++)
        cout << hex << int(p[i]) << ' ';
    cout << endl;
    return 0;
}
```

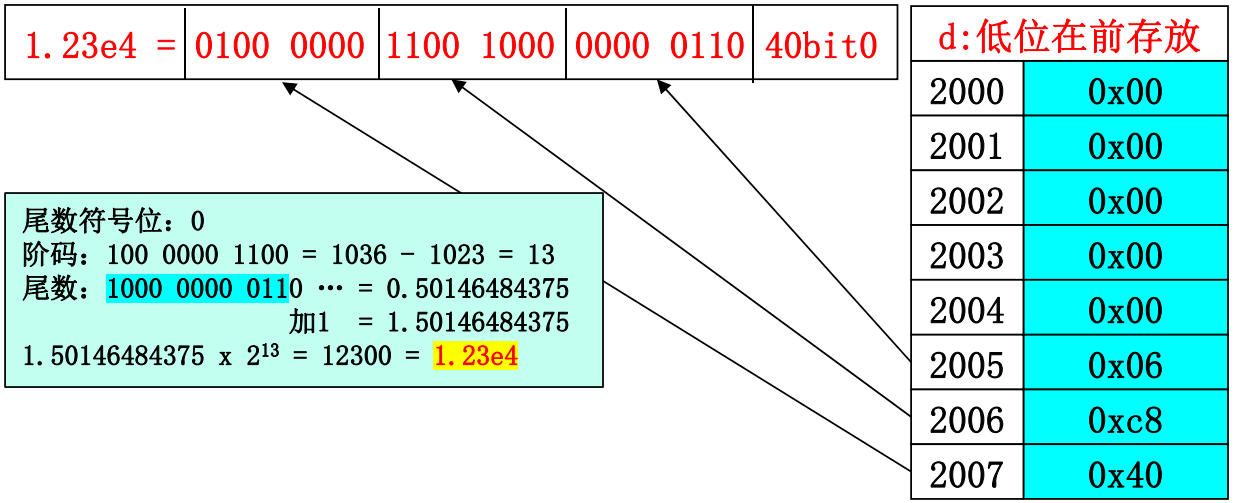
Microsoft Visual Studio 调试控制台

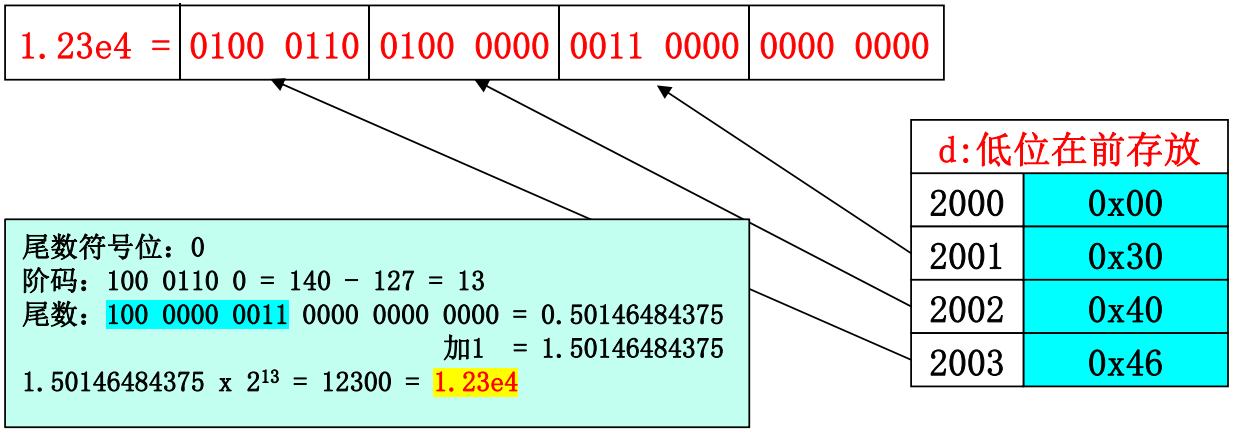
```
0 0 0 0 0 6 c8 40
```

```
#include <iostream>
using namespace std;
int main()
{   float d=1.23e4; //无warning
    unsigned char *p=(unsigned char *)&d;
    for (int i=0; i<4; i++)
        cout << hex << int(p[i]) << ' ';
    cout << endl;
    return 0;
}
```

Microsoft Visual Studio 调试控制台

```
0 30 40 46
```





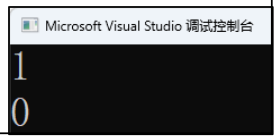
§6. 指针基础

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; //无warning
    cout << (d1==1.23e4) << endl;

    float d2 = 1.2; //有warning
    cout << (d2==1.2) << endl;

    return 0;
}
```



§.基础知识题 - C方式输入输出的格式化控制

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

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

    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
```

运行结果：

参考printf的格式控制符和附加格式控制符，给出解释：

%ld　：以_____类型的数据类型输出
%10ld：以_____类型输出，总宽度____，__对齐
%-10ld：以_____类型输出，总宽度____，__对齐

%d　：以_____类型的数据类型输出
%10d：以_____类型输出，总宽度____，__对齐
%-10d：以_____类型输出，总宽度____，__对齐

%hd　：以_____类型的数据类型输出
%10hd：以_____类型输出，总宽度____，__对齐
%-10hd：以_____类型输出，总宽度____，__对齐

如果输出负数且指定宽度，负号_____(占/不占)总宽度

§.基础知识题 – C方式输入输出的格式化控制

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
```

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

§.基础知识题 - C方式输入输出的格式化控制

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;`
`}`
	`#define _CRT_SECURE_NO_WARNINGS`
	`#include <stdio.h>`
	
	`int main()`
	`{`
	`  int c;`
	
	`  scanf("%hd", &c);`
	`  printf("c=%d\n", c);`
	
	`  return 0;`
	`}`
		`#define _CRT_SECURE_NO_WARNINGS`
		`#include <stdio.h>`
		
		`int main()`
		`{`
		`  short c;`
		
		`  scanf("%hd", &c);`
		`  printf("c=%hd\n", c);`
		
		`  return 0;`
		`}`

假设键盘输入为：10↙ 则输出为：	假设键盘输入为：10↙ 则输出为：	假设键盘输入为：10↙ 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_____

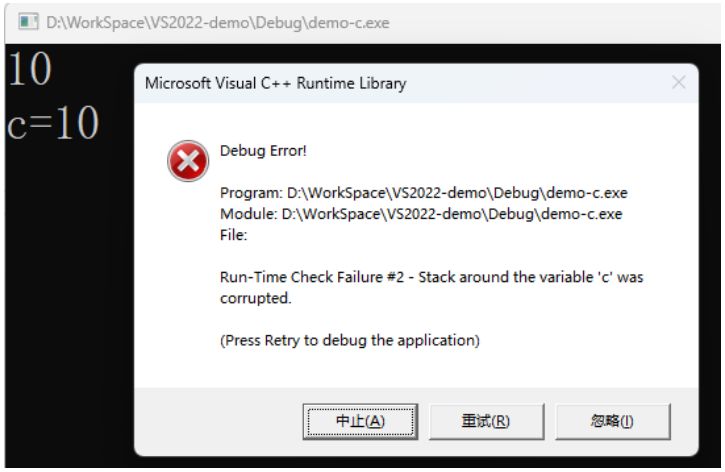
2、如果格式控制符的数据类型和要读取的变量类型的字节大小不一致（例：4/2字节），则_____

§.基础知识题 - C方式输入输出的格式化控制

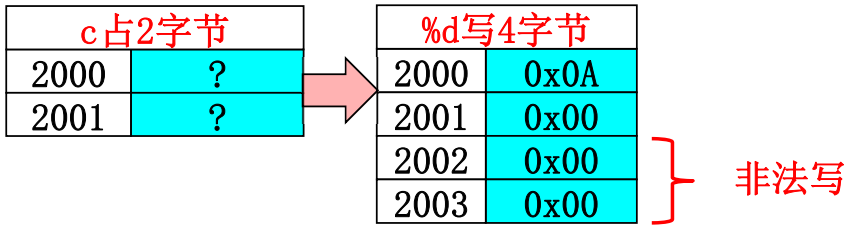
2.格式化输入函数scanf的基本理解

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”
```



§.基础知识题 - C方式输入输出的格式化控制

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()
{
    int c;

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

    return 0;
}
```

```
Microsoft Visual Studio 调试控制台
10
c=-859045878
```

c占4字节	
2000	?
2001	?
2002	?
2003	?

%hd只写2字节	
2000	0x0a
2001	0x00
2002	?
2003	?

```
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”
```

§.基础知识题 - C方式输入输出的格式化控制

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)


```
  | -859045878 |
+| 3435921418 |
----------------
   4294967286 (2^32)
```

上一页的验证程序

```c
#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));
    scanf("%hd", &c);
    printf("c=%d\n", c);
    printf("%x %x %x %x\n", *p, *(p+1), *(p+2), *(p+3));
    return 0;
}
```

c占4字节	
2000	0xcc
2001	0xcc
2002	0xcc
2003	0xcc

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

§.基础知识题 – C方式输入输出的格式化控制

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

    return 0;
}
```

```
Microsoft Visual Studio 调试控制台
70000
c=4464
```

%hd读入2字节	
2000	01110000
2001	00010001

%hd打印2字节	
2000	01110000
2001	00010001

键盘输入的70000给c赋值时，舍弃高16位

70000 = ~~00000000 00000001~~ 00010001 01110000