第 2 章　C++基础知识

> 别以为你知道计算机终端是个什么东西。计算机终端可不是什么乏味的旧电视，前头再摆个打字机。它是一种接口，使身体和心灵可以和宇宙相连接，并且把其中的一些东西移来移去。
>
> ——*道格拉斯·亚当斯，《银河系漫游指南》第五卷·基本无害*

概述

本章将解释更多的 C++示范程序，展示 C++语言足够多的细节，便于你写出简单的 C++程序。

预备知识

第 1 章简单介绍了一个 C++示范程序，本章将使用那个程序(如果还没有阅读对那个程序的描述，请在继续后面的学习之前阅读它，这对你很有帮助)。

2.1 变量和赋值

> 一旦理解变量在编程中的用法，就可以说理解了编程的精髓。
>
> ——*艾兹格·戴克斯特拉，“结构化编程”课堂笔记*

程序要处理数字和字母之类的数据。C++和其他常用编程语言一样，使用名为**变量**的编程构造来命名和存储数据。变量是编程语言(如 C++)的核心，所以要从变量开始介绍 C++。下面将围绕图 2.1 的程序展开讨论，并解释该程序中的所有元素。虽然此程序的常规思路应该是很清楚的，但某些细节是新的，需要进行一些解释。

变量

C++变量可容纳一个数字或其他类型的数据。目前只关心用于存储数字的变量。这些变量类似于可在上面写数字的小黑板。黑板上写的数字可以更改，C++变量容纳的数字也可以。不过，黑板可能不包含任何数字，但变量肯定包含了某个值——可能是以前运行过的程序在内存中留下的垃圾数字。变量容纳的数字或其他类型的数据称为这个变量的**值**。也就是说，变量值是在一个假想的黑板上写下的内容。图 2.1 中，`numberOfBars`，`oneWeight` 和 `totalWeight` 是变量。例如，如果运行程序并提供示范对话中的输入，`numberOfBars` 的值会被设为 `11`，这是通过以下语句来实现的：

```
cin >> numberOfBars;
```

之后，当程序执行以上语句的另一个拷贝时，变量 `numberOfBars` 的值会被更改为 `12`。稍后会详细解释这是怎样发生的。

图 2.1　一个 C++程序

```
#include <iostream>
using namespace std;
int main()
{
    int numberOfBars;    // 糖的数量
    double oneWeight, totalWeight; // 每块糖的重量和总的重量

    cout << "Enter the number of candy bars in a package\n"; // 输入一包糖中有多少块糖
    cout << "and the weight in ounces of one candy bar.\n"; // 每块糖的重量(盎司)
    cout << "Then press return.\n"; // 输入后按 Enter 键
    cin >> numberOfBars;
    cin >> oneWeight;

    totalWeight = oneWeight * numberOfBars;

    cout << numberOfBars << " candy bars\n";
    cout << oneWeight << " ounces each\n";
    cout << "Total weight is " << totalWeight << " ounces.\n";

    cout << "Try another brand.\n";
    cout << "Enter the number of candy bars in a package\n";
    cout << "and the weight in ounces of one candy bar.\n";
    cout << "Then press return.\n";
    cin >> numberOfBars;
    cin >> oneWeight;

    totalWeight = oneWeight * numberOfBars;

    cout << numberOfBars << " candy bars\n";
    cout << oneWeight << " ounces each\n";
    cout << "Total weight is " << totalWeight << " ounces.\n";

    cout << "Perhaps an apple would be healthier.\n";

    return 0;
}
```

示范对话

```
Enter the number of candy bars in a package and the weight in
ounces of one candy bar.
Then press return.
11 2.1
11 candy bars
2.1 ounces each
Total weight is 23.1 ounces.
Try another brand.
Enter the number of candy bars in a package and the weight in
ounces of one candy bar.
Then press return.
12 1.8
12 candy bars
1.8 ounces each
Total weight is 21.6 ounces.
Perhaps an apple would be healthier.
```

当然，变量不是黑板。在编程语言中，变量作为内存位置来实现。编译器将内存位置(参见第 1 章的讨论)分配给程序中的每个变量名。0 和 1 形式的变量值存储在为变量分配的内存位置。例如，在图 2.1 的程序中，为这三个变量分配的内存位置可能分别是 1001，1003 和 1007。具体编号取决于计算机、编译器和其他很多因素。我们不知道、也不关心编译器为变量选择什么内存地址。可以简单地认为内存位置就是用变量名作为标签。

程序无法运行

如果无法编译和运行 C++程序，请阅读 1.3.8 节了解如何应对不同的 C++编译器和 C++环境。

名称：标识符

在示范程序中，首先注意到的可能是变量名比平时在数学课上使用的变量名长。为了使程序容易理解，务必为变量使用有意义的名称。变量(或者在程序中定义的其他项目)的名称叫**标识符**。标识符以字母或下划线开头，其余字符必须是字母、数字或下划线。例如，下面的标识符是有效的：

```
x x1 x_1 _abc ABC123z7 sum RATE count data2 Big_Bonus
```

编译器会接受这些标识符，但前 5 个不太理想，它们没有表达出用途。以下标识符则是无效的，编译器拒绝接受：

```
12 3x %change data-1 myfirst.c PROG.CPP
```

前 3 个之所以无效，是因为不是以字母或下划线开头。其余 3 个之所以无效，是因为包含了除字母、数字和下划线之外的其他符号。

C++是**对大小写敏感**的语言。也就是说，它会区别对待标识符中的大写和小写字母。因此，以下 3 个标识符是不同的标识符，可命名 3 个不同的变量：

```
rate RATE Rate
```

但最好不要在同一个程序中使用这样的变体，因为它们太容易混淆。虽然 C++没有专门要求，但变量名最好全部小写。预定义标识符(比如 `main`，`cin` 和 `cout` 等)则必须全部小写。本章后面会讲到一些采用大写字母的标识符。

C++标识符长度没有限制，但有的编译器设置了最大允许长度，超出的会被忽略。

标 识 符

标识符用于命名 C++程序中的变量和其他元素。标识符必须以字母或下划线开头，后续每个字符只能是字母、数字或下划线。

还有一类特殊标识符称为**关键字**或**保留字**，它们在 C++中有预定义的含义，不能用作变量或其他元素的名称。附录 1 列出了所有 C++关键字。

你可能会问，为什么定义成 C++语言一部分的其他单词未被归为关键字？`cin` 和 `cout` 等单词为什么不是关键字？原来，程序员可以重新定义这些单词(虽然这样容易使人混淆)。但这些预定义的单词确实不是关键字，它们是在 C++语言标准要求的库中定义的。本书后面会讨论库，目前暂时不必关心库的问题。毫无疑问，为预定义标识符赋予非标准的含义，肯定会产生误导，而且很危险，所以应尽量避免。最安全、最简单的做法是将所有预定义标识符也视为关键字。

变量声明

C++程序中的每个变量都必须声明。**声明**变量实际是告诉编译器(最终是告诉计算机)：准备在该变量中存储什么类型的数据。例如，图 2.1 用两个语句声明了三个变量：

```
int numberOfBars;
double oneWeight, totalWeight;
```

在一个语句中声明多个变量要用逗号分隔不同的变量。另外，每个语句以分号结尾。

上面两个声明语句中，第一个语句中的 int 是 integer(整数)一词的缩写(但在 C++程序里，必须使用缩写形式 int，千万不能写全称)。这行代码将标识符 numberOfBars 声明为 int 类型的变量。这表示 numberOfBars 的值必须是整数，比如 1，2，-1，0，37 或-288。

第二个语句中的 double 将两个标识符 oneWeight 和 totalWeight 声明为 double 类型的变量。double 类型的变量可存储带小数部分的值，如 1.75 或-0.55。变量能容纳的数据的种类称为这个变量的**类型**，类型的名称(如 int 或 double)称为**类型名称**。

变 量 声 明

所有变量必须在使用之前声明。变量声明语法如下：

语法

Type_Name VariableName1, *VariableName2*, …;

示例

```
int count, numberOfDragons, numberOfTrolls;
double distance;
```

命 名 规 范

命名规范是指标识符(比如变量名)的一套命名规则。本书变量名以小写字母开头。如变量名由多个单词复合，之后各单词以大写字母开头。这个规范称为 camelCase。C 风格的规范要求各单词以下划线连接。还有的规范要求在变量名中指定变量类型。

C++程序的每个变量都必须在使用前声明。有两个非常自然的位置可供声明变量：刚好在使用变量之前，或者在程序 main 部分的起始处，也就是在以下两行代码之后：

```
int main()
{
```

总之，选择能使程序变得更清晰的位置。

变量声明提供了编译器实现变量所需的信息。前面说过，编译器将变量实现为内存位置，而变量的值保存在为这个变量分配的内存位置中。变量的值被编码为一连串 0 和 1。不同类型的变量需要不同大小的内存位置，还需要用不同的编码方式把它们的值编码为一连串 0 和 1。计算机用一种方式将整数编码为一连串 0 和 1，用另一种方式编码带有小数部分的数字，再用另一种方式将字母编码为一连串 0 和 1。变量声明实际是告诉编译器(最

终是告诉计算机)两点：第一，应该为一个变量分配多大的内存位置；第二，用哪种编码方式将变量的值表示为一连串 0 和 1。

语法和语义

编程语言(或其他任何语言)的**语法**(syntax)指该语言的一套语法规则。例如，谈论变量声明的语法时(参见上一个小结框“变量声明”)，实际是说为了写一个具有良好格式的变量声明，需要遵循哪些规则。遵循了 C++的所有语法规则，编译器就肯定接受你的程序。当然，这只能保证程序有效。虽然能保证程序能做某事，但不能保证它做的是你真正想做的事情。

语义(semantics)是指程序运行时所做事情的含义。程序的语法和语义都应正确。

赋值语句

更改变量值最直接的方式就是使用赋值语句。**赋值语句**是表示“将变量设为指定值”的计算机指令。以下赋值语句摘自图 2.1 的程序：

```
totalWeight = oneWeight * numberOfBars;
```

它要求计算机将 totalWeight 的值设为两个变量(oneWeight 和 numberOfBars)中的数字的乘积(第 1 章讲过，*是 C++中的乘号)。

赋值语句总是由等号左侧的变量和等号右侧的表达式组成。赋值语句以分号结尾。等号右侧的表达式可以是变量、数字或者由变量、数字和算术操作符(如*和+)构成的较复杂的表达式。赋值语句指示计算机对等号右侧的表达式进行求值(计算这个表达式的值)，并把等号左侧的变量的值设为求值的结果。多研究一些例子有助于掌握赋值语句的工作原理。

可用任何算术操作符代替乘号(*)。例如，以下语句也是有效的赋值语句：

```
totalWeight = oneWeight + numberOfBars;
```

该赋值语句与示范程序的赋值语句一样，区别在于执行加法而非乘法。它将 totalWeight 的值设为两个变量值(oneWeight 和 numberOfBars)之和。当然，如果在图 2.1 中进行这样的改动，程序虽会运行，但会给出不正确的输出。

赋值语句的等号右侧的表达式可以是另一个变量。以下语句使变量 totalWeight 的值变得和变量 oneWeight 一样：

```
totalWeight = oneWeight;
```

在图 2.1 的程序中进行上述修改，程序给出的一包糖的重量将错误地小于实际值(假定一包糖内不止一块糖)。但在其他程序中，这样赋值可能是有意义的。

作为另一个例子，以下赋值语句将 numberOfBars 的值变成 37：

```
numberOfBars = 37;
```

代码中的实际数字(如本例的 37)称为**常量**。和变量不同，常量的值不能改变。

由于变量值可以随时间更改，同时赋值操作符是更改其值的手段，所以赋值语句的含义必须考虑到时间因素。在时间上，首先求值的是等号右侧的表达式。然后，等号左侧的

变量的值被更改为那个表达式的求值结果。这就意味着一个变量可以同时出现在赋值操作符的两侧。例如以下赋值语句：

```
numberOfBars = numberOfBars + 3;
```

初学者可能不解其意。如果像普通的句子那样读，就是“numberOfBars 等于 numberOfBars 加 3。”但实际含义是“让 numberOfBars 的新值等于 numberOfBars 的旧值加 3。”在 C++中，等号的用法与日常语言或数学中的等号不同。

赋 值 语 句

赋值语句中，等号右侧的表达式先求值，该值赋给等号左侧的变量。

语法

```
Variable = Expression;
```

示例

```
distance = rate * time;
count = count + 2;
```

陷阱：未初始化的变量

除非程序为变量赋值，否则变量不包含有意义的值。例如，如果变量 minimumNumber 既没有放在一个赋值语句的左侧而被赋值，也没有通过其他手段来赋值(比如用 cin 语句赋一个由用户输入的值)，以下语句就是错误的：

```
desiredNumber = minimumNumber + 10;
```

这是由于 minimumNumber 不包含有意义的值，所以等号右侧的整个表达式也不会产生有意义的值。如变量(比如 minimumNumber)未被赋值，就说它**未初始化**。事实上，这比 minimumNumber 不包含任何值还要糟。未初始化的变量(比如 minimumNumber)将包含“垃圾值”。未初始化的变量的值由留在其内存位置中的 0,1 序列来决定(可能是由用过该内存位置的上一个程序留下的)。因此，如果程序运行两次，未初始化的变量每次都可能获得不同的值。只要程序为完全相同的输入数据产生了两个不同的输出，而且程序本身未进行任何修改，就应该怀疑其中含有未初始化的变量。

为了避免出现未初始化的变量，一个办法是在声明变量的同时初始化。这可通过添加一个等号和一个值来完成，如下所示：

```
int minimumNumber = 3;
```

它除了将 minimumNumber 声明为 int 类型的变量，还将 minimumNumber 的值设为 3。像这样在同一行声明并初始化时，可以使用涉及运算(比如加法或乘法运算)的一个更复杂的表达式。然而，最常见的还是使用简单的常量来初始化。可在一个声明内列出多个变量，同时初始化其中的部分或全部变量，或者一个都不初始化。例如，以下语句声明了三个变量，但只初始化其中两个：

```
double rate = 0.07, time, balance = 0.0;
```

C++允许采用另一种方式在声明变量的同时初始化它。以下语句等价于刚才的声明：

```
double rate(0.07), time, balance(0.0);
```

具体是在声明的同时初始化，还是以后再初始化，要取决于实际情况。你选择的方式应该使程序更容易阅读。 ■

声明时初始化变量

可在声明变量的同时初始化它(向它赋值)。

语法

Type _Name variableName1 = Expression _for_Value_1,
variableName2 = Expression _for_Value_2, . . .;

示例

```
int count = 0, limit = 10, fudgeFactor = 2;
double distance = 999.99;
```

声明变量同时初始化的另一种方式：

Type_Name variableName1(*Expression_for_Value_1*),
variableName2(*Expression_for_Value_2*), . . .;

示例

```
int count(0), limit(10), fudgeFactor(2);
double distance(999.99);
```

编程提示：使用有意义的名称

变量名和其他名称至少应该让人联想到它们所命名的事物的含义或用途。有意义的变量名可以使程序更容易理解。所以，以下语句：

```
x = y * z;
```

最好更改为：

```
distance = speed * time;
```

两个语句作用一样，但第二个更容易理解。

√ 自测题

1. 给出变量 feet 和变量 inches 的声明。两个变量都是 int 类型，都在声明中初始化为 0。请同时使用前面介绍的两种初始化方式。
2. 给出变量 count 和变量 distance 的声明。count 是 int 类型，初始化为 0；distance 是 double 类型，初始化为 1.5。
3. 给出 C++语句，将变量 sum 的值更改为变量 n1 和 n2 的值之和。所有变量都是 int 类型。
4. 给出 C++语句，使变量 length 的值增大 8.3。变量 length 已经声明为 double 类型。
5. 给出 C++语句，将变量 product 的值更改为它的旧值乘以变量 n 的值。这些变量都是 int 类型。
6. 写程序来输出虽已声明但尚未初始化的五六个变量的值。编译并运行。会输出什么？请说明原因。

7. 为以下每个变量提供有意义的名称：
   a. 一个用于保存车速的变量；
   b. 一个用于保存小时工每小时薪酬的变量；
   c. 一个用于保存一次考试的最高分的变量。

2.2　输入和输出

垃圾入，垃圾出。

——程序员的口头禅

C++程序可采取几种不同的方式执行输入和输出。我们描述其中的一种，即“流”。**输入流**是提供给计算机并由程序使用的一系列输入。“流”这个词表明程序将以相同的方式处理所有输入，无论这些输入是从什么地方来的。换言之，程序看到的只是输入流本身，看不到流的来源。好比山间一条小溪，溪水在你面前流过，而你不知道它来自哪里。本节假设输入来自键盘。第 6 章将讨论程序如何从文件读取输入。到时就会知道，从键盘读取输入的语句也可用于从文件读取。类似地，**输出流**是程序生成的一系列输出。本节假设输出到终端屏幕(第 6 章将讨论如何输出到文件)。

使用 cout 进行输出

可以使用 cout 将变量值和文本字符串输出到屏幕。变量和字符串可组合输出。以图 2.1 的下面这行代码为例：

```
cout << numberOfBars << " candy bars\n";
```

它要求计算机输出两项内容：变量 numberOfBars 的值和用引号封闭的字符串" candy bars\n"。注意，不需要为每个输出项都单独使用单词 cout。可以列出所有输出项，在每一项之前附加箭头符号<<。上述 cout 语句等价于以下两个 cout 语句：

```
cout << numberOfBars;
cout << " candy bars\n";
```

cout 语句可以包括算术表达式，如下例所示，其中的 price 和 tax 是变量：

```
cout << "The total cost is $" << (price + tax);
```

用于封闭算术表达式(如 price + tax)的圆括号是一些编译器要求的，最好不要遗漏。

两个<(小于)符号应连续输入，中间不要有空格。箭头符号<<通常称为**插入操作符**。整个 cout 语句以分号结尾。

只要连续出现了两个 cout 语句，就可以将其合并成一个较长的 cout 语句。以图 2.1 的以下两行代码为例：

```
cout << numberOfBars << " candy bars\n";
cout << oneWeight << " ounces each\n";
```

这两个语句可改写为如下所示的一个语句，程序执行结果一样：

```
cout << numberOfBars << " candy bars\n" << oneWeight << " ounces each\n";
```

为了防止代码行超出屏幕边界，可将较长的 cout 语句分解为两行或更多的行。一种较好的做法是将上述较长的 cout 语句改写成如下形式：

```
cout << numberOfBars << " candy bars\n"
     << oneWeight << " ounces each\n";
```

不要在引号字符串中断行。但是，凡是能插入空格的地方都能另起一行。计算机接受任何合理的间隔与换行风格，但上例和本书的其他示范程序是你应该学习的“好榜样”。一个较好的策略是，针对直观上可视为一个整体的每组输出都使用一个 cout。注意，每个 cout 都只对应一个分号，即使一个 cout 语句被拆分成若干行。

在图 2.1 的程序中，特别注意要输出的引号字符串。注意，要输出的字符串必须包含在一对双引号内。每个双引号都是一个单独的字符(按一次键即可输入)，不要连续键入两个单引号来取代它。还要注意，字符串两端使用的是同一个双引号，没有独立的左引号和右引号。

还要注意引号内的空格。计算机不会在 cout 输出的内容前后自动添加空格。因此，示例中的引号字符串通常都会以一个空格开始和/或结束。空格防止不同的字符串和数字紧挨在一起。如果只想输出空格，就使用只包含空格的字符串。如下所示：

```
cout << firstNumber << " " << secondNumber;
```

如第 1 章所述，\n 告诉计算机从一个新行输出。除非告诉计算机换行，否则它会将所有输出放到同一行。取决于计算机屏幕设置，这可能导致输出时任意换行[①]，或者一行的剩余内容跑到屏幕外面。注意\n 是在引号内部使用的。在 C++中，“换行”被视为一个特殊字符(特殊符号)。在引号字符串内，这个特殊字符要拼写成\n。注意，符号\和 n 之间没有空格。虽然这个特殊字符在输入时要使用两个符号，但 C++将\n 视为一个字符，并称之为**换行符**。

include 预编译指令和命名空间

我们所有程序都以下面这两行代码开始：

```
#include <iostream>
using namespace std;
```

这两行使 iostream 库进入可用状态。cin 和 cout 的定义就包含在这个库中。所以，假如程序使用了 cin 或 cout，就应在程序文件起始位置包含上述两行代码。

下面这一行代码称为 include **预编译指令**。它将 iostream 库“包含”到程序中，使程序能使用 cin 和 cout：

```
#include <iostream>
```

cin 和 cout 在一个名为 iostream 的文件中定义，以上 include 指令相当于将那个文件复制到程序中。第二行代码比较复杂，三言两语很难说清。

C++使用**命名空间**组织名称。命名空间是很多名称(比如 cin 和 cout)的集合。通过以下方式指定命名空间的语句称为 using **预编译指令**：

① 例如，单词“word”的“ord”部分可能跑到下一行。——译注

```
using namespace std;
```

这个特定的 using 指令表明程序准备使用 std(指 standard)命名空间。这意味着你使用的名称具有 std 命名空间为其定义的含义。这儿的重点在于，cin 和 cout 等名称在 iostream 中定义时，它们的定义指出它们在 std 命名空间中。所以，要使用 cin 和 cout 等名称，就要告诉编译器你准备“using namespace std;”。

对于命名空间并不需要了解太多(就目前而言)，但简单澄清一下，有助于解除你在使用命名空间时可能产生的一些困惑。C++之所以有命名空间，是因为有太多的东西需要命名。结果就是可能有两个或更多的项同名。换句话说，一个名称可能具有两个不同的定义。为消除歧义，C++将不同的项划分到不同的集合中，确保同一个集合(即同一个命名空间)中没有任何两个项同名。

注意，命名空间并不只是一个名称集合。它代表了一个 C++代码主体，其中指定了某些名称的含义(比如一些定义和/或声明)。命名空间的作用是将所有 C++名称规范划分成不同的集合(称为命名空间)，使命名空间内的每个名称在那个命名空间中都只有一个“规范”(一个定义)。命名空间对名称进行划分，但和那些名称配合的还有大量 C++代码。

可以使用两个命名空间中的两个同名元素吗？答案是肯定的，而且并不复杂，但那是本书后面要讲的一个主题。目前不需要这样做。

有的C++版本使用 include 预编译指令的一种古老形式(没有任何 using namespace)：

```
#include <iostream.h>
```

使用以下语句时，如果程序无法编译或者不能运行：

```
#include <iostream>
using namespace std;
```

请尝试将上述两行代码统一更换为如下形式：

```
#include <iostream.h>
```

但如果程序要求使用 iostream.h(而不是 iostream)，则表明你使用的是老版本的C++编译器。升级一下吧。

转义序列

字符前的符号\告诉编译器：\之后的字符具有特殊含义，不能沿用其字面含义。这样的一个字符序列称为**转义序列**。转义序列肯定由两个字符构成，而且两个字符之间没有空格。C++定义了几个转义序列。

如果希望在一个字符串常量中插入反斜杠\或插入双引号"，则必须使用\"来转变"的原有功能(结束一个字符串常量)，或者使用\\来转变\的原有功能(转义)。\\向编译器表明你需要一个真正的反斜杠\，而不是一个转义序列。\"表明需要一个真正的双引号，而不是结束一个字符串常量。

如果在字符串常量中出现未定义的转义序列(比如\z)，有的编译器会返回一个 z，有的则会报错。ANSI 标准规定，对于未定义的转义序列，其行为是“未定义”的。所以，编译器的设计者可采用自己觉得方便的任何方式处理它。其后果是，如果在代码中使用了未定义的转义序列，就失去了“可移植性”。因此，不要使用任何未定义的转义序列。下面列

出了 C++定义的部分转义序列：

- 换行符 \n
- 水平制表符 \t
- 响铃符 \a
- 反斜杠 \\
- 双引号 \"

此外，C++11 支持所谓的**原始字符串字面值**(raw string literals)，它适合有太多字符需要转义的情况。该格式要求字符串以 R 开头，而且字符串内容要放到一对圆括号中。例如，以下代码输出字符串字面值"c:\files\"。

```
cout << R"(c:\files\)";
```

要在输出中插入空行，可单独输出一个\n 换行符，如下所示：

```
cout << "\n";
```

输出空行的另一种方式是使用 endl，含义和"\n"相同，所以还能像下面这样输出空行：

```
cout << endl;
```

虽然"\n"和 endl 含义相同，但用法稍有区别：\n 必须放到双引号内，endl 则不能。

\n 和 endl 应该如何选择呢？一个较好的依据是：如果\n 可以放到一个较长的字符串的末尾，就像下面这样使用\n：

```
cout << "Fuel efficiency is "
     << mpg << " miles per gallon\n";
```

而如果需要单独使用一个"\n"，就改为使用 endl，如下所示：

```
cout << "You entered " << number << endl;
```

在输出中开始新行

要在输出中换行，可将\n 包含到引号字符串内，如下所示：

```
cout << "You have definitely won\n"
<< "one of the following prizes:\n";
```

记住，\n 要作为两个字符来输入，两个字符之间不能有空格。

另一种方法是通过 endl 来换行，如下所示：

```
cout << "You have definitely won" << endl
<< "one of the following prizes:" << endl;
```

编程提示：用\n 或 endl 终止每一个程序

最好在每个程序末尾都输出一个换行指令。如果程序输出的最后一项是字符串，就在该字符串的末尾添加一个\n；如果不是，就将输出 endl 作为程序的最后一个动作。这样做有两个目的：有的编译器不输出程序中的最后一行，除非在末尾包含一个换行指令。在另一些系统上，即使没有这个最后的换行指令，程序也能正常运行，但运行下一个程序时，它会将自己的第一行输出与上一个程序的最后一行输出混在一起。即使这两个问题在你的

系统上都不存在，在程序末尾添加一个换行指令，起码也能使程序具有更强的移植性。 ■

格式化带小数点的数字

计算机输出 double 类型的值时，格式可能与你预期的不同。例如，下面这个简单的 cout 语句就会产生很多种不同的输出：

```
cout << "The price is $" << price << endl;
```

如果 price 的值是 78.5，输出结果可能如下：

```
The price is $78.500000
```

或者如下：

```
The price is $78.5
```

甚至可能是以下形式(详情参见 2.3 节)：

```
The price is $7.850000e01
```

以下输出恐怕是最不可能出现的，但这种格式才是最有意义的：

```
The price is $78.50
```

为保证得到预期输出，要在程序中包含一些指令告诉计算机如何输出数字。

可以在程序中插入一个“魔法配方”，指定含有小数点的数字(比如 double 类型的数字)以日常生活中习惯的方式来输出。换言之，可以确切地指定小数位数。例如，为了显示两位小数，需要以下“魔法配方”：

```
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
```

在程序中插入上述 3 个语句，后续的任何 cout 语句都会按你指定的格式输出 double 类型的值。换言之，小数点之后刚好 2 位。例如，假设以下 cout 语句出现在上述“魔法配方”之后的某个地方，并假设 price 的值为 78.5：

```
cout << "The price is $" << price << endl;
```

则输出结果肯定如下：

```
The price is $78.50
```

可用任何非负的整数代替“魔法配方”中的 2，从而指定不同的小数位数。甚至能用 int 类型的变量代替 2。

这个“魔法配方”将在第 6 章进一步解释。现在只需将其视为一长串指令，目的是告诉计算机你想以什么格式输出含小数点的数字。

要更改小数位数，使程序中不同的值输出不同的小数位数，可以重复使用这个“魔法配方”，用其他数字代替 2。但是，重复这些“魔法配方”时，只需重复其中的最后一行。假如该“魔法配方”已在程序中出现过一次，那么只需添加下面这一行，即可让后续的所有 double 类型的变量输出 5 位小数：

```
cout.precision(5);
```

输出 double 类型的值

在程序中插入以下“魔法配方”，后续所有 double 类型(或允许保留小数的其他类型)的数字将以日常生活中最习惯的方式来输出，即保留两位小数：

```
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
```

可用其他任何非负的整数代替其中的 2，从而指定不同的小数位数。甚至能用 int 类型的变量代替 2。

用 cin 进行输入

使用 cin 输入时，方式与使用 cout 输出差不多。两者语法相似，区别在于 cin 代替了 cout，而且箭头方向相反。例如，在图 2.1 的程序中，numberOfBars 和 oneWeight 变量由以下 cin 语句来填充(同时给出了 cout 语句，告诉用户应该如何操作)：

```
cout << "Enter the number of candy bars in a package\n";
cout << "and the weight in ounces of one candy bar.\n";
cout << "Then press return.\n";
cin >> numberOfBars;
cin >> oneWeight;
```

可以在一个 cin 语句中列出多个变量。所以，以上代码可以重写为以下形式：

```
cout << "Enter the number of candy bars in a package\n";
cout << "and the weight in ounces of one candy bar.\n";
cout << "Then press return.\n";
cin >> numberOfBars >> oneWeight;
```

如果愿意，还可将上述 cin 语句拆成以下两行代码：

```
cin >> numberOfBars
    >> oneWeight;
```

注意，和 cout 语句一样，每个 cin 只对应一个分号。

程序抵达 cin 语句时，它会等待用户从键盘输入。它将第一个变量设为从键盘输入的第一个值，第二个变量设为从键盘输入的第二个值，依此类推。但除非按 Enter 键，否则程序不会真正读取输入。利用这个设计，可在输入一行内容时按 Backspace 键纠错。

输入的各个数字必须以一个或多个空格或者以一个换行符来分隔。例如，假定要输入两个数字 12 和 5，但输入时没有用空格分隔它们，计算机就认为输入的是一个数字，即 125。使用 cin 语句时，计算机会跳过任意数量的空格或换行符，径直找到下一个输入值。因此，无论你用一个还是多个空格，甚至用一个换行符来分隔，都是无关紧要的。

cin 语句

cin 语句将变量设为从键盘键入的值。

语法

```
cin >> Variable_1 >> Variable_2 >>…;
```

示例

```
cin >> number >> size;
cin >> timeToGo
    >> pointsNeeded;
```

设计输入和输出

输入和输出通常称为 I/O，是程序中用户看得见的部分。I/O 设计不佳，就无法取悦于用户。

计算机执行 cin 语句时，它希望收到从键盘输入的数据。用户不输入数据，计算机会一直等下去。程序必须提示用户什么时候应该输入数字(或其他数据项)。计算机不会自动要求输入，所以示范程序包含了以下输出语句：

```
cout << "Enter the number of candy bars in a package\n";
cout << "and the weight in ounces of one candy bar.\n";
cout << "Then press return.\n";
```

这些输出语句**提示**用户输入。程序希望获取输入时一定要明确提示。

从终端输入数据时，输入的内容会在屏幕光标位置出现。虽然如此，始终应该在程序终止之前，主动显示输入的值。这称为**回显输入**，作用是让用户检查输入是否被正确读取。之所以要回显，是因为有时在屏幕上看起来没有问题的输入不一定能被计算机正确读取。例如，其中可能含有被忽视的打字错误或其他问题。通过回显输入，能对输入数据的完整性进行测试。

编程提示：I/O 中的换行

可在同一行输出和输入，而且对用户而言，这有时能产生更友好的界面。如果在最后一个提示行末尾省略\n 或 endl，用户的输入就会在这个提示行上出现。例如，假设使用以下提示和输入语句：

```
cout << "Enter the cost per person: $";
cin >> costPerPerson;
```

程序执行 cout 语句时，屏幕上将出现以下提示：

```
Enter the cost per person: $
```

用户输入的数据将在同一行上出现，如下所示：

```
Enter the cost per person: $1.25
```

■

√ 自测题

8. 写一个输出语句在屏幕上生成以下消息[①]：

```
The answer to the question of
Life, the Universe, and Everything is 42.
```

9. 写一个输入语句，使用从键盘输入的值来填充 int 类型的变量 theNumber。在输入语句前添加一个提示语句，提示用户输入一个整数。

① “那个伟大问题的答案……关于生命、宇宙以及一切……是……42。”这句话来自道格拉斯•亚当斯的科幻名著《银河系漫游指南》。——译注

10. 输出 double 类型的数字时，在程序中添加什么语句能确保这种类型的值以正常格式输出，并具有 3 个小数位。

11. 写一个完整的 C++程序，要求在屏幕上显示 Hello world。不要求该程序具备其他功能。

12. 写一个完整的 C++程序，要求读入两个整数，并输出两个整数之和。一定要提示输入，回显输入，并清楚说明输出的是什么。

13. 写一个输出语句来输出一个换行符和一个制表符。

14. 写一个小程序，声明 double 类型的变量 one，two，three，four 和 five，分别初始化为 1.000，1.414，1.732，2.000 和 2.236。写一些输出语句来生成以下表格。使用制表符转义序列\t 来对齐不同的列。如果还不熟悉制表符，请在做这道题的时候试验它。制表符相当于打字机上的一个机械跳格位置，会导致输出从下一列(栏)开始，两列间距通常是 8 个空格的倍数。许多编辑器和大多数字处理程序都允许调整制表位。我们的输出不调整。

    输出应该如下所示：

```
N  Square Root
1  1.000
2  1.414
3  1.732
4  2.000
5  2.236
```

2.3 数据类型和表达式

他俩没戏。他不是她喜欢的类型。

——鸡尾酒会上的闲言碎语

int 类型和 double 类型

2 和 2.0 在概念上固然是同一个数字，但 C++将它们视为不同类型。2 是 int 类型，2.0 是 double 类型，因为后者包含了小数部分(即使小数部分为 0)。再次强调，计算机编程所涉及的数学概念与数学课上学到的略有区别。计算机的一些实际问题造成计算机中的数字有别于这些数字的抽象定义。C++中，整数的行为和你预期的一样。int 类型保存的就是整数。但 double 类型要麻烦一些。由于 double 类型只能存储有限位数的数字，所以计算机将 double 类型的值保存为近似值。相反，int 类型的值则作为精确值来保存。double 值的具体精度在不同计算机上是不同的。但无论如何，double 值的精度至少能达到 14 位。这个精度对于大多数应用程序已经足够。不过，即使在一些简单的情况下，它也有可能造成不易察觉的问题。因此，如果已知某个变量保存的值是整数，而且大小在计算机能接受的范围之内，就最好把它声明为 int。

double 数字常量在书写时有别于 int 数字常量。int 常量绝对不能包含小数点。double 常量则有两种写法。对于 double 常量，最简单的形式就是日常生活中的写法(带小数)。采用这种写法，double 常量必须包含小数点。double 和 int 常量有一个共同点：C++的任何数字都不能包含逗号。

double 类型的常量的一种更复杂的写法称为**科学记数法**或**浮点记数法**，特别适合表达

非常大的数和非常小的小数。例如，数字 3.67×10^{17} 相当于 367 000 000 000 000 000.0，在 C++中最好表示成常量 `3.67e17`。再如数字 5.89×10^{-6}，它相当于 0.000 005 89，在 C++中最好表示成常量 `5.89e-6`。`e` 代表 exponent(指数)。`e-6` 相当于“乘以 10 的-6 次方”。

之所以要使用 **e 记数法**，是因为通常无法直接通过键盘输入作为上标的指数。`e` 后面的数字指出了两点：第一，小数点的移动方向；第二，要移多少位。例如，要把 `3.49e4` 变成一个没有 `e` 的数字，需要让小数点右移 4 位，得到 `34900.0`，这是同一个数的另一种写法。如果 `e` 后面的数字为负，小数点就左移指定位数，必要时插入 `0`。所以，`3.49e-2` 相当于 `0.0349`。

`e` 之前的数字可包含小数点，但这并非必须。但 `e` 之后的指数绝对不能有小数点。

由于计算机内存有限，所以数字通常只能用数量有限的字节来存储(也就是说，只能占用有限的存储空间)。所以数字大小也是有限的，而且这一限制因数字的类型而异。`double` 类型的最大值肯定大于 `int` 类型的最大值。目前大多数 C++实现都规定 `int` 类型的值不得超过 2 147 483 647，而 `double` 类型的值不得超过 10^{308}。

double 缘起

为什么带小数的数是 `double` 类型？是否还有 `single` 类型，大小只有 `double` 的一半？答案是既错又对。过去，许多编程语言用两种类型表示带小数的数。一种类型使用内存较少，但不太精确(也就是说，它不允许非常多的有效位)。第二种类型占用的内存是第一种类型的两倍(double 由此而来)，所以精度更高。与此同时，它还支持更大的数(尽管程序员对精度的关心远胜于大小)。使用两倍内存的这类数字称为“双精度”数字，使用较少内存的则称为“单精度”数字。在 C++中，根据这一传统约定，与双精度类型对应的类型称为 `double` 类型，与单精度对应的则是 `float` 类型。C++还为带小数的数提供了第三种类型，即 `long double` 类型。这些类型将在下一小节描述。不过，本书没有任何场合需要使用 `float` 和 `long double` 类型。

其他数值类型

除了 `int` 和 `double`，C++还有其他数值类型，图 2.2 总结了其中一部分。

图 2.2　部分数值类型

类型名称	占用的内存	取值范围	精度
`short`(或 `short int`)	2 字节	−32 767～32 767	(不适用)
`int`	4 字节	−2 147 483 647～2 147 483 647	(不适用)
`long`(或 `long int`)	4 字节	−2 147 483 647～2 147 483 647	(不适用)
`float`	4 字节	约 10^{-38}～10^{38}	7 位
`double`	8 字节	约 10^{-308}～10^{308}	15 位
`long double`	10 字节	约 10^{-4932}～10^{4932}	19 位

<本表格仅供参考，目的是让你体会不同类型的区别。在具体每个人的系统上，情况可能不同。“精度”是指有效位数，其中包括小数点前的位。`float`，`double` 和 `long double` 类型显示的取值范围是正数范围。负数也有一个类似的范围，但要在每个数字之前添加负号>

不同数值类型允许不同取值范围，也允许或高或低的精度。图 2.2 给出的占用内存、取值范围和精度仅供参考，目的是让你体会各种类型的区别。这些指标因系统而异。在你的系统上，情况可能不同。

虽然有的类型要用两个单词来拼写，但在声明这些类型的变量时，与声明 int 及 double 类型的变量并无区别。例如，以下语句声明一个 long double 类型的变量：

```
long double bigNumber;
```

类型名称 long 和 long int 代表同一个类型。所以，以下两个声明是等价的：

```
long bigTotal;
long int bigTotal;
```

当然，程序只能在上述两个声明中选择一个来声明变量 bigTotal。但具体选择哪个无关紧要。还要记住，类型名称 long 等价于 long int，而不是等价于 long double。

用于表示整数的类型(比如 int 和其他类似的类型)统称为**整数类型**或者**整型**。用于表示带小数点的数字的类型(比如 double 和其他类似类型)则统称为**浮点类型**或者**浮点型**。之所以称为“浮点”，是因为计算机在存储像 392.123 这样正常写法的数字时，首先会将数字转换成与 e 记数法类似的一种形式(本例是 3.92123e2)。计算机执行这种转换时，小数点会“浮动”(也就是移动)到一个新位置。

虽然 C++支持其他数值类型，但本书只用到了 int 和 double，偶尔用一下 long。对于大多数简单应用程序，除了 int 和 double，其他任何类型都派不上用场。需要大整数的程序可考虑 long。

C++11 类型

整型的大小在不同计算机上可能不一样。例如，32 位机器的整数可能是 4 字节，而 64 位机器可能是 8 字节。如需确定一个整型的取值范围，这就可能成为问题。C++11 增加了新整型来解决问题。新整型指定了确切大小以及是否有符号。这些类型通过包含<cstdint>来使用。图 2.3 列出了部分类型。本书主要使用传统 int 和 long。但如果要指定确切大小，请考虑使用 C++11 类型。

图 2.3 部分 C++11 定宽整型

类型名称	占用的内存	取值范围
int8_t	1 字节	−128～127
uint8_t	1 字节	0～255
int16_t	2 字节	−32 768～32 767
uint16_t	2 字节	0～65 535
int32_t	4 字节	−2 147 483 648～2 147 483 647
uint32_t	4 字节	0~4 294 967 295
int64_t	8 字节	−9 223 372 036 854 775 808～9 223 372 036 854 775 807
uint64_t	8 字节	0～18 446 744 073 709 551 615
long long	至少 8 字节	

C++11 还提供 auto 类型，能根据等号右侧的表达式推断变量类型。例如，以下代码定义名为 x 的变量，它的数据类型由 expression 的求值结果决定。

```
auto x = expression;
```

该功能目前效果不彰，但以后定义自己的数据类型时可利用它简化代码。

C++11 还增加了判断变量或表达式类型的功能。decltype (expr)是变量或表达式 expr 的已声明类型，可在声明新变量时使用。例如：

```
int x = 10;
decltype (x*3.5) y;
```

上述代码声明 y 的类型和 x*3.5 一样。表达式 x*3.5 的结果是 double，所以 y 被声明为 double。


视频讲解：*C++11 Fixed Width Integer Types*

char 类型

我不想让你觉得计算机和 C++只会执行数值计算。接下来要介绍一种非数值类型。以后还会遇到其他更复杂的非数值类型。char(character 的缩写)类型的值表示单独一个符号，比如字母、数字或标点符号。在书籍和日常对话中，经常将这种类型的值称为**字符**。在 C++程序中，这种类型必须拼写成缩写形式 char。例如，char 类型的变量 symbol 和 letter 可以像下面这样声明：

```
char symbol, letter;
```

char 类型的变量可容纳键盘上的任何单独字符。所以，变量 symbol 可容纳一个'A'、一个'+'或者一个'a'。注意，字母大写和小写形式被视为不同字符。

使用 cout 输出的双引号中的文本则称为**字符串**值。例如，下面就是一个字符串，它来自图 2.1 的程序：

```
"Enter the number of candy bars in a package\n"
```

注意，字符串常量放到双引号内，而 char 类型的常量放到单引号内。两种引号含义不同。例如，'A'和"A"不同。'A'是 char 值，可存储到 char 类型的变量中。而"A"是字符串值。虽然这个字符串恰好只包含一个字符，但这一事实并不能使"A"成为 char。还要注意，无论字符串还是字符，左右两个引号是一样的(都是"或')。

图 2.4 演示如何使用 char 类型。注意在第一个和第二个姓名首字母之间输入了一个空格。但程序会跳过空格，将字母 B 作为输入的第二个字母来读取。通过 cin 将输入读入 char 类型的变量时，计算机会忽略所有空白间隔和换行符，直到遇到第一个非空白字符，并将其读入变量。输入中有无空白字符(比如空格、制表符等)对计算机没有区别。运行图 2.4 的程序，无论在两个姓名首字母之间插入一个空格(如示范对话所示)，还是连续输入两个首字母(不插入空格，即直接输入 JB)，结果都是一样的。

图 2.4　char 类型

```
1  #include <iostream>
2  using namespace std;
```

```
 3  int main()
 4  {
 5      char symbol1, symbol2, symbol3;

 6      cout << "Enter two initials, without any periods:\n"; // 输入两个姓名首字母，不要加点号
 7      cin >> symbol1 >> symbol2;
 8      cout << "The two initials are:\n";
 9      cout << symbol1 << symbol2 << endl;
10      cout << "Once more with a space:\n";
11      symbol3 = ' ';
12      cout << symbol1 << symbol3 << symbol2 << endl;
13      cout << "That's all.";
14      return 0;
15  }
```

示范对话

```
Enter two initials, without any periods:
J B
The two initials are:
JB
Once more with a space:
J B
That's all.
```

bool 类型

下个类型是 bool。该类型由 ISO/ANSI(国际标准化组织/美国国家标准化组织)于 1998 年增补到 C++语言中。bool 类型的表达式称为布尔(Boolean)表达式，这一名称源于英国数学家 George Boole(1815－1864)，他是数学逻辑规则的奠基人。

布尔表达式的求值结果只有两个：true 或 false。布尔表达式在分支和循环语句中使用。2.4 节更详细地讨论布尔表达式和 bool 类型。

string 类简介

虽然 C++缺乏原生数据类型来直接操作字符串，但在 string 类的帮助下，可采取和原生数据类型相似的方式处理字符串。类和原生数据类型的区别在第 10 章讨论。第 8 章更详细地讨论 string 类。

使用 string 类必须先包含 string 库：

```
#include <string>
```

程序还必须包含以下代码，通常放到文件起始处：

```
using namespace std;
```

声明 string 类型的变量和声明 int 或 double 类型的变量一样。例如，以下代码声明 string 类型的一个变量，并在变量中保存单词"Monday"：

```
string day;
day = "Monday";
```

如图 2.5 所示，可用 cin 将数据读入字符串。在两个字符串之间使用“+”符号可将两个字符串连接成一个更长的。例如以下代码：

```
string day, day1, day2;
day1 = "Monday";
```

```
day2 = "Tuesday";
day = day1 + day2;
```

会连接成以下字符串：

```
"MondayTuesday"
```

注意，连接两个字符串不会自动添加空格。这样的空格只能显式添加，如下所示：

```
day1 + " " + day2
```

用 cin 将输入的内容读入 string 变量时，除非遇到空白字符，否则计算机会一直读下去。空白字符是指在屏幕上所有显示为空白的字符，包括空格、制表符和换行符'\n'。这意味着不能输入含空格的字符串，否则就会出错。图 2.5 的示范对话 2 演示了一个例子。在本例中，用户试图输入"Mr. Bojangles"作为宠物名，但字符串只能读到"Mr."为止，因为下个字符是空格。"Bojangles"字符串被程序忽略，但假如有另一个 cin 语句，那么接下来就会读入它。第 8 章将介绍输入含空格字符串的一个技术。

图 2.5　string 类

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4  int main()
5  {
6      string middleName, petName;
7      string alterEgoName;
8      // ego 是拉丁文“我”的意思，alter ego 就是“另一个我”，或者说“知己”
9      cout << "Enter your middle name and the name of your pet.\n";
10     cin >> middleName;
11     cin >> petName;
12
13     alterEgoName = petName + " " + middleName;
14
15     cout << "The name of your alter ego is ";
16     cout << alterEgoName << "." << endl;
17
18     return 0;
19 }
```

示范对话 1

```
Enter your middle name and the name of your pet.
Parker Pippen
The name of your alter ego is Pippen Parker.
```

示范对话 2

```
Enter your middle name and the name of your pet.
Parker
Mr. Bojangles
The name of your alter ego is Mr. Parker.
```

类型的兼容性

一种类型的值一般不能存入另一种类型的变量。例如，大多数编译器对以下语句报错：

```
int intVariable;
intVariable = 2.99;
```

这是因为类型不匹配。常量 2.99 是 double 类型，变量 intVariable 是 int 类型。遗憾的是，并非所有编译器都以相同的方式响应上述赋值语句。有的显示错误消息，有的显示警告消息，有的则完全无动于衷。但即使编译器允许这种赋值语句，赋给变量 intVariable 的值也可能是 2，而非四舍五入后的 3。由于无法保证编译器允许这种赋值语句，所以要避免直接将 double 值赋给 int 变量。

常量 2.99 替换成 double 类型的变量问题依旧。大多数编译器对以下赋值语句报错：

```
int intVariable;
double doubleVariable;
doubleVariable = 2.00;
intVariable = doubleVariable;
```

虽然 2.00 不需要四舍五入，但问题依旧存在。2.00 是 double 而不是 int。如后文所述，可将 2.00 替换为 2 赋给 doubleVariable，但这依旧无法让编译器正常赋值。变量 intVariable 和 doubleVariable 的类型不同，这才是关键。

即使编译器允许在赋值语句中混用不同类型，但大多数情况下都不应该这样做。这不利于程序的移植，而且会引起混淆。例如，如果编译器允许将 2.99 赋给 int 变量，该变量获得的将是 2 而非 2.99，这便产生了混淆，因为程序的意思是把 2.99 赋给变量。

一些特殊情况允许将一种类型的值赋给另一种类型的变量。将 int 类型的值赋给 double 类型的变量是允许的。例如，以下语句既有效也合乎情理：

```
double doubleVariable;
doubleVariable = 2;
```

以上语句将 doubleVariable 变量的值设为 2.0。

虽然不建议，但确实可以将 int 值(如 65)保存到 char 类型的变量中。另外，还可将字母(如'Z')保存到 int 类型的变量中。从多方面考虑，C 语言将字符视为小整数值。遗憾的是，C++从 C 沿袭了这一点。之所以选择这样做，是因为 char 类型的变量占用内存比 int 类型少。所以，用 char 类型的变量进行计算可节省一些内存。不过，更清晰的方案是在操作整数时使用 int 类型，操作字符时使用 char 类型。

一般的规则是，不要将一种类型的值放到另一种类型的变量中——虽然违反这个规则的情况似乎比遵守这个规则的情况多。即使编译器没有严格贯彻这一规则，你也应该主动遵守。将一种类型的数据放入另一种类型的变量中可能造成问题。因为值必须转换成相应类型的值，但转换后的值可能不是你想要的。

bool 类型的值可赋给整型(short，int 和 long)变量，整数也可赋给 bool 变量。但这样做不好，应尽量避免。考虑到知识的全面性，并帮助你阅读其他人写的代码，我打算说具体点：赋给 bool 类型的变量时，任何非零的整数都将保存为 bool 值 true，值 0 则保存为 bool 值 false。反之，将 bool 值赋给整数变量时，true 作为 1 存储，false 则作为 0 存储。

算术操作符和表达式

C++程序对象程序利用算术操作符合并变量和/或数字。这些操作符包括+(加)、-(减)、*(乘)和/(除)。例如，图 2.1 的程序用操作符*求两个变量值的积，结果放入等号左侧的

变量：

```
totalWeight = oneWeight * numberOfBars;
```

所有算术操作符都可以作用于两个 int 或两个 double，甚至可以一样一个。但确切的结果值及其类型取决于被合并的两个数字的类型。两个操作数(也就是两个数字)都是 int，用算术操作符合并的结果也是 int。一个(或两个)操作数是 double，结果就会变成 double。例如，假定变量 baseAmount 和 increase 都是 int，则以下表达式的结果也是 int：

```
baseAmount + increase
```

但其中任何一个是 double，结果就是 double。将+换为-，*或/，结论一样。

结果类型的重要性可能超乎你的想象。例如，7.0/2 有一个 double 类型的操作数，即 7.0，所以结果是 double 值 3.5。但 7/2 的两个操作数都是 int，所以结果是 int 值 3。即使结果表面上相等，但实际可能还是有区别。例如，6.0/2 有一个 double 操作数，即 6.0，所以结果是 double 值 3.0，它只是一个近似值。但 6/2 有两个 int 操作数，所以结果是 int 值 3，这是一个精确值。除法操作符是受参数类型影响最大的操作符。

使用除法操作符时，如果一个或两个操作数是 double 类型，结果是符合预期的。但为 int 类型的两个操作数使用除法操作符，只能获得除法运算结果的整数部分。也就是说，整数除法丢弃了小数部分。所以，10/3 的结果是 3(而不是 3.3333…)，5/2 的结果是 2(而不是 2.5)，11/3 的结果是 3 (而不是 3.6666…)。注意这些数字没有四舍五入，小数部分被直接丢弃，不管它有多大。

将操作符%应用于 int 类型的操作数，可还原使用/对两个整数进行除法运算时丢失的信息。应用于 int 值时，两个操作符/和%将生成执行长除法时获得的两个数(长除法是中学知识)。例如，17 用 5 除，商 3 余 2。/运算得到商，%得到余。例如，以下语句：

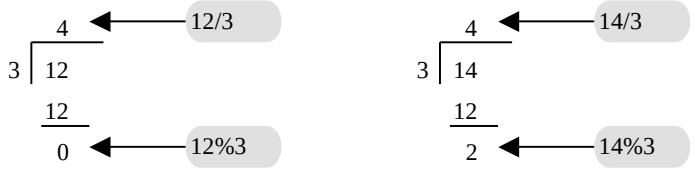
```
cout << "17 divided by 5 is " << (17/5) << endl;  // 商
cout << "with a remainder of " << (17%5) << endl;  // 余
```

输出如下：

```
17 divided by 5 is 3
with a remainder of 2
```

图 2.6 演示了将操作符/和%应用于 int 值时的情况。

图 2.6　整数除法



向负 int 值应用/和%操作符的结果因 C++实现而异。所以，为确保程序的可移植性，只有在确定两个 int 都非负时，才为它们使用操作符/和%。

算术表达式中可加入合理的空白间距。可在操作符和圆括号前后插入空格，当然也可以不加。只要易于阅读，加或不加可自由选择。

可插入圆括号来指定运算顺序，如以下两个表达式所示：

```
 (x + y) * z
x + (y * z)
```

对第一个表达式求值，计算机首先使 x 和 y 相加，再将结果乘以 z。对第二个表达式求值，计算机首先使 y 和 z 相乘，再将结果加到 x 上。尽管数学公式可以使用方括号和其他形式的括号，但 C++不允许。C++只允许在算术表达式中使用圆括号来指定运算顺序。其他形式的括号有其他用途。

如省略圆括号，计算机将根据**优先级规则**决定操作符(如+和*)的求值顺序。这些优先级规则与代数和其他数学课采用的规则相似。例如，对以下表达式求值时：

```
x + y * z
```

将先乘后加。除了一些非常标准的情况(比如一连串的加法，或者在加法运算中嵌入简单乘法)，否则最好坚持用圆括号指定执行顺序。圆括号使表达式更易理解，并避免出错。附录 2 给出了完整的 C++优先级规则。


视频讲解：*Precedence and Arithmetic Operators*

图 2.7 列出了一些常见的算术表达式和对应的 C++表达式。

图 2.7 算术表达式

数学公式	C++表达式
$b^2 - 4ac$	`b*b - 4*a*c`
$x(y + z)$	`x * (y + z)`
$\frac{1}{x^2 + x + 3}$	`1 / (x*x + x + 3)`
$\frac{a + b}{c - d}$	`(a + b)/(c - d)`

陷阱：除法中的整数

两个整数相除结果也是整数。但如果期望的是小数就会成为问题。另外，该问题往往不易察觉，导致程序看起来正确，但会产生不正确的输出。例如，假定园林设计师设计公路景观时按每英里 5000 美元收费。已知以英尺为单位的公路长度，使用以下 C++语句能轻松计算出应收取费用：

```
totalPrice = 5000 * (feet/5280.0);
```

1 英里 5280 英尺，上述语句可行。假定公路长度 15000 英尺，以下公式获得总价格：

```
5000 * (15000/5280.0)
```

C++程序按以下顺序获得终值：15000/5280.0 结果是 2.84。然后 5000 和 2.84 相乘，得到 14200.00。借助于你写的 C++程序，可知该工程应收 14200 美元。

假定 feet 是 int 类型，同时忘记包含小数点和后面的 0，赋值语句变为以下形式：

```
totalPrice = 5000 * (feet/5280);
```

看起来没错，但会造成严重问题。使用这种形式的赋值语句，表示要使两个 int 类型

的值相除，所以 feet/5280 相当于 15000/5280，结果是 int 值 2(而不是你希望的 2.84)。所以，赋给 totalPrice 的值是 5000*2，或者说 10000。忘记小数点，收取的费用就变成 10000 美元。但如前所述，实际应收款是 14200 美元。遗漏小数点就蒙受了 4200 美元的损失。注意，totalPrice 的类型无论是 int 还是 double，都无法改变这一事实。损失发生在向 totalPrice 赋值之前。 ■

√ 自测题

15. 将以下数学公式转换为 C++表达式：

$3x$　　$3x + y$　　$\frac{x+y}{7}$　　$\frac{3x+y}{z+2}$

16. 以下几行程序输出什么(假定它们嵌入一个正确的程序，而且所有变量都声明为 char 类型)：

```
a = 'b';
b = 'c';
c = a;
cout << a << b << c << 'c';
```

17. 以下两行程序输出什么(假定它们嵌入一个正确的程序，而且 number 声明为 int 类型)：

```
number = (1/3) * 3;
cout << "(1/3) * 3 is equal to " << number;
```

18. 写完整 C++程序将两个整数读入两个 int 变量，输出第一个数除以第二个数的商和余。可用操作符/和%来实现。

19. 给定以下代码段：

```
double c = 20;
double f;
f = (9/5) * c + 32.0;
```

它的目的是将摄氏温度(c)换算为华氏温度(f)，请回答以下问题。

a. 赋给 f 的值是什么？
b. 解释实际发生的情况和程序员的本意。
c. 重写代码使之符合程序员的本意。

20. 以下程序的输出结果是什么(假定它们嵌入一个正确的程序，已将 month, day, year 和 date 声明为 string 类型)？

```
month = "03";
day = "04";
year = "06";
date = month + day + year;
cout << date << endl;
```

更多赋值语句

赋值操作符(=)可以和算术操作符合并，使变量和值进行加、减、乘、除来改变自身。常规形式如下：

```
Variable Op= Expression
```

它等价于：

```
Variable = Variable Op (Expression)
```

其中，*Op* 是操作符(比如+，-和*)。*Expression* 可以是另一个变量、常量或者更复杂的表达式。下面给出一些例子。

示例	等价形式
`count += 2;`	`count = count + 2;`
`total -= discount;`	`total = total - discount;`
`bonus *= 2;`	`bonus = bonus * 2;`
`time /= rushFactor;`	`time = time / rushFactor;`
`change %= 100;`	`change = change % 100;`
`amount *= cnt1 + cnt2;`	`amount = amount * (cnt1 + cnt2);`

2.4 简单控制流程

“如果你以为我们是蜡做的人像，那你就应该先付钱，”他说，“你知道，蜡像不是做来给人白看的。嘿！不是的!”

“反过来说，”那个有着“弟”字的小胖子说，“如果你认为我们是活的，你就应该说话。”

——*刘易斯·卡洛尔，《爱丽丝镜中奇遇记》*

前面的程序都由一些简单的语句构成。语句按给定顺序逐行执行。但更复杂的程序要求能以某种方式改变语句执行顺序。语句执行顺序通常称为**控制流程**或**控制流**。本节介绍向程序添加控制流程的两种简单方式。要讨论一种分支机制，允许程序在两个备选行动中选一个，具体选择哪个由变量值决定。还要讨论一种循环机制，允许重复一个行动。

一个简单的分支机制

有时需要让程序根据输入从两个备选行动中选一个。以计算计时员工周薪为例，假定公司薪水制度是每周工作 40 小时以内，按正常工资计算。超出 40 小时算加班，按正常工资 1.5 倍计算。员工一周内的工作时间大于或等于 40 小时，工资计算公式如下：

```
rate * 40 + 1.5 * rate * (hours - 40)  // rate 是时薪
```

但员工工作时间也可能少于 40 小时。这种情况下上述公式会算出负值(要体会这一点，可将 10 代入 hours，将 1 代入 rate，并执行计算。倒霉的员工领到的薪水将为负数。如工作时间少于 40 小时，正确薪水计算公式其实是非常简单的，如下所示：

```
rate * hours
```

要同时支持这两种情况(工作时间大于或等于 40 小时，或小于 40 小时)，程序必须能在这两个公式之间做出选择。为计算员工的周薪，程序要采取的行动如下所示：

```
判断(hours > 40)是否为 true
如果是，执行以下赋值语句：
   grossPay = rate * 40 + 1.5 * rate * (hours - 40);
如果不是，执行以下赋值语句：
   grossPay = rate * hours;
```

有一个 C++语句专门执行这种分支行动。**if-else 语句**会在两个备选行动之间选择。例如，前面讨论的周薪计算可用以下 C++语句完成：

```
if (hours > 40)
    grossPay = rate * 40 + 1.5 * rate * (hours - 40);
else
    grossPay = rate * hours;
```

图 2.8 用一个完整程序演示了该语句的用法。

图 2.8　if-else 语句

```
#include <iostream>
using namespace std;
int main()
{
   int hours;
   double grossPay, rate;
   cout << "Enter the hourly rate of pay: $";
   cin >> rate;
   cout << "Enter the number of hours worked,\n"
        << "rounded to a whole number of hours: ";
   cin >> hours;

   if (hours > 40)
      grossPay = rate*40 + 1.5*rate*(hours - 40);
   else
      grossPay = rate*hours;
   cout.setf(ios::fixed);
   cout.setf(ios::showpoint);
   cout.precision(2);
   cout << "Hours = " << hours << endl;
   cout << "Hourly pay rate = $" << rate << endl;
   cout << "Gross pay = $" << grossPay << endl;
   return 0;
}
```

示范对话 1

```
Enter the hourly rate of pay: $20.00
Enter the number of hours worked,
rounded to a whole number of hours: 30
Hours = 30
Hourly pay rate = $20.00
Gross pay = $600.00
```

示范对话 2

```
Enter the hourly rate of pay: $10.00
Enter the number of hours worked,
rounded to a whole number of hours: 41
Hours = 41
Hourly pay rate = $10.00
Gross pay = $415.00
```

图 2.9 总结了 if-else 语句的两种形式。第一种是最简单的 if-else 语句，第二种是 2.4.4 节“复合语句”要讨论的主题。第一种形式，两个语句可为任何可执行语句。*Boolean_Expression* 是测试，求值结果要么为 true，要么为 false。检查该表达式的值就知道是否满足条件。之前的 if-else 语句的 *Boolean_Expression* 是：

```
hours > 40
```

程序到达 if-else 语句时，实际执行两个内嵌语句之一。*Boolean_Expression* 为 true(满足条件)，执行 *Yes_Statement*。*Boolean_Expression* 为 false(不满足条件)，执行 *No_Statement*。注意，*Boolean_Expression* 必须包含在圆括号内(这是 C++ if-else 语句的语法规则)。

图 2.9 if-else 语句的语法

每个备选行动只有一个语句：

```
if (Boolean_Expression)
    Yes_Statement
else
    No_Statement
```

每个备选行动有一组语句：

```
if (Boolean_Expression)
{
    Yes_Statement_1
    Yes_Statement_2
        . . .
    Yes_Statement_Last
}
else
{
    No_Statement_1
    No_Statement_1
        . . .
    No_Statement_Last
}
```

布尔表达式是求值结果为 true 或 false 的表达式。if-else 语句肯定包含一个 *Boolean_Expression*。最简单的 *Boolean_Expression* 由两个表达式(比如数字或变量)构成，它们使用图 2.10 总结的某个比较操作符进行比较。注意有的操作符由两个符号构成，例如==，!=，<=和>=。注意，必须用双等号==表示相等，用!=表示不等。在两个符号构成的操作符中，两个符号之间不能有空格。执行 if-else 语句时，首先对要比较的两个表达式进行求值，再用操作符进行比较。结果为 true 时，执行第一个语句，否则执行第二个。

图 2.10 比较操作符

数学符号	中文说法	C++表示法	C++示例	等价的数学表示
=	等于	==	x + 7 == 2*y	x + 7 = 2y
≠	不等于	!=	ans != 'n'	ans ≠ 'n'
<	小于	<	count < m + 3	count < m + 3
≤	小于或等于	<=	Time <= limit	time ≤ limit
>	大于	>	Time > limit	time > limit
≥	大于或等于	>=	age >= 21	age ≥ 21

可用 and(逻辑与)操作符(在 C++中表示为&&)合并两个比较。例如，假定 x 大于 2，而且小于 7，则以下布尔表达式为 true(即条件满足)：

```
(2 < x) && (x < 7)
```

用&&连接两个比较时，只有两个比较的结果都为 true(两个条件都满足)，整个表达式才为 true；否则整个表达式为 false。

“逻辑与”操作符&&

可用逻辑与(and)操作符&&将两个简单测试合并成一个更复杂的布尔表达式。

语法(针对使用&&的布尔表达式)

```
(Comparison_1) && (Comparison_2)
```

示例(包含在一个 if-else 语句内)

```
if ( (score > 0) && (score < 10) )
   cout << "score is between 0 and 10.\n";
else
   cout << "score is not between 0 and 10.\n";
```

如果 score 的值大于 0，同时小于 10，就执行第一个 cout 语句；否则执行第二个。

还可使用 or(逻辑或)操作符(在 C++中表示为||)来合并两个比较表达式。例如，假定 y 小于 0 或 y 大于 12，则以下表达式为 true：

```
(y < 0) || (y > 12)
```

用||连接两个比较时，只要其中任何一个为 true 或者两个都为 true(满足任何条件)，整个表达式就为 true；否则，整个表达式为 false。

“逻辑或”操作符||

可用逻辑或(or)操作符||将两个简单测试合并为一个更复杂的布尔表达式。

语法(针对使用||的布尔表达式)

```
(Comparison_1) || (Comparison_2)
```

示例(包含在一个 if-else 语句内)

```
if ( (x == 1) || (x == y) )
   cout << "x is 1 or x equals y.\n";
else
   cout << "x is neither 1 nor equal to y.\n";
```

如果 x 等于 1，或者 x 等于 y(或者同时满足这两个条件)，就执行第一个 cout 语句，否则执行第二个。

记住，在 if-else 语句中使用布尔表达式时，必须用圆括号将布尔表达式封闭起来。例如，在一个 if-else 语句中，如 if 的条件使用了操作符&&，并包含两个比较，就要像下面这样用圆括号封闭 if 之后的整个比较表达式：

```
if ( (temperature >= 95) && (humidity >= 90) )
   . . .
```

注意，总共有两层圆括号，外层的圆括号是必需的，但内层的圆括号(用于封闭不同的比较)则不是必需的。但由于能使语句结构更清楚，所以通常都会包括它们。

可用求反操作符!对任何布尔表达式求反。如希望对一个布尔表达式求反，可以把表达式放入一对圆括号中，再把操作符!放到它的前面。例如，!(x < y)表示“x 不小于 y”。

由于 if-else 语句中的布尔表达式必须包含在一对圆括号中，所以在 if-else 语句中使用求反表达式时，要再用一对圆括号封闭求反表达式。例如，if-else 语句可能这样开始：

```
if (!(x < y))
    ...
```

可以避免使用!操作符。例如，上述 if-else 语句可更换成以下版本，它是等价的，而且更容易理解：

```
if (x >= y)
    ...
```

本书很晚的时候才会真正用到!操作符，届时更详细讨论。

有时希望 if-else 语句中的一个备选行动不做任何事情。在 C++中，这可以通过省略 else 部分来实现。这种形式的语句称为 **if 语句**，以区别于 if-else 语句。例如，以下两个语句中的第一个语句就是 if 语句：

```
if (sales >= minimum)          // 如果实际销售额大于或等于公司规定的最小值，
    salary = salary + bonus; // 就把奖金加上
cout << "salary = $"  << salary;
```

假如 sales 的值大于或等于 minimum 的值，就执行赋值语句，再执行后续的 cout 语句。但是，假如 sales 的值小于 minimum，就不执行缩进的赋值语句。在这种情况下，if 语句不会导致任何更改(也就是说，不会有奖金加到基本工资里)，程序直接执行 cout 语句。

陷阱：连续的不等式

不要在程序中使用如下所示的连续不等式：

```
if  (x < z < y)        <-- 不要这样做!
cout << "z is between x and y.";
```

在程序中使用以上语句，程序也许能通过编译并运行，但毫无疑问，它会产生错误的结果。我们将在了解 C++语言更多的细节之后，再讨论这个问题的根源。使用任何比较操作符(而非仅仅是<)来进行一连串的比较时，都会发生同样的问题。表示连续不等式的正确方式是使用“逻辑与”操作符&&，如下所示：

```
if ( (x < z ) && (z < y) )        <-- 正确的形式
    cout << "z is between x and y.";
```

■

陷阱：该用==的时候用了=

视频讲解：*Common Bugs with = and ==*

遗憾的是，在 C++中，你认为正确的许多 C++语句实际上都会产生歧义。换言之，即使语句在语法上正确，程序能成功编译并运行，不报告任何错误，但结果仍然和希望的不符。你可能没有意识到一些语句写错了，所以可能导致很严重的问题。即使发现结果不正确，但绞尽脑汁也想不出症结在哪里。一个常见的错误是在本该使用==的时候使用了=。例如像下面这样开头的一个 if-else 语句：

```
if (x = 12)
```

```
  Do_Something
else
  Do_Something_Else
```

假定要测试 x 的值是否等于 12，所以本意是用==而不是=。你也许以为编译器能捕捉这个错误，因为对于以下表达式来说：

```
x = 12
```

它不是逻辑表达式，不能表示一个条件是否满足。相反，它是赋值语句，所以编译器理应报错。遗憾的是，现实并非如此。在 C++中，表达式 x = 12 和 x + 12 与 2 + 3 一样，是一个能返回值的表达式(或者说是具有值的表达式)。整个赋值表达式的值就是传给等号左侧变量的值。例如，x = 12 的值就是 12。从前面对布尔值兼容性的讨论可知，int 值可转换成 true 或 false。由于 12 是一个非 0 的值，所以会转换成 true。如果将 x = 12 用作 if 语句中的布尔表达式，这个布尔表达式将始终为 true，所以，程序始终会执行第一个分支(*Do_Something*)。

这个错误很难发现，因为它看起来是正确的！相反，假如将 12 放在比较表达式左侧(如下所示)：

```
if (12 == x)
  Do_Something
else
  Do_Something_Else
```

那么一旦错误地用操作符=来取代上面的操作符==，编译器就会报错。

记住，在操作符==中少写一个=是许多编译器都不能捕捉的常见错误，这个错误很难发现，而且几乎肯定不会产生你希望的结果。C++的许多可执行语句可作为几乎任何表达式使用，其中包括 if-else 语句的布尔表达式。在本该使用布尔表达式的地方使用了赋值语句，赋值语句会被解释为布尔表达式。但此时的“测试”结果肯定和你希望的不符。上述 if-else 语句乍一看是对的，也能正常编译和运行，但结果可能出乎预料。 ■

复合语句

经常都要求 if-else 语句的每个分支执行多个语句。为此，可将每个分支的语句封闭到一对花括号中，如图 2.9 第二个语法模板所示，图 2.11 则给出了一个实际的例子。花括号内的一组语句统称为**复合语句**。在 C++中，复合语句被视为单个语句。凡是能使用单个语句的地方都能换成复合语句(因此，图 2.9 的第二个语法模板实际是第一个模板的特例)。图 2.11 包含两个复合语句，它们嵌入一个 if-else 语句中。

图 2.11　在 if-else 语句中使用复合语句

```
1  if (myScore > yourScore)
2  {
3      cout << "I win!\n";
4      wager = wager + 100;
5  }
6  else
7  {
8      cout << "I wish these were golf scores.\n";
9      wager = 0;
10 }
```

if-else 的语法规则要求 *Yes_Statement* 和 *No_Statement* 分别只能是一个语句。一个分支要执行若干个语句，就必须将它们放到花括号中，转换成复合语句。编译器发现 if 和 else 之间有两个或多个不在花括号中的语句会报错。

√ 自测题

21. 写一个 if-else 语句，使其在 score 变量的值大于 100 时输出 High，在 score 的值小于等于 100 时输出 Low。score 变量是 int 类型。

22. 假设 savings (存款) 和 expenses (开销) 是 double 类型的变量，而且已经赋值。写一个 if-else 语句，如果 savings 大于等于 expenses，就使 savings 的值减去 expenses 的值，结果再赋给 savings。然后，将 expenses 的值设为 0，并输出单词 Solvent(有偿还能力)。但是，如果 savings 小于 expenses，if-else 语句就只输出单词 Bankrupt(破产)，不更改任何变量的值。

23. 写一个 if-else 语句，使其在变量 exam(考试成绩)的值大于或等于 60，而且变量 programsDone 的值大于或等于 10 的条件下，输出单词 Passed(通过)；否则，if-else 语句输出单词 Failed(没通过)。变量 exam 和 programsDone 都是 int 类型。

24. 写一个 if-else 语句，使其在变量 temperature(温度)大于或等于 100，或者变量 pressure(气压)的值大于或等于 200，或者同时满足这两个条件的前提下，输出单词 Warning(报警)；否则，if-else 语句输出单词 OK(正常)。变量 temperature 和 pressure 都是 int 类型。

25. 假设有以下二次方程式：

$$x^2 - x - 2$$

描述它在什么情况下为正(也就是大于 0)。换言之，需要描述要么小于较小根(-1)，要么大于较大根(+2)的一系列数字。写一个 C++布尔表达式，使其在方程式为正时求值为 true。

26. 假设有以下二次方程式：

$$x^2 - 4x + 3$$

描述它在什么情况下为负。换言之，需要描述不仅大于较小根(+1)，还要小于较大根(+3)的一系列数字。写一个 C++布尔表达式，使其在方程式为负时求值为 true。

27. 在 if-else 语句中嵌入的以下 cout 语句将输出什么？假设已将这些代码嵌入一个完整、正确的程序中。请解释你的答案。

a.
```
if (0)
  cout << "0 is true";
else
   cout << "0 is false";
cout << endl;
```

b.
```
if (1)
   cout << "1 is true";
else
   cout << "1 is false";
cout << endl;
```

c.
```
if (-1)
     cout << "-1 is true";
else
     cout << "-1 is false";
cout << endl;
```

注意：仅供练习，不应该遵循这样的编程风格。

简单的循环机制

大多数程序都包含需要多次重复的行动。以图 2.8 用于计算员工工资总额的程序为例。如果公司有 100 名员工，一个更完善的工资程序将重复这一计算 100 次。程序中，用于重复一个或一组语句的那部分称为**循环**。C++语言提供了多种方式创建循环。其中一个称为 **while 语句**，或者称为 **while 循环**。为了展示它的用法，首先介绍一个简单的练习程序，再介绍一个更实用的程序。

图 2.12 的程序含有简单的 `while` 语句(以粗体显示)。花括号{和}之间的部分称为 `while` 循环**主体**，也就是需要重复采取的行动。花括号内的语句依次执行，执行完一次之后，再从头继续执行，如此重复，直到 `while` 循环结束。在第一个示范对话中，循环主体在循环结束之前，重复执行了三次，所以程序输出了三次 `Hello`。循环主体的每一次重复都称为循环的一次**迭代**。因此，第一个示范对话表明循环进行了三次迭代。

图 2.12　while 循环

```
 1  #include <iostream>
 2  using namespace std;
 3  int main()
 4  {
 5     int countDown;

 6     cout << "How many greetings do you want? ";
 7     cin >> countDown;

 8     while (countDown > 0)
 9     {
10        cout << "Hello ";
11        countDown = countDown - 1;
12     }

13     cout << endl;
14     cout << "That's all!\n";

15     return 0;
16  }
```

示范对话 1

```
How many greetings do you want? 3
Hello Hello Hello
That's all!
```

示范对话 2

```
How many greetings do you want? 1
Hello
That's all!
```

示范对话 3

```
How many greetings do you want? 0
                                  ←——循环主体执行 0 次
That's all!
```

“while”(当……时)一词已暗示了 `while` 语句的含义。当圆括号内的布尔表达式满足条件时(求值结果为 `true`)，就重复执行循环。在图 2.12 中，这意味着只要变量 `countDown` 大于 `0`，就重复执行循环主体。下面以第一个示范对话为例，研究一下 `while` 循环是如何执行的。用户输入 `3`，所以 `cin` 语句将 `countDown` 的值设为 `3`。在这种情况下，当程序到

达 while 语句时，肯定满足 countDown 大于 0 这个条件，所以执行循环主体中的语句。每次重复循环主体时，都会执行以下两个语句：

```
cout << "Hello ";
countDown = countDown - 1;
```

因此，循环主体每次重复都会输出"Hello"，同时变量 countDown 的值递减 1。计算机重复了三次循环主体后，变量 countDown 的值递减至 0，圆括号中的布尔表达式不再满足。所以，重复了三次循环主体后，这个 while 语句就会终止。

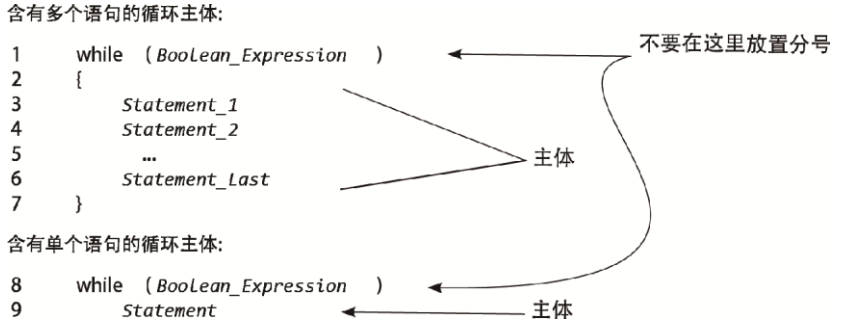
图 2.13 总结了 while 语句的语法。对 *Boolean_Expression* 的要求和对 if-else 语句中的布尔表达式的要求完全一样。和 if-else 语句一样，while 语句中的布尔表达式也必须用圆括号括起来。图 2.13 针对两种情况给出了语法模板：第一种情况是循环主体有多个语句。第二种情况是循环主体只有一个语句。注意，只有一个语句时不必使用花括号。

现在着重讨论 while 语句采取的行动。执行 while 语句时，发生的第一件事情是检查单词 while 之后的布尔表达式。该表达式的求值结果要么为 true，要么为 false。例如：

```
countDown > 0
```

如果 countDown 的值为正，则求值结果为 true；如果为 false，就不采取行动，程序将继续执行 while 语句之后的下一个语句。如果比较结果为 true，就执行整个循环主体。在被比较的表达式中，通常至少要有一个包含了即将由循环主体改变的东西，比如图 2.12 的 while 语句中的 countDown 的值。执行循环主体之后，会再次进行比较。只要比较结果为 true，这个过程就会不断地重复。循环主体每次迭代后，会再次进行比较，假如结果为 true，就再次执行整个循环主体。一旦比较结果不为 true，就结束 while 循环。

图 2.13　while 语句的语法



while 语句执行时，第一件事是检查布尔表达式。表达式不为 true，就永远不执行循环主体，具体可参见图 2.12 的示范对话 3。许多时候都要求循环主体执行 0 次。例如，假定 while 循环要读取所有不及格分数，但实际没人不及格，就应该让循环主体执行 0 次。

一个 while 循环可能执行 0 次循环主体，这是很正常的情况。但是，如果要求循环主体在任何情况下都至少执行一次，就应该使用 do-while 循环。do-while 语句与 while 语句相似，只是它的循环主体至少执行一次。do-while 语句的语法请参见图 2.14。

图 2.15 是使用了 do-while 循环的示范程序。do-while 循环的第一件事情就是执行循环主体语句。循环主体第一次迭代之后，do-while 的行为就和 while 一样了。换言之，

就是检查布尔表达式，结果为 true 就再次执行循环主体，再次检查布尔表达式，如此反复。

图 2.14　do-while 语句的语法

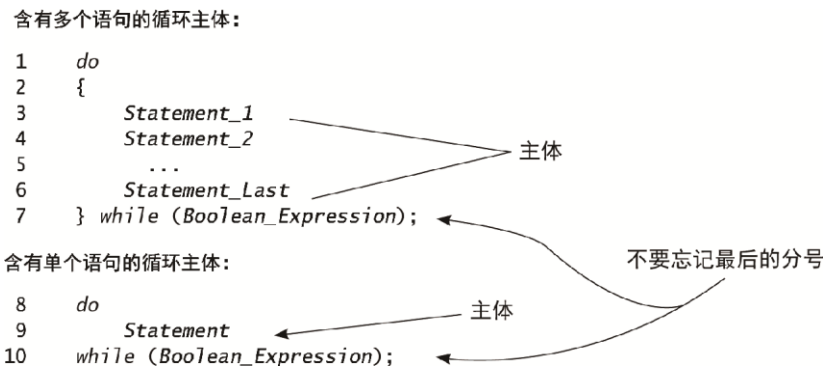


图 2.15　do-while 循环

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      char ans;

6      do
7      {
8          cout  << "Hello\n";
9          cout  << "Do you want another greeting?\n"
10                << "Press y for yes, n for no,\n"
11                << "and then press return: ";
12         cin   >> ans;
13     } while (ans == 'y' || ans == 'Y');

14     cout << "Good-Bye\n";

15     return 0;
16 }
```

示范对话

```
Hello
Do you want another greeting?
Press y for yes, n for no, and then press return: y
Hello
Do you want another greeting?
Press y for yes, n for no, and then press return: Y
Hello
Do you want another greeting?
Press y for yes, n for no, and then press return: n
Good-Bye
```

递增操作符和递减操作符

2.3 节讨论了二元操作符。**二元操作符**有两个操作数。**一元操作符**则只有一个。之前用过两个一元操作符，即+和-，它们在+7 和-7 这样的表达式中使用。C++语言还有另外两个很常用的一元操作符，即++和--。操作符++称为**递增操作符**，操作符--称为**递减操作符**。它们通常与 int 类型的变量一起使用。假定 n 是 int 类型的变量，n++使 n 递增 1，n--使 n 递减 1。所以，n++和 n--(后跟一个分号)是可执行语句。例如以下语句：

```
int n = 1, m = 7;
n++;
cout << "The value of n is changed to " << n << endl;
m--;
cout << "The value of m is changed to " << m << endl;
```

产生的输出如下：

```
The value of n is changed to 2
The value of m is changed to 6
```

这时你想必会恍然大悟，猜到 C++这个名称中的“++”是怎么来的。

递增和递减语句常在循环中使用。例如，图 2.12 的 while 循环中使用了以下语句：

```
countDown = countDown - 1;
```

但大多数有经验的 C++程序员都会选择使用递减操作符。所以整个 while 循环可这样修改：

```
while (countDown > 0)
{
   cout << "Hello ";
   countDown--;
}
```

编程实例 信用卡余额

假定有一张信用卡，卡上已产生应还金额 50 美元，银行按 2%月利率收费。假定一直不还款，多少个月之后，这张卡的应还金额会超过 100 美元？解决这个问题的一个办法是查看每月账单，统计在应还金额达到或超过 100 美元之前，总共会经历多少个月。但更好的办法是用程序计算每月应还金额，而不必等着银行寄账单。通过这种方式，不需要漫长的等待(也不会影响自己的信用评级)，就能迅速得到答案。

一个月后，卡的应还金额是 50 美元加 50 美元的 2%，也就是 51 美元。两个月后，卡的应还金额是 51 美元加 51 美元的 2%，也就是 52.02 美元。三个月之后，卡的应还金额是 52.02 美元加 52.02 美元的 2%，依此类推。总之，每月应还金额都会增加 2%。该程序将应还金额保存到名为 balance 的变量中对其进行跟踪。每月对 balance 变量值的修改可以像下面这样进行：

```
balance = balance + 0.02 * balance;
```

重复这个行动，直到 balance 的值达到(或超过)100 美元，并对重复次数进行计数，就知道多少月后应还金额达到 100 美元。为此，需要用另一个变量对 balance 的修改次数进行计数。假定新变量名称是 count。while 循环中最终的主体将包含以下语句：

```
balance = balance + 0.02 * balance;
count++;
```

为了使循环正确执行，必须在循环执行之前将恰当的值赋给变量 balance 和 count。本例在声明变量的同时初始化。完整程序参见图 2.16。

图 2.16　信用卡程序

```
#include <iostream>
using namespace std;
int main()
{
   double balance = 50.00;
   int count = 0;

   cout << "This program tells you how long it takes\n"
        << "to accumulate a debt of $100, starting with\n"
        << "an initial balance of $50 owed.\n"
        << "The interest rate is 2% per month.\n";

   while (balance < 100.00)
   {
      balance = balance + 0.02 * balance;
      count++;
   }

   cout << "After " << count << " months,\n";
   cout.setf(ios::fixed);
   cout.setf(ios::showpoint);
   cout.precision(2);
   cout << "your balance due will be $" << balance << endl;

   return 0;
}
```

示范对话

```
This program tells you how long it takes
to accumulate a debt of $100, starting with
an initial balance of $50 owed.
The interest rate is 2% per month.
After 36 months,
your balance due will be $101.99
```

陷阱：无限循环

只要 while 之后的布尔表达式为 true，while 或 do-while 循环就不会终止。布尔表达式通常包含一个将由循环主体更改的变量。变量应不断更改，最终使布尔表达式求值为 false，从而终止循环。但假如不小心让布尔表达式始终为 true，循环就会一直运行。这种循环称为**无限循环**。

首先描述一个能终止的循环。以下 C++代码输出小于 12 的所有正偶数。也就是说，它将逐行输出数字 2，4，6，8 和 10，然后循环终止：

```
x = 2;
while (x != 12)
{
   cout << x << endl;
   x = x + 2;
}
```

每次循环迭代，x 的值都递增 2，直到变成 12。此时，单词 while 之后的布尔表达式不再为 true，循环结束。

现在假设要输出小于 12 的奇数，而不是偶数。你可能以为只需这样修改初始化语句：

```
x = 1;
```

但这是错误的，它会导致无限循环。因为 x 的值会从 11 变成 13，永远不等于 12，所以循环永远不会终止。

用==或!=检查一个数来终止循环，很容易造成无限循环问题。处理数字时，更安全的做法始终是测试它是否超过了一个值。例如，将前面 while 循环的第一行改为以下语句，就可将 x 初始化为任意数字，循环始终都能正常终止：

```
while (x < 12)
```

无限循环的程序不能自动停止，除非强行终止。初学编程的读者可能写出含有无限循环的程序，所以最好学会怎样强行终止程序。强行终止程序的方法因系统而异。在很多系统上，按 Control+C(按 Ctrl 键的同时按 C 键)就可以终止程序。 ■

√ 自测题

28. 以下代码的输出是什么(假定它们已嵌入一个正确的程序，而且 x 已声明为 int 类型)?

```
x = 10;
while (x > 0)
{
    cout << x << endl;
    x = x - 3;
}
```

29. 在上一题中，将>替换为<会输出什么？

30. 以下代码的输出是什么(假定它们已嵌入一个正确的程序，而且 x 已声明为 int 类型)?

```
x = 10;
do
{
    cout << x << endl;
    x = x - 3;
} while (x > 0);
```

31. 以下代码的输出是什么(假定它们已嵌入一个正确的程序，而且 x 已声明为 int 类型)?

```
x = -42;
do
{
    cout << x << endl;
    x = x - 3;
} while (x > 0);
```

32. while 语句和 do-while 语句有何重要区别？

33. 以下代码的输出是什么(假定它们已嵌入一个正确的程序，而且 x 已声明为 int 类型)?

```
x =10;
while (x > 0)
{
    cout << x << endl;
    x = x + 3;
}
```

34. 写完整 C++程序逐行输出 1～20 的整数。该程序不执行别的任务。

2.5　程 序 风 格

在非常重要的事情上，最要紧的并非真诚，而是格调[①]。

——*奥斯卡·王尔德，《不可儿戏》*

我们的示范程序中，所有变量名都准确表达了它们的用途。示范程序在编排上采用了一种特定的格式。例如，声明和语句的缩进量是相同的。这些风格不仅仅是为了美观。风格好的程序更容易阅读，更容易纠正，也更容易修改。

缩进

程序应合理编排，使原本就是一个整体的各元素能组合到一起。一种方法是在逻辑上独立的各个部分之间插入空行。合理的缩进也有助于澄清程序结构。包含在一个语句内部的其他语句应该缩进。特别是，`if-else` 语句、`while` 循环和 `do-while` 循环应该缩进。缩进可采用示范程序的方式，也可采用其他类似的方式。

花括号{}用于确定程序结构中的一个较大的部分。像示范程序中那样，使每个花括号都单独占一行，以便定位一对匹配的花括号。注意，我们有意缩进了一些花括号。如果一对花括号要嵌入另一对中，嵌入的那一对花括号的缩进量要大于外面那一对，具体可参见图 2.16。`while` 循环主体的花括号缩进量应该大于程序 `main` 部分的花括号的缩进量。

至于在何处放置花括号，至少有两种理论。一种是像本书那样，每个花括号都单独占一行。这种形式更容易阅读。第二种是起始花括号不单独占一行。如果小心使用，第二种方法将更高效，而且更省空间。采用某种风格时，关键在于澄清程序结构。最终采用什么风格是因人而异的，你应该在自己的所有程序中坚持自己既定的风格。

注释

为使程序容易理解，应在程序的关键位置添加一些解释性批注。这些批注称为**注释**。C++和其他大多数编程语言一样都允许添加程序注释。在 C++中，符号//指出一条注释开始。符号//与行末之间的所有文本都是注释文本。编译器会忽略//之后这一行的所有内容。如果一条注释需要占据多行，就在每行注释之前添加符号//。符号//由两个连续的斜杠构成，之间无空格。

本书代码没有使用特殊格式，但在某些代码编辑器中，会使用和其他程序文本不同的颜色显示注释。

在 C++程序中，还有另一种方式插入注释。符号/*与*/之间的所有内容都会被视为注释，并被编译器忽略。/*到*/之间的注释可自由跨越多行。这样一来，就不需要像//注释那样，在每一行上都要添加一个额外的//。如下所示：

```
/* 这是一条跨越三行的注释。
注意第二行没有任何形式的
注释符号. */
```

① 格调、风格、样式在英语里面都是 Style。——译注

在程序中，但凡允许插入空格或换行符的位置，都允许插入/* */风格的注释。但不要把它插入不利于阅读的位置，也不要插入会干扰程序布局的位置。注释通常只应放在行末，或单独占一行。

至于哪种注释风格更好，人们的看法不一。但是，只要使用得当，任何一种注释都能取得理想的效果。本书统一采用//注释。

很难说程序中应该包含多少注释。唯一正确的答案是“够用就好”，但这对刚入门的程序员来说意义不大。只有积累一定经验之后，才能更好地把握添加注释的时机。一般在重要和不容易理解的地方添加注释。但注释太多有“喧宾夺主”之嫌。每行都有注释的程序使人难以看清程序的结构。例如，以下注释意义不大，不如不用：

```
distance = speed * time;  // 计算旅行的距离
```

注意图 2.17 的程序开头给出的注释。所有程序都应该以类似的注释开头。它给出了程序的所有基本信息：程序包含在哪个文件中、谁写的程序、如何联系作者、程序有何用途、程序的最新修改日期以及其他需要传达的信息，比如作业编号(如果该程序是课堂作业的话)。注释的具体内容要视情况而定。我们不打算在本书的其他程序中添加如此长的注释。但在你的程序中，应坚持在程序开头包含这样的注释。

图 2.17 注释和命名常量

```
1  // 文件名：health.cpp(你的程序可能需要一个不同于 cpp 的扩展名)
2  // 作者：<在此填写你的姓名>
3  // 电子邮件：you@yourmachine.bla.bla
4  // 题号：2
5  // 说明：判断用户是否生病的程序
6  // 上一次修改：2017 年 9 月 23 日
7
8  #include <iostream>
9  using namespace std;
10 int main()
11 {
12     const double NORMAL = 98.6; // 正常体温(华氏度)
13     double temperature;
14
15     cout << "Enter your temperature: ";
16     cin >> temperature;
17
18     if (temperature > NORMAL)
19     {
20         cout << "You have a fever.\n";  // 你发烧了
21         cout << "Drink lots of liquids and get to bed.\n";  // 多喝水，卧床休息
22     }
23     else
24     {
25         cout << "You don't have a fever.\n";  // 你没有发烧
26         cout << "Go study.\n";  // 去上学
27     }
28
29     return 0;
30 }
```

程序坚持以这种注释开头

示范对话

```
Enter your temperature: 98.6
You don't have a fever.
Go study.
```

为常量命名

计算机程序的数字存在两个问题。第一是不好记。例如，数字 10 在程序中出现时，不能就其含义传达任何有意义的信息。例如在银行程序中，10 既可能是分行数量，也可能是总行出纳窗口数量。理解程序需知道每个常量的含义。第二是程序更改部分数字时容易出错。假定数字 10 在上述银行程序中出现了 12 次，其中 4 次代表分行数，另外 8 次代表总行出纳窗口数。一旦银行开设新的分行，并需要更新程序，有的 10 可能需要变成 11，有的根本不能改动。为了避免这些问题，一个办法是为每个数字命名，在程序中使用名称而不是数字本身。例如，可在银行程序中包含两个常量，分别命名为 BRANCH_COUNT(分行数)和 WINDOW_COUNT(窗口数)。两个常量的值可能都为 10，但在开设新分行时，只需更改 BRANCH_COUNT 的定义就可以了。

如何命名 C++程序中的数字呢？一个办法是将变量初始化为那个数字，示例如下：

```
int BRANCH_COUNT = 10;
int WINDOW_COUNT = 10;
```

但是以这种方式命名数字常量也有问题：你可能会不小心更改上述变量的值。C++提供了一种方式来标记初始化好的变量，使其不能更改。程序试图更改就会出错。要标记一个不能更改的变量，请在变量声明之前添加单词 const(constant 的缩写)。例如：

```
const int BRANCH_COUNT = 10;
const int WINDOW_COUNT = 10;
```

如果变量为同一类型，上述两行可合并为一个声明，如下所示：

```
const int BRANCH_COUNT = 10, WINDOW_COUNT = 10;
```

但大多数程序员都认为，每个名称定义单独占一行显得更清楚。const 这个词通常称为**修饰符**，因其修饰(限制)了被声明的变量。

使用 const 修饰符声明的变量称为**声明常量**或**命名常量**(named constant)。C++语言不要求声明常量的名称全部大写，但这已成了 C++程序员约定俗成的标准。

一旦像这样命名了数字，在需要使用该数字的任何地方，都可以使用它的名称。名称和数字本身具有完全相同的含义。更改命名常量只需更改 const 变量声明的初始值。例如，要将 BRANCH_COUNT 的值从 10 改为 11，只需修改 BRANCH_COUNT 声明的初始值 10。

尽管可以在程序中使用未命名的数字常量，但尽量少用。通常，只有已知的、容易识别的而且不会改变的量(比如“1 米等于 100 厘米”中的 100)才可以使用未命名的数字常量。但其他所有数字常量都应该像刚才描述的那样指定名称。这会使程序更容易阅读和修改。

图 2.17 用一个简单程序展示了如何使用 const 修饰符。

用修饰符 const 命名常量

在声明中初始化变量时，可以标记变量，使程序不能更改该变量的值(使之成为常量)。为此，可在声明的开始处添加修饰符 const，如下所示：

语法

```
const Type_Name Variable_Name = Constant;
```

示例

```
const int MAX_TRIES = 3;
const double PI = 3.14159265;
```

√ 自测题

35. 以下 if-else 语句能正确编译和运行。但风格和本章其他程序不一致。修改它使风格(缩进和换行)一致：

```
if (x < 0) {x = 7; cout << "x is now positive.";}
else {x = -7; cout << "x is now negative.";}
```

36. 以下两行语句的输出是什么(假定已把它们嵌入一个完整和正确的程序)?

```
// cout << "Hello from";
cout << "Self-Test Exercise";
```

37. 写完整的 C++程序，要求用户输入加仑数(gallon)，再输出等价的公升数(liter)。1 加仑等于 3.78533 公升。请使用一个声明常量。这只是练习题，不需要在程序中添加任何注释。

小　结

- 为变量使用有意义的名称。
- 务必将变量声明为正确的数据类型。
- 变量使用前必须初始化。可在声明时初始化，也可在使用前用赋值语句初始化。
- 在算术表达式中合理使用圆括号，清楚指定运算顺序。
- 每次要求用户从键盘输入数据的时候都显示提示，并始终回显输入。
- if-else 语句允许程序在两个备选行动中选择一个。if 语句允许程序决定是否执行一个特定的行动。
- do-while 语句至少执行一次循环主体。而 while 循环有可能一次都不执行。
- 程序中，几乎所有数字常量都应被赋予有意义的名称，并在程序中使用名称代替数字。在变量声明中添加修饰符 const 来实现。
- 使用与本书示范程序类似的缩进、间距和换行风格。
- 插入注释来解释一个程序的重要小节或者任何不清楚的地方。

自测题答案

1. ```
   int feet = 0, inches = 0;
       int feet(0), inches(0);
   ```

2. ```
   int count = 0;
       double distance = 1.5;
   ```

   也可以使用:

   ```
       int count(0);
       double distance(1.5);
   ```

3. ```
   sum = n1 + n2;
   ```

4. ```
   length = length + 8.3;
   ```

5. ```
   product = product * n;
   ```

6. 这种程序的实际输出取决于系统和系统的使用历史。

   ```
   #include <iostream>
   using namespace std;
   int main()
   {
   int first, second, third, fourth, fifth;
   cout << first << " " << second << " " << third << " " << fourth << " " <<
   fifth << endl;
       return 0;
   }
   ```

7. 没有标准答案，以下答案仅供参考:

   a. speed
   b. payRate
   c. highest 或 maxScore

8. ```
   cout << "The answer to the question of\n"
      << "Life, the Universe, and Everything is 42.\n";
   ```

9. ```
   cout << "Enter a whole number and press return: ";
       cin >> theNumber;
   ```

10. ```
    cout.setf(ios::fixed);
        cout.setf(ios::showpoint);
        cout.precision(3);
    ```

11. ```
    #include <iostream>
    using namespace std;
    int main()
    {
        cout << "Hello world\n";
        return 0;
    }
    ```

12. ```
    #include <iostream>
    using namespace std;

    int main()
        {
        int n1, n2, sum;
        cout << "Enter two whole numbers\n";
        cin >> n1 >> n2;
    ```

```
    sum = n1 + n2;
    cout << "The sum of " << n1 << " and "
         << n2 << " is " << sum <<endl;
    return 0;
}
```

13. `cout << endl << "\t";`

14.
```
#include <iostream>

using namespace std;
int main()
   {
   double one(1.0), two(1.414), three(1.732), four(2.0), five(2.236);
   cout << "\tN\tSquareRoot\n";
   cout << "\t1\t" << one << endl
        << "\t2\t" << two << endl
        << "\t3\t" << three << endl
        << "\t4\t" << four << endl
        << "\t5\t" << five << endl;
   return 0;
   }
```

15.
```
3*x
3*x + y
(x + y)/7
(3 * x + y)/(z + 2)
```
注意，`x + y/7` 是错误的。

16. `bcbc`

17. `(1/3) * 3 is equal to 0`

由于 1 和 3 是 int 类型，操作符/执行整数除法，会丢弃余数。因此，1/3 的值为 0，而不是 0.3333…。这使整个表达式变成 `0 * 3`，终值为 0。

18.
```
#include <iostream>
using namespace std;

int main()
{
    int number1, number2;
    cout << "Enter two whole numbers: ";
    cin >> number1 >> number2;
    cout << number1 << " divided by " << number2
         << " equals " << (number1/number2) << endl
         << "with a remainder of " << (number1%number2)
         << endl;
    return 0;
}
```

19. a. `52.0`
b. 9/5的值为 int 值 1。分子和分母都是 int，所以执行整数除法，小数部分会被丢弃。
c. `f = (9.0 / 5) * c + 32.0;` 或 `f = 1.8 * c + 32.0;`

20. `030406`

这些字符串用操作符+连接到一起。

21.
```
if (score > 100)
    cout << "High";
    else
    cout << "Low";
```

可能要在以上双引号字符串的末尾添加\n，具体视程序的其他细节而定。

22.
```
if (savings >= expenses)
{
    savings = savings - expenses;
    expenses = 0;
    cout << "Solvent";
```

```
}
else
{
    cout << "Bankrupt";
}
```

可能要在以上双引号字符串的末尾添加\n，具体视程序的其他细节而定。

23.
```
if ( (exam >= 60) && (programsDone >= 10) )
      cout << "Passed";
    else
    cout << "Failed";
```

可能要在以上双引号字符串的末尾添加\n，具体视程序的其他细节而定。

24.
```
if( (temperature >= 100) || (pressure >= 200) )
      cout << "Warning";
    else
      cout << "OK";
```

可能要在以上双引号字符串的末尾添加\n，具体视程序的其他细节而定。

25. `(x <-1) || (x > 2)`

26. `(1 < x) && (x < 3)`

27. a. 输出 `0 is false`。在讲解类型兼容性的小节中，我们指出 `int` 值 `0` 会转换成 `false`。
    b. 输出 `1 is true`。在讲解类型兼容性的小节中，我们指出非零的 `int` 值会转换成 `true`。
    c. 输出`-1 is true`。在讲解类型兼容性的小节中，我们指出非零的 `int` 值会转换成 `true`。

28.
```
10
7
4
1
```

29. 没有任何输出，因为布尔表达式`(x < 0)`不满足，所以 `while` 语句直接终止，不执行循环主体。

30. 输出和自测题 28 一样。

31. 循环主体在检查布尔表达式之前执行。布尔表达式为 `false`，所以输出结果如下：

```
-42
```

32. 使用 `do-while` 语句，循环主体至少执行一次。使用 `while` 语句，循环主体可能一次都不执行。

33. 这是一个无限循环。输出将从以下内容开始，而且理论上会永远执行下去。

```
10
13
16
19
```

(一旦 `x` 的值超过计算机允许的最大整数，程序可能终止，或者产生一些奇怪的行为。但理论上这是一个无限循环。)

34.
```
#include <iostream>
using namespace std;

int main()
{
    int n = 1;
    while (n <= 20)
    {
        cout << n << endl;
        n++;
    }
    return 0;
}
```

```
35. if (x < 0)
    {
       x = 7;
       cout << "x is now positive.";
    }
    else
    {
       x = -7;
       cout << "x is now negative.";
    }
```

36. 第一行是注释，编译器会将其忽略。所以整个输出如下所示：

```
Self-Test Exercise
```

37.
```
#include <iostream>
using namespace std;

int main()
{
   const double LITERS_PER_GALLON =3.78533;
   double gallons, liters;

   cout << "Enter the number of gallons:\n";
   cin >> gallons;

   liters = gallons* LITERS_PER_GALLON;
   cout << "There are " << liters << " liters in "
      << gallons << " gallons.\n";
   return 0;
}
```

编 程 练 习

编程练习一般只需写很小的程序，运用本章提到的编程概念。

1. 1 公吨(metric ton)等于 35273.92 盎司(ounce)。写一个程序，以盎司为单位读入一盒麦片(cereal)的重量，换算成公吨之后，输出这个重量，同时输出总共需要多少盒麦片才能凑足 1 公吨。你的程序应该允许用户任意重复计算。

2. 计算数字 *n* 的平方根的巴比伦算法如下所示：

   步骤 1：猜一个答案(第一次可以猜 *n/2*).，将值赋给 *guess*

   步骤 2：计算 *r = n / guess*

   步骤 3：*guess = (guess + r) / 2*

   步骤 4：回到步骤 2 重复。步骤 2 和 3 重复得越多，*guess* 就越接近 *n* 的平方根。

   写程序要求用户输入代表 *n* 的 `double` 值，使用巴比伦算法迭代 100 次。一个更有挑战性的版本是加入判断，如果当前 *guess* 和上一次 *guess* 相差不到 1%就输出答案。

3.  视频讲解：*Solution to Practice Program 2.3*

   许多跑步机在控制面板上显示英里/小时(mph)速度，但大多数跑步爱好者都用“步速”(pace)来衡量自己的跑步速度。步速一般是指跑一英里用了多少分钟多少秒，而不是使用 mph 作为单位。

   写程序以 mph 为单位输入一个值，将其换算为跑一英里的分秒数。例如，假定输入 6.5 mph，输出结果应该是跑一英里用了 9 分钟 13.8 秒。将 `double` 值转换为 `int`(丢弃小数点后的任何值)请使用：

   ```
   intValue=static_cast<int>(dblVal);
   ```

4. 写程序玩 Mad Lib 游戏。提示用户输入以下字符串：

- 你的老师的姓或名(注意要么输入姓，要么输入名)
- 你的姓或名(注意要么输入姓，要么输入名)
- 一种食物
- 100 到 120 之间的一个数字
- 一个形容词
- 一种颜色
- 一种动物

输入这些字符串后，它们被代入下面的故事，并输出到控制台：

```
Dear Instructor,[老师的姓或名]

I am sorry that I am unable to turn in my homework at this time. First, I ate a rotten [食物], which made me turn [颜色] and extremely ill. I came down with a fever of [100～120]. Next, my [形容词] pet [动物] must have smelled the remains of the [食物] on my homework, because he ate it. I am currently rewriting my homework and hope you will accept it late.

Sincerely,
[你的姓或名]
```

如果愿意写一个中文版的程序，请使用以下故事模板[①]：

```
亲爱的[老师姓名]老师：
    对不起,我暂时交不了作业。我先是吃了一个烂掉的[食物],我全身都变[颜色]了,感觉很糟糕,现在发烧[100~120]度。然后，我的[形容词]宠物[动物]肯定是闻到了作业本上的[食物]的味道，因为它居然把作业本给吃掉了。现在我正在重新写作业，希望晚些时候能交上来。

你的忠实的[你的姓名]
```

5. 以下程序计算给定半径的球体的体积。能编译和运行，但不符合 2.5 节推荐的编程风格。请根据本章描述的风格重写程序，添加缩进和注释，并使用合理命名的常量。

```
#include <iostream>
using namespace std;
int main() {
    double radius, vm;
    cout << "输入球体半径: " << endl; cin >> radius;
    vm = (4.0 / 3.0) * 3.1415 * radius * radius * radius;
    cout << " 体积是: " << vm << endl;
    return 0;
 }
```

编 程 项 目

编程项目要求综合运用多方面的知识来解决问题，程序一般比编程练习大，解题方式多样化。

1. 一家政府研究机构的研究表明，减肥碳酸饮料中常用的一种人造甜味剂会导致实验室里的小白鼠死亡。你的一位朋友很想减肥，不肯放弃可能导致死亡的碳酸饮料。她想知道，在不危及生命的情况下，可以喝多少碳酸饮料。写一个程序来提供答案。需要向程序输入让一只小白鼠致死所需的人造甜味剂重量(5 克)、小白鼠体重(35 克)和节食者预期的体重(如输入磅数，记住 1 磅等于 454 克)。假定致鼠死亡的剂量和致人死亡的剂量成正比。一罐碳酸饮料的重量假定是 350 克。为保证朋友的安全，程序一定要提示输入停止减肥时的体重，而不要输入当前体重。假定碳酸饮料含 1‰(千分之一)人造甜味剂，在表示这个比例的变量声明中使用修饰符 const。注意，可能需要将这个比例表示成 double 值 0.001。程序应该允许用户任意重复计算。

① 中文姓名可以全部输入，比如可以输入“张三”，而不必像英文版那样，只能输入 Zhang 或者 San。但无论如何，姓和名之间不能有空格。——译注

2. 某公司决定为员工普涨 7.6%的工资，同时按新标准补发前 6 个月工资。写程序输入员工去年年薪，输出应补发金额、新的年薪和新的月薪。在表示涨幅的变量声明中使用修饰符 const。程序应该允许用户任意重复计算。

3. 修改编程项目 2 的程序，让它能计算补发任意月数的工资，而不是固定补发 6 个月。具体月数由用户输入。

4. 消费贷款的计算并不一定简单。有一种贷款方式称为“贴现式分期贷款”(discount instalment loan)，它的计算方式如下：假设贷款金额为 1000 美元，年利率为 15%，贷款期限为 18 个月。贷款金额 1000 美元乘以 0.15，年利息是 150 美元。它乘以贷款期限 1.5 年(18 个月)，总利息是 225 美元。这个金额立即从 1000 美元中扣除，消费者实际拿到手上的贷款只有 775 美元。还贷时，消费者还是以贷款金额为准，每月偿还相同的金额。所以，月还贷额是 1000 美元除以 18，即 55.56 美元。如果消费者需要的恰好是 775 美元，那么很快就能完成计算。但是，如果消费者需要的是 1000 美元，人工计算就显得比较繁琐。请写一个程序来获取三项输入：消费者想实际拿到手的金额、利率和贷款期限(以月为单位)。程序要计算需要多大的贷款金额，消费者才能拿到所需的金额。程序还应该计算月还款额。程序应该允许用户任意重复计算。

5. 写程序判断一间会议室是否违反了消防条例中有关房间最大容积(最多容纳人数)的条款。程序将读取房间最大容积和与会人数。如果人数小于或等于最大容积，程序宣布会议合法，并指出还能容纳多少人。人数超过最大容积，程序宣布依据消防条例，会议不能举行，并指出要减去多少人才能符合消防条例的要求。可为这个程序写一个更复杂的版本，允许用户任意重复计算。如果这是课堂作业，请询问教师是否需要写复杂版本。

6. 某公司规定，在每周的正常上班时间中(40 小时)，员工工资标准是每小时 16.78 美元。超过正常时间算加班。每加班 1 小时，工资是正常标准的 1.5 倍。在员工的应发工资中，缴纳 6%作为社会保险税，14%作为美国联邦税，5%作为美国州税，再加上每周 10 美元的工会会费。假如员工有 3 名或更多家属，还要多缴 35 美元的额外医疗保险费。写程序读取员工在某一周的工作时数及其家属人数，输出员工在那一周的应发工资、每一笔代扣金额和最后拿到手的实发工资。可为此程序写一个更复杂的版本，允许用户任意重复计算。如果这是课堂作业，请询问教师是否需要写复杂版本。

7. 制定跨度为几年的预算并不容易，因为价格并非一成不变。如果你的公司每年需要 200 支铅笔，就不能简单地根据今年的价格来计算今后两年的铅笔开支。由于通货膨胀，开支可能比现在高。写程序来估算在指定的年数之后，某件商品的预计开支。程序要求输入今年购买每一件商品的开支、从现在起的年数和通货膨胀率。然后，程序输出在经过那么多年之后，每一年购买同一件商品所需的开支。要求用户以百分数的形式输入通货膨胀率，比如 5.6(它代表 5.6%)。随后，程序将百分数转换成小数，如 0.056，并用一个循环来估算根据通货膨胀率而得出的开支(**提示：**这类似于计算信用卡账户的利息，详情参见本章前面的讨论)。

8. 假设分期贷款购买一套立体声音响，价格 1000 美元，还贷计划如下：无首付款，年利率 18%(月利率 1.5%)，每月还款 50 美元。在这 50 美元中，扣除利息之后，剩下的金额用于偿还本金。所以，第一个月支付的利息是 1000 美元的 1.5%，即 15 美元，偿还的本金则为 35 美元。现在，贷款额从 1000 美元变成 965.00 美元。第二个月支付的利息为 965.00 美元的 1.5%，即 14.48 美元。50 美元减去 14.48 美元得到 35.52 美元，这是第二个月偿还的本金。写一个程序，计算还清贷款的期限和在这期间支付的总利息额。使用一个循环来计算利息和每月还款之后的剩余贷款(最终的程序不需要输出每月应支付多少利息和剩余贷款，但你可考虑写程序的一个增强版本，从而输出这些值)。使用一个变量对循环迭代进行计数，它的终值就是还清贷款(剩余贷款减至 0)所需的月数。可能还需要使用其他变量。如果剩余贷款已经不多，那么最后一笔还款可能小于 50 美元(但不要忘记还有利息)。如果尚余 50 美元的贷款额，那么月付 50 美元还不能还清贷款，虽然此时已非常接近还清。注意，对于 50 美元的贷款，月利息仅为 75 美分。

9. 写程序读入 10 个整数，输出大于 0 的所有整数(正数)之和、小于 0 的所有整数(负数和 0)之和与所有整数(无论正数、负数还是 0)之和。用户可一次性输入这 10 个整数，而且可以采用任何顺序。程序不应该要求用户单独输入正数和负数。

10. 修改编程项目 9 的程序，输出所有正整数之和、所有正整数的平均数、所有负整数之和、所有负整数的平均数、所有正整数和负整数之和以及所有数的平均数。

11. 声音在空气中通过分子之间的碰撞来传输。气温影响分子速度，进而影响音速。干燥空气中的音速大致可以使用以下公式：

$$v \approx 331.3 + 0.61 \times T_c$$

其中的 T_c 是以摄氏度为单位的空气温度，而 v 是以 m/s 为单位的音速。

写程序允许用户输入一个起始和结束温度。在这个温度范围内，程序应输出每一档温度和对应的音速，每一档都提高 1℃。例如，假定起始和结束温度分别是 0℃和 2℃，那么程序输出如下：

英文版：
At 0 degrees Celsius the velocity of sound is 331.3 m/s
At 1 degrees Celsius the velocity of sound is 331.9 m/s
At 2 degrees Celsius the velocity of sound is 332.5 m/s

中文版：
0℃时，音速为 331.3 m/s
1℃时，音速为 331.9 m/s
2℃时，音速为 332.5 m/s

12.  视频讲解：*Solution to Programming Project 2.12*

许多私人打的水井每分钟只能出 1 或 2 加仑的水。为了防止没水的情况，一般的办法是安装临时储水舱。一家四口每日用水 250 加仑。但是，井本身就是一个“自然”的储水舱。挖得越深，储水越多，甚至会多过家庭的用水量。但是，具体有多少水可供使用？

写程序让用户以英寸为单位输入水井半径(典型水井半径为 3 英寸)，以英尺为单位输入水井深度(假设水充满整个深度，虽然这实际上不可能，因为水平面一般低于地平面 50 英尺以上)。程序输出水井的储水量(加仑)。为方便你参考，圆柱体体积公式是$\pi r^2 h$，其中 r 是半径，h 是高度。另外，1 立方英尺=7.48 加仑，1 英尺=12 英寸。

例如，半径 3 英寸、深 300 英尺的水井在充满水的情况下，可容纳 441 加仑的水，足够一家四口使用，不需要安装单独的储水缸

13. Harris–Benedict 公式估算在不锻炼的情况下保持体重所需的热量(卡路里)。这称为基础代谢速率(basal metabolic rate，BMR)。

女性保持体重所需的热量使用以下公式：

BMR = 655 + (4.3 × 体重磅数) + (4.7 × 身高英寸数) – (4.7 × 年龄)

男性保持体重所需的热量使用以下公式：

BMR = 66 + (6.3 × 体重磅数) + (12.9 × 身高英寸数) – (6.8 × 年龄)

普通巧克力条提供 230 卡路里热量。写程序让用户输入体重(以磅为单位，100 磅等于 45.4 公斤)、身高(以英寸为单位，1 英寸等于 2.54 厘米)和年龄。输入字符 M 代表男性，输入 F 代表女性。输出每天要吃多少巧克力条才能保持体重。

14. 写程序计算 N 次课堂作业的总成绩，然后用百分数输出该成绩。用户首先输入 N 的值，表明总共有多少次课堂作业。接着输入每次作业的实际成绩(Score received for exercise)和满分成绩(Total points possible for exercise)。所有作业的实际成绩除以满分成绩，获得总成绩。以百分数形式输出。下面是示例输出：

```
How many exercises to input? 3

Score received for exercise 1: 10
Total points possible for exercise 1: 10
```

```
Score received for exercise 2: 7
Total points possible for exercise 2: 12

Score received for exercise 3: 5
Total points possible for exercise 3: 8

Your total is 22 out of 30, or 73.33%.
```

15. 为适应气候变化，建筑必须考虑热膨胀效应。例如，金属顶梁高温膨胀。附加应力可能造成建筑受损。类似地，材料低温收缩。对于一种能自由膨胀的材料，其长度的线性变化公式如下所示：

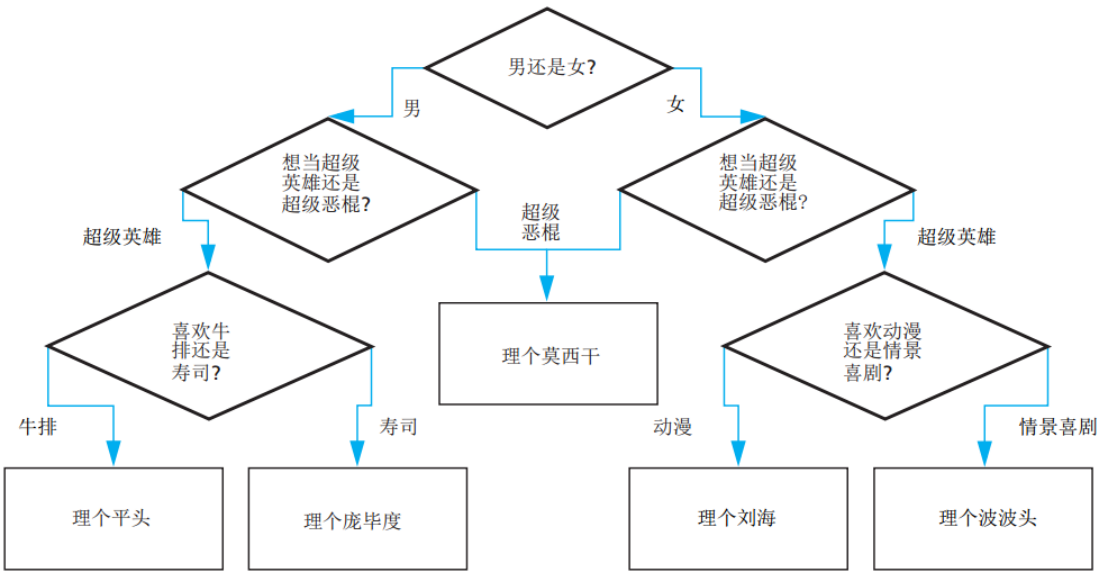
$$L_\Delta = \alpha L_0 T_\Delta$$

其中，L_0是材料初始长度(米)，L_Δ是线性位移(米)，T_Δ是温度变化(℃)，而α是线性热膨胀系数。

写程序输入α，L_Δ和T_Δ，计算并输出线性位移。如位移为正，就输出“The material will expand by *x* meters”(将膨胀 *x* 米)。为负就输出“The material will contract by *x* meters”(将收缩 *x* 米)，此时的 *x* 应显示成负数。下面列出了一些常见材料的α值：

铝	2.31×10^{-5}
铜	1.70×10^{-5}
玻璃	8.50×10^{-6}
钢	1.20×10^{-5}

16. 下图通过一系列问题决定发型。写程序问问题并输出推荐发型[①]。



① 译者特意为好奇的读者提供以下参考。另外，典型情景喜剧包括《生活大爆炸》和《破产姐妹》等。

莫西干

头顶两边光秃秃，中间头发向上翘起好似马鬃

平头

从脑后到两鬓的头发全部推光，上端头发稍长齐平

庞毕度

刘海向后梳，前面头发微微膨起

刘海

从前额垂下一些头发，遮住额头或脸部的一部分

波波头

齐耳短发