第2章 C运算符和表达式

——赋值中的自动类型转换

（数值溢出问题）

本节要讨论的主要问题

- 在不同类型的数据间赋值是安全的吗？
- 为什么会出现数值溢出？何为数值溢出？数值溢出的危害是什么？

自动类型转换

- 问题：赋值操作中，何时发生自动类型转换？
  - 左侧（目标侧）与右侧类型不一致
- 问题：自动类型转换的规则是什么？

变量 = 表达式;

类型2 ← 类型1

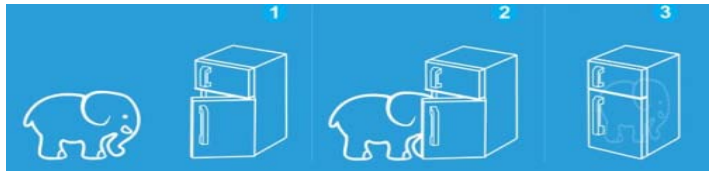
自动类型转换

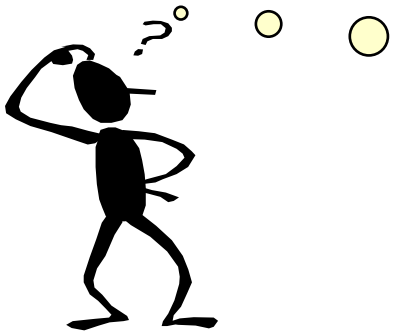
- 问题：在不同类型数据间赋值，是安全的吗？
  - 取值范围大的类型→取值范围小的类型，通常是不安全的
    - 数值溢出（Overflow）

数据类型	所占字节数	取值范围
char	1	-128~127
unsigned char	1	0~255
short int	2	-32768~32767
unsigned short int	2	0~65535
int, long	4	-2147483648~2147483647
unsigned int, unsigned long	4	0~4294967295
float	4	-3.4×10^{38}~3.4×10^{38}
double	8	-1.7×10^{308}~1.7×10^{308}

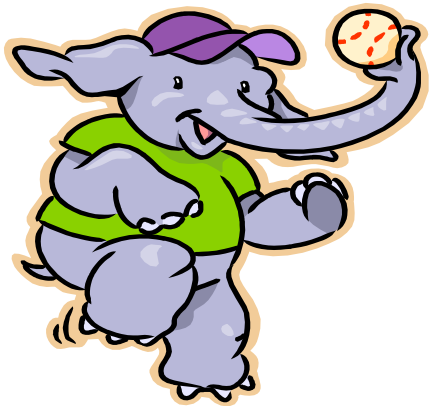
数值溢出

- 问题：何为数值溢出？为什么会发生数值溢出？
  - 任何类型都只能用有限的位数来存储数据，表数范围有限
  - 向变量赋的值超出了其类型的表数范围
  - 1996年，阿丽亚娜火箭发射失败
    - 将浮点数转换成整数，发生溢出





小蛇能吞下
大象吗？



数值溢出

- 生活中数值溢出的例子
  - 20世纪末爆发的千年虫问题
    - 在99年存钱，到01年取出，该怎样计算利息呢？
    - 第一代身份证号码中的出生年——如何区分百岁老人和婴儿？

整数的数值溢出

```
#include  <stdio.h>
int main()
{
   long a;
   a = 200*300*400*500;
   printf("%ld\n", a);
   return 0;
}
```

Visual C++6.0下给出的警告

```
integral constant overflow
```

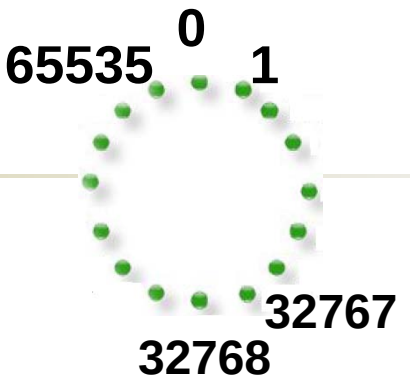
Code::Blocks下给出的警告

```
integer overflow in expression
```

```
-884901888
```

运算结果超出了long所能表示的数的上界，
进位到达了最前面的符号位（0→1）

整数的数值溢出



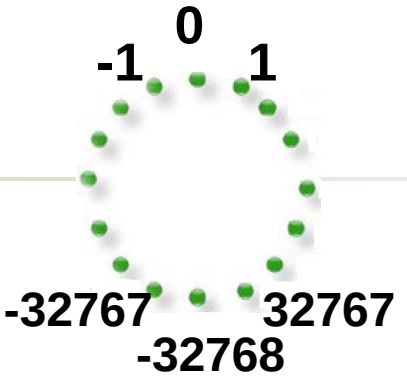
- 上溢出
  - |一个数值运算结果| > |类型能表示的最大数|
  - 进位超过最高位而发生进位丢失
  - 或进位到达最高位而改变符号位

a(65535) 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

a+1(0) 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

无符号短整型（最高位是数据位）		有符号短整型（最高位是符号位）	
二进制补码	十进制	二进制补码	十进制
00000000 00000000	0	00000000 00000000	0
00000000 00000001	1	00000000 00000001	1
00000000 00000010	2	00000000 00000010	2
00000000 00000011	3	00000000 00000011	3
…		…	
01111111 11111111	32767	01111111 11111111	32767
10000000 00000000	32768	10000000 00000000	-32768
10000000 00000001	32769	10000000 00000001	-32767
…		…	
11111111 11111110	65534	11111111 11111110	-2
11111111 11111111	65535	11111111 11111111	-1

整数的数值溢出



- 上溢出
  - |一个数值运算结果| > |类型能表示的最大数|
  - 进位超过最高位而发生进位丢失
  - 或进位到达最高位而改变符号位

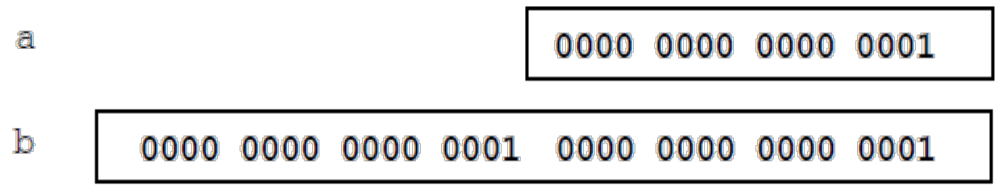
a(32767)	0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
a+1(32768)	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
a+1(-32768)	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

无符号短整型（最高位是数据位）		有符号短整型（最高位是符号位）	
二进制补码	十进制	二进制补码	十进制
00000000 00000000	0	00000000 00000000	0
00000000 00000001	1	00000000 00000001	1
00000000 00000010	2	00000000 00000010	2
00000000 00000011	3	00000000 00000011	3
…		…	
01111111 11111111	32767	01111111 11111111	32767
10000000 00000000	32768	10000000 00000000	-32768
10000000 00000001	32769	10000000 00000001	-32767
…		…	
11111111 11111110	65534	11111111 11111110	-2
11111111 11111111	65535	11111111 11111111	-1

整数的数值溢出

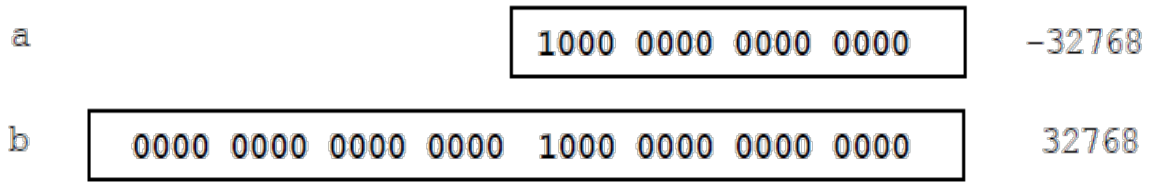
```
#include  <stdio.h>
int main()
{
   short a;
   int b = 65537;
   a = b;
   printf("%hd,%d\n", a, b);
   return 0;
}
```

1,65537




```
#include  <stdio.h>
int main()
{
   short a;
   int b = 32768;
   a = b;
   printf("%hd,%d\n", a, b);
   return 0;
}
```

-32768,32768



整数的数值溢出

```
#include  <stdio.h>
int main()
{
   unsigned short a = 8;
   unsigned short b = 10;
   printf("%hu\n", a - b);
   return 0;
}
```

65534

%hu，输出无符号短整数

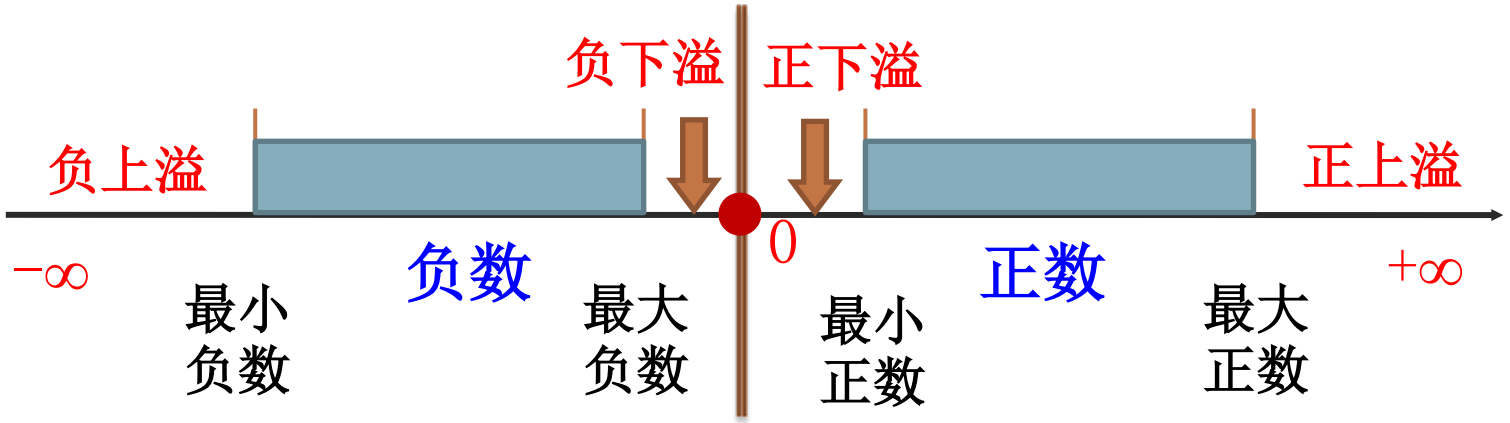
借位借到超出了最高位

a(8)	0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
b(10)	0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0
a-b(65534)	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0

对于无符号数，不能随意用a-b<0取代a<b

浮点数的数值溢出

- **上溢出：**
  - **|浮点数运算结果| > |类型能表示的最大数|**
- **下溢出**
  - **|浮点数运算结果| < |类型能表示的最小数|**
  - **此时，系统将运算结果处理成机器0**



数值溢出

- 数值溢出的危害
  - 编译器有时对它熟视无睹，只是输出奇怪的结果
  - 在平台间移植时，例如程序从高位计算机向低位计算机移植（如从64位系统移植到32位系统）时，可能出现溢出，这种问题经常被忽视

数值溢出

- 解决对策
  - 用取值范围更大的类型，有助于防止数值溢出
    - 但可能会导致存储空间的浪费
  - 了解处理问题的规模，选取恰当的数据类型
  - 同种类型在不同的平台其占字节数不尽相同
    - 不要对变量所占的内存空间字节数想当然
    - 用sizeof获得变量或者数据类型的长度

讨论题

- 为什么整数没有下溢出，而浮点数有下溢出？