# C程序设计语言

## 第2章 类型、运算符与表达式

孙志岗
sun@hit.edu.cn
http://sunner.cn

# 标识符（Identifiers）

- 用户自定义的符号叫标识符
  - 如变量名、函数名、宏和类型名
- 标识符由字母、数字和下划线组成，大小写敏感
- 不可以是数字开头
- 标识符要直观，能表达它的功能
- 下划线和大小写通常用来增强可读性
  - `variablename`
  - `variable_name, VARIABLE_NAME`
  - `VariableName, variableName`
- 关键字（keyword）不可作为标识符
  - `int, float, for, while, if`等（教材164页）
- 某些功能的变量采用习惯命名
  - 如：`for`语句所采用的循环变量习惯用`i, j, k`




# b, B, KB, MB, GB, TB

- Megabyte(MB)，中文叫法：兆
- Kilobyte(KB)，中文叫法：K
- Byte，中文叫法：字节
- bit，中文叫法：位
- Gigabyte(GB)，中文叫法：G
- Terabyte(TB)，中文叫法：T

1 TB == 1,024 GB
1 GB == 1,024 MB
1 MB == 1,024 KB
1 KB == 1,024 B
1 B == 8 b




# b, B, KB, MB, GB, TB

- 世界上有10种人，1种人懂二进制，1种人不懂二进制
- 一个位有多大？
  - 只能是“0”或者“1”，这叫二进制
- 二进制诠释了计算机的哲学
  - 种类众多的复杂事物都是由若干种简单事物构成


# b, B, KB, MB, GB, TB

- 一个字节有多大？
  - 可以表示数字0~255
  - 保存一个字符（英文字母、数字、符号），ASCII编码
  - 两个字节保存一个汉字
    - GB2312，6763字
    - GB13000.1，20902字
    - GB18030，27533字
    - BIG5，13000字
  - 两个字节保存一个宽字符，UNICODE编码



# 基本数据类型（Data Type）

- int
  - 整数，在目前绝大多数机器上占4个字节
  - TC2中是2个字节
  - 所占字节数取决于机器字长
- float
  - 单精度浮点数，一般是4个字节长
- double
  - 双精度浮点数，一般是8个字节长
- char
  - 字符，一般是1个字节长
  - 用来表示256个ASCII字符，或者0~255的整数



# 数据类型修饰符

- short
  - short int，短整数，一般2个字节长。通常简写为short
- long
  - long int，长整数，一般是4个字节长。通常简写为long
  - long double，高精度浮点数，一般是10个字节长。
- signed
  - 用来修饰char、int、short和long，说明他们是有符号的整数（正整数、0和负整数）。一般缺省都是有符号的，所以这个修饰符通常省略
- unsigned
  - 用来修饰char、int、short和long，说明他们是无符号的整数（正整数和0）



# 整型类型的取值范围

| | | |
|---|---|---|
| CHAR_MAX | UCHAR_MAX或SCHAR_MAX | char类型的最大值 |
| CHAR_MIN | 0或SCHAR_MIN | char类型的最小值 |
| INT_MAX | +32767 | int类型的最大值 |
| INT_MIN | -32767 | int类型的最小值 |
| LONG_MAX | +2147483647 | long类型的最大值 |
| LONG_MIN | -2147483647 | long类型的最小值 |
| SCHAR_MAX | +127 | signed char类型的最大值 |
| SCHAR_MIN | -127 | signed char类型的最小值 |
| SHRT_MAX | +32767 | short类型的最大值 |
| SHRT_MIN | -32767 | short类型的最小值 |
| UCHAR_MAX | 255 | unsigned char类型的最大值 |
| UINT_MAX | 65535 | unsigend int类型的最大值 |
| ULONG_MAX | 4294967295 | unsigned long类型的最大值 |
| USHRT_MAX | 65535 | unsigned short类型的最大值 |

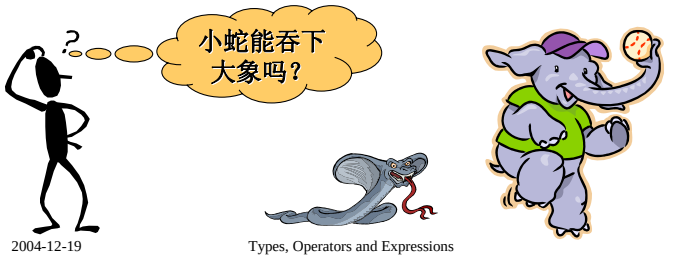
# 浮点类型的取值范围

| | | |
|---|---|---|
| FLT_RADIX | 2 | 指数表示的基数，如2、16 |
| FLT_ROUNDS | | 加法的浮点舍入规则 |
| FLT_DIG | 6 | 精度的十进制数字数 |
| FLT_EPSILON | 1E-5 | 使1.0+$x$ (1.0成立的最小的$x$ |
| FLT_MANT_DIG | | 基数为FLT_RADIX的尾数中的数字数 |
| FLT_MAX | 1E+37 | 最大浮点数 |
| FLT_MAX_EXP | | 使FLT_RADIX$^{n-1}$可表示的最大的$n$ |
| FLT_MIN | 1E-37 | 最小规范化的浮点数 |
| FLT_MIN_EXP | | 使10$^n$为规范化数的最小的$n$ |
| DBL_DIG | 10 | 精度的十进制字数 |
| DBL_EPSILON | 1E-9 | 使1.0+$x$ 1.0成立的最小的$x$ |
| DBL_MANT_DIG | | 基数为FLT_RADIX的尾数中的数字数 |
| DBL_MAX | 1E+37 | 最大双精度浮点数 |
| DBL_MAX_EXP | | 使FLT_RADIX$^{n-1}$可以正常表示的最大的$n$ |
| DBL_MIN | 1E-37 | 最小双精度浮点数 |
| DBL_MIN_EXP | | 使10$^n$为规范化数的最小的$n$ |



# 超出取值范围会怎样？

- TC2中int的范围是-32767~32767
- 如果我们给它一个小于-32767或者大于32767的数会如何呢？
- 现场编程测验……






# 溢出（Overflow）造成的危害

- 一台安装了Windows 95/98的机器，如果连续运行49.7天没有重新启动，可能死机
- 原因：
  - Windows自启动时刻起，有一个计数器，记录系统已经运行了多少毫秒。这个计数器是个unsigned long 类型的变量
  - unsigned long的最大值是：4294967295
  - 一天有 24*60*60*1000 = 86400000毫秒
  - 4294967295 / 86400000 = 49.71026961805……
  - 当49.7天的时候，此计数器会溢出，引起死机



# 浮点数的陷阱

```
#include <stdio.h>

main()
{
   float f;

   f = 123.456;

   if (f == 123.456)
      printf("f is equal to 123.456 indeed.");
   else
      printf("In fact, f is equal to %f\n", f);
}
```

- 运行结果会是什么？

**float.c**

# 浮点数的陷阱

- float的精度低，较易发生精度带来的相等性判断问题
- double精度高，这个问题发生的概率小一些，但也存在
- 解决办法：

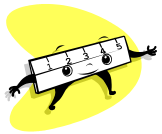
```
if (fabs(f - 123.456) < 1E-5)
   ……
```

根据精度要求设定



# 使用变量要注意

- 不要对变量所占的字节数想当然
  - 用sizeof获得变量或者数据类型的长度
  - 用ANSI C定义的宏确定数据的表示范围，解决溢出问题



sizeof.c



# 常数（Constant）

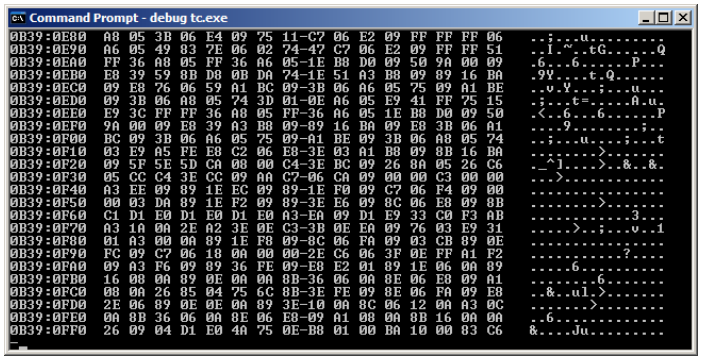
- 整型常数
  - 123、456
  - 123456
  - 123l、123L、123456l、123456L
- 浮点常数
  - 123.45、456.78
  - 1e-2、4.5e3
  - 123.45f、456.78F、1e-2f、4.5e3F
  - 123.45l、456.78L、1e-2l、4.5e3L



# 八进制与十六进制常数

- 以数字“0”开始的整型常数是八进制数
  - 010和10大小不一样
  - 因为八进制并不常用，所以此种表示法比较少见，因而常被用错
- 以“0x”或者“0X”开始的整型常数是十六进制
  - A~F和a~f用来表示十进制的10~15
    - 0x11, 0x05, 0xFA, 0xFF
  - 十六进制的形式比较常用，尤其在进行位一级的控制的时候

# TC.EXE 中的一段16进制代码






# 字符常数

- 字符常数的表示方法
  - 'a', 'A', '5', '%', '$'……
  - 单引号内只能有一个字符，除非用“\”开头
- 就是一个普通整数，也可以参与各种数学运算
  - 每个字符具有一个0~255之间的数值，可从ASCII表查出
  - 注意：'5'和5的区别，A和'A'的区别
  - 字符的数学运算在密码学内用得比较多

**ascii.c**



# 字符常数

- 转义字符
  - 一些特殊字符（无法从键盘输入或者在C语言里有它用）用转义字符表示

| | | | |
|---|---|---|---|
| \a | 响铃符 | \\ | 反斜杠 |
| \b | 回退符 | \? | 问号 |
| \f | 换页符 | \' | 单引号 |
| \n | 换行符 | \" | 双引号 |
| \r | 回车符 | \ooo | 八进制数 |
| \t | 横向制表符 | \xhh | 十六进制数 |
| \v | 纵向制表符 | | |

  - 转义的思想在网络协议和文件格式中经常使用



# 字符串（String）常数

- 用双引号括住的由0个或多个字符组成的字符序列
  - "I am a string"
  - ""表示空字符串
  - 转义字符也可以在字符串中使用
  - 引号只作为字符串开始和结束的标志
  - C语言内部用'\0'表示字符串的结束
  - 除注释外，是唯一可以出现中文的地方
- "x"和'x'是不同的
- <string.h>里定义了一系列专门的字符串处理函数

# 枚举（Enumeration）常数

- 一个几乎被遗忘的角色
- 从程序来窥其一斑

```
enum weeks {
    MON, TUE, WED, THU, FRI, SAT, SUN
};
enum weeks today, tomorrow;

today       = MON;
tomorrow    = today + 1;
if (tomorrow == TUE)
    printf("Tomorrow is Tuesday.\n");
else
    printf("Tomorrow is NOT Tuesday.\n");
```

enum.c



# 变量声明

- 变量必须“先定义，后使用”
  - 所有变量必须在第一条可执行语句前定义
- 声明的顺序无关紧要
- 一条声明语句可声明若干个同类型的变量，变量名之间用逗号分隔
- 变量定义后，即占用内存，可向其存入各种数据，并可通过变量名使用数据
- 声明变量，是初始化变量的最好时机
  - 不被初始化的变量，其值为危险的随机数

```
char  esc = '\\';
int   i = 0;
int   limit = MAXLINE + 1;
float eps = 1.0e-5;
```



# 常量

- 用const修饰定义的变量为常量
  - `const int i=0;`
- 常量只能在定义时赋值，然后不能再改变其值
- 常数、常量、宏和枚举，都可以用来表示一个永远不会改变的数
  - 前者不建议直接使用，而用后三者代替
  - 后三者的工作机理是完全不同的，达到的效果也不尽相同



# C语言中的运算

- 计算机只会计算
- 任何事物都要被表示成数字和公式的形式后，才能被计算机计算（被计算机处理）
  - 事物到数字和公式的转换过程叫数学建模
- 因为：事物在计算机内的处理都是一种计算
- 又因为：计算就要有操作数、运算法则和计算结果
- 所以：事物在计算机内的处理都有操作数、运算法则和计算结果
- 计算结果你可以留用，也可以忽略

# 算术运算符

- +, -, *, /
  - 加、减、乘、除运算
  - 四则混合运算中，先算乘除，后算加减，先算左，后算右
- %
  - 求余运算



# 关系运算符

- >, >=, <, <=, ==, !=
  - 大于，大于等于，小于，小于等于，等于，不等于
  - 关系运算符运算出的结果为0和1
  - 0，表示假，即该关系不成立
  - 1，表示真，即该关系成立
- 在所有涉及到真假判断的地方，0表示假，非0表示真



# 找别扭

```
int a=1;
if (a == 0)
   printf("OK");
```

```
int a=1;
if (a = 0)
   printf("OK");
```

```
int a=0;
if (a == 0)
   printf("OK");
```

```
int a=0;
if (a = 0)
   printf("OK");
```



# == 和 =

```
int a;
a = 0;
a == 1;
```

```
int a;
a == 0;
a = 1;
```

- 一定要分清==和=
- 下面用法能起点小作用：

```
int a=0;
if (0 == a)
   printf("OK");
```

编译出错

```
int a=0;
if (0 = a)
   printf("OK");
```

# 逻辑运算符

- 逻辑运算也被称为布尔（Boolean）运算，运算结果也是1和0
- &&
  - 与运算
  - ( a > b && b > c)；a大于b，并且b大于c
- ||
  - 或运算
  - ( a > b || b > c)； a大于b，或者b大于c
- !
  - 求反
  - ( !a )；如果a是0，结果非0；如果a是非0，结果是0
  - 并不改变a的值



# 类型转换

- 在进行赋值操作时，会发生类型转换
- 将取值范围小的类型转为取值范围大的类型是安全的
- 反之是不安全的
  - 如果大类型的值在小类型能容纳的范围之内，则平安无事
  - 但是，浮点数转为整数，会丢失小数部分（非四舍五入）
  - 反之，转换后的结果必然是错误的，具体结果与机器和实现方式有关。避免如此使用



# 字符串与数值类型之间的转换

- int i = "123"
  - 这样用是不行地
- atof(), atoi(), atol()
  - 把字符串转为double，int和long
  - 定义在stdlib.h中
- sprintf()
  - 可以用来把各种类型的数值转为字符串
  - 定义在stdio.h中



# 自动类型转换

- 两个同种数据类型的运算结果，还是该类型
- 两个不同种数据类型的运算结果，是两种类型中取值范围更大的那种
  - long double > double > float > long > int > short > char
  - 只要两者中有一个是unsigned，就都转为unsigned再计算
- 把数据赋值给另外一种类型变量也会发生自动类型转换
  - 从小到大，顺利转换
  - 从大到小，发出警告（好的编译器会给出）

# 类型强转

- 可以通过“(类型)表达式”的方式把表达式的值转为任意类型
  - 强转时，你必须知道你在做什么
  - 强转与指针，并称C语言两大神器，用好了可以呼风唤雨，用坏了就损兵折将





# 加一和减一运算符

- i++，i--，++i，--i
  - ++让参与运算的变量加1，--让参与运算的变量减1
  - 运算符为后缀，先取i的值，然后加/减1
  - 运算符为前缀，先加/减1，然后取i的值
- 在一行语句中，使用加1或者减1运算的变量只能出现
  - 不仅可读性差，而且因为编译器实现的方法不同，容易导致不同编译器运行效果不一样，贻害无穷



# 位操作运算符

- &
  - 按位与运算
- |
  - 按位或运算
- ^
  - 按位异或运算
- <<
  - 按位左移运算
- >>
  - 按位右移运算
- ~
  - 按位求反



# 赋值运算符

- 赋值运算的结果是被赋值变量赋值后的值
  - a = b = c = 0;
- 下面两个语句是等价的
  - i = i + 2;
  - i += 2;
- +、-、*、/、%、<<、>>、&、^、|运算符都可以按此种方式处理
- 这种形式看起来更直观，而且执行效率一般也能更高一些

# 条件表达式

- 把a和b中的最大值放入z中
  - ```
    if (a > b)
        z = a;
    else
        z = b;
    ```
  - ```
    z = (a > b) ? a : b;
    ```
- 此种表达式切忌用得过于繁杂



# 优先级

- ( ) [ ] -> .
- ! ~ ++ -- + - * & (类型) sizeof
- * / %
- + -
- << >>
- < <= > >=
- == !=
- &
- ^
- |
- &&
- ||
- ? :
- = += -= *= /= %= &= ^= |= <<= >>=
- ,



# 优先级

- 能背下优先级表的人凤毛麟角
  - 脑细胞太宝贵了，不能用来死记硬背
- 用括号来控制运算顺序更直观、方便，并减少出错的概率
  - 先算乘除，后算加减，有括号就先算括号里的
  - 括号太多，有时候不清晰
    - 注意用空格做好分隔
    - 实在不行就拆分表达式



# 这一章我们学到了

- 标识符的命名规则
- 数据类型
  - char, short, int, long, float, double, long double
  - signed, unsigned
  - enum
- sizeof
- 常数、转义字符
- 算术运算、关系运算、逻辑运算、加一/减一运算、位运算、赋值运算
- 类型转换
- ? :
- 优先级

# ASCII字符表1

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 000 | (nul) | 016 | ▶ | (dle) | 032 | sp | 048 | 0 | 064 | @ | 080 | P | 096 | ` | 112 | p |
| 001 | ☺ | (soh) | 017 | ◀ | (dc1) | 033 | ! | 049 | 1 | 065 | A | 081 | Q | 097 | a | 113 | q |
| 002 | ☻ | (stx) | 018 | ↕ | (dc2) | 034 | " | 050 | 2 | 066 | B | 082 | R | 098 | b | 114 | r |
| 003 | ♥ | (etx) | 019 | ‼ | (dc3) | 035 | # | 051 | 3 | 067 | C | 083 | S | 099 | c | 115 | s |
| 004 | ♦ | (eot) | 020 | ¶ | (dc4) | 036 | $ | 052 | 4 | 068 | D | 084 | T | 100 | d | 116 | t |
| 005 | ♣ | (enq) | 021 | § | (nak) | 037 | % | 053 | 5 | 069 | E | 085 | U | 101 | e | 117 | u |
| 006 | ♠ | (ack) | 022 | ▬ | (syn) | 038 | & | 054 | 6 | 070 | F | 086 | V | 102 | f | 118 | v |
| 007 | • | (bel) | 023 | ↨ | (etb) | 039 | ' | 055 | 7 | 071 | G | 087 | W | 103 | g | 119 | w |
| 008 | ◘ | (bs) | 024 | ↑ | (can) | 040 | ( | 056 | 8 | 072 | H | 088 | X | 104 | h | 120 | x |
| 009 | | (tab) | 025 | ↓ | (em) | 041 | ) | 057 | 9 | 073 | I | 089 | Y | 105 | i | 121 | y |
| 010 | | (lf) | 026 | | (eof) | 042 | * | 058 | : | 074 | J | 090 | Z | 106 | j | 122 | z |
| 011 | ♂ | (vt) | 027 | ← | (esc) | 043 | + | 059 | ; | 075 | K | 091 | [ | 107 | k | 123 | { |
| 012 | ♀ | (np) | 028 | ∟ | (fs) | 044 | , | 060 | < | 076 | L | 092 | \ | 108 | l | 124 | \| |
| 013 | | (cr) | 029 | ↔ | (gs) | 045 | - | 061 | = | 077 | M | 093 | ] | 109 | m | 125 | } |
| 014 | ♫ | (so) | 030 | ▲ | (rs) | 046 | . | 062 | > | 078 | N | 094 | ^ | 110 | n | 126 | ~ |
| 015 | ☼ | (si) | 031 | ▼ | (us) | 047 | / | 063 | ? | 079 | O | 095 | _ | 111 | o | 127 | ⌂ |

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 128 | Ç | 143 | Å | 158 | ₧ | 172 | ¼ | 186 | ‖ | 200 | ╚ | 214 | ╓ | 228 | Σ | 242 | ≥ |
| 129 | ü | 144 | É | 159 | ƒ | 173 | ¡ | 187 | ╗ | 201 | ╔ | 215 | ╫ | 229 | σ | 243 | ≤ |
| 130 | é | 145 | æ | 160 | á | 174 | « | 188 | ╝ | 202 | ╩ | 216 | ╪ | 230 | µ | 244 | ⌠ |
| 131 | â | 146 | Æ | 161 | í | 175 | » | 189 | ╜ | 203 | ╦ | 217 | ┘ | 231 | τ | 245 | ⌡ |
| 132 | ä | 147 | ô | 162 | ó | 176 | ░ | 190 | ╛ | 204 | ╠ | 218 | ┌ | 232 | Φ | 246 | ÷ |
| 133 | à | 148 | ö | 163 | ú | 177 | ▒ | 191 | ┐ | 205 | ═ | 219 | █ | 233 | Θ | 247 | ≈ |
| 134 | å | 149 | ò | 164 | ñ | 178 | ▓ | 192 | └ | 206 | ╬ | 220 | ▄ | 234 | Ω | 248 | ° |
| 135 | ç | 150 | û | 165 | Ñ | 179 | │ | 193 | ┴ | 207 | ╧ | 221 | ▌ | 235 | δ | 249 | ∙ |
| 136 | ê | 151 | ù | 166 | ª | 180 | ┤ | 194 | ┬ | 208 | ╨ | 222 | ▐ | 236 | ∞ | 250 | · |
| 137 | ë | 152 | ÿ | 167 | º | 181 | ╡ | 195 | ├ | 209 | ╤ | 223 | ▀ | 237 | ø | 251 | √ |
| 138 | è | 153 | Ö | 168 | ¿ | 182 | ╢ | 196 | ─ | 210 | ╥ | 224 | α | 238 | ∈ | 252 | ⁿ |
| 139 | ï | 154 | Ü | 169 | ⌐ | 183 | ╖ | 197 | ┼ | 211 | ╙ | 225 | ß | 239 | ∩ | 253 | ² |
| 140 | î | 155 | ¢ | 170 | ¬ | 184 | ╕ | 198 | ╞ | 212 | ╘ | 226 | Γ | 240 | ≡ | 254 | ■ |
| 141 | ì | 156 | £ | 171 | ½ | 185 | ╣ | 199 | ╟ | 213 | ╒ | 227 | π | 241 | ± | 255 | |
| 142 | Ä | 157 | ¥ | | | | | | | | | | | | | | |



# ASCII字符表2