

C语言与程序设计

The C Programming Language



第2章 基本数据类型与表达式

华中科技大学计算机学院

黄宏

第2章 基本词法语法规则与程序元素

本章重点

2.3 标识符、关键字及分隔符

2.4 基本数据类型

2.5 常量与变量

2.6 运算符和表达式

2.7 位运算符和位表达式

2.8 类型转换

2.9 枚举类型

(2.1、2.2, 2.10了解)



2.1 字符及词法元素

2.1.1 字符集

C源程序由字符序列构成，其字符集为：

- 英文字母：**a~z** 和**A~Z**
- 数字字符：**0~9**
- 特殊字符：**! “ # % & ‘ () * + , - . / :**
 ; < > = ? [] \ ^ _ { } | ~
- 空白字符：空格、换行符、水平制表符（**HT**）、
垂直制表符（**VT**）、换页符（**FF**）

2.1.2 词法元素

词法元素称为记号(token)，记号是程序中具有语义的最基本组成单元。记号共分5类：标识符、关键字、常量、运算符和标点符号。编译器从左至右收集字符，总是尽量建立最长的记号，即使结果并不构成有效的C语言程序。相邻记号可以用空白符或注释语句分开。



词法分析(记号)举例

- 例2.1 `sum=x+y`

分解成`sum`、`=`、`x`、`+`和`y` 共5个记号。

- 例2.2 `int a, b=10;`

分解成`int`、`a`、`,`、`b`、`=`、`10`和`;` 共7个记号

- 例2.3 `x++++++y`

分解成`x`、`++`、`++`、`+`、`y` 共5个记号

2.3 标识符、关键字及分隔符

2.3.1 标识符

- 标识符是用来标识用户定义的常量、变量、数据类型和函数等名字的符号。其命名规则:
- 以一个字母(a~z, A~Z)或下划线(_)开头, 后跟字母、下划线或数字(0~9)

例: K, _id, month, time1



20_sum, not#me, void





注意

- 大小写字母表示不同意义。
- 不能使用类似 **int** 和 **void** 这样的**C**关键字为自己的对象命名，也要避免使用**C**程序库中函数和常量的名称，例如 **scanf** 。
- 良好的编程风格是选择有助于记忆且有一定含义的标识符，这样可增强程序的可读性和程序的文档性。

2.3.2 关键字

- 是被系统赋予特定含义并有专门用途的标识符，不能作为普通标识符，但可以作为宏名。

int 、 void、 return、 while、 if

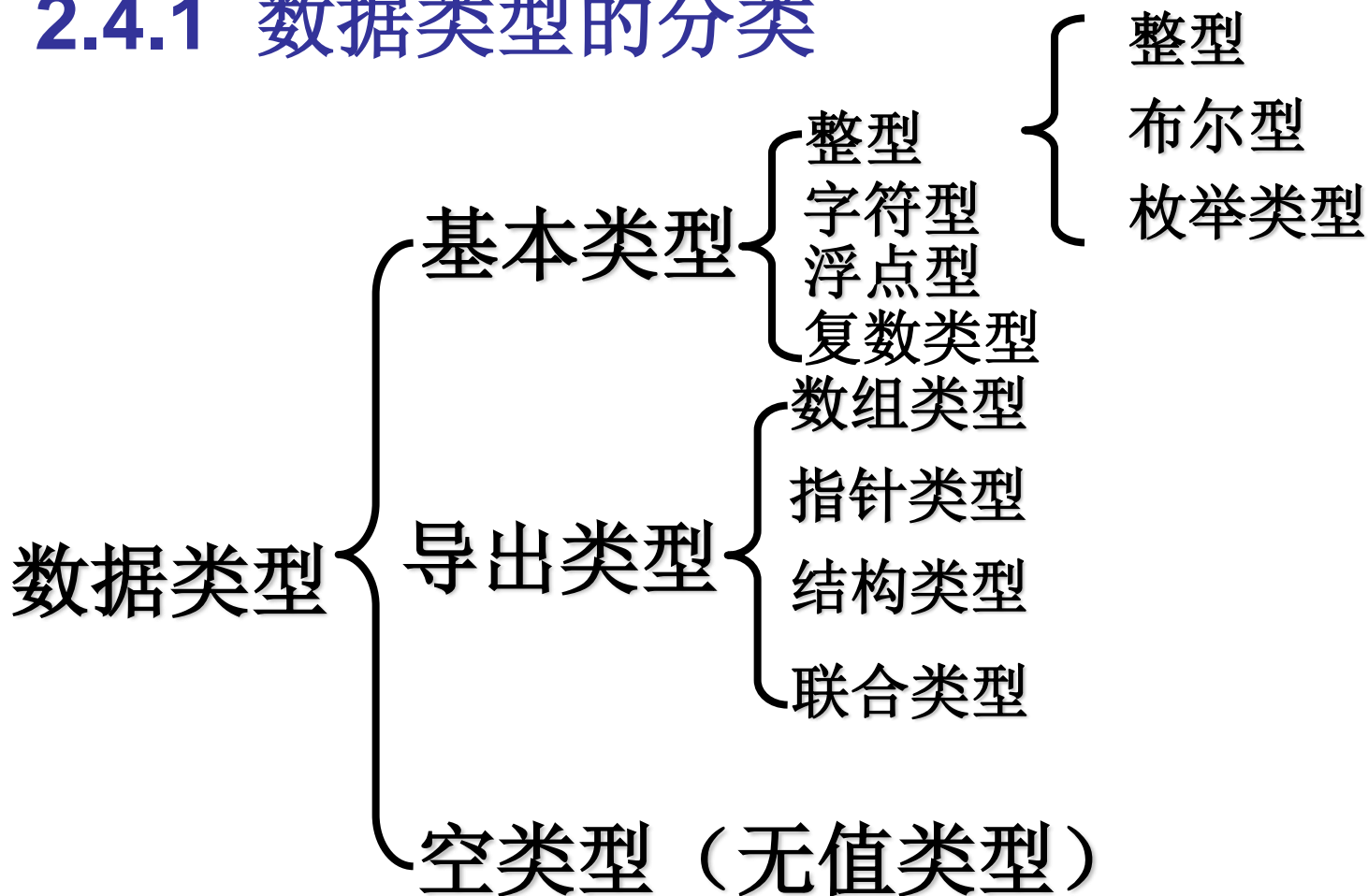
- 小写
- 见教材表2.1

2.3.3 分隔符

- 分隔符统称为空白字符(包括空格符、制表符、换行符、换页符及注释符)，在语法上仅起分隔单词的作用。
- 当程序中两个相邻的单词之间如果不用分隔符就不能区分开时则必须加分隔符（通常用空格符）。
- 例如，`int x,y;` 不能写成 `intx,y;`
能写成 `int x , y ;`

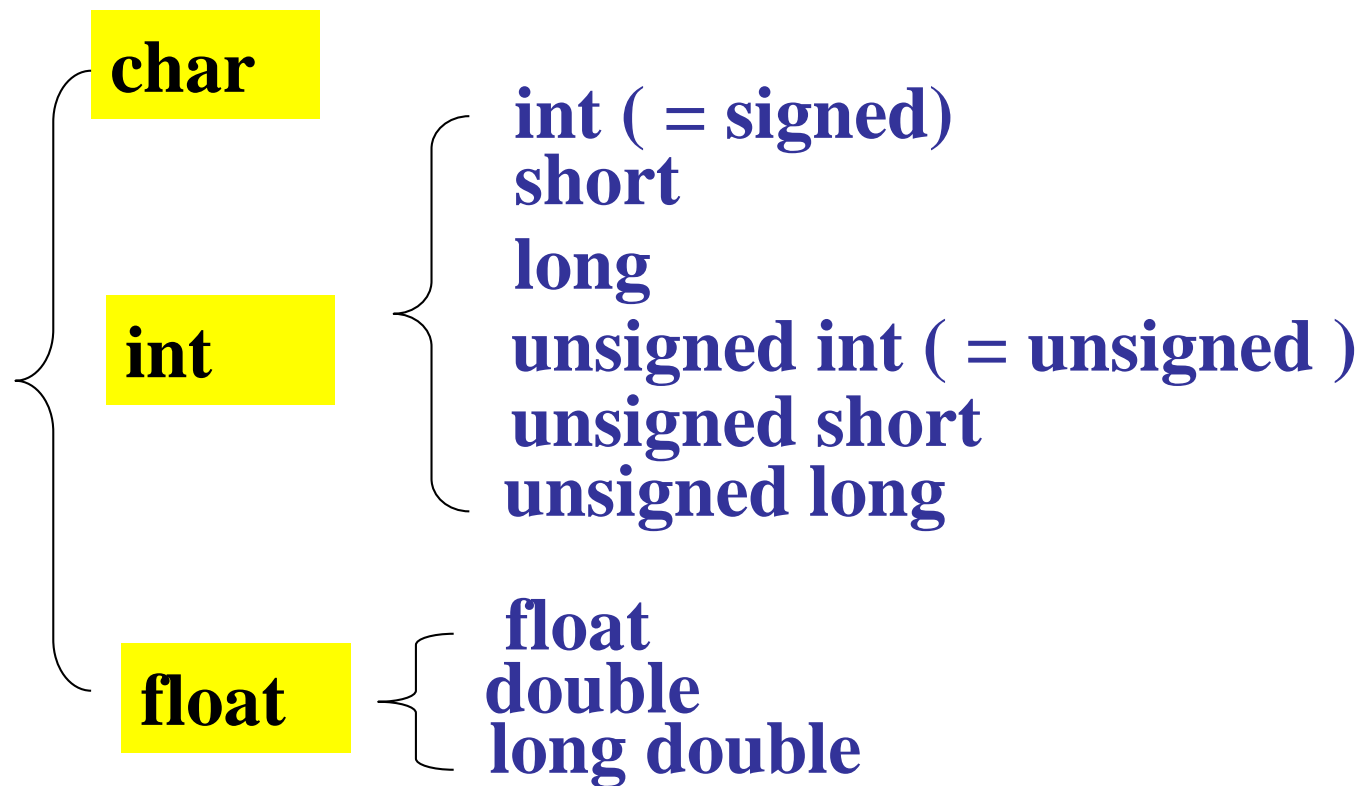
2.4 基本数据类型

■ 2.4.1 数据类型的分类



2.4.2 基本类型的名字

- 本小节介绍字符型、整型、浮点型 (表2.4)



2.4.3 字符类型char

- char的存储长度是一字节。
- 多数系统中char与signed char同(-128~127).
- 字符数据以ASCII码存储在内存中。
- 在不要求大整数的情况下，可用字符型代替整型。

2.4.4 整型类型

- **int**型值存储在一个机器字中。
- 假设字长为2B，**int**取值范围为-32768~32767，**unsigned**取值范围为0 ~ 65535
- 下面的代码是否正确

```
#define BIG 30000
```

```
int main(void)
```

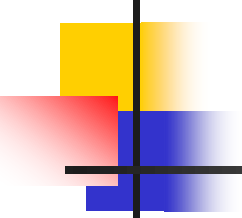
```
{ short x,y,z;
```

```
  x=y=BIG;
```

```
  z=x+y; .....
```

```
}
```

/* 短整数溢出 */

- 
- 程序员必须时刻保证整数表达式的值在合理范围内。
 - 引入**short**和**long**的目的是为了提供各种满足实际要求的不同长度的整数。**int**通常反映特定机器的自然大小，**short**一般为2B，**long**一般为4B。因此，当关心存储时，用**short**；当需要较大的整数值时，用**long**。

2.4.5 浮点类型

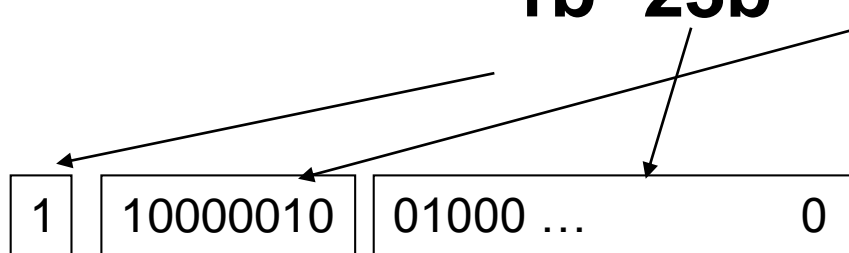
- 一个浮点数N可表示为：
$$V = (-1)^s \times M \times 2^E$$

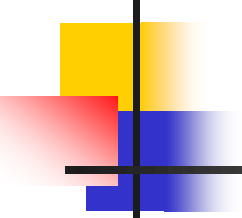
\downarrow
数符
1b

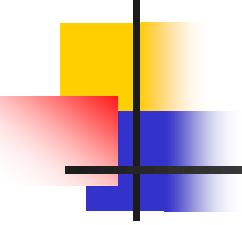
\downarrow
尾数
23b

\downarrow
指数
8b

$$-10.0 = -1.010 \times 2^3 =$$



- 
- 尾数所占的位数决定值的精度，指数所占的位数决定值的范围。
 - **float**占4B，其中符号1b，指数8b，尾数23b，其精度大约为7位，范围约 $10^{-38} \sim 10^{+38}$ 。
 - **double**占8B，其中符号1b，指数11b，尾数52b，其精度大约为15位，范围约为 $10^{-308} \sim 10^{+308}$ 。
 - 很多编译器将**long double**处理为**double**，在某些系统中，它占用10或12B。很少被使用。

- 
- 浮点数的表示可能只是近似的。其值与表示法之间的差称为“可表示误差”。
 - 计算也可能造成可表示误差。不能使用 `==` 和 `!=` 运算符比较浮点数据。
 - 可以用两个数值之差同一个预定的小正数 **epsilon** 比较的方法解决这个问题。



2.5 常量与变量

2.5.1 文字常量

1. 整型常量

有三种表示方法（通过前缀字符区分）：

- 十进制：无前缀
- 八进制：前缀为0
- 十六进制：前缀为0x或0X时。

例如，31可写成037，也可写成0x1f或0X1F



整型常量可以带有**后缀**，用以指定其类型：

- 字母**u或U**表示unsigned
- 字母**l或L**表示long
- 字母**ul或UL**表示unsigned long
- 字母**ll或LL**表示long long (C99)
- 字母**ull或ULL**表示unsigned long long (C99)
- 无后缀时，**一般表示int**
- 当常量值超出指定类型的范围时，其实际类型取决于数值大小、前缀等，确定类型的规则很复杂，在标准化前的C语言、C89和C99中各不相同

2. 浮点型常量

有两种表示方式:

(1) 带小数点的**十进制数形式** (可以小数点开头, 也可以小数点结尾)

如**23.7**, **14.**, **.126**

(2) **指数形式** (科学计数法)

将指数部分跟在尾数部分后面。尾数部分的书写规则与第一种相同, 但可以没有小数点, 指数部分 **e(E)±n**, 代表 **10±n**。

如 **45e-3 = 45×10⁻³**, **.15e5 = 0.15×10⁵**。



可以使用**后缀**来指定其类型

- 无后缀: **double**,
- 后缀**f或F**: **float**,
- 后缀**l或L**: **long double**。

3. 字符常量

(1) 用单引号包含的一个字符是字符常量

(2) 只能包含一个字符



‘a’ , ‘A’ , ‘1’

‘abc’ 、 “a” ❌



转义序列

⑩ 以\开头的特殊字符称为转义序列,有两种形式:

⑩ 一种是“字符转义序列”，即反斜线后面跟一个图形符号，用于表示字符集中的非图形符号和一些特殊的图形字符。

\n 换行

\t 水平制表符

\\ 反斜杠

\ ' 单引号

\ “ 双引号

\0 空字符

\? 问号

源程序\ex2 4.c



数字转义序列

- 转义序列的另一种是“数字转义序列”，即
 - `\ooo`（1~3个八进制数字）
 - `\xhh`（1~2个十六进制数字）
- 例如，
 - `'A'`、`'\101'` 和 `'\x41'` 字符A;
 - `'\t'`、`'\11'`、`'\011'`、`'\x9'`和`'\x09'` 水平制表符

4. 字符串常量

- 写成用一对双引号括住0至多个字符的形式。

"string\n"

/* 包含7个字符的字符串 */

""

/* 包含0个字符的空字符串 */

字符串中的单引号可以用图形符号表示，但双引号和反斜线必须用转义序列表示。例如：

"3'40\""

/* 包含5个字符的字符串: 3'40" */

"c:\tc"

/* 包含4个字符的字符串 */

"c:\\tc"

/* 包含5个字符的字符串 */

如何将一个较长的字符串写成多行？

有两种方法：

(1) 行连接：在前一行的末尾输入续行符（\）再换行。

```
"Hello,\
```

```
how are you" /* 换行后应紧靠行首 */
```

(2) 字符串连接：将字符串分段，分段后的每个字符串用双引号括起来。

```
"Hello, "
```

```
"how are you" /* 换行后不必紧靠行首*/
```

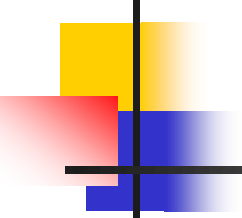


‘a’与 “a”有何区别？

- ‘a’：字符常量，占1 B内存空间
“a”：字符串常量，占2B内存空间

a	\0
---	----

- 存储时，系统自动在后面补上\0（空字符，ASCII值为0，作为字符串结束标志）
- 字符串的存储长度比字符串的实际长度大1



求字符串的长度

```
#include <stdio.h>
```

```
int main( )
```

```
{
```

```
    char s[20]="world"; /* 用于保存串 */
```

```
    int i; /* 计数器变量，用于保存串的长度 */
```

```
    i=0; /* 计数器变量清零 */
```

```
    while(s[i] != '\0')
```

```
        ++i;
```

```
    printf ( "%d", i) ;
```

```
    return 0;
```

```
}
```



求字符串的长度

将“求字符串的长度”定义成函数

\源程序\ex2 5.c

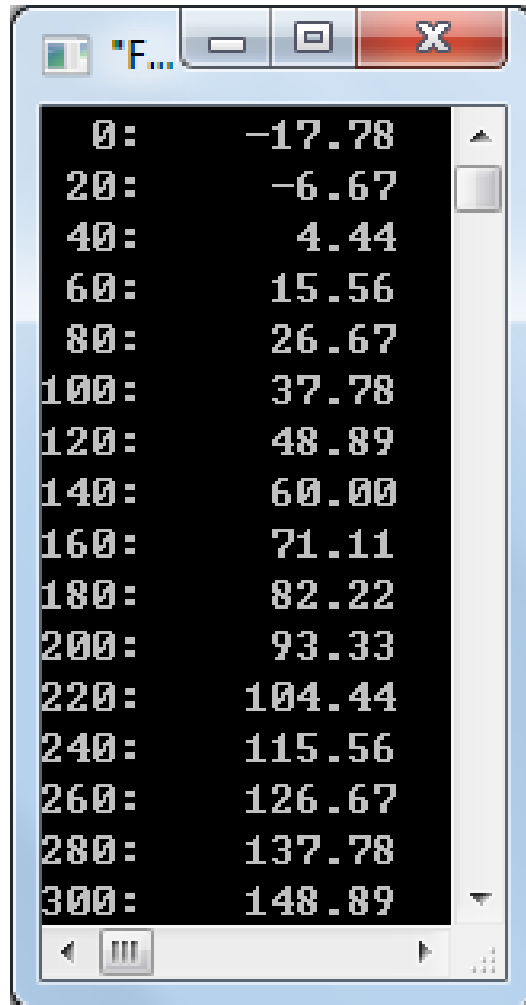
2.5.2 符号常量

用一个标识符表示一个常量.

C语言中有三种定义符号常量的方法:

- (1) 用`#define`指令（常用）
- (2) 用`const`声明语句（只读变量、常变量）
- (3) 用枚举类型（在2.9节介绍）

例2.6 打印华氏和摄氏温度对照表， 温度转换公式为： $^{\circ}\text{C}=(5/9)(^{\circ}\text{F}-32)$

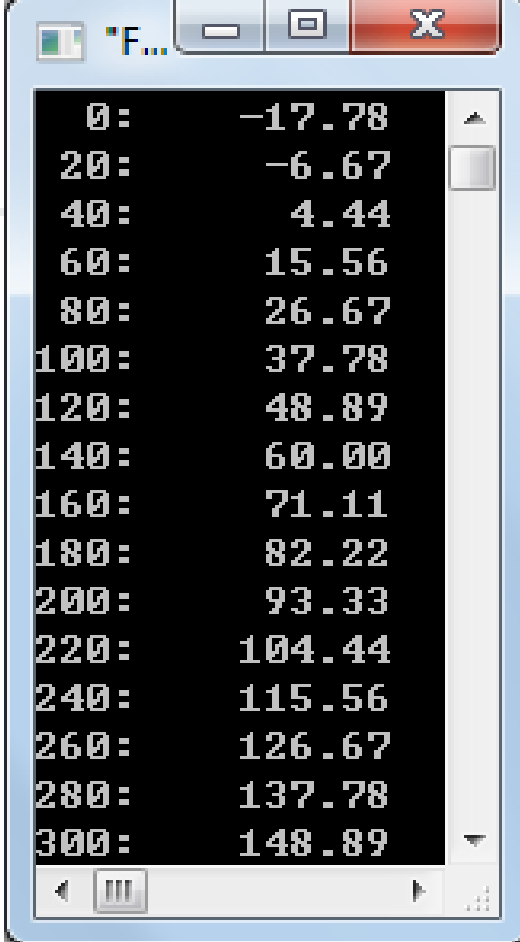


0:	-17.78
20:	-6.67
40:	4.44
60:	15.56
80:	26.67
100:	37.78
120:	48.89
140:	60.00
160:	71.11
180:	82.22
200:	93.33
220:	104.44
240:	115.56
260:	126.67
280:	137.78
300:	148.89

例2.6 打印华氏和摄氏温度对照表， 温度转换公式为： $^{\circ}\text{C}=(5/9)(^{\circ}\text{F}-32)$

```
#include <stdio.h>
/* print Fahrenheit-Celsius table */
int main(void)
{
    int fahr;

    fahr=0;
    while (fahr <= 300) {
        printf( "%3d:  %10.2f\n" , fahr, (5.0/9)*(fahr - 32) );
        fahr = fahr + 20;
    }
    return 0;
}
```



0:	-17.78
20:	-6.67
40:	4.44
60:	15.56
80:	26.67
100:	37.78
120:	48.89
140:	60.00
160:	71.11
180:	82.22
200:	93.33
220:	104.44
240:	115.56
260:	126.67
280:	137.78
300:	148.89


```
#include <stdio.h>
```

```
/* print Fahrenheit-Celsius table */
```

```
#define LOWER 0    /* 表的下限 */
```

```
#define UPPER 300  /* 表的上限 */
```

```
#define STEP 20    /* 步长 */
```

```
int main(void)
```

```
{
```

```
    int fahr;
```

```
    for(fahr = LOWER; fahr <= UPPER; fahr = fahr + STEP)
```

```
        printf( "%3d %10.2f\n" , fahr, (5.0/9)*(fahr - 32) );
```

```
    return 0;
```

```
}
```

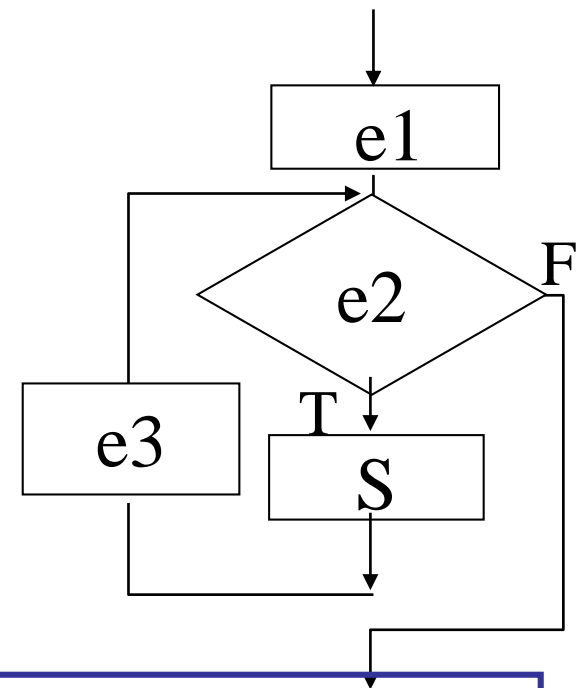
```
    fahr=LOWER;
```

```
    while ( fahr <= UPPER ) {
```

```
        printf( "%3d %10.2f\n" , fahr, (5.0/9)*(fahr - 32) );
```

```
        fahr = fahr +STEP;
```

```
    }
```



1. 用#define定义符号常量

#define是一种编译预处理指令,格式为:

#define 标识符 常量



符号常量(一般用大写,以区分变量)

2. 用const定义符号常量

const是关键字，称为类型限定符。格式为：

const 类型名 标识符=常量;

例如：

const double PI=3.14159;

const int DOWN=0x5000; /* 下光标键的扫描码 */

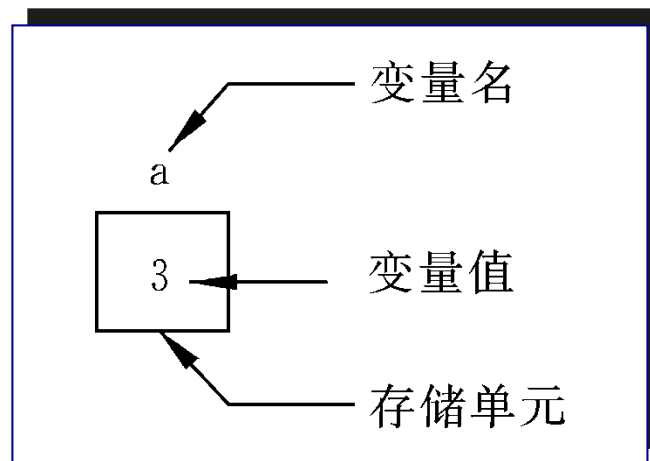
const int YES=1, NO=0;

用const和#define定义的符号常量的区别？

- **const**声明的标识符是一个只读变量，编译时系统会根据定义的类型为该标识符分配存储单元，并把对应的常量值放入其中，该值不能再被更改，此后，程序中每次出现该标识符都是对所代表存储单元的访问。
- **#define**定义的标识符没有对应的存储单元，只是在编译之前由预处理程序进行简单的文本替换。

2.5.3 变量定义

- 变量代表内存中具有特定属性的一个存储单元，它用来存放数据，这就是变量的值，在程序运行期间，这些值是可以改变的。



- 
- 要求对所有用到的变量作定义，也就是“先定义，后使用”。

类型名 变量表；

int total, average;

- 变量在声明时可以同时赋一个初值（称为变量的显示初始化），每个变量必须分别显示初始化。

int count=0, sum=0;

char alert='a', c ;

int count=0,sum=0; （不能 **int count=sum=0;**）



2.6 运算符和表达式

- 运算符是运算的符号表示，执行对运算对象（称为操作数）的各种操作。
- 单个的操作数（包括常量、变量和函数调用）是表达式，由运算符和操作数组成的有意义的计算式子更是表达式，如：
 - $\text{sqrt}(b*b-4*a*c)$
 - $x=x*PI/180$
 - $\text{fabs}(an) \geq \text{EPS}$

2.6.1 C运算符简介

学习每一种运算符都应掌握以下四点：

(1) 运算符的运算功能

(2) 操作数的个数和类型要求

单目（或一元）运算符

双目（或二元）运算符

三目（或三元）运算符

- 运算符都对操作数的类型有规定，比如%的操作数不能为浮点型。



(3) 运算符的优先级和结合性

(4) 运算所得结果的类型

- 运算的结果是一个具有确定类型的值，这个类型称为表达式值的类型。尤其当两个不同类型的操作数进行运算时，会引起数据类型的转换，特别要注意结果值的类型。

2.6.2 运算符的优先级和结合性

- 当表达式中包括多个运算符时，C语言会先按优先级规则解释表达式的意义。如：

$1+2*3$ 等价于 $1+(2*3)$

- 当一个操作数两侧的运算符优先级别相同时，则按“结合性”规则。

$1+2-3$ 等价于 $(1+2)-3$

---从左至右的结合性（左结合）



-a++ 等价于-(a++)

----从右至左的结合性（右结合）

- **所有运算符的优先级和结合性规则见表2.4**

2.6.3 算术运算

■ 运算符:

	双目	单目	
+	加法	正值	$3+6$, $+3$
-	减法	负值	$6-4$, -5
*	乘法		$3*8$
/	除法		$8/5$
%	求余		$7\%4$ 值为3

【例2.7】 求出所有的水仙花数

- “水仙花数”是一个三位数，其各位数字立方和等于该数本身。例如，153是一个“水仙花数”，因为 $153=1^3+5^3+3^3$ 。
- 找出“水仙花数”的关键是怎样从一个三位数中分离出百位数、十位数和个位数。

$i=x/100;$ $/*$ 分解百位数 $*/$

$j=(x-i*100)/10;$ $/*$ 分解十位数 $*/$

$k=x\%10;$ $/*$ 分解个位数 $*/$

```
#include<stdio.h>
```

```
int main(void)
```

```
{ int x,i,j,k;
```

```
for(x=100;x<=999;x++) { /* 枚举每一个3位数 */
```

```
    i=x/100;          /* 分解百位数 */
```

```
    j=(x-i*100)/10; /* 分解十位数 */
```

```
    k=x%10;          /* 分解个位数 */
```

```
    if(x==i*i*i+j*j*j+k*k*k) /* 是水仙花数 */
```

```
        printf("%5d",x);
```

```
}
```

```
return 0;
```

```
}
```

```
#include<stdio.h>
```

```
int main(void)
```

```
{ int x;
```

```
    for(x=100;x<=999;x++) {
```

```
        if( isNarcissus (x) ) /*x是水仙花数*/
```

```
            printf("%5d",x);
```

```
    }
```

```
    return 0;
```

```
}
```

如何定义函数**isNarcissus**，判断一个数**x**是否水仙花数。
是，返回**1**，不是返回**0**。

```
int  isNarcissus (int m)
{
    int i,j,k;
    i=m/100;          /* 分解百位数 */
    j=(m-i*100)/10;   /* 分解十位数 */
    k=m%10;           /* 分解个位数 */
    if(m==i*i*i+j*j*j+k*k*k) /* 是水仙花数 */
        return 1;
    else return 0;
}
```




注:

两个整型数据相除（结果为整, 舍去小数部分）

$$-5/3 \Rightarrow -1 \quad 1/2 \Rightarrow 0 \quad 1./2 \Rightarrow 0.5$$

使用时千万注意 `int / int` 出现数据丢失。

%操作数必需为整数

2.6.4 关系运算

- 有6个关系运算符：

- $>$ 大于

- $>=$ 大于等于

- $<$ 小于

- $<=$ 小于等于

- $=$ 等于

- \neq 不等于

常见的C语言编程错误

- 将运算符**`==`**写成运算符**`=`**（赋值）。可能会因为运行时的逻辑错误而导致不正确的结果。例如，

```
if (grade == 'A') printf("Very Good! ");  
/*当成绩的等级为A等时，输出Very Good!*/
```

而：

```
if (grade ='A') printf("Very Good!");  
/*不论成绩的等级是几等，总输出Very Good!*/
```



关系表达式的类型和值

- 关系表达式的类型: **int**
- 关系表达式的值:
 - 关系成立, 值为**1** (代表“真”)
 - 关系不成立, **0** (代表“假”)

举例

根据变量说明，给出表达式的值。

int x=4, y=3, z=2;

char c='a';

(1) **c == 'A'+32**

1

(2) **c+1 != 'b'**

0

(3) **x-y<=10**

1

(4) **z=x>y**

1

(5) **x>y>z**

0

注意

- 数学上判断 x 是否在区间 $[a,b]$ 中时，习惯写成：

$$a < x < b$$

- C中“ $a < x < b$ ”的含义与数学中的含义不同，应写成：

$$a < x \ \&\& \ x < b$$

2.6.5 逻辑运算

- 有3个逻辑运算符
- **!** 逻辑非
- **&&** 逻辑与
- **||** 逻辑或

逻辑表达式：用逻辑运算符将关系表达式
或逻辑量连接起来的式子



逻辑表达式的类型和值

- 逻辑运算的操作数可以是**0**和任何非**0**的数值，
 - 系统最终以**0**判断属于“**假**”（以**0**代表“假”），
以**非0**判断属于“**真**”（以**1**代表“真”）。
 - 逻辑表达式的值：**1** (“真”)
0 (“假”)
- ⑩ 逻辑表达式的类型： **int**

& 和 || 的真值表

e1	e2	&&	
0	0	0	0
0	非0	0	1
非0	0	0	1
非0	非0	1	1

! 的真值表

e	!
0	1
非0	0

熟练掌握C语言的关系运算符和逻辑运算符后，可以巧妙地表示一个复杂的条件。

- (1) 整数a是偶数

!(a%2) 或 a%2==0

- (2) 字符c的值是英文字母。

- **c>='a' && c<='z' || c>='A' && c<='Z'**

(3) 某一年year是闰年。如果某一年的年份能被4整除但不能被100整除，那么这一年就是闰年，此外，能被400整除的年份也是闰年。

- **!(year%4) && year%100 || !(year%400)**

```
/* 判断闰年,year 是闰年, 返回1; 否则, 返回0 */  
int isleap (int year)  
{  
    if (!(year%4) && year%100 || !(year%400))  
        return 1;  
    else  
        return 0;  
}
```

main函数怎么调用?

【例2.10】编程判断闰年

- 输入年份year，如果year是闰年，输出“yes”，否则，输出“no”。

```
#include<stdio.h>
```

```
int main(void)
```

```
{ int year;
```

```
scanf("%d",&year);
```

```
if ( isleap(year) )
```

```
    printf("%d is a leap year\n",year);
```

```
else
```

```
    printf("%d is not a leap year\n",year);
```

```
return 0;
```

```
}
```

函数调用

注意

- 编译程序在处理含有**&&**、**||** 表达式时，
- 往往采用优化算法(提高速度)。
- **e1&&e2** 一旦发现**e1=0**，不再计算 **e2**
- **e1 || e2** 一旦发现**e1=1**，不再计算 **e2**
- 例如，
- **x>=0.0 && sqrt(x)<=7.7**
- **/* 如果x值为负，不求x的平方根 */**

2.6.6 自增和自减运算

++: 自增, 使内存中存储的变量值加1

--: 自减, 使内存中存储的变量值减1

- **++**和**--**的奇特之处: 前缀式和后缀式都行

++X

X++ 相当于 **X=X+1;**

--X

X-- 相当于 **X=X-1;**

2.6.6 自增和自减运算

- 运算符多、操作灵活是C语言的一大特色。
- 在诸多运算符中，最难理解、最容易出错的是自增、自减运算符。
- ++、-- 运用得非常广泛，且具有一定的使用技巧和难度。
- 合理使用++、--，可以节省代码，提高效率；否则，容易造成错误。



注意前缀与后缀的区别

$x = 10;$

$y = ++x; \quad /* y=11, x=11 */$

---- 先执行对操作数的加运算，
再使用该操作数的值

$y = x++ \quad /* y=10, x=11 */$

---- 先使用该操作数的值，
再对它作加运算，

【例2.12】 计算 $1+2+3+\dots+n$, n 从键盘输入

```
#include<stdio.h>
```

```
int main(void)
```

```
{
```

```
    int i,sum,n;
```

```
    scanf("%d",&n);
```

```
    i=1; sum=0;
```

```
    while(i<=n) {
```

```
        sum=sum+i;
```

```
        i++;
```

```
    }
```

```
    printf("sum=%d\n",sum);
```

```
    return 0;
```

```
}
```

sum+=i++;

sum=sum+i++;



序列点

- 后缀++(或--)计算延迟的终止点称为**序列点**。在序列点之前，用原值，序列点之后，该操作数是更改后的新值。下列条件出现序列点：
- (1) **&&**、**||**、**?:** 或 **,** 运算符，即这些运算符的第一个操作数之后
- (2) **完整表达式结束时**，即表达式语句、**return**语句中的表达式、**if**、**switch**或循环语句中的条件表达式（包括**for**语句中的每个表达式）之后

【例2.13】后缀式++(或--)表达式举例

```
int a=1,b=0;
```

(1) $a--\&\&a$

(2) $b++ ? b : -b$

(3) $b++ + b++$

几点注意:

- 1. 只能用于变量（即左值表达式）。

如 $5++$, $(a+b)--$ 均不合法。

- 2. 结合性为从右至左。

如 $-i++$ 相当于 $-(i++)$

若 $i=3$, 则该表达式结果为 -3 , i 为 4



表达式的副作用

- **表达式的副作用**是指：表达式在求值过程中要改变该表达式中作为操作数的某个变量的值。
- **具有副作用的运算符**：赋值、复合赋值、增1、减1
- **c标准**未规定函数参数的求值顺序，也没有规定除了 **&&**、**||** 和 **,** 外的双目运算中两个操作数的计算顺序。由编译器自行规定计算顺序。
- 由于编译器对求值顺序的不同处理，当表达式中带有副作用的运算符时，就有可能产生二义性。

a.c

```
#include <stdio.h>
int main()
{
    int n=2;

    printf("%d,%d,%d\n",n,n++,n*=2);
    return 0;
}
```

"C:\Users\luping\Documents\Deb

```
4,4,4
Press any key to continue
```

VC6.0 结果

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    int n=2;
```

```
    printf("%d,%d,%d\n",n,n++,n*=2);
```

```
    return 0;
```

```
}
```

C:\Program Files\Dev-Cpp\ConsolePauser.exe

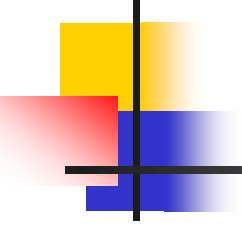
```
5,4,5
```

```
-----
Process exited with return value 0
```

```
Press any key to continue . . .
```

Dev C++ 结果

调试时打开汇编窗口看每句C对应的汇编、单步执行并观察相应堆栈、内存和寄存器变化，就明白了。

- 
- 要慎重使用自增、自减运算符
 - 在一个表达式中不要多处出现变量的自增、自减等运算。
 - 尽量不要在函数参数中用自增减表达式



【例2.11】 统计输入正文的字符数和行数

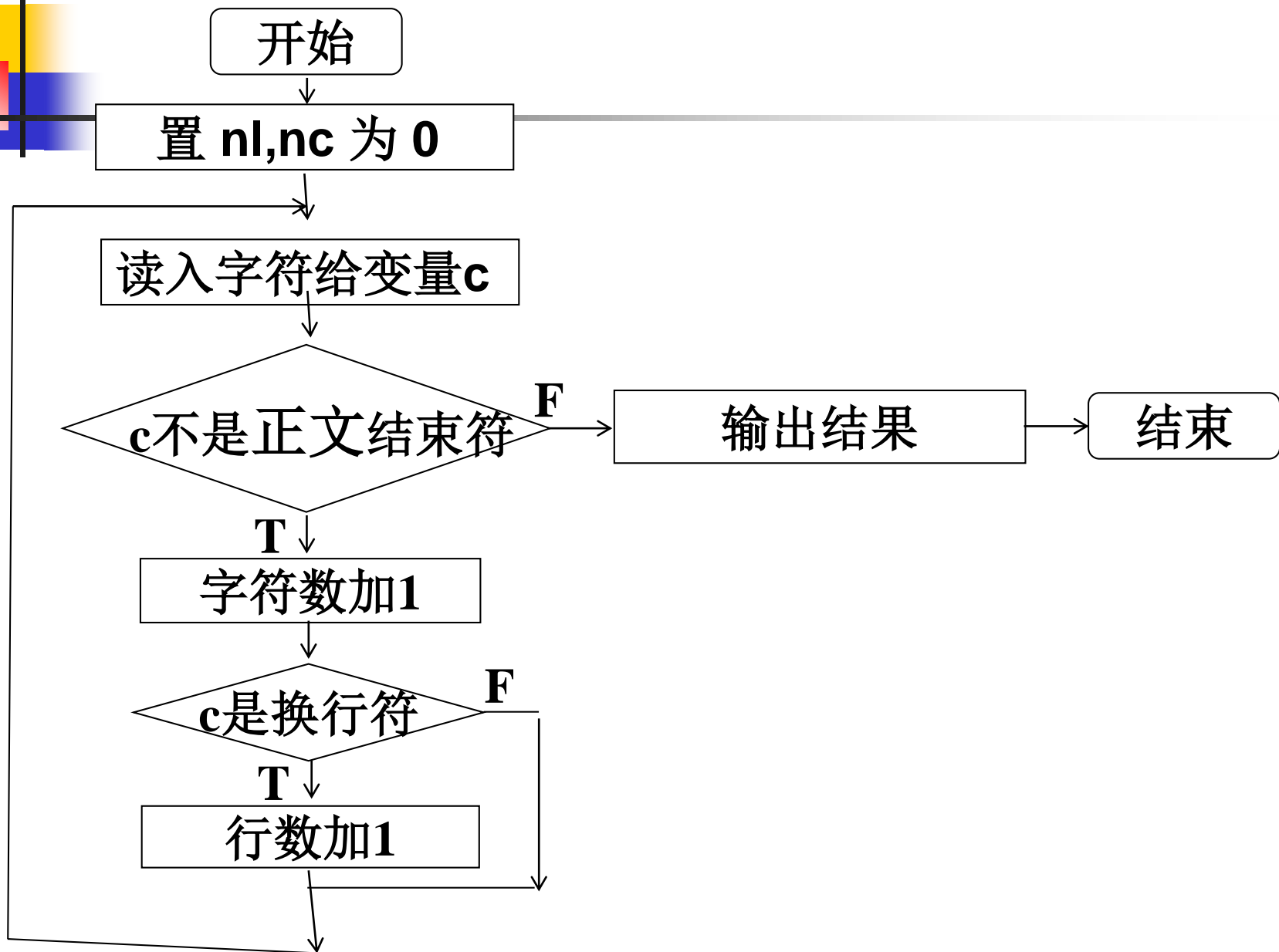
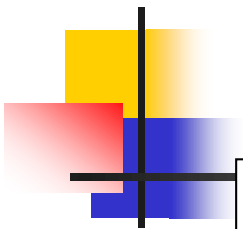
- 正文是一行行字符组成的字符序列，每一行是以换行符为结束标志的一串字符。

Upload file ✓

Special pages ✓

Page information ✓

Ctrl+z



2.6.7 赋值运算

1. 简单的赋值运算

赋值运算符：=

赋值表达式一般形式为：<变量>=<表达式>

右侧的“表达式”的值赋给左侧的变量

左值 (lvalue)：赋值运算符左侧的标识符

- **变量可以作为左值**
- **而表达式就不能作为左值(如a+b)**
- **常量也不能作为左值**

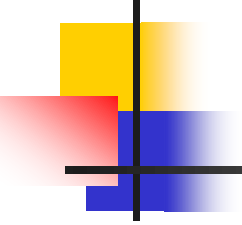
赋值表达式的值和类型

- 与左操作数的值和类型相同。

```
int x;
```

```
x=5.6; // 等价于 x= (int) 5.6;
```

- 强制类型转换运算符
(类型名) 操作数



■C把赋值处理为运算符，其好处是使赋值表达式可以像其它任何表达式一样当作一个数据来处理。 如

a=2;

b=3;

x=a+b; 可以被简化为

x= (a=2) + (b=3)

赋值运算符的结合性

- 右结合性，如
 a=b=c=3;
 等价于
 a= (b= (c=3)) ;

2. 复合的赋值运算

- 在“=”号之前加一个双目运算符：

$+=$, $-=$, $*=$, $/=$, $\%=$

如

$i+=2$ 等价于 $i=i+2$

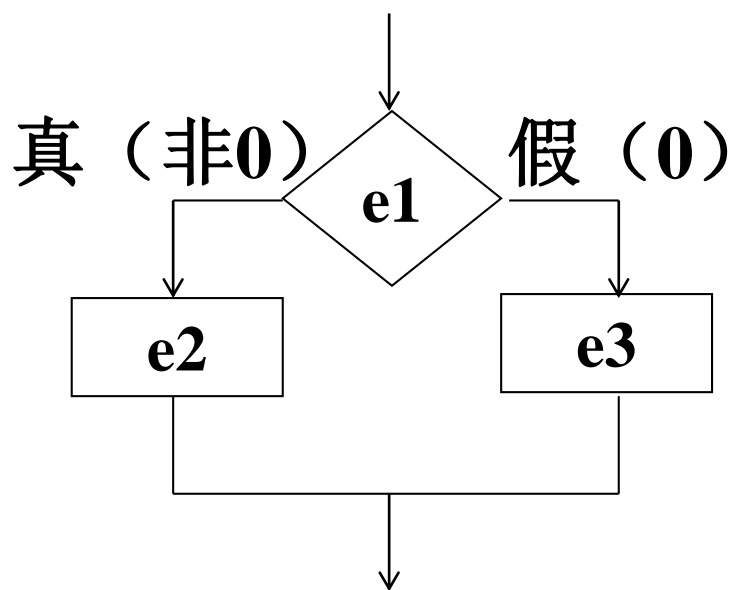
$y/=x+10$ 等价于 $y=y / (x+10)$

$x*=k=m+5$ 等价于 $x=x * (k=m+5)$

$s[i++] += 1$ 和 $s[i++] = s[i++] + 1$ 不等价

2.6.8 条件运算

■ $e1 ? e2 : e3$



if($e1$)

值= $e2$;

else

值= $e3$;



例：求三整数的最大值

- 输入3个整数，输出最大值。

[分析] 设置m保存最大值

步骤1：将a、b的最大值给m。

步骤2：将m、c的最大值给m。



程序一

```
#include<stdio.h>
int main(void)
{
    int a,b,c, m;
    printf("输入3个整数\n");
    scanf("%d%d%d",&a,&b,&c);
    m=(a>b)?a:b;  /* a>b?a:b  */
    m=(m>c)?m:c;
    printf("最大值为: %d\n",m);
    return 0;
}
```



```
#include<stdio.h>
```

```
int max(int,int);
```

```
int main(void)
```

```
{
```

```
    int a,b,c, max;
```

```
    printf("输入3个整数\n");
```

```
    scanf("%d%d%d",&a,&b,&c);
```

```
    m=max(a, b);
```

```
    m=max(m, c);
```

```
    printf("最大值为: %d\n",max);
```

```
    return 0;
```

```
}
```

```
int max(int x,int y)
{
    return(x>y?x:y);
}
```



统计正数、负数、零的个数

```
k=1; positive=negative=zero=0;
```

```
while(k<11) {
```

```
    scanf("%d",&x);
```

```
        x>0? positive++: (x<0? negative++: zero++);
```

```
    k++;
```

```
}
```

按照结合性，括号可省略

举例

- 1) **$x=10;$**

- **$y=x>9?100:200$ */* y: 100 */***

$(2) \ x = a > 0 ? 1 : a < 0 ? -1 : 0$

等价于(因为是右结合性)

$x = a > 0 ? 1 : (a < 0 ? -1 : 0)$



2.6.9 逗号运算

- $e1, e2$
- 计算规则:
- 先计算 $e1$ ，再计算 $e2$ ，逗号表达式的值和类型与 $e2$ 的值和类型相同



【例】计算 $1+2+3+\dots+n$ ， n 从键盘输入

```
#include<stdio.h>  
#define N 10  
int main(void)  
{  
    int i,sum;  
    for(sum=0,i=1;i<=N;i++)  
        sum+=i;  
    printf('sum=%d\n',sum);  
    return 0;  
}
```

举例

- (1) $x=(i=4, i\%3)$
- 表达式的值为: 1, x 为1

- (2) $x=i=4, i\%3$
- 表达式的值为: 1, x 为4



扩展形式

逗号表达式的一般形式可以扩展为

$e_1, e_2, e_3, \dots, e_n$

它的值为表达式 e_n 的值。

注意

并不是任何地方出现的逗号都是作为逗号运算符。例如

“2, 3”并不是一个逗号表达式，它是printf函数的2个参数

```
printf(“%d,%d”, 2, 3); /* 输出
```

```
printf(“%d,%d”, (2, 3), 3 ); /* 输出: 3, 3  
*/
```

“ (2, 3) ”
是一个逗号表达式，它的值等于3。



反转字符串

```
i=0;    // 首字符位置
```

```
j=strlen(s)-1; // 最后一个字符位置 */
```

```
while(i<j) {
```

```
    mid=s[i];
```

```
    s[i]=s[j];
```

```
    s[j]=mid; /*交换s[i]、s[j]
```

```
    i++;
```

```
    j--; /* i、j向中间靠拢 */
```

```
}
```

```
for(i=0,j=strlen(s)-1;i<j;i++,j--)
```

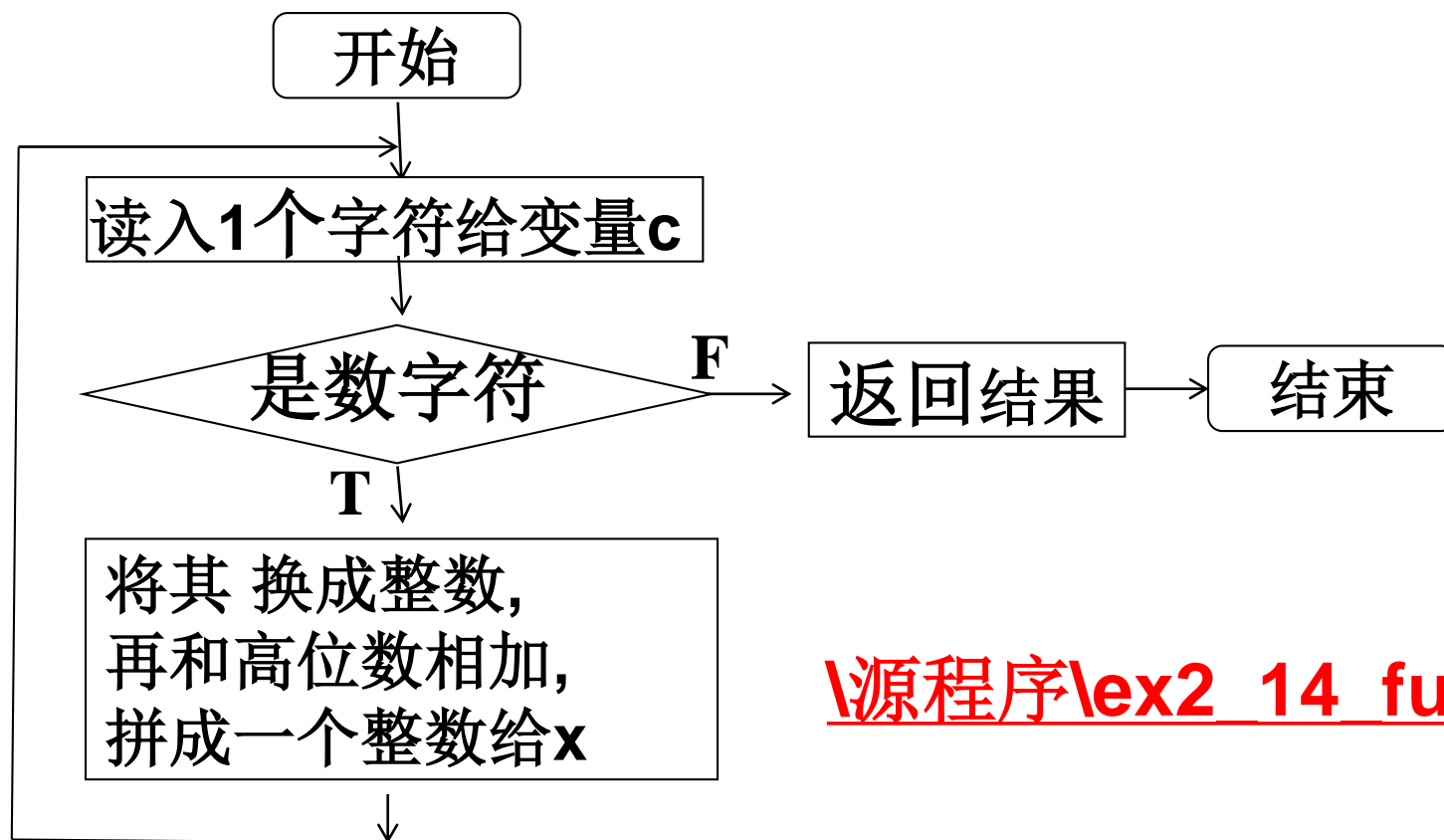
```
    mid=s[i],s[i]=s[j],s[j]=mid;
```

【例2.14】 输入一串数字字符，将其转换为一个十进制整数赋值给x（模拟scanf("%d",&x)的功能）。

- 当用scanf("%d",&x)输入一个整数时，实际上输入的是组成该整数的各位数字的一个字符串，scanf函数读到的每个数字都是该字符的字符码。为了将字符串转换为整数，必须先将每个数字字符转换成数字对应的一位整数，然后按相应的数位拼成一个十进制整数，最后赋给变量x。



```
int getint (void) ;
```



[\源程序\ex2_14_fun.c](#)

```
int getint(void)
{
    char c;    /* 用于存放当前输入的字符 */
    int x;     /* 用于存放转换的整数 */
    for(x=0, c=getchar(); c>='0' && c<='9'; c=getchar()) /* 遇非数字字符结束 */
        x=10*x+c-'0';    /* c-'0': 将数字字符转换成对应的一位整数 */
    return x;
}

int main(void)
{
    int score;
    score=getint();
    printf("%d\n", score);
    return 0;
}
```



2.6.10 sizeof运算

一个单目运算符，有两种形式：

(1) sizeof (类型名)

给出指定数据类型占用的存储字节数

(2) sizeof 表达式

给出表达式结果的类型占用的存储字节数

举例

假设int类型占用2字节。

- `sizeof(long)` `/* 值为4 */`
- `double x; sizeof x` `/* 值为8 */`
- `short a[10]; sizeof (a)` `/* 值为20 */`
- `int a=1,b=1; sizeof (a+b)`

`/* 值为2, 而sizeof a+b, 先求sizeof a, 再和b加, 值为3 */`



sizeof是一个常量表达式

- 其运算是在编译时执行的。因此，当**sizeof**的操作数是表达式时，则在编译时分析表达式以确定类型，运行时不对这个表达式求值。
- 例如：
- **short x=1;**
- **sizeof(++x);** */* x不递增，仍为1 */*



2.7 位运算符和位表达式

6个位运算符:

\sim (求反)

$\&$ (按位与)

$|$ (按位或)

\wedge (按位异或, 或按位加)

\ll (左移)

\gg (右移)

2.7.1 按位求反 (\sim)

- 对操作数的每个二进制位取相反值
- 例如:
- `short a=5; unsigned short b=5;`
- `a`和`b`的二进制为: **00000000 00000101**
- $\sim a$ 的二进制为: **11111111 11111010**
- $\sim a$ 的值为`short`型 **-6**。
- $\sim b$ 的值为`unsigned short`型 **65530**。

2.7.2 按位与(&)、或(|)、加(^)运算

表达式	二进制表示	十六进制值	说明
x	01101000 11010001	0x68d1	待处理数据
mask	11111111 00000000	0xff00	逻辑尺
x & mask	01101000 00000000	0x6800	将x的低字节置为0， 保留高字节
x mask	11111111 11010001	0xffd1	将x的低字节保留， 高字节置为1
x ^ mask	10010111 11010001	0x97d1	将x的低字节保留， 高字节翻转



2.7.3 左移(<<)和右移(>>)

- **$e \ll n$** 将 **e** 的值向左移 **n** 位，低 **n** 位填入**0**。
左移**1**位相当于该数值乘以**2**
- **$e \gg n$** 将 **e** 的值向右移 **n** 位，而高 **n** 位可能填入**?**。
 - **e** 是无符号类型，填入**0**；
 - **e** 是有符号类型，一些机器填入**0**（即“逻辑移位”），
而另一些机器则填入符号位（即“算术移位”）。
- 在使用右移运算符时应经常用无符号类型。
- 右移一位相当于该数值除以**2**

移位运算符的例子

表达式	二进制表示	值	行为
a	00000000 00001011	11	对a进行负号运算 a左移2位 b右移3位 -a右移2位(填入符号位1, 而其他机器可能是0)
b	00000000 00001111	15	
-a	11111111 11110101	-11	
a<<2	00000000 00101100	44	
b>>3	00000000 00000001	1	
-a>>2	11111111 11111101	-3	

【例2.15】写一个表达式

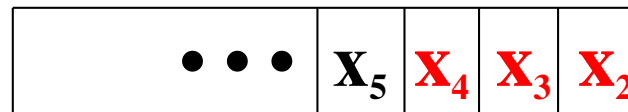
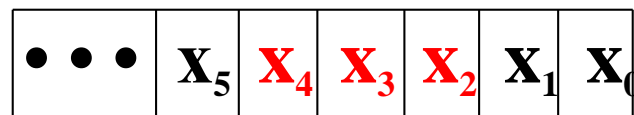
将整数x的第m位起的右n位取出来，作为另一个数

注：一个整数的各个二进制位从右至左依次编号为第0位、第1位、第2位、.....

如： $m=4$ $n=3$

① 将要取出的那n位移到最右端

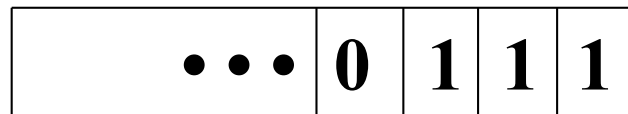
$$x \gg m-n+1$$



② 设计一个逻辑尺：低n位全为1，其余位全为0

$$\sim(\sim 0 \ll n)$$

$$(1 \ll n) - 1$$



③ 将上面二者进行&的运算

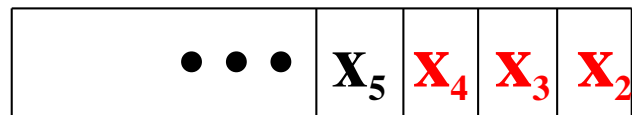
$$x \gg (m-n+1) \& \sim(\sim 0 \ll n)$$

【例2.15】 写一个表达式，将整数x的第m位起的右n位取出来，作为另一个数。（一个整数的各个二进制位从右至左依次编号为第0位、第1位、第2位、……）

如： $m=4$ $n=3$

① 将要取出的那n位移到最右端

$x \gg m-n+1$



② 设计一个逻辑尺：低n位全为1，其余位全为0

$\sim(\sim 0 \ll n)$

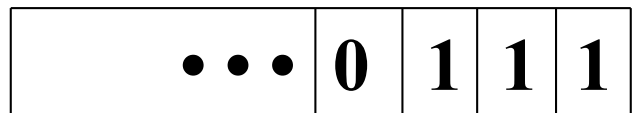
$(1 \ll n) - 1$

$\sim 0 \gg 16-n$



$\sim 0u \gg 16-n$

$\sim 0u \gg \text{sizeof(int)} * 8 - n$



③ 将上面二者进行&的运算

$x \gg (m-n+1) \& \sim(\sim 0 \ll n)$

2.7.4 位运算符应用举例

【例2.16】 压缩和解压示例

可以把表示21世纪日期的日、月和年3个整数压缩成一个16位的整数(short)。写一个表达式，实现压缩存储日、月和年。

15	11 10	7 6	0
日	月	年	

$\text{day} \ll 11 \mid \text{month} \ll 7 \mid \text{year}$

分析:

- 可以把表示21世纪日期的日、月和年3个整数压缩成1个16位的整数，因为日有31个值，月有12个值，年有100个值，所以可以在一个整数中用5b表示日，用4b表示月，用7b表示年.

15	11 10	7 6	0
日	月	年	

$\text{day} \ll 11 \mid \text{month} \ll 7 \mid \text{year}$



编写一个压缩存储日、月和年的函数。

函数名: **pack**

输入参数: **day**-日

month-月

year-年

返回值: 压缩后的整数

```
short pack(short day, short month, short year)
{
    return(day<<11 | month<<7 | year) ;
}
```

压缩存储日、月和年

```
short pack(short day, short month, short year); //函数声明语句
```

```
int main()
```

```
{ short day, month, year, date;
```

输入/出 short 数据 用 %hd

输入/出 unsigned short 用 %hu

```
printf("Input year, month, day (year, month, day): ");
```

```
scanf("%hd%hd%hd", &year, &month, &day);
```

```
date = pack(day, month, year); //函数调用
```

```
printf("%hx\n", date);
```

```
return 0;
```

```
}
```

```
short pack(short day, short month, short year) //函数定义
```

```
{ return (day << 11 | month << 7 | year);
```

```
}
```



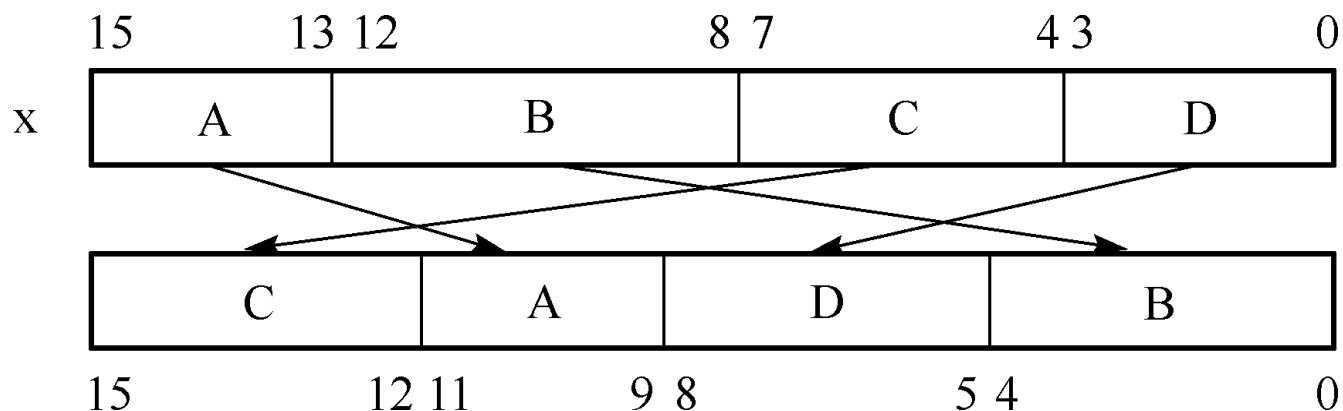
练习：编写解压函数

```
void unpack(short x)
{
    将x解压为 day, month, year
    printf :  day, month, year
}
```

【例2.17】 简单加密示例

将一个短整型数 x 分成4个长度不等的部分：

A (3b)、**B (5b)**、**C (4b)** 和 **D (4b)**，然后将它们按照**CADB**的顺序重新拼凑在一起，实现对其加密的功能。写一个表达式，要求其值为 x 的密文。



(1) 设计4个逻辑尺，分别用于取出A、B、C和D 4个部分。A的逻辑尺应设计成：高3位全为1，其余全为0，即0xE000；B的逻辑尺应设计成：第8~12位全为1，其余全为0，即0x1F00；同理C和D的逻辑尺分别为0x00F0和0x000F。

(2) 取出A、B、C和D，再移到新位置。将x分别和逻辑尺进行&运算，可取出每一部分。用每部分第1位的初始位置减去其新位置得到移位数，正数右移，负数左移。

(3) 将各部分重新组装在一起。

得到表达式为：

$(x \& 0xE000) \gg 4 \mid (x \& 0x1F00) \gg 8 \mid (x \& 0xF0) \ll 8 \mid (x \& 0xF) \ll 5$



编写该加密函数

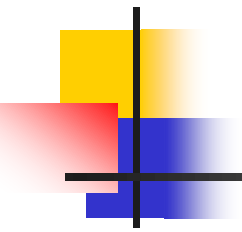
函数名: **encrypt**

函数参数:

x-待加密短整数

函数返回值:

加密后的短整数



```
#define AE  0xE000    /* 取出A的逻辑尺 */
#define BE  0x1F00    /* 取出B的逻辑尺 */
#define CE  0x00F0    /* 取出C的逻辑尺 */
#define DE  0x000F    /* 取出D的逻辑尺 */
unsigned short encrypt(unsigned short x)
{
    unsigned short a,b,c,d;
    a=(x&AE)>>4
    b=(x&BE)>>8
    c=(x&CE)<<8
    d=(x&DE)<<5
    return (a|b|c|d);
}
```

【例2.18】 计算ASCII码字符的奇偶校验位

二进制数据在传送中会发生**误码**（1变成0或0变成1），这就有如何发现及纠正误码的问题。解决方法是在原始数据（数码位）基础上增加几位校验（冗余）位。

奇偶校验：分奇校验和偶校验。

奇校验：整个码字中奇数个“1”

偶校验：整个码字中偶数个“1”

data='A'

	1	0	0	0	0	0	1
--	---	---	---	---	---	---	---

校验位 parity

校验位在每个字符的最高位

```
int main()
```

```
{ unsigned char data,backup,t;
```

```
int parity=0; /* 奇偶校验位,0-偶校验, 1-奇校验 */
```

```
data=getchar(); /* 被校验的数据 */
```

```
backup=data;
```

```
while (data) {
```

```
    t=data&1; /* 取该数的最低位 */
```

```
    parity^=t; /* 进行异或操作 */
```

```
    data>>=1; /* 为取下一位做准备 */
```

```
}
```

```
data=backup|(parity<<7); /* 生产校验码 */
```

```
printf("The data is %#x\n",backup);
```

```
printf("Parity-Check Code is %#x\n",data); /* 输出校验码 */
```

```
return 0;
```

```
}
```



2.8 类型转换

- C语言允许双精度、单精度、整型及字符数据之间混合运算

$10 + '1' + 6.5$

但有一个规则：先转换成同一类型,再计算。

2.8.1 整数提升

- 任何表达式中的char、unsigned char、short和unsigned short都要先转换成int或unsigned，如果原始类型的所有值可以用int表示，则转换成int，否则转换成unsigned，把这称为“整数提升”。
- short a;
sizeof(a)=? sizeof(-a)=?
2 4

2.8.2 一般算术转换

char / short → int → unsigned → long → unsigned long
→ float → double → long double



2.8.3 赋值转换

- 右操作数的值被转换为左操作数的类型
- 例如:

```
short s=5;  double d=2.9;  long l=0x11118001;
```

```
则 s = d;
```

```
/*把d转换为short, 再赋给s。值为2*/
```

```
d = s;
```

```
/*把s转换为double, 再赋给d。值为5.0*/
```

```
s = l;    /* s值为 ? */
```

```
printf("%hx %hd %hu\n",s,s,s);
```

```
// 输出  8001  -32767  32769
```

2.8.4 强制类型转换

■ (类型名) 操作数

■ 例如，

(double) i

/* 将i转换成double，i的类型保持不变*/

(long)('a'-32)

/* 'a' 被自动转换成int，相减的结果被强制转换为long*/

(float)x+y

/*等价于((float)x)+y*/



2.9 枚举类型

- 使代码更具可读性，理解清晰，易于维护。
- 它是由用户定义的若干枚举常量的集合
- 枚举常量用标识符命名,缺省状态下，其值是所有列举元素的序号，序号从**0**开始

```
enum color { WHITE,YELLOW,RED,BLUE};
```

```
enum week { SUN, MON, TUE, WED, THU, FRI, SAT };
```

2.9.1 枚举类型的定义

```
enum week { SUN, MON, TUE, WED, THU, FRI, SAT } ;
```

↑
关键字

↑
枚举名

枚举常量

在未指定值的缺省情况下，第一个枚举常量的值为0，以后的值依次递增1。

可以指定一个或多个枚举常量的值，未指定值的枚举常量的值比前面的值大1。

```
enum sizes { SMALL, MEDIUM=10, BIG , TOO_BIG=20 } ;
```

enum 后面也可以不出现枚举名。

```
enum { WIN, LOSE, TIE, ERROR} ;
```

2.9.2 用枚举类型定义符号常量

```
#define WIN 0
#define LOSE 1
#define TIE 2
#define ERROR -1
```

- 可用下面的枚举类型定义来代替。

```
enum { WIN, LOSE, TIE, ERROR=-1};
```

2.9.3. 枚举变量的声明

(1) 在定义枚举类型的同时说明枚举变量

```
enum color { RED, GREEN, BLUE} c1, c2;
```

(2) 利用枚举名来说明枚举变量

```
enum color { RED, GREEN, BLUE} c1;
```

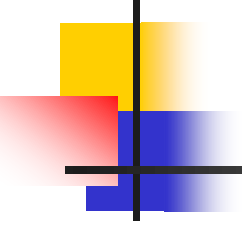
```
enum color c2;
```

或者

```
enum color { RED, GREEN, BLUE};
```

```
enum color c1, c2;
```

- 一个枚举变量的值是**int**型整数，但值域仅限于列举出来的范围。枚举变量值的输入和输出都只能是整数。
- **c1=BLUE;**
- */* 等价于 **c1=2;** 使用有意义的标识符有助于读者理解程序 */*
- **printf(“%d”,c1);**
- */* 输出**2**，而不是**BLUE** */*
- **scanf(“%d”,c2);**
- */* 输入**0**，不能输入**RED** */*



下面的语句是错误的:

```
c1=3;
```

```
/* 变量c1的值域为: 0、1和2 */
```

```
printf("%s", GREEN );
```

```
/* 输出错误的结果, 而不是GREEN */
```

```
if(c1==RED) printf("Red");
```

【例2.22】用枚举变量`day`控制循环，输出存储在数组`weekName`中的英文星期名。

- 一个枚举变量的值是一个`int`整数，它可以出现在整数允许出现的任何地方。如果要输出与枚举值相对应的枚举常量标识符或代表其含义的完整字符串，可以定义一个字符串数组，以枚举值作为下标。
- 源程序\ex2_22.c

/*简要描述：用枚举变量控制循环，输出存储在数组weekName中的英文星期名。*/

```
#include<stdio.h>
```

```
enum week { SUN, MON, TUE, WED, THU, FRI, SAT } ;
```

```
int main(void)
```

```
{
```

```
enum week day;
```

```
char *weekName[ ]={ "Sunday", "Monday", "Tuesday",  
    "Wednesday", "Thursday", "Friday", "Saturday"  
};
```

```
for(day=SUN;day<=SAT;day++)
```

```
    printf("\n%3d\t%s",day,weekName[day]);
```

```
return 0;
```

```
}
```