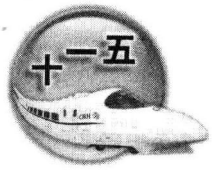
普通高等教育“十一五”规划教材

C++面向对象程序设计

张　俊　张彦铎　主编

中国铁道出版社
CHINA RAILWAY PUBLISHING HOUSE



普通高等教育“十一五”规划教材

C++面向对象程序设计

张　俊　张彦铎　主　编
吕　品　副主编

中国铁道出版社
CHINA RAILWAY PUBLISHING HOUSE

内 容 简 介

本书综合考虑了“关于进一步加强高等学校计算机基础教学意见”中C++语言程序设计基础的大纲要求，以及CC2001、中国计算机科学与技术学科教程和计算机学科专业规范中关于程序设计基础、算法和复杂性、程序设计语言、软件工程、数值科学计算等领域中的相关知识单元要求，并结合多年来在面向对象程序设计和C++语言教学实践中的经验编写而成。

本书以面向对象程序设计思想和方法为主线，以C++语言为载体，并基于标准模板库STL，详细讲述面向对象程序设计与C++语言中的主要内容：引用与函数、数组、指针与字符串、结构、类和对象、运算符重载、模板、STL、继承与派生、虚函数与多态性、I/O流、异常处理等。

全书共分为11章，体系编排完整，内容结构合理，强调重要概念，各章节所选择的例题贴合重点、丰富适度。同时本书在编排体系作了重要尝试，即：面向应用，强调实践，以C++语言标准库STL的应用为主线贯穿全教材，注重反映C++语言的新规范、新技术和新发展，这是本书的一个重要特色。

本书以培养程序设计、分析能力和计算机综合应用能力为目的，遵循计算机学科专业规范要求，适合作为计算机科学与技术及相关专业的课程教材，也可供读者自学使用。

图书在版编目（CIP）数据

C++面向对象程序设计/张俊，张彦铎主编. —北京：
中国铁道出版社，2008.8
普通高等教育“十一五”规划教材
ISBN 978-7-113-08807-1

Ⅰ.C… Ⅱ.①张…②张… Ⅲ.C语言－程序设计－高等学校－教材 Ⅳ.TP312

中国版本图书馆CIP数据核字（2008）第120144号

书　　名：C++面向对象程序设计
作　　者：张　俊　张彦铎　主编

策划编辑：严晓舟　秦绪好
责任编辑：黄园园　　编辑部电话：（010）63583215
封面设计：付　巍　　封面制作：白　雪
编辑助理：杨　勇　　责任印制：李　佳

出版发行：中国铁道出版社（北京市宣武区右安门西街8号　　邮政编码：100054）
印　　刷：三河市华丰印刷厂
版　　次：2008年8月第1版　　2008年8月第1次印刷
开　　本：787mm×1092mm　1/16　　印张：24.5　　字数：579千
印　　数：5 000册
书　　号：ISBN 978-7-113-08807-1/TP·2838
定　　价：35.00元

版权所有　侵权必究

本书封面贴有中国铁道出版社激光防伪标签，无标签者不得销售

凡购买铁道版的图书，如有缺页、倒页、脱页者，请与本社计算机图书批销部调换。

FOREWORD

前言

面向对象程序设计作为一种主流的程序设计思想和方法，因其能够更好的对现实世界中的各种数据、概念及其特征和相互联系进行真实的建模和抽象，使得程序设计与实体行为能够更加接近。此外，基于面向对象程序设计思想和方法，能够更好的组织和管理大型程序项目，并有利于继承发展程序设计领域中的各种杰出的智慧和闪亮的思想，例如各种程序库的出现和基于模式的设计。这些都极大的促进、推动了面向对象程序设计方法及其语言在各个领域的广泛应用。

C++语言是当今最流行的一种高级程序设计语言。它完全兼容 C 语言，既支持结构化的程序设计方法，也支持面向对象的程序设计方法。与其它程序设计语言相比，C++语言在运行效率、语法及语义、组件及类库、代码与资源等方面都有着显著的优越性。因此，学好 C++，很容易在一个较高平台上架设强大、易用的应用软件。

本书综合考虑了“关于进一步加强高等学校计算机基础教学意见”中 C++语言程序设计基础的大纲要求，以及 CC2001、中国计算机科学与技术学科教程和计算机学科专业规范中关于程序设计基础、算法和复杂性、程序设计语言、软件工程、数值科学计算等领域中的相关知识单元要求，并结合多年来在面向对象程序设计和 C++语言教学实践中的经验编写而成。

本套教材分为《C++面向对象程序设计》和《C++面向对象程序设计习题与实验指导》。《C++面向对象程序设计》教材以 C++语言为载体，结合 C++语言发展中的新特性、新技术，重点讲授面向对象程序设计的思想和方法。《C++面向对象程序设计习题与实验指导》是与《C++面向对象程序设计》配套的教材，包括测试习题、试验指导、相关程序库及算法参考等三部分。本套教材以面向对象和 C++程序设计的实践指导为主，重在培养学生的分析、设计、抽象和综合应用能力。

本教材的内容特点在于：体系编排完整，内容结构合理，例题适度丰富。本书对于面向对象和 C++程序设计中的重要内容，如引用与函数、数组、指针与字符串、结构、类和对象、运算符重载、模板、STL、继承与派生、虚函数与多态性、I/O 流、异常处理等都有着详细的讲解，同时强调重要概念，各章节所选择的例题贴合重点，同时具有较强的背景和完整性。

作为本教材的一个重要特色，其编排体系作了重要尝试，也是迄今为止，较鲜见于国内同类教材中。这个鲜明的特色是：面向应用，强调实践，以 C++语言标准库 STL 应用为主线贯穿全教材。在设计例题、选择习题时，以 STL 算法和容器的应用为主题，同时注重反映 C++语言的新规范、新技术和新发展。实践证明，这种安排有利于在一个较高起点和较高平台上迅速培养学生的应用能力。本教材中的示例程序全部在 Visual C++ 2005 上调试通过，程序代码符合 C++标准。

本书共分为 11 章，内容包括：

第 1 章 C++语言基础：对 C++程序设计中的基础知识、基本概念、基本运算和基本结构和数据类型进行了讨论。同时基于 STL 中的相关内容，对基本概念、基本运算和基本结构进行了新的阐述和诠释。

第 2 章面向对象概述：介绍了面向对象程序设计的基本思想和方法以及重要概念，同时，基于统一建模语言 UML，介绍了面向对象设计和分析的基础知识。

第 3 章类与对象的定义：介绍了 C++语言中类定义的语法，重点讨论了构造函数、析构函数、默认构造函数、转换构造函数、复制构造函数等重要概念。这是本书的重点之一。本章最后讨论了类的复合关系及其应用，并讨论了指向成员的指针及其在 STL 中的应用。

第 4 章类的几个主题：结合 C++语言中几个关键字 this、const、new/delete、friend 和 static 的用法，讨论了 this 指针、const 成员函数、动态内存分配及其在构造、复制、赋值、析构中的应用，讨论了友元、static 成员的用法。

第 5 章运算符重载：介绍了运算符重载常用的两种形式，重点讨论了包括算术运算符、赋值运算符、复合运算符、关系运算符、增量/减量运算符、流插入运算符/流提取运算符、下标运算符等常用运算符的重载，并结合函数调用运算符的重载讨论了函数对象的概念及应用。这是本书的重点之一。

第 6 章模板：讨论了函数模板和类模板的概念及其应用，并适当讨论了在 STL 中常用的模板新技术。

第 7 章标准模板库 STL：介绍了 STL 中关于容器、迭代器、算法等基本概念，重点讨论了函数对象、算法、容器及常见迭代器的应用。

第 8 章继承与派生：介绍了继承的基本概念，讨论了不同继承方式中的访问权限控制，以及派生类对象的构造语法及其顺序等，并讨论了赋值兼容规则、虚基类等内容。

第 9 章虚函数与多态性：介绍了面向对象程序设计中虚函数、抽象类等基本概念，讨论了纯虚函数、抽象基类等概念在多态性中的应用。

第 10 章 C++的 I/O 流：介绍了 C++中关于流的概念，重点讨论了标准 I/O 流的成员函数及运算符的用法、文件 I/O 流的操作及常见的文件操作，讨论了流格式控制的不同方式、字符串 I/O 流和流错误状态。

第 11 章异常处理：介绍了异常的概念，结合 C++标准库中的异常类及其应用，讨论了 C++中关于异常处理的方法以及常见规则。

教学安排建议：课堂教学授课 40 学时，课内实验 24 学时，课外实验 40 学时。由于学时有限，在组织教学时重点应放在重要概念、主要思想、基本算法和常用结构等方面，可根据学生的特点适当取舍，部分内容可安排自学。

本书第 1、3、4、5、7、11 章由张俊编写，第 2、6、8、9 章由吕品编写，第 10 章由张彦铎编写。在本书编写过程中，得到了江世宏、王庆春、李晓林老师的热情指导，他们根据自己丰富的教学经验提出了大量宝贵的意见，在此表示衷心的感谢！同时感谢王海晖、何成万、赵彤洲、王邯、吕涛、姬涛等老师的热情支持！

本书在编排内容上采用了一些新的尝试，贯穿于其中的教学方法和思想也还有待改进和提高。虽然经过了多年的教学实践，但是由于计算机科学与技术的迅速发展，以及编者水平有限，本教材编写中难免存在错误和不足之处，我们诚恳期待并接受读者的批评和指正，以供今后进一步完善。

编　者

2008 年 6 月

此为试读，需要完整PDF请访问：www.ertongbook.com

CONTENTS

目录

第 1 章 C++语言基础

C++语言对 C 语言作了重要的改进和扩展，具有丰富的功能和重要的特性。C++语言完全兼容结构化程序设计，并引入了主流的面向对象程序设计思想和方法，同时它还支持基于模板的泛型程序设计。标准模板库（STL）的加入和 boost 等程序库的出现，也极大丰富了 C++语言。作为一种主流程序设计语言，C++语言在越来越多的领域得到了越来越多的应用。

本章简要介绍 C++语言的基础知识，包括数据类型、运算符及基本运算、命名空间，讨论了左值/右值的概念及其应用，并用典型示例诠释了控制结构的应用。引用作为 C++新增的一种重要机制，广泛应用于函数。本章还简要讨论了数组、指针及字符串的应用。为顺利过渡到类和对象，本章最后讨论了结构及其应用。

通过本章的学习，应加深理解并熟练掌握数据类型、运算符、控制结构等基本概念，理解左值/右值、引用的概念，并结合具体问题灵活应用，掌握数组、指针和字符串的应用，能定义结构类型解决简单的问题。

1.1 程序设计基础

1.1.1 数据类型

1. 数据类型

数据泛指可以用计算机处理、计算的各类信息。为了计算，需要先把复杂数据抽象为计算机可理解的模型，并且用这个模型描述出数据的主要属性。数据类型就是这样一种模型，它是对现实世界各类复杂数据的抽象、模拟，对数据内在属性的描述。数据类型都会定义各自的类型标识符。例如，为满足计数要求，并能够对数据进行汇总、比较等运算，C++语言抽象定义了 int 数据类型来对应数学中的整数概念，并定义了 int 类型的加（+）、减（-）等运算。

一般来说，数据类型 T 具有如下三种作用，或需要定义如下三个功能：

（1）限定数据的取值集合及其表示范围。

（2）定义该数据类型可进行的运算集合或者合法操作。

（3）规定该类型数据占用内存空间的大小。

数据类型的上述作用与计算机系统及所用程序语言紧密相关，第一种功能受限于系统所用的字符集，第二种功能取决于系统所定义或支持的运算，第三种功能则密切相关于系统所用的软件、硬件平台。例如，C++中定义的浮点数类型 double 主要用于描述数学中的实数类型，其合法的二

元算术运算只包括加（+）、减（-）、乘（*）、除（/），而没有取余（%）运算。在32位平台上，double类型数据占用8个字节长度的内存空间，因而表示数据的范围为[2.225 073 858 507 201 4e-308，1.797 693 134 862 315 8e+308]。又例如，C++中定义的另外一种数据类型bool，则只能取两个值false和true，bool类型数据所占用内存空间只有1个字节，它可进行的运算有关系运算、逻辑运算，也包括因为具有整型数据的属性而可以进行的部分算术运算。

C++语言除了提供了bool、char、int、float和double等基本数据类型之外，还提供了较多自定义数据类型的机制。为定义一种数据类型，需要定义数据的组织方式，还需要定义对数据的操作，或者数据之间的运算，从而使得该自定义数据类型具有与基本数据类型相同的能力。例如，标准模板库（STL）以概念（Concept）的形式对数据类型提出了明确需求，要求某数据类型可赋值（Assignable）、可默认构造（Default Constructible）、可比较相等性（Equality Comparable）。可赋值是指该数据类型能够通过赋值运算符“=”产生值的副本；可默认构造是指该数据类型T允许以T()方式构造变量或对象，即在构造时不指定初始值；可比较相等性是指该数据类型能够用运算符“==”比较变量（或对象）的相等性。

2. 基本数据类型

基本数据类型是指已经内置于系统中的数据类型，对基本数据类型的各种操作已经得到系统“天然的”支持。C++中的基本数据类型分为三类：整型、浮点型和void类型。

整型泛指可对应于整型数据的数据类型，包括char、bool、short、int、long类型，以及可用unsigned/signed、short/long等修饰符限制而产生的数据类型。整型对应于数学中的整数概念。char型数据因为在C++中存储为对应的ASCII码，因而被认为整型类型，bool类型的两个值false和true分别对应于0和1，因而也被认为是整型。各种整型类型因其占用系统内存空间的大小不同，所表示数据的范围也不同，对整型的运算包括：算术运算、赋值运算、增量/减量运算、关系运算、逻辑运算、位运算等。

浮点型对应于数学中的实数概念，主要包括float、double类型，以及用long修饰而产生的long double数据类型。浮点数可以进行整型数据的大多数运算和操作，除了取余运算（%）、位运算外。

void类型不能直接用于定义变量，常用类型标识符void*定义指针类型，表示通用指针，即该类型指针可以容纳其他任意类型的指针类型数据。该关键字也常用于函数，用于函数参数时，表示空参数列表，即不带任何参数；用于函数的返回类型时，表示该函数不返回任何值。对void*类型可以施加的运算主要是指针的常用运算。

对于C++系统中基本数据类型的运算，例如，输入/输出、算术运算等，都已经实现于系统中。各基本类型所占系统内存空间的大小可以通过sizeof运算符求得，它们所能表示数据的范围也有多种方式获得。下列程序说明了基本数据类型数据的输入/输出，以及求取该类型表示范围的方法。

【例1.1】基本数据类型数据的输入/输出、大小及表示范围。

```
#01     #include <iostream>
#02     #include <limits>
#03     #include "XR.hpp"
#04     using namespace std;
#05     int main()
#06     {
#07         char c; cin >> c;   cout << c;
```

此为试读，需要完整PDF请访问：www.ertongbook.com

```
#08         int n;      cin >> n;   cout << n;
#09         double d;   cin >> d;   cout << d;
#10         XR(sizeof(char));
#11         XR(sizeof(int));
#12         XR(sizeof(double));
#13         XR(numeric_limits<int>::max());
#14         XR(numeric_limits<int>::min());
#15         XR(numeric_limits<double>::max());
#16         XR(numeric_limits<double>::min());
#17     }
```

上述程序以 char、int、double 三种常用的数据类型演示了基本类型数据的键盘输入、屏幕输出等基本操作，以及这三种类型所占内存字节的大小和能够表示数据的范围。

对基本类型数据的输入/输出操作，即通过流插入运算符“<<”和流提取运算符“>>”而实现的插入/提取操作，都已经定义于 C++的 I/O 流库，只要包含 C++标准头文件<iostream>，程序就可以得到输入/输出的能力。

运算符 sizeof 主要用于计算标识符所代表类型或者数据占用内存字节的大小。

C++标准模板库（STL）中的类模板 numeric_limits<T>为基本数据类型 T 定义了各种数值属性，其中函数 max 和 min 分别返回数据类型 T 的最大值和最小值。为使用该类模板，需要包含标准头文件<limits>。

上述程序中，标识符 XR 是本书编者为了观察和分析表达式及其计算结果而定义的带参数的宏，是 expression 和 result 的缩写。该宏首先输出待分析表达式所在的行号，然后输出该表达式，最后输出该表达式的值。该宏定义在头文件"XR.hpp"中，具体实现请参见与本书配套辅助教材的附录。

借助本书第一个例程，这里想强调的是：C++中 main 函数最为常用的标准形式一般如下：

```
int main()
{}
```

虽然函数体中并没有 return 语句返回某个 int 型值（这一点会被 VC++ 6.0 编译器警告），但是 C++隐式地在函数末尾定义了语句：return 0;，本书中所有程序都会采用这一种 main 函数形式。

3. 变量（对象）的定义、赋值及初始化

数据类型能够进行的运算或者所具有的功能，必须通过该类型的实例（也可称为对象）体现出来。通过定义某数据类型的对象，并对该对象施加该数据类型所允许的操作，从而体现出该数据类型的作用及其价值。以基本数据类型为例，为测试 int 类型所具有的功能，需要首先“构造”出该类型的变量（或者对象），然后对该变量施行诸如加减乘除的算术运算、比较大小的关系运算等。

为“构造”变量（或对象），以便让该变量（或对象）存活且具有明确的值，需要用到定义、赋值、初始化等概念。

（1）定义：生成数据类型的一个实例。定义某类型的变量（或对象）时，会根据该数据类型要求的内存组织方式给该变量（或对象）分配内存，一个拥有了内存的变量（或对象）即开始“存活”，可以进行各种运算。以基本数据类型为例，定义变量的语法格式如下：

```
T t;
```

其中 T 表示任意基本数据类型，t 是该类型的变量（或对象）。定义变量（或对象）涉及对其“构造”的运算。

（2）赋值：为了让已经定义好的变量（或对象）t具有某个数值，可以采用赋值运算来实现，以基本数据类型为例，给变量赋值的语法格式如下：

```
t=val;
```

其中val表示与t同类型的常量、变量或者表达式。赋值运算需要该数据类型定义赋值运算符（=）。

（3）初始化：如果在定义变量（或对象）的同时给定初始值，则称对该变量（或对象）初始化。初始化的方式可以两种形式进行，它们之间明显的区别是所用的运算符：=和()，以赋值运算符完成的初始化称为“复制初始化”，以圆括号运算符完成的初始化称为“直接初始化”。例如，下列程序中，变量a以复制初始化的方式初始化为int型常量2，变量b以直接初始化的方式初始化为int型常量3；除了初始化为同类型的某个初始常量值外，还可以用同类型的变量相互初始化，例如，c初始化为a，d初始化为b。

```
#01     int a=2;
#02     int b(3);
#03     int c=a;
#04     int d(b);
```

对于上述变量（或对象）的初始化方式，需要强调如下几点。

（1）比较复制初始化和直接初始化，以圆括号运算符实现的直接初始化方式更能说明：初始化是对变量或对象的“构造”行为；这种“构造”是通过函数调用的方式实现的，因为圆括号是函数调用运算符，是函数调用的标志。当构造变量（或对象）调用函数所需要的参数不只一个时（例如T t(2,3)），这种方式更适合对象的初始化过程。

（2）复制初始化容易给人一种错觉，认为这是赋值行为。例如，对变量a和c的初始化，虽然形式上是通过“=”运算符完成，但一定不要认为它们是“赋值”行为，这两者是差异很大的行为。

（3）赋值和复制初始化的相同点在于：两者的结果都会使得赋值运算符左边的变量（或对象）具有与右边的变量（或对象）相同的值；两者的区别在于：赋值时，变量（或对象）已经存在，而复制初始化时变量（或对象）正在生成。在本书后面讲到构造函数部分时，会更加清晰地揭示两者的差别：赋值和初始化是通过完全不同的函数实现的。但是在实现过程中，这两个函数的相同点会使得它们具有相似的功能及程序代码，两者的不同点会使得它们在处理内存资源时具有不同的操作。

关于变量（或对象）还有一个重要的概念——声明。需要注意如下几点：

（1）声明是为了使用，所有变量（或对象，甚至可以扩展至标识符）必须先声明，然后才能使用。

（2）声明只需表明变量（或对象）的类型和名字，其目的是为了声明该变量（或对象）的存在，而不是像定义一样，是为了使得该变量（或对象）开始“存活”。

（3）声明不一定是定义，两者的区别在于：声明可以多次，但定义只能一次；定义时会分配存储空间，而声明时不会。如果声明同时也是定义，才可以对变量（或对象）初始化。

（4）声明可以采用两种方式进行：定义性声明；以extern实现的声明。所谓“定义性声明”，大多数情况下，定义完一个变量（或对象），就意味着已经声明了该变量（或对象）。所谓“以extern

实现的声明”，当需用引用定义在文件作用域中的变量（或对象）时，则在引用之前，要以 extern 声明该变量（或对象）。例如，当变量 a 定义在文件 file1 中时，在另外一个文件 file2 中需要引用变量 a，则在引用点之前需要以 extern 语句声明变量 a 的类型和名字。

1.1.2　命名空间

当多个程序员写的程序合在一个项目中时，可能会发生标识符之间的命名冲突问题。例如，程序员 A 在文件作用域定义了函数 f，程序员 B 在文件作用域也定义了同样的函数 f，则这两个标识符会发生冲突，导致程序错误。

C++中的命名空间正是解决标识符命名冲突问题的有力工具。关键字 namespace 可用于定义一个具名空间（或者无名空间），在其中定义某些变量、函数等标识符。当需要访问该空间中的某个标识符时，需要用到该空间的名字和二元作用域运算符（::），而对无名空间中成员的访问则无需空间名，二元作用域运算符也是可选的。命名空间的定义体不需要以分号结束。

【例 1.2】命名空间的定义及其成员的访问。

```
#01     #include <iostream>
#02     using namespace std;
#03     //namespace {
#04         int x = 0;
#05         void f() {
#06             cout << "f at unnamed scope." << endl;
#07         }
#08     //}
#09     namespace TomSpace {
#10         int x = 2;
#11         void f() {
#12             cout << "f in Tom's space." << endl;
#13         }
#14     }
#15     int main()
#16     {
#17         f();
#18         cout << x << endl;
#19         TomSpace::f();
#20         cout << TomSpace::x << endl;
#21     }
```

上述程序中，第 3～8 行在全局空间定义了变量 x 和函数 f；第 9～14 行定义了命名空间 TomSpace，并在其中定义了成员 x 和 f。虽然在这两个空间中都出现了同名的变量 x 和函数 f，但是由于它们分处在不同的空间中，因而能够“和平共处”。

程序的输出结果：

```
f at unnamed scope.
0
f in Tom's space.
2
```

需要说明的是，在上述程序中，第 3～8 行在全局空间定义了变量 x 和函数 f，由于全局空间等价于无名的命名空间，因此去掉上述程序中第 3、8 行的注释就定义了一个等价于全局空间的无名空间。

从上述例子也可以看出，当命名空间中的成员很多时，对每个成员的访问都需要先写命名空间的名字，再写二元作用域运算符，然后写该成员的名字，这会使得对成员的访问变得比较麻烦。

using 关键字的出现会简化对命名空间中成员的访问。在访问成员之前，用 using namespace 声明需要访问的命名空间，则在程序中引入了该空间中的所有标识符，从而可以直接访问这些标识符了。一个典型的例子有如第 2 行：当程序需要用到 C++的输入/输出时，会用到定义在标准命名空间 std 中的流对象 cout、cin，因此，通常在程序的开始位置通过 using namespace std;声明来引入 std 空间。但有时候，这种直接引入整个空间的做法会得到批评，因为这种方式也向程序中引入了其他不需要访问的标识。

因此，本书中大多数程序并没有使用 using 声明所使用的命名空间，而是直接用标准空间 std 和二元作用域运算符访问需要用到的标识符。这种做法虽然麻烦一点，但是能够很清楚地表达标识符及其所属命名空间，特别是当程序中用到多个命名空间时，例如在程序中同时用到了标准模板库（STL）和 boost 库，它们分别定义在 std 空间和 boost 空间，这种做法的优点就显现出来了。

1.1.3 常用运算及其运算符

为方便后续章节对运算符重载的讨论，本节将对常用运算符（及其表达式）的计算过程，从函数的角度来讨论如何实现其运算过程。在讨论过程中，并不涉及表达式的值和类型的确定，以及运算符的优先级和结合性，若需参考，请查阅相关文献。

1. 左值与右值

在讨论基本运算之前，先介绍关于两种表达式——左值/右值表达式的概念。这里的左、右是相对赋值运算符而言。

左值（left value，Lvalue）是指能够出现在赋值运算符左边的表达式，右值（right value，rvalue）是指只能出现在赋值运算符右边的表达式。需要注意的是，这两个概念所强调的语气是不同的：左值表达式“能够”出现在赋值运算符左边，也“能够”出现在赋值运算符右边；而右值表达式“只能够”出现在赋值运算符右边，“不能”出现在赋值运算符左边。

对于常量和变量而言，它们与左值/右值的关系，总结如下：

（1）变量可以用作左值，也可以用作右值，但是常量只能够用作右值，不能用作左值。

（2）当变量用作左值时，可以容纳数据，其保存的数据可以被修改；用作右值时，则可以为其他表达式提供值。

（3）常量只能用作右值，不允许被修改，因而只能为其他表达式提供值。

【例 1.3】变量/常量与左值/右值。

```
#01     int main()
#02     {
#03         int a, b;
#04         a = 1;
#05         b = a;
#06         //? 2 = b;
#07     }
```

上述程序中，第 4 行的变量 a 用作左值，int 型常量 1 用作右值，给变量 a 提供数值。第 5 行的变量 b 用作左值，变量 a 用作右值，a 给 b 提供值。第 6 行，常量 2 不能用作左值，因而错误。

2. 算术运算

算术运算中加、减、乘、除、取余运算都是二元运算，它们分别求左右操作数的和、差、积、商、余数，对应的运算符分别为+、-、*、/、%。作为二元运算，算术运算需要两个参数，在对这两个参数分别执行加、减、乘、除、取余等运算后，运算过程会返回运算结果。当需要以函数形式实现算术运算过程时，该函数的伪代码如下：

```
#01     T operator # (T left, T right)
#02     {
#03         T result(left # right);
#04         return result;
#05     }
```

上述伪代码中，operator 是 C++的关键字，表示其后是一个运算符，并且这个函数是运算符函数，operator #表示函数的名字。T 表示操作数的数据类型，可以是基本数据类型，也可以是自定义数据类型。#表示要定义的运算符，第 3 行对左操作数 left、右操作数 right 执行#表示的运算，并用该结果直接初始化变量（或对象）result，第 4 行返回运算结果。由于返回的是临时变量，因此算术运算表达式不能作为左值表达式，只能用作右值。

为更灵活的应用算术运算，STL 对上述五种算术运算都定义了相应的函数对象（functor），其名称以及与算术运算之间的对应关系如表 1-1 所示。

表 1-1　算术运算类函数对象

函 数 对 象	应 用 示 例	结　果
plus<T>()	plus<T>()(a, b);	a + b
minus<T>()	minus<T>()(a, b);	a - b
multiplies<T>()	multiplies<T>()(a, b);	a * b
divides<T>()	divides<T>()(a, b);	a / b
modulus<T>()	modulus<T>()(a, b);	a % b
negate<T>()	negate<T>()(a)	-a

T 表示数据类型，a 和 b 是 T 的实例

以 plus 为例，plus<T>是类模板数据类型，该类型可以对 T 类型数据执行算术加运算。plus<T>()则是该类模板的一个默认对象，plus<T>()(a, b)则表示由默认对象对参数 a 和 b 执行算术加运算。关于类模板的详细讨论，请参见本书第 7 章相关内容。

为使用上述函数对象，需要包含标准头文件<functional>。

【例 1.4】算术运算类函数对象应用示例。

```
#01     #include <iostream>
#02     #include <functional>
#03     #include "XR.hpp"
#04     int main()
#05     {
#06         XR(std::plus<int>()(5, 3));
#07         XR(std::minus<int>()(5, 3));
#08         XR(std::multiplies<int>()(5, 3));
#09         XR(std::divides<int>()(5, 3));
#10         XR(std::modulus<int>()(5, 3));
#11         XR(std::negate<int>()(5));
#12     }
```

程序的输出结果（输出对齐有调整）：

```
#06: std::plus<int>()(5, 3)          ==>8
#07: std::minus<int>()(5, 3)         ==>2
#08: std::multiplies<int>()(5, 3)    ==>15
#09: std::divides<int>()(5, 3)       ==>1
#10: std::modulus<int>()(5, 3)       ==>2
#11: std::negate<int>()(5)           ==>-5
```

3. 赋值运算

赋值运算（assignment）是最重要的运算之一，其运算符是“=”，执行赋值运算的目的是用右操作数的值改写左操作数，此时要求左操作数必须是左值。赋值运算是具有副作用的运算之一，除了作为表达式提供一个值之外，它还会修改左操作数的值。赋值表达式的值是左操作数的值。

作为二元运算，赋值运算需要两个参数：左操作数和右操作数。但是与其他函数不同的是，实现赋值运算的函数通常只带一个参数，即右操作数，而左操作数作为函数的隐含参数，以一种特殊的方式传递给赋值运算符函数。赋值运算的结果是返回左操作数，作为整个表达式的值。当需要以函数形式实现赋值运算时，该函数的伪代码如下：

```
#01     T& operator = (T* this, T right)
#02     {
#03         复制 right 到 this 所指对象中;
#04         return *this;
#05     }
```

上述伪代码中，this 是 C++的关键字，它表示一个指针，指向待赋值的对象。赋值操作主要是把右操作数 right 的值复制到待赋值对象中，其中的复制操作可能会非常简单，例如，根据右操作数直接改写左操作数对应的数据，也可能非常复杂，例如在复制右操作数的值到左操作数之前，先要处理左操作数的内存资源。第 1 行中 T&表示引用类型，引用作为 C++新增的机制，具有重要的应用，本书 1.2.3 节会详细讲解。由于返回的是左操作数，因此赋值表达式能够用作左值表达式，下面的程序证明了这一点：

```
#01     int a, b, c = 1;
#02     b = a = c;
#03     c = 2;
#04     (b = a) = c;
```

4. 增量/减量运算

增量/减量运算（increment/decrement）是最重要而又容易被“误算”的运算之一。之所以被“误算”，不是因为结果常常算错，而是其计算过程常常没有被正确的理解。增量运算的运算符为“++”，其功能是把操作数加 1，减量运算的运算符为“--”，其功能是把操作数减 1。如同赋值运算，增量/减量运算也是具有副作用的运算之一，除了作为表达式提供一个值之外，它们会隐式的对操作数执行赋值运算，从而修改操作数。

增量/减量运算符可以前置于操作数，也可以后置于操作数。前置增量运算（prefix increment）的计算过程如同前置减量运算（prefix decrement）的计算过程，后置增量运算（postfix increment）的计算过程如同后置减量运算（postfix decrement）的计算过程，不同的只是增减的操作。因此，后面的讨论以增量运算为例，不再专门讨论减量运算。

作为一元运算，增量运算需要一个操作数。前置增量的运算过程分为两步：首先把操作数自增 1，然后返回增加之后的操作数作为表达式的值。当需要以函数形式实现前置增量运算时，该

此为试读，需要完整PDF请访问：www.ertongbook.com