第二讲 基本数据类型及其运算

(第一部分)

周水庚

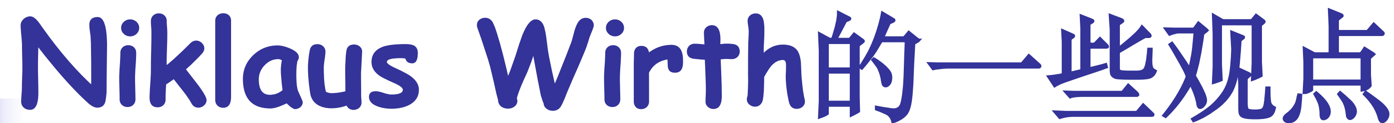
2024年9月12日

Niklaus Wirth (1934–2024)

(Is *Nih-klaus Virt*, not *Nickles Worth*)

Niklaus Wirth

- 1934年2月15日出生于瑞士
- 1959年获ETH学士学位；1960年获加拿大Laval大学硕士学位；1963年获UC Berkely博士学位
- 1963 – 1967在Stanford任教，后短期在 University of Zurich任教。 1968年成为 ETH教授，直到1999年退休
- 主要贡献
  - 程序设计语言：Euler (1965), Algol-W (1966), Pascal (1970), Modula (1975), Modula-2 (1978), Oberon (1987)等
  - 结构化程序设计
- 1984年图灵奖获得者/1988年IEEE先驱奖获得者
- 1994年当选美国工程院外籍院士
- 2024年1月1日去世



"Software is getting slower more rapidly than hardware is becoming faster."
------ Niklaus Wirth Law

Increasingly, people seem to misinterpret complexity as sophistication, which is baffling—the incomprehensible should cause suspicion rather than admiration.

Systems should be simple, efficient, and "elegant."

Algorithms + Data Structures = Programs

Top 10 Fascinating Facts about Niklaus Wirth

- **He graduated from various institutions until he got his Doctor of Philosophy**
- **He taught at Stanford and Zurich University**
- **Wirth was a member of International Federation of Information Processing (IFIP)**
- **He became a Fellow of the Computer History Museum in 2004**
- **Wirth was the chief designer of the programming languages Euler**
- **He was a major designer of Lilith working station**
- **Wirth received the Turin ward in 1984**
- **He is the founding father of Wirth's law**
- **Wirth is the writer of *Algorithms + Data Structures = Programs***
- **Wirth also published a textbook called *Systematic Programming: An Introduction***

姚期智（Andrew Chi Chih Yao）

姚期智（Andrew C. Yao）

- 1967年获得台湾大学物理学士学位
- 1972年获得美国哈佛大学物理博士学位
- 1975年获得美国伊利诺依大学(UIUC)计算机科学博士学位
- 1975年至1986年曾先后在美国麻省理工学院数学系、斯坦福大学计算机系、加利福尼亚大学伯克利分校计算机系任助教授、教授
- 1986年至2004年在普林斯顿大学计算机科学系担任Wiliam and Edna Macaleer工程与应用科学教授；2004起在清华大学任教；2009年提出“中国计算科学2020计划”
- 1998年当选为美国科学院院士；2000年当选为美国科学与艺术学院院士，获图灵奖；2004年当选为中国科学院外籍院士， 2016年转为中国科学院院士
- 2017年加入中国籍

提要

- C语言中的数据类型
- 常量与变量
- 整型数据
- 字符型数据
- 实型数据
- 基本数据类型混合运算和类型转换
- 数据运算
- 表达式和表达式语句

提要

- C语言中的数据类型
- 常量与变量(Constant & Variable)
- 整型数据(Integer)
- 字符型数据(Character or Char)
- 实型数据(Real)
- 基本数据类型混合运算和类型转换
- 数据运算(Operation)
- 表达式(Expression)和表达式语句

C语言中的数据类型

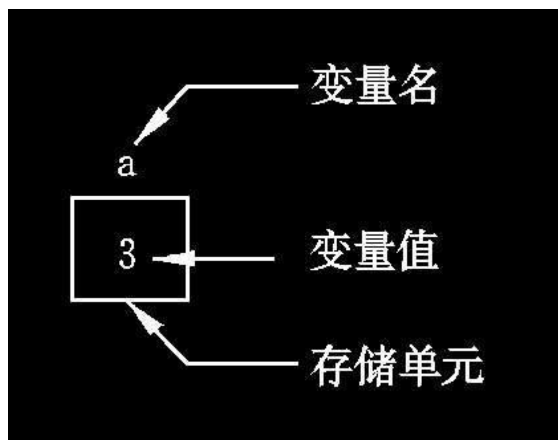
- 基本类型
  - 整型
  - 字符型
  - 实型(浮点型):单精度型、双精度型
- 指针类型
- 复合类型
  - 数组类型
  - 结构类型
  - 联合类型：union
  - 枚举类型：enum
  - 自定义类型

提要

- C语言中的数据类型
- 常量与变量
- 整型数据
- 字符型数据
- 实型数据
- 基本数据类型混合运算和类型转换
- 数据运算
- 表达式和表达式语句

常量与变量

- 常量（constant）：在程序运行过程中，其值不能被改变的量
- 变量：在程序运行过程中，其值可以改变的量
  - 构成：变量类型、变量名、变量值



变量（Variable)

- C规定在程序中所有用到的变量都必须在程序中定义
- C中变量对大小写敏感
- 变量的定义一般是在一个函数开头的声明部分
  - 也可放在函数中某一段程序内，但其作用域（范围）只限于其所在的程序段

提要

- C语言中的数据类型
- 常量与变量
- **整型数据**
- 字符型数据
- 实型数据
- 基本数据类型混合运算和类型转换
- 数据运算
- 表达式和表达式语句

整型数据

- 整型数据按其数值范围划分成三种
  - 基本型、短整型、长整型
- 按其内部表示的最高位的不同理解，可分为两类
  - 带符号整型（最高位为符号位）、不带符号整型
- 带符号整型的标记方法
  - 基本型：用 int 标记
  - 短整型：用 short int 标记，简写为 short
  - 长整型：用 long int 标记，简写为 long

整型数据（续）

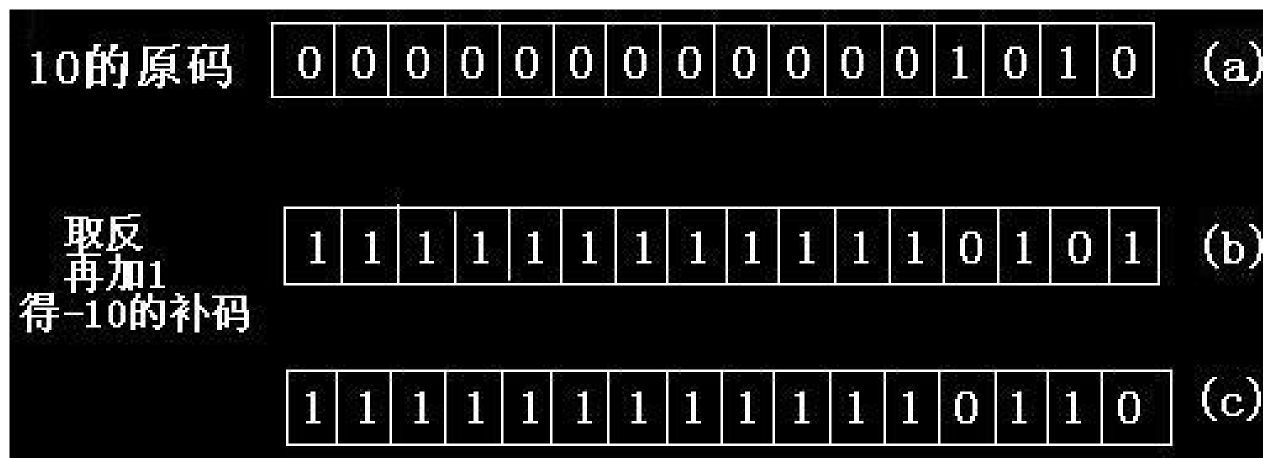
- 不带符号整型指存储一个整数的存储空间中的全部二进位都用作存放数据本身，而不存储符号位
- 不带符号整型的标记方法
  - 不带符号基本型：unsigned int
  - 不带符号短整型：unsigned short
  - 不带符号长整型：unsigned long

整型数据（续）

- 一般一个**机器字（word）**存放一个**int**型整数；**long**型整数的字节数不小于**int**型整数的字节数；**short**型整数的字节数不多于**int**型整数的字节数
- 如**TURBO C**，短整型整数和基本型整数是**2**个字节，**16**个二进位；长整型整数为**4**个字节，**32**个二进位。**在VC中，基本型整数和长整型整数都为4个字节**

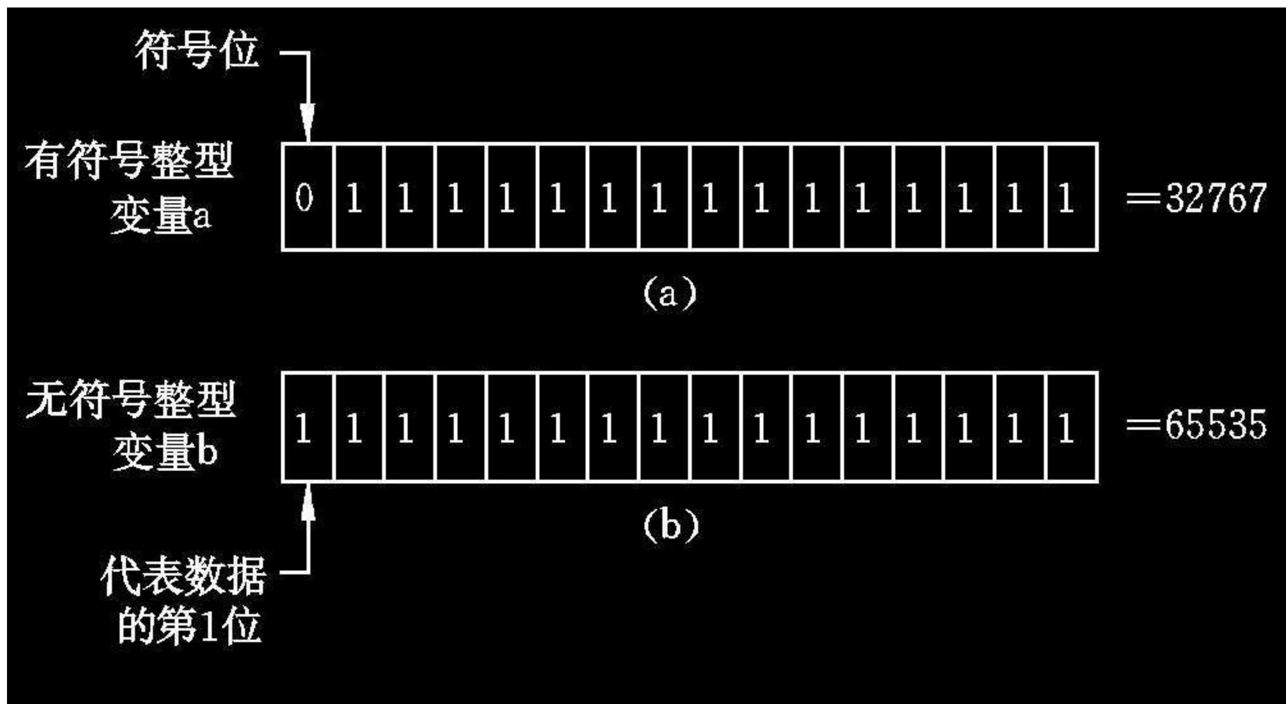
整型数据（续）

- 在机器内部，负整数是以补码(complement) 表示的
  - 采用补码，是为了简化计算
- 负数的补码计算
  - 将该数的绝对值的二进制形式，1）按位取反；2）加1



- **整数表示的数值范围**
  - 用**16**个二进位（**2 bytes**）存储一个整数
    - 带符号整数的数值范围是：**-32768 ～ 32767**
    - 不带符号整数的数值范围是：**0 ～ 65535**
  - 若用**32**个二进位存储一个整数
    - 带符号整数的数值范围是
      - **－2147483648 ～ 2147483647**
    - 不带符号整数的数值范围是
      - **0 ～ 4294967295**

整型数据（续）



signed vs. unsigned

带符号的整数（2 bytes）

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

=-1

1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

=-32768

整型数据（续）

不同类型的13

整型数据（续）

- 整型常量的书写形式有三种
  - 十进制整数：如 0，123，-45
  - 八进制整数：以数字符0开头，并由数字符0～7组成的数字符序列。如0123表示八进制整数，其值等于十进制整数：$1*8^2 + 2*8^1 + 3 = 83$
  - 十六进制整数 表示十六进制数的数字符有 16 个，它们分别是 0～9 和 A、B、C、D、E、F，其中六个字母也可以小写。以0x(或0X)开头，并由十六进制数的数字符组成的数字符序列
    - 如0x123表示十六进制整数，其值等于十进制整数：$1*16^2 + 2*16^1 + 3 = 291$
    - 0xabc，其值等于$10*16^2 + 11*16^1 + 12 = 2748$

将任意进制数转换为10进制数

$$N = \sum_{i=-k}^{m} D_i * r^i$$

- N　代表一个10进制的正整数
- r　是这个数制的基(Radix)
  - 10进制，r=10
  - 8进制，r=8
- i　表示这些符号排列的位号
- D_i 是位号为i的位上的一个符号
- r^i 是位号为i的位上的 1 代表的值
- D_i*r^i 是第i位的所代表的实际值
- 表示m+k+1位的值求累加和
- 例如：0x123= $1*16^2 + 2*16^1 + 3$ = 291

- 十进制与其它进制正整数的转换
  - w进制数据到十进制数据的转换

    w进制的数据$a_1a_2…a_n$转换为十进制数:

    $a_1×w^{n-1}+ a_2×w^{n-2} …+a_{n-1}×w+a_n$
  - 十进制数据到w进制数据的转换

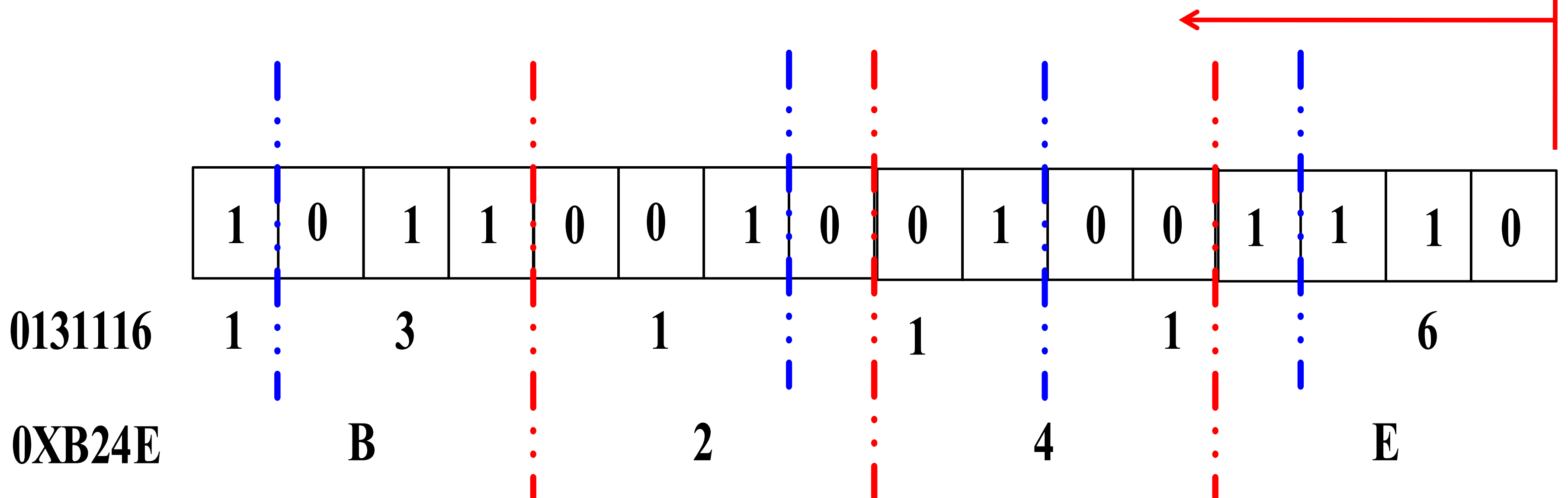
    十进制的数据$a_1a_2…a_m$=

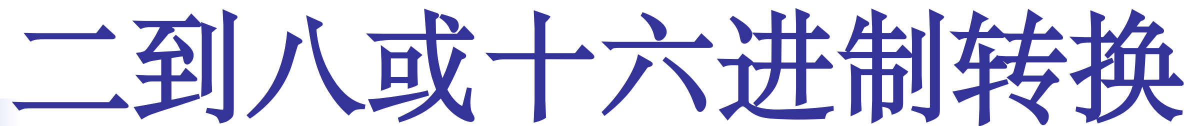
    $b_1×w^{n-1}+ b_2×w^{n-2} …+b_{n-1}×w+b_n$

    如何求b_i(i=1 ~ n)?

整型数据（续）

- 二进制数据向八进制、16进制数据转换
- 16进制数据向八进制、二进制数据转换





- 二进制转到八进制

  从小数点向左右三位一分组

  $(10\ 011\ 100\ .\ 01)_2 = (\ 234\ .\ 2\ )_8$

- 二进制转十六进制

  从小数点向左右四位一分组

  $(1001\ 1100\ .\ 01)_2 = (\ 9C\ .\ 4\ )_{16}$

  0100

十进制转二进制

整数部分除2取余

```
2 |1 1   ----- 1  低
  ------
 2|   5  ----- 1   |
  ------           |
  2|  2  ----- 0   ↓
   -----
   2| 1  ----- 1  高
    ----
      0
```

除尽为止 **1011**

小数部分乘2取整

```
          0.625 * 2
       -------------
高  1     0.25  * 2
↑      -------------
|   0     0.5   * 2
|      -------------
低  1     0.0
```

求得位数满足要求为止

整型数据（续）

- 长整型常数和无符号整型常数表示
  - Long型常数：在整型常数之后接上字母 L(或l)，即为long型常数
    - 例如：0L、132L等
  - 不带符号的整型常数：在整型常数后面接上字母U(或u)，则指明该整数常数是 unsigned 型的。例如1U、122U等
  - 为指明不带符号的long型常数，则需在整型常数之后同时加上字母U和L，表明该整型常数是 unsigned long 型的
    - 例如：22UL、35LU 等

整数的溢出

```
#include <stdio.h>
int main() {
 int a=32767, b;
 b=a+1;
 printf("%d, %d", a, b);
}
```

运行结果为：32767，-32768

a:	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	+ 32767
b:	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	- 32768

前提是：int用2 bytes表示

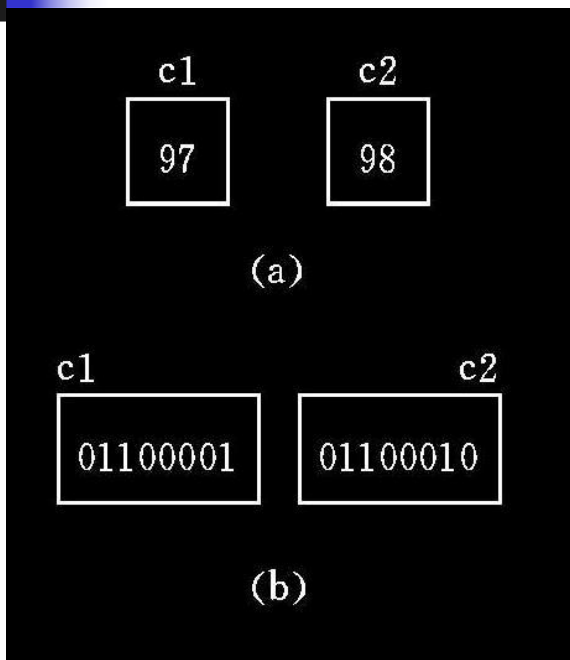
提要

- C语言中的数据类型
- 常量与变量
- 整型数据
- 字符型数据
- 实型数据
- 基本数据类型混合运算和类型转换
- 数据运算
- 表达式和表达式语句

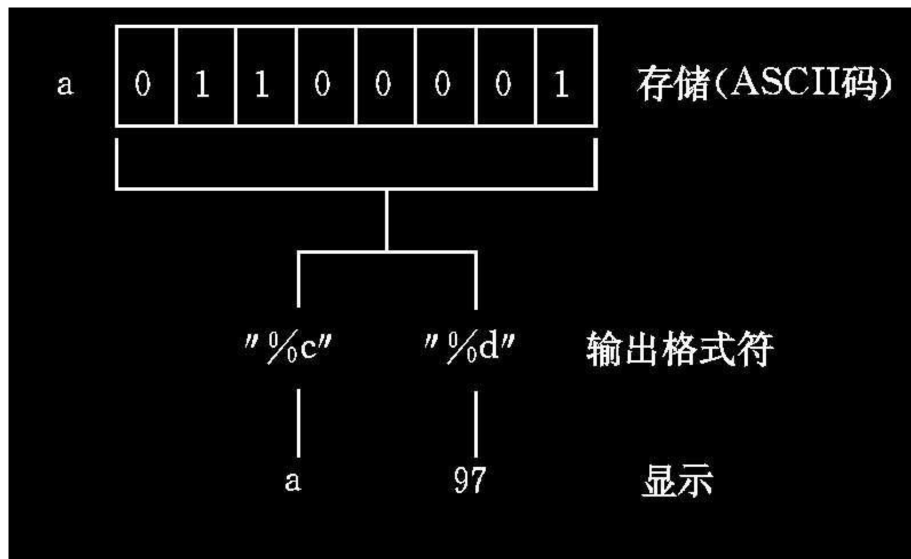
字符型数据

- 字符型数据用于表示字符及其代表的值
- 字符型数据的内部表示是字符的ASCII（American Standard Code for Information Interchange）代码（二进制形式）
- 字符型数据的类型符用 char 来标记
  - char c1，c2; /* 定义两个字符变量 */
- 字符型数据占一个字节（8位二进制位）表示，也可当作一个8位的整型数据
- 中文汉字，用2个字节编码（UNICODE）

字符型数据(续)



字符a、b的存储



字符输出

字符型数据(续)

- 字符型常量是单个字符，它的书写方法:
  - 普通字符：用单引号括起一个字符，如 'a'
  - 特殊字符：用‘\字符或字符列’来标记，如‘ \r ’、‘ \n ’、‘\t ’，表示特殊字符
    - 回车符‘ \r ’：carriage return (CR)，表示回到当前行首，覆盖已有输出
    - 换行符‘ \n ’：光标移到下一行的行首
    - 制表符‘ \t ’：使输出位置横向跳至下一个输出区开始列
    - 键盘上的return相当于' \n'

特殊字符表

标记形式	功　　　能
\n	换行符(打印位置移到下一行的行首)
\t	制表符，横向跳格到下一个输出区首
\v	竖向跳格符
\b	退格
\r	回车(打印位置移到当前行首，覆盖已有打印内容)
\f	走纸换页
\a	产生响铃声
\\	反斜杠字符 \
\'	单引号符 ’
\"	双引号符 ”
\ddd	ddd为1至3个8进制数字,以该值为ASCII码的字符
\xhh	hh为1至2个16进制数字,以该值为ASCII码的字符

- 打印机与显示屏输出的组织方法稍有不同
  - 打印机：仅当一行字符填满或遇换行符时才输出，即整行一次性输出。当输出空格符或制表符时，作跳格处理，不用空格符填充
  - 显示器：逐个字符输出，空格符及制表符经过位置都用空格符输出

实例

```
int main(){
 printf(" ab c\t de\rf\tg\n");
 printf("h\ti\b\bj k");
}
```

打印结果:

|f|a|b| |c| | | |g|d|e|
|h| | | | | | |j|i|k|

屏幕输出:

|f| | | | | | | |g|d|e|
|h| | | | | | |j| |k|

注：每个表单元**8**个字符宽。

字符型数据(续)

- 字符型常量与字符串常量书写形式的区别
  - 字符串常量是一对双引号括起来的字符序列
    - "I am a student."
    - "China", "a", "$1234.00"
- 字符型常量'a' 与字符串常量"a"不同，不允许将字符串常量赋给字符变量
- 字符型数据可与整型数据混合运算
  - 由于字符型数据以ASCII代码的二进制形式存储，它与整数的存储形式相类似。因此，在C程序中，字符型数据和整型数据之间可以通用，字符型数据与整型数据可混合运算。一个字符型数据可以用字符格式("%c")输出，显示字符本身；也可以用整数形式输出，显示字符的 ASCII 码值

字符型数据(续)

- 有些系统(如Turbo c)将字符变量中的最高位作为符号位，也就是将字符处理成带符号的整数，即signed char型。它的取值范围是-128-127
- 如果不想按有符号处理，可以将字符变量定义为unsigned char类型， 这时其取值范围是0-255。signed char 和unsigned char的含义及用法与signed int和unsigned int相仿， 但它只有一个字节

字符串常量

- C规定：在每一个字符串的结尾加一个“字符串结束标志”，以便系统据此判断字符串是否结束
- C规定以字符‘\0’作为字符串结束标志。‘\0’是一个ASCII码为0的字符，这是“空操作字符”，不引起任何控制动作，也不是一个可显示的字符
- 如果有一个字符串“CHINA”，实际上在内存中是
  - C H I N A \0，存储上包含6个字符，代表前5个字符，最后一个字符为‘\0’，输出时不输出‘\0’
- C中没有专门的字符串类型，如果想将一个字符串存放在变量中，必须使用字符数组

实例

```
#include  <stdio.h>
int main()
{  char c1, c2; /* 定义两个字符型变量    */
   c1 = 97;    /* 'a'的 ASCII 码值为 97  */
   c2 = c1+1; /* 字符型与整型数据混合运算 */
   printf("c1 = %c, c2 = %c\n", c1, c2);
   printf("%c's ASCII code = %d\n", c2, c2);
   printf("I am \100 Fudan.\10\n");
   printf("Shanghai\0Beijing");
   return 0;
 }
```

程序输出：

```
    c1 = a, c2 = b
    b's ASCII code = 98
    I am @ Fudan
    Shanghai
```

\100代表ASCII码为64的字符，即@；
\10代表ASCII码为8的字符，退格；
\0代表字符串结束符。

提要

- C语言中的数据类型
- 常量与变量
- 整型数据
- 字符型数据
- 实型数据
- 基本数据类型混合运算和类型转换
- 数据运算
- 表达式和表达式语句

实型数据

- 实型数据（real number）又称浮点数（float number）
- 因计算机只能表示有限位的实数，故实型数据是实数的有限子集。实型数据有数值大小（范围）和有效位数多少（精度）两个方面
- 实型有三种
  - 单精度实型:用float标记，单精度实型又称浮点型
    - float x, y;
  - 双精度实型:用double标记
    - double result;
  - 长双精度实型:用long double标记
    - long double z;

实型数据(续)

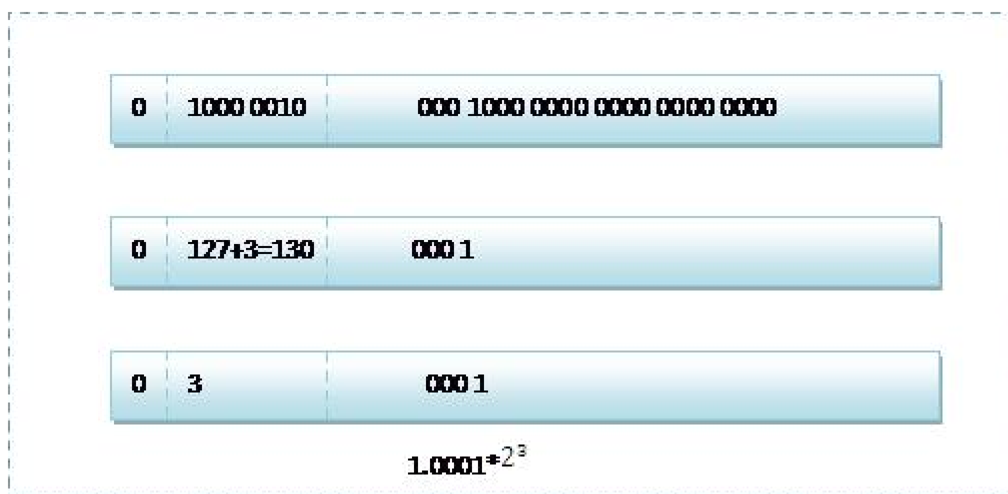
- 实型数据在内存中的存放形式
  - 系统把一个实型数据分成符号部分、小数（尾数）部分和指数部分，分别存放

实型数据(续)

- 单精度浮点数使用32位来表示实数：31位是符号位；30位到23位这8位用于存放偏置指数（biased exponent）；22位到0位这23位存放有效数字（significant)。
- 将十进制实数转化为单精度浮点数步骤如下
  - 将十进制实数转化为对应的二进制实数
  - 将得到的二进制实数以类似于科学计数法的方法写出：+1.xxxxEyyyy；xxxx称为有效数字，yyyy称为指数
  - 实数若为正，31位为0；若为负，31位为1
  - 将指数加上十六进制数7F得到偏置指数，然后将所得结果放到30位到23位的右边，左边补0
  - 将有效数字放到22位到0位的左边，右边补0

实型数据(续)

- 8.5的二进制表示为： 1000.1
- **进一步地**,用二进制的科学计数法表示为:1.0001*2^3



注：指数位采用“移位存储”方式，原始数+127

实型数据(续)

- 计算机中，是用二进制数来表示小数部分以及用2的幂次来表示指数部分的
- 在4个字节(32位)中，究竟用多少位来表示小数部分，多少位来表示指数部分，标准C并无具体规定，由各C编译系统自定
- 不少C编译系统以24位表示小数部分(包括符号)，以8位表示指数部分(包括指数的符号)
- 小数部分占的位(bit)数愈多，数的有效数字愈多，精度愈高；指数部分占的位数愈多，则能表示的数值范围愈大

实型数据(续)

- **float 型实型数据**
  - 在内存中占用4个字节(32个二进位)，约6-7个十进位有效数字，能表示绝对值最接近0的实数约为10的-38次方，最大的实数约为10的38次方
- **double型实型数据**
  - 占用8个字节(64个二进位)，15-16个十进位有效数字，能表示绝对值最接近0的实数约为10的-308次方，最大的实数约为10的308次方
- **long double型实型数据**
  - 占用16个字节(128个二进位)，18-19个十进位有效数字，能表示绝对值最接近0的实数约为10的-4392次方，最大的实数约为10的4392次方

实型数据(续)

- 实型常量的一般书写格式
  - 正负号　整数部分 . 小数部分 指数部分
  - 其中，正负号可有可无，无正负号即为正实数；整数部分和小数部分都是十进数字符序列；指数部分是 e(或E)，接上正负号和十进数字符序列
- 三条限制
  - 整数部分和小数部分可以任选，但不可同时都没有
  - 小数点和指数部分不可以同时都没有
  - 指数部分以1个e或者E开头，后跟一个整数
    - 合法的实型常数：7.、.457、1E5、1.5e-6
    - 不正确的实型常数： E4、.E5、4.0E
- 用 f 标识 float 型常数；用l(或 L)标识 long double 型常数，而无后缀符的实型常数被认为是 double 型常数

实型数据(续)

- 计算机接受的实型值与书写的实数会有一定的误差
- 例如：float x = 111111.111，因x只有约7位有效数字，实数111111.111所对应的二进制形式中，超出存储位数的那些位就不会被存储
- 由于实数运算的计算误差，两个数学上完全等价的计算公式，会因计算顺序不同，得到的两个结果不能保证相等
- 常用判别两实数非常相近的表达式
  - 绝对误差　fabs(x-y) < 1e-6
  - 相对精度　fabs(x-y) <= fabs(x*1e-6)

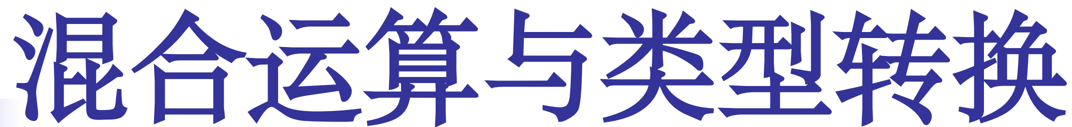
实例

```
int main() {
    float a, b;
    a = 123456.789e5;
    b = a + 20 ;
    printf("%f", b);
}
```

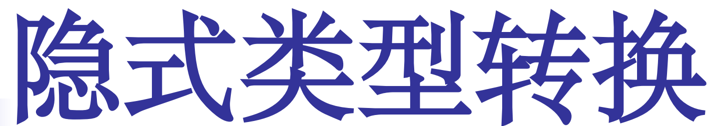

- 准确应该是：12345678920
- 运行结果可能是：a=b= 12345678848.000000

提要

- C语言中的数据类型
- 常量与变量
- 整型数据
- 字符型数据
- 实型数据
- 基本数据类型混合运算和类型转换
- 数据运算
- 表达式和表达式语句

混合运算与类型转换

- 基本数据类型可以混合运算，这时需要进行数据类型转换
- 数据类型转换有两种
  - 隐式类型转换(Implicit transformation)
  - 显式类型转换(Explicit transformation)



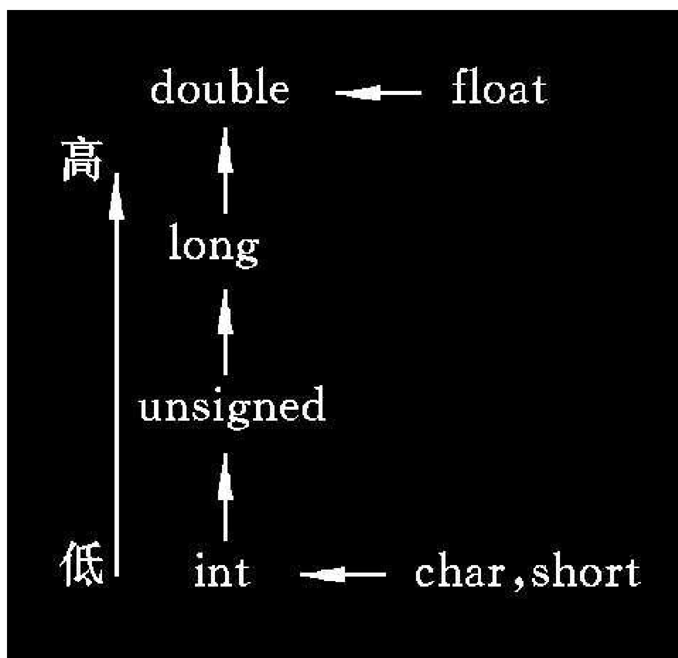
隐式类型转换

- 数据类型不同，数据所占内存字节数和其内部表示形式也会不相同
- 在算术运算中（其它运算例外），一个运算的两个运算分量，能根据运算分量类型的不同情况，自动将其中一个运算分量的值从一种类型转换成另一种类型

隐式类型转换规则

- 按所列优先顺序实行
  - 如有long double型，则其余运算分量也转换成long double 型
  - 如有double型，则其余运算分量也转换成 double 型
  - 如有float型，则其余运算分量也转换成 float 型。
  - 如有unsigned long型，则其余运算分量也转换成unsigned long型
  - 如有long型，则其余运算分量也转换成 long型
  - 如有unsigned型，则其余运算分量都转换成 unsigned型
  - 最后，运算分量和结果都是 int 型

类型转换



类型转换优先级

显式类型转换

- 显式地将一种类型表达式的值强制转换成另一类型
- 显式类型转换的书写形式：（类型名）表达式
  - 其中（类型名）是对其后的表达式作强制类型转换，它将表达式的值强制地转换成类型名所指明的类型
  - 例如，库函数 sqrt()是求一个double型值的平方根。为求整型变量m的平方根，正确的写法是：
    - rm = (int) sqrt((double)m)
    - 在求m的平方根之前，先将m的值转换成double型，然后去调用函数sqrt()，并将结果转换成int型。
- 类型转换不只改变表达式值的类型，也可能会因两种表示形式上的差异，值的大小会有一些误差

显式类型转换(续)

- (double) a /*将a转换成double类型*/
- (int) (x+y) /*将x+y的值转换成整型*/
  - 不同于(int) x+y
- (float) (5%3) /*将5%3的值转换成float型*/
- 在强制类型转换时，得到一个所需类型的中间变量，原来变量的类型未发生变化
  - 例如：(int) x
  - 如果x原指定为float型，进行强制类型运算后得到一个int型的中间变量，它的值等于x的整数部分，而x的类型不变(仍为float型)

显式类型转换(续)

```
int main(){
  float x;
  int i;  x=3.6;
  i=(int) x;
  printf("x=%f,  i=%d", x, i);
}
```

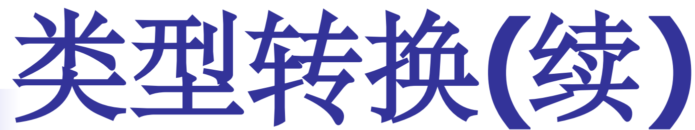
运行结果如下：

x=3.600000, i=3

x类型仍为float型，值仍等于3.6

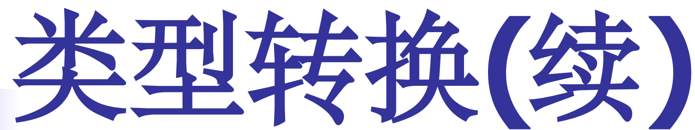
类型转换(Type Conversion)

- 将实型数据(包括单、双精度)赋给整型变量时，舍弃实数的小数部分
  - 如i为整型变量，执行“i=3.56”的结果是使i的值为3， 在内存中以整数形式存储
- 将整型数据赋给单、双精度变量时，数值不变，但以浮点数形式存储到变量中
  - 如将23赋给float变量f，即f=23，先将23转换成23.00000，再存储在f中
  - 如将23赋给double型变量d，即d=23，则将23补足有效位数字为23.00000000000000，然后以双精度浮点数形式存储到d中



类型转换(续)

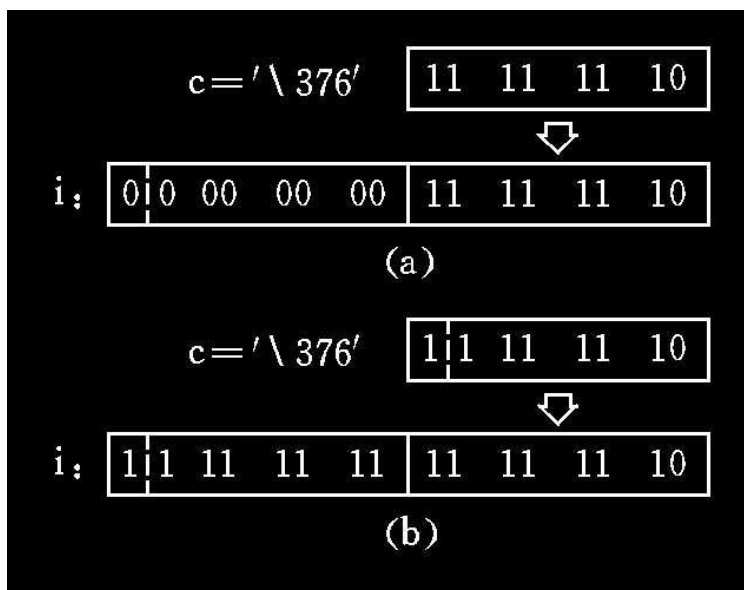
- 将一个double型数据赋给float变量时，截取其前面7位有效数字，存放到float变量的存储单元(32位)中。但应注意数值范围不能溢出
  - 如：float f;double d=123.456789e100; f=d;就出现溢出的错误
- 将一个float型数据赋给double变量时，数值不变，有效位数扩展到16位，在内存中以64位(bit)存储



类型转换(续)

- 字符型数据赋给整型变量时，有两种情况：
  - 如果所用系统将字符处理为无符号的量或对unsigned char型变量赋值，则将字符的8位放到整型变量低8位，高8位补零
  - 如果所用系统(如turbo c)将字符处理为带符号的(即signed char)，若字符最高位为0，则整型变量高8位补0；若字符最高位为1，则高8位全补1。这称为“符号扩展”，目的是使数值保持不变

类型转换(续)



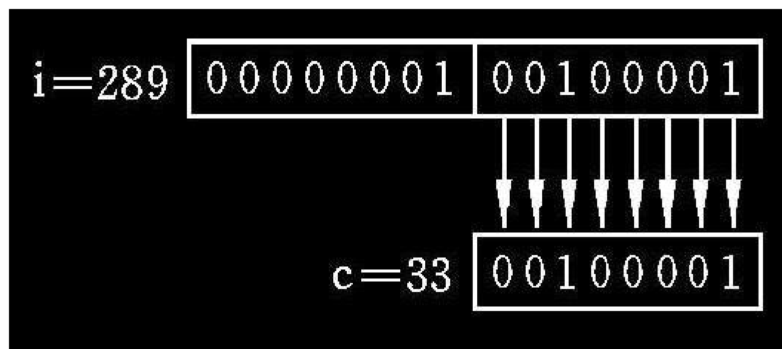
无符号字符

有符号字符

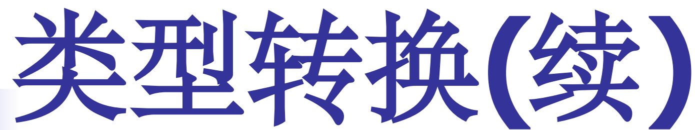
字符型向整型转换

类型转换(续)

- 将一个int、short、long型数据赋给一个char型变量时，只将其低8位原封不动地送到char型变量(即截断)



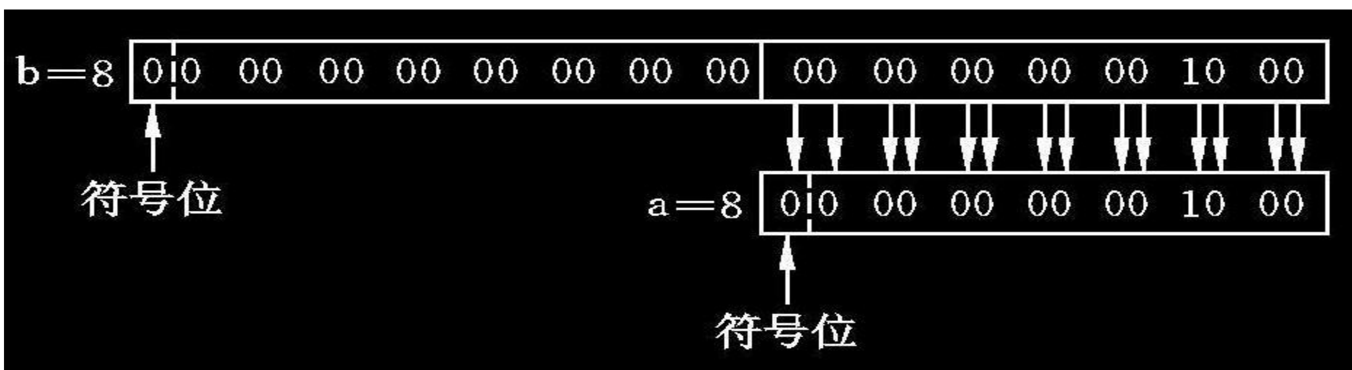
i=289转换为“c”



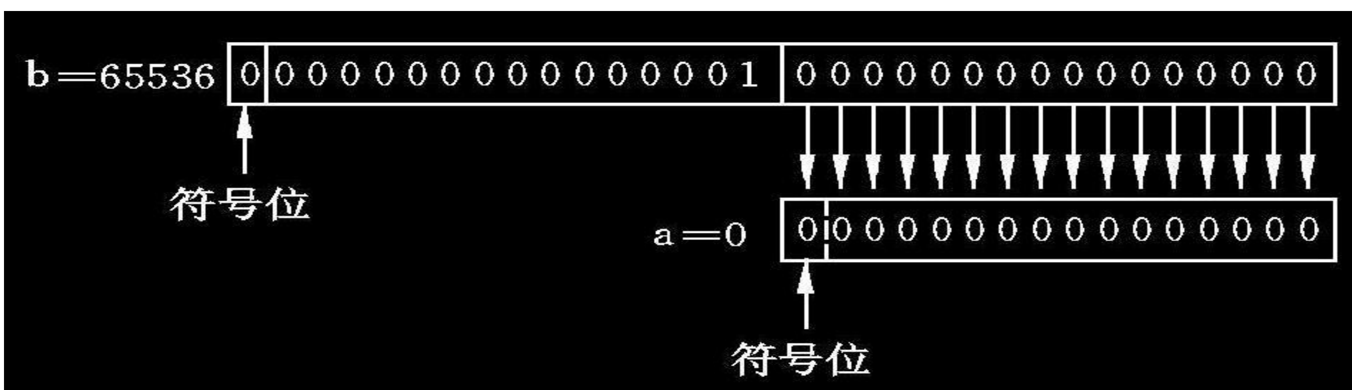
类型转换(续)

- 将带符号的整型数据(int型)赋给long型变量时，要进行符号扩展，将整型数的16位送到long型低16位中，如果int型数据为正值(符号位为0)，则long型变量的高16位补0；如果int型变量为负值(符号位为1)，则long型变量的高16位补1，以保持数值不改变
- 反之，若将一个long型数据赋给一个int型变量，只将long型数据中低16位原封不动地送到整型变量(即截断)

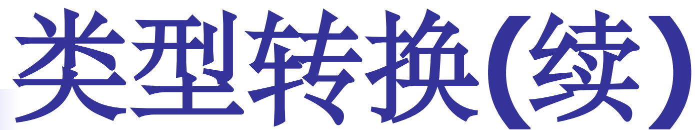
类型转换(续)



8l -> (int) 8



65536l -> (int) 0



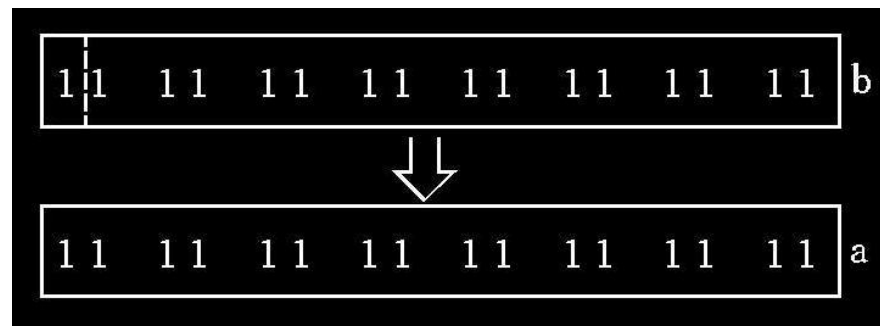
类型转换(续)

- 将unsigned int型数据赋给long int型变量时，不存在符号扩展问题，只需将高位补0即可
- 将一个unsigned类型数据赋给一个占字节数相同的整型变量(例如：unsigned int=>int，unsigned long=>long，unsigned short=>short)，将unsigned型变量的内容原样送到非unsigned型变量中，但如果数据范围超过相应整型的范围，则会出现数据错误
  - unsigned int a=65535; int b; b=a;
  - 结果是：b=-1

类型转换(续)

- 将非unsigned型数据赋给长度相同的unsigned型变量，也是原样照赋(连原有的符号位也作为数值一起传送)

```
main() {unsigned a;
    int b=-1;
    a=b;
    printf("%u", a);
}
```



结果是：65535

类型转换：小结

- 整形（字符型）互相转换
  - 长的向短的转：截取长数的低位（短数位数）
  - 短的向长的转：复制短的数位至长的低位，长的高位补0（短数为正）或1（短数为正）
  - 相同长度：数位复制。因解析不同，数据可能出错
- 实型互相转换
  - 长的向短的转：从长数中截取短的有效位数
    - 有可能会溢出
  - 短的向长的转：复制短数，不足的有效位补0
  - 相同长度：数位复制

提要

- C语言中的数据类型
- 常量与变量整型数据
- 字符型数据
- 实型数据
- 基本数据类型混合运算和类型转换
- 数据运算
- 表达式和表达式语句

运算符的优先级和结合性

- 运算符的优先级
  - 指表达式求值时，如果存在多个运算符，则按运算符的优先级由高到低的次序计算。如大家习惯的“先乘除后加减”，表达式 x-y*z，因y的左侧减号优先级低于y右侧乘号，上式相当于x-(y*z)
- 运算符的结合性
  - 是指运算分量对运算符的结合方向，是从左至右（左结合性），或自右至左（右结合性）
  - 结合性确定了在相同优先级运算符连续出现时的计算顺序，或者说，当一个运算对象两侧运算符的优先级别相同时，运算对象与运算符的结合顺序
    - 算术运算符的结合性是从左至右的，即连续的加减或连续的乘除是从左向右计算。如：a+b-c
    - 赋值运算符的结合性是从右至左的，即连续的赋值运算是从右向左逐个计算赋值。如：a=b=c

算术运算(Arithmetic Operation)

- 种类：+(加)、-(减)、*(乘)、/(除)、%(求余)、+(取正)、-(取负)、++(自增)、--(自减)
  - 两个整数相除结果为整数，如 7/4 的结果为 1
- 求余运算符(%)要求参与运算的两个运算分量均为整型数据，如 5 % 3 的值为 2。一般来说，求余运算所得结果的符号与被除数的符号相同
  - -5 % 3 的值为 -2，5 % -3 的值为 2
  - 在 b≠0 的情况下，a = (a / b) * b + a % b
- 加、减、乘、除和求余运算都是双目（两个运算分量）运算符，结合性都是从左至右的
- 取正 (+) 和取负 (-) 是单目运算符，结合性是从右至左，优先级高于 +、-、*、/、% 等双目运算符

优先级

- 算术运算： +/-；++/--；*、/、%；+、-

前缀 ++ （Prefix Increment)

- 前缀++的一般形式：＋＋变量
  - 基本类型（整型/字符型）变量
  - 指针变量
  - 整型/字符型数组的元素
  - 构造型变量的成分变量
  - 在＋＋变量中，变量的数据类型必须是整型/字符型或某种指针类型
- “++变量”先使变量值增加1个单位，并以新值为变量的结果

前缀 ++

- 若x是满足这种要求的变量，则 ++ x使变量x的值增大1个单位，并以x的新值作为表达式“++ x”的结果，它等价于
  - x = x+1
- 一个单位的含义是
  - 如果 x 是整型的，则 ++x 就是普通的解释：“x 的值比原值增加 1”
  - 如果 x 是指针，并且是指向数组的某个元素，则 ++x 使它指向数组的后一个元素

后缀 ++ (Postfix Decrement)

- 后缀++的一般形式是：变量 + +
  - 表达式“变量 ++”运算结果是该变量的原来值，在确定了表达式结果之后，用与前缀 ++ 相同的方式增大该变量的值 1 个单位
- 前缀 ++ 和后缀 ++ 都能使表达式所对应的变量的值增加 1 个单位，但是它们所代表的表达式的值却不相同
  - 前者是变量增加后的值，后者是变量还未增加的原先值

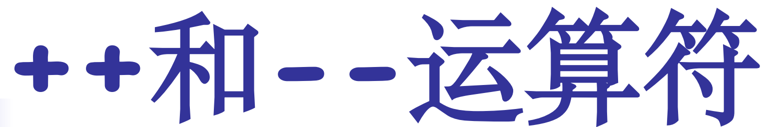
- 例如 i,j 为整型变量，且 i 现有值为 4，求
  - ① j = ++i
  - ② j = i++
- 结果：都使变量i的值变为5，但
  - ①使 j 的值为 5
  - ②使 j 的值为 4

前缀 --

- 前缀--的一般形式是： - - 变量
- 值表达式与前述的意义一样
- 与前缀++相似，不同的只是
  - 前缀 - - 使表达式所对应的变量的值减少(或后退)1个单位
  - 并以变量的新值为表达式“- 变量”的运算结果

后缀 --

- 后缀--的一般形式是：变量 - -
- 表达式与前述的意义一样
- 后缀 - - 作用于表达式对应的变量时，以该变量的值作为表达式“变量 - -”的运算结果，即先取其值为结果，然后用与前缀 - - 相同的方式减少该变量1个单位



++和--运算符

- ++和--运算符仅适用于变量(赋值存储对象)，不能用于常量和表达式
  - 4++ 或 (i+j)++ 都是不合法的
- ++和--是带有副作用的运算符。不要在一个表达式中对同一变量多次使用这样的运算符
  - 如i =4，表达式(i++)+(i++)可能认为它的值为9(4+5)。然而在有些C系统中，它的值为 8；表达式(++i)+(++i)的值为12，因为这些系统
    - 在处理 i++ 时，先使用 i 的原值计算整个表达式，然后再让 i 连续两次自增
    - 处理 ++i 时，在计算表达式值之前，先对 i 执行两次自增，然后才计算表达式

- 函数调用中，多个实参表达式的求值顺序，因从左到右与从右到左的不同，会产生不同的结果
- 例子：设i为4，函数调用
  - printf("%d %d\n", i, i++)
  - 如参数表的求值顺序从左到右，则输出
    - 4　　4
  - 反之，将输出
    - 5　　4

- 因 + 与 ++ (- 与 - - 类似) 是两个不同运算符，对于类似表达式 i+++j会有不同的理解
  - (i++) + j 或 i + (++j)
- C编译的处理方法是自左至右让尽可能多的字符组成一个合法的句法单位(如标识符、数字、运算符等)。因此，i+++j被解释成
  - (i++) + j，不是 i + (++j)
- 自增(++)和自减(- -)运算符结合方向自右至左

关系运算（Relational Operation）

- 关系运算符
  - <(小于)、<=(小于等于)、>(大于)、>=(大于等于)、==(等于)、!=(不等于)
- 关系运算符用于描述对两个值进行关系比较，判定比较条件是否满足
  - 在高级语言中，习惯称条件满足为“真”，不满足为“假”
  - 在*C*中，约定条件满足时，表达式的值为1；条件不满足为 0

- 关系运算符的优先级：
  - <、<=、>、>= 高于 ==，!=
- 如表达式 x > y == c < d，等价于表达式
  - (x > y) == (c < d)
- 关系运算符的优先级低于算术运算符的优先级
  - x > u+v 等效于 x > (u+v)
- 关系运算符的结合方向是自左至右

优先级

- 算术运算： +/-; ++/--; *、/、%; +、-
- 关系运算： <、<=、>、>=; ==, !=

逻辑运算(Logical Operation)

- 运算符： **&&(逻辑与)、||(逻辑或)、!(逻辑非)**
  - 运算符 && 和 || 是双目运算符，要求有两个整型的运算分量，用于连接多个判定条件，构成更复杂的条件判定
  - 运算符 ! 是单目运算符，用于描述对给定条件的否定判定
- **逻辑运算产生的结果也是一个真值或假值**
  - **用1表示逻辑运算结果为真**
  - **用0表示运算结果为假**
- 判定一个运算分量的值为真或假时，**以运算分量的值不等于零为真，值等于0 为假**

逻辑运算优先级

- 逻辑运算符按优先级排列为
  - ！、(&&、||)
- && 和 || 的优先级低于关系运算符的优先级;
- ！的优先级高于算术运算符的优先级
- 逻辑运算符||和&&的结合方向是自左至右
- 而逻辑运算符!的结合方向是自右至左

优先级

- 逻辑运算：!
- 算术运算：+/-；++/--；*、/、%；+、-
- 关系运算：<、<=、>、>=； ==，!=
- 逻辑运算：&&、||

实例

- a > b && x > y 等效于
  - (a > b) && (x > y)
- a != b || x != y 等效于
  - (a != b) || (x != y)
- x == 0 || x < y && z > y 等效于
  - (x == 0) || ((x < y) && (z > y))
- !b || x > y || a > b 等效于
  - ((!b) || (x > y)) || (a > b)
- !a && b || x > y && z < y 等效于
  - ((! a) && b) || ((x>y) &&(z<y))

实例

- 设 a、b 的值分别为 2 和 3，则
  - 表达式!a 的值为 0
  - 表达式 a && b 的值为 1
  - 表达式 !a || b 的值为 1
  - 表达式 !a ||!b 的值为 0
  - 表达式 !a && b 的值为0

逻辑运算真值表

a	b	!a	a && b	a || b
非0	非0	0	1	1
非0	0	0	0	1
0	非0	1	0	1
0	0	1	0	0

关系运算与逻辑运算（1）

- 在算术、关系、逻辑混合运算的表达式中，不同位置上出现的运算分量，应区分哪些是算术运算分量、哪些是关系运算分量和哪些是逻辑运算分量
- 例如：2 > 1 && 4 && 7 < 3 + !0 等效于
  - ((2 > 1) && 4) && (7 < (3 + (!0)))。
  - 从左至右计算：2 > 1 的值为 1；1 && 4 的结果亦为1；1 && 7 < 3 + !0 先计算 !0 ，其值为1；3 + 1 结果为 4；再进行 7 < 4 计算，结果为 0 。最后进行 1 && 0 的运算，得到上述表达式的计算结果为 0

关系运算与逻辑运算（2）

- 逻辑运算的分量也可以是字符型、指针型的。以0和非0判定它们的 “假”或“真” ，求得0或1为结果
- “逻辑与”和“逻辑或”运算符分别有以下性质
  - a && b，当 a 为0时，不管 b 为何值（不再计算 b），结果为 0
  - a || b，当 a 为1时，不管 b 为何值（不再计算b），结果为 1
  - 对于表达式 a && b，仅当子表达式a为非零时,才计算子表达式b
  - 对于表达式 a || b，仅当子表达式a为0时，才计算子表达式b

关系运算与逻辑运算（3）

- 利用以上性质，对于两个条件的逻辑与，如当条件1不成立情况，条件2的值没有意义或不可计算时，逻辑表达式应写成：

  条件1 && 条件2

  避免在条件1不成立情况下，计算条件2。
- 如有条件：y/x > 2且 x != 0，应写成：

  ```
  x != 0 && y/x > 2
  ```

  当 x 为0时，不会计算 y/x。而写成：

  ```
  y/x > 2 && x != 0
  ```

  是不正确的，因为当 x 为0时，不能计算 y/x。对于逻辑或也有类似情况

关系运算与逻辑运算（4）

- 在C中计算连续的逻辑与和逻辑或运算时，不分优先级，顺序从左至右计算
  - 在计算逻辑与时，若左运算分量的值为0，则不再继续计算与运算，以0为与运算的结果
  - 在计算逻辑或时，若有左运算分量的值为1，则不再继续计算或运算，以1为或运算的结果
  - 在顺序计算逻辑表达式的过程中，一旦确定了表达式的最终结果，就不再继续计算
- 如有：int a = 1, b = 1, c = 1;计算 ++a || ++b && ++c时，因++a非0，不必再计算逻辑或的右运算分量子表达式++b&&++c，并立即得到表达式的值为1。因此，该表达式计算后，变量a的值变为2，而变量b和c的值不变，依旧为1

例子

- 编制日历程序需要判定某年是否闰年。每四年一个闰年，但每100年少一个闰年，每400年又增加一个闰年。记年份为 year，则year年是闰年的条件是：
  - (year 能被4整除，不能被100整除)或(year 能被400整除)
- 用逻辑表达式可描述如下：
  - (year % 4 == 0 && year %100 != 0) || year % 400 == 0
- 在C语言的逻辑表达式中，对一个数值的不等于0的判断，可用其值本身代之。上式判断 year为闰年的逻辑算式可简写为：
  - (year % 4 == 0 && year % 100) || year % 400 == 0

赋值运算

- 赋值运算最简单形式：变量 = 表达式
- 赋值运算的执行过程
  - 计算赋值运算符右端表达式
  - 如表达式类型与赋值运算符左边变量类型不同（仅限于基本类型），将表达式值的类型转换成变量的类型
  - 将求得的值赋给变量
- 赋值运算也有结果，经赋值运算后，赋值表达式具有赋值后赋值运算符左边变量同样的类型和值
- 因赋值运算有值，所以可以进一步参与运算，特别是可以再赋值给其它变量

赋值运算（续）

- 赋值运算符的结合性是“自右至左”的，当连续有多个赋值运算时，是从右至左逐个赋值
- 如：x = y = 4.0，是先将值4.0赋给变量y，再赋给变量x
- 如有变量说明：int k; double x;则赋值表达式：x = k = 3.5
  - 是先将实数3.5自动转换成整数3赋给整型变量k，然后又将整数3自动转换成实数3.0赋给实型变量x，所以k的值是3，x的值是3.0

赋值与算术混合运算

- 表达式i=4+(j=7)的值为11，使i值为11，使j值为7
- 表达式i = (j = 3)+(k = 8) 的值为11，使i值为11，使j值为3，k值为 8
- 在程序中，经常遇到在变量现有值的基础上作某种修正的运算，如 x = x + 5.0。这类运算的特点是：
  - 变量既是运算分量，又是赋值对象
- 为避免对同一存储对象的地址重复计算，C语言引入复合赋值运算符，它们是
  - +=、-=、*=、/=、%=、<<=、>>=、&=、^=、|=

复合赋值运算

- x += 5.0 等效于x = x + 5.0
- x *= u + v 等效于x = x * (u + v)
- a += a -= b + 2 等效于a = a+(a=a-(b+2))
- 记θ为某个二目运算符，复合赋值运算 x θ= e,其等效的表达式为 x = x θ (e)
- 当e是一个复杂表达式时，等效表达式括号是必需的。如 y *= a+b 的等效形式是 y = y * (a+b)
- 赋值运算符和所有复合赋值运算符的优先级全相同，都是“自右至左”结合，优先级高于逗号运算符的优先级，低于C语言中其它所有运算符的优先级

优先级

- 逻辑运算： !
- 算术运算： +/-； ++/--； *、/、%； +、-
- 关系运算： <、<=、>、>=； ==，!=
- 逻辑运算：&&、||
- 复合/赋值运算： =、Θ=

条件运算

- 一般形式为 表达式$_1$ ？ 表达式$_2$ ： 表达式$_3$
- 条件运算的计算规则是
  - 计算表达式$_1$的值
  - 如果表达式$_1$的值非0(真)，则计算表达式$_2$，并以表达式$_2$的值为条件运算的结果(不再计算表达式$_3$)
  - 如果表达式$_1$的值为0(假)，则计算表达式$_3$，并以表达式$_3$的值为条件运算的结果(不再计算表达式$_2$)
- 例如：x > y ? x : y。如 x > y 为真，则条件运算取x值，否则取y值
- 条件运算符(?:)的优先级高于赋值运算符，低于逻辑运算符，也低于关系运算符和算术运算符

优先级

- 逻辑运算：!
- 算术运算：+/-；++/--；*、/、%；+、-
- 关系运算：<、<=、>、>=；==，!=
- 逻辑运算：&&、||
- 条件运算：?:
- 复合/赋值运算：=、Θ=

条件运算（续）

- 例如：max = x > y ? x : y + 1 等效于max = ((x > y) ? x : (y + 1))
- 条件运算符的结合性为“自右至左”
- 例如：x > y ? x : u > v ? u : v 等效于 x > y ? x : (u > v ? u : v)
- 条件运算三个运算分量的数据类型可以各不相同。在条件运算中，表达式2与表达式3的类型不一致时，C语言约定在表达式2与表达式3中，类型低的向类型高的转换
  - 如，表达式i > j ? 2 : 3.5，当 i > j 时，条件表达式的值为2.0；否则，为3.5

sizeof 运算

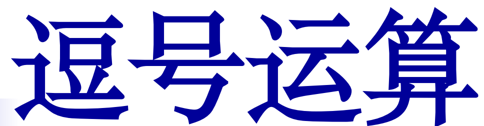
- **sizeof 运算以字节为单位给出其运算对象所需（或所占）的字节数，**习惯称为运算对象的长度。在语义上它是一个整常量，可以用于需要这个常量的任何地方。**sizeof** 运算有两种书写形式：

  sizeof 变量名 或 **sizeof （类型名）**

- 如果有某种类型为**t**的变量**v**，其中**t**可以是系统提供的类型或用户自已定义的类型，可以是简单的，也可以是数组、结构等。则 **sizeof v**就是变量**v**所占的字节数，用它知道变量**v**所占据的内存空间大小。如有 **int j; double x;** 表达式**sizeof j**和**sizeof x**分别是**j**和**x**所占字节数

sizeof 运算

- sizeof(t)是系统为分配一个类型为t的数据对象所需的字节数。如sizeof(int)和sizeof(double)分别是系统为分配一个类型为int和double变量所需的字节数
- sizeof(t)这种形式常用于程序向系统动态申请一个类型为t的数据对象的存储空间时，作为实参给出要申请的动态存储空间的大小



- 逗号运算用于将若干表达式连接起来顺序地逐个计算。连续逗号运算的一般形式为：
  - 表达式1，表达式2，…，表达式n
  - 计算顺序从左到右逐一计算各表达式，并以表达式n的值为连续逗号运算的结果
- 例如表达式：x = (i = 3, i * 2) 使i等于3，x等于6
- 逗号运算符的优先级最低，结合性是“自左向右”

优先级

- 逻辑运算：!
- 算术运算：+/-；++/--；*、/、%；+、-
- 关系运算：<、<=、>、>=；==，!=
- 逻辑运算：&&、||
- 条件运算：?:
- 复合/赋值运算：=、Θ=
- 逗号运算：,

位运算

- 位运算的运算分量只能是整型或字符型数据，位运算把运算对象看作是由二进位组成的位串信息，按位运算，得到位串信息的结果
- 位运算种类：
  - &(按位与)、|(按位或)、^(按位异或)、~ (按位取反)
- 按位取反是单目运算符，其余均为双目运算符
- 位运算符优先级从高到低，依次为~、&、^、|
- ~按自右至左结合，优先级高于算术运算符，其余运算符都是自左至右结合，优先级低于关系运算符

优先级

- 逻辑/位运算：！/~
- 算术运算：+/-；++/--；*、/、%；+、-
- 关系运算：<、<=、>、>=； ==，!=
- 位运算：&、^、|
- 逻辑运算：&&、||
- 条件运算：?:
- 复合/赋值运算：=、Θ=
- 逗号运算：，

按位与运算符(&)

- 将两个运算分量的对应位按位遵照以下规则进行计算：
  0 & 0 = 0， 0 & 1 = 0， 1 & 0 = 0， 1 & 1 = 1
- 即同为 1 的位，结果为 1，否则结果为 0
- 例如，设3的内部表示为00000011，5的内部表示为00000101。则 3&5 的结果为 00000001
- 按位与运算有两种典型用法
  - 取一个位串信息的某几位，如以下代码截取x的最低7位：x & 0177
  - 二是让某变量保留某几位，其余位置0，如以下代码让x只保留最低6位：x = x & 077
  - 以上用法都先要设计好一个常数，该常数只有需要的位是1，不需要的位是0。用它与指定的位串信息按位与

按位或运算符(|)

- 将两个运算分量的对应位按位遵照以下规则进行计算： 0 | 0 = 0， 0 | 1 = 1， 1 | 0 = 1， 1 | 1 = 1
  - 只要有1个是1的位，结果为1，否则为0
- 例如，023(010011) | 035(011101) 结果为037(011111)
- 按位或运算的典型用法是将一个位串信息的某几位置成1。如要获得最右4位为1，其他位与变量j的其他位相同，可用逻辑或运算017(01111)|j。若要把这结果赋给变量j，可写成： j = 017|j

按位异或运算符(^)

- 将两个运算分量的对应位按位遵照以下规则进行计算：0^0 = 0，0^1 = 1，1^0 = 1，1^1 = 0
- 相应位的值相同的，结果为 0，不相同的结果为 1
- 例如，013(001011)^035(011101)结果为026(010110)
- 异或运算的意思是求两个运算分量相应位值是否相异，相异的为1，相同的为0。按位异或运算的典型用法是求一个位串信息的某几位信息的反。如欲求整型变量j的最右4位信息的反，用逻辑异或运算017^j，就能求得j最右4位的信息的反，即原来为1的位，结果是0，原来为0的位，结果是1

按位取反运算符(~)

- 用来求一个位串信息按位的反，即哪些为0的位，结果是1，而哪些为1的位，结果是0
- 例如，~7的结果为0xfff8
- 取反运算可生成与系统无关的常数。如要将变量x最低6位置成0，其余位不变，用代码x = x & ~077实现
- 当两个长度不同的整型数据进行位运算时，将两个运算分量的右端对齐进行位运算。如果短的数为正数，高位用0补满；如果短的数为负数，高位用1补满。如果短的为无符号整数，则高位总是用0补满
- 位运算用来对位串信息进行运算，得到位串信息结果。如以下代码能取下整型变量k的位串信息的最右边为1的信息位：((k-1)^k) & k

移位运算

- 将整型或字符型数据作为二进位信息串作整体移动。有两个运算符：

  << (左移) 和 >> (右移)

- 移位运算是双目运算，有两个运算分量，左分量为移位数据对象，右分量的值为移位位数。移位运算将左运算分量视作由二进位组成的位串信息，对其作向左或向右移位，得到新的位串信息
- 移位运算符的优先级低于算术运算符，高于关系运算符，它们的结合方向是自左至右

优先级

- 逻辑/位运算：! /~
- 算术运算：+/-；++/--；*、/、%；+、-
- 移位运算：<<、>>
- 关系运算：<、<=、>、>=；==，!=
- 位运算：&、^、|
- 逻辑运算：&&、||
- 条件运算：?:
- 复合/赋值运算：=、Θ=
- 逗号运算：，

左移运算符(<<)

- 左移运算将一个位串信息向左移指定的位，右端空出的位用0补充。例如014(01100)<<2，结果为060(0110000),即48
- 左移时，空出的右端用0补充，左端移出的位的信息就被丢弃。在二进制数运算中，在信息没有因移动而丢失的情况下，每左移1位相当于乘2。如4 << 2，结果为16

右移运算符(>>)

- 右移运算将一个位串信息向右移指定的位，右端移出的位的信息被丢弃
- 例如：**12>>2**,结果为**3**。与左移相反，对于小整数，每右移**1**位，相当于除以**2**
- 在右移时，需要注意符号位问题。对无符号数据，右移时，左端空出的位用0补充。对于带符号的数据，如果移位前符号位为0(正数)，则左端也是用0补充；如果移位前符号位为1(负数)，则左端用0或用1补充，取决于计算机系统。对于负数右移，称用0 补充的系统为“逻辑右移”，用1补充的系统为“算术右移”

- 以下代码能说明系统所采用的右移方法：
  - `printf("%d\n\n\n", -2>>4)`
  - 若输出结果为-1，是采用算术右移；输出结果为一个大整数，则为逻辑右移
- 移位运算与位运算结合能实现许多与位串运算有关的复杂计算

例子

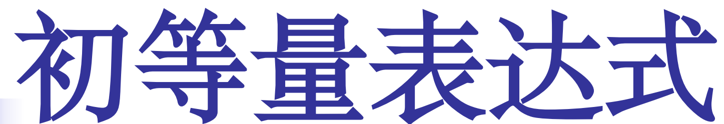
- 设变量的位自右至左顺序编号，自0位至15位，有关指定位的表达式是不超过15的正整数
- `~ (~0 << n)`
  - 实现最低n位为1，其余位为0的位串信息
- `(x >> (1+p-n)) & ~(~0 << n)`
  - 截取变量x自p位开始的右边n位的信息
- `new |= ((old >> row) & 1) << (15 – k)`
  - 截取old变量第row位，并将该位信息装配到变量new的第15-k位
- `s &= ~(1 << j)`
  - 将变量s的第j位置成0，其余位不变
- `for(j = 0; ((1 << j) & s) == 0; j++);`
  - 设s不等于全0，代码寻找最右边为1的位的序号j

提要

- C语言中的数据类型
- 常量与变量整型数据
- 字符型数据
- 实型数据
- 基本数据类型混合运算和类型转换
- 数据运算
- 表达式和表达式语句

表达式（Expression）

- 从表达式的构成形式区分，可分以下几类
  - 初等量表达式
  - 单目运算表达式
  - 双目运算表达式
  - 条件运算表达式
  - 赋值表达式
  - 逗号表达式



初等量表达式

- 初等量又可有以下几种形式
  - 常量，如12,NULL
  - 变量，如x，y
  - 字符串，如"ABCD"
  - 函数调用，如sin(x)
  - 数组元素变量，如a[i]
  - 结构或联合的成分变量，如 s.no
  - 通过指针变量引用结构或联合的成分，如 p->no
  - 带圆括号的表达式(表达式)，如(x+y)

初等量表达式优先级

- 表达式加上圆括号成为初等量，实现用圆括号强制改变运算符的优先级。如表达式(x+y)*z，因(x+y)是初等量，使加法优先于乘法
- 初等量中，使用的运算符有：()、[]、.、->，称它们为**初等量运算符**。因表达式中初等量最先计算其值，它们的优先级是最高的，这些运算符的结合性都是“自左向右”的

优先级

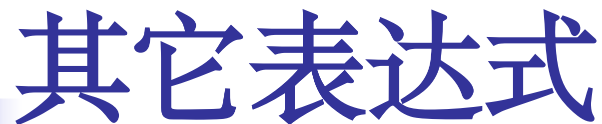
- 初等运算符： ()、[]、.、->
- 逻辑/位运算：! /~
- 算术运算： +/-; ++/--; *、/、%; +、-
- 移位运算： <<、>>
- 关系运算： <、<=、>、>=; ==, !=
- 位运算：&、^、|
- 逻辑运算： &&、||
- 条件运算： ?:
- 复合/赋值运算： =、Θ=
- 逗号运算： ,

单目运算表达式

- 由单目运算符和一个运算分量构成，有以下形式：
  - * 表达式　　(对表达式所指数据存储对象的引用)
  - & 变量　　(求变量所指数据存储对象的地址)
  - - 表达式　　+ 表达式　　! 表达式　　~ 表达式
  - ++变量　--变量　　变量++　　变量--
  - sizeof 表达式　(其中表达式是指数据存储对象，如变量)
    - sizeof(类型名);
  - (类型名)表达式　　(强制类型)
- 单目运算符优先级低于初等量运算符，结合性是“自右向左”
- 变量的表现形式有以下几种
  - 变量名、数组元素变量、结构或联合成分变量、* 表达式，指针所指的结构或联合的成分变量

双目运算表达式

- 一般形式为： 表达式 双目运算符 表达式
- 双目运算符自左向右结合，优先级从高到低如下
  - * / %　　/* 算术运算 */
  - + -　　/* 算术运算 */
  - >> <<　　/* 移位运算 */
  - < > <= >=　　/* 关系运算 */
  - == !=　　/* 关系运算 */
  - &, ^, |　　/* 位运算 */
  - &&,||　　/* 逻辑运算 */



其它表达式

- 条件运算表达式
  - 一般形式为：表达式 ？ 表达式 ： 表达式
  - 条件运算符自右向左结合
- 赋值表达式
  - 一般形式为：左值 赋值运算符 表达式
  - 所有赋值运算符的优先级全相同，自右向左结合
- 逗号表达式
  - 一般形式为：表达式，表达式
  - 逗号运算符的优先级最低，且自左向右结合

优先级

- 初等运算符： ()、[]、.、->
- 单目运算符
  - 地址/指针运算：*/&
  - 逻辑/位运算：! /~
  - 算术运算： +/++、-/--
  - sizeof (变量)、sizeof(类型名)
  - (强制类型)
- 双目运算符
  - 算术运算：*、/、%； +、-
  - 移位运算：<<、>>
  - 关系运算：<、<=、>、>=； ==，!=
  - 位运算：&、^、|
  - 逻辑运算：&&、||
- 条件运算：?:
- 复合/赋值运算：=、Θ=
- 逗号运算：,

```c
#include <stdio.h>
int main()
{  int i = 1, j = 2, k = 3;
   i += j += k;  /* i = 6, j = 5, k = 3 */
   printf("i=%d\tj=%d\tk=%d\n", i, j, k);
   printf("(i<j?i++:j++)=%d\n", i < j ? i++ : j++);
   printf("i=%d\tj=%d\n", i, j);
   printf("(k+=i>j?i++:j++)=%d\n", k += i>j ? i++ : j++);
   printf("i=%d\tj=%d\tk=%d\n", i, j, k);
   i = 3; j = k = 4;   printf("(k>=j>=i)=%d", k >= j >= i);
   printf("\t(k>=j&&j>=i)=%d\n", k >= j && j >= i);
   i = j = 2;    k = i++ - 1;
   printf("i=%d\tj=%d\tk=%d\n", i, j, k); /* i=3 ,j=2,k=1 */
   k += -i++ + ++j;     printf("i=%d\tj=%d\tk=%d\n", i, j, k);
   return 0;
}
```

6 5 3

5

6 6

9

6 7 9

0

1

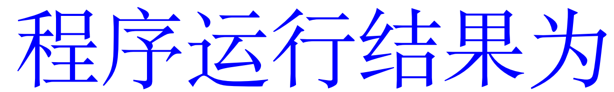
程序运行结果是:

```
i=6     j=5     k=3
(i<j?i++:j++)=5
i=6     j=6
(k+=i>j?i++:j++)=9
i=6     j=7     k=9
(k>=j>=i)=0     (k>=j&&j>=i)=1
i=3     j=2     k=1
i=4     j=3     k=1
```

两点说明

- C表达式 k>=j>=i与数学式子k>=j>=i的区别
  - C表达式 k>=j>=i因运算符 >= 自左向右结合，可写成加括号的等价形式(k>=j)>=i
  - 数学式子 k>=j>=i写成C表达式应该是 k>=j && j>=i
- 运算符++和--在不同系统中实现上的差别
  - 表达式 i / ++i的值，在 TURBO C 中，结果为1。但是，可能在有的系统中，这个表达式的值却为 0
  - 表达式 "i/++i"可以写成 "i/(++i)"。C语言对上述两种可能的选择未作明确规定，由编译程序自行决定。不同的编译程序可能采用不同的方案，这会引起表达式的副作用

```c
#include  <stdio.h>
 int main()
 {  int a, b, c;    a = b = c = 1;    ++a || ++b && ++c;
    printf("a=%d\tb=%d\tc=%d\n", a, b, c);
    a = b = c = 1;    ++a && ++b || ++c;
    printf("a=%d\tb=%d\tc=%d\n", a, b, c);
    a = b = c = 1;    ++a && ++b && ++c;
    printf("a=%d\tb=%d\tc=%d\n", a, b, c);
    a = b = c = 1;    --a && --b || --c;
    printf("a=%d\tb=%d\tc=%d\n", a, b, c);
    a = b = c = 1;    --a || --b && --c;
    printf("a=%d\tb=%d\tc=%d\n", a, b, c);
    a = b = c = 1;    --a && --b && --c;
    printf("a=%d\tb=%d\tc=%d\n", a, b, c);    printf("\n\n\n");
    return 0;
 }
```



a=2 b=1 c=1

a=2 b=2 c=1

a=2 b=2 c=2

a=0 b=1 c=0

a=0 b=0 c=1

a=0 b=1 c=1

表达式语句

- 表达式是求值规则的一种描述，要使计算机按表达式所描述的求值规则作计算，必须将它写成语句，习惯称为表达式语句。在表达式之后接上分号，即 表达式;
- 最典型的表达式语句是赋值表达式构成的语句，如：k = k+2; m = n = j = 3;习惯称为赋值语句
- 另一个典型的应用是函数调用之后接上分号，即 函数调用;
  - 该表达式语句完成特定的函数处理工作

空语句

- 空语句是只有一个分号组成的语句，其形式为 ；
- 空语句是什么也不做的语句。语言引入空语句是出于以下实用上的考虑
  - 一是为了构造特殊控制结构。如循环控制结构需要一个语句作为循环体，当要循环执行的动作已全部由循环控制部分完成时，就用空语句作为循环体
  - 二是在复合语句的末尾设置空语句，用作被转向的位置，以便能用goto语句将控制转移到复合语句的末尾。另外，语言引入空语句使程序的语句序列部分中连续出现多个分号不再是一种错误

goto语句

- 任何语句都可带语句标号
  - 标识符 ： 语句
  - 如 start : i = 0;
- 程序可以用goto语句，将程序的控制转移到指定的语句处继续执行：
  - goto 语句标号；
  - 如goto start;