



第2章 C运算符和表达式

——赋值中的自动类型转换

（精度损失问题）

哈尔滨工业大学

苏小红

sxh@hit.edu.cn

本节要讨论的主要问题

- 从取值范围小的类型向取值范围大的类型赋值时，一定都是安全的吗？
- 从高精度向低精度转换时，会损失什么信息？为什么会出现精度损失？
- 浮点数是实数的精确表示吗？为什么不能直接比较两个浮点数是否相等？为什么不能用浮点数取代整数？



自动类型转换

- 在不同类型数据间赋值时，会发生自动类型转换
 - * 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）
 - * 反之，一定都是安全的吗？
 - * 数值精度损失

自动类型转换

- 问题：从高精度向低精度转换时，会损失什么信息？
 - * 低精度的数据位数比高精度的少，容纳不下高精度的所有信息
- 舍入（Round），也称截断（Truncation）

数值精度损失

int ← float

丢失小数部分（非四舍五入）

float ← double

数值溢出或损失精度（位数超过7位时）

float ← long

整数的位数超过7位时，损失精度

精度损失实例分析

- 为什么long型的123456789不能用float型精确保存呢？
- 为什么浮点数的输出结果也不准确呢？

```
#include <stdio.h>
int main()
{
    long a = 123456789;
    float b;
    double c = 123456789123.456765;
    b = a;
    printf("%ld\n", a);
    printf("%f\n", b);
    printf("%f\n", c);
    b = c;
    printf("%f\n", b);
    return 0;
}
```

Code::Blocks下的运行结果

123456789

123456792.000000

123456789123.456770

123456790528.000000

数值精度损失

- C语言中，浮点数在内存中是以阶码和尾数的形式存储的
- ANSI C未规定3种浮点类型的长度、精度和表数范围

阶码**E**（指数部分）

尾数**M**（小数部分）

$$N = r^E \times M$$

使用更多的
位存储**阶码**

- 扩大了变量值域
（即**表数范围**），
但精度降低

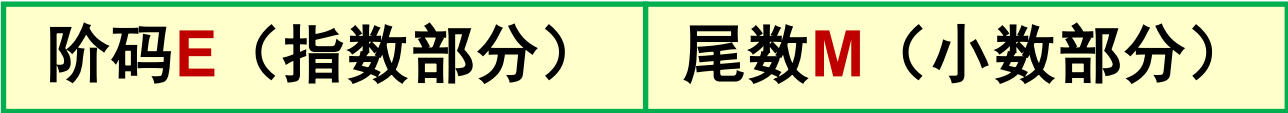
使用更多的
位存储**尾数**

- 增加了有效数字位数，
提高了**数值精度**，但
表数范围缩小

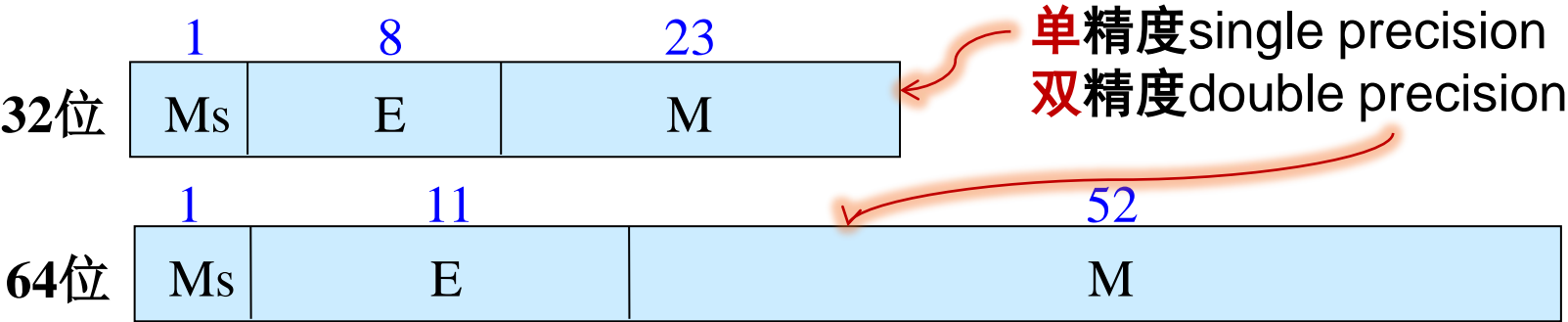


数值精度损失

- C语言中，浮点数在内存中是以阶码和尾数的形式存储的
- ANSI C未规定3种浮点类型的长度、精度和表数范围



$$N = r^E \times M$$



float	23位尾数	6~7位
double	52位尾数	16位

浮点数的标准格式 IEEE754

二进制纯小数

精度损失实例分析

- 为什么long型的123456789不能用float型精确保存呢？
- 为什么浮点数的输出结果也不准确呢？

```
#include <stdio.h>
int main()
{
    long a = 123456789;
    float b;
    double c = 123456789123.456765;
    b = a;
    printf("%ld\n", a);
    printf("%f\n", b);
    printf("%f\n", c);
    b = c;
    printf("%f\n", b);
    return 0;
}
```

有效数字(Significant Digit):
从左边第一个非0的数字起，到精确到的位数为止，其间的
所有数字

Code::Blocks

float

23位尾数

6~7位

double

52位尾数

16位

123456789

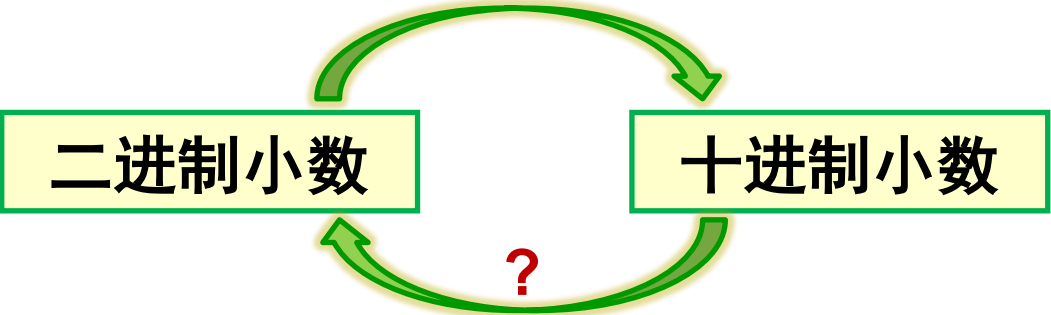
123456792.000000

123456789123.456770

123456790528.000000

数值精度损失

- **二进制**小数与**十进制**小数之间并不是一一对应的关系

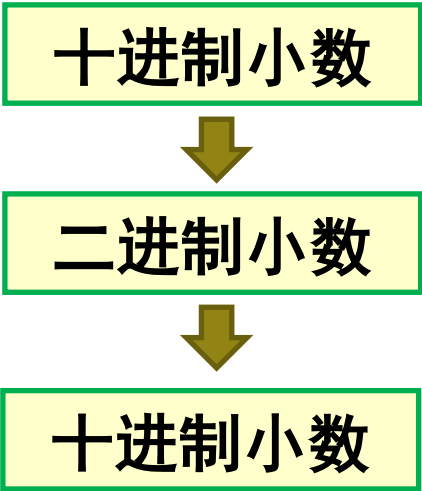


浮点数并非真正意义上的实数，只是其在某种范围内的近似，受精度限制不能直接比较相等

连续
↓

二进制小数	→对应的十进制小数
2^{-23}	0.00000011920928955078125
2^{-22}	0.0000002384185791015625
2^{-21}	0.000000476837158203125
2^{-20}	0.00000095367431640625
2^{-19}	0.0000019073486328125
.....

离散
↓



并非所有实数都能用有限位的尾数精确表示

精度损失实例分析

- 定点整数可准确表示123456789，而单精度浮点数则只能近似表示123456789

```
#include <stdio.h>
int main()
{
    long a = 123456789;
    float b;
    double c = 123456789123.456765;
    b = a;
    printf("%ld\n", a);
    printf("%f\n", b);
    printf("%f\n", c);
    b = c;
    printf("%f\n", b);
    return 0;
}
```



Code::Blocks

123456789

123456792.000000

123456789123.456770

123456790528.000000

精度损失实例分析

- 两个数量级相差很大的浮点数做加减运算时，数值小的数会受浮点数精度限制而被忽略

```
#include <stdio.h>
int main()
{
    float a, b;
    a = 1234567800;
    b = a + 20;
    printf("a=%.0f, b=%.0f\n", a, b);
    return 0;
}
```

a=1234567808, b=1234567808

```
#include <stdio.h>
int main()
{
    float a, b;
    a = 1234567800;
    b = a + 70;
    printf("a=%.0f, b=%.0f\n", a, b);
    return 0;
}
```

a=1234567808, b=1234567936

精度损失实例分析

- 两个数量级相差很大的浮点数做加减运算时，数值小的数会受浮点数精度限制而被忽略

```
#include <stdio.h>
int main()
{
    long a, b;
    a = 1234567800;
    b = a + 20;
    printf("a=%ld, b=%ld\n", a, b);
    return 0;
}
```

a=1234567800, b=1234567820

```
#include <stdio.h>
int main()
{
    double a, b;
    a = 1234567800;
    b = a + 20;
    printf("a=%.0f, b=%.0f\n", a, b);
    return 0;
}
```

a=1234567800, b=1234567820

讨论

- 对两个浮点数如何比较其相等？

