第 4 章　创建表达式和语句

从本质上说，程序是一组按顺序执行的命令。程序的强大功能源自它能够根据某个条件为真还是假来执行一组或另一组命令。

本章将学习：

- 什么是语句？
- 什么是语句块？
- 什么是表达式？
- 如何根据条件执行不同的程序代码？
- 什么是真值（truth）？如何使用它？

4.1　语句简介

在 C++中，语句控制程序的执行顺序、计算表达式的值或什么都不做（空语句）。所有的 C++语句都以分号结尾。最常见的语句之一是下面的赋值语句，如下所示：

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

与代数中不同，该语句并不表示 x 等于 a + b。它应该读作：将 a 与 b 的和赋给 x 或将 x 的值设置为 a 与 b 的和。

这条语句完成两项工作：将 a 和 b 相加，并使用赋值运算符（=）将结果赋给 x。虽然这条语句完成两项工作，但它只是一条语句，因此只有一个分号。

注意：赋值运算符将右边的值赋给左边的变量。

4.1.1　使用空白

空白（制表符、空格和换行符）是不可见的，它们被称为空白字符，因为在白纸上打印它们时，看到的只是纸张的白色。

语句中的空白通常被忽略。例如上面讨论的赋值语句可写成：

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

或

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

虽然后一种写法完全合法的，但这么做太傻。空白可使程序更易读和维护，但如使用不当，也可使程序变得很糟糕。在这方面，C++提供了强大功能，但程序员需要判断力。

4.1.2　语句块和复合语句

在任何可以使用单条语句的地方都可以使用复合语句（也叫语句块）。语句块以左大括号（{）开始，以

右大括号（}）结束。

虽然语句块中的每条语句都必须以分号结尾，但语句块本身不需要，如下例所示：

```
{
   temp = a;
   a = b;
   b = temp;
}
```

这个代码块是一条语句，它交换变量 a 和 b 的值。

应该：

语句一定要以分号结尾。

使用空白来提高代码的可读性。

不应该：

使用左大括号后，别忘了与之匹配的右大括号。

4.2 表 达 式

在 C++中，任何结果为一个值的东西都是表达式。表达式总是返回一个值。语句 3+2;返回值 5，因此它是表达式。所有表达式都是语句。

读者可能感到惊讶，有些代码也是合格的表达式。下面是 3 个这样的例子：

```
3.2                  // returns the value 3.2
PI                   // float constant that returns the value 3.14
SecondsPerMinute     // int constant that returns 60
```

假设 PI 是一个被初始化为 3.14 的常量，SecondsPerMinute 是一个值为 60 的常量，则上面 3 条语句都是表达式。

下面的表达式要复杂些：

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

它将 a 和 b 相加，将结果赋给 x，并返回所赋的值（x 的值）。因此这条语句也是表达式。

注意，任何表达式都可以放在赋值运算符的右边，这包括上述赋值语句。在 C++中，下面的代码是完全合法的：

```
y = x = a + b;
```

这行代码的计算顺序如下：

将 a 和 b 相加；

将表达式 a+b 的结果赋给 x；

将赋值表达式 x=a+b 的结果赋给 y。

如果 a、b、x、y 均为整型变量，且 a 的值为 9，b 的值为 7，则 x 和 y 的值都将为 16。程序清单 4.1 说明了这一点。

程序清单 4.1　计算复杂表达式的值

```
1:  #include <iostream>
2:  int main()
3:  {
4:     using std::cout;
```

```
 5:    using std::endl;
 6:
 7:    int a=0, b=0, x=0, y=35;
 8:    cout << "a: " << a << " b: " << b;
 9:    cout << " x: " << x << " y: " << y << endl;
10:    a = 9;
11:    b = 7;
12:    y = x = a+b;
13:    cout << "a: " << a << " b: " << b;
14:    cout << " x: " << x << " y: " << y << endl;
15:    return 0;
16: }
```

输出：

```
a: 0 b: 0 x: 0 y: 35
a: 9 b: 7 x: 16 y: 16
```

分析：

在第 7 行，程序声明并初始化了 4 个变量；第 8 和 9 行打印了它们的值。在第 10 行，将 9 赋给了 a；在第 11 行将 7 赋给了 b。第 12 行将 a 和 b 的值相加，并将结果赋给 x。表达式 x=a+b 的结果为一个值（a 与 b 的和），这个值被赋给了 y。在第 13 和 14 行，打印了这 4 个变量的值，以验证上述结果。

4.3 使用运算符

运算符是一种让编译器执行某种操作的符号。运算符作用于操作数，在 C++中，任何表达式都可用作操作数。在 C++中，运算符分几类，读者将学习的前两类运算符是：

- 赋值运算符。
- 数学运算符。

4.3.1 赋值运算符

赋值运算符（=）读者已经见过，它将左边的操作数的值修改为右边的表达式的值。下面的表达式将 a 加 b 的结果赋给操作数 x。

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

左值和右值

可放在赋值运算符左边的操作数称为左值，可放在右边的称为右值。

注意，所有的左值都是右值，但并非所有的右值都是左值。例如，字面常量是右值，但不是左值。因此，可以编写下面的代码：

```
x = 5;
```

但不能编写下面的代码：

```
5 = x;
```

x 可以用作左值或右值，但 5 只能用作右值。

注意：常量是右值。由于它们的值不能修改，因此不能放在赋值运算符的左边，这意味着它们不是左值。

4.3.2 数学运算符

另一类运算符是数学运算符，5 个数学运算符是加（+）、减（-）、乘（*）、除（/）和求模（%）。

加法和减法运算符的工作原理与读者预期的相同：将两个数字相加或相减。乘法的工作原理相同，但乘法运算符是星号（*）。除法运算符为斜杠。下面的范例表达式使用了所有这些运算符。在每个表达式中，结果都被赋给变量 result；右边的注释指出了 result 的值：

```
result = 56 + 32     // result = 88
result = 12 - 10     // result = 2
result = 21 / 7      // result = 3
result = 12 * 4      // result = 48
```

减法运算存在的问题

对 unsigned 整型变量执行减法运算时，如果结果为负，将出现奇怪的结果。这与前一章介绍的变量溢出时的情形类似。程序清单 4.2 说明了将较小的无符号数与较大的无符号数相减时出现的情况。

程序清单 4.2　减法和整型变量溢出

```
1:  // Listing 4.2 - demonstrates subtraction and
2:  // integer overflow
3:  #include <iostream>
4:
5:  int main()
6:  {
7:     using std::cout;
8:     using std::endl;
9:
10:    unsigned int difference;
11:    unsigned int bigNumber = 100;
12:    unsigned int smallNumber = 50;
13:
14:    difference = bigNumber - smallNumber;
15:    cout << "Difference is: " << difference;
16:
17:    difference = smallNumber - bigNumber;
18:    cout << "\nNow difference is: " << difference <<endl;
19:    return 0;
20: }
```

输出：

```
Difference is: 50
Now difference is: 4294967246
```

分析：

第 14 行使用了减法运算符，第 15 行打印了运算结果，这与预料的一致。第 17 行再次使用了运算符，但将一个较小的无符号数减去一个较大的无符号数。结果本应为负，但由于结果被赋给一个无符号变量，因此将发生前一章介绍的溢出。附录 C 对这一主题有详细的介绍。

4.3.3　整数除法和求模

整数除法与小学学习的情况相同。将 21 除以 4（21/4）时，执行的就是整数除法，结果为 5（余 1）。

第 5 个数学运算符对读者来说可能是新的。求模运算符（%）用于计算整数除法的余数。为计算 21 除以 4 的余数，可对 21 和 4 求模（21%4），结果为 1。

求模很有用。例如，你可能想每执行 10 次操作就打印一条语句。任何 10 的倍数对 10 取模后的结果均为 0。也就是说，1%10 为 1，2%10 为 2，依次类推，直至 10%10，其结果为 0。而 11%10 又回到 1，这种模式不断持续下去，直至下一个 10 的倍数 20，20%10 的结果再次为 0。在第 7 章讨论循环时将使用这种技术。

FAQ

我将 5 除以 3 时，结果为 1，请问哪里出现了问题？

答：将两个整数相除时，结果仍为整数。

因此 5/3 的结果为 1（正确的答案是，结果为 1 余 2。要得到余数，可使用 5%3，其结果为 2）。

要得到小数结果，必须使用浮点数（类型 float、double 或 long double）。

5.0/3.0 的结果为小数 1.66 667。

如果除数或被除数为浮点数，编译器将生成浮点商。然而，将结果赋给一个整型左值，结果将再次被截短。

4.4 赋值运算符与数学运算符的组合

经常需要将一个值和一个变量相加，并将结果赋给该变量。如果有变量 myAge，要将其值加 2，可以编写这样的代码：

```
int myAge = 5;
int temp;
temp = myAge + 2;    // add 5 + 2 and put it in temp
myAge = temp;        // put it back in myAge
```

前两行创建变量 maAge 和一个临时变量。第三行将变量 myAge 的值和 2 相加，并将结果赋给 temp。在接下来的一行中，将这个值放回到 myAge 中，从而更新该变量。

然而，这种方法既然迂回又浪费。在 C++中，可以在赋值运算符两边使用同一个变量，因而前面的代码变为：

```
myAge = myAge + 2;
```

这更好，也更清晰。在代数学中，该表达式毫无意义，但在 C++中，它的意思是：将 myAge 和 2 相加，并将结果赋给 myAge。

还有一种更简单但更难懂的写法：

```
myAge +=2;
```

这行代码使用了自赋值加运算符（+=）。这种运算符将右值与左值相加，并将结果重新赋给左值。如果 myAge 原来的值为 24，则执行上述语句后，其值为 26。

另外，还有自赋值减法运算符（-=）、自赋值乘法运算符（*=）、自赋值除法运算符（/=）、自赋值求模运算符（%=）。

4.5 递增和递减

相加（相减）后，再将结果赋给变量时，最常用的值是 1。在 C++中，增加 1 被称为递增，减 1 被称为递减。C++有专门用于执行这些运算的运算符。

递增运算符（++）将变量的值加 1；递减运算符（--）将变量的值减 1。因此，如果有变量 Counter，要对其进行递增，可使用如下语句：

```
Counter++;            // Start with Counter and increment it.
```

该语句与如下更冗长的语句等价：

```
Counter = Counter + 1;
```

也与如下中等冗长的语句等价：

```
Counter += 1;
```

注意：正如读者可能已经猜到的，名称 C++意味着对其前身 C 执行递增运算，其含义是 C++在 C 语言的基础上进行了增量改进。

前缀和后缀

递增运算符（++）和递减运算符（--）有两种版本：前缀和后缀。前缀版本是将运算符放在变量名前（++myAge），后缀版本是放在变量名后（myAge++）。

在简单语句中，使用前缀还是后缀版本无关紧要；但在对变量进行递增（递减），然后将结果赋给另一个变量这样的复杂语句中，它们的区别将相当大。

使用前缀运算符时，在赋值前进行递增或递减；使用后缀运算符时，在赋值后进行递增或递减。前缀的语义是：将变量递增，然后再使用它；后缀的语义不同：先使用变量，然后再递增。

这一点刚开始可能让你感到糊涂，但通过下例你就明白了。如果整型变量 x 的值为 5，并使用前缀运算符编写了如下代码：

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

这告诉编译器，先将 x 递增（其值变为 6），再将该变量的值赋给 a。这样，执行该语句后，a 和 x 的值均为 6。

如果执行上述操作后，使用后缀运算符编写了如下语句：

```
int b = x++;
```

该语句告诉编译器，先将 x 的值（6）赋给 b，再将 x 递增。因此，执行该语句后，b 的值为 6，x 的值为 7。程序清单 4.3 说明了前缀和后缀运算符的用法和含义。

程序清单 4.3　前缀和后缀运算符

```
// Listing 4.3 - demonstrates use of
// prefix and postfix increment and
// decrement operators
#include <iostream>
int main()
{
   using std::cout;

   int myAge = 39;    // initialize two integers
   int yourAge = 39;
   cout << "I am: " << myAge << " years old.\n";
   cout << "You are: " << yourAge << " years old\n";
   myAge++;          // postfix increment
   ++yourAge;        // prefix increment
   cout << "One year passes...\n";
   cout << "I am: " << myAge << " years old.\n";
   cout << "You are: " << yourAge << " years old\n";
   cout << "Another year passes\n";
   cout << "I am: " << myAge++ << " years old.\n";
   cout << "You are: " << ++yourAge << " years old\n";
   cout << "Let's print it again.\n"
   cout << "I am: " << myAge << " years old.\n";
   cout << "You are: " << yourAge << " years old\n";
   return 0;
```

```
25: }
```

输出：

```
I am      39 years old
You are   39 years old
One year passes
I am      40 years old
You are   40 years old
Another year passes
I am      40 years old
You are   41 years old
Let's print it again
I am      41 years old
You are   41 years old
```

分析：

在第 9 和 10 行，声明了两个整型变量，并将它们都初始化为 39。第 11 和 12 行打印了这两个变量的值。

在第 13 行，使用后缀递增运算符将 myAge 递增；在第 14 行，使用前缀递增运算符将 yourAge 递增。第 16 和 17 行打印结果，它们的值相同，都是 40。

在第 19 行，在打印语句中，使用后缀递增运算符将 myAge 递增。由于是后缀运算符，递增发生在打印后，因此再次打印 40，然后将变量 myAge 递增。而在第 20 行，使用前缀递增运算符将 yourAge 递增，因此递增在打印前进行，显示的值为 41。

最后，在第 22 和 23 行，再次打印两个变量的值。由于递增运算已经完成，因此 myAge 的值为 41，与 yourAge 的值相同。

4.6 理解运算符优先级

对于下述复杂语句：

```
x = 5 + 3 * 8;
```

先执行加法还是乘法运算呢？如果先执行加法，结果为 8*8（64）；如果先执行乘法，结果为 5 + 24（29）。

C++标准对运算顺序做了规定。每个运算符都有优先级，详情请参阅附录 C。乘法的优先级比加法高，因此上述表达式的值为 29。

当两个数学运算符的优先级相同时，执行顺序为自左至右。因此，对于下面的语句：

```
x = 5 + 3 + 8 * 9 + 6 * 4;
```

先自左至右执行乘法。8*9=72，6*4=24，因此该表达式变成：

```
x = 5 + 3 + 72 + 24;
```

现在自左至右执行加法，5+3=8，8+72=80，80+24=104。

有些运算符，如赋值运算符（=），其执行顺序为自右至左。

如果优先级不能满足需求怎么办呢？请看如下表达式：

```
TotalSeconds = NumMinutesToThink + NumMinutesToType*60
```

在这个表达式中，你不想将变量 NumMinutesToType 乘以 60，再加上 NumMinutesToThink，而是想先将这两个变量相加得到总分钟数，然后再乘以 60 得到总秒数。

在这种情况下，可以使用括号来改变优先级顺序。用括号括起的项的优先级比任何数学运算符都高。因此，上例应改成这样：

```
TotalSeconds = (NumMinutesToThink + NumMinutesToType)*60
```

4.7　括号的嵌套

对于复杂的表达式，可能需要将一对括号嵌套到另一对括号内。例如，可能要首先计算总秒数和参加人数，再求其乘积：

```
TotalPersonSeconds = ((( NumMinutesToThink + NumMinutesToType) * 60) *
(PeopleInTheOffice + PeopleOnVacation))
```

这个复杂表达式将由内向外计算。首先，将最内层括号中的 NumMinutesToThink 和 NumMinutesToType 相加，并将结果乘以 60；然后，将 PeopleInTheOffice 与 PeopleOnVacation 相加，并将结果与总秒数相乘。

这个例子提出了一个相关的问题。上述表达式虽然对于计算机来说很易理解，但对于人来讲难以阅读、理解和修改。使用一些临时整型变量，可将该表达式修改如下：

```
TotalMinutes = NumMinutesToThink + NumMinutesToType;
TotalSeconds = TotalMinutes * 60;
TotalPeople = PeopleInTheOffice + PeopleOnVacation;
TotalPersonSeconds = TotalPeople * Total_Seconds;
```

修改后，需要输入的代码更多，使用的临时变量也更多，但更容易理解得多。如果在代码前加上注释，对其功能进行解释，并用符号常量代替 60，代码将易于理解和维护。

应该：

请切记表达式都有值。

要先对变量进行递增或递减，然后使用其值，请使用前缀运算符。

要先使用变量的值，然后进行递增或递减，请使用后缀运算符。

使用括号来改变优先顺序。

不应该：

不要过深地嵌套括号，否则表达式将难以理解和维护。

不要将前缀运算符和后缀运算符混淆。

4.8　真值的本质

可以将每个表达式的值视为真或假。如果表达式的值为零，则返回 false，否则返回 true。

在早期的 C++版本中，真和假都用整数表示；新的 ANSI 标准引入了 bool 类型。bool 变量只有两个可能的取值：false 或 true。

注意：以前很多编译器也提供了 bool 类型，它在内部表示为 long int，因此占用 4 字节。现在，遵循 ANSI 标准的编译器提供的 bool 类型通常只占 1 字节。

关系运算符

关系运算符用来对两个数字进行比较，判断它们相等、前者大于后者还是前者小于后者。关系语句的值要么为 true，要么为 false。表 4.1 列出了关系运算符。

注意：所有关系运算符都返回一个 bool 值，即 true 或 false。在老版本的 C++中，这些运算符要么返回 0（表示 false），要么返回非零值（通常为 1，表示 true）。

如果整型变量 myAge 和 yourAge 的值分别为 45 和 50，可以用关系运算符（==）来判断它们是否相等：

```
myAge == yourAge;  // is the value in myAge the same as in yourAge?
```

该表达式的值为 false，因为这两个变量不相等。同样，可以使用下面的表达式来判断 myAge 是否小于 yourAge：

```
myAge < yourAge;  // is myAge less than yourAge?
```

该表达式的结果为 true，因为 45 小于 50。

警告：很多 C++初学者将赋值运算符（=）与相等运算符（==）混为一谈，这将在程序中造成重大错误。

6 个关系运算符分别是：等于（==）、小于（<）、大于（>）、小于等于（<=）、大于等于（>=）、不等于（!=）。表 4.1 列出了每个关系运算符及其使用范例。

表 4.1　　关系运算符

名　称	运 算 符	范　例	结　果
等于	==	100 == 50;	false
		50 == 50;	true
不等	!=	100 != 50;	true
		50 != 50;	false
大于	>	100 > 50;	true
		50 > 50;	false
大于等于	>=	100 >= 50;	true
		50 >= 50;	true
小于	<	100 < 50;	false
		50 < 50;	false
小于等于	<=	100 <= 50;	false
		50 <= 50;	true

应该：

请切记，关系运算符返回 true 或 false。

不应该：

不要将赋值运算符（=）与关系运算符（==）混为一谈。这是最常见的 C++编程错误之一，一定要当心。

4.9　if 语句

通常情况下，程序按语句在源代码中出现的顺序逐行执行。if 语句让你能够测试某个条件（如两个变量是否相等），然后根据结果，执行不同的代码片段。

最简单的 if 语句如下所示：

```
if (expression)
    statement;
```

括号中的 expression 可以是任何表达式，但通常都包含一个关系表达式。如果表达式的值为 false，则跳过下条语句。如果表达式的值为 true，则执行该语句。请看下面这个例子：

```
if (bigNumber > smallNumber)
    bigNumber = smallNumber;
```

上述代码对 bigNumber 和 smallNumber 进行比较。如果 bigNumber 更大，则执行第 2 行代码，将 smallNumber 的值赋给它；否则，跳过这行代码。

由于用大括号括起来的语句块相当于一条语句，因此分支可能很大，且功能很强大：

```
if (expression)
{
    statement1;
    statement2;
    statement3;
}
```

下面是一个这种用法的简单例子：

```
if (bigNumber > smallNumber)
{
   bigNumber = smallNumber;
   std::cout << "bigNumber: " << bigNumber << "\n";
   std::cout << "smallNumber: " << smallNumber << "\n";
}
```

如果 bigNumber 比 smallNumber 大，不仅将 smallNumber 的值赋给 bigNumber，还将打印一条消息。程序清单 4.4 是一个更详细的基于关系运算符执行不同分支的例子。

程序清单 4.4　基于关系运算符进行分支

```
 1: // Listing 4.4 - demonstrates if statement
 2: // used with relational operators
 3: #include <iostream>
 4: int main()
 5: {
 6:    using std::cout;
 7:    using std::cin;
 8:
 9:    int MetsScore, YankeesScore;
10:    cout << "Enter the score for the Mets: ";
11:    cin >> MetsScore;
12:
13:    cout << "\nEnter the score for the Yankees: ";
14:    cin >> YankeesScore;
15:
16:    cout << "\n";
17:
18:    if (MetsScore > YankeesScore)
19:       cout << "Let's Go Mets!\n";
20:
21:    if (MetsScore < YankeesScore)
22:    {
23:       cout << "Go Yankees!\n";
24:    }
25:
26:    if (MetsScore == YankeesScore)
27:    {
28:       cout << "A tie? Naah, can't be.\n";
29:       cout << "Give me the real score for the Yanks: ";
30:       cin >> YankeesScore;
31:
```

```
32:       if (MetsScore > YankeesScore)
33:          cout << "Knew it! Let's Go Mets!";
34:
35:       if (YankeesScore > MetsScore)
36:          cout << "Knew it! Go Yanks!";
37:
38:       if (YankeesScore == MetsScore)
39:          cout << "Wow, it really was a tie!";
40:    }
41:
42:    cout << "\nThanks for telling me.\n";
43:    return 0;
44: }
```

输出：

```
Enter the score for the Mets: 10

Enter the score for the Yankees: 10

A tie? Naah, Can't be
Give me the real score for the Yanks: 8
Knew it! Let's Go Mets!
Thanks for telling me.
```

分析：

该程序要求用户输入两个棒球队的得分，它们被保存在整型变量 MetsScore 和 YankeesScores。在第 18、21 和 26 行，使用 if 语句对这两个变量进行比较。

如果一个得分比另一个高，则打印一条消息。如果得分相等，则执行第 27～40 行的代码块：再次要求用户输入得分，并对它们进行比较。

注意，如果一开始 Yankee 队的得分比 Mets 队高，第 18 行的 if 语句的值将为 false，因此第 19 行不会被执行。在这种情况下，第 21 行的测试将为 true，并执行第 23 行的语句。接下来，测试第 26 行的 if 语句，而其值为 false，因此程序跳过整个语句块，执行第 41 行。

这个范例表明，某条 if 语句的值为 true，并不会导致程序跳过后续的 if 语句。

前两条 if 语句对应的操作代码都只有 1 行（打印 Let's Go Mets!或 Go Yankees!）。在第 1 条 if 语句（18 行）中，对应的操作代码没有用大括号括起；因为单行代码块不需要。然而，用大括号括起也是合法的，如 22～24 行所示。

避免使用 if 语句时的常见错误

许多 C++新手会无意中在 if 语句后面加一个分号：

```
if(SomeValue<10);     // Oops! Notice the semicolon!
   SomeValue = 10;
```

程序员的本意是测试 SomeValue 是否小于 10，如果是，将其值设置为 10，从而使 SomeValue 的最小值为 10。如果运行上述语句片段，将发现 SomeValue 的值总为 10。为什么会这样呢？原因就在于在 if 语句后面加了一个分号（什么也不做的运算符）。

还记得吗，缩进对编译器来说没有任何意义。上面这段代码可以更准确地写成下面这样：

```
if (SomeValue < 10)  // test
;  // do nothing
SomeValue = 10;  // assign
```

删除分号将使最后一行成为 if 语句的一部分，这样代码按程序员的本意执行。

为最大限度地减少出现这种问题的机会，可以在编写 if 语句时总是使用大括号，即使 if 语句体只有 1 行：

```
if (SomeValue < 10)
{
    SomeValue = 10;
};
```

4.9.1　缩进风格

程序清单 4.4 演示了一种 if 语句缩进风格。然而，如果询问一组程序员，最佳的大括号对齐风格是什么，无疑会爆发一场宗教战争。尽管目前有十几种风格，但最流行的是以下 3 种：

- 将左大括号放在条件后，右大括号与 if 对齐：

```
if (expression){
   statements
}
```

- 将左、右大括号均与 if 对齐，并缩进语句：

```
if (expression)
{
   statements
}
```

- 缩进大括号和语句：

```
if (expression)
   {
   statements
   }
```

本书采用第二种风格，因为将大括号和测试条件对齐时，更容易知道语句块从何处开始到何处结束。同样，选用哪种风格关系并不大，只要保持一致即可。

4.9.2　else 语句

程序经常需要在条件为真时执行一个分支，而条件为假时执行另一个分支。在程序清单 4.4 中，如果第一个条件（MetsScore>Yankees score）为真，将打印一条消息（Let's Go Mets!）；如果该条件为假，将打印另一条消息（Go Yankees!）。

迄今为止使用的方法（先测试一个条件然后再测试另一个条件）是可行的，但有点啰嗦。使用关键字 else 可使代码的可读性更好：

```
if (expression)
    statement;
else
    statement;
```

程序清单 4.5 演示了关键字 else 的用法。

程序清单 4.5　关键字 else 的用法

```
1:  // Listing 4.5 - demonstrates if statement
2:  // with else clause
3:  #include <iostream>
4:  int main()
5:  {
6:     using std::cout;
7:     using std::cin;
```

```
 8:
 9:    int firstNumber, secondNumber;
10:    cout << "Please enter a big number: ";
11:    cin >> firstNumber;
12:    cout << "\nPlease enter a smaller number: ";
13:    cin >> secondNumber;
14:    if (firstNumber > secondNumber)
15:       cout << "\nThanks!\n";
16:    else
17:       cout << "\nOops. The first number is not bigger!";
18:
19:    return 0;
20: }
```

输出：

```
Please enter a big number: 10

Please enter a smaller number: 12
Oops. The first number is not bigger!
```

分析：

执行第 14 行的 if 语句。如果条件为真，执行第 15 行的语句，然后程序流程进入第 18 行（else 语句的后面）；如果第 14 行的条件为假，则进入 else 子句，执行第 17 行的语句。如果删掉第 16 行的 else 子句，则无论 if 语句的条件是真还是假，程序都将执行第 17 行的语句。

请记住，if 语句在第 15 行结束。如果没有 else 子句，第 17 行将是程序的下一条语句。另外，if 语句体和 else 语句体都可以是用大括号括起的代码块。

if 语句

if 语句的语法有如下两种形式：

第 1 种形式：

```
if (expression)
      statement;
next_statement;
```

如果 expression 为真，将执行 statement，再执行 next_statement；如果为假，将不执行 statement，而直接执行 next_statement。

请记住，statement 可以是以分号结尾的单条语句，也可以是用大括号括起的代码块。

第 2 种形式：

```
if (expression)
       statement1;
else
       statement2;
next_statement;
```

如果 expression 为真，则执行 statement1；否则执行 statement2。然后，程序继续执行 next_statement。

范例 1：

```
if (SomeValue < 10)
   cout << "SomeValue is less than 10");
else
```

```
    cout << "SomeValue is not less than 10!");
cout << "Done." << endl;
```

4.9.3　高级 if 语句

值得注意的是，任何语句都可用于 if 或 else 子句中，甚至可以是另一条 if 或 else 语句。因此你可能看到下面这种形式的复杂 if 语句：

```
if (expression1)
{
   if (expression2)
      statement1;
   else
   {
      if (expression3)
         statement2;
      else
         statement3;
   }
}
else
   statement4;
```

这个复杂 if 语句的意思是：如果 expression1 和 expression2 均为真，则执行 statement1。如果 expression1 为真而 expression2 为假，且 expression3 为真，则执行 statement2。如果 expression1 为真，而 expression2 和 expression3 均为假，则执行 statement3。最后，如果 expression1 为假，则执行 statement4。正如读者看到的，复杂 if 语句很容易令人迷惑。

程序清单 4.6 给出了一个复杂 if 语句的例子。

程序清单 4.6　嵌套的复杂 if 语句

```
// Listing 4.6 - a complex nested
// if statement
#include <iostream>
int main()
{
   // Ask for two numbers
   // Assign the numbers to bigNumber and littleNumber
   // If bigNumber is bigger than littleNumber,
   // see if they are evenly divisible
   // If they are, see if they are the same number

   using namespace std;

   int firstNumber, secondNumber;
   cout << "Enter two numbers.\nFirst: ";
   cin >> firstNumber;
   cout << "\nSecond: ";
   cin >> secondNumber;
   cout << "\n\n";

   if (firstNumber >= secondNumber)
   {
     if ( (firstNumber % secondNumber) == 0) // evenly divisible?
```

```
24:       {
25:          if (firstNumber == secondNumber)
26:             cout << "They are the same!\n";
27:          else
28:             cout << "They are evenly divisible!\n";
29:       }
30:       else
31:          cout << "They are not evenly divisible!\n";
32:    }
33:    else
34:       cout << "Hey! The second one is larger!\n";
35:    return 0;
36: }
```

输出：

```
Enter two numbers.
First: 10

Second: 2
They are evenly divisible!
```

分析：

程序提示用户输入两个数，每次输入一个，然后对它们进行比较。第 21 行的第 1 条 if 语句检查第一个数是否大于等于第 2 个数。如果不是，则执行第 33 行的 else 子句。

如果第 1 条 if 语句为真，则执行从第 22 行开始的代码块，并测试第 23 行的第 2 条 if 语句。这条 if 语句检查将第 1 个数除以第 2 个数时，余数是否为 0。如果是，这两个数要么相等，要么前者是后者的整数倍。第 25 行的第 3 条 if 语句检查两数是否相等，并根据结果打印相应的消息。

如果第 23 行的 if 语句为假，将执行第 30 行的 else 子句。

4.10 在嵌套 if 语句中使用大括号

虽然 if 语句体只有一条语句时，不使用大括号也完全合法，对嵌套 if 语句也如此，但这样做将令人迷惑。下面的代码在 C++中是完全合法的，但看起来让人感到迷惑：

```
if (x > y)          // if x is bigger than y
   if (x < z)       // and if x is smaller than z
      x = y;        // set x to the value in y
   else             // otherwise, if x isn't less than z
      x = z;        // set x to the value in z
else                // otherwise if x isn't greater than y
   y = x;           // set y to the value in x
```

空格和缩进只是为程序员提供方便，对编译器来说毫无意义。如果不使用大括号，很容易对其中的逻辑感到迷惑，将 else 与 if 的对应关系搞错。程序清单 4.7 演示了这种问题。

程序清单 4.7　为什么大括号有助于阐明 else 语句与 if 语句的对应关系

```
1: // Listing 4.7 - demonstrates why braces
2: // are important in nested if statements
3: #include <iostream>
4: int main()
```

```
 5: {
 6:    int x;
 7:    std::cout << "Enter a number less than 10 or greater than 100: ";
 8:    std::cin >> x;
 9:    std::cout << "\n";
10:
11:    if (x >= 10)
12:      if (x > 100)
13:         std::cout << "More than 100, Thanks!\n";
14:    else                          // not the else intended!
15:      std::cout << "Less than 10, Thanks!\n";
16:
17:    return 0;
18: }
```

输出：

```
Enter a number less than 10 or greater than 100: 20

Less than 10, Thanks!
```

分析：

程序员的本意是要用户输入一个小于 10 或大于 100 的数，然后检查输入的值是否正确，并打印一条致谢消息。

如果第 11 行的 if 语句为真，将执行第 12 行的语句。在这个例子中，当输入的数字大于 10 时，将执行第 12 行的语句。第 12 行也包含一条 if 语句；如果输入的数大于 100，该 if 语句为真。如果输入的数大于 100，将执行第 13 行的语句：打印一条相应的消息。

如果输入的数小于 10，第 11 行的 if 语句将为假。程序将跳到该 if 语句后面的语句，这里为第 16 行。如果用户输入一个比 10 小的数，将出现如下输出：

```
Enter a number less than10 or greater than 100: 9
```

正如读者看到的，没有打印消息。程序员的本意是，第 14 行的 else 与第 11 行的 if 匹配，并进行了相应的缩进。不幸的是，编译器将这个 else 与第 12 行的 if 相对应，于是出现了上述微妙的错误。

之所以称之为微秒的错误，是因为编译器没有指出。这是一个合法的 C++程序，只不过没有实现程序员的意图。另外，程序员测试该程序时，往往看似能正常运行：只要输入的数大于 10 时，程序将看似一切正常。然而，如果输入一个 11～99 的数，将发现程序明显存在问题！

程序清单 4.8 通过加入必要的大括号，解决了这个问题。

程序清单 4.8　在 if 语句中正确地使用大括号

```
 1: // Listing 4.8 - demonstrates proper use of braces
 2: // in nested if statements
 3: #include <iostream>
 4: int main()
 5: {
 6:    int x;
 7:    std::cout << "Enter a number less than 10 or greater than 100: ";
 8:    std::cin >> x;
 9:    std::cout << "\n";
10:
11:    if (x >= 10)
12:    {
```

```
13:      if (x > 100)
14:         std::cout << "More than 100, Thanks!\n";
15:    }
16:    else                          // fixed!
17:      std::cout << "Less than 10, Thanks!\n";
18:    return 0;
19: }
```

输出：

```
Enter a number less than 10 or greater than 100: 9
Less than 10, Thanks!
```

分析：

第 12 行和 15 行的一对大括号使得它们之间的所有代码成为一条语句。现在，第 16 行的 else 与第 11 行的 if 相匹配，这与程序员的本意相符。

如果用户输入 9，第 11 行的 if 语句将为真；然而第 13 行的 if 语句为假，因此不打印任何消息。如果在第 14 行后再加一个 else 子句，以捕获输入错误并打印一条消息，程序将更完美。

提示：可以在 if 和 else 子句中使用大括号（即使条件后面只有一条语句），以最大限度地减少 if...else 语句可能带来的问题：

```
if (SomeValue < 10)
{
    SomeValue = 10;
}
else
{
    SomeValue = 25;
};
```

注意：本书所有的程序都是针对要讨论的问题而编写的。因此都很简单，没有包含防范用户操作错误的代码。在专业品质的代码中，应考虑到用户可能犯的所有操作错误，并妥善进行处理。

4.11 使用逻辑运算符

经常需要同时问多个关系型问题，例如，x 大于 y 且 y 大于 z 吗？程序可能需要确定这两个条件（或一组其他的条件）都为真时，才执行某项操作。

假设一个复杂的警报系统，其逻辑为：如果在非节假日的下午 6 点以后或周末，大门警报器响起，则报警。C++提供了 3 个逻辑运算符，用于完成这种判断，如表 4.2 所示。

表 4.2　　逻辑运算符

运 算 符	符 号	范 例
AND	&&	*expression1 && expression2*
OR	||	*expression1 || expression2*
NOT	!	*!expression*

4.11.1 逻辑 AND 运算符

逻辑 AND 语句使用 AND 运算符来连接和计算两个表达式的值，如果两个表达式均为真，逻辑 AND 语句也为真。如果你饿了，且身上有钱，则会买午餐。因此，如果 x 和 y 都为 5，在下面的语句为真；如果任

何一个不为 5，则为假：

```
if ( (x == 5) && (y == 5) )
```

注意，要使整个表达式为真，&&两边都必须为真。

逻辑 AND 运算符为&&。单个&是另一种运算符，将在第 21 章介绍。

4.11.2　逻辑 OR 运算符

逻辑 OR 语句也对两个表达式进行计算。如果任何一个为真，则整个表达式为真。如果有现钱或信用卡，可以付账。不必既有现金又有信用卡，而只需有一种就行了，尽管两者都有也行。因此如果 x 或 y 等于 5，或者 x、y 均等于 5，下面的语句为真：

```
if ( (x == 5) || (y == 5) )
```

逻辑 OR 运算符为||。单个 | 是另一种运算符，将在第 21 章介绍。

4.11.3　逻辑 NOT 运算符

如果被测试的表达式为假，则逻辑 NOT 语句为真。同样，如果表达式为假，则逻辑 NOT 测试为真。因此，仅当 x 不等于 5 时，下面的测试才为真：

```
if ( !(x == 5) )
```

该语句与下面的语句等价：

```
if (x != 5)
```

4.12　简化求值

编译器计算如下 AND 语句的值时，首先判断第一个表达式（x == 5）是否为真，如果为假（即 x 不等于 5），则不再判断第二个表达式（y==5）是否为真，因为仅当两个表达式都为真时，AND 语句才为真。

```
if ( (x == 5) && (y == 5) )
```

同样，如果编译器计算如下所示的 OR 语句的值时，如果第一个表达式（x==5）为真，编译器将不再计算第二个表达式的值，因为任何一个表达式为真，整个 OR 语句就为真。

```
if ( (x == 5) || (y == 5) )
```

这虽然看似不重要，但请看下面的范例：

```
if ( (x == 5 ) || (++y == 3) )
```

如果 x 不为 5，将不计算表达式（++y == 3）。如果程序员指望无论在什么情况下都将 y 递增，这种愿望将落空。

4.13　关系运算符的优先级

和所有 C++表达式一样，使用关系运算符和逻辑运算符时，都将返回一个值：true 或 false。和所有表达式一样，它们都有优先级顺序（参见附录 C），这决定了先计算哪个关系的值。在计算下述语句的值时，这一点非常重要：

```
if ( x > 5 && y > 5 || z > 5)
```

程序员可能希望这样：如果 x 和 y 均大于 5 或 z 大于 5，则该表达式为真。另一方面，程序员可能希望这样：仅当 x 大于 5 且 y 或 z 大于 5 时，该表达式的值为真。

如果 x 为 3，y 和 z 均为 10，且采用第一种解释，则整个表达式为真（z 大于 5，因此忽略 x 和 y）；而采用第二种解释时，整个表达式为假（x 大于 5 为假，因此&&右边是什么无关紧要，因为两边都必须为真，整个表达式才为真）。

虽然运算符的优先级决定了先计算哪个关系，但使用括号可以改变优先级，使语句更清晰：

```
if ( (x > 5) && (y > 5 || z > 5) )
```

使用前面的取值时，表达式为假。由于 x 大于 5 为假，即 AND 语句的左边为假，因此整条语句为假。别忘了，仅当 AND 语句两边均为真时整个语句才为真：如果某种东西的味道不好，那么它不可能既好吃又对身体有益。

提示：通常使用括号，可以阐明你要将什么组合起来，这样做通常是个不错的主意。别忘了，目标是编写能够正确运行且易于阅读和理解的程序。使用括号有助于阐明意图，避免由于错误地理解了运算符的优先级所带来的错误。

4.14 再谈真和假

在 C++中，0 被解释为代表假，其他值都被解释为真。表达式总是有一个值，许多程序员在 if 语句中利用这一特点。例如，下述语句的含义为，如果 x 不为零，将其设置为 0：

```
if (x)          // if x is true (nonzero)
   x = 0;
```

这有点玩技巧，如果写成下面这样将更清楚：

```
if (x != 0)     // if x is not zero
   x = 0;
```

这两种写法都合法，但后者更清楚。良好的编程习惯是，用前一种方法来判断逻辑真和假，而不使用它来判断非 0 值。

下面这两条语句也是等效的：

```
if (!x)          // if x is false (zero)
if (x == 0)      // if x is zero
```

但第 2 条语句更容易理解，同时如果要判断的是 x 的数学值而不是逻辑状态，这条语句更明确。

应该：

请在逻辑测试中使用括号，使它们更清楚，优先顺序更明确。

务必在嵌套 if 语句中使用大括号，使 else 语句的对应关系更清楚并防止出错。

不应该：

不要用 if（x）来代替 if（x!=0）；因为后者更清晰。

不要用 if（!x）来代替 if（x==0）；因为后者更清晰。

4.15 条件运算符（三目运算符）

条件运算符（?:）是 C++中惟一一个三目运算符；即它是惟一一个需要 3 个操作数的运算符。

条件运算符接受 3 个表达式并返回一个值：

```
(expression1) ? (expression2) : (expression3)
```

其含义是：如果 expression1 为真，则返回 expression2 的值；否则返回 expression3 的值。通常，返回的值将赋给一个变量。程序清单 4.9 演示了如何使用条件运算符来代替 if 语句。

程序清单 4.9　条件运算符

```
1:  // Listing 4.9 - demonstrates the conditional operator
2:  //
3:  #include <iostream>
4:  int main()
5:  {
6:     using namespace std;
7:
8:     int x, y, z;
9:     cout << "Enter two numbers.\n";
10:    cout << "First: ";
11:    cin >> x;
12:    cout << "\nSecond: ";
13:    cin >> y;
14:    cout << "\n";
15:
16:    if (x > y)
17:       z = x;
18:    else
19:       z = y;
20:
21:    cout << "After if test, z: " << z;
22:    cout << "\n";
23:
24:    z = (x > y) ? x : y;
25:
26:    cout << "After conditional test, z: " << z;
27:    cout << "\n";
28:    return 0;
29: }
```

输出：

```
Enter two numbers.
First: 5

Second: 8

After if test, z: 8
After conditional test, z: 8
```

分析：

创建了 3 个整型变量：x、y 和 z。前两个变量的值由用户提供。第 16 行的 if 语句检查 x 和 y 哪个大，然后将较大的值赋给 z。第 21 行打印这个值。

第 24 行的条件运算符进行同样的检查，并将较大的值赋给 z。这行代码的含义是：如果 x 大于 y，返回 x 的值；否则返回 y 的值；然后将返回的值赋给 z。第 26 行打印这个值。正如读者看到的，可用条件语句替代 if...else 语句，但它更简短。

4.16　小　　结

在本章中，读者学习了什么是 C++语句和表达式、C++运算符的作用以及 C++ if 语句的工作原理。

在可以使用单条语句的任何地方，也可以使用一对大括号括起的语句块。

每个表达式都有一个值，可以使用 if 语句或条件运算符对其进行测试。另外，读者还学会了如何使用逻辑运算符计算多条语句的值，如何使用关系运算符对两个值进行比较以及如何使用赋值运算符进行赋值。

本章还介绍了运算符优先级。读者知道了如何使用括号来改变优先顺序，并使优先顺序更清晰，更容易理解。

4.17 问 与 答

问：在优先级能够确定运算符的执行顺序时，为什么还要使用不必要的括号？

答：虽然不用括号，编译器也知道优先级，而程序员也可以查阅优先级顺序；但通过使用括号，可使程序更容易理解，进而更容易维护。

问：如果关系运算符总是返回真或假，为什么要规定任何非零值都表示真？

答：这种约定是从 C 语言那里继承而来的，编写低级软件（如操作系统和实时控制软件）时经常会使用这种约定。这种用法可能成为检测掩码或变量的所有位是否为 0 的简捷方式。

虽然关系运算符返回真或假，但任何表达式都会返回一个值，在 if 语句中也可以对这些值进行计算。下面是一个这样的例子：

```
if ( (x = a + b) == 35 )
```

这是一条完全合法的 C++语句。即使 a 与 b 的和不等于 35，该表达式也会返回一个值。另外，无论什么情况下，a 与 b 的和都将被赋给 x。

问：制表符、空格、换行符对程序有何影响？

答：制表符、空格和换行符（被称为空白符）对程序没有影响，虽然明智地使用空白可以提高程序的可读性。

问：负数是真还是假？

答：所有非零值（无论正负）均为真。

4.18 作 业

作业包括测验和练习，前者帮助加深读者对所学知识的理解，后者提供了使用新学知识的机会。请尽量先完成测验和练习题，然后再对照附录 D 中的答案，继续学习下一章之前，请务必弄懂这些答案。

4.18.1 测验

1. 什么是表达式？
2. x=5+7 是表达式吗？它的值是多少？
3. 201/4 的值是多少？
4. 201%4 的值是多少？
5. 如果 myAge、a 和 b 都是 int 变量，则执行下列代码后，它们的值是多少？

```
myAge = 39;
a = myAge++;
b = ++myAge;
```

6. 8+2*3 的值是多少？
7. if（x = 3）和 if（x == 3）有何不同？
8. 下列值为 true 还是 false？

   a. 0

b. 1

c. -1

d. x = 0

e. x == 0 //assume that x has the value of 0

4.18.2　练习

1. 编写一条 if 语句，检查两个整型变量，并将较大的变成较小的，只能使用一条 else 子句。

2. 查看下列程序。假定输入的 3 个数，并写出预期的输出。

```
1:  #include <iostream>
2:  using namespace std;
3:  int main()
4:  {
5:      int a, b, c;
6:      cout << "Please enter three numbers\n";
7:      cout << "a: ";
8:      cin >> a;
9:      cout << "\nb: ";
10:     cin >> b;
11:     cout << "\nc: ";
12:     cin >> c;
13:
14:     if (c = (a-b))
15:         cout << "a: " << a << " minus b: " << b <<
16:                         " equals c: " << c;
17:     else
18:         cout << "a-b does not equal c: ";
19:   return 0;
20: }
```

3. 输入练习 2 中的程序，然后编译、链接并运行它。输入 20、10 和 50。得到了预期的输出吗？为什么没有？

4. 查看下面的程序并预测输出。

```
1:    #include <iostream>
2:    using namespace std;
3:    int main()
4:    {
5:        int a = 2, b = 2, c;
6:        if (c = (a-b))
7:            cout << "The value of c is: " << c;
8:        return 0;
9:    }
```

5. 输入、编译、链接并运行练习 4 中的程序。输出是什么？为什么？