关于c语言的零零碎碎

double 类型

应用scanf读入转换说明为%lf

应用Printf输出转换说明为%f(感觉应该是可以用%lf的 这样输出相当于强转了一波）

一些类型转换

（数据类型）表达式---结果储存为目标表达式

区别java中的类型强转 目标数据类型 变量一 = （目标数据类型）变量二---其中变量二是被强制转的变量

%-09.2f

0表示如果位数不够用0补齐 否则空补齐

-表示左对齐

9表示最小字段宽度 小数点也算一位

2表示小数点后保留几位

%3d表示输出三位有效整数

C语言中不含有布尔类型，所以用非0表示真，0表示假：判断时1为真，0为假，储存类型为整数型

德摩根定律（逻辑判定）：非a或非b==非(a与b) 非a与非b ==非（a与b）

**请记住!!!!咱就是说%d的%是在前面 \n的\也在前面！！**

Putchar只存一个字符 单引号

Define 对象宏式 define a b 指运行时把a全部替换为b 类似于Java里的常量（编译时被宏替换）

For循环的（）为（定义I ,限制I 的运行次数，i的计数方式）

多维数组 int[4] [3] 表示元素是三个整数的数组 有4个这样的数组

C语言中的函数 就是一个方法（感觉？）

返回值类型 函数名 （形参声明）

C语言的函数引用前需提前声明；在类中创建即定义变量表示该类中所有的函数均可调用

一些声明方便阅读：extern （未创建）函数的非定义声明即原型声明：更像一个抽象方法

Const:不改变数组值 类似final

哨兵查数据：安插一个目标数据在数组末 如果找到数据记录数在数组长度之前 说明有目标存在

Int[a]={0}是对数组的初始化

数据定义---就近原则:优先采用方法内的数据

C中也含有静态变量，成员变量（自动存储期）

整型包括枚举型

sighed int ：0正负整数 （第一位是符号）unsighed：0,正整数---二者存储范围相等 但是无符号存的数据更多 ---显示输出%u %lu

判断char型有无符号：找CHAR\_MIN(在头文件limits.h中)

--计算机以二进制存储u 即将整数转为二进制

sizeof 函数求字符长度: 输出为unsighed 转换说明为%u typedef声明：为数据类型取别名

一些编译器将unsighed别名为 size\_t 即sizeof 的输出结果为size\_t 需说明：有些编译器将其视作unsighed long/short的别名



Sizeof可知道数据类型长度，数组一共占了多少长度或者索引处的元素长度（元素个数=总长度/索引处长度）：sizeof函数过后的类型为size\_t类型

在某个头文件中使用typedef(即给某类型取别名) size\_t在不同编译器的别名不同（有些为u,ulong,ushort）故一般我们输出强转为unsighed

Int+double=double

无符号整数型直接二进制存储 高位------------低位

计算机内编译有符号整数型数据的形式通常有三种

符号+绝对值 反码 补码

最高位代表符号

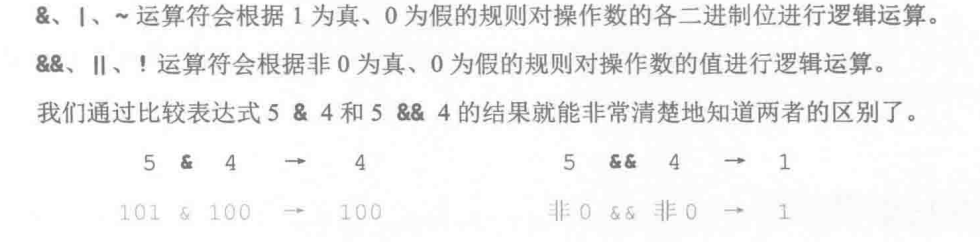
补码：（反转所有位最后+1 保证正负数相加为0

对于1后面16个0的理解 计算机取的负值）

反码：反转所有位 正负数相加所有数字均为1）

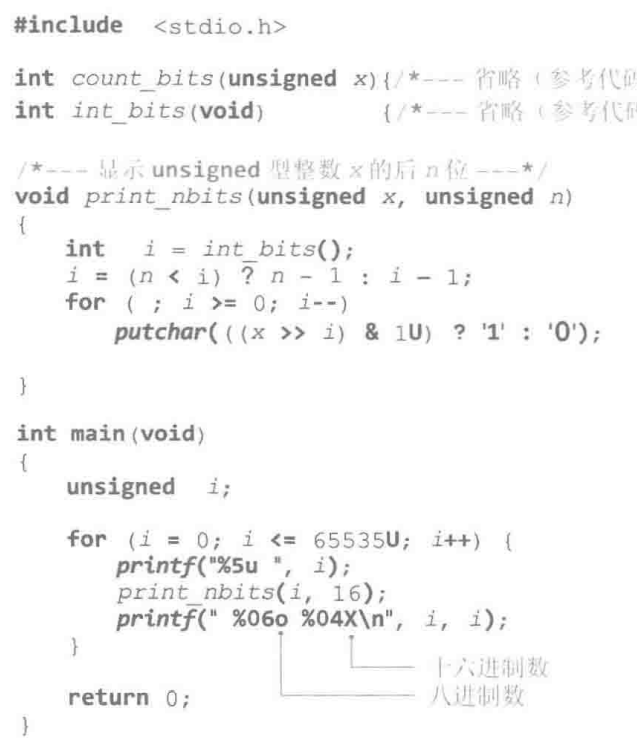
符号和绝对值： 第一位表示正负号（不用浪费一位）

区别主要在于负值



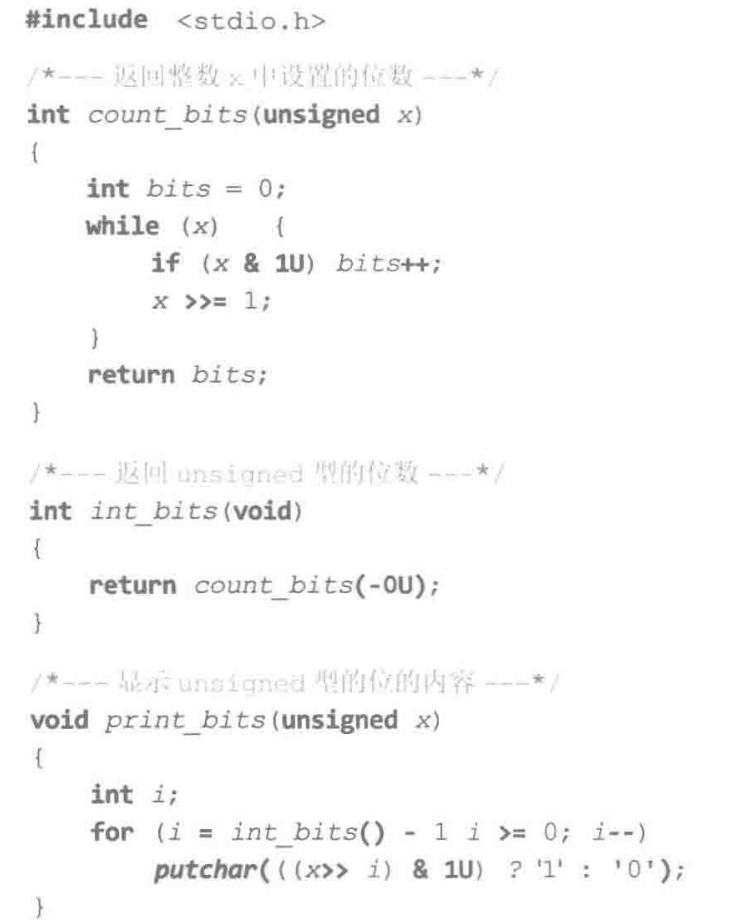
位移运算符：<< 向左移剩下用0补齐\*2^n >>同理可知

不要用位移移负数——储存方式不同

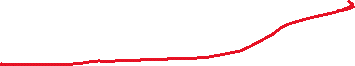


判断是否输入位数：x是输入的数值，n是我想要显示出的二进制位数，小于x位只读n，如果我的x过大，我只读取后面n位。（此题不存在该问题因为限定范围为65535）%o 八进制 %X 十六进制





有多少个1：即可判断位数 x=x>>1 平移后其他位数以0补充



返回位数



~OU:把二进制数中所有的数转为1



三个函数的目的：输出二进制

Print函数：打印

Int函数：计算二进制位数

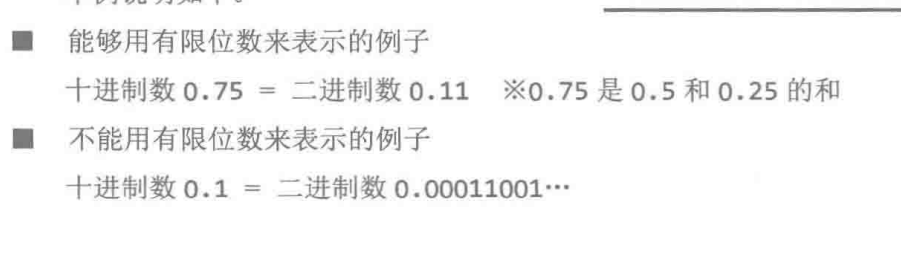
&1U：判断与1—控制输出（在这里隐含了一个电脑默认的二进制转换且为末尾比较（逻辑运算）；否则>>输出整个二进制

使用过程用 “elemtype” 代表所有可能的数据类型，简单明了的概括整体。

在算法中，除特别说明外，规定ElemType的默认是int型。

定义ElemType为int类型你想让它是什么类型自己用typedef重定义就行。

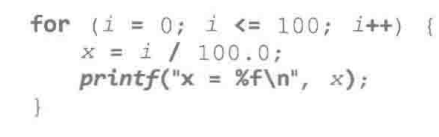
浮点型表示范围由长度和精度共同就决定—储存数据的盒子

精度不够会自动四舍五入 判断精度和长度：科学计数法但二进制

问题：不能表示所有数——有误差

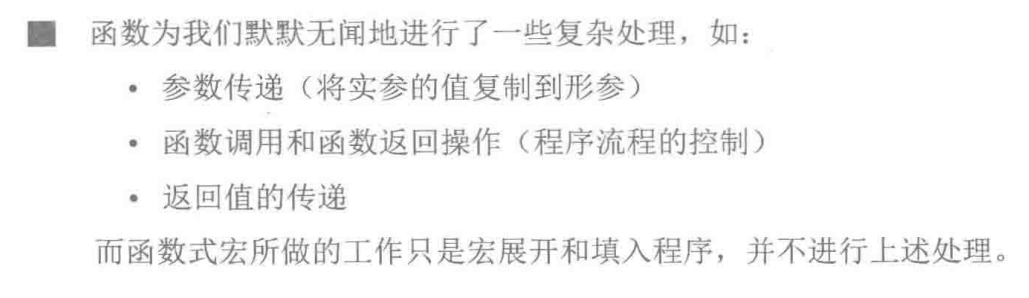
可以加后缀表明数据类型

1.3F 1.3L

以及科学计数法：1.3E-5 = 1.3\*10^-5 C语言提供函数计算

小数累加不要轻易用== 误差 改进：整数控制—防止累加误差

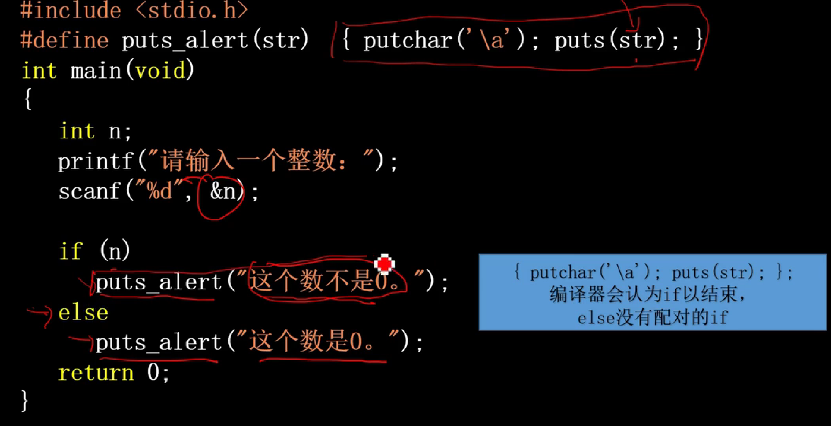
函数式宏 （出现了是枚举+方法！） 直接参数带进去编译时直接替换（区别方法）



如图，函数运算不改变实参，函数式宏要改变（非常死板）



函数名和括号间不能有空格 否则被理解为对象式宏（即把SQR换成x）



此时编译中变成了if{ ; ;};代表结束 else 出现没有if的情况

改：去掉；或者使用逗号运算符

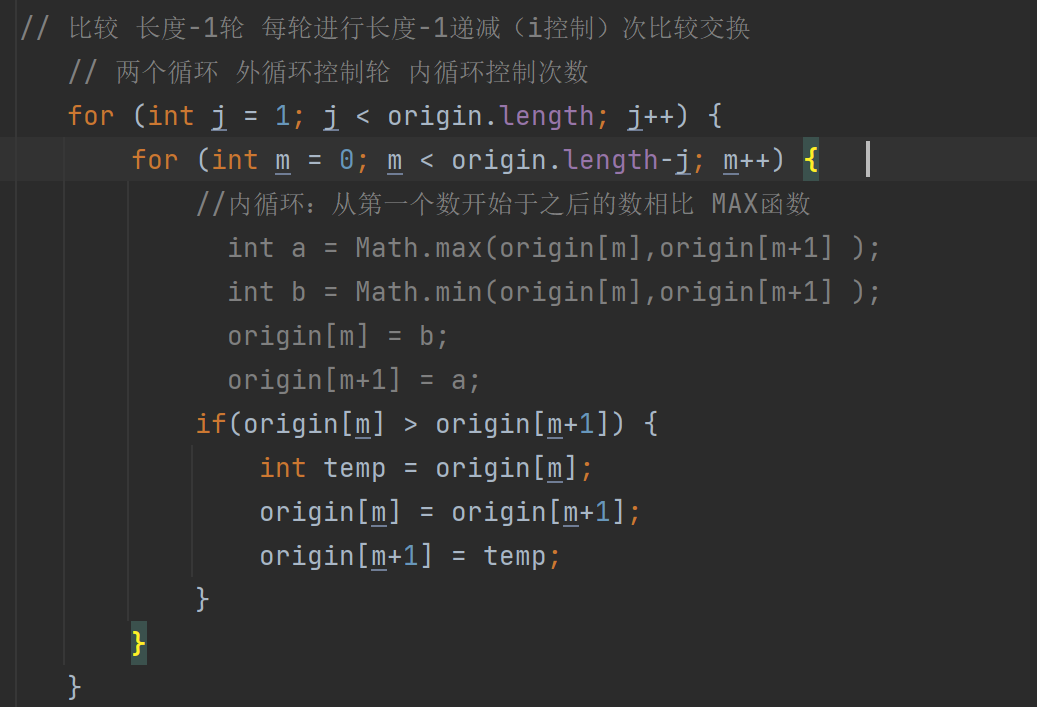
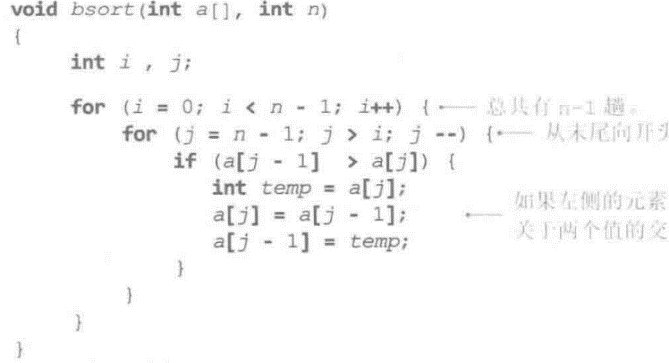
整个函数式宏用括号括起来（防止运算差错）

如图，函数式宏进行一个纯粹的替换

函数式宏中的逗号运算符（语句1，语句2）返回语句2的值---不完整的一个语句 使用处需要加分号 X=(1,2) ----x=2

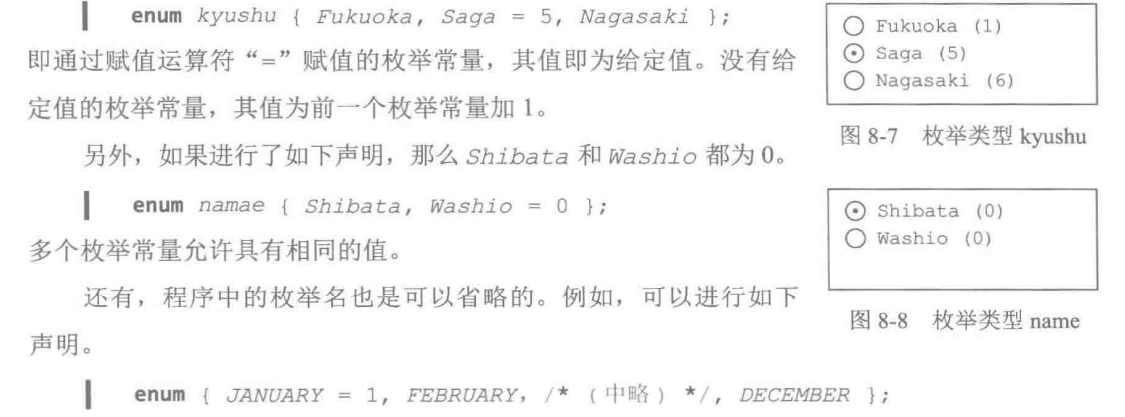
冒泡排序（出现了）

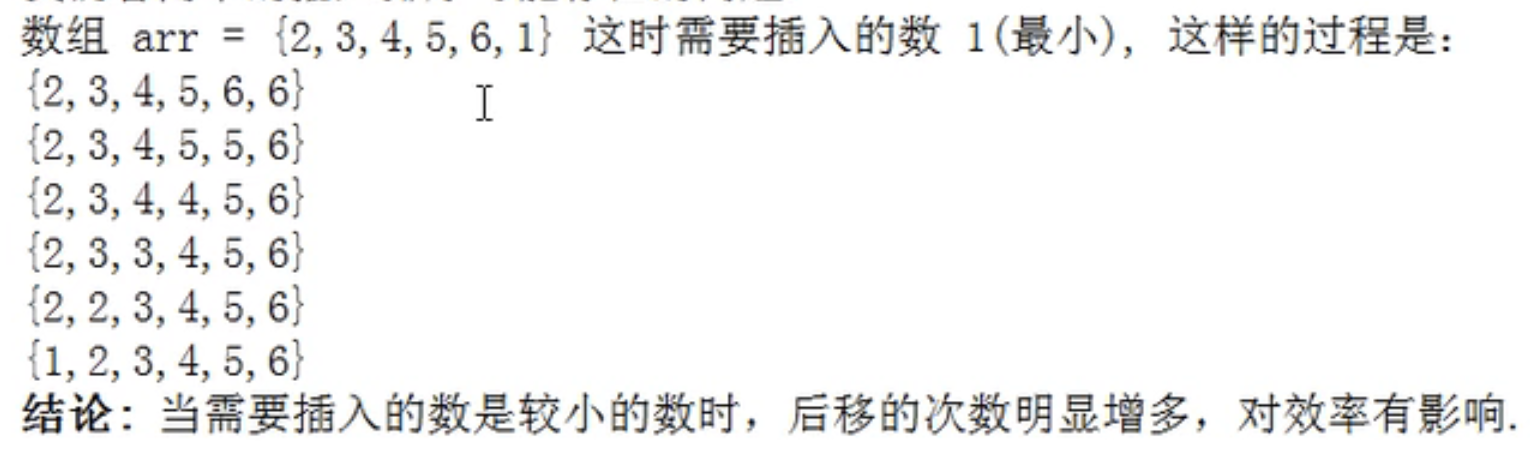
总体思路：每一个数都与前一个数比较按大小交换 两个循环控制



枚举 enum 枚举名称{枚举常量} 默认值为0，1，2，3… 中途定义第二个枚举常量为6 则依次为0，6，7，8…（类似低配版的子父类?）

相当于你创建了一个类型 enum animal (类)可以再创造一个该类型的对象 enum animal animal（创建对象）animal就等于枚举常量之一



插入排序

递归：自己定义自己 自己引用自己 （类似复合函数）

输出和输入字符

getchar() 等着用户输入 将用户输入内容（包括回车符）存入键盘缓冲区 按下回车后

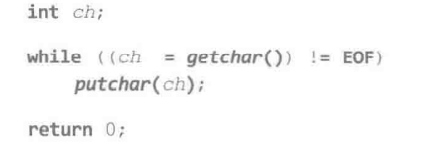
getchar()只有读到换行符才会停止存入开始输出 一次读一个进行运算

实际上在c/c++中EOF并不是一个字符，它仅仅是一个宏定义，其值就是-1。当然了，getchar()函数的返回值也不是字符而是一个整型（读取成功时就返回该字符的ASCⅡ值，失败时就返回一个-1）

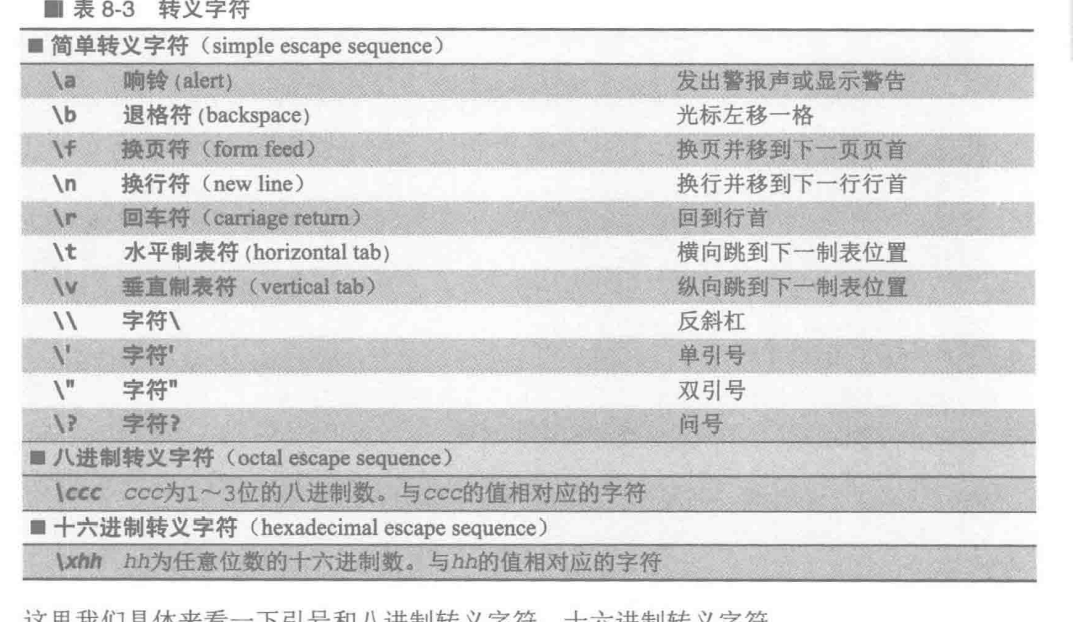
实际上，^Z(ctrl+z)的ASCⅡ值是26，它是一个字符，只不过它含有一个流结束标志的作用

单独出现时系统

如果^Z在可读字符后出现也会认为字符读入，存储ascarII表值



！ASCARII表与实际数之间的运算逻辑



'\063'表示的是字符'3'，因为'3'的ASCII码是30（十六进制），48（十进制），63（八进制）。

'\x41'表示的是字符'A'，因为'A'的ASCII码是41（十六进制），65（十进制），101（八进制）。

字符串

字符与数字转换 巧用表 s[ s[i]-‘0’]

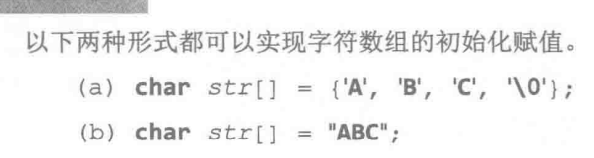
“”括起来的叫字符串字面量 字符串字面量的末尾默认被加上null字符 其值为0 八进制转义字符为’\0’---代表字符编码为null 所有字符串最后都有

字符串转义字符 ：‘%s

字符串存储于字符数组中

除了初始化时 我们不能将数组的初始值或字符串直接赋值

字符数组可以用字符串输出



系统对字符串常量自动加一个'\0'，例如： char str[ ]="liao";和 char \* string="liao"; 则 sizeof(string)=5，输出到控制台为liao

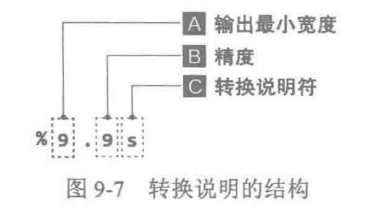
对于没有指定长度的字符数组，例如char str[ ]={'a','b','c','d','e'},系统不会在最后自动添加结束符'\0', 如果puts(str); 将会输出和abcde和一些未知的东西。

对于指定了长度的字符数组（初始化字符个数小于字符数组长度），例如char string[4]={'a','b','c'}; 系统会在最后自动添加结束符'\0'

字符数组并不要求它的最后一个字符为'\0'，甚至可以不包含'\0'。 例如char c[5]={'a','b','c','d','e'};也是合法的，但是用printf("%s",c)，输出数组时会出错。 "%s"格式符对字符串输出时，遇结束符'\0'就停止输出。 而在前面这个字符数组中并没有结束符'\0',所以输完abcde还会继续输出一些未知的东西。 这种情况是能用"%c"输出格式循环输出每个字符。

Scanf的“&“用于找地址 所以当对象是数组等就不用加 字符串+null存储

这是因为scanf只有在遇到\n，也就是是回车时才结束输入，但是遇到空格和tab时就会停止读取

精度为最多显示几位 +-表示左右对齐

字符串数组

数组的数组 char s[ ][6]初始值 [ ]由字符数组个数决定可省略（无初始值不可省略，只有一维数组可省略） [6]指字符数组内部长度 不足的由/0补齐s是字符串名字

Const 表示所传送参数不可更改 有时候我们希望定义这样一种变量，它的值不能被改变

字符串处理—

遍历字符串while(s[i])

打印字符串 1 用printf+%s 2 用putchar 逐个遍历逐个打印----看题目是否有必要单独打印/0

字符长度 int len=0; while (str[i]){len++;} 字符串中字符出现次数 定义一个数组存放次数 循环遍历（int m[] ch s[]）{ while(s[i]){if (s[i]>=’0’ && s[i]<=’9’){m[s[i]-‘0’]++;}}

大小写转换 #include <ctype.h> tosuper转大写 tolower 转小写

指针

值的传递 形参 复制不影响原参数 数组 传的地址会被更改---将基本数据类型也更改

显示对象地址（存储在内存中的编号） %p (pointer 指针) &表示获取对象地址生成一个（生成指向对象的指针）

补充：十六进制英文字母A，B，C，D，E，F分别表示数字10～15。

计数到F后，再增加1个，就进位。

十六进制数的基数是16，采用的数码是0、1、2、3、4、5、6、7、8、9、A、B、C、D、E、F。

数组中对象挨着放 比如整数型数组 相邻元素差值为4（int所占字节）

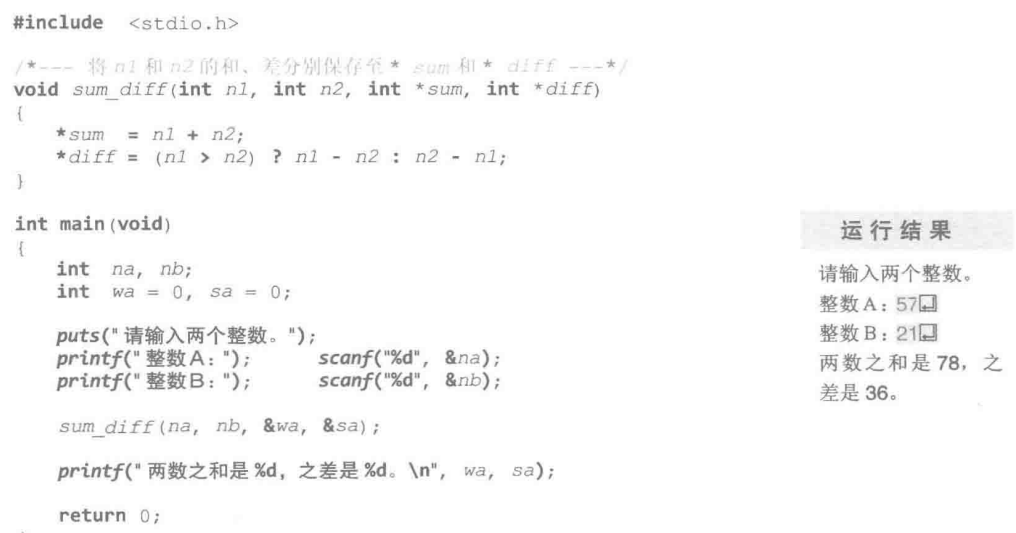
 这是一个指向int类型的指针 即对象类型 \*指针名

指针是用于装地址的盒子 指针名=&对象名---在指针中放入对象地址。\*指针名---访问指针内对象（相当于对象别名，可以对数据处理）

指针：盒子在内存的编号 \*指针名：打开盒子读取里面的东西

传入指针 对其中对象实际操作，修改原参数





此处完成了一个地址存放进指针



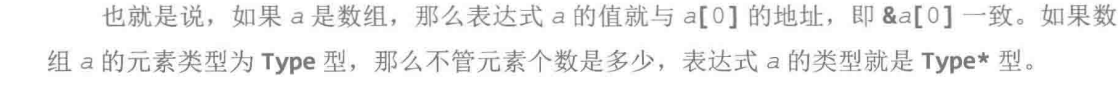
数值互换---地址所指对象改变

空指针 —— #define null

数组和指针

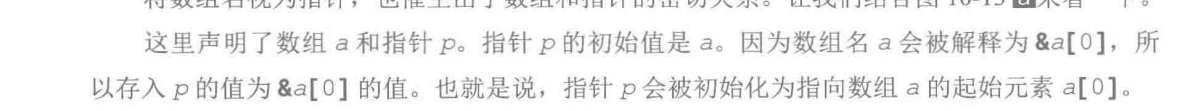
指针 指向数组 \*p

数组名原则上会被解释为指向该数组起始元素地址（指针名=&对象名---在指针中放入对象地址）但他没有另辟空间

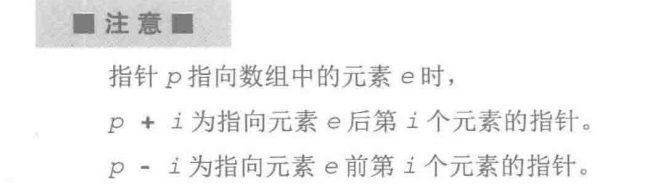




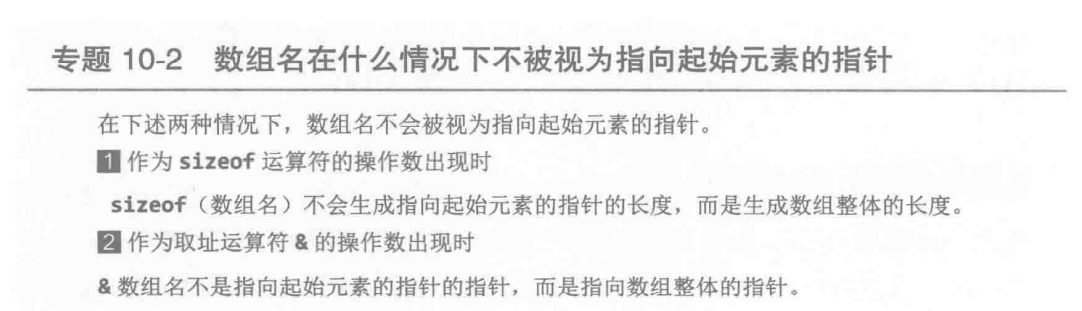




数组名一般被解释为指向该数组起始元素的指针



p+1 地址相当于往后推一位 +sizeof（数组类型）--（元素间隔） 然后数组的元素挨着的p+i=a[ i ]



Int a[ ]={0,1,2,3,4}

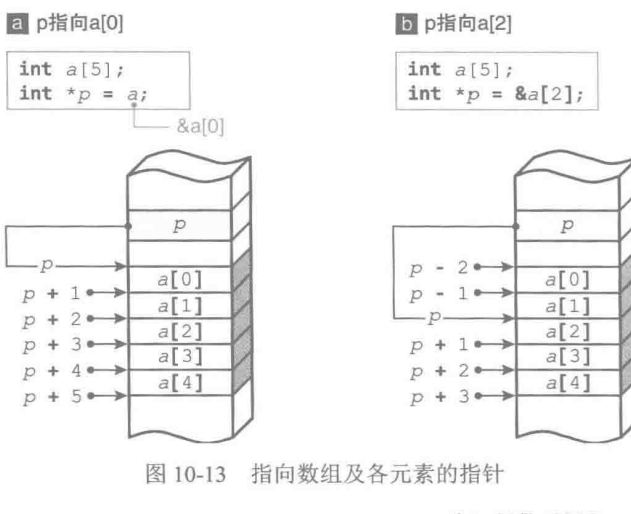
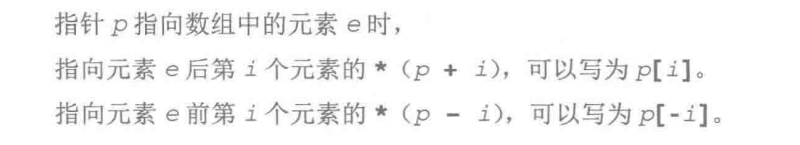
Int \*p = a（第一个元素的地址）; 等价于 Int \*p=&a[0];

p是指针变量，用来存放地址，此指针变量指向的内存地址（地图）

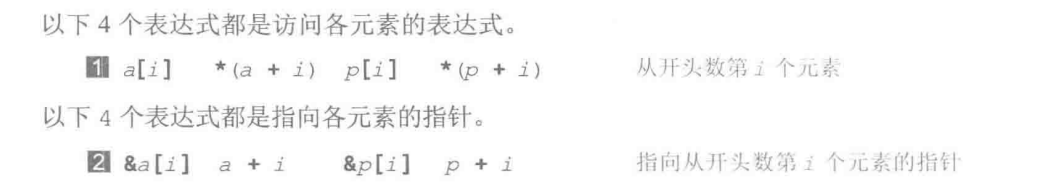
你可以认为是上面所说的盒子的号码，“ \* ”是解引用操作符，你可以把它理解成打开盒子，\*p就是打开p号盒子，用里面的数据。一般是一个和指针类型一致的变量或者常量。（比如一个整数）

&p就表示编译器为变量p分配的内存地址，而因为p是一个指针变量，这种特殊的身份注定了它要指向另外一个内存地址，程序员按照程序的需要

让它指向一个内存地址，这个它指向的内存地址就用p表示。而且，p指向的地址中的内容就用\*p表示。







比如 \*&a[0] &a[0]找到a0的位置 \*&a[0]取地址中的内容 其实就是取a2的值

给指针赋值 只能用地址

可以这样理解\* 1。用于定义指针：类型+\* 2.用于取指针里的值

定义数组时，向编译器申请一段连续的空间。

指针[ i ] 表示以指针当前位置为起点，向后跳过i个该指针所指类型的大小的空间。

p[q] 等价于 \*(p+q)，若pq都不是指针则会报错! i[arr] 等价于 \*(i+arr) 等价于 \*(arr + i) 等价于 arr[i]

数组在函数中传递实际为指针本身地址 比如我定义数组a a本身是一个&a[0]，是a[0]的地址也可以理解为指向a[0]的指针 我将这个地址（指针）传给函数中的定义的数组v v同样也是一个放地址的东西或者理解为a指针为一个变量，将&a传给&v

关于sprintf

sprintf最常见的应用之一莫过于把整数打印到字符串中，所以，spritnf 在大多数场合可以替代itoa（把一个整数转换为字符串）。如：

把整数123打印成一个字符串保存在s中。

|  |  |
| --- | --- |
| 1 | sprintf(s, "%d", 123); // 产生"123" |

在格式串内部使用一些以“%”开头的格式说明符（format specifications）来占据一个位置，在后边的变参列表中提供相应的变量，最终函数就会用相应位置的变量来替代那个说明符，产生一个调用者想要 的字符串。

最开始的第一个变量通过后面变量对第二个变量的补充输出

Strcmp 比较两个数组是否一致

Strcpy 复制数组

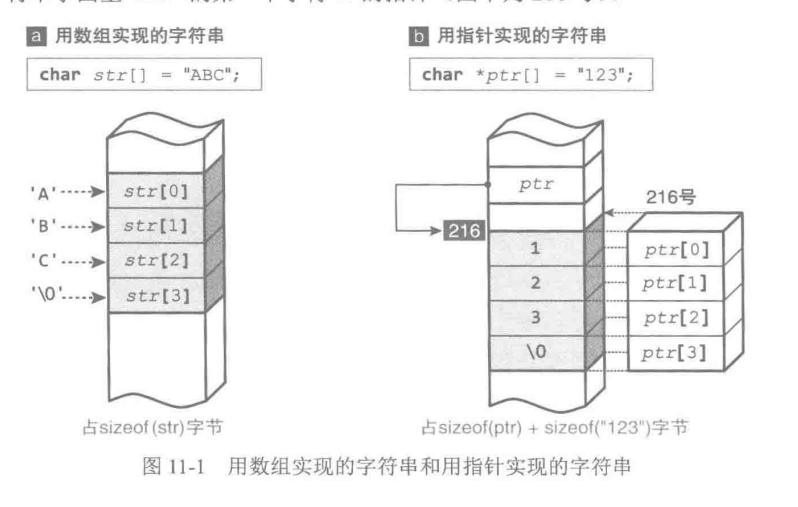
Strlen 获取字符串长度 while(\*s++) len++

| **情况** | **自增对象** | **自增(前置or后置)** |
| --- | --- | --- |
| \*s++ | 指针 | 后置自增 |
| \*(s++) | 指针 | 后置自增 |
| (\*s)++ | 指针内容 | 后置自增 |
| ((\*s)++) | 指针内容 | 后置自增 |
| \*++s | 指针 | 前置自增 |
| ++\*s | 指针内容 | 前置自增 |

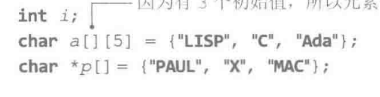
数组实现字符串 char s[4]={‘A’,’B’,’C’,’\0’} char s [4] =”ABC” sizeof 就是数组长度

指针实现字符串 \*str=”ABC” sizeof 是指针长度 指针str和“ABC”分别存储再不同内存

S ,str均指向字符串的首位指针 s本身没有位置 不可以直接修改值



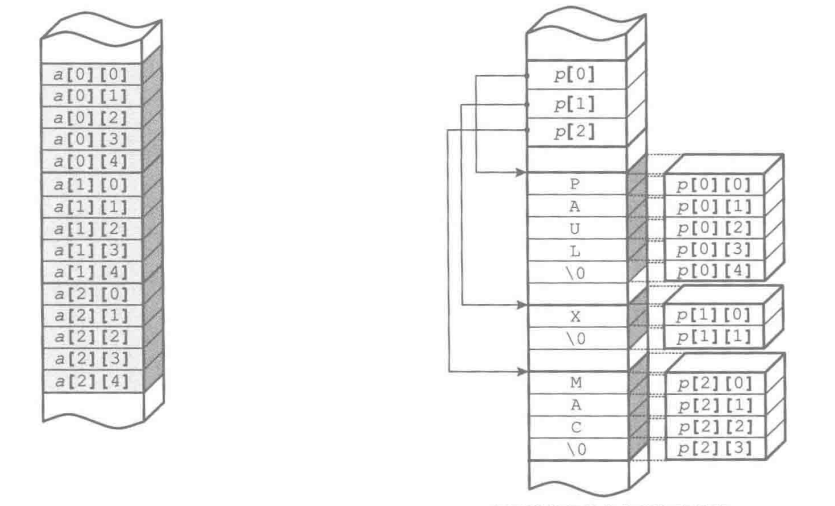
字符串数组



第一个是一个二维数组 用数组定义字符串

第二个是一个指针数组 char\* p[ ] 修饰p的为数组 最终表现形式数组 里面每一个都是指针p[1],p[2],p[3]分别指向字符串第一位

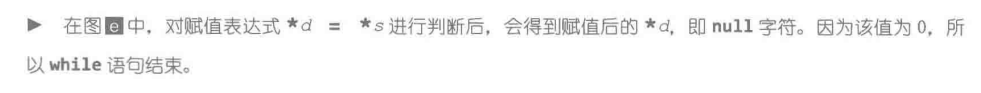
区别指针数组 数组指针 后者char (\*p)[ ] 先定义一个指针指向数组





一些编译器不允许通过指针更改指针内部的数据 复制后的内容长度可能超过初始指向数组长度 预留空间不足





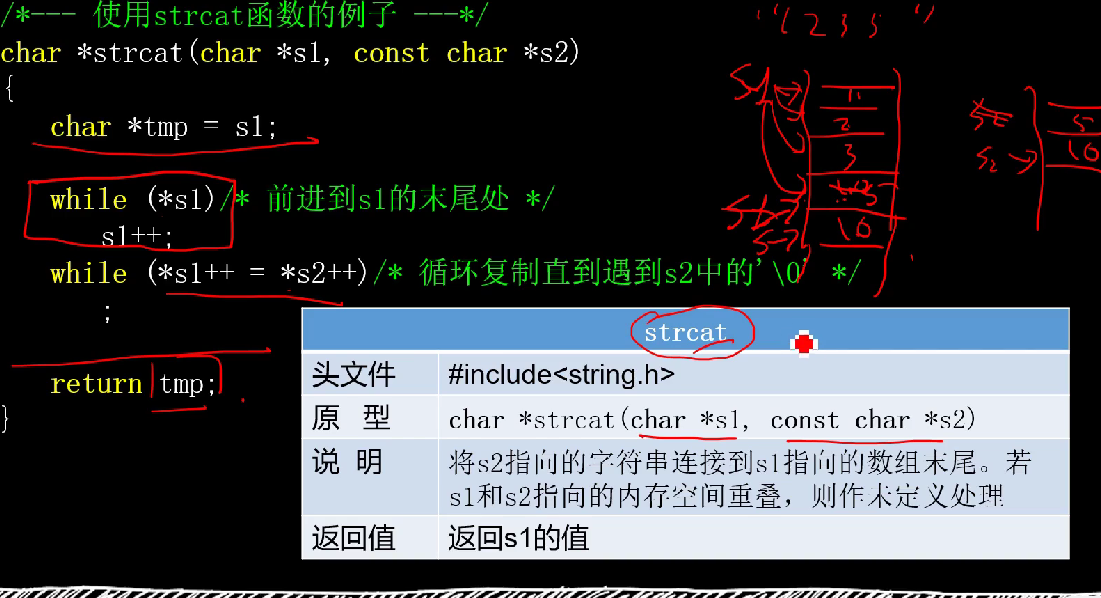
对赋值表达式的结果进行判定

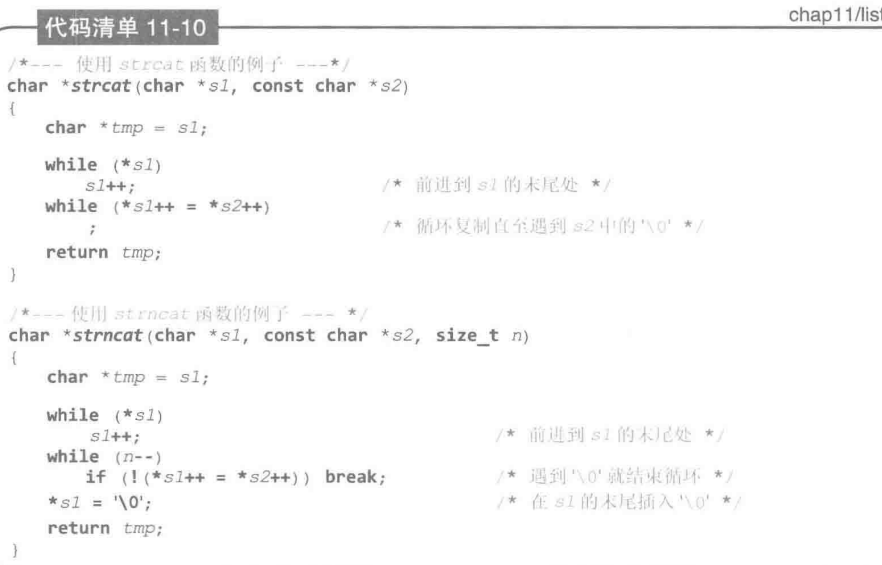
字符串函数库#include<string.h>

Strncpy(char \*s1,const char\*s2, sixe\_t n) 复制到n为止



Strcat 链接字符串 Strncat s2最多拿n 个





Strcmp :相等返回0 s1>s2返回正整数 s1<s2 返回负整数

Strncmp:比较前面n个大小

头文件：<stdlib.h>

atoi字符串转为整数