计　算　机　科　学　丛　书

C++
程序设计教程

(美) H. M. Deitel
P. J. Deitel 著
薛万鹏　等译

机械工业出版社
China Machine Press
Prentice Hall

计算机科学丛书

C++程序设计教程

(美) H.M.Deitel
P.J.Deitel 著

薛万鹏 等译
马鸣远 审校

机械工业出版社
China Machine Press

本书详细介绍了C++语言和面向对象的程序设计。全书共分7章，分别介绍了C++中的类和数据抽象、运算符重载、继承、虚函数和多态性、C++输入/输出流等。语言流畅、简洁，可作为高等院校面向对象编程课程的教科书使用，同时也可作为C++爱好者的参考书。

本书版权登记号：图字：01－97－0506

图书在版编目（CIP）数据

C++程序设计教程/（美）迪特尔（Deitel, H.M.），（美）迪特尔（Deitel, P.J.）著；薛万鹏等译. －北京：机械工业出版社，2000.6
（计算机科学丛书）
书名原文：C How To Program, Second Edition.
ISBN 7－111－07951－5
Ⅰ.C… Ⅱ.①迪… ②迪… ③薛… Ⅲ.C语言－程序设计 Ⅳ.TP312
中国版本图书馆 CIP 数据核字（2000）第 16459 号

机械工业出版社（北京市西城区百万庄大街22号 邮政编码 100037）
责任编辑：赵红燕
北京市昌平环球印刷厂印刷·新华书店北京发行所发行
2000年6月第1版·2000年8月第2次印刷
787mm×1092mm 1/16·15.25印张
印数：6 001-9 000册
定价：22.00元

出 版 说 明

《C/C++程序设计大全》是国外最受欢迎的大学教科书之一，采用率很高，已由机械工业出版社于1997年8月翻译出版，读者反映良好。为满足大专院校师生学习要求和社会上广大读者的需要，现在我社将此书重新修订，分为两册出版，书名为《C程序设计教程》和《C++程序设计教程》。

译 者 序

在硬件技术飞速发展的今天，人们对软件功能的要求也越来越高。利用面向对象的程序设计技术实现软件的重用是加速软件开发进程的根本途径。C++已经成为当今主流程序设计语言。

市面上介绍C++语言的书是很多的，这些书的内容可以说是“大而全”。C++语言入门并不难，但是，正如许多人认为的那样，掌握程序设计语言最困难之处是用其灵活高效地开发实际软件系统，这需要大量的实践和学习，而C++因其丰富的功能和复杂的特点更是如此。

与其它书不同，本书从软件工程的角度介绍并讨论了C++语言，读者能在学习的同时为实际使用打下坚实的基础，初学者和有经验的程序员都会从中受到启发。原书作者有着丰富的软件开发经验。Harvey M Deitel教授是虚拟存储系统（VMS）的先驱研究者之一，如今这种系统已经广泛地应用于UNIX、OS/2和Windows NT等等的操作系统上；Paul J.Deitel在网络数据管理、数据库查询翻译器、金融财务管理系统的开发方面具有丰富的经验。本书的许多内容都体现了作者对程序设计技术的理解。

本书主要由薛万鹏、纪宁、韩磊、许文轩、梅开、谢立、薛鸾、李岩、沈长华翻译，参加翻译工作的还有濮玉民、赵晓蓉、颜先杰、任映梅、治中、瞿跃龙、苏泳民、史荣光、上官立新、单力、汪梓鸣、成文、魏莲方、马蔚、杨开开、段群慧、蒋星、文达、韩青云等，马鸣远、梁敏、沈维亮、李昕怡对本书进行了全面的审校。机械工业出版社华章公司和华译工作室也为本书的翻译给予了大力的支持和帮助，在此深表感谢。

原书内容严谨，具有较强的理论性和实用性。译者力求反映原书的特点和风貌，但由于时间关系及水平所限，不当和疏漏之处在所难免，敬请广大读者批评指正。

1997年4月

前　言

欢迎进入C++语言世界！C++是贝尔实验室的Stroustrup（Sy186）开发的。C++有许多特点是对C语言的修饰，但是更重要的是它提供了面向对象的程序设计能力。

对象实际上是模拟真实世界特点的可重用的软件组件。面向对象的程序设计是软件领域的一场革命。因为，快速、正确和经济地开发软件仍然是不断追求的目标，而当今正是对功能更强的新软件需求不断上升的时代。

软件开发人员发现：使用模块化和面向对象的设计方法比用常规的程序设计方法能够使软件开发的生产率提高10～100倍。

本书概况

本书共分7章。详细叙述了C++的特点及基本功能。

第1章“把C++看作更好的C”，介绍了C++中非面向对象的特点。这些特点改进了常规的面向过程的程序编写方法。本章讨论了单行注释、输入/输出流、声明、建立新的数据类型、函数原型和类型检查、内联函数（可取代宏）、引用参数、const限定符、动态内存分配、默认参数、单目作用域运算符以及函数模板。

第2章“类和数据抽象（I）”，把数据抽象的知识放在这一章中介绍是非常合适的。C++被认为是用来实现抽象数据类型（ADT）的。第2、3和4章涉及到了对抽象数据的处理。第2章讨论的内容包括：用结构实现抽象数据类型，用C++风格的类实现抽象数据类型，访问函数和工具函数的使用，用构造函数初始化对象，用析构函数撤销对象，用默认的逐个成员拷贝的方式给对象赋值以及软件的可重用性。

第3章“类和数据抽象（II）”，继续介绍类和数据抽象。本章讨论的内容包括：const对象、const成员函数、复合类（把其它类作为成员的类）、友元函数和友元类（它们对类的私有成员有特别的访问权）、this指针（使对象能够知道自己的地址）、动态内存分配、类的所有对象共享的静态类成员、流行数据类型的范例（数组、字符串和队列）、包容器类、递取类以及模板类。模板类是C++语言最近新增的内容，它让程序员抓住抽象数据类型（如堆栈、数组和队列）的精髓，然后用最少的代码为特定的类型建立的ADT（如整数类型的堆栈、浮点数类型的堆栈、整数类型的队列等等），模板类因此也常常称为带参数的类型。

第4章“运算符重载”，运算符重载是C++教程中最流行的话题之一。人们非常欣赏这方面的内容，他们发现这一章的内容与第2和第3章对抽象数据类型的讨论非常匹配。利用运算符重载，程序员能够告诉编译器怎样把现有的运算符用在新类型的对象上。C++已经知道怎样把运算符用在内部类型的对象上（如整数、浮点数和字符）。但是，假设我们建立了字符串类，那么加号用在它上面有什么含义呢（许多程序员用加号表示连接字符串）。这一章要学习怎样重载加号实现下述目的：把重载后的加号用于表达式中的两个字符串时，编译器要能够产生把两个字符串连接起来的运算符函数调用。本章讨论了运算符重载的基本知识，运算符重载的限制，成员函数和非成员函数的重载，单目和双目运算符的重载以及类型之间的转换。本章研究了丰富的实例，包括数组类、字符串类、日期类、巨型整数类和复数

类（最后两个实例在练习中提供源代码）。

第5章“继承”，继承是面向对象语言的基本功能之一，本章探讨了这方面的内容。继承是软件重用的一种方式，它能够在吸收现有类功能的基础上快速地开发出新的类和给新类添加新的功能。本章讨论了基类、派生类、受保护成员、公有继承、受保护继承、私有继承、直接基类、间接基类、基类和派生类中的构造函数和析构函数的使用以及有关继承的软件工程。本章比较了继承（即“是”关系）和复合（即“有”关系），还介绍了对象的“使用”关系和“知道”关系。本章有丰富的实例研究，特别是用很长的篇幅实现了点、圆和圆柱体的类层次结构。还包含有关多重继承的实例，多重继承是C++的高级特点。利用多重继承、派生类可以继承多个基类的属性和行为。

第6章“虚函数和多态性”，讨论了面向对象程序设计的另一种基本功能，即多态行为。当许多类因为继承共同基类而相关时，每一个派生类的对象都可以当作基类的对象处理，这能够使程序的编写不依赖于派生类对象的具体类型。同一个程序能够处理新类型的对象使得系统具有更好的可扩展性。多态性能够用更简单“直行”逻辑取代复杂的switch逻辑。例如，视频游戏的屏幕管理程序在绘制对象时只要简单地给链表中的每一个对象发送一条绘制消息就可以了。只要新类型的对象知道怎样绘制其自身，那么无需修改程序就可以把新的对象添加到程序中。当今流行的大量图形用户界面正是使用了这种程序设计风格。本章讨论了通过虚函数实现多态行为的机制，区分了抽象基类和具体类的差别（不用能抽象基类建立实例对象，而具体类可用来建立实例对象）。抽象基类可用于给派生提供可继承的接口。本章研究了两个重要的多态性实例，即工资单系统和在第5章中讨论过的点、圆和圆柱体这样一个类层次结构。

第7章“C++输入/输出流”，详细介绍了C++中面向对象风格的输入/输出处理。许多C教程是基于C++编译器教学的，教员通常更喜欢给学生们讲授新的C++风格的I/O（而不是C风格的printf/scanf函数）。本章讨论了C++的各种输入/输出能力，包括用流插入运算符输出数据、用流提取运算符输入数据、类型安全的I/O（是对C的改进）、格式化I/O、无格式I/O（用于提高性能）、用流操纵算子控制基数（十进制、八进制和十六进制）、浮点数、域宽控制、用户自定义算子、流格式状态、流错误状态、用户自定义类型的对象的状态以及把输出流系到输入流上（保证在要求用户通过键盘响应前先显示出提示信息）。

欢迎使用本书。本书作者期待您的意见、批评、指正和建议。

目　录

第 1 章　把C++看作更好的C

学习目标：

- 熟悉 C++ 对 C 的增强功能
- 认识到为什么 C 是进一步学习程序设计（特别是 C++）的基础

1.1　引言

欢迎进入 C++ 世界！C++ 增强了 C 的许多特点，并且提供了面向对象的程序设计能力。面向对象的程序设计有望提高软件的生产率、软件的质量和软件的可重用性。本章将讨论许多 C++ 对 C 的增强功能。

C 语言的设计者和早期的实现者从来没有想到 C 语言会这么流行（UNIX 操作系统也存在这种现象）。像 C 这样应用广泛的程序设计语言，新的需求要求人们发展该语言而不是简单地用新的语言代替它。

C++ 是贝尔实验室的 Bjarne Stroustrup 开发的，起初被称为“带类的 C”。为了表明它是 C 的增强版，所以在名字中使用了 C 语言中的自增运算符 ++，从而形成了 C++。

因为 C++ 是 C 的超集，所以程序员可以用 C++ 编译器编译已有的 C 程序，然后逐步把这些程序改进为 C++ 程序。一些大软件供应商已经提供了 C++ 编译器，并且不再提供单独的 C 编译器。

1.2　C++ 的单行注释

程序员经常要在一行代码的最后插入少量注释。C 要求一条注释用/* 和 */分界。C++ 允许用//开始一条注释，该行//之后的文本被认为是注释，注释自动在行尾结束。这种注释形式可以少键入字符，但它只用于单行注释。

例如，C 注释

```
/* This is a single-line comment. */
```

要求使用定界符/* 和 */（即使是单行注释）。C++ 的单行注释形式为：

```
// This is a single-line comment.
```

对于如下的多行注释：

```
/* This is one way to   */
/* write neat multiple- */
/* line comments.       */
```

C++ 的注释形式更为简洁：

```
// This is one way to
// write neat multiple-
// line comments.
```

但是折成多行的 C 注释实际上可以像下面那样只用一对/* 和 */标识（而不是用三对）：

```
/* This is one way to
write neat multiple-
line comments.        */
```

因为C++是C的超集，所以上述两种注释方法都可以用在C++程序中。

常见的程序设计错误是忘记了结束C注释的*/。

因此为避免因为忘记C风格注释的结束符号*/而带来的错误，最好使用C++风格的注释符号//。

这种看起来简单的错误可导致严重的后果。忘记*/会使编译器认为注释还没有结束，从而把其后的代码行都认为是注释，直到遇到一个*/或到达程序结尾时才认为注释结束。这种错误会使编译器跳过程序中的关键部分。

1.3　C++的输入/输出流

C++对处理标准类型的数据和字符串提供了取代printf和scanf函数调用的备选方法。例如，如下简单的对话：

```
printf ("Enter new tag:");
scanf ("%d", &tag);
printf ("The new tag is:%d\n", tag);
```

可用C++语句可写成：

```
cout<<"Enter new tag:";
cin>>tag;
cout<<"The new tag is:"<<tag<<'\n';
```

第一条语句用到了标准输出流cout和运算符<<（流插入运算符）。该语句的读法为：
把字符串“Enter new tag”插入输出流cout中。
注意，流插入运算符也是左移位运算符。第二条语句用到了标准输入流cin和运算符>>（流提取运算符）。该语句读法为：
从输入流cin中提取tag的值。
注意，当左边的参数是一个整数类型时，流提取运算符也是右移位运算符。与printf和scanf不同，流插入运算符和流提取运算符不需要指示输出/输入数据类型的格式控制串和转换说明符，运算符能够自动识别要用的类型，C++中有许多这样的情况。还要注意，变量tag在和流提取运算符一起使用时不需要像scanf函数那样在变量的前面加上地址运算符&。

C++程序必须包含头文件iostream.h后才能使用输入/输出流。图1-1中的程序提示用户输入变量myAge和friendsAge的值，然后比较这两个值（你与你朋友的年龄）。可以注意到，对年龄的声明放在了紧靠引用该年龄的输入语句之前。第7章“C++输入/输出流”详细地介绍了C++的输入/输出流的特点。

与使用C语言中的printf和scanf函数调用相比，用C++风格的面向流的输入/输出可使程序具有更好的可读性，并且能减少出错的可能。

```
//简单的输入/输出流
#include<iostream.h>

main ()
 {
  cout<<"Enter your age:";
  int myAge;
```

```
    cin>>myAge;

    cout<<"Enter your friend's age:";
    int friendsAge;
    cin>>friendsAge;
    if (myAge>friendsAge)
      cout<<"You are older.\n";
    else
      if (myAge<friendsAge)
        cout<<"You are younger.\n";
      else
      cout<<"You and your friend are the same age.\n"

    return 0;
  }
```

输出范例：

```
Enter your age: 23
Enter your friend's age: 20
You are older.
Enter your age: 20
Enter your friend's age: 23
You are younger.

Enter your age: 20
Enter your friend's age: 20
You and your friend are the same age.
```

图 1-1　使用流插入运算符和流提取运算符的 I/O 流

1.4　C++ 中的声明

在 C 中，程序块的所有的声明都必须出现在所有可执行语句之前。在 C++ 中，声明可放在使用所声明的内容之前的任何地方。例如：

```
cout<<"Enter two integers:";
int x, y;
cin>>x>>y;
cout<<"The sum of"<<x<<"and"<<y
    <<"is"<<x+y<<'\n';
```

声明了变量 x 和 y，声明语句出现在可执行的 cout 语句之后和使用它们的 cin 语句之前。变量也可以在 for 结构的初始化部分予以声明，其作用域仍然是在定义 for 结构的程序块内。例如：

```
for (int i=0; i<=5; i++)
  cout<<i<<'\n';
```

在 for 结构中把变量 i 声明为一个整数并把它初始化为 0。

C++ 的局部变量的作用域从其声明开始到结束程序块的右花括号终止。因此，变量声明之前的语句即使在同一个程序块内也不能引用该变量。变量声明不能放在 while、do/while、for 和 if 结构的条件中。

把变量声明放在靠近首次使用的位置，即用到时再声明或声明后写上使用，可提高程序的可读性。

常见的程序设计错误是把变量声明放在引用它的语句之后。

1.5　在C++中建立新的数据类型

C++提供了用关键字enum、struct、union和class建立用户自定义数据类型的能力。和C一样，C++中的枚举也是用关键字enum声明的。但是，与C不同的是，C++中声明的枚举是一种新的数据类型。关键字struct、union和class也是用来建立一种新的数据类型。例如，如下的声明

```
enum Boolean {FALSE, TRUE};

struct Name {
    char first [10];
    char last10;
};

union Number {
     int i;
     float f;
};
```

建立了三种用户定义的数据类型，这三种类型的标记名分别为Boolean、Name和Number。可以用这些标记名声明变量，例如：

```
Boolean done=FALSE;
Name student;
Number x;
```

这些声明建立了Boolean类型的变量done（初始化为FALSE），Name类型的变量student和Number类型的变量x。

和C中一样，枚举值在默认情况下是从0开始的，以后每个值依次加1。因此，Boolean类型的枚举把0赋给FALSE，1赋给TRUE。可以给枚举的任何一个元素赋一个整数值，其后没有明确赋值的元素被自动赋予前一个元素的整数值加1后的值。

1.6　函数原型和类型检查

函数原型能够使C编译器对函数调用进行精确的类型检查。在ANSI C中，函数原型是可有可无的。在C++中，所有的函数都要有函数原型。定义在函数调用之前的函数不要求有单独的函数原型。在这种情况下，函数的头部充当函数原型。C++还要求函数的所有的参数在函数定义和函数原型的圆括号中声明。例如，有一个整数参数并且返回值也是整数的函数square，其函数原型为：

```
int square (int);
```

对于不返回值的函数，其返回类型要声明为void。

常见的程序设计错误是试图让返回类型为void的函数返回一个值，或使用该函数调用的结果。

在C中，指定空参数列表是把void关键字放在圆括号中。如果C函数原型的圆括号中什么也没有，编译器就不检查参数并且不对参数个数和类型作任何假定。在调用该函数时，给函数传递任何参数编译器都不会报错。

在C++中，指定空参数列表的方法是在圆括号中写入void或什么也不写。声明语句

```
void print ();
```

指定函数print不接收任何参数并且没有返回值。图1-2中的程序演示了在C++中声明和使

用不带参数的函数的方法。

注意，C++ 和 C 中的空参数列表的含义是相当不同的。在 C 中，空参数列表意味着所有的参数都不予以检查。在 C++ 中，空参数列表意味着函数没有任何参数。因此，具有这种特点的 C 程序在 C++ 中编译可能会产生不同的效果。

对每一个函数必须先提供函数原型或在使用前定义，否则 C++ 程序不能通过编译。

```
//不带参数的函数
    #include<iostream.h>

    void f1 ();
    void f2 (void);

    main ()
     {
      f1 ();
      f2 ();

      return 0;
    }

    void f1 ()
     {

      cout<<"Function f1 takes no arguments\n";
    }

    void f2 (void)
     {
      cout<<"Function f2 also takes no arguments\n";
    }
输出结果:
      Function f1 takes no arguments
      Function f2 also takes no arguments
```

图 1-2　用两种方法声明和使用不带参数的函数

1.7　内联函数

从软件工程的角度来看，用一组函数编写程序是一种好方法。但是，函数调用（特别是小型函数的调用）牵涉到执行调用时的时间开销。为了避免函数调用，在定义函数时，把限定符 inline 放在函数返回类型之前可使编译器把产生函数代码的拷贝插入到合适的位置上。矛盾之处在于，这样会产生函数代码的多份拷贝，并分别插入到程序中每一个调用该函数位置上，而不是只产生一份拷贝。编译器能够忽略 inline 限定符。典型情况下，除了最小的函数外，编译器会忽略用于其它函数的 inline 限定符。

对内联函数所作的任何修改要求使用该函数的所有地方都要重新编译。在程序开发和维护的某些场合，这是有意义的。

内联函数优于预处理程序的宏。优点之一是内联函数就像其它 C++ 函数一样。因此，在调用内联函数时，编译器会进行正确的类型检查，而预处理程序的宏不支持类型检查。另一个优点是内联函数不像宏那样在使用不正确时会产生意想不到的副作用。最后，内联函数可以用调试程序调试。因为预处理程序的宏仅仅是在程序编译前让预处理程序替换文本，所

以调试程序不认为预处理程序的宏是一个特殊的部分。调试程序虽然有助于定位宏替换的逻辑错误，但是不能定位与具体的宏有关的错误。

使用内联函数可缩短执行时间，但是会增加程序的长度。因此，inline 限定符应该只用于经常使用的小函数。

图 1-3 中的程序用内联函数 cube 计算边长为 s 的立方体的体积。函数 cube 的参数列表中的关键字 const 是告诉编译器函数不修改变量 s 的值。第 1.9 节还要对 const 关键字作进一步讨论。

预处理程序的宏是用预处理指令 #define 定义的操作。宏由名字（宏标识符）和替换文本组成。可以定义带有或不带有参数列表的宏。对不带参数的宏的处理就像处理常量一样，即用替换文本替换程序中的宏标识符。带有参数的宏先替换掉替换文本中的参数，然后再扩展程序中的宏。

```
//用内联函数计算立方体的体积
#include<iostream.h>

inline float cube (const float s) {return s * s * s;}

main ()

 {
  cout<<"Enter the side length of your cube:";

  float side;

  cin>>side;
  cout<<"Volume of cube with side"
    <<side<<"is"<<cube (side) <<'\n';

  return 0;
 }
```

输出范例：

```
Enter the side length of your cube: 3.5
Volume of cube with side 3.5 is 42.875
```

图 1-3　用内联函数计算立方体的体积

考虑如下用来计算正方形面积的带一个参数的宏：

```
#define VALIDSQUARE (x)      (x) * (x)
```

预处理程序定位出现在程序中的每一个 VALIDSQUARE (x)，用一个值或一个表达式替换掉替换文本中的 x，然后扩展程序中的宏。例如：

```
cout<<VALIDSQUARE (7);
```

被扩展为：

```
cout<< (7) * (7);
```

替换文本中的圆括号强迫实现正确的计算顺序。例如：

```
cout<<VALIDSQUARE (2+3);
```

被扩展为：

```
cout<< (2+3) * (2+3);
```

它正确地输出了 5 * 5 = 25。因为预处理程序不计算作为参数提供给宏的表达式，仅仅用整个表达式的一份拷贝取代出现在替换文本中的每一个参数名，所以为了保证计算的正确性，圆括号是必需的。

考虑替换文本中不带参数的一个相应的宏：

```
#define INVALIDSQUARE (x)      x * x
```

如果提供的参数是 2 + 3，宏就会被扩展为：

```
2+3*2+3
```

因为乘法比加法的优先级高，所以该表达式就会被不正确地计算为 2 + 6 + 3 = 11。

C++ 提供的内联函数消除了与宏有关的问题，省去了与函数调用有关的开销，并且提供了函数的参数检查。内联函数的参数是在“传递”给调用函数之前计算的，所以内联函数体中出现的每一个参数都不需要圆括号。

内联函数

```
inline int square (int x) {return x * x;}
```

计算其整数参数 x 的平方。函数调用 square (2 + 3) 先计算参数 2 + 3 = 5，然后用 5 替换函数体中的参数值。图 1-4 中的程序演示了宏 VALIDSQUARE、INVALIDSQUARE 和内联函数 square。

```
//正确的宏、不正确的宏以及内联函数举例
#include<iostream.h>

#define VALIDSQUARE (x)    (x) * (x)
#define INVALIDSQUARE (x) x * x

inline int square (int x) {return x * x;}

main ()
 {
  cout <<"VALIDSQUARE (2+3) ="
       <<VALIDSQUARE (2+3)
       <<"\nINVALIDSQUARE (2+3) ="
       <<INVALIDSQUARE (2+3)
       <<"\n     square (2+3) ="
       <<square (2+3) <<'\n';
       return 0;
 }
```

输出结果：

```
VALIDSQUARE    (2+3) =25
INVALIDSQUARE (2+3) =11
square              (2+3) =25
```

图 1-4　预处理程序的宏和内联函数

表 1-1 完整地列出了 C++ 关键字。图中先列出 C 和 C++ 都有的关键字，然后列出了 C++ 特有的关键字。每一个新的 C++ 关键字在本书后面作详细解释。

注意，C++ 是一种正在发展的语言，某些特点可能是你的编译器所不具有的。使用没有实现的特点会导致语法错误。

表 1-1　C++ 关键字

C 和 C++ 公有的关键字				
auto	break	case	char	const
continue	*default*	do	double	else
enum	extern	float	for	goto

（续）

C 和 C++ 公有的关键字				
if	int	long	register	return
short	signed	sizeof	static	struct
switch	typedef	union	unsigned	void
volatile	while			

C++ 特有的关键字	
asm	编译器定义的在 C++ 中使用汇编语言的方法（参阅自己系统的手册）
catch	处理由 throw 产生的异常
class	定义新类。可以建立新类的对象
delete	从内存中撤销用 new 建立的对象
friend	把函数或类声明为是另一个类的友元。友元可访问类的所有的数据成员和成员函数
inline	建议编译器把某个特定函数的代码插入到函数调用位置
new	在自由存储区中动态分配内存对象，从而使程序在执行时获得附加的内存。对象的大小是自动确定的
operator	声明重载的运算符
private	使类的成员能够被成员函数和类的友元函数访问到
protected	私有访问的扩展。受保护的成员也能够被派生类和派生类的友元访问到
public	使类的成员能够被任何函数访问到
template	声明怎样构造使用各种数据类型的类或函数
this	在类的每一个非静态成员函数中隐式声明的一个指针。它指向激活该成员函数的对象
throw	把控制权传给异常处理程序，或者在不能定位合适的处理程序时终止程序的执行
try	建立一个包含了一组能够产生异常的语句的程序块，并且能够处理所产生的任何异常
virtual	声明虚函数

1.8　引用参数

在 C 中，所有的函数调用都是传值调用，传引用调用是通过模拟实现的，即传递指向对象的指针并在函数调用时通过复引用该指针访问这个对象。传递的数组名也是指针（常量指针），数组是通过模拟传引用调用自动传递的。其它程序设计语言直接提供了传引用调用，如 Pascal 中的 var 参数。C++ 通过提供引用参数纠正了 C 的这种不足之处。

引用参数是其相应函数变量的别名。要说明函数的某个参数是以传引用方式传递的，只要在函数原型中该参数类型之后跟上 & 就行了（与把函数的参数表示为指向变量的指针的方式完全一样）。在函数头部列出参数类型时也要使用 &。例如，函数头部的如下声明

```
int & count
```

可读成：count 是对一个整数变量的引用。在调用函数时，只要传递该变量名，变量就会自动以传引用方式调用。在函数体中通过局部变量名引用该变量实际上是引用调用函数中的原始变量，并且被调用函数可直接修改原始变量。

图 1-5 中的程序比较了传值调用、使用指针的传引用调用和使用引用参数的传引用调用。在调用函数 squareByValue 和 squareByReference 时，所用的参数是一样的。因为如果不