数据类型

C是有类型的语言

- C语言的变量，必须：
  - 在使用前定义，并且
  - 确定类型
- C以后的语言向两个方向发展：
  - C++/Java更强调类型，对类型的检查更严格
  - JavaScript、Python、PHP不看重类型，甚至不需要事先定义

类型安全

- 支持强类型的观点认为明确的类型有助于尽早发现程序中的简单错误
- 反对强类型的观点认为过于强调类型迫使程序员面对底层、实现而非事务逻辑
- 总的来说，早期语言强调类型，面向底层的语言强调类型
- C语言需要类型，但是对类型的安全检查并不足够

C语言的类型

- 整数
  - char、short、int、long、long long
- 浮点数
  - float、double、long double
- 逻辑
  - bool
- 指针
- 自定义类型

蓝色的是C99的类型

类型有何不同

- 类型名称：int、long、double
- 输入输出时的格式化：%d、%ld、%lf
- 所表达的数的范围：char < short < int < float < double
- 内存中所占据的大小：1个字节到16个字节
- 内存中的表达形式：二进制数（补码）、编码

sizeof

- 是一个运算符，给出某个类型或变量在内存中所占据的字节数
  - sizeof(int)
  - sizeof(i)

sizeof

- 是静态运算符，它的结果在编译时刻就决定了
- 不要在sizeof的括号里做运算，这些运算不会做的

整数

整数

- char
- short
- int
- long
- long long

整数

- char：1字节（8比特）
- short：2字节
- int：取决于编译器（CPU），通常的意义是“1个字”
- long：取决于编译器（CPU），通常的意义是“1个字”
- long long：8字节

*整数的内部表达

- 计算机内部一切都是二进制
  - 18 —> 00010010
  - 0 —> 00000000
  - -18 —> ?

*如何表示负数

- 十进制用“-”来表示负数，在做计算的时候
  - 加减是做相反的运算
  - 乘除时当作正数，计算完毕后对结果的符号取反

*二进制负数

- 1个字节可以表达的数：
  - 00000000 — 11111111 (0-255)
- 三种方案：
  1. 仿照十进制，有一个特殊的标志表示负数
  2. 取中间的数为0，如1000000表示0，比它小的是负数，比它大的是正数
  3. 补码

*补码

- 考虑-1，我们希望-1 + 1 —> 0。如何能做到?
  - 0 —> 00000000
  - 11111111 + 00000001 —> 100000000
- 因为0 - 1 —> -1，所以，-1 =
  - (1)00000000 – 00000001 —> 11111111
  - 11111111被当作纯二进制看待时，是255，被当作补码看待时是-1
- 同理，对于-a，其补码就是0-a，实际是2^n - a，n是这种类型的位数

补码的意义就是拿补码和原码可以加出一个溢出的“零”

数的范围

- 对于一个字节（8位），可以表达的是：
  - 00000000 - 11111111
- 其中
  - 00000000 —> 0
  - 11111111 ~ 10000000 —> -1 ~ -128
  - 00000001 ~ 01111111 —> 1 ~ 127

整数的范围

- char：1字节：-128 ~ 127
- short：2字节：-32768 ~ 32767
- int：取决于编译器（CPU），通常的意义是“1个字”
- long：4字节
- long long：8字节

unsigned

- 在整形类型前加上unsigned使得它们成为无符号的整数
- 内部的二进制表达没变，变的是如何看待它们
  - 如何输出
- 11111111
  - 对于char，是-1
  - 对于unsigned char，是255

unsigned

- 如果一个字面量常数想要表达自己是unsigned，可以在后面加u或U
  - 255U
- 用l或L表示long(long)
- *unsigned的初衷并非扩展数能表达的范围，而是为了做纯二进制运算，主要是为了移位

整数越界

- 整数是以纯二进制方式进行计算的，所以：
  - 11111111 + 1 —> 100000000 —> 0
  - 01111111 + 1 —> 10000000 —> -128
  - 10000000 - 1 —> 01111111 —> 127

整数的输入输出

- 只有两种形式：int或long long
  - %d：int
  - %u：unsigned
  - %lld：long long
  - %llu：unsigned long long

8进制和16进制

- 一个以0开始的数字字面量是8进制
- 一个以0x开始的数字字面量是16进制
- %o用于8进制，%x用于16进制

- 8进制和16进制只是如何把数字表达为字符串，与内部如何表达数字无关

* 8进制和16进制

- 16进制很适合表达二进制数据，因为4位二进制正好是一个16进制位
- 8进制的一位数字正好表达3位二进制
  - 因为早期计算机的字长是12的倍数，而非8

选择整数类型

- 为什么整数要有那么多种?
  - 为了准确表达内存，做底层程序的需要
- 没有特殊需要，就选择int
  - 现在的CPU的字长普遍是32位或64位，一次内存读写就是一个int，一次计算也是一个int，选择更短的类型不会更快，甚至可能更慢
  - * 现代的编译器一般会设计内存对齐，所以更短的类型实际在内存中有可能也占据一个int的大小（虽然sizeof告诉你更小）
- unsigned与否只是输出的不同，内部计算是一样的

浮点数

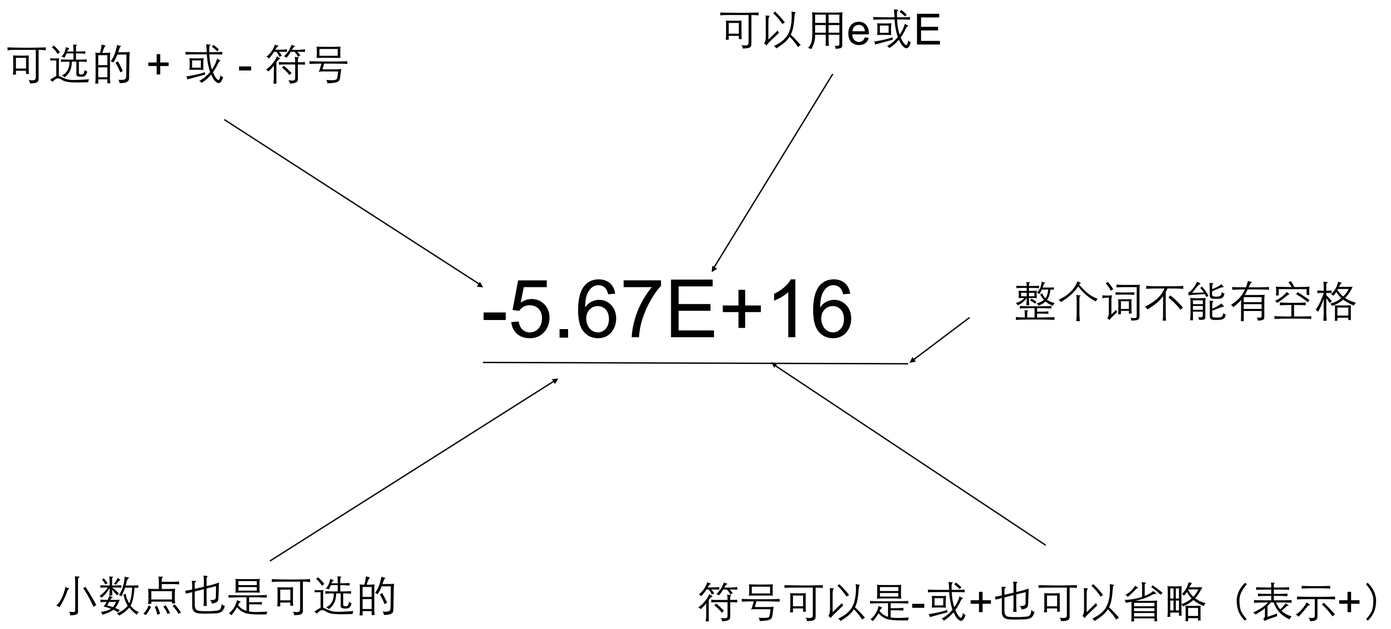
浮点类型

类型	字长	范围	有效数字
float	32	±(1.20x10 0,±inf,nan	7
double	64	±(2.2x10 0,±inf,nan	15

浮点的输入输出

类型	scanf	printf
float	%f	%f, %e
double	%lf	%f,%e

科学计数法



输出精度

- 在%和f之间加上.n可以指定输出小数点后几位，这样的输出是做4舍5入的
  - printf("%.3f\n", -0.0049);
  - printf("%.30f\n", -0.0049);
  - printf("%.3f\n", -0.00049);

超过范围的浮点数

- printf输出inf表示超过范围的浮点数：$\pm\infty$
- printf输出nan表示不存在的浮点数

浮点运算的精度

```c
float a, b, c;

a = 1.345f;
b = 1.123f;
c = a + b;
if (c == 2.468)
    printf("相等\n");
else
    printf("不相等！ c=%.10f，或%f\n",c,c);
```

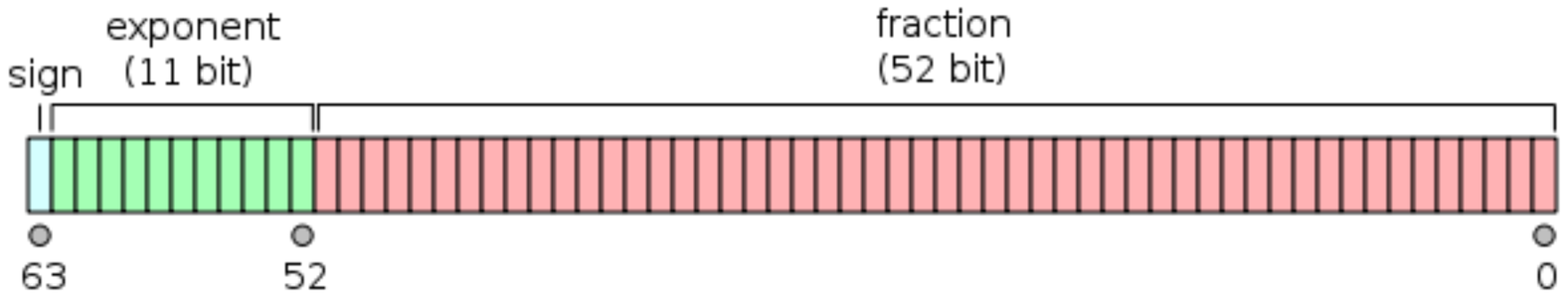
- 带小数点的字面量是double而非float
- float需要用f或F后缀来表明身份

Android计算器低级错误？都是二进制惹的祸！

- f1 == f2可能失败
- fabs(f1-f2) < 1e-12

http://www.guokr.com/article/27173/

* 浮点数的内部表达



- 浮点数在计算时是由专用的硬件部件实现的
- 计算double和float所用的部件是一样的

图片来源：wikipedia.org

选择浮点类型

- 如果没有特殊需要，只使用double
- 现代CPU能直接对double做硬件运算，性能不会比float差，在64位的机器上，数据存储的速度也不比float慢

字符

字符类型

- char是一种整数，也是一种特殊的类型：字符。这是因为：
  - 用单引号表示的字符字面量：'a', '1'
  - ''也是一个字符
  - printf和scanf里用%c来输入输出字符

字符的输入输出

- 如何输入'1'这个字符给char c?
  - scanf("%c", &c);—>1
  - scanf("%d", &i); c=i; —>49
- '1'的ASCII编码是49，所以当c==49时，它代表'1'
  - printf("%i %c\n", c,c);
- 一个49各自表述!

混合输入

- 有何不同?
  - scanf("%d %c", &i, &c);
  - scanf("%d%c", &i, &c);

字符计算

```c
char c = 'A';
c++;
printf("%c\n", c);
```

```c
int i = 'Z' - 'A';
printf("%d\n", i);
```

- 一个字符加一个数字得到ASCII码表中那个数之后的字符
- 两个字符的减，得到它们在表中的距离

大小写转换

- 字母在ASCII表中是顺序排列的
- 大写字母和小写字母是分开排列的，并不在一起
- ‘a’-‘A’可以得到两段之间的距离，于是
  - a+’a’-‘A”可以把一个大写字母变成小写字母，而
  - a+’A’-‘a’可以把一个小写字母变成大写字母

逃逸字符

- 用来表达无法印出来的控制字符或特殊字

```
printf("请分别输入身高的英尺和英寸，"
    "如输入\"5 7\"表示5英尺7英寸: ");
```

个字符

逃逸字符

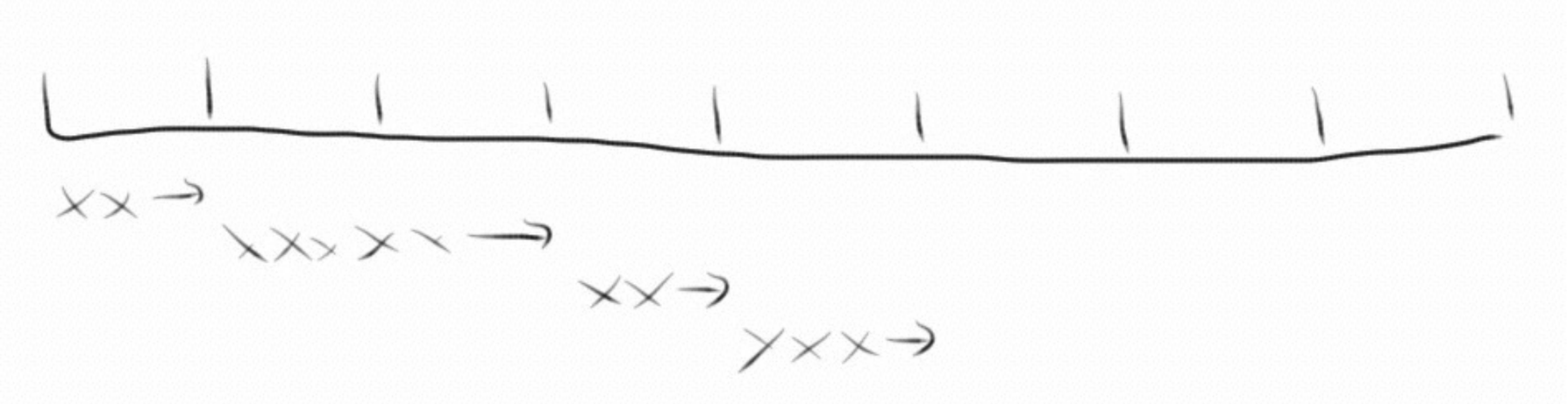
字符	意义	字符	意义
\b	回退一格	\"	双引号
\t	到下一个表格位	\'	单引号
\n	换行	\\	反斜杠本身
\r	回车		

回车换行

- 源自打字机的动作

回车换行

制表位



- 每行的固定位置
- 一个\t使得输出从下一个制表位开始
- 用\t才能使得上下两行对齐

逻辑类型

bool

- #include <stdbool.h>
- 之后就可以使用bool和true、false

bool的运算

- bool实际上还是以int的手段实现的，所以可以当作int来计算
- 也只能当作int来输入输出

类型转换

自动类型转换

- 当运算符的两边出现不一致的类型时，会自动转换成较大的类型
  - 大的意思是能表达的数的范围更大
  - char —> short —> int —> long —> long long
  - int —> float —> double

自动类型转换

- 对于printf，任何小于int的类型会被转换成int；float会被转换成double
- 但是scanf不会，要输入short，需要%hd

强制类型转换

- 要把一个量强制转换成另一个类型（通常是较小的类型），需要：
  - (类型)值
- 比如：
  - (int)10.2
  - (short)32
- 注意这时候的安全性，小的变量不总能表达大的量
  - (short)32768

- 只是从那个变量计算出了一个新的类型的值，它并不改变那个变量，无论是值还是类型都不改变

强制类型转换

```
double a = 1.0;
double b = 2.0;
int i = (int)a / b;
```

```
int i = (int)(a / b);
```

- 强制类型转换的优先级高于四则运算

```
int a = 5;
int b = 6;
double d = (double)(a / b);
```