第3章

运算符与表达式

程序中的逻辑需要通过对数据进行运算来实现，运算的方式与数学运算一样，也是使用运算符。运算符作用于操作数，指明了需要进行的运算和操作，操作数包括常量、变量或者表达式。运算符是用于对两个或者两个以上操作数进行运算的符号，主要包括算术运算符、赋值运算符、关系运算符、逻辑运算符等，运算符和操作数组成了表达式。

3.1 简单赋值运算符及逗号运算符

运算符是组成表达式的基本元素。在介绍运算符和表达式之前先介绍两个最基本的运算符，即简单赋值运算符和逗号运算符。

3.1.1 简单赋值运算符及表达式

简单赋值运算符用于变量的赋值操作，赋值表达式语法为：

```
[类型] 变量 = 数值;
```

其中，变量是一个合法的标识符，用于标识内存中的一个存储单位，其存储空间由类型决定。在进行运算之前，该变量存储的数值通过赋值运算进行赋值。

在声明变量的同时赋值，称为变量初始化，例如：

```
int x = 10;
```

或者，在程序运行的过程中，改变已声明变量的值，例如：

```
x = 20;
```

3.1.2 逗号运算符及表达式

多个表达式用逗号连接起来，构成一个更大的表达式，其表达式语法为：

```
[类型] 变量 1 [= 值 1],变量 2[=值 2],…,变量 n[=值 n]
```

例 3.1 定义整型变量 x、y，并分别赋值。

```
int x = 10, y = 20;
```

上述语句利用逗号表达式声明两个整型变量 x、y，并分别赋初值 10、20。

3.2　算术运算符及表达式

算术运算符用于数值运算，其表达式主要用于双目运算，通过加(＋)、减(－)、乘(＊)、除(/)、取余(％)表达变量之间的计算方法，如表 3.1 所示。

表 3.1　算术运算符的意义、表达式示例及运算结果

符号	意　义	表达式示例及运行结果
＋	双目运算，计算两个操作数的和	int x＝1，y＝2； 执行：x＋y 结果：3
－	一般用于双目运算，计算两个操作数的差值；也可用于单目运算，用于变量取负值	例：int x＝1，y＝2； 执行：x－y 结果：－1 例：执行－x 结果：－1
＊	用于双目运算，计算两个操作数的乘积	例：int x＝1，y＝2； 执行：x ＊ y 结果：2
/	用于双目运算，计算两个操作数的商。如果参与运算的两个操作数都为整数，则计算结果为去掉小数部分得到整数	例：int x＝4，y＝2； 执行：x/y 结果：2 执行：y/x 结果：0
％	求两个操作数的取余操作，参与运算的数要求为整数	例：int x＝2，y＝5； 执行：x％y 结果：2

3.3　自增自减运算符

自增运算符(＋＋)、自减运算符(－－)用于整型变量的自增、自减运算，运算符可以位于整型变量的左侧或者右侧。

例 3.2　自增运算示例。

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

此时，x 的值为 11，等价为 x＝x＋1。因此，自增自减运算符可以视为一个赋值表达式。根据运算符在变量左侧或者右侧的不同，计算结果存在如表 3.2 所示两种情况：

表 3.2 自增自减示例

符号	表达式示例	运行结果
++	int x=10; (1) int y=x ++; (2) int y=++ x;	(1) x=11、y=10 (2) x=11、y=11
--	int x=10; (1) int y=x --; (2) int y=-- x;	(1) x=9、y=10 (2) x=9、y=9

可见在赋值语句中,自增自减运算符位于变量的左侧或者右侧时,其结果并不相同。由表 3.2 中示例可知表达式 x++,x--的值为 x,表达式++x,--x 的值分别为 x+1,x-1,变量 x 执行相应的加 1 或减 1 操作。

3.4 关系运算符及表达式

关系运算符用于数值之间的比较运算,包括大于(>)、小于(<)、等于(==)、大于或等于(>=)、小于或等于(<=)和不等于(!=)。比较运算的结果取值为"真"或者"假",C 语言用 0 表示假、非 0 表示真,如表 3.3 所示。

表 3.3 关系运算符

符号	意　义	表达式示例及运行结果
>	双目运算,比较两个数值之间是否存在大于关系	int x=1,y=2; 执行:x>y 结果:假(值为 0)
<	双目运算,比较两个数值之间是否存在小于关系	例:int x=1,y=2; 执行:x<y 结果:真(值为非 0)
==	双目运算,判断两个数是否相等	例:int x=1,y=2; 执行:x==y 结果:假(值为 0)
>=	双目运算,比较两个数值之间是否存在大于或等于关系	例:int x=1,y=2; 执行:x>=y 结果:假(值为 0)
<=	双目运算,比较两个数值之间是否存在小于或等于关系	例:int x=2,y=2; 执行:x<=y 结果:真(值为非 0)
!=	双目运算,比较两个数值之间是否存在不相等的关系	例:int x=2,y=2; 执行:x !=y 结果:假(值为 0)

例 3.3 判断一个数是否为偶数。

```
int x = 10;
int result = (x % 2 == 0);
//通过 x%2 的结果是否为 0 可以判断 x 是否为偶数,如果等于 0 则为偶数,否则为结果为非 0,
//表示奇数
```

3.5　逻辑运算符及表达式


逻辑运算符及其表达式

逻辑运算符用于逻辑运算，其表达式由表 3.4 逻辑运算符与(&&)、或(||)、非(!)组成，如表 3.4 所示。

表 3.4　逻辑运算符

符号	意　义	表达式示例及运行结果
&&	双目运算，用于计算两个操作数逻辑与的结果。当两个操作数都为真时(非 0)，结果为真(非 0)，否则为假(0)	执行： 1&&1、1&&0、0&&1、0&&0 结果分别为： 1、0、0、0
||	双目运算，用于计算两个操作数逻辑或的结果。当两个操作数中有一个操作数为真时(非 0)，结果为真(非 0)，否则为假(0)	执行： 1||1、1||0、0||1、0||0 结果分别为：1、1、1、0
!	单目运算，将操作数的逻辑值取反	执行：!0、!1 结果分别为：1、0

例 3.4　判断给定年份是否为闰年。

判定年份 year 是否为闰年需要满足如下两个条件中的一个：

(1) 能被 4 整除但不能被 100 整除；

(2) 能被 400 整除。

条件(1)中包含两个关系，且两个关系之间是逻辑与的关系；条件(2)中包含一个关系。条件(1)、(2)之间是逻辑或的关系。

因此，判断年份 year 是否为闰年可按如下代码列出表达式：

```
int year = 2000;
int result = ((year %4 == 0 && year %100 != 0) || (year %400 == 0));
```

3.6　位操作运算符及表达式


位操作运算符及其表达式

位运算符能够对参与运算的数按二进制位进行运算，包括位与(&)、位或(|)、位非(~)、位异或(^)、左移(<<)、右移(>>)，如表 3.5 所示。

表 3.5　位操作运算符

符号	意　义	表达式示例及运行结果
&	双目运算，将两个数的二进制对应位上的数值进行与操作，当两者都为 1 时，结果为 1，否则为 0	int x=10，y=6； 执行：x & y 1 0 1 0 & 0 1 1 0 结果为：2(二进制表示 0010)

续表

符号	意　义	表达式示例及运行结果
|	双目运算,将两个数的二进制对应位上的数值进行或操作,当两者中任意一位为 1 时,结果为 1,否则为 0	int x=10, y=6; 执行: x | y 1 0 1 0 | 0 1 1 0 结果为: 14(二进制表示 1110)
~	单目运算,将操作数的二进制按位取反,将 0 变为 1,1 变为 0	int x=10; 执行: ~x(即执行~1010) 结果为: 5(二进制表示 0101)
^	双目运算,将操作数的二进制按位异或,两者相同时为 0,否则为 1	int x=10, y=6; 执行: x ^ y 1 0 1 0 ^ 0 1 1 0 结果为: 12(二进制表示 1100)
<<	单目运算,将操作数的二进制按位左移,每左移 1 位,用 0 填充最低位	int x=1; 执行: x<<1(即将二进制 1 左移1 位) 结果为: 2(二进制表示 10)
>>	单目运算,将操作数的二进制按位右移,每右移 1 位,去掉最低位	int x=10; 执行: x>>1(即将二进制 1010 右移 1 位) 结果为: 5(二进制表示 101)

例 3.5　给定整数 x,测试 x 二进制编码的十位上数字。

```
int x = 10;
int y = x >> 1 & 1;              //二进制右移 1 位,将十位变为个位,并与 1 按位与,
                                 //结果赋值给变量 y
```

3.7　复合赋值运算符及表达式

复合赋值运算符用于变量赋值,在简单赋值运算符的基础上结合算术运算符(+=、-=、*=、/=、%=)和位运算赋值(&=、|=、^=、>>=、<<=)形成符合的赋值表达式,如表 3.6 所示。

表 3.6　复合赋值运算符

符号	意　义	表达式示例及运行结果
=	将赋值符号右侧的值赋给左侧的变量	int x; x=10; 变量 x 中的值为 10

+=	将赋值符号左侧的变量加上右侧的值,然后再赋值给左侧的变量	int x=10;
		x +=2;
		即执行 x=x+2,结果为 12

续表

符号	意　义	表达式示例及运行结果
−=	将赋值符号左侧的变量减去右侧的值，然后再赋值给左侧的变量	int x=10; x −=2; 即执行 x=x − 2，结果为 8

*=	将赋值符号左侧的变量乘以右侧的值，然后再赋值给左侧的变量	int x=10;
		x *=2;
		即执行 x=x * 2，结果为 20

/=	将赋值符号左侧的变量除以右侧的值，然后再赋值给左侧的变量	int x=10;
		x /=2;
		即执行 x=x / 2，结果为 5

%=	将赋值符号左侧的变量对右侧的值取余，然后再赋值给左侧的变量	int x=10;
		x%=2;
		即执行 x=x%2，结果为 0

&=	将运算符左侧的变量与右侧的值进行按位与操作，执行结果赋值给左侧的变量	int x=10;
		x&=2;
		即执行 x=x&2，结果为 2(二进制位 1010&10=10)

|=	将运算符左侧的变量与右侧的值进行按位或操作，执行结果赋值给左侧的变量	int x=10;
		x |=2;
		即执行 x=x | 2，结果为 10(二进制位 1010 | 10=1010)

^=	将运算符左侧的变量与右侧的值进行按位异或操作，执行结果赋值给左侧的变量	int x=10;
		x^=2;
		即执行 x=x|2，结果为 8(二进制位 1010|10=1000)

>>=	将运算符左侧的变量右移右侧值指定的位数，执行结果赋值给左侧的变量	int x=10;
		x>>=2;
		即执行 x=x>>2，结果为 2(二进制位 1010>>2=10)

<<=	将运算符左侧的变量左移右侧值指定的位数，执行结果赋值给左侧的变量	int x=10;
		x<<=2;
		即执行 x=x<<2，结果为 40(二进制位 1010<<2=101000)

例 3.6　定义变量并执行复合赋值运算。

```
int a=10,b=12,c=7;          //定义并初始化 3 个整型变量 a、b、c
double d=6.1;               //定义并初始化 double 变量 d
a+=3;                       //执行 a=a+3
b-=a+1;                     //执行 b = b - (a + 1)
c*=a-2;                     //执行 c = c * (a - 2)
```

3.8　条件运算符及表达式

条件运算符是 C 语言中唯一的三目运算符，条件表达式语法为：

```
表达式 1? 表达式 2: 表达式 3
```

条件表达式的含义是根据表达式 1 的结果计算表达式 2 或者表达式 3,当表达式 1 的结果为真(非 0)时,计算表达式 2 的值,否则计算表达式 3 的值。

例 3.7 求整数 x、y 中较大的数。

```
int a = 10, b = 20;
int c = a > b ? a : b;
```

当 a>b 成立(为真)时,返回 a,否则返回 b,因此,上述语句计算的结果为 a、b 中较大的数。

3.9 其他运算符

(1) 指针运算符,用于取内容(*)和取地址(&)两种运算。

(2) 求字节数运算符,用于计算数据类型所占的字节数(sizeof)。

(3) 特殊运算符,有圆括号()、下标[]、成员(→、.)等几种。

3.10 类型转换

参与运算的数据类型往往不一致,C 语言编译器提供了数据类型之间进行类型转换的机制。根据实际需要,类型转换分为自动类型转换和强制类型转换。

3.10.1 自动类型转换

自动类型转换指的是编译器在编译时自动进行的类型转换。

1. 赋值时的类型转换

赋值时变量类型与值类型不一致时会发生自动类型转换,例如:

```
double a = 10;
```

赋值符号右侧的数据为 10,其类型为整型,而变量 a 是一个浮点型数据,两者的类型不一致,因此,在赋值时整数 10 被转换为 double 类型的数据。这种转换是将低精度数据自动转换为高精度数据,不存在数据丢失。

同时,C 语言也支持高精度数据向低精度数据的自动转换,例如:

```
int b = 3.5;
```

该赋值语句将高精度的浮点型数据 3.5 放入低精度的整型变量,转换后的值为 3,存在数据丢失。

2. 整型提升

整型提升是指在做整型运算时,把短整型(short)或字符整型(char)提升为默认整型(int)再进行运算的一种机制。

整型提升的原因是运算在运算器内执行,操作数的长度是 int 的字节长度。因此,在计算时,长度小于 int 的整型数据(如 short、char 等)都必须转换为 int 后才能由运算器执行

运算。

整型提升是按照变量的数据类型的符号位类提升的，提升的时候高位补的是符号位，例如：

```
char ch = 10;
int c = ch;          //整型提升
```

在提升的过程中，字符 ch 的二进制编码是 00001010，所以在整型提升时高位补符号位 0，提升后 n 的二进制编码是：00000000 00000000 00000000 00001010。

对于存储负数的情况，其在内存中的二进制编码为补码，因此需要依据补码来填补高位，例如：

```
char ch = - 10;
int c = ch;
```

－10 的补码是 11110110，整型提升时高位补符号位 1，即 n 中存储的数为：

11111111 11111111 11111111 11110110

3. 算数转换

许多操作符的操作数不止一个，当不同类型的操作数在进行运算时，必须进行类型转换，即算数转换。

算数转换一般都是向着精度更高、长度更长的类型转换，如图 3.1 所示。

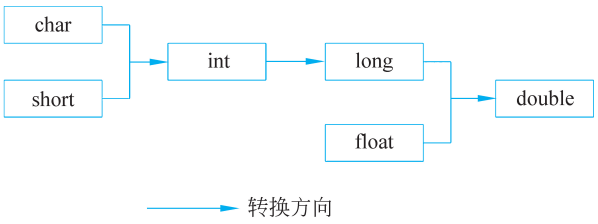

图 3.1　算数转换

3.10.2　强制类型转换

强制类型转换的语法格式为：

```
[数据类型] 变量 = (数据类型)表达式
```

例如：

```
int a = (int)3.5;
```

3.10.3　类型转换的规则

自动类型转换与强制类型转换都遵循如下的规则。

(1) 浮点型和整型之间的转换：将浮点型数据转换成整型数据时，会截断小数部分，只保留整数部分。

(2) 浮点型和浮点型之间的转换：低精度向高精度转换时数值不会发生变换；高精度向低精度转换时会将编译器近似值作为转换结果。

3.11 运算符的优先级与结合性

在程序设计中或程序阅读时需要根据运算符的优先级与结合性来理解程序，因此，程序员有必要对运算符的优先级和结合性进行全面的了解。

表 3.7 列出了所有运算符的优先级与结合性。

表 3.7 运算符的优先级与结合性

优先级	运算符	含　义	运算类型	结合性
1	() [] -> .	圆括号 数组下标运算符 用指针访问结构体成员运算符 结构体成员运算符	单目	自左向右
2	! ~ ++、-- (类型) +、- * & sizeof	逻辑非运算符 按位取反运算符 自增、自减运算法 强制类型转换 正、负号运算符 指针运算符 地址运算符 求变量长度运算符	单目	自右向左
3	*、/、%	乘、除、取余运算符	双目	自左向右
4	+、-	加、减运算	双目	自左向右
5	<<、>>	左移、右移运算符	双目	自左向右
6	<、<=、>、>=	小于、小于或等于、大于、大于或等于	关系	自左向右
7	==、!=	等于、不等于	关系	自左向右
8	&	按位与运算符	位运算	自左向右
9	^	按位异或运算符	位运算	自左向右
10	|	按位或运算符	位运算	自左向右
11	&&	逻辑与运算符	逻辑	自左向右
12	||	逻辑或运算符	逻辑	自左向右
13	?:	条件运算符	三目	自右向左
14	=、+=、-=、*=、/=、%=	赋值运算符	双目	自右向左
15	,	逗号运算符	顺序	自左向右

习　　题

(1) 定义整型变量 x、y、z，并分别赋值 10、20、20。

(2) 掌握自增运算符，并指出 i++、++i 的区别。

(3) 写出表达式判断能够被 5 整除的偶数。

(4) 根据自己计算机的整数表示位数，写出表达式测试整数的最高位。

(5) 写出表达式求整数 x、y、z 中最大的数。

(6) 类型提升的原因是什么？整数、负数在提升时的方法分别是什么？

(7) 掌握运算符的结合性和优先级。