# 第 3 章　运算符和表达式

先看一道小学数学中经常出现的四则运算：(12 * (34+56)– 9)/ 9。按照四则运算法则，本例首先要计算括号中的 34 + 56，假设得到的结果为 A，得到 A 后再计算 12*A 的值，得到结果 B，再次计算 B-9 的值，得到 C，然后再计算 C/9 的值，得到最终的结果 D。

对于 C++编程来说，这就是比较复杂的算术运算符和算术表达式的例子。C++提供了很多运算符，既继承了 C 语言的所有运算符，又添加了 new、delete 运算符，还支持运算符的重载。由运算符组成的表达式和语句构成了 C++程序的基础。

本章主要内容如下：

- 运算符和表达式的基础知识；
- 常用运算符和表达式的使用；
- 运算符的结合性和优先级。

## 3.1　运算符和表达式

本章开始的例子对于 C++来说，属于比较复杂的算术表达式。此例出现的+、-、*、/的符号在 C++中称为运算符，运算符提供的是进行何种运算，而 12、34 等称为操作数，它的作用是提供运算符的操作对象。

如果某个运算符需要两个操作数，则称此运算符为双目运算符，如上例中出现的 4 个算术运算符均属于双目运算符。如果只需要一个操作数，则称该运算符为单目运算符，如在后面将要介绍的++运算符。在 C++中，运算符最多支持 3 个操作数，这类被称为三目运算符。运算符和操作数组成的式子称为表达式。在运算符左边的操作数称为运算符的左值，在运算符右边的操作数称为运算符的右值。不同的运算符对于左值和右值有着不同的要求。有的只需要右值，如取值运算符等，有的却需要左值和右值，如所有的算术运算符。

在 C++中还存在一类特殊的运算符，它是由两个运算符组合而成，这类运算符被称为复合运算符，由复合运算符构成的表达式称为复合运算表达式。该表达式在计算的过程中有两个运算符的效果，相当于两个普通表达式的集合。如复合表达式 a +=5，表示 a = a+5。这样表示虽然对于初学者来说理解上有点困难，但是这样的表达式在编译处理的时候却非常有利，能提高编译效率并产生质量较高的目标代码。

在 C++中总共存在 45 个运算符，经常使用的运算符和对应的示例表达式如表 3-1 所示。本章仅介绍常用的运算符，而下标运算符则在数组一章中详细介绍，取址运算符、取值运算符在介绍指针的章节中将详细介绍，而取成员变量运算符将会在介绍结构体时详细介绍。

表 3-1　常用运算符和对应示例表达式

| 运算符类型 | | 运算符 | 对应运算符表达式举例 |
|---|---|---|---|
| 算术运算符 | 加法运算符 | + | a + b |
| | 减法运算符 | - | a - b |
| | 乘法运算符 | * | a * b |
| | 除法运算符 | / | a / b |
| | 取模运算符 | % | a % b |
| | 自增运算符 | ++ | a++或++a |
| | 自减运算符 | -- | a--或--a |
| 关系运算符 | 等于 | == | a == b |
| | 大于 | > | a > b |
| | 大于等于 | >= | a >= b |
| | 小于 | < | a < b |
| | 小于等于 | <= | a <= b |
| | 不等于 | != | a != 0 |
| 逻辑运算符 | 逻辑或运算符 | \|\| | a > 0 \|\| a == 0 |
| | 逻辑与运算符 | && | a>0 && a< 10 |
| | 逻辑非运算符 | ! | !a |
| 赋值运算符 | | = | a = 2 |
| 逗号运算符 | | , | a = (a+2, a++) |
| 条件运算符 | | ？： | a>b?a:b |
| 取指运算符 | | & | &a |
| 取值运算符 | | * | *a |
| 取成员变量运算符 | | -> | a->b |
| | | . | a.b |
| 取字节长度运算符 | | sizeof | sizeof（int） |
| 位运算符 | 按位与运算符 | & | a & b |
| | 按位或运算符 | \| | a \|\| b |
| | 按位非运算符 | ~ | ~a |
| | 按位异或运算符 | ^ | a^b |
| 常用复合运算符 | | += | a += 10 (a = a + 10) |
| | | -= | a -= 10 (a = a - 10) |
| | | *= | a *= 10 (a = a * 10) |
| | | /= | a /= 10 (a = a / 10) |

## 3.2　赋值运算符和赋值表达式

在 C++中，赋值运算符只有一个，即数学中常用的等号“=”，但是这个等号已经不再表示两个数值具有相等的关系，而是表示将等号的右值赋给左值。这是一个数据传递的过程，而不表示二者之间具有数值相同的关系。

### 3.2.1　赋值运算的基本使用

赋值运算的左值只能是个变量而不能是常量，而右值可以是常数、变量、表达式等多

种形式。如下列赋值表达式：

```
a = 5;
a = b;
a = a + 5;
```

第 1 个表达式表示的是将常量 5 赋值给变量 a，第 2 个表达式表示的是将变量 b 的内容传递给变量 a，即 a 的最终结果为变量 b 的值。而第 3 个表达式的作用是将 a 与 5 求和后再次赋值给 a。

赋值运算符的左值不能为常量，如表达式 5 = a 表示的含义是将变量 a 的值赋值给常量 5。而第 2 章关于变量和常量的介绍中已经明确，常量之所以被称为常量，就是因为其数值不能被随意改变，必须保持不变，只有变量的值才能发生改变。如示例程序 3-1.cpp 所示。

```
//-------------------------示例程序 3-1.cpp-------------------------
#include <stdio.h>
int main(void)
{
    int a, b = 2;                    //定义需要的变量，并将常量 2 赋值给变量 b
    a = 5;
    printf("first a = %d\n", a);     //将常量 5 赋值给变量 a，此时 a=5
    a = b;
    printf("second a = %d\n", a);    //将变量 b 的值赋值给 a，此时 a=b=2
    a = a + 5;                       //将表达式 a+5 的值赋值给 a，此时 a=2+5 = 7
    printf("last a = %d\n", a);

    return 0;
}
```

程序编译后运行结果如下：

```
first a = 5
second a = 2
last a = 7
```

在此示例程序中，首先将常量 5 赋值给变量 a，然后重新给变量 a 赋值，其值为变量 b 的值，即 2。最后一次赋值是将表达式 a+5 的值赋值给变量 a，最终得到的结果为 7。由此可见，变量的值可以多次被赋值，因此在程序编写过程中一定要明确变量的值具体是多少。

**注意**：在程序的编写过程中应尽量减少使用连等号，如 a=b=c。此情况在变量 c 或变量 b 未初始化时，变量 a 的值会变成随机数。因此为了减少错误的发生，建议不要使用连等的形式。

## 3.2.2　赋值运算过程中的类型转换

前面的赋值运算都是赋值运算符的左值和右值均为相同数据类型的情况。如果左值和右值的数据类型不一致，就会发生数据类型的隐式转换，由于不同类型变量表示的精度不同，因此会对程序的最终结果造成影响。如示例程序 3-2.cpp 所示。

```
//-------------------------示例程序 3-2.cpp-------------------------
#include <stdio.h>
int main(void)
```

```
{
    float f = 12.345;
    int num;

    num = f;                          //浮点型赋值给整型
    printf("num = %d, f = %f\n", num, f);
    num = 10;                         //给变量 num 赋值
    f = num;                          //整型赋值给浮点型
    printf("num = %d, f = %f\n", num, f);

    double d;
    f = 12.345;                       //给浮点数 f 赋值
    d = f;                            //浮点型赋值给双精度类型
    printf("d= %ld, f = %f\n", num, f);
    d = 12.345678901;                 //重新给 d 赋值
    f = d;                            //双精度类型赋值给浮点型
    printf("d = %ld, f = %f\n", num, f);
    return 0;
}
```

程序编译后运行结果如下：

```
num = 12, f = 12.345000
num = 10, f = 10.000000
d= 10, f = 12.345000
d = 10, f = 12.345679
```

通过程序的运行结果可以看出，当将浮点型的数据赋值给整型的数据时，编译器会自动将小数点后面的内容舍弃，只保留整数部分。如在示例程序中，将浮点数 12.345 赋值给整型数据 num 时，得到的结果为舍弃了小数部分的 12。将整型数据赋值给浮点数时，会在整数的末尾加上小数部分，一般是 6 个 0。如示例程序中将整型数据 10 赋值给浮点型变量 f 时，得到的结果为 10.000000。双精度数和整型数据之间的赋值和浮点型类似，当由精度高的部分变为精度较低的部分时，会舍弃部分精度来达到数据类型的要求。而当将精度较低的数据赋值给精度较高的数据时，会在数据的后面补 0，从而达到高精度数据类型的要求。

**注意：**在 C++中等号=表达式，如 a=b，表示的是将 b 的值赋值给 a，而不是表示 a 和 b 的值相等。C++表示相等的符号为==。

## 3.3　算术运算符和表达式

算术运算符包括加运算（运算符为“+”）、减运算（运算符为“-”）、乘运算（运算符为“*”）、除运算（运算符为“/”）和取模运算（运算符为“%”）。前四者在运算的过程中和普通的四则运算的计算过程是一致的，即运算顺序是从左至右，先计算乘或除，后计算加或减，如果有括号的话，则先要计算括号中的内容，在计算括号内的内容的过程中也要遵循从左至右，先计算乘或除，后计算加或减的基本原则。关于算术运算如示例程序 3-3.cpp 所示。

```
//------------------------示例程序 3-3.cpp-------------------------
#include <stdio.h>
int main(void)
{
    int a = 12, b = 34, c = 56;
    int result;                              //定义需要的变量

    result = (a + b) * c;                    //先进行加运算然后进行乘运算
    printf(" (12 + 34) * 56 =  %d\n", result);

    result = a - c / b;                      //先进行除运算，后进行减运算
    printf("12 - 56/ 34 =  %d\n", result);
    return 0;
}
```

程序编译后的运行结果如下：

```
(12 + 34) * 56 =  2576
12 - 56/ 34  =  11
```

在此示例程序中，第 1 次运算由于括号的作用，先计算 12 + 34，得到结果 A，然后再进行 A 和 56 的乘法运算，最终得到结果 2576。在第 2 次运算中，首先计算 56/34，得到结果 1，然后再计算 12 – 1，最终得到结果 11。

在进行除法运算时，右值和在数学中的算术运算一样不能为 0。如果右值为 0，则会在编译的过程中出现错误信息提示。如示例程序 3-4.cpp 所示。

```
//------------------------示例程序 3-4.cpp-------------------------
#include <stdio.h>
int main(void)
{
    int a = 12, s;
    int zero = 0;;                  //定义需要的变量

    s = a / zero;                   //进行操作运算符右值为 0 的除操作
    printf("12 / 0 = %d\n", s);
    return 0;
}
```

程序在编译的过程中，不会出现错误，但是在运行的过程中会出现如下错误：

```
"浮点数例外"
```

出现此种错误的原因就是除运算操作符的右值为 0，只要将变量 zero 的值变成非零值即可消除此种错误。

**说明**：在 Windows 和 Linux 系统中，对于除数为 0 的错误提示信息可能不相同，但是其根源是相同的。

取模运算又称为求余运算，其左值和右值只能是整数，而不能为其他类型的数值。在运算过程中，先进行除法运算，将得到的余数作为整个表达式的值，而获取的商则直接舍去。因此除法运算又可称为取商运算，而取模运算又可称为取余数运算。

正常的取余操作和除法操作如示例程序 3-5.cpp 所示。

```
//------------------------示例程序 3-5.cpp-------------------------
```

```
#include <stdio.h>
int main(void)
{
    int a, b;
    int f = 3;
    int mod1, mod2;
    int shang1, shang2;                          //定义需要的变量

    a = 12; b = 34;                              //给变量 a、b 赋值
    printf("a%%f = %d, f %%a = %d\n", a%f, f %a);
    mod1 = a % b;                                //取 a/b 的余数
    shang1 = a / b;                              //取 a/b 的商
    mod2 = b % a;                                //取 b/a 的余数
    shang2 = b / a;                              //取 b/a 的商
    printf("a %% b = %d, a / b = %d\n", mod1, shang1);     //使用%%输出
    printf("b %% a = %d, b / a = %d \n", mod2, shang2);    //输出计算结果

    return 0;
}
```

程序编译后运行结果如下：

```
a%f = 0, f %a = 3
a / b = 12, a % b = 0
b / a = 10,  b % a = 2
```

在示例程序 3-5.cpp 中出现了两个%，作用是输出一个%，如果只有一个%的话，会出现如下警告信息：

```
error: invalid operands of types 'int' and 'float' to binary 'operator%'
error: invalid operands of types 'int' and 'float' to binary 'operator%'
```

根据程序的运行结果得知，除法运算的结果是按照整除时进行计算获得的商，如果不整除的话就会自动舍弃余数而只保留得到的商。取模运算只保留得到的余数，而舍弃商。如程序中出现的除法运算 a / b 的结果为 0，而 b / a 的结果为 2。对于取余运算 a % b，因为 12 除以 34 最终的结果为商为 0 而余数为 12，因此该表达式的最终结果为 12。而运算 b % a 的商为 2 余数为 10，所以最终结果为 10。

上面讲述的算术运算都是在相同数据类型之间进行的运算，而在实际的程序编写过程中，会出现不同数据类型之间的计算。解决此类问题的方法有两个：一个是由系统自动地进行类型转换；一个是强制类型转换。这部分内容会在数据类型转换的章节中详细介绍。

**说明：**

（1）进行除法运算时，如果取值运算符表示的操作作为除法运算的右值，一定要注意在运算符/和运算符*的之间添加一个空格，从而避免被当作注释的开头部分。

（2）相对于其他运算，除法运算和取模运算在运算过程中会耗费过多的系统资源，因此在编程过程中，应尽量减少除法运算和取模运算。

## 3.4　比较大小的关系运算符和关系表达式

关系运算符即对操作数之间的大小关系进行操作的运算符，和数学中的>、<等运算符

是一致的。包含关系运算符的表达式称为关系表达式。基本的关系运算符包括等运算符（==）、大于运算符（>）、小于运算符（<）、不等于运算符（!=）。在实际的运算中还有大于等于运算符（>=）、小于等于运算符（<=）这两种复合运算符。

关系运算符在操作时是将运算符的左值 a 减去右值 b。如果获得的结果大于 0，则称为 a 大于 b，记作 a>b。如果得到的结果小于 0，则称为 a 小于 b，记作 a <b。如果得到的结果等于 0，那么称为 a 等于 b，记作 a==b。在比较过程中，如果表达式成立，则得到的结果为真，可以记为 TRUE 或是 1，否则记作 FALSE 或是 0。关系运算符如示例程序 3-6.cpp 所示。

```
//-------------------------示例程序 3-6.cpp-------------------------
#include <stdio.h>
int main(void)
{
    int a = 12, b = 34,  c = 12;                //定义需要的变量

    printf(" (a > b) = %d\n", a > b);           //输出比较表达式的结果
    printf(" (a < b) = %d\n", a < b);
    printf(" (a == b) = %d\n", a == b);
    printf(" (a == c) = %d\n", a == c);
    return 0;
}
```

程序编译后的运行结果如下所示：

```
(a > b) = 0
(a < b) = 1
(a == b) = 0
(a == c) = 1
```

在此程序中，根据变量在定义过程中的赋值可知，a 大于 b，并且 a 等于 c。因此在执行 a>b 时，得到的结果为 0，而 a < b 的值为 1，a == b 的值为 0，a == c 的值为 1。

**注意：**关系运算中的==和赋值运算符=仅差一个=，但是二者所表达的意思却截然不同。a==b 表示 a 和 b 相等，是关系表达式，而 a=b 则表示将 b 的值赋值给 a，是赋值运算表达式。

## 3.5 “真真假假”的逻辑运算符和逻辑表达式

逻辑运算又称布尔运算，是一种判断表达式是否成立的运算。逻辑运算的结果只有 0 或 1 两种。如果表达式成立，则结果为 1，也称结果为逻辑真，如果运算的结果 0，则称为逻辑假。在实际中，常用非零正值表示真，而零表示假。

逻辑运算符包括逻辑与运算符（&&）、逻辑或运算符（||）和逻辑非运算符（!）。逻辑与运算符和逻辑或运算一样，都是双目运算符，需要两个表达式，如表达式 a>0 && a < 10 或 a>0 || a < 10。计算顺序为从左至右，先计算最左面的表达式的值，然后依次对后面的表达式进行计算，根据逻辑运算的真值表判断整个表达式的结果。有且只有两个表达式的值均为 1 时，整个逻辑与表达式的值才为 1。当其中一个表达式的值为 1 时，整个逻辑或

表达式的值就为 1。而逻辑非运算符为单目运算符，只需要一个表达式，得到的结果和给出的表达式的结果正好相反。

逻辑运算的求值方法如表 3-2 所示。

表 3-2　逻辑运算求值规律

| 运算类型 | 表达式 1 的值 | 表达式 2 的值 | 最终逻辑表达式的值 |
|---|---|---|---|
| 逻辑或运算 | 1 | 1 | 1 |
| | 1 | 0 | 0 |
| | 0 | 1 | 0 |
| | 0 | 0 | 0 |
| 逻辑与运算 | 1 | 1 | 1 |
| | 1 | 0 | 1 |
| | 0 | 1 | 1 |
| | 0 | 0 | 0 |
| 逻辑非运算 | 0 | 无第 2 表达式 | 1 |
| | 1 | 无第 2 表达式 | 0 |

由逻辑运算符构成的表达式称为逻辑表达式。在逻辑与的计算过程中，如果一个表达式的值为假，则不再进行第二个表达式的计算，而直接将表达式的值记作 0。而对于逻辑或运算来说，如果一个表达式的结果为 1 时，则不再对后面的表达式进行计算，直接将整个表达式的值记作 1，这种方式一般被称为短路运算。

逻辑运算经常和关系运算结合使用，用来判断在几个条件满足时程序要进行的操作。如示例程序 3-7.cpp 是一个猜数游戏的示例程序，程序的功能是判断给定的一个整数在什么范围之内，直至最终确定该整数的值。为了简单起见，我们只判断从 1～5 范围的数，而对其他范围内的数据不做具体分析。在程序中出现的 while 以及 if…else 语句是通过判断表达式是否成立来决定程序执行哪部分的内容，此部分内容会在后面的章节中详细介绍，在此不做赘述。

```
//-------------------------示例程序 3-7.cpp-------------------------
#include <stdio.h>
int main(void)
{
    int num;                                //定义需要的变量
   printf("input the value of num (1~5 and 0 is quit) :  ");
    while (scanf("%d", &num) == 1 && num != 0) //确保输入的数为整数并且不为 0
    {
       if (num < 0 || num > 5 )             //符合条件：为负数或是大于 5
       {
          if (num > 0 && num < 3)           //在 0～3 之间即 1 或 2
          {
             if (num != 1)                  //不为 1 则为 2
                printf("num is 2\n");
             else
                printf("num is 1\n");
          }
          else if (num > 3)                 //在 4 和 5 之间
          {
             if (num == 4)                  //不为 4 则为 5
                printf("num is 4\n");
             else
```

```
                printf("num is 5\n");
            }
            else                        //num 为 3
                printf("num is 3\n");
        }
        else
        {
            printf("num < 0 ");
        }
        printf("input the value of num:(1~5 and 0 is quit)");
    }
    printf("bye\n");
    return 0;
}
```

程序在编译后，运行结果如下：

```
input the value of num (1~5 and 0 is quit) :  1
num is 1
input the value of num (1~5 and 0 is quit) :  3
num is 3
input the value of num (1~5 and 0 is quit) :  6
num > 5
input the value of num (1~5 and 0 is quit) :  -1
num < 0
input the value of num (1~5 and 0 is quit) :  0
bye
```

在此程序中，首先判断输入的数据是否为整数并且该整数是否不为 0。只有这两个条件都满足，程序才对数字的数值进行判断，如果有一项不满足，则直接退出程序。然后程序判断输入的整数是否是负数或者是大于 5 的数，如果是在 1～5 之间，则继续判断。如果是负数或者是大于 5 的数，则直接退出。依次类推，直至判断出数据的具体数值为止。

**技巧**：程序中的 while 用来判断输入的数据是否是整数，并且判断输入的整数是否为 0，此种情形可以更好地控制输入的数据类型和变量的数值，在后续的章节中还会出现，请读者牢记。

## 3.6　特殊的逗号运算符和逗号表达式

逗号运算符又称为顺序求值运算符，其作用是将两个或两个以上的表达式连接起来，按照先后顺序进行求值，并将最后一个表达式的值作为逗号表达式的值。逗号表达式的一般形式为：

```
表达式 1，表达式 2，表达式 3,……,表达式 n;
```

该表达式的求解过程是先计算表达式 1，然后计算表达式 2，最终计算至表达式 n，表达式 n 的结果就是该逗号表达式的结果。

逗号表达式的优先级低于其他的所有运算表达式，而且逗号表达式也可以嵌套使用，即在逗号表达式中的其中某一个表达式也是逗号表达式。逗号表达式如示例程序 3-8.cpp 所示。

```
//-------------------------示例程序 3-8.cpp-------------------------
#include <stdio.h>
int main(void)
{
    int a = 10, b = 3;
    int s;                          //定义需要的变量

    s =( b= a +4, a++, a + b); //先计算 b 的值，然后计算 a++，最后计算 a+b
    printf("s = %d\n", s);
    s = (s++, s - b);               //先计算 s++，然后计算 s-b
    printf("s = %d\n", s);
    return 0;
}
```

程序编译以后的运行结果如下：

```
s = 25
s = 12
```

在此示例程序中的第 1 次计算过程中，先计算 b 的值，然后再计算 a++的值，最后计算 a+b，并将该值作为 s 的值。第 2 次的运算过程是先计算 s++，然后再计算 s-b。而变量 s 的最终值为 s-b 的值。

**注意：**函数的参数列表中出现的逗号不属于逗号表达式，而仅是分隔符，用来分隔不同的参数。

## 3.7 “特色的”加 1 和减 1 运算

在算术运算中，还经常使用加 1 或减 1 运算，因此，C++特地为这两种运算制定了特殊的运算符，即自增运算符++和自减运算符--。前者表示+1 运算，后者表示-1 运算。自增运算符和自减运算符在运算规则和运算过程上相同。因此，本文仅以自增运算符++为例进行讲解。

自增运算既可以有左值，也可以有右值，但是在同一时刻只能有一个操作数。如表达式 a++或++a。虽然二者均表示自增运算，但是它们之间还是有一定的区别。在计算 b + (a++)时，计算的过程是先进行 b+ a 的运算，然后再进行 a+1 运算。最终得到的结果是 a 变成了 a+1 以后的值，而该表达式的最终的结果是 b+a 的值。在计算 b +(++a)时，首先进行的是 a++运算得到结果 a，然后将新得到的 a 与 b 的值进行加运算。简单来说，a++表示先进行其他运算，然后再执行 a+1，而++a 则表示先进行 a+1 的运算，然后再进行其他运算。

**注意：**对于表达式 b +(a++)和 b+(++a)，自增运算的优先级要高于加运算的优先级，但是为了更直观地表示计算的先后顺序，使用了括号来改变优先级。如果不添加括号其运算顺序还是一样的。关于优先级的内容请参照本章的最后一节。

自增运算如示例程序 3-9.cpp 所示。

```
//-------------------------示例程序 3-9.cpp-------------------------
#include <stdio.h>
```

```
int main(void)
{
    int a, b;                                    //定义需要的变量
    int sum1, sum2;

    a = 12; b = 34;                              //给变量 a、b 赋值
    sum1 = b+ (a++);                             //计算 a++
    printf("a = %d , b+ (a++)= %d\n", a, sum1);
    a = 12; b = 34;                              //重新对 a 和 b 赋值
    sum2 = b+ (++a);                             //计算++a
    printf("a = %d , b+ (++a)= %d\n", a, sum2);

    return 0;
}
```

程序编译以后的运行结果如下：

```
a = 13, b+ (a++) = 46
a = 13, b+ (++a) = 47
```

在此示例程序中，在第 1 次计算时，先计算了 b+a 的值，即计算 34 + 12。得到结果 46，而 a 在进行了一次自加运算后，由 12 变成了 13。在第 2 次的计算时，先对 a 进行了一次自加运算，得到 a 的值为 13，然后再次计算 13 + 34，得到值为 47。

## 3.8　唯一需要三个表达式的条件运算符和表达式

条件运算符在一定程度上也属于一个复杂的运算符，其表示的基本原型如下：

```
(逻辑表达式) ? (语句 1) : (语句 2)
```

在计算过程中，首先判断逻辑表达式是否成立，即其值是否为真。如果逻辑表达式的结果为真，那么程序将执行语句 1。如果最终的结果为假，那么将执行语句 2。如条件表达式 a > b ? a:b，如果 a 大于 b 成立，则表达式的值为 a。如果 a 大于 b 不成立，则表达式的值为 b。如示例程序 3-10.cpp 所示。

```
//-------------------------示例程序 3-10.cpp-------------------------
#include <stdio.h>
int main(void)
{
    int a = 12, b = 34;                          //定义需要的变量
    a>b?printf("a>b\n"):printf("a< b\n");   //条件运算符的使用
    printf("a > b ?a:b = %d\n ", a > b ?a:b );
    printf("a < b ?a:b = %d\n ", a < b ?a:b );
    return 0;
}
```

程序编译后运行结果如下：

```
a< b
a > b ?a:b = 34
a < b ?a:b = 12
```

在此示例程序中，在进行第 1 次运算时，因为 a 的值 12，而 b 的值为 34，所以 a>b

不成立，所以 s 的值为 b 的值。而第 2 次运算由于 a<b 成立，所以整个表达式的值为 a，即 12。

通过示例程序 3-10.cpp 所示，条件运算符可以很简单地获取两个数的最大值或最小值。条件运算符可以嵌套使用，即条件表达式中的表达式还是一个条件表达式，如示例程序 3-11.cpp 所示。

```
//-------------------------示例程序 3-11.cpp-------------------------
#include <stdio.h>
int main(void)
{
    int a = 12, b = 34, c = 56;                 //定义需要的变量
    int s;

    a = 12; b = 34, c = 56;                     //给变量 a、b、c 赋值

    s = (a > b) ?(a > c?a:c); (b > c ? b :c);  //条件运算符的嵌套使用
    printf(" (a > b) ?(a > c?a:c); (b > c ? b :c) = %d\n", s);
    return 0;
}
```

程序编译后运行结果如下：

```
s = 56
```

在此示例程序中，首先判断 a 是否大于 b，如果 a 大于 b 成立，则再进行 a 和 c 的比较，如果 a 大于 c，则整个表达式的值为 a，否则 s 的值为 c。如果 a 大于 b 不成立，则进行 b 和 c 的比较，如果 b 大于 c 成立，表达式的值为 b，否则为 c。通过这个表达式，可以很简单地获取到 a、b、c 三者中的最大数。

注意：在使用条件表示复杂的运算时，要对表达式使用括号，防止由于运算符的结合顺序而造成运算错误。

# 3.9　取字节数操作 sizeof 和括号运算符

在 C++程序的运行过程中，执行完变量的声明语句后，就会给变量分配适当的内存，并将变量在初始化的过程中获取的值存储在该内存中。如果是复杂的数据类型，手动计算其获取的内存大小显然不是很实际的，因此在 C++中使用 sizeof 操作符来获取变量占用的字节数。

## 3.9.1　取字节数操作 sizeof

sizeof 操作符的作用是取操作数在内存中占用的字节数，其表达方式如下：

```
sizeof(操作数)
```

该操作数可以是数据类型名，也可以是具体的变量名，还可以是常数，甚至是函数名。按照 C99 的规定，除函数或不能确定类型的表达式以及位域成员以外，其余的都能作为 sizeof 的操作数。sizeof 的使用如示例程序 3-12.cpp 所示。

```
//-------------------------示例程序 3-12.cpp-------------------------
#include <stdio.h>
int main(void)
{
    int a = 12;                                    //定义需要的变量
    float f = 1.234;
    char ch = 'c';

    printf("sizeof(2) = %d\n", sizeof(2));     //获取常量 2 占用的字节数
    printf("sizeof(a) = %d\n", sizeof(a));     //获取整型变量 a 占用的字节数
    printf("sizeof(f) = %d\n", sizeof(f));     //获取浮点型变量 f 占用的字节数
    printf("sizeof(ch) = %d\n", sizeof(ch));   //获取字符型变量 ch 占用的字节数

    return 0;
}
```

程序编译后运行结果如下：

```
sizeof(2) = 4
sizeof(a) = 4
sizeof(f) = 4
sizeof(ch) = 1
```

通过程序的运行结果可知，整型变量在内存中占用的字节数为 4，而浮点类型的变量占用的字节数也是 4，字符型数据占用的字节数为 1。

注意：

（1）在使用变量时，可以不加括号，但是在使用数据类型名时，一定要添加括号。

（2）示例程序 3-12.cpp 中得到的各种数据类型占用的字节数是电脑的 CPU 为 32 位时获取的数值，如果 CPU 不为 32 位，则得到的结果会与本例中得到的结果不一致。

### 3.9.2　括号运算符

括号运算符是一对小括号，而小括号中间为一个表达式，这个表达式可以是任意形式的表达式。括号运算符的作用和数学中的四则运算中括号的作用是一样的，都是将优先级较低的运算级别提高，从而满足实际运算的需要。

如需要先计算 a 和 b 两个数的和，然后再计算与 c 的积，可以使用中间变量 t，使用赋值表达式 t=a+b，然后再使用 t*c 得到最终结果。另外一种方式是使用括号运算符，改变加法的运算优先级，使其高于乘运算的优先级，从而满足了先进行加运算，后进行乘运算的要求。

在前面的章节中已经出现了很多使用括号运算符的示例，在此不再重新举例，对该运算符不是很明了的读者可以参考前面章节中的示例。

## 3.10　运算符的结合顺序和优先级

像在本章开始时说过的数学中的四则运算一样，在一个表达式中出现多个运算符时，

会按照一定的顺序进行计算，就像是四则运算中的基本原则：先算乘或除，后算加或减。在 C++中也有类似的规则，这个规则就是运算符的结合顺序和运算符的优先级。

运算符的结合顺序是在运算的过程中运算符先和哪个操作数结合进行运算的一种规则。运算符的结合顺序包括两种：一种按照从左至右的顺序进行计算；一种是按照从右至左的顺序进行计算。除算术运算符、条件运算符和复合算术运算符的计算顺序是从右至左以外，其他的运算符的计算顺序都是从左至右。

在明白了运算符的结合顺序以后，在运算的过程中也会出现首先应该计算哪个运算，后进行哪个运算的问题，解决这个问题的方法就是运算优先级。就像在小学中学习的四则运算一样，虽然是先计算乘或除，后计算加或减。但是如果出现括号的话，我们就先计算括号中的内容。在 C++中，运算符的优先级分为 15 级，只有当高优先级的运算符计算完成后，低优先级的运算符才能进行计算。运算符的优先级详细说明见表 3-3。

**表 3-3　运算符优先级和结合顺序**

| 运算符 | 优先级 | 结合顺序 |
|---|---|---|
| () [] -> . | 最高 ↑ | 从左至右 |
| ! ~ ++ -- + - * & sizeof | | 自右向左 |
| * / % | | 自左向右 |
| + - | | 自左向右 |
| > | | 自左向右 |
| >= | | 自左向右 |
| == != | | 自左向右 |
| & | | 自左向右 |
| ^ | | 自左向右 |
| \| | | 自左向右 |
| && | | 自左向右 |
| \|\| | | 自左向右 |
| ?: | | 自右向左 |
| = += -= *= /= %= &= ^= \|= >= | ↓ 最低 | 自右向左 |

通过表 3-3 可知，优先级最高的依次是括号运算符、下标运算符、取成员变量运算符，而优先级最低的是各种复合运算符和赋值运算符。在优先级不同的运算符之间先进行优先级高的运算，然后进行优先级低的运算。在相同优先级的情形下，按照相应的结合顺序进行逐个运算。如示例程序 3-13.cpp 所示。

```
//-------------------------示例程序 3-13.cpp-------------------------
#include <stdio.h>
int main(void)
{
    Int num = 10, a = 2, b = 8,  c= 3;
    Int sum;                                  //定义需要的变量
    sum = num +( b++ - ++c) / a;              //进行运算
    printf("sum = %d\n", sum);                //结果输出
    return 0;
}
```

程序编译后运行结果如下：

```
sum = 12
```

在进行示例程序 3-13.cpp 的运算过程中，因为出现了括号，而括号运算的优先级最高，所以要先运算括号内的表达式。括号内是个减运算，按照自右至左的结合顺序，先计算++c 的值得到 4，然后在计算 b++，最终得到结果 4。随后表达式变成了加运算和除运算的复合表达式，由于除运算的优先级高于加运算，所以先计算 4/a，然后再将结果和 num 相加，最终得到 sum 的值为 12。

在 C++中，运算符的结合还会按照最大结合的原则进行运算符结合，即尽量为每一个变量结合最多的运算符，如示例程序 3-14.cpp 所示。

```
//-------------------------示例程序 3-14.cpp-------------------------
#include <stdio.h>
int main(void)
{
    int a = 10;
    int b = 20;                                    //定义变量
    printf("a+++++b = %d", a+++ ++b);              //复杂的算术运算
    return 0;
}
```

程序编译后运行结果如下：

```
a+++++b = 31
```

通过程序运行结果可知，当出现不同的运算符可以组成不同的表达式的情况时，编译器会自动为变量添加更多的运算符。但是除非必要，一般不要使用示例中的方式，这样会降低程序的可读性，并使结果变得非常“难以琢磨”。

**注意：**在复杂表达式中，为了更好地表现作者的意图，可以使用括号运算符来明显地表明运算符的优先级，从而使得表达式清晰易懂。

## 3.11　小　　结

本章主要介绍了 C++中的运算符和与之对应的表达式的使用方式，着重介绍了赋值运算符、算术运算符、关系运算符、逻辑运算符、逗号运算符、条件运算符以及取字节数运算符，并简单介绍了括号运算符，最后介绍了运算符的结合性和运算符的优先级。

在使用赋值运算符时，需要注意该符号的作用是将右值赋值给左值，而不是表示二者是相等的关系，表示关系运算中相等的关系是使用符号“==”，因此不要将二者混淆。而且在不同类型数值之间赋值时，会产生数值类型的变化或原数值精度的损失，因此在赋值时尽量使用类型相同的变量。

在使用算术运算时，要注意取余运算只适用于整数类型，而不能对其他类型进行取余运算。在算术运算中还有自增或自减操作符，表示加 1 或减 1 运算。

## 3.12　练　习　题

1．循环输入一个整数，判断其是偶数还是奇数。提示：对输入的整数使用对 2 取模运算，如果得到的余数为 0，则此数为偶数，否则为奇数。

2．输入一个字符，获取其 ASCII 码值。如输入字符‘A’，输出 A 的 ASCII 值为 65。

3．编写程序使用 sizeof 运算符获取 double 类型数据占用的字节数。

4．使用条件表达式获取 a 和 b 中的最小的数。假设 a = 12，b = 34。

5．使用条件表达式获取 a、b、c 中最小的数。假设 a = 12，b = 34，c = 56。