第2章　数据类型、运算符和表达式

2.1　C语言的数据类型

C语言有五种基本数据类型：字符、整型、单精度实型、双精度实型和空类型。尽管这几种类型数据的长度和范围随处理器的类型和 C语言编译程序的实现而异，但以 bit为例，整数与CPU字长相等，一个字符通常为一个字节，浮点值的确切格式则根据实现而定。对于多数微机，表2-1给出了五种数据的长度和范围。

表2-1　基本类型的字长和范围

类　型	长　度（bit）	范　围
char(字符型)	8	0~255
int（整型）	16	-32768~32767
float（单精度型）	32	约精确到6位数
double（双精度型）	64	约精确到12位数
void（空值型）	0	无值

表中的长度和范围的取值是假定 CPU的字长为16bit。

C语言还提供了几种聚合类型（aggregate types），包括数组、指针、结构、共用体（联合）、位域和枚举。这些复杂类型在以后的章节中讨论。

除void类型外，基本类型的前面可以有各种修饰符。修饰符用来改变基本类型的意义，以便更准确地适应各种情况的需求。修饰符如下：

- signed（有符号）。
- unsigned（无符号）。
- long（长型符）。
- short（短型符）。

修饰符signed、short、long和unsigned适用于字符和整数两种基本类型，而 long还可用于double（注意，由于long float与double意思相同，所以ANSI标准删除了多余的long float）。

表2-2给出所有根据ANSI标准而组合的类型、字宽和范围。切记，在计算机字长大于 16位的系统中，short int与signed char可能不等。

表2-2　ANSI标准中的数据类型

类　型	长　度（bit）	范　围
char(字符型)	8	ASCII字符
unsigned char（无符号字符型）	8	0~255
signed char(有符号字符型)	8	-128~127
int（整型）	16	32768~32767
unsigned int(无符号整型)	16	0~65535
signed int(有符号整型)	16	同int

（续）

类 型	长 度（bit）	范 围
short int(短整型)	8	128~127
unsigned short int(无符号短整型)	8	0~255
signed short int(有符号短整型)	8	同shortint
long int(长整型)	32	2147483648~2147483649
signed long int(有符号长整型)	32	2147483648~2147483649
unsigned long int(无符号长整型)	32	0~4294967296
float（单精度型）	32	约精确到6位数
double（双精度型）	64	约精确到12位数

*表中的长度和范围的取值是假定CPU的字长为16bit。

因为整数的缺省定义是有符号数，所以singed这一用法是多余的，但仍允许使用。

某些实现允许将unsigned用于浮点型，如unsigned double。但这一用法降低了程序的可移植性，故建议一般不要采用。

为了使用方便，C编译程序允许使用整型的简写形式：

- short int 简写为short。
- long int 简写为long。
- unsigned short int 简写为unsigned short。
- unsigned int 简写为unsigned。
- unsigned long int 简写为unsigned long。

即，int可缺省。

2.2 常量与变量

2.2.1 标识符命名

在C语言中，标识符是对变量、函数标号和其它各种用户定义对象的命名。标识符的长度可以是一个或多个字符。绝大多数情况下，标识符的第一个字符必须是字母或下划线，随后的字符必须是字母、数字或下划线（某些C语言编译器可能不允许下划线作为标识符的起始字符）。下面是一些正确或错误标识符命名的实例。

正确形式	错误形式
count	2count
test23	hi! there
high_balance	high..balance

ANSI标准规定，标识符可以为任意长度，但外部名必须至少能由前 8个字符唯一地区分。这里外部名指的是在链接过程中所涉及的标识符，其中包括文件间共享的函数名和全局变量名。这是因为对某些仅能识别前8个字符的编译程序而言，下面的外部名将被当作同一个标识符处理。

```
counters        counters1        counters2
```

ANSI标准还规定内部名必须至少能由前31个字符唯一地区分。内部名指的是仅出现于定

义该标识符的文件中的那些标识符。

C语言中的字母是有大小写区别的，因此count Count COUNT是三个不同的标识符。

标识符不能和C语言的关键字相同，也不能和用户已编制的函数或C语言库函数同名。

2.2.2 常量

C语言中的常量是不接受程序修改的固定值，常量可为任意数据类型，如下例所示：

数据类型	常量举例
char	'a'、'\n'、'9'
int	21、 123 、2100 、-234
long int	35000、 -34
short int	10、-12、90
unsigned int	10000、 987、 40000
float	123.23、 4.34e-3
double	123.23、 12312333、 -0.9876234

C语言还支持另一种预定义数据类型的常量，这就是串。所有串常量括在双撇号之间，例如"This is a test"。切记，不要把字符和串相混淆，单个字符常量是由单撇号括起来的，如'a'。

2.2.3 变量

其值可以改变的量称为变量。一个变量应该有一个名字(标识符)，在内存中占据一定的存储单元，在该存储单元中存放变量的值。请注意区分变量名和变量值这两个不同的概念。

所有的C变量必须在使用之前定义。定义变量的一般形式是：

```
type variable_list;
```

这里的type必须是有效的C数据类型，variable_list（变量表）可以由一个或多个由逗号分隔的多个标识符名构成。下面给出一些定义的范例。

```
int i, j, l;
short int  si;
unsigned int ui;
double balance, profit,loss;
```

注意 C语言中变量名与其类型无关。

2.3 整型数据

2.3.1 整型常量

整型常量及整常数。它可以是十进制、八进制、十六进制数字表示的整数值。

十进制常数的形式是：

```
digits
```

这里digits可以是从0到9的一个或多个十进制数位，第一位不能是0。

八进制常数的形式是：

```
0digits
```

在此，digits可以是一个或多个八进制数（0～7之间），起始0是必须的引导符。

十六进制常数是下述形式：

```
0xhdigits
0Xhdigits
```

这里hdigits可以是一个或多个十六进制数（从 0～9的数字，并从“ a ”～“ f ”的字母）。引导符0是必须有的，X即字母可用大写或小写。

注意，空白字符不可出现在整数数字之间。表 2-3列出了整常数的形式。

表2-3 整常数的例子

十 进 制	八 进 制	十六进制
10	012	0Xa或0XA
132	0204	0X84
32179	076663	0X7db3或0X7DB3

整常数在不加特别说明时总是正值。如果需要的是负值，则负号“ - ”必须放置于常数表达式的前面。

每个常数依其值要给出一种类型。当整常数应用于一表达式时，或出现有负号时，常数类型自动执行相应的转换，十进制常数可等价于带符号的整型或长整型，这取决于所需的常数的尺寸。

八进制和十六进制常数可对应整型、无符号整型、长整型或无符号长整型，具体类型也取决于常数的大小。如果常数可用整型表示，则使用整型。如果常数值大于一个整型所能表示的最大值，但又小于整型位数所能表示的最大数，则使用无符号整型。同理，如果一个常数比无符号整型所表示的值还大，则它为长整型。如果需要，当然也可用无符号长整型。

在一个常数后面加一个字母l或L，则认为是长整型。如10L、79L、012L、0115L、0XAL、0x4fL等。

2.3.2 整型变量

前面已提到，C规定在程序中所有用到的变量都必须在程序中指定其类型，即“ 定义 ”。这是和BASIC、FORTRAN不同的，而与Pascal相似。

[例2-1]

```
main()
  {
       int a,b,c,d;                          /* 指定a,b,c,d 为整型变量 */
       unsigned u;                           /* 指定u为无符号整型变量 */
       a=12; b=-24; u=10;
       c=a+u; d=b+u;
       printf("a+u=%d, b+u=%d\n",c,d);
  }
```

运行结果为：

```
RUN
a+u=22, b+u=-14
```

可以看到不同类型的整型数据可以进行算术运算。在本例中是 int型数据与unsingned int型数据进行相加减运算。

2.4 实型数据

2.4.1 实型常量

实型常量又称浮点常量，是一个十进制表示的符号实数。符号实数的值包括整数部分、尾数部分和指数部分。实型常量的形式如下：

```
[digits][.digits][E|e[+|-]digits]
```

在此digits是一位或多位十进制数字（从 0～9）。 E（也可用e）是指数符号。小数点之前是整数部分，小数点之后是尾数部分，它们是可省略的。小数点在没有尾数时可省略。

指数部分用 E或e开头，幂指数可以为负，当没有符号时视为正指数的基数为10，如1.575E10表示为：1.575×10^{10}。在实型常量中不得出现任何空白符号。

在不加说明的情况下，实型常量为正值。如果表示负值，需要在常量前使用负号。

下面是一些实型常量的示例：

```
15.75, 1.575E10, 1575e-2, -0.0025, -2.5e-3, 25E-4
```

所有的实型常量均视为双精度类型。

实型常量的整数部分为0时可以省略，如下形式是允许的：

```
.57, .0075e2, -.125, -.175E-2
```

注意 字母E或e之前必须有数字，且E或e后面指数必须为整数，如e3、2.1e3.5、.e3、e等都是不合法的指数形式。

2.4.2 实型变量

实型变量分为单精度（float型）和双精度（double型）。对每一个实型变量都应再使用前加以定义。如：

```
float x,y；          /*指定x,y为单精度实数*/
double z；           /*指定z为双精度实数*/
```

在一般系统中，一个 float型数据在内存中占 4个字节（32位）一个double型数据占8个字节（64位）。单精度实数提供7位有效数字，双精度提供15~16位有效数字，数值的范围随机器系统而异。

值得注意的是，实型常量是double型，当把一个实型常量赋给一个 float型变量时，系统会截取相应的有效位数。例如

```
float a;
a=111111.111;
```

由于float型变量只能接收7位有效数字，因此最后两位小数不起作用。如果将 a改为double型，则能全部接收上述9位数字并存储在变量a中。

2.5 字符型数据

2.5.1 字符常量

字符常量是指用一对单引号括起来的一个字符。如‘ a ’, ‘ 9 ’, ‘ ! ’。字符常量中的单引号只起定界作用并不表示字符本身。单引号中的字符不能是单引号（ ’）和反斜杠（\），它们特有的表示法在转义字符中介绍。

在C语言中，字符是按其所对应的ASCII码值来存储的，一个字符占一个字节。例如：

字符	ASCII码值（十进制）
!	33
0	48
1	49
9	57
A	65
B	66
a	97
b	98

注意 字符'9'和数字9的区别，前者是字符常量，后者是整型常量，它们的含义和在计算机中的存储方式都截然不同。

由于C语言中字符常量是按整数（short型）存储的，所以字符常量可以像整数一样在程序中参与相关的运算。例如：

```
'a'－32;        /* 执行结果97-32 = 65        */
'A' + 32；      /* 执行结果65+32 = 97        */
'9'－9；        /* 执行结果57-9  = 48        */
```

2.5.2 字符串常量

字符串常量是指用一对双引号括起来的一串字符。双引号只起定界作用，双引号括起的字符串中不能是双引号（"）和反斜杠（\），它们特有的表示法在转义字符中介绍。例如：

```
"China" ，"C program", "YES&NO", "33312-2341", 等A"
```

C语言中，字符串常量在内存中存储时，系统自动在字符串的末尾加一个“ 串结束标志 ”，即ASCII码值为0的字符NULL，常用\0表示。因此在程序中，长度为 n个字符的字符串常量，在内存中占有n+1个字节的存储空间。

例如，字符串China有5个字符，作为字符串常量"China"存储于内存中时，共占6个字节，系统自动在后面加上NULL字符，其存储形式为：

C	h	i	n	a	NULL

要特别注意字符串与字符串常量的区别，除了表示形式不同外，其存储性质也不相同，字符'A'只占1个字节，而字符串常量"A"占2个字节。

2.5.3 转义字符

转义字符是C语言中表示字符的一种特殊形式。通常使用转义字符表示 ASCII码字符集中不可打印的控制字符和特定功能的字符，如用于表示字符常量的单撇号（'），用于表示字符串常量的双撇号（"）和反斜杠（\）等。转义字符用反斜杠 \后面跟一个字符或一个八进制或十六进制数表示。表2-4给出了C语言中常用的转义字符。

表2-4 转义字符

转义字符	意 义	ASCII码值（十进制）
\a	响铃(BEL)	007
\b	退格(BS)	008
\f	换页(FF)	012
\n	换行(LF)	010
\r	回车(CR)	013
\t	水平制表(HT)	009
\v	垂直制表(VT)	011
\\	反斜杠	092
\?	问号字符	063
\'	单引号字符	039
\"	双引号字符	034
\0	空字符(NULL)	000
\ddd	任意字符	三位八进制
\xhh	任意字符	二位十六进制

字符常量中使用单引号和反斜杠以及字符常量中使用双引号和反斜杠时，都必须使用转义字符表示，即在这些字符前加上反斜杠。

在C程序中使用转义字符 \ddd或者\xhh可以方便灵活地表示任意字符。 \ddd为斜杠后面跟三位八进制数，该三位八进制数的值即为对应的八进制 ASCII码值。\x后面跟两位十六进制数，该两位十六进制数为对应字符的十六进制 ASCII码值。

使用转义字符时需要注意以下问题：

1) 转义字符中只能使用小写字母，每个转义字符只能看作一个字符。

2) \v 垂直制表和 \f 换页符对屏幕没有任何影响，但会影响打印机执行响应操作。

3) 在C程序中，使用不可打印字符时，通常用转义字符表示。

2.5.4 符号常量

C语言允许将程序中的常量定义为一个标识符，称为符号常量。符号常量一般使用大写英文字母表示，以区别于一般用小写字母表示的变量。符号常量在使用前必须先定义，定义的形式是：

```
#define <符号常量名> <常量>
```

例如：

```
#define PI     3.1415926
#define TRUE   1
#definr FALSE  0
#define STAR  '*'
```

这里定义PI、TRUE、FLASE、STAR为符号常量，其值分别为 3.1415926，1，0，'*'。

#define是C语言的预处理命令，它表示经定义的符号常量在程序运行前将由其对应的常量替换。

定义符号常量的目的是为了提高程序的可读性，便于程序的调试和修改。因此在定义符号常量名时，应使其尽可能地表达它所代表的常量的含义，例如前面所定义的符号常量名PI(π)，表示圆周率3.1415926。此外，若要对一个程序中多次使用的符号常量的值进行修改，只须对预处理命令中定义的常量值进行修改即可。

2.5.5 字符变量

字符变量用来存放字符常量，注意只能存放一个字符，不要以为在一个字符变量中可以放字符串。

字符变量的定义形式如下：

```
char c1, c2;
```

它表示c1和c2为字符变量，各放一个字符。因此可以用下面语句对c1、c2赋值：

```
c1 = 'a';     c2 = 'b';
```

[例2-2]

```
main()
  {
       char c1,c2;
       c1=97;  c2=98;
       printf("%c %c",c1,c2);
  }
```

c1、c2被指定为字符变量。但在第3行中，将整数97和98分别赋给c1和c2，它的作用相当于以下两个赋值语句：

```
c1='a';    c2='b';
```

因为'a'和'b'的ASCII码为97和98。第4行将输出两个字符。"%c"是输出字符的格式。程序输出：

```
RUN
   a      b
```

[例2-3]

```
main()
  {    char c1,c2;
       c1='a' ;c2='b';
       c1 = c1 - 32;c2 =c2 - 32;
       printf("%c  ␣%c",c1,c2);
  }
```

运行结果为：

```
RUN
   A      B
```

它的作用是将两个小写字母转换为大写字母。因为'a'的ASCII码为97，而'A'为65，'b'为98，'B'为66。从ASCII代码表中可以看到每一个小写字母比大写字母的ASCII码大32。即'a'='A' + 32。

2.6 运算符

C语言的内部运算符很丰富，运算符是告诉编译程序执行特定算术或逻辑操作的符号。 C语言有三大运算符：算术、关系与逻辑、位操作。另外， C还有一些特殊的运算符，用于完成一些特殊的任务。

2.6.1 算术运算符

表2-5列出了C语言中允许的算术运算符。在 C语言中，运算符“ + ”、“ － ”、“ * ”和“ / ”的用法与大多数计算机语言的相同，几乎可用于所有 C语言内定义的数据类型。当“ / ”被用于整数或字符时，结果取整。例如，在整数除法中， 10/3=3。

一元减法的实际效果等于用 -1乘单个操作数，即任何数值前放置减号将改变其符号。模运算符“ % ”在C语言中也同它在其它语言中的用法相同。切记，模运算取整数除法的余数，所以“ % ”不能用于float和double类型。

表2-5 算术运算符

运 算 符	作 用	运 算 符	作 用
－	减法，也是一元减法	%	模运算
+	加法	--	自减（减1）
*	乘法	++	自增（增1）
/	除法		

下面是说明%用法的程序段。

```
int x,y;
x=10;
y=3;
printf("%d",x/y);              /* 显示 3 */
printf("%d",x%y);              /* 显示 1,整数除法的余数 */

x=1;
y=2;
printf("%d,%d",x/y,x%y);       /* 显示 0,1 */
```

最后一行打印一个0和一个1，因为1/2整除时为0，余数为1，故1%2取余数1。

2.6.2 自增和自减

C语言中有两个很有用的运算符，通常在其它计算机语言中是找不到它们的——自增和自减运算符，++和--。运算符“ ++ ”是操作数加1，而“ -- ”是操作数减1，换句话说：

```
x=x+1; 同++x;
x=x-1; 同--x;
```

自增和自减运算符可用在操作数之前，也可放在其后，例如： x=x+1；可写成++x；或x++；但在表达式中这两种用法是有区别的。自增或自减运算符在操作数之前， C语言在引用

操作数之前就先执行加1或减1操作；运算符在操作数之后，C语言就先引用操作数的值，而后再进行加1或减1操作。请看下例：

```
x=10;
y=++x;
```

此时，y=11。如果程序改为：

```
x=10;
y=x++;
```

则y=10。在这两种情况下，x都被置为11，但区别在于设置的时刻，这种对自增和自减发生时刻的控制是非常有用的。

在大多数C编译程序中，为自增和自减操作生成的程序代码比等价的赋值语句生成的代码要快得多，所以尽可能采用加1或减1运算符是一种好的选择。

下面是算术运算符的优先级：

最高 ++、--

-（一元减）

*、/、%

最低 +、-

编译程序对同级运算符按从左到右的顺序进行计算。当然，括号可改变计算顺序。 C语言处理括号的方法与几乎所有的计算机语言相同：强迫某个运算或某组运算的优先级升高。

2.6.3 关系和逻辑运算符

关系运算符中的“关系”二字指的是一个值与另一个值之间的关系，逻辑运算符中的“逻辑”二字指的是连接关系的方式。因为关系和逻辑运算符常在一起使用，所以将它们放在一起讨论。

关系和逻辑运算符概念中的关键是True（真）和Flase（假）。C语言中，非0为True，0为Flase。使用关系或逻辑运算符的表达式对Flase和Ture分别返回值0或1(见表2-6)。

表2-6 关系和逻辑运算符

关系运算符	含 义	关系运算符	含 义
>	大于	<=	小于或等于
>=	大于等于	==	等于
<	小于	!=	不等于

逻辑运算符	含 义
&&	与
||	或
!	非

表2-6给出于关系和逻辑运算符，下面用1和0给出逻辑真值表。

关系和逻辑运算符的优先级比算术运算符低，即像表达式10>1+12的计算可以假定是对表达式10>(1+12)的计算，当然，该表达式的结果为Flase。

在一个表达式中允许运算的组合。例如：

```
10>5&&!(10<9)||3<=4
```

p	q	p&&q	p||q	!p
0	0	0	0	1
0	1	0	1	1
1	1	1	1	0
1	0	0	1	0

这一表达式的结果为True。

下表给出了关系和逻辑运算符的相对优先级：

最高 ！

>= <=

== !=

&&

最低 ||

同算术表达式一样，在关系或逻辑表达式中也使用括号来修改原计算顺序。

切记，所有关系和逻辑表达式产生的结果不是 0就是1，所以下面的程序段不仅正确而且将在屏幕上打印数值1。

```
int x;
x=100;
printf("%d",x>10);
```

2.6.4 位操作符

与其它语言不同，C语言支持全部的位操作符（Bitwise Operators）。因为C语言的设计目的是取代汇编语言，所以它必须支持汇编语言所具有的运算能力。位操作是对字节或字中的位（bit）进行测试、置位或移位处理，这里字节或字是针对 C标准中的char和int数据类型而言的。位操作不能用于 float、double、long double、void或其它复杂类型。表 2-7给出了位操作的操作符。位操作中的 AND、OR和NOT（1的补码）的真值表与逻辑运算等价，唯一不同的是位操作是逐位进行运算的。

表2-7 位操作符

操 作 符	含 义	操 作 符	含 义
&	与（AND）	~	1的补（NOT）
|	或（OR）	>>	右移
^	异或（XOR）	<<	左移

下面是异或的真值表。

表2-8 异或的真值表

P	q	p^q
0	0	0
1	0	1
1	1	0
0	1	1

如表2-8所示，当且仅当一个操作数为True时，异或的输出为True，否则为Flase。

位操作通常用于设备驱动程序，例如调制解调器程序、磁盘文件管理程序和打印机驱动程序。这是因为位操作可屏蔽掉某些位，如奇偶校验位（奇偶校验位用于确保字节中的其它位不会发生错误通常奇偶校验位是字节的最高位）。

通常我们可把位操作AND作为关闭位的手段，这就是说两个操作数中任一为0的位，其结果中对应位置为0。例如，下面的函数通过调用函数read_modem()，从调制解调器端口读入一个字符，并将奇偶校验位置成0。

[例2-4]

```
Char get_char_from_modem()
{
char ch;
ch=read_modem();  /从调制解调器端口中得到一个字符 */
return(ch&127);
}
```

字节的位8是奇偶位，将该字节与一个位1到位7为1、位8为0的字节进行与操作，可将该字节的奇偶校验位置成0。表达式ch&127正是将ch中每一位同127数字的对应位进行与操作，结果ch的位8被置成了0。在下面的例子中，假定ch接收到字符"A"并且奇偶位已经被置位。

```
奇偶位
↓
 110000001   内容为‘A’的ch，其中奇偶校验位为1
 011111111   二进制的127执行与操作
 &           与操作
=010000001   去掉奇偶校验的‘A’
```

位操作OR与AND操作相反，可用来置位。任一操作数中为1的位将结果的对应位置1。如下所示，128|3的情况是：

```
1000000   128的二进制
0000011   3的二进制
|         或操作
=1000011  结果
```

异或操作通常缩写为XOR，当且仅当做比较的两位不同时，才将结果的对应位置位。如下所示，异或操作127^120的情况是：

```
01111111  127的二进制
01111000  120的二进制
^         异或操作
=00000111 结果
```

一般来说，位的AND、OR和XOR操作通过对操作数运算，直接对结果变量的每一位分别处理。正是因为这一原因（还有其它一些原因），位操作通常不像关系和逻辑运算符那样用在条件语句中，我们可以用例子说明这一点：假定 X=7，那么x&&8为Ture(1),而x&8却为Flase(0)。

记住，关系和逻辑操作符结果不是0就是1。而相似的位操作通过相应处理，结果可为任

意值。换言之，位操作可以有0或1以外的其它值，而逻辑运算符的计算结果总是0或1。

移位操作符>>和<<将变量的各位按要求向或向左移动。右移语句通常形式是：

variable >>右移位数

左移语句是：

variable<<左移位数

当某位从一端移出时，另一端移入0（某些计算机是送1，详细内容请查阅相应C编译程序用户手册）。切记：移位不同于循环，从一端移出的位并不送回到另一端去，移去的位永远丢失了，同时在另一端补0。

移位操作可对外部设备（如D/A转换器）的输入和状态信息进行译码，移位操作还可用于整数的快速乘除运算。如表2-9所示（假定移位时补0），左移一位等效于乘2，而右移一位等效于除以2。

表2-9 用移位操作进行乘和除

字 符 x	每个语句执行后的x	x 的 值
x=7	00000111	7
x<<1	00001110	14
x<<3	01110000	112
x<<2	11000000	192
x>>1	01100000	96
x>>2	00011000	24

每左移一位乘2，注意x<<2后，原x的信息已经丢失了，因为一位已经从一端出，每右移一位相当于被2除，注意，乘后再除时，除操作并不带回乘法时已经丢掉的高位。

反码操作符为~。~的作用是将特定变量的各位状态取反，即将所有的1位置成0，所有的0位置成1。

位操作符经常用在加密程序中，例如，若想生成一个不可读磁盘文件时，可以在文件上做一些位操作。最简单的方法是用下述方法，通过1的反码运算，将每个字节的每一位取反。

原字节 00101100

第一次取反码 11010011

第二次取反码 00101100

注意，对同一行进行连续的两次求反，总是得到原来的数字，所以第一次求反表示了字节的编码，第二次求反进行译码又得到了原来的值。

可以用下面的函数encode()对字符进行编码。

[例2-5]

```
char encode(ch)
char ch;
{
return (~ch);
}
```

2.6.5 ?操作符

C语言提供了一个可以代替某些if-then-else语句的简便易用的操作符？。该操作符是三元

的，其一般形式为：

```
EXP1?EXE2:EXP3
```

EXP1，EXP2和EXP3是表达式，注意冒号的用法和位置。

操作符“ ？ ”作用是这样的，在计算EXP1之后，如果数值为True，则计算EXP2，并将结果作为整个表达式的数值；如果 EXP1的值为Flase，则计算EXP3，并以它的结果作为整个表达式的值，请看下例：

```
x=10;
y=x>9?100:200;
```

例中，赋给y的数值是100，如果x被赋给比9小的值，y的值将为200，若用if-else语句改写，有下面的等价程序：

```
x=10;
if(x>9) y=100;
else y=200;
```

有关C语言中的其它条件语句将在第3章进行讨论。

2.6.6 逗号操作符

作为一个操作符，逗号把几个表达式串在一起。逗号操作符的左侧总是作为 void(无值)，这意味着其右边表达式的值变为以逗号分开的整个表达式的值。例如：

```
x=(y=3,y+1);
```

这行将3赋给y，然后将4赋给x，因为逗号操作符的优先级比赋值操作符优先级低，所以必须使用括号。

实际上，逗号表示操作顺序。当它在赋值语句右边使用时，所赋的值是逗号分隔开的表中最后那个表达式的值。例如，

```
y=10;
x=(y=y-5,25/y);
```

执行后，x的值是5，因为y的起始值是10，减去5之后结果再除以25，得到最终结果。

在某种意义上可以认为，逗号操作符和标准英语的 and是同义词。

2.6.7 关于优先级的小结

表2-10列出了C语言所有操作符的优先级，其中包括将在本书后面讨论的某些操作符。注意，所有操作符（除一元操作符和？之外）都是左结合的。一元操作符（ *，&和-）及操作符“ ？ ”则为右结合。

表2-10 C语言操作符的优先级

最 高 级	() [] → ! ~ ++ -- -(type) * & sizeof * / % + - << >> <= >= == !=

（续）

	&
	^
	|
	&&
	||
	?
	= += -= *= /=
最低级	,

2.7 表达式

表达式由运算符、常量及变量构成。C语言的表达式基本遵循一般代数规则，有几点却是与C语言紧密相关的，以下将分别加以讨论。

2.7.1 表达式中的类型转换

混合于同一表达式中的不同类型常量及变量，应均变换为同一类型的量。C语言的编译程序将所有操作数变换为与最大类型操作数同类型。变换以一次一操作的方式进行。具体规则如下：

```
char ch;
int  i;
float f;
double d;
result=(ch / i) + ( f * d ) - ( f + i );
```

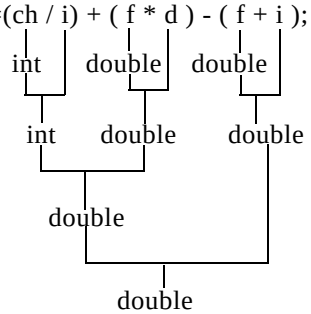

图2-1 类型转换实例

1) 所有char及short int 型量转为int型，所有float转换为double。

2) 如操作数对中一个为 long double，另一个转换为 long double。① 要不然，一个为double，另一个转为double。② 要不然，一个为long，另一个转为long。③ 要不然，一个为unsigned，另一个转为unsigned。

一旦运用以上规则。每一对操作数均变为同类型。注意，规则 2)有几种必须依次应用的条件。

图2-1示出了类型转换。首先， char ch转换成 int，且float f 转换成double；然后ch/i的结果转换成double，因为f*d是double；最后由于这次两个操作数都是 double，所以结果也是

double.

2.7.2 构成符cast

可以通过称为cast的构成符强迫一表达式变为特定类型。其一般形式为：

(type) expression

(type)是标准C语言中的一个数据类型。例如，为确保表达式 x/2的结果具有类型 float，可写为：

(float) x/2

通常认为cast是操作符。作为操作符， cast是一元的，并且同其它一元操作符优先级相同。

虽然cast在程序中用得不多，但有时它的使用的确很有价值。例如，假设希望用一整数控制循环，但在执行计算时又要有小数部分。

[例2-6]

```
main()
{
int i ;
for (i+1;i<=100;++i)
printf("%d/2 is :%f",i,(float)i/2);
}
```

若没有cast(float)，就仅执行一次整数除；有了 cast就可保证在屏幕上显示答案的小数部分。

2.7.3 空格与括号

为了增加可读性，可以随意在表达式中插入tab和空格符。例如，下面两个表达式是相同的。

```
x=10/y*(127/x);
x=10/y*(127/x);
```

冗余的括号并不导致错误或减慢表达式的执行速度。我们鼓励使用括号，它可使执行顺序更清楚一些。例如，下面两个表达式中哪个更易读一些呢？

```
x=y/2-34*temp&127;
x=(y/2)-((34*temp)&127);
```

2.7.4 C语言中的简写形式

C语言提供了某些赋值语句的简写形式。例如语句：

```
x=x+10;
```

在C语言中简写形式是：

```
x+=10 ;
```

这组操作符对 +=通知编译程序将 X+10的值赋予 X。这一简写形式适于 C语言的所有二元操作符（需两个操作数的操作符）。在C语言中，

```
variable=variable1 operator expression;
```

与variable1 operator=expression相同。

请看另一个例子：

```
x=x-100;
```

其等价语句是

```
x-=100;
```

简写形式广泛应用于专业C语言程序中，希望读者能熟悉它。