# 第3章 自由函数的运用

**前章回顾**

我们现在应该已经在自己的系统中亲身体验了这门语言的语法、报错信息及其程序开发从头到尾的过程。而且，本书前几章中的大部分编程项目实施的都是IPO这个算法模式，相信读者现在应该也可以在实践中按照正常的顺序安排这三个步骤，并能理解省略其中某个步骤或以某种混合顺序执行的情况了。

**本章提要**

为了节省时间和资金，软件开发者通常会选择利用现有的软件来完成他们的工作。在本章，我们将介绍重用现有软件的其中一种方式，程序员们通常会将这些经过大量测试的软件作为自身工作的为起点。我们将学习如何通过函数的头信息（function headings）了解现有函数的用法，并通过阅读这些函数用法中的前置和后置条件来确定它们的功能。在本章的最后，我们还会列出一些你可能会遇到的错误种类。我们希望在完成本章的学习之后，你将：

* 学会使用一些数学函数和三角函数来进行求值运算。
* 学会在调用函数时使用参数。
* 理解为什么程序员们要将软件划分成一系列函数。
* 学会通过函数的头信息来了解现有函数的用法。

## 3.1 cmath函数

C++为我们定义了大量可用于双精度浮点运算的数学函数和三角函数。下面是其中的两个：

sqrt(x) // Return the square root of x  
pow(x,y) // Return x to the yth power

如你所见，这些函数的*调用*都是通过指定函数名称，并紧随其后的括号中加上指定数量和类型的*参数*来执行的。这样我们就得到了一般函数调用的通用格式：

**通用格式 3.1**：*函数调用*

function-name(*arguments*)

在这里，*function-name*是一个已被声明的标识符，它代表的是目标函数的名标。而*arguments*则通常是一组由逗号分隔的零个或多个表达式。譬如在下面这个函数调用中，函数的名称是sqrt（即平方根），它的参数为81.0：

sqrt(81.0) // An example of a function call

函数通常会有零个或多个参数。尽管大多数数学函数都只需要一个参数，但也会有像pow()这种需要两个参数的函数。在接下来的这个函数调用中，函数的名称是pow（即power），参数是底数和指数。也就是说，函数调用pow(base, power)代表的就是指数运算：

double base = 2.0;  
double power = 3.0;  
cout << pow(base, power); // Output: 8.0

在函数调用的过程中，我们使用的所有参数都必须是一个该函数可接受类型的表达式。例如sqrt("Bobbie")这个调用就会出错，因为该参数并不属于数字类型。

另外，我们提供给函数的参数还必须要合乎要求。例如sqrt(-4.0)这个函数调用就可能是有问题的，因为-4.0并不在sqrt函数所要求的域中，平方根函数并没有为负数值做出相因的定义。也就是说，事实上sqrt函数只有在某些特定条件得到满足时才会正常执行。对于sqrt函数而言，这个条件就是它的参数值必须大于或等于0.0。下面，我们再来看一下cmath函数库中还有哪些可用的数学函数和三角函数。

**cmath函数库中的部分函数** *提示：下面的double代表的是函数的返回值类型*

|  |  |  |  |
| --- | --- | --- | --- |
| 函数 | 返回值说明 | 调用示例 | 示例结果 |
| double ceil(double x) | 返回大于等于x的最小整数 | ceil(2.1) | 3.0 |
| double cos(double x) | 返回角度x的余弦值 | cos(1.0) | 0.5403 |
| double fabs(double x) | 返回x的绝对值 | fabs(-1.5) | 1.5 |
| double floor(double x) | 返回小于等于x的最大整数 | floor(2.9) | 2.0 |
| double pow(double x, double y) | 返回 | pow(2, 4) | 16.0 |
| double sin(double x) | 返回角度x的正弦值 | sin(1.0) | 0.84147 |
| double sqrt(double x) | 返回x的平方根 | sqrt(4.0) | 2.0 |

程序员只需要在#include <iostream>这条语句的上面加上一条#include <cmath>，就可以成功调用到cmath函数库中声明的函数了。也就是说，下面这段程序将会被成功编译：

// Show some mathematical functions available from cmath  
#include <cmath> // For fabs, ceil, floor, and pow  
#include <iostream> // For cout using namespace std;  
  
int main() {  
 double x = -2.1;  
 cout << "fabs(-2.1): " << fabs(x) << endl  
 << "ceil(-2.1): " << ceil(x) << endl  
 << "floor(-2.1): " << floor(x) << endl  
 << "pow(-2.1, 2.0): " << pow(x, 2.0) << endl;  
 return 0;  
}

**程序输出**

fabs(-2.1): 2.1  
ceil(-2.1): -2  
floor(-2.1): -3  
pow(-2.1, 2.0): 4.41

需要提醒的是，这些cmath函数的参数也都可以使用整数表达式来调用。和赋值操作一样，它们会执行将整数值升格成double类型，所以sqrt(4)与sqrt(4.0)会返回相同的结果，不会有任何错误。

**自检题**

3-1 请求取pow(4.0, 3.0)的值。

3-2 请求取pow(3.0, 4.0)的值。

3-3 请求取floor(1.6 + 0.5)的值。

3-4 请求取ceil(1.6 - 0.5)的值。

3-5 请求取fabs(1.6 - 2.6)的值。

3-6 请求取sqrt(16.0)的值。

## 3.2 使用cmath函数解决问题

*问题：* 请编写一个能将指定数字四舍五入到某个小数位的程序。例如，将3.4589四舍五入到两位小数就是3.46，而四舍五入到一位小数就是3.5。

### 3.2.1 分析

下面我们按照软件开发的分析-设计-实现的步骤来进行，先进行如下分析动作：

1. 阅读并理解目标问题。
2. 定义用来表示答案的对象——即输出。
3. 定义用为获得答案必须键入相关内容的对象——即输入。
4. 编写测试用例（我们在上面已经提供了两个）。

### 3.2.2 设计

设计阶段的任务是提出算法。我们在这里可以借助IPO算法模式先开发出给予伪代码的算法。为了方便后续讲解，我们在这里先复习一下这个算法模式：

|  |  |
| --- | --- |
| 模式： | 输入/处理/输出（IPO） |
| 问题： | 程序需要基于用户的输入来计算并显示我们所需的信息。 |
| 纲要： | 1. 获取输入数据。 2. 用某种有意义的方式处理数据。 3. 输出结果。 |
| 示例： | 请参考我们接下来对于将x四舍五入到n位小数这个问题的描述。 |

IPO算法模式可以很好地帮助我们安排出良好的操作顺序。在这里，算法代表的是一般性设计，它是解决方案的纲要。为了让算法看起来更为详尽一些，我们还在其中的两处引入了先提示再输入模式，具体如下：

1. 提示用户指定要进行四舍五入的数字（并将其命名为x）。
2. 用户输入x的值。
3. 提示用户指定目标的小数位数（并将其命名为n）。
4. 用户输入n的值。
5. 将x四舍五入到n位小数。
6. 显示x被修改之后的值。

步骤1、2、3、4和6用C++实现起来都非常简单，它们直接用输入/输出语句就可以实现了。但第五步“将x四舍五入到n位小数”描述得还不够详尽。下面我们要继续改进第五步的设计。在其余部分不需要再考虑的情况下，我们现在可以将注意力集中到如何将x四舍五入到n位小数这个更难的问题上来了。这部分的解决方案的确会有点没法，下面我们来看其中一种方法。

为了将数字x四舍五入到n位小数，首先我们要让x先乘以。然后再将新的x值加上0.5。接着再对x调用floor(x)。最后再让x除以。于是，上述算法就多增加了以下四个步骤：

1. 提示用户指定要进行四舍五入的数字（并将其命名为x）。
2. 用户输入x的值。
3. 提示用户指定目标的小数位数（并将其命名为n）。
4. 用户输入n的值。
5. 将x四舍五入到n位小数的步骤如下：
   1. 将x的值修改成。
   2. 将x的值加上0.5。
   3. 将x的值修改成floor(x)。
   4. 将x的值修改成。
6. 显示x被修改之后的值。

下面我们来模拟跟踪一下程序执行的过程，看看当x为3.4567时，它是如何被四舍五入到两位小数的。

**将3.4567四舍五入到两位小数的过程：**

**自检题**

3-7 请模拟跟踪同一个算法在不同问题示例下的运行过程，当我们要将9.99四舍五入到一位小数是，其结果会是什么？请在下列空白处写下对应的新的x的值（即x在输入之后的四次变化）。

|  |  |  |
| --- | --- | --- |
| 算法步骤 | x | n |
| 1. 提示用户指定要进行四舍五入的数字（并将其命名为x）。 | ? | ? |
| 2. 用户输入x的值。 | 9.99 | ? |
| 3. 提示用户指定目标的小数位数（并将其命名为n）。 | 9.99 | ? |
| 4. 用户输入n的值。 | 9.99 | 1 |
| 5. 将x的值修改成。 | \_\_\_\_\_ | 1 |
| 6. 将x的值加上0.5。 | \_\_\_\_\_ | 1 |
| 7. 将x的值修改成floor(x)。 | \_\_\_\_\_ | 1 |
| 8. 将x的值修改成。 | \_\_\_\_\_ | 1 |
| 9. 显示x被修改之后的值。 | \_\_\_\_\_ | 1 |

### 3.2.3 实现

下面，我们来看上述算法被转换成完整C++源码之后的版本。请留意一下源代码中的注释，我们在其中标出了各算法步骤在转换成c++之后的位置。

// Round a given number to a specific number of decimal places  
  
#include <iostream> // For cin and cout   
#include <cmath> // For pow(10, n) and floor(x)  
using namespace std;  
  
int main() {  
 // Declare objects identied during analysis  
 double x = 0.0;  
 double n = 0.0;  
 // Algorithm step number:  
 // Input  
 cout << "Enter number to round : "; // 1.  
 cin >> x; // 2.  
 cout << "Enter number of decimal places : " ; // 3.  
 cin >> n; // 4.  
  
 // Process (Round x to n decimals)  
 x = x \* pow(10, n); // 5a.  
 x = x + 0.5; // b.  
 x = floor(x); // c.  
 x = x / pow(10, n); // d.  
  
 // Output (Display the modied state of x)  
 cout << "Rounded number : " << x << endl; // 6.  
  
 return 0;  
}

**程序会话**

Enter number to round : ***3.4567***  
Enter number of decimal places : ***2***  
Rounded number : 3.46

**自检题**

3-8 请针对上述程序再列举三个测试用例。

3-9 在上述程序中，当用户输入x为3.15，n为1时，x的最终值是多少？

3-10 请根据第56页列出的“cmath函数库中的部分函数”，找出一个稍微有些不同的算法来完成相同的任务，只不过这次3.15四舍五入到一位小数的结果应该是3.1而不是3.2。（*提示：* 我们可以考虑将“加上0.5”这个步骤改成“减去0.5”。）

3-11 请写出以下函数调用的返回值。

* 1. pow(2.0, 4.0)
  2. sqrt(16.0)
  3. ceil(-1.7)
  4. floor(1.0)
  5. fabs(-23.4)
  6. pow(4.0, 2.0)

## 3.3 调用已被文档化的函数

在C++中，所有的函数都必须先要声明函数头信息，然后编译器才能确定相关的函数调用操作是否正确。另外，这些函数的头信息也能帮助程序员正确地调用他们。譬如，如果我们仔细阅读一下cmath这个文件，就会看到许多这样的函数头信息。

在这一节，我们就来重点介绍一下应该如何阅读这些函数的头信息，以及如何用其他说明文档来了解相关函数的预期和所要执行的动作，我们通常分别将它们称之为函数的前置条件与后置条件。

### 3.3.1 前置条件与后置条件

为了让函数能够正常执行它的操作，我们通常都会为其预设一些使用条件。譬如，对于sqrt函数来说，其预置条件就是要求函数参数是一个大于或等于0.0的数。函数的*前置条件*通常都是一些过于调用参数的一些假定。如果这些前置条件得不到满足，所有的预置动作都会被取消——这时函数行为将是未定义的，有些系统会以算术溢出错误终止程序，也有些系统会返回一个类似-1.#IND或NaN这样的只，告诉我们该值“不是一个数字”。总而言之，如果我们想让自己的函数调用得到可预期的结果，就必须要满足它的这些前置条件。

使用条件的另一部分叫做*后置条件*——这部分描述的是在前置条件得到满足的情况下函数会执行的动作。函数的前置条件和后置条件通常会被写入到该函数的文档部分中。例如下面就是sqrt这个函数关于其前置条件和后置条件的文档。

double sqrt(double x)  
// precondition: x is not negative (x >= 0)  
// postcondition: Square root of x replaces the function call

如你所见，注释说明了参数必须是一个大于或等于0.0的数字。如果该前置条件得到了满足，该参数的平方根就会被返回给*客户*——即调用该函数的代码。反之，结果将是未定义的。

|  |  |
| --- | --- |
| 函数调用 | 返回的结果 |
| sqrt(4.0) | 前置条件得到满足，返回结果为2.0。 |
| sqrt(-1.0) | 前置条件未被满足，该函数调用将返回NaN（即非数字）。 |

当然，要想在调用函数之后获得预期的结果还有一个隐藏的前置条件，那就是客户代码必须提供正确的参数类型。例如，ceil函数需要的是一个double类型的参数，这意味着我们提供的参数必须要能被转换成double类型，包括short、int、float以及char。例如。ceil函数不接受string类型的参数，这可以说四个显而易见的先决条件：

double ceil(double x)  
// precondition: Argument must be convertible to a double  
// postcondition: Return the smallest integer >= x

但是，这类信息通常会被隐藏在参数声明中，编译器会根据声明自行检测出不正确的的参数。因此我们不会将这些内容写入前置条件。

前置条件的内容通常不是编译器能检测到的，例如，下面的程序在语法上是完全正确的：

cout << sqrt(-1.0); // Return depends on the system in use

从现在开始，我们会将前置条件的标签缩写成pre:，而后置条件则用post:来表示，这样的话，我们就可以上面同一个函数（ceil）的文档写成这样：

double ceil(double x) // post: Return the smallest integer >= x

当然，我们应该留意一下pre:和post:的用法，它们不一定非要放在函数头信息之后。不同的人对函数文档化的方式是不一样的，这里只是本书所采用的风格。

### 3.3.2 函数头信息

前置条件和后置条件可以帮助程序员确定相关函数的正确用法。如果我们想将这些作为文档提供给用户，它们通常会被列在函数的头信息之后。函数的头信息也是用于说明函数返回值和所需参数的非常重要的信息。下面我们来看一些函数头信息的通用格式：

**通用格式 3.2**：*函数头信息*

*return-type function-name* (*parameter-1, parameter-2, parameter-n*)

在这里，*return-type*可以是任何有效的C++类型或关键字void。函数的返回值为void时就表示它没有返回值。另外，在括号( )之间的参数可以是值类型参数、引用类型参数或者const的引用类型参数和值类型参数。下面我们先来看值类型参数。

一个函数通常都会有一个或多个参数，通过添加值类型参数来让我们可以将值传递给函数的格式如下：

**通用格式 3.3**：*值类型参数*

*class-name identifier*

**标准C++函数的头信息示例：**

int isapha(int c)  
int tolower(int c);  
double round (double x);  
double remainder(double numerator, double denom);

如你所见，函数头信息指定了函数的返回值类型、函数的名称以及其要程序员体的参数数量。函数的参数类型也通过括号之间各参数的*class-name*部分做了指定。例如，由于下面的pow函数的参数x和y被声明成了double类型，所以我们可以确定调用pow函数的每个参数都必须是double类型，或者至少可以转换成double类型的，譬如整数类型。

double pow(double x, double y)  
// pre: When y has a fractional part, x must be positive  
// When y is an integer, x may be negative  
// post: Returns x to the yth power

另外不要忘了，这里还声明了函数名是pow，返回值类型是double。

虽然这里并没有提供pow函数的完整实现，但前置条件、后置条件以及函数头信息所提供的信息已经足以有效地帮助人们了解该函数的用法了。

总而言之，函数头信息连同该函数的前置条件与后置条件一起为我们提供了以下信息：

1. *return-type*告诉了我们函数返回值的类型。
2. *function-name*告诉了我们如何启动一个有效的函数调用。
3. *parameter-list*告诉了我们在执行函数调用时要用到的参数数量以及这些参数的类型。
4. pre:告诉了我们在执行函数调用之前应满足哪些要求。
5. post:告诉了我们当一个函数的前置条件被满足之后，它会做哪些事。

除了为程序员提供这些信息之外，函数头信息也向编译器提供了用于验证函数调用是否有效的信息。如果相关函数的调用不正确，编译器会根据这些信息来通知我们。譬如我们以floor函数的头信息为例：

double floor(double x)  
// post: Returns the largest integer <= x

如你所见，该函数的返回值类型是double。这意味着所有double对象所在的地方都可以放置一个对floor函数的有效调用。因此这个函数调用可以被当作double对象来使用——譬如使用在算术表达式中，这是符合语法的。除此之外，函数名floor也是一个非常重要的信息，它能帮助我们调用并指定目标函数。最后，参数列表中是一个名为x的double类型参数，这决定了该函数的调用代码必须提供一个数字参数才能正确地调用floor函数。例如，我们在下面的代码中看到的是关于如何将该函数的有效调用赋值给double对象的正确示范：

double x;  
x = floor(5.55555); // This assignment is okay

但以下这些函数调用就属于无效操作了：

string s;  
s = floor(5.5555); // Error: floor doesn't return a string  
cout << floor(1.0, 2.0); // Error: too many arguments  
cout << floor("wrong type"); // Error: wrong type argument  
cout << floor(); // Error: too few arguments

**自检题**

3-12 请根据给定的函数头信息，写出以下”有效“的正确函数调用，并解释其他调用无效的原因。

double ceil(double x)

* 1. ceil(1.1)
  2. floor(2.9)
  3. ceil(1.2, 3.0)
  4. ceil("Ceila")
  5. ceil -0.1
  6. ceil(-3)

3-13 请说明以下函数头信息中各自所犯的错误：

* 1. double f ( x )
  2. int smaller(int n1 int n2)
  3. toUpper(string s)
  4. myClass g()
  5. int twoStrings(string s1, string s2,)
  6. unknownType initialize(" lename.dat")

请根据以下文档回答下面的问题：

double floor(double x)  
// post: The oor function returns a oating-point value  
// representing the largest integer that is less than or  
// equal to x

3-14. 请写出四个（参数不相同的）的函数调用，帮助从未见过floor函数的人了解该函数是做什么的。

3-15. 请写出你在上个问题中所写的那四个函数调用返回的值。

### 3.3.3 实参与形参的关联

函数头信息中通常会列出0个、一个、两个参数（有时甚至会是多个）。如果参数的个数超过一个，这些参数之间就必须要用逗号来做区隔。例如，下面的函数头信息中列出了两个参数，分别是str和x。

double twoParameters(string str, double x)

函数头信息中所列出的每个参数（我们称之为形参）都需要有一个相应类型的实参。因此，我们在调用twoParameters函数时必须要提供两个对应的实参，用任何其他数量的实参来调用这个函数都会导致编译器报错。除此之外，实参的类型和位置必须要要和形参的类型和位置一一一对应。例如，double类型的实参不能与string类型的形参相关联。下面我们来看几个正确调用twoParameters函数的示例：

**几个对于twoParameters函数的有效调用：**

twoParameters("abc", 1.2);  
twoParameters("another string", 15);  
twoParameters("$", 3.4);

但以下对twoParameters函数调用就会导致编译时错误了：

|  |  |
| --- | --- |
| 错误调用 | 导致错误的原因 |
| twoParameters("a"); | 该函数需要的是两个实参。 |
| twoParameters("1.1", "2.2"); | 字符串”2.2“不能被赋值给double类型的形参。 |
| twoParameters(1.1, 1.1); | 数字1.1不能被赋值给string类型的形参。 |
| twoParameters("a", 2.2, 3.3); | 实参个数过多。 |
| twoParameters; | 产生一个警告，说明该语句无效。 |

实参与形参在位置上是一一对应的，第一个实参对应第一个形参，第二个实参对应第二个形参，以此类推。例如在调用twoParameters函数时，它的第一个形参被赋值的是第一个实参的值，而传递给该函数的第二个实参将被拷贝给它的第二个形参x。也就是说，当我们用“abc”和1.2这两个实参调用twoParameters函数时，情况是这样的：

int twoParameters(string str, double x)  
 ↑ ↑  
result4 = twoParameters ("abc", 1.2);

整个过程就像执行了如下两个赋值操作：

str = "abc";  
x = 1.2;

接下来，twoParameters函数中发生的一切操作都要依赖于这两个形参的值，毕竟该函数是利用形参来产生返回结果的。

**自检题**

3-16 在twoParameters("1st", 1.2)这个调用中传递给str形参的值是什么？

3-17 在twoParameters("2nd", 3.4)这个调用中传递给x形参的值是什么？

如你所见，当函数的前置条件与后置条件都被满足时，我们就可以从函数头信息中推导出如此多的内容。我们可以再来复习一下，下面是sin函数的完整头信息以及它的前置条件与后置条件：

double sin(double x) // post: Returns the sine of x radians

同样地，我们可以从中得到如下信息：

* 函数会做的事： 返回角度x的正弦值
* 函数返回值类型： double
* 函数的名称： sin
* 函数参数的数量： 一个
* 函数参数的类型： double（或其他可被转换成double类型的表达式）

接下来，我们可以来确定返回结果了（这里可以借助科学计算器的角度模式）。

|  |  |
| --- | --- |
| 函数调用 | 函数返回结果 |
| sin(3.1415926/2.0) | 1.0 |
| sin(1.0) | 0.8421 // Approximately |
| sin(3.1415926) | 5.35898e-08 // close to 0.0 |

**自检题**

3-18 请根据下面pow函数（来自cmath程序库）的完整头信息及其前置条件与后置条件，确认以下事项的信息：

double pow(double x, double y)  
// pre: When y has a fractional part, x must be positive.  
// When y is an integer, x may be negative.  
// post: Returns x to the yth power

* 1. 函数的返回值类型
  2. 函数的名称
  3. 函数参数的数量
  4. 第一个参数的类型
  5. 第二个参数的类型
  6. 第三个参数的类型

3-19. 请写出一个针对pow函数的正确调用。

3-20. pow(-81.0, 0.5)是一个有效的函数调用吗？它的返回值是什么？

3-21. pow(-10.0, 2)是一个有效的函数调用吗？它的返回值是什么？

3-22. pow(2, 5)是一个有效的函数调用吗？它的返回值是什么？

3-23. pow(4.0, 0.5)是一个有效的函数调用吗？它的返回值是什么？

3-24. pow(5.0)是一个有效的函数调用吗？它的返回值是什么？

3-25. 请为一个用第一个数去除以第二个数，并返回其结果的函数编写相应的函数头信息，并为其编写相应的前置条件与后置条件。例如， 在执行remainder(5.0, 2.0)这个调用时它得返回0.5、而当执行remainder(1, 3)时则返回0.3333333。

### 3.3.4 面向int、char和bool这些类型的一些函数

当然，也有一些自由函数是作用于其他基本类型的。例如，本章末尾的编程项目中会用到的一些C++标准库中的只有函数：min、max和abs。

#include <iostream>  
using namespace std;  
  
int main() {  
 cout << min(5, 7) << endl;  
 cout << min(5.5, 7.7) << endl;  
  
 cout << max(5, 7) << endl;  
 cout << max(5.5, 7.7) << endl;  
  
 cout << abs(5 - 7) << endl;  
  
 return 0;  
}

**程序输出**

5  
5.5 7  
7.7  
2

min和max这两个函数的定义方式使其可以被不同类型的参数调用，当然，它们的实参类型必须同时是两个int或两个double，不能混合在一起。

C++中还有一些名称为islower、isdigit的这一类布尔型的方法。另外还有些以char为实参和返回值类型的函数，它们会将当前字母转换成等价的大写或小写形式。我们在通过#include <cctype>指令引入这组函数对相关字母进行分类和转换时都可以参考这些自由函数的函数头信息。

int islower(int c);

该函数会负责检查参数c中的值是否是一个小写字母，似乎该函数的参数应该是char类型，返回值是bool类型才对，像这样：

bool islower(char ch); // This is not the function heading

但在C++中，int与char这两种类型之间是可以相互赋值的，它们在算术运算中是可以彼此混用的。

#include <iostream>  
using namespace std;  
  
int main() {  
 int anInt = 'A'; // 'A' equals 65  
 char aChar = 67; // 67 equals 'C'  
  
 cout << "anInt: " << anInt << endl;  
 cout << "aChar: " << aChar << endl;  
 cout << "aChar + anInt: " << (aChar + anInt) << endl;  
 cout << "anInt % aChar: " << (anInt % aChar - 2) << endl;  
  
 return 0;  
}

**程序输出**

anInt: 65  
aChar: C  
aChar + anInt: 132  
aChar % anInt: 63

而由于C++同时也将true视为1，false为0，这就可能会带来更多的混乱。

#include <iostream>  
using namespace std;  
  
int main() {  
 bool aBool = 1; // C++ allows assignment of int to bool  
 int anotherBool = false; // and a bool literal to an int  
  
 cout << aBool << " " << anotherBool << endl;  
  
 return 0;  
}

**程序输出**

1 0

如你所见，这里输出的true是1，false是0。

如果我们需要对一个字符做“A”或“a”这样大小写区分或者“9”或“3”这样的数字区分，就会用到<cctype>中的自由函数。下面我们就来演示一下这个函数库的使用：

#include <iostream>  
#include <cctype> // For isalpha isblank isdigit  
using namespace std;  
  
int main() {  
 char ch = 'a';  
 cout << "isalpha('" << ch << "')? " << isalpha(ch) << endl;  
 ch = '?';  
 cout << "isalpha('" << ch << "')? " << isalpha(ch) << endl;  
  
 ch = ' ';  
 cout << "isblank('" << ch << "')? " << isblank(ch) << endl;  
 ch = 'N';  
 cout << "isblank('" << ch << "')? " << isblank(ch) << endl;  
  
 ch = 'P'; // Oh, not zero  
 cout << "isdigit('" << ch << "')? " << isdigit(ch) << endl; ch = '5';  
 cout << "isdigit('" << ch << "')? " << isdigit(ch) << endl;  
  
 return 0;  
}

**程序输出**

isalpha('a')? 1  
isalpha('?')? 0  
isblank(' ')? 1  
isblank('N')? 0  
isdigit('P')? 0  
isdigit('5')? 1

另外，我们还可以用toupper、tolower这两个函数将相关字母转换成等价的大写或小写字母。另外，由于这两个函数的返回值类型都是int而不是char，所以我们在代码中必须要要将它们强制转换成char，否则程序返回的会是88、97这样的数字。

#include <iostream>  
#include <cctype> // For toupper and tolower  
using namespace std;  
  
int main() {  
 char lower = 't';  
 char upper = 'A';  
  
 // (char) makes sure we the character, not the int  
 cout << (char)toupper(lower) << endl; // Cast required  
 cout << (char)tolower(upper) << endl; // to see chars  
  
 return 0;  
}

**程序输出**

T  
a

## 本章小结

* 本章涉及了大量C++编程语言的细节，包括表达式、编程开发、函数调用以及编程过程中可能会发生的错误类型。这对初学者来说会有一定的压力，尤其是之前没有任何编程经验的初学者。但哪怕是实现最简单的程序，这些细节中的大部分都是必须要掌握的。
* #include <cmath>指令所引入是一个包含了许多数学函数和三角函数的库。#include <cctype>所引入的是一个是用于对个别字符进行分类和转换的函数库。
* 返回值为double类型的函数可以放在任何一个使用double对象（或浮点数表达式）的地方，cmath库中大部分函数的返回值类型都是double。
* cmath库中大部分函数都只有一个数字类型参数，pow有两个参数。
* 前置条件与后置条件是函数与调用函数的客户代码之间的一种约定，这种文档或者其他类似的文档形式的作用就是帮助人们了解函数的功能。
* 函数头信息中也包含了许多与函数用法相关的重要信息，譬如函数的返回值类型、函数的名称、函数参数的个数等，程序员们通常会根据这些信息来了解要用多少参数来调用函数。
* 函数的实参与形参是一一匹配的，这与它们各自的名称无关。第一个实参必须匹配第一个形参，第二个实参必须配对第二个形参，以此类推。\* 实参传递给形参的过程与赋值语句很类似，也就是说，实参与形参要相互兼容（譬如类型相同），将一个double值传递给int会导致部分值被丢失。

## 练习题

1. 请写出下列各函数调用的返回结果或出错原因。
   * 1. pow(3.0, 2.0)
     2. pow(-2, 5)
     3. ceil(1.001)
     4. ceil(-1.2)
     5. pow(16.0, 0.5)
     6. pow(-16.0, 2)
     7. fabs(-123.4)
     8. sqrt(-1.0)
     9. sqrt(sqrt(16.0))
     10. ceil 1.1
     11. floor()
     12. sqrt(0)
2. 请根据给出的初始化语句计算出下列表达式的值。

* double x = 5.0;  
  double y = 7.5;
  + 1. sqrt(x - 1.0)
    2. ceil(y - 0.5)
    3. sqrt(y - x + 2.0)
    4. pow(10, 2)
    5. floor(y + 0.5)
    6. pow(x, 3.0)
    7. fabs(y - x)
    8. pow(10, 3)

1. pow(4, pow(2, 3))的返回值会是什么？
2. 请编写一个计算弹道范围的算法，弹道公式为：

* 在这里，*angle*是炮弹的发射角度（用弧度来表示），*velocity*是炮弹的初始速度（以米/每秒为单位），*gravity*是重力加速度，一般为9.8米/每秒。

1. 如果调用方代码的行为不符合其调用函数的前置条件，会发生什么情况？
2. 后置条件主要提供哪些信息？
3. 在以下函数头信息中，哪几项是有效的？
   * 1. int large(int a, int b)
     2. double(double a, double b)
     3. int f(int a; int b;)
     4. int f(a, int b)
     5. double f()
     6. string c(string a)
4. 请列举出三种C++函数返回值的名称（事实上有很多）。
5. 请根据给定函数的头信息及其前置条件与后置条件，写出该函数的六个调用（要用不同的实参），这些调用要能对fmod函数进行充分的测试，并帮助之前没见过该函数的人们了解它的功能。

* double fmod(double x, double y)  
  // post: Calculates the oating-point remainder.  
  // fmod returns the oating-point remainder of x / y.  
  // If the value of y is 0.0, fmod returns Not a Number.  
  // Header required: <cmath>

1. 请写出你在上一题中回答的那六个函数的返回值。

## 编程小技巧

1. 在调用现有函数时，我们必须要提供正确数量及类型的实参。函数的头信息及其文档（如果存在的话）会为我们提供这方面的信息。请数清楚( )之间的形参个数，并确保每个相关的实参都具有与之相同的类型，或者可以转换成该类型。譬如，int的值可以赋予float，float的值可以赋予double，int的值可以赋予long。
2. 不要在调用min和max这两个函数时混用实参类型，譬如max(2, 3.0)和min(1.0, 4)都会导致编译时错误。
3. C++中有三种类型在本质上是一样的。该语言允许将整数常量视为字符常量，反之亦然。另外，在下面的代码中，你会看到false输出为0，true输出为1。这是因为存储在aChar中的不是一个可打印的字符，其值为1。

* char aChar = true; // assign 1  
  cout << ">" << aChar << "<" << endl; // Output: ><

1. 如果我们没有在代码中加入using namespace std;这条语句，就需要在每个用到的cmath函数前面都加上std::前缀。

* #include <iostream> // For cout  
  #include <cmath> // for ceil and oor  
  // using namespace std; Without this, prepend with std::  
    
  int main() {  
   std::cout << std::ceil(5.99) << std::endl; // 6  
   std::cout << std:: oor(5.99) << std::endl; // 5  
  }

## 编程项目

### 3A 使用cmath函数库

请编写一个程序，先让用户输入任意的数字，然后设置相应的标签，显示出以下这些函数的返回值（这里假设x为用户输入的数字）：

1. 计算x平方根的函数。
2. 计算x的2.5次方的函数。
3. 计算大于等于x的最小整数的函数。
4. 计算小于等于x的最大整数的函数。
5. 计算x绝对值的函数。

该程序的会话过程应该如下：

Enter a number for x: ***2.5***  
sqrt(x) : 1.5814  
pow(x, 2.5) : 6.25  
ceil(x) : 3  
floor(x) : 2  
fabs(x) : 2.5

### 3B 圆形问题

请编写一个C++程序，它会从键盘输入中读取圆的半径值，然后输出圆的直径、周长和面积。另外，我们要求你在计算圆的面积时使用pow函数。

在这里，PI应被初始化为一个值为3.14159的常量对象。整个程序的会话过程应如下（*请注意：* 浮点数的输出情况是因C++编译器而异的，所以你们得到的输出结果可能会稍有不同，尤其是Circumference和Area这两个值所显示的小数位）：

Enter Radius: ***1.0***  
Diameter: 2.0  
Circumference: 6.28318  
Area: 3.14159

请以radius = 1.0的情况运行这个程序. 验证一下你所得到的Circumference和Area是否与上述会话样例相匹配。然后，再将半径值分别改成2.0和2.5，验证一下程序的输出是否符合你的预期。

### 3C 多种形式的四舍五入

请编写一个程序，让用户输入一个数字，然后输出这个数字被四舍五入成0位、1位、2位、3位小数的的结果。该程序的会话过程应该如下：

Enter the number to round: ***3.4567***  
3.4567 rounded to 0 decimals = 3  
3.4567 rounded to 1 decimal = 3.5  
3.4567 rounded to 2 decimals = 3.46  
3.4567 rounded to 3 decimals = 3.457

### 3D RANGE

请编写一个计算弹道范围的算法，弹道公式为：

在这里，*angle*是弹道路径的发射角度（用弧度来表示），*velocity*是炮弹的初始速度（以米/每秒为单位），*gravity*是重力加速度，一般为9.8米/每秒。由于程序中的发射角度一定要以弧度为单位来输入，所以我们还需要将相关的角度值都转换成等价的弧度值。这样做是很有必要的，因为三角函数sin(x)所设定的实参x是用弧度来表示的。要想将角度值（x）转换成相应的弧度值，只需要将角度值乘以π/180即可（π ≈ 3.14159）。例如，，弧度值为0.7853975。另外，速度则被设定以以米/每秒为单位来输入。所以整个程序的会话过程应该如下：

Takeoff angle (in degrees)? \*\*\*45.0\*\*\*  
Initial velocity (meters per second)? 100.0  
  
Range = 1020.41 meters

### 3E 时间旅行问题

在以光速飞行的宇宙飞船中，宇航员会感觉到时间变慢了，而他们飞船的重量却增加了。我们可以用洛伦茨因子（Lorentz factor）来表示飞船速度v与重量和时间的关系，其公式如下：

在这里，*v*代表的是飞船的速度，*c*是光速（即299,792,458米/秒）。这个*因子*可以用来计算出当飞船速度增加时，宇航员减少的感知时间和飞船增加的重量。例如，但飞船以74948114.5米/秒（1/4光速）飞行时，该因子的值是1.038，我们可以看到时间减少了，重量增加了。

请根据以上描述编写一个程序，要求该程序需读取飞船在地球上的重量（譬如为90,000公斤）， 以光速的分数值来表示速度（譬如0.25，不能大于1.0），以及用光年为单位来描述旅行的距离（譬如地球到半人马座阿尔法星的距离是4.35光年）。

**程序会话 1:**

Weight of spaceship on earth in kilograms? ***90000***  
Velocity as a fraction of the speed of light 0.0 to 1.0? ***0.25***  
Distance to travel in light years? ***4.35***  
  
 Travel time: 4.35 light years  
 Perceived time: 4.21187 years  
 Earth weight of spaceship: 90000 kg  
 Weight of spaceship: 92951.6 kg traveling at 7.49481e+07 m/s

**程序会话 2:**

Weight of spaceship on earth in kilograms? ***90000***  
Velocity as a fraction of the speed of light 0.0 to 1.0? ***0.9***  
Distance to travel in light years? ***4.35***  
  
 Travel time: 4.35 light years  
 Perceived time: 1.89612 years  
 Earth weight of spaceship: 90000 kg  
 Weight of spaceship: 206474 kg tra veling at 2.69813e+08 m/s