第3章

运算符和表达式

无论如何，自然科学现在已发展得再也不能回避辩证综合了。

—— 恩格斯

表达式是最简单的程序。如果说程序是用程序语言实现的算法，那么表达式就是利用语言中的运算符构造的具有确定值的算法。C++语言表达式的形式多样，功能强大。

3.1 表达式

常量、变量是表达式 —— 最简单的表达式；常量和变量作为运算符对象与运算符的结合是表达式 —— 运算符表达式；每一个表达式都有一个确定的值，这个值作为运算符对象又可以和运算符构成表达式 —— 复合表达式。例如：

```
5                   //常量表达式，其值是5
a                   //变量表达式，其值是存储的值
a=5                 //赋值运算符表达式，其值是赋值后的a的值5
7*a                 //算术运算符表达式，其值是两者的乘积35
b=7*a               //复合表达式，相当于b=(7*a)
c=a=b               //复合表达式，相当于c=(a=b)
(a=b)=c             //复合表达式
```

每一个表达式都可以作为一条语句，而每条语句未必都是一个表达式，例如有空语句，但没有空表达式。

程序3.1　检验表达式的值。

```
#include<iostream.h>
int main()
{
     int a,b,c;
     cout<<(a=5)<<endl;          //a=5; cout<<a<<endl;
     cout<<(b=7*5)<<endl;        //b=7*5; cout<<b<<end;
     cout<<(c=a=b)<<endl;        //c=(a=b); cout<<c<<endl;
     return(0);
}
```

<运行结果>

```
5
35
35
```

3.2 关系操作符

关系运算，也称比较运算，是两个表达式的比较，其结果是布尔型值。如果关系成立，结果等于1，表示“真”，否则，结果等于0，表示“假”（见表3-1）。例如：

```
a>3
```

其中，大于号“>”是关系操作符，a和3是操作数。如果a的值为5，则a>3的结果为1；如果a的值为2，则a>3的结果为0。

表3-1 关系操作符

符号	功能	符号	功能
<	小于	<=	小于或等于
>	大于	>=	大于或等于
==	等于	!=	不等于

程序3.2 检验关系表达式。

```
#include<iostream.h>
int main()
{
     int a, b,c,d;
     cin>>a>>b>>c;
     cout<<"a="<<a<<endl;
     cout<<"b="<<b<<endl;
     cout<<"c="<<c<<endl;
     cout<<(c<(a+b))<<endl;
     cout<<((d=c)>=(a+b))<<endl;
     return(0);
}
```

<运行结果>

```
4 7 9                    //从键盘输入
a=4
b=7
c=9
1
0
```

3.3 逻辑操作符

逻辑运算中的表达式取布尔值，表达式的值非0为真，0为假。逻辑操作符如表3-2所示。

逻辑运算中的操作数，非0值表示真，0表示假。表3-3是逻辑运算的真值表。

表3-2 逻辑操作符

符 号	功 能
&&	逻辑与
||	逻辑或
!	逻辑非

表3-3 逻辑运算真值表

a	b	a&&b	a||b	!a
真	真	真	真	假
真	假	假	真	假

（续）

a	b	a&&b	a||b	!a
假	真	假	真	真
假	假	假	假	真

程序3.3 检验逻辑表达式。

```
#include<iostream.h>
int main()
{
      int a,b,x,y;
      a=13;
      b=15;
      x=158/a;
      y=642%b;
      cout<<a<<'\t'<<b<<'\t'<<(a<b)<<endl;
      cout<<x<<'\t'<<y<<'\t'<<(x<y)<<endl;
      cout<<((a<b)&&(x<y))<<endl;
      cout<<((a==b)||(x==y))<<endl;
      cout<<((!a)||(a!=b))<<endl;
      return(0);
}
```

<运行结果>

```
13    15    1
12    12    0
0
1
1
```

注意 逻辑与（&&）和逻辑或（||）的运算均为从左至右进行。在逻辑与和逻辑或的运算中，如果左元的结果能代表整个逻辑表达式的结果，那么右元就不再计算，这种情况称为短路。具体地说，在逻辑与的运算中，如果左元已为假，则结果一定为假，这时右元就无须计算。在逻辑或的运算中，如果左元已为真，则结果一定为真，这时的右元也不用计算。表3-4给出了短路示例。

表3-4 逻辑与、逻辑或的短路示例

假设：int a=5,b=6,c=7;

逻辑表达式	结 果	说 明
(a>b)&&(c=c*2)	0，c的值仍为7	短路，因为左元a>b的结果为0，故右元c=c*2不执行
(a<b)&&(c=c*2)	1，c的值为14	不短路，因为左元a<b的结果为1，故右元c=c*2需执行
(a<b)&&(c=c-7)	0，c的值为0	不短路，因为左元a<b的结果为1，故右元c=c-7需执行
(a<b) ||(c=c*5)	1，c的值仍为7	短路，因为左元a<b的结果为1，故右元c=c*5不执行
(a>b)||(c=c*5)	1，c的值为35	不短路，因为左元a>b的结果为0，故右元c=c*5需执行
(a>b)||(c=c-7)	0，c的值为0	不短路，因为左元a>b的结果为0，故右元c=c-7需执行

3.4 自增自减操作符

自增自减是C++语言特有的运算，如表3-5所示。

表3-5 自增自减运算

自增自减运算	解 释	自增自减运算	解 释
i++; ++i;	i=i+1;	x=--j;	--j; x=j;
i--; --i;	i=i-1;	k=(i++)+(++j);	++j; k=i+j; i++;
x=i++;	x=i; i=i++;	k=(i--)+(--j);	--j; k=i+j; i--;
x=++j;	++j; x=j;	++(++i);	++i; ++i;
x=i--;	x=i; i--;		

程序3.4 检验自增自减运算。

```
#include<iostream.h>
int main()
{
     int i=5,j=6,k;
     i++;  ++j;                          //相当于i=i+1;j=j+1;
     cout<<"i="<<i<<'\t';
     cout<<"j="<<j<<endl;
     i--;  --j;                          //相当于i=i-1;j=j-1;
     cout<<"i="<<i<<'\t';
     cout<<"j="<<j<<endl;
     k=(i++)+(++j);                      //相当于++j;k=i+j;i++;
     cout<<"i="<<i<<'\t';
     cout<<"j="<<j<<endl;
     k=(i--)+(--j);                      //相当于--j;k=i+j;i--;
     cout<<"i="<<i<<'\t';
     cout<<"j="<<j<<endl;
     return(0);
}
```

<运行结果>

```
i=6     j=7
i=5     j=6
i=6     j=7
i=5     j=6
```

3.5 赋值和复合赋值操作符

复合赋值操作符是C++语言特有的一种赋值运算，它是一种普通赋值表达式的缩写形式。复合赋值操作符如表3-6所示。

表3-6 复合赋值操作符

操作符	功 能	举 例
+=	加法赋值	x+=y; 相当于x=x+y;
-=	减法赋值	x-=y; 相当于x=x-y;
*=	乘法赋值	x*=y; 相当于x=x*y;
/=	除法赋值	x/=y; 相当于x=x/y;
%=	模运算赋值	x%=y+z; 相当于x=x%(y+z);

程序3.5 检验复合赋值运算。

```
#include<iostream.h>
int main()
{
     int a,b,c;
     a=b=c=7;                                  //c=7;b=c;a=b
     c+=a++;                                   //c=c+a; a++;
     cout<<"a="<<a<<'\t';
     cout<<"c="<<c<<endl;
     c%=++a;                                   //++a; c=c%a;
     cout<<"a="<<a<<'\t';
     cout<<"c="<<c<<endl;
     c*=a--;                                   //c=c*a; a--;
     cout<<"a="<<a<<'\t';
     cout<<"c="<<c<<endl;
     return(0);
}
```

<运行结果>

```
a=8     c=14
a=9     c=5
a=8     c=45
```

3.6 条件操作符

条件表达式的格式如下：

表达式1？表达式2：表达式3

它的意义是：若表达式1为真，则条件运算表达式的值等于表达式2的值，否则等于表达式3的值。例如：

```
c=(a>b?a:b);
```

若a大于b，则c的值是a的值，否则，c的值是b的值。

程序3.6 检验条件表达式。

```
#include<iostream.h>
int main()
{
     int a=1,b=7,c,d;
     c=a>b?a:b;
     d=a<b?a:b;
     cout<<"a="<<a<<'\t';
     cout<<"b="<<b<<endl;
     cout<<"c="<<c<<'\t';
     cout<<"d="<<d<<endl;
     return(0);
}
```

<运行结果>

```
a=1     b=7
c=7     d=1
```

3.7 逗号操作符

多个表达式以逗号分隔，每个表达式从左至右分别计算，最后一个表达式的值是整个表达式的值，如下所示：

表达式**1**，表达式**2**，…，表达式**n**

例如：

```
int a,b,c;
c=(a=7,b=5,a+b);                    //将逗号表达式的值赋给c
```

逗号表达式的执行过程如下：

```
a=7;
b=5;
c=a+b;
```

3.8 复合表达式

含有多个操作符的表达式称为复合表达式。复合表达式的计算顺序为：先按操作符的优先级从高到低计算，如果优先级相同，按操作符的结合性处理。

操作符优先级按以下顺序从高到低（操作符优先级详见附录B）：算术操作符（+，-，*，/，%），关系操作符（<，<=，>，>=），关系操作符（==，!=），赋值操作符（=）。例如：

```
c<a+b                    //c<(a+b)
a>b!=c                   //(a>b)!=c
a==b<c                   //a==(b<c)
```

建议用括号明确计算顺序。例如：

```
(d=c)>=(a+b)             //用赋值表达式d=c的值和算术表达式a+b的值进行比较。
```

3.9 内部类型转换

在表达式中，不同类型的数据进行混合运算是很常见的，这时不同类型的运算对象先要转换为同一种类型再进行计算，这个类型称为目标（转换）类型。转换应该遵循的原则是：把小类型提升为大类型，以防止数据被截断（truncation），损失精度。C++语言根据这个原则，制定了一组内置类型的标准转换：定点型转换为浮点型，短存储型转换为长存储型，有符号数据转换为无符号数据，整型转换为浮点型。

3.9.1 赋值兼容性

赋值表达式中的目标类型是左元的类型，因此左元类型不能“小于”右元类型，称为赋值兼容性。例如：

```
int a;
double f;
f=100;                   //正确
a=3.14;                  //错误！系统要警告：possible loss of data（数据可能丢失）
```

赋值操作是程序语言特有的最基本的操作。赋值主要分为转换赋值和复制赋值。转换赋值是右元和左元类型不一致，需要转换的赋值；复制赋值是右元和左元都是类型相同的变量。表

3-7做了分类举例。

表3-7 赋值分类举例

常量初始化	转换初始化	复制初始化	常量赋值	转换赋值	复制赋值
int a=5;	long n=5;	long m=n;	a=5;	n=5;	m=n;
double x=3.1415;	double y=3.1415f;	double z=x;	x=3.1415;	y=3.1415f;	z=x;

3.9.2 表达式计算中的类型转换过程

在赋值表达式的计算中，系统从右元所标识的空间取出数据，转换为左元类型，存储到左元所标识的空间。在这个过程中，右元所标识的空间类型是不会改变的。例如：

```
double b;
b=3.1415f;
```

从单浮点型字面值常量3.1415f的空间中取出3.1415，转换为双浮点型，存储到b。

在其他表达式的计算中，系统从操作数空间中取出数据，转换为目标类型进行计算，将计算结果存储到一个临时匿名的目标类型变量中，作为表达式的值。例如：

```
3.1315*2L
```

从字面量3.1315和2L的空间中取值，按照双浮点型计算，将计算结果赋值给一个临时匿名的双浮点型变量中，不妨将这个变量设为_temp。于是输出语句

```
cout<<3.1315*2L;
```

可以理解为

```
cout<<_temp;
```

复合表达式的计算实际上是赋值操作的反复应用。以表达式3.5f+3.1315*2L为例，按照先乘除后加减、先左后右、先括号内后括号外的规则，先计算3.1315*2L：从字面量3.1315和2L中取值，按照双浮点型计算，将结果赋值给临时双浮点型变量，假设为_temp1。再计算3.5f+_temp1：从自变量3.5f和变量_temp1中取值，按双浮点型计算，将结果赋值给临时双浮点型变量，假设为_temp2。

3.9.3 强制类型转换

如果不想按照标准进行自动转换，可以采用显式转换（explicit type conversion），也称强制类型转换（cast），转换为程序员自己需要的类型。强制类型转换的格式为：

（类型标识符）转换对象

例如：

```
int a=(int)3.14;          //a=3
(float)3/4                //3转换为单浮点数，然后做实数除法，结果为0.75
```

习题

1. 经过下面的运算后，x的结果是_____。

```
float x=2.5, y=5.5;
```

```
x=x+(4/2*(int)y/2)%4;
```

A. 3.5　　B. 2.5　　C. 1.5　　D. 0.5

2. 复合赋值语句“a+=6;”的等价式是_____。

A. a=a+6;　　B. a=(a++)+6;　　C. a=6+6　　D. a=++a+6;

3. 复合赋值语句“a*=b+7;”的等价式是_____。

A. a=a*b+7;　　B. a=a*(b+7);　　C. a=a+b*7　　D. a=a*b+b*7;

4. 复合赋值语句“a+=b++;”的等价式是_____。

A. b++; a=a+b;　　B. a=a+(++b);　　C. a=a+b; b++;　　D. a=b++;a++;

5. 复合赋值语句“a+=++b;”的等价式是_____。

A. b++; a=a+b;　　B. a=a+(b++);　　C. a=(a++)+(b++);　　D. a=b++;a++;