第 3 章

运算与操作

3.1 表达式

表达式是 C 程序中另一基本概念。量是 C 语言的数据处理对象，表达式则是处理数据对象的方法和步骤，它们的有机组合才构成了一条一条的语句，并由此构成最终的程序。

在 C 程序中，表达式总是由操作符(运算符)和操作数组成。从数学上讲，表达式规定了数据对象的运算过程。例如，表达式 a+b 表示两个数 a 与 b 的加法运算。

3.1.1 操作数

在表达式 a+b 中，a、b 就是最简单的操作数。一般地，表达式的操作数总是为：

(1) 文字常数与符号常数。任何显式地出现在程序文件中数据值都是文字常数，如整数 0 与浮点数格式的圆周率 3.14。

(2) 量。那些在程序中按照特定类型定义的数据对象都是量。量既可以是常量也可以是变量。

(3) 函数调用。某些变量的值在程序开始运行时是不存在的，而只有在完成了某项操作后才可能出现。为了使用变量的值参与表达式运算，就必须执行该操作。在 C 程序中，完成某项操作的典型技术手段就是函数，并在需要获得该变量值时进行函数调用。

(4) 括号。在表达式中出现小括号对括起来的表达式也是允许的，其意义与普通数学运算中的小括号类似，主要用于解决表达式各个成分的计算顺序问题。例如，表达式 (a+b) * c 保证了 a、b 的加法运算一定会在乘法运算之前进行，如果没有括号，则会首先进行 b、c 乘法运算。

3.1.2 操作符与表达式求值

C 语言的操作符种类十分丰富，它既有熟悉的算术运算、逻辑运算和关系运算，也支持一些特殊的操作，如指针及位操作等。

最简单的表达式是只含一个常量或变量的表达式，即只含一个操作数而不含操作符。

一个表达式完成一个或多个操作，并最终得到一个结果(即表达式的值)。程序运行时，计算表达式值的过程称为表达式求值。对于任何操作符，必须获得它所需要的操作数才能完成其计算过程。例如，要计算 a+b 的值，程序首先必须获得 a、b 的值，然后进行加

法运算。

当表达式中所有操作符都已计算完毕，程序将得到一个数据值，该值就是表达式的运算结果(或称表达式的值)，结果的数据类型由参加运算的操作数类型决定，此后程序就可以输出该值或者将其作为操作数再参与以后的操作。

3.2 算术操作符

C 程序中的算术操作符既包含标准的数学算术运算符，也包含一些仅在 C 程序中才存在的算术操作符变体。

3.2.1 基本算术操作符

基本的算术操作符包括加(+)、减(-)、乘(*)、除(/)与取模(%)等 5 种。注意，键盘上并没有乘法操作符“×”，所以使用“*”代替。这些操作符接受两个操作数。一个位于操作符左边，为左操作数；一个位于操作符右边，为右操作数。称需要两个操作数的操作符为二元操作符。

C 程序的基本算术操作符与数学运算符的意义基本相同。例外情况是如果除法操作符作用于整数类型的数据对象上，C 语言规定其运算结果仍然为整数，即进行所谓的整数除法。例如，7/2 的结果为 3，而不是 3.5，余数被舍弃，也不会进行四舍五入。

为了表达除法的余数，C 语言定义了一个新的操作符——取模或取余%，如 7%2 的结果为整数除法后的余数 1。

C 语言中也存在只有单个操作数的操作符，如取负值的“-”操作符：

```
int a=1;
int b=-a;
```

在复杂表达式中可能存在多个操作符与操作数。例如，a*3+b*2，其中的 a*3 与 b*2 就称为子表达式。要计算表达式的值，所有子表达式的值都必须先计算完毕。

3.2.2 递增递减操作符

在 C 程序中有时需要对某个数据对象进行加一减一操作，总是使用：

```
a=a+1;
b=b-1;
```

这样的方式很不方便。

为此，C 语言特别引入了两个一元操作符“++”与“--”来表达对数据对象值进行递增递减 1 的操作。例如上述语句其实可以书写为：

```
a++;
b--;
```

在 C 语言中，递增和递减操作符非常特殊，它们既可以出现在数据对象的前面，也可以出现在数据对象的后面。当出现在数据对象前时称为前缀递增或前缀递减操作符；出

现在数据对象后时称为后缀递增或后缀递减操作符。例如，上述语句还可以书写成：

```
++a;
--b;
```

注意：前缀递增递减操作符和后缀递增递减操作符的优先级是不同的，虽然它们的优先级都非常高，但后缀递增递减操作符的优先级更高(后者位于最高一级)。

当递增递减操作符出现在复杂表达式中时，其求值按照特殊规则进行。具体原则是，若递增递减操作符为前缀操作符，则该数据对象值在参与表达式运算前先进行递增递减；若递增递减操作符为后缀操作符，则该数据对象值在参与表达式运算后再进行递增递减。

例如，考虑下述语句：

```
int a=1, b=2, c;
c=++a*b;
```

a、b、c 的值分别为多少？答案是 2、2、4。前缀递增操作符首先计算，a 值递增 1 变为 2，然后再参与乘法运算，c 的结果自然为 4。

将最后一条赋值语句更换为：

```
c=a++*b;
```

a、b、c 的值分别为多少？答案是 2、2、2。程序首先取 a 值 1 与 2 相乘，c 的结果为 2，然后再进行 a 的后缀递增操作，a 值也变为 2。

尽量不要在复杂表达式中使用递增递减操作符。

3.3　赋值操作符

赋值操作符是完成将某个数据对象的值传递给另一数据对象的典型技术手段。在 C 程序中，赋值操作符使用“=”表示。

3.3.1　赋值操作符与赋值表达式

如果一个表达式中带有一个赋值符号，此时就称该表达式为赋值表达式。典型的赋值表达式如下：

```
int a, b;                    /*变量定义*/
a=a+1;                       /*直接给变量赋值*/
b=b+a;                       /*直接给变量赋值*/
```

再如：

```
const double pi=3.1416;      /*常量定义*/
double area, radius;         /*变量定义*/
radius=1.3;                  /*直接给变量赋值*/
area=pi*radius*radius;       /*计算右侧子表达式的值后再给变量赋值*/
```

赋值语句兼有表达式计算和赋值的双重功能。赋值号“=”将其右边表达式的运算结

果赋值给左边的目标变量。

注意：赋值号左边一定是变量名(或等价于变量的数组元素名)，赋值语句对该变量实施写操作。

当右边表达式的计算结果类型与左边变量的类型不一致，则先将右边表达式的值转换为与左边目标变量相同的类型，然后进行赋值。例如：

```
i=d+1;
```

其中 i 为 int 类型，d 为 double 类型。于是此运算的处理过程是先将 1 转换为 double 类型(1.000000)，然后执行 d+1.000000，结果为 double 类型，最后再把 double 类型的结果转换为 int 类型再赋给 i。

当然这种由“长型”向“短型”的转换可能会导致数据精度上的一些损失。例如，将浮点数赋值给一个整型变量，会舍弃小数点后面的数据，此种情况称之为“截断”。

3.3.2 复合赋值表达式

在 C 程序中，类似于

```
x=x+a;
```

这样的赋值语句很多，虽然在数学上可能是错误的，但在程序设计中却是有效的——它隐含了修改变量值的过程。

C 语言特别规定，上述赋值语句可以使用下述简写形式：

```
x+=a;
```

称这样的“＋＝”操作符为加赋。类似地，还有减赋(“－＝”)、乘赋(“＊＝”)、除赋(“/＝”)、模赋(“%＝”)等。例如：

```
x+=a;        等价于      x=x+a;
x-=a;        等价于      x=x-a;
x*=a;        等价于      x=x*a;
x/=a;        等价于      x=x/a;
x&=a;        等价于      x=x%a;
```

注意：赋值语句的简写形式并不是简单的文本替换，例如：

```
x*=a+b;
```

表示

```
x=x*(a+b);
```

而不是

```
x=x*a+b;
```

也就是说，在进行复合赋值时，操作符右边的子表达式首先得到计算，然后才将整个表达式转换为标准加法与赋值的两步操作。

C 语言引入复合赋值操作符的主要目的是为了方便程序员的编码。

3.4　表达式求值

C 语言的表达式求值过程比较复杂，除了要遵照与数学上的运算类似的操作符优先级与结合性之外，还需要遵照一些特殊的规定。

3.4.1　表达式的一般求值规则

在解决实际问题时，表达式可能非常复杂，并且包含多种算术操作符。这些算术操作符的计算顺序像数学表达式一样由其优先级和结合性确定。例如对于上面给出的表达式 a＊3＋b＊2，首先获得计算的是两个乘法操作，其次才是加法操作，这种计算顺序与数学上的“先乘除后加减”完全相同。

当两个操作符的优先级相同时，表达式的计算顺序由操作符的结合性确定。例如，从左到右的结合性（简称左结合）表示先计算左边的操作符，再计算右边的操作符。

3.4.2　操作符的优先级与结合性

C 程序还包含了很多数学上并不存在的操作符，这些操作符的计算顺序比较复杂，表 3-1 列出了 C 语言全部操作符的优先级，其中优先级数字大的优先级较高，一般在表达式中最先获得计算。

表 3-1　操作符的优先级

优先级	操作符	操作符名称
15	()	函数调用
	[ ]	下标
	.	选员
	－＞	选员
	＋＋	后缀递增
	－－	后缀递减
14	sizeof	对象尺寸
	sizeof()	类型尺寸
	!	逻辑取非
	～	按位取反
	＋	一元加
	－	一元减
	＋＋	前缀递增
	－－	前缀递减
	&	取址
	＊	引领
	(type)	类型转换

续表

优先级	操作符	操作符名称
13	*	乘法
	/	除法
	%	取模
12	+	加法
	-	减法
11	<<	左移
	>>	右移
10	<	小于
	>	大于
	<=	不大于
	>=	不小于
9	==	等于
	!=	不等于
8	&	位与
7	^	位异或
6	|	位或
5	&&	逻辑与
4	||	逻辑或
3	? :	条件表达式
2	=	简单赋值
	+=	加赋
	-=	减赋
	*=	乘赋
	/=	除赋
	%=	模赋
	<<=	左移赋
	>>=	右移赋
	&=	位与赋
	|=	位或赋
	^=	位异或赋
1	,	顺序求值

注意：这里说的“一般”真的就只是一般——确实有例外。

按照 C 语言规定，在表达式中，递增递减操作符的执行时机并不按照本表执行，而是遵照自己的独立规定。

建议读者不必背诵此表，当把握不准某些操作符的优先级时，多加括号即可，C 语言的编译器会自动忽略不必要的括号。另外，尽量不要在程序中书写复杂表达式，尤其是带有递增递减操作符的复杂表达式。

3.5　逗号操作符

在 C 语言中，逗号也是一种运算符，用逗号把几个运算表达式连接起来所构成的表达式叫逗号表达式。

一般地，逗号表达式的运算模式是自左向右逐个计算每个表达式的值，并将最后一个表达式的值作为逗号表达式的运算结果。

例如，在下面的赋值语句中，括号里边就是一个逗号表达式：

```
z=(x=m+n, y=m-n, x / y);
```

如果 m 为 5，n 为 3，则 z 的值为 4。

逗号表达式将几个前后相关的表达式联系起来，构成单一的表达式，也可以起到简化（或者“复杂化”）代码的作用。

除非特别必要，不建议读者在程序中使用逗号表达式。多说一句，按照笔者的理解，这里所说的“特别必要”仅限于循环语句的循环头部，第 6 章将专门讨论这个问题。

3.6　混合运算与类型转换

在一个表达式中，同类型数据的运算（如两个整数相加、两个实数相乘）是没有问题的。但是不同类型的数据由于长度不同，在内存中存储的方式不同，若不加处理而直接运算是有问题的。在编程时，如果表达式中参与运算的数据类型出现不一致，情况会如何呢？

C 语言允许对不同类型的数值型数据进行混合运算，只是在实际运算前先按一定规则将不同类型数据转换成相同类型数据，然后再进行计算。

数据类型的转换有隐式类型转换和显式类型转换两种方式。

3.6.1　隐式类型转换

隐式类型转换是在编译时由编译程序按照一定规则自动完成，不需要人为干预。如果在表达式中有不同类型的两个数据参与同一数学运算，编译器就将其中一个数据转换为与另一个数据相同的类型。C 语言规定的转换规则是由“短型”向“长型”转换。例如，一个整型数与一个实型数相加，系统会将整型数转换为数值上等价的实型数后再参与运算。这种转换规则可以确保运算的精度不会降低。

【例 3-1】 指出下述代码执行后 result 的结果。

```
char c='A';
int i=6;
float f=10.0;
double d=0.2, result;
result=(c/i)+(4+f * d) * (f-i) ;  //发生类型转换
```

运算过程如下。

(1) 首先计算 c / i：c、i 的类型不同，c 由 char 转换为 int 型，由于字母'A'的 ASCII 码为 65，所以运算变为 65 / 6，结果为整数 10，而不是 10.833333。

(2) 接下来计算 4+f * d：f 值转换为 double 型，与 d 相乘后，再与常数 4(也变为双精度实型数)相加。其结果为 double 类型的 6.000000。

(3) 计算 f—i：即 float 类型数据与 int 类型数据进行计算，int 型转换为 float 型。运算结果为 float 类型的 4.000000。

(4) 对三个中间结果进行计算：

```
6.000000(double 类型) * 4.000000(转换为 double 类型)=24.000000(double 类型)
10(转换为 double 类型)+24.000000(double 类型)=34.000000(double 类型)
```

在赋值语句中，如果赋值号(等号)左右两端的类型不同，则是将赋值号右边的值转换为赋值号左边的类型，然后再赋值。

3.6.2 显式类型转换

显式类型转换不是按照前面所述的转换规则由系统自动进行转换，而是由程序员指定将一种类型的数据转换成另一种数据类型。显式类型转换可在很多情况下简化转换，提高程序执行效率。

如将一浮点数赋值给整数：

```
int a;
double g=9.80665;
a=g;
```

其中，数据对象 a、g 的类型显然是不同的，但是 C 编译器仍然接受这样的赋值操作，其结果为 9。小数点后的数据被丢弃。

因为赋值过程导致部分信息丢失，程序员可能会对结果数据的准确性产生怀疑。为避免隐式类型转换对程序员理解程序的负面影响，推荐使用显式类型转换将此过程编码出来，即如果类型转换是必要的，就应该直接将它表达在程序中：

```
a=(int)g;
```

显式类型转换的基本格式是在待转换的数据对象或表达式前面使用小括号括起来的转换后目标数据类型进行限定，这里的括号与类型名称都是不能省略的。

显式类型转换总在用户规定的时刻进行，而隐式类型转换仅在必要时才由系统进行，因此不恰当的使用显式类型转换可能使得程序结果是错误的，如对于下述代码：

```
a=(int)g * 4;
```

a 的最终结果为 36，与精确值 39.2266 有较大差距。

导致此现象的原因是类型转换操作符的优先级比乘法运算优先级高，如果需要表达式值的类型转换，则应该如下编码：

```
a=(int)(g * 4);
```

如此将使得 a 的最终结果为 39。

注意：无论是隐式类型转换还是显式类型转换，都仅是为了本次运算或赋值需要而对数据所做的临时性转换，变量本身的值与类型并没有发生任何变化。例如在上述代码中，即使将 g 作为整数参与运算，g 本身也仍然为 double 类型，并且其值也依然为 9.80665。

习　题　3

1. 如下代码的输出结果是什么？

```
char c='z';
int k=7;
c-=k;
k=c%(k+60)+20;
printf("%d, %c\n", c, k);
```

2. 编写程序，接受用户通过键盘输入的任意字母并输出其 ASCII 码值。

3. 编写程序，接受用户通过键盘输入的 4 个整数，输出它们的和与平均值。提示：如果使用整数表示平均值，除法结果可能并不准确。若要精确表达最终结果，平均值应该使用 double 类型，并且在进行除法运算时，至少一个操作数必须为双精度浮点数，这保证结果为 double 类型而不是整数类型。