第3章　运算符、表达式和语句

C语言中大部分的操作都是通过运算符(Operator)实现的。C语言提供了非常丰富的运算符,如算术运算符、关系运算符、逻辑运算符、赋值运算符、位运算符、强制类型转换运算符等。

表达式(Expression)由运算符连接运算对象组成。对应于运算符的类型,表达式也可以分为算术表达式、关系表达式、逻辑表达式、赋值表达式等。任何表达式都有值和类型。

语句(Statement)是C程序的基本组成单位,大多数的语句都由表达式构成。语句的种类很多,如表达式语句、函数调用语句、流程控制语句等。

本章重点讨论算术运算符和表达式、赋值运算符和表达式以及相应的表达式语句,旨在让读者能够编写不包含复杂流程控制的简单程序。3.1节分别介绍算术运算符、赋值运算符和递增、递减运算符及它们对应的表达式;3.2节讨论数据类型的隐式转换和强制类型转换运算符的概念;3.3节介绍表达式语句的概念;3.4节是对本章的内容进行总结。

3.1　运算符和表达式

3.1.1　算术运算符和算术表达式

C语言的算术运算符如表3-1所示。

表3-1　算术运算符

一元运算符	二元运算符	
	乘除类	加减类
正号运算符+ 负号运算符−	乘法运算符* 除法运算符/ 模运算符%	加法运算符+ 减法运算符−

二元运算符要求有两个操作数,而一元运算符只要求有一个操作数。使用算术运算符时应注意以下5点。

(1) 运算符%表示模运算(mod)或取余运算(rem)。表达式a%b的值是a除以b后的余数。例如,10%2的值为0,10%3的值为1。

(2) %运算符要求两个操作数必须是整数,其他运算符允许操作数可以是整数或实数。

(3) 当操作数均是整型时,运算符/的计算结果也是整型(在C语言中,算术运算结果的类型和操作数的类型相同),结果是通过舍去小数部分得到。所以,1/2的结果是0而不是0.5,要想使结果为0.5可以使用1.0/2或1/2.0或1.0/2.0。

(4) 避免使运算符/和运算符%的第二个操作数为0。

(5) 对于运算符/和运算符%,若两个操作数均为正数,计算结果比较容易确定。若操作数中含有负数,计算结果由程序的运行环境决定。例如,对于-10/3和-10%3,一种处理方式是-10/3等于-3,-10%3等于-1;另一种处理方式是-10/3等于-4,-10%3等于2。但无论哪种处理方式都应使(a/b)*b+(a%b)等于a。

使用算术运算符将操作数连接起来得到的式子称为算术表达式。算术表达式也有数据类型和值。算术表达式的类型由参与运算的运算数决定,值就是表达式的计算结果。

3.1.2 运算符的优先级和结合性

1. 优先级(Priority)

考虑表达式i+j*k,其含义应为i+(j*k),而不应该是(i+j)*k。这是因为数学中有一个"先进行乘除运算,后进行加减运算"的基本准则,它表明乘除运算的优先级高于加减运算。在C语言中,当表达式中含有多个运算符的时候,也要根据运算符的优先级对表达式进行解释,这和人们在表达式中使用括号是一样的含义。优先级高的运算符先进行计算。

C语言中的运算符优先级共分15级,1级最高,15级最低。算术运算符的优先级如表3-2所示。

表3-2 算术运算符的优先级

算术运算符	优先级
+(正号)、-(负号)	2
*、/、%	3
+(加)、-(减)	4

所以:

$$i+j*k \Leftrightarrow i+(j*k)$$

$$-i*-j \Leftrightarrow (-i)*(-j)$$

2. 结合性(Association)

考虑表达式i+j+k,其含义应为(i+j)+k,而不是i+(j+k)。这是因为数学中加法运算是从左向右进行结合的。当表达式中含有多个优先级相同的运算符时,运算符的结合性开始起作用。如果运算符是从左向右结合的,称这种运算符是左结合的。如果运算符是从右向左结合的,称这种运算符是右结合的。二元算术运算符都是左结合的,一元算术运算符都是右结合的。所以:

$$i+j+k \Leftrightarrow (i+j)+k$$

$$i*j/k \Leftrightarrow (i*j)/k$$

为了正确地使用运算符,一种做法是记住或查看这些运算符的优先级和结合性规则,另一种做法就是在表达式中加括号(括号的优先级最高,1级)。

3.1.3 赋值运算符与表达式

C语言把修改变量的值的操作称为赋值操作,其对应的物理操作是将数据对应的二进

制编码存入变量对应的存储空间中。赋值运算通过赋值运算符实现。赋值运算符可分为简单赋值运算符和复合赋值运算符。

1. 简单赋值运算符

简单赋值运算符用“=”表示，它是一个二元运算符。由“=”连接左、右操作数形成的表达式称为赋值表达式。例如：

```
num = 10
```

假定 num 为 int 类型的变量，且赋值前 num 的值为 2，则赋值前后 num 变量对应的存储空间的变化如图 3-1 所示。

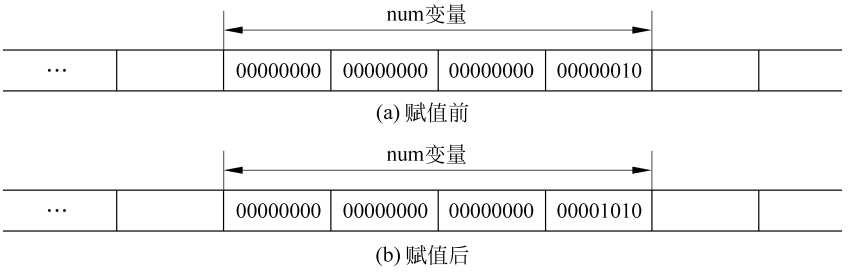

图 3-1　赋值操作前后内存空间的变化

赋值运算符的左操作数必须是一个变量；右操作数可以是一个简单的表达式，如一个常量或变量，也可以是一个由一个或多个运算符构成的复杂表达式。赋值运算符连接运算对象构成赋值表达式。赋值表达式的值和类型等于赋值后的左操作数的值和类型。因此表达式 num=10 的类型为 int 类型，值为 10。

赋值运算符的优先级是 14，远低于算术运算符的优先级；结合方向是自右向左。所以表达式 num1=num2=num3=1，等价于 num1=(num2=(num3=1))，执行过程是，先将 1 赋值给 num3，再将赋值表达式(num3=1)的值，即 1，赋值给 num2，最后将赋值表达式(num2=(num3=1))的值，即 1，赋值给 num1。表达式最终的执行结果为 num1、num2 和 num3 均赋值为 1，且整个赋值表达式的值也为 1。

2. 复合赋值运算符

C 语言提供的复合赋值运算符包括+=、-=、*=、/=、%=等。复合赋值运算符将算术运算与赋值运算合为一个运算。例如：

$$a+=1 \quad \Leftrightarrow \quad a=a+1$$

$$b-=c+1 \quad \Leftrightarrow \quad b=b-(c+1)$$

$$i*=j/k \quad \Leftrightarrow \quad i=i*(j/k)$$

与基本赋值运算符类似，复合赋值运算符的左侧只能是变量，而不能是常量或表达式。

复合赋值运算符的优先级同样是 14 级，结合方向也是自右向左。例如，表达式 a*=a+=2 等价于 a*=(a+=2)，也等价于 a=a*(a=a+2)。如果变量 a 的原值为 1，赋值运算后变量 a 的值为 9。

3.1.4 递增、递减运算符

针对C程序中频繁使用的变量加1和减1的操作，C语言提供了递增(Increment)、递减(Decrement)运算符来实现。递增和递减运算符继承自Ken Thompson编写的B语言，具有简洁和高效的优点。

递增、递减运算符分别用++和--来表示。运算符++可以使变量的值加1，运算符--可以使变量的值减1。递增、递减运算符是一元运算符，使用时有两种形式。一种形式是运算符出现在其作用的变量前面，称为前缀模式，如++num；另一种形式是运算符出现在其作用的变量后面，称为后缀模式，如num++。这两种模式都可以使变量的值加1或减1。它们的区别在于递增或递减操作执行的时间不同，具体表现为表达式的值不同，如程序清单3-1所示。

程序清单 3-1

```
#include <stdio.h>
int main(){
    int num1 = 0, num2 = 0;
    printf("表达式 num1++的值为%d\n", num1++);
    printf("num1 的值为%d\n", num1);
    printf("表达式++num2 的值为%d\n", ++num2);
    printf("num2 的值为%d\n", num2);
    return 0;
}
```

运行程序，输出结果如下。

```
表达式 num1++的值为 0
num1 的值为 1
表达式++num2 的值为 1
num2 的值为 1
```

不难看出，尽管前置递增和后置递增运算都会使变量的值加1，但两个表达式的值不同，前置递增表达式的值是变量加1后的值，而后置递增表达式的值是变量加1前的值。递减运算同理。因此，如果仅做加1或减1操作，前置模式和后置模式没有什么区别，但当需要使用表达式的值，如进行赋值操作时，一定要明白二者的区别而谨慎使用。

后缀递增、递减运算符的优先级是2级，结合方向是自左向右；前缀递增、递减运算符的优先级是2级，结合方向是自右向左。例如，表达式j=-i++等价于j=-(i++)。

使用递增、递减运算符应注意以下5个方面。

(1) 如果仅需要使变量i的值加1或减1，而不需要在表达式中引用变量的值，则i++和++i的作用是一样的；否则，需要对这两种形式严格区分。

(2) 良好的编程设计风格建议在一行语句中，一个变量最多只进行一次递增或递减操作，尤其不要出现如(++i)+(i++)的表达式。这是由于C语言标准并没有规定算术运算符的运算数的计算顺序，因此有的编译器可能会先计算左侧的表达式(++i)，而有的编译器

可能会先计算右侧的表达式(i++),这会造成在不同的编译环境下得到不同的运算结果。

(3) 递增、递减运算符的操作数只能是变量,因此常量和 i+j、i++、++i 等表达式都不能进行递增、递减运算。所以表达式 1++、(i+j)++、(i++)++、++(++i)等都是错误的。

(4) 由于和实际的机器语言指令很相似,递增、递减运算生成的机器语言代码通常比等价的赋值语句的效率要更高。

(5) 递增、递减运算符通常用于整型变量或指针变量,特别是在循环结构中应用比较普遍。

3.2　数据类型转换

计算机只能对相同类型的数据(即位数相同,存储方式相同等)进行运算。例如,计算机不能直接对一个 16 位的整数和一个 32 位的整数做加法运算,更不能将一个整数和一个浮点数进行计算。为了提高程序的灵活性,C 语言允许在表达式中使用不同类型的变量和常量,在执行计算之前自动根据一定的规则自动将表达式中的操作数转换为同一类型,然后进行计算。由于这一转换由编译器自动处理而不需要程序员的介入,因此又称为隐式转换(Implicit Conversion)。C 语言还允许程序员使用强制类型转换符主动进行显式转换(Explicit Conversion)。

3.2.1　数据类型的隐式转换

隐式转换会在以下几种情况下发生。

① 进行算术运算的操作数类型不一致时(C 语言执行所谓的常用算术转换);

② 赋值运算符的左侧变量和右侧表达式的类型不一致时;

③ printf()函数中格式转换符与输出的表达式的类型不一致时;

④ 函数调用中实际参数与对应的形式参数类型不一致时;

⑤ 函数中 return 语句返回的值与函数返回值类型不一致时。

本节重点介绍前三种情况的类型转换,其他情况将在后续章节中进行介绍。

1. 常用算术转换

常用算术转换可以用于大部分二元运算符。常用算术转换的基本策略是把操作数转换成可以安全地适用于两个数值的“最小的”数据类型。粗略地说,如果某个数据类型数据占用的字节数比另一种类型少,那么这一类型就比另一种类型更小。将较小的数据类型的变量转换成较大的数据类型称为提升(Promotion)。

假设变量 d 为 double 类型,变量 i 为 int 类型,当计算表达式 d+i 时,由于变量 d 和 i 数据类型不同,C 语言会执行常用算术转换。由于 double 类型数据占 8 个字节的存储空间,int 类型数据占 4 个字节的存储空间,将一个 double 类型的变量转换成一个 int 类型的变量会造成精度的损失,甚至有可能会造成数据错误。因此,C 语言会将 int 类型的数据转换成 double 类型。

执行常用算术转换的规则可以分为如下两种情况。

(1) 任一操作数的类型是浮点类型。此时在进行常用算术转换时遵循如下规则。

① 若一个操作数是 long double 类型,则另一个操作数将被转换为 long double 类型;

② 否则,若一个操作数是 double 类型,则另一个操作数将被转换为 double 类型;

③ 否则,若一个操作数是 float 类型,则另一个操作数将被转换为 float 类型。

(2) 两个操作数都不是浮点类型。此时在进行常用算术转换时遵循如下规则。

① 若一个操作数是 unsigned long int 类型,则另一个操作数将被转换为 unsigned long int 类型;

② 否则,若一个操作数是 long int 类型,则另一个操作数将被转换为 long int 类型;

③ 否则,若一个操作数是 unsinged int 类型,则另一个操作数将被转换为 unsigned int 类型;

④ 否则,若一个操作数是 int 类型,则另一个操作数将被转换为 int 类型;

⑤ 否则,若两个操作数都是 short 或 char 类型,则两个操作数都将被转换为 int 类型。这种提升称为整值提升(Integral Promotion)。

【例 3-1】 常用算术转换实例:分析表达式的计算结果。

假设变量如下定义:

```
int a=1;
char ch='a';
float f=10.0;
double d=20.0;
```

表达式 a/ch+a * d-f/d 的计算过程如下。

(1) 计算表达式 a/ch,首先取变量 ch 的值进行整值提升,结果为 97(int 类型),然后计算 1/97,得结果 0,表达式的类型为 int 类型。

(2) 计算表达式 a * d,首先取变量 a 的值进行类型转换得结果 1.0(double 类型),然后计算 1.0 * 20.0,得结果 20.0,表达式的类型为 double 类型。

(3) 计算表达式 f/d,首先取变量 f 的值进行类型转换得结果 10.0(double 类型),然后计算 10.0/20.0,得结果 0.5,表达式的类型为 double 类型。

(4) 计算(1)的结果加(2)的结果,首先对(1)的结果进行类型转换得结果 0(double 类型),然后计算 0+20.0,得结果 20.0(double 类型)。

(5) 计算(4)的结果减(3)的结果,即计算 20.0-0.5,得结果 19.5(double 类型)。

因此,表达式 a/ch+a * d-f/d 的值为 19.5,类型为 double。

2. 赋值运算及 printf()函数中的类型转换

进行赋值运算时类型转换的基本原则是:将赋值运算符右边表达式的值转换成赋值运算符左边变量的类型。printf()函数中的类型转换的基本原则与赋值运算类似:将参数列表中的表达式的值转换成格式转换符指定的类型。由于赋值运算和 printf()函数在进行类型转换时采用的策略相同,因此将二者放在一起介绍。

赋值运算和 printf()函数中的隐式转换策略可以概括为如下几种情况。

1) 将占用存储空间小的类型转换成占用存储空间大的类型

这种情况与常用算术转换相同,这里不再介绍。

2）将占用存储空间大的类型转换成占用存储空间小的类型

由于赋值过程中的自动类型转换是以赋值运算符左侧变量的类型为依据的。因此，当将一个较大类型（如 double）的表达式赋值给一个较小的类型（如 int）变量时，可能会造成精度的丢失，甚至会得到一个无意义的结果。对于这种情况，编译器通常会在编译时发出警告信息。具体情况如下。

（1）将占用存储空间大的浮点型值转换成占用存储空间小的浮点型值时，若要转换的值超出了可表示的范围，则出现溢出错误；若要转换的值在可表示的范围内且能确切表示时，转换结果和原值相等；若要转换的值在可表示的范围内但不能确切表示时，转换结果是原值的近似值。

（2）将占用存储空间大的整型值转换成占用存储空间小的整型值时，若要转换的值在可表示的范围，其值不变；若要转换的值超出了可表示的范围，将得到一个无意义的值，如程序清单 3-2 所示。

程序清单 3-2

```
1  #include <stdio.h>
2  int main(){
3        short num;
4        num = 65536;
5        printf("num = %d\n", num);
6        return 0;
7   }
```

（3）将浮点型值转换成整型值时，小数部分将被直接舍去。若整数部分在该整数类型的表示范围内，转换结果是整数部分；否则，将得到一个无意义的值。

程序运行结果如下。

```
num = 0
```

很明显，这个结果是无意义的。那么为什么 num 的值会变为 0？只是因为 65536 默认为 int 类型，占 4 个字节，而 num 变量只占 2 个字节，将 4 个字节的内容放到 2 个字节的存储空间中，势必会丢失一部分数据。C 语言通常按从低到高的顺序依次将数据放入变量对应的存储空间，如果存储空间已满，则丢掉剩余字节，如图 3-2 所示。图中虚线部分的高位字节内容会被丢弃，即只将 65536 的低 16 位数据放入 num 变量中，因此 num 的值为 0。

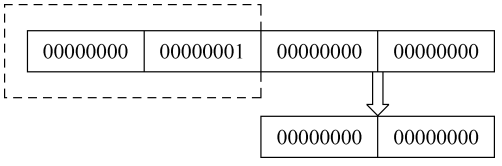

图 3-2　将 int 类型数据放入 short 类型变量

事实上，当执行 printf("%hd\n", 65536)时，C 语言也会执行上述转换，即将 65536 这个 int 类型的常量转换为一个 short 类型，最终输出结果也是 0。

3）字符型与整型的转换

字符型值向整型值转换时，unsigned char 类型值将被作为无符号整数进行处理。char 类型值既可以作为有符号整数也可以作为无符号整数进行处理，多数计算机系统是作为有符号整数进行处理的。

整型值向字符型值转换时，若要转换的值在字符型的可表示范围内，其值不变；否则，将得到一个无意义的值。如程序清单 3-3 所示。

程序清单 3-3

```
1   #include <stdio.h>
2   int main(){
3       char ch = 321;   /*第 3 行*/
4       printf("ch = %c\nch 的 ASCII 码值为%d\n", ch, ch);
5       return 0;
6   }
```

程序运行结果如下。

```
ch = A
ch 的 ASCII 码值为 65
```

代码第 3 行中整型字面量 321 默认是 int 类型，占 4 个字节，而一个 char 类型只占 1 个字节，因此在赋值前需要先将 321 转换为 char 类型，即只保留低 8 位的数据，然后将该值赋给 ch 变量，如图 3-3 所示。

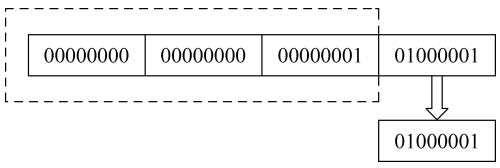

图 3-3　整数类型转字符类型

代码第 4 行中，printf()函数格式字符串中有两个格式转换符，其中%d 说明要以十进制整数的形式输出字符 ch，即输出 ch 的 ASCII 码值。需要注意的是，在 printf()的格式字符串中有两个换行符\n，也就是说当输出 ch=65 后要换行。

4）有符号整型与无符号整型的转换

有符号整数和无符号整数的转换与有符号整数类型之间的转换方式类似，只是因为二者的编码方式不同，可能会出现预料之外的结果，具体如程序清单 3-4 所示。

程序清单 3-4

```
1   #include <stdio.h>
2   int main(){
3       unsigned short us1, us2;
4       short s;
5       us1 = -1;
6       us2 = 65530;
```

```
7        s = us2;
8        printf("us1 = %u, us2 = %u, s = %d", us1, us2, s);
9        return 0;
10   }
```

程序的运行结果如下。

```
us1 = 65535, us2 = 65530, s = -6
```

代码第 5 行将一个 int 类型的常量赋值给一个 unsigned short 类型的变量。这与将一个 int 类型整数赋值给一个 short 类型变量类似，C 语言会将 int 类型数据的低 16 位放入 short 类型变量的内存空间。然而，负整数在计算机中以补码的形式存储，而 unsigned short 变量会把内存空间内的数据当成无符号数来解释，如图 3-4 所示。因此 us1 的结果为 65535。

代码第 7 行将一个 unsigned short 类型的数据赋值给一个 short 类型。由于两个变量所占的字节数相等，C 语言会直接将变量 us2 的数据放在变量 s 所占的内存空间中，如图 3-5 所示。但由于 s 为有符号类型，因此 C 语言会认为内存空间的数据是－6 的补码。

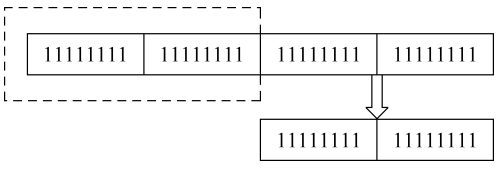

图 3-4　int 类型转 unsigned short 类型

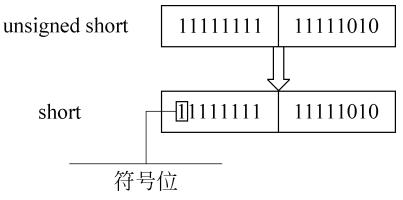

图 3-5　无符号数转换为有符号数

3.2.2　强制类型转换运算符

为了使程序设计人员更灵活地控制类型转换，C 语言提供了强制类型转换运算符(Cast Operator)。强制类型转换运算符的通用形式如下。

(类型名)表达式

强制类型转换运算符的作用是将右侧表达式的值转换成指定类型的值。比如(int)12.3 的结果是整数 12，此结果通过将 12.3 转换成整型值而得到。

强制类型转换运算符是一元运算符，其优先级为 2 级。例如表达式(float)1/2 等价于((float)1)/2，先将整数 1 转换成浮点数，然后将进行浮点数的除法运算，结果是 0.5。

强制类型转换运算不会改变右侧表达式的值和类型，而是得到一个指定类型的结果值。例如：

```
float f=1.2;
int i;
i=(int)f;
```

赋值后变量 i 的值为 1，而变量 f 的值仍然为 1.2。

3.3 表达式语句

语句是C语言程序的最小组成单位。在C语言程序中,大部分语句都是以分号结尾的。以分号结尾的表达式称为表达式语句。例如:

```
i++;
num1 = 10;
```

值得一提的是,C语言认为函数调用也是表达式,函数调用语句也属于表达式语句。例如:

```
printf("num1 = %d\n", num1);
scanf("%d", &num1);
```

另外,需要注意的是,尽管变量声明也是以分号结尾的,但C语言标准规定声明不是语句。

【例 3-2】 正整数 m 是一个 3 位数,按逆序输出 m 的 3 位数,如程序清单 3-5 所示。

程序清单 3-5

```
1   #include <stdio.h>
2   int main(){
3        int num3, d0, d1, d2;
4        printf("请输入一个三位数: ");
5        scanf("%d", &num3);
6        d0 = num3 % 10;
7        d1 = (num3/10) % 10;
8        d2 = num3/100;
9        printf("个位数: %d\n", d0);
10       printf("十位数: %d\n", d1);
11       printf("百位数: %d\n", d2);
12       return 0;
13  }
```

程序与用户的交互示例如下。

```
357↙
个位数: 7
十位数: 5
百位数: 3
```

程序中变量 d0、d1、d2 分别用于存放整数 num3 的个位数字、十位数字和百位数字。

【例 3-3】 读取两个整数存入变量 a、b,然后交换变量的值,如程序清单 3-6 所示。

程序清单 3-6

```
1   #include <stdio.h>
2   int main( ){
```