数据结构 C++语言描述

[美] William Ford, William Topp 著

刘卫东 沈官林 译

严蔚敏 审校

清华大学出版社

前　言

本书从面向对象(object-oriented)的角度来讲述数据结构的基础知识。数据结构是计算机科学专业的一门核心课程,它的研究对象为问题求解方法、程序设计方法及一些典型数据结构的算法。

本书使用通用的C++语言作为算法描述语言。它的类和面向对象结构可以有效地实现数据结构的算法。虽然目前有多种面向对象的语言,但由于C++起源于广泛流行的C语言,因而在这些语言中占有优势。抽象数据类型(ADT)定义了数据组织和数据处理运算,本书将围绕这一概念来讨论每一种数据结构,并采用C++语言中的类来表示ADT,在对象中有效地使用这些结构。

本书结构

《数据结构C++语言描述》围绕多数据集类——表、树、集合、图和字典来组织数据结构的学习。本书包括数据结构基本内容和面向对象程序设计方法两部分,给出了许多完整程序或程序段例子,并引入了描述算法复杂度的大O方法。

第1章至第11章给出了初级数据结构课程(CS 2)的内容,第12章介绍继承和抽象类,第13和14章介绍非线性结构及其排序和查找算法。这几章覆盖了后续的数据结构与算法课程(CS 7)和高级程序设计课程的内容。另外,本书还介绍了模板和运算符扩充,以支持样板结构,并使用C++语言创建数据结构和简化数据结构的使用。

本书可作为计算机专业学生学习数据结构和面向对象程序设计方法的教材,也可供计算机专业工作者自学提高时使用。

各章简介

本书的大部分章节介绍抽象数据类型及它们作为C++的类的应用,包括类及其主要方法的声明(declaration)。除少数情况下只给出类的简单定义和方法外,大多数情况下均给出了类的完整定义,并用程序给出了类的完整实现。

第1章　概述

本章引入抽象数据类型(ADT)的概念及其基本性质,即数据封装、信息隐藏、继承性

和多态性,并简单介绍了用 C++进行面向对象程序设计的方法。第 12 章中将对继承性和多态性进行全面介绍。

第 2 章　基本数据类型

程序设计语言提供最基本的数值和字符类型——整型、浮点型、字符型和用户定义的枚举型,并用这些基本类型组合产生了数组、记录、串和文件类型。本章以 C++为例描述了这些基本数据类型的 ADT。

第 3 章　抽象数据类型和类

本书描述了抽象数据结构及其用 C++类的表示。本章定义了类的基本概念及其建立和使用的方法。

第 4 章　集合类

集合是具有增加、删除、修改数据项功能的存储类。本书重点是对集合类的研究。本章给出了不同集合类型的例子。另外,还引入了衡量算法复杂度的大 O 方法。本书使用该方法来比较不同的算法。本书最后以一个 SeqList 类的研究结束,这是一个普通表结构的典型示例。

第 5 章　栈和队列

本章讨论栈和队列,它们分别以后进先出(LIFO)和先进先出(FIFO)顺序处理数据。另外,还讨论具有优先级的队列,这是一种特殊的队列,它每次从队列中删除权限最高的元组。

第 6 章　抽象运算

抽象数据类型定义了初始化和处理数据的方法。本章扩展 C++语言定义的运算符(如+,-,*,<<等)来应用于抽象数据类型,这被称为运算符重载,它重新定义标准运算符使之适用于 ADT 的抽象运算。最后,用有理数类作为例子说明如何进行运算符重载及类型转换,并介绍了用友函数重载标准 C++的输入/输出运算符的方法。

第 7 章　形式数据类型

C++使用模板机制提供支持不同数据类型的模板函数和类。模板为数据结构提供了强大的概括能力。本章以基于模板的 Stack 类及其对中缀表达式求值的应用为例说明了这些概念。

第 8 章　类和动态存储

动态数据结构使用运行时才由系统分配的内存,这可以让用户定义没有大小限制的结构,增强了类的可用性。当然,在使用它们时需要特别小心。本章介绍复制构造函数、重载赋值运算和析构函数方法,用它们可以正确地复制和赋值动态数据,并在对象被删除时释放内存空间。我们用 Array,String 和 Set 类说明动态数据的强大功能。这些类的使用将贯穿于本书。

第 9 章　链表

由于许多应用可用表来实现，用表来存储和查找数据是贯穿本书的一个论题。本章介绍可动态处理的链表。建立链表的一种方法是，首先定义一个基本的结点类，然后创建函数来增加或删除表中的数据项；另一种方法是建立一个包含遍历机制的链表类。类 LinkedList 用来实现类 SeqList 和类 Queue。这些方法提供了开发数据结构的有力工具。

第 10 章　递归

递归在计算机科学和数学中都是一个重要的问题求解工具。本章介绍递归并给出它的多个应用，如数学公式计算、组合算法、遍历迷宫和猜谜。最后，以 Fibonacci 队列为例，比较了递归算法、交互算法和直接计算三种算法。

第 11 章　树

链表是从表头顺序存取的结点集，这种数据结构称为线性表。在许多应用中，对象中的一个成员可产生多个后代，不具备线性性质。本章介绍一种基本的非线性结构——树，它的所有结点都由称为根(root)的结点产生的，树是用来描述继承关系，如计算机文件系统和商业报表的理想结构。首先集中讨论二叉树，即每个结点最多有两个后代。我们给出类 TreeNode 来实现二叉树及其应用，包括前序、中序和后序遍历算法。最后，在应用举例中，给出了以类 BinSTree 实现的二叉查找树的结构，它可用于高效地存取大批量数据。

第 12 章　继承和抽象类

继承是面向对象程序设计的基本概念。本章讨论继承的主要性质，给出其在 C++ 中的应用，并引入虚函数作为使用继承的工具。另外，它还给出了只有虚函数的基类。虚函数是面向对象程序设计的基础，在本书的后续章节中将会用到。本章引入迭代算子的概念，定义了对本书中出现的不同的表进行遍历的一个通用的算法。最后，给出了以继承和虚函数实现的异构数组及链表的例子。

第 13 章　高级非线性结构

本章继续讨论二叉树，并介绍其它的非线性结构，描述了基于数组的树，即将数组看成完全二叉树。本章对堆进行了研究，用它来实现堆排序及有权队列。尽管在一般情况下，二叉查找树是实现表的最好结构，但也存在一些不足。数据结构提供不同的深度平衡结构来保证最快的查找时间。通过继承，派生了一个新的查找树类，称为 AVI 树。本章最后介绍图的基本性质及关于图的一些传统算法。

第 14 章　集合数据的组织

本章介绍一般的数据集合的查找和排序算法，包括传统的以树组为基础的选择排序、冒泡排序和插入排序算法，以及著名的快排(QuickSort)算法。本书主要讨论的是存放在内存中的内部数据，但大量数据可能存放在外存上，需要相应的查找和排序算法。为此，我们为文件直接查找给出类 BinFile，并用它实现了外部索引顺序查找及外部归并排序算法。

背景知识

本书假定读者已修完程序设计方面的先修课，并熟悉基本的C++语言。第2章给

出了 C++的初级数据结构,用几个完整的例子给出了它们的应用。该章可作为学习本书所需 C++知识的概述。对感兴趣的读者,作者还给出 C++语言中初级类型的定义、数组、流程控制、I/O、函数和指针等的句法。并分别给出了一些例子、程序和习题。

补充材料

本书用到的所有类和程序的完整的源代码可通过 Internet 的 ftp 从作者所在的 Pacific 大学得到。本书所用 C++代码已在最新的 Borland 编译器上测试过。除了极少数例外,这些程序也可在用 Symantec C++的 Macintosh 系统和用 GNU C++的 UNIX 系统上编译和运行。

在 Internet 上,用 ftp 联结 ftp.cs.uop.edu,联上后,用 anonymous 用户名登录,口令是用户自己的 Internet 电子邮件地址。所有的软件均在/pub/C++目录下。

需要上述 C++辅导材料的读者,可和作者直接联系,通过电子邮件和普通邮件均可。电子邮件地址:billf @ uop.edu。通信地址:Bill Topp, 456 S. Regent, Stockton, CA 95204。

教师指导书给出了每章的讲课要点、大多数习题的答案和考试的样题,还给出了多数上机题的解题思路。需要教师指导书的任课教师可联络本地 Prentice Hall 公司的代表。

(使用本书中文版作为教材授课的教师,请联络 Prentice Hall 公司北京代表处,电子邮件地址:ssbj@bupt.edu.cn,通信地址:北京 100086,知春里 28 号开源商务写字楼 102 室。目前教师指导书仅能免费提供英文版。)

致谢

作者在准备《数据结构 C++语言描述》的过程中,得到许多朋友、学生和同事的支持。Pacific 大学慷慨地提供了许多资源支持完成此项工作。Prentice Hall 出版公司派出精干队伍完成本书的设计和出版。我们要特别感谢编辑 Elizabeth Jones, Bill Zobrist 和 Alan Apt, Bayani de Leon。本书由 Prentice Hall 和 Spectrum 出版服务公司共同完成,作者也得到 Spectrum 的 Kelly Ricci 和 Kristin Miller 的大力支持。

学生们通过直接和间接的反馈给我们的初稿提出了许多有价值的意见,我们的编审对初稿从内容到组织方式都给出了许多指导和建议。在此,我们要特别列出他们的姓名,Georgia 大学的 Hamid R. Arabnia;Florida 技术学院的 Rhoda A. Baggs;Michigan - Ann Arbor 大学的 Sandra L. Bartlett;美国海岸警卫队学院的 Richard T. Close;美国空军学院的 David Cook;Cotonsville(Baltimore 县)社会学院的 Charles J. Dowling;Mankato 州立大学的 David J. Haglin;California 州立大学 Chicago 分校的 Jim Murphy 和 Herbert Schildt。作者的两位同事,Texas-El Paso 大学的 Ralph Ewton 和 Pacific 大学的 Douglas Smith 对本书的出版做出了巨大的贡献。他们敏锐的洞察力和对作者的全力支持是无法估量的,并极大地提高了本书的质量。

William Ford
William Topp

目　录

第1章　概述

本书使用面向对象的程序设计语言 C++介绍数据结构和算法。我们把每种数据结构均视为抽象类型,它不但定义了数据的组织方式,还给出了处理数据的运算。这种结构,称为**抽象数据类型**(Abstract Data Type, ADT),是一种描述用户和数据之间接口的抽象模型。本书用 C++语言来表示每一种抽象数据结构,即用**类**(class)来表示 ADT,在具体应用中用**对象**(object)来存储和处理数据。

本书介绍 ADT 的基本概念及其相关属性——数据封装和信息隐藏,并通过一系列实例来阐述 ADT 的设计方法和定义数据组织及运算的一般途径。

C++类的创建是我们学习数据结构的基础,本书第 3 章将对其进行详细讨论。在本章中,我们讲述 C++类的概况及其如何表示 ADT;在带星号(*)的选读节中,给出了一些 C++类的实例;概述了对象设计的基本内容及其继承性,这是面向对象程序设计的基石;还综述了本书所用到的程序设计方法。继承性和多态性扩充了面向对象程序设计的能力,使其可用于开发基于类库的大型软件系统。我们将在第 12 章中进一步阐述继承性和多态性,并有选择地将其用在一些高级数据结构中。

本章是对书中所用概念的预习,在正式学习关键数据结构和面向对象这些概念之前,读者就可以逐渐熟悉它们。

1.1 抽象数据类型

数据抽象是程序设计的中心内容。这种抽象被称为抽象数据类型(ADT),它定义了数据取值范围和表现结构,以及对数据的操作集。也就是说,ADT 给出了一种用户定义的数据类型,其运算符指明了用户如何操作数据。ADT 与具体应用无关,这可使程序员把注意力集中在数据及其操作的理想模型上。

例 1.1

1. 小型公司维护进货信息的程序。每项进货由一条包括货号、当前库存、单价及进货级别的数据记录描述,其操作应包括修改上述不同数据,并在库存量低于一定数量时修改进货级别。数据抽象应该描述一条包括上述信息域的记录及可供库房经理维护进货记录的一系列操作。这些操作应包括在售出货物时修改库存量、调价时修改价格及在库存小于应再进货级别时给出需进货的信息。

数据

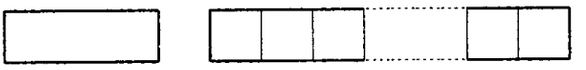
货 号	当前库存	单 价	进货级别

操作

修改库存	UpdateStockLevel
调整单价	AdjustUnitPrice
需订货信息	ReorderItem

2. 掷骰子的游戏程序。在设计中,骰子可描述成下述 ADT:其数据包括被掷骰子数目、掷出骰子的总点数和每个骰子的点数;操作包括掷骰子、返回该次投掷的骰子的总点数以及打印所掷每个骰子的点数。

数据



操作

掷骰子　　Toss
求骰子总点数　Total
打印点数　　DisplayToss

ADT 描述规范

下面给出一种描述 ADT 的规范。它包括由 ADT 名称组成的头，对数据类型的描述及操作列表。对每种操作，我们用 input(输入)来指定由用户给定的输入值，precondition(前提)表示该操作可执行前必须具有的数据，process(加工)表示由该操作完成的动作。执行操作后，用 output(输出)来表示返回给用户的值，用 postcondition(结果)来表示在数据内部所作的任何改变。大多数 ADT 都有 initialize(初始化)操作来对数据赋初值。在C++语言环境下，初始化操作称为构造函数(Constructor)。我们用它来简化从 ADT 到它在 C++ 中的表示的转变。

综上所述，ADT 的描述规范为：

```
ADT    ADT 名称        is
  Data
    描述数据的结构
  Operations
      构造函数
        Initial values:     用来初始化对象的数据
        Process:            初始化对象
      操作 1
        Input:              用户输入的数值
        Preconditions:      系统执行本操作前数据所必需的状态
        Process:            对数据进行的动作
        Output:             返回给用户的数据
        Postconditions:     系统执行操作后数据的状态
      操作 2
                ……
      操作 n
                ……
  end ADT   ADT 名称
```

例 1.2

1. 抽象数据类型 Dice 的数据包括每次所掷骰子数 N，所掷出的总点数和一个有 N 项的存放每个骰子被掷出点数的表。

```
ADT    Dice    is
Data
```

该次投掷骰子的个数。它是一大于或等于1的整数。

该次掷出的总点数。它是一个整数。如果掷N个骰子,则该值在N与6N之间。

该次投掷所掷出的每个骰子的点数表。该表的每个数值均为从1到6的整数。

Operations

Constructor

Initial values:	被掷骰子个数
Process:	初始化数据,给定每次投掷骰子的个数

Toss

Input:	无
Preconditions:	无
Process:	掷骰子并计算总点数
Output:	无
Postconditions:	所掷骰子总点数及每个骰子的点数

Total

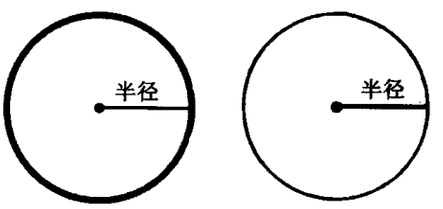
Input:	无
Preconditions:	无
Process:	检索该次投掷的总点数数据项
Output:	返回该次投掷总点数
Postconditions:	无

DiaplayToss

Input:	无
Preconditions:	无
Process:	打印该次掷出的各骰子的点数
Output:	无
Postconditions:	无

end ADT Dice

2. 圆是平面上与圆心等距的所有点的集合。从图形显示角度看,圆的抽象数据类型包括圆心和半径,而从计量的角度看,其抽象数据类型只需要半径。我们以从计量角度看的圆为例给出圆的ADT。它包括求面积和求周长的操作。本书不打算过细讨论操作的数学细节。在下节中,我们用该ADT来说明其C++类的定义和程序中对象的使用。



周长　　面积

ADT Circle is

Data

非负实数,给出了圆的半径

Operations

Constructor

Initial values: 圆的半径
Process: 给圆的半径赋初值

Area
Input: 无
Preconditions: 无
Process: 计算圆的面积
Output: 返回面积值
Postconditions: 无

Circumference
Input: 无
Preconditions: 无
Process: 计算圆的周长
Output: 返回圆的周长
Postconditions: 无

end ADT Circle

1.2 C++类和抽象数据类型

C++语言使用用户定义的**类**(Class)类型来表示抽象数据结构。类由多个存放数据值的**成员**和加工数据的运算组成。这些运算也称为"**方法**",因为它们定义了存取数据的方法。类型为类的变量称为**对象**。类可分为两个部分,其**公共**(public)部分描述用户使用类的界面,它使用户不必了解对象的内部细节而使用对象;其**私有**(private)部分由帮助实现数据抽象的数据和内部操作组成。例如,表示 ADT 圆的类中包含一个私有数据成员——radius(半径);其**公共**成员包括构造函数和计算面积和周长的方法。

类

private:
数据成员: 值 1 值 2 内部操作
public:
构造函数 操作 1 操作 2

类 Circle

private:
radius(半径)
public:
Constructor(构造函数) Area(求面积) Circumference(求周长)

数据封装和信息隐藏

类通过把数据和方法包装在一起并将它们视为整体来**封装**(encapsulates)信息。类在结构上隐藏了应用细节并严格限制对其数据和操作的外部访问。我们把类的这种特性称为**信息隐藏**(information hiding),它保护了数据的完整性。

类通过其私有和公共部分来控制外部应用对它的访问。私有部分的成员由类内部的方法在内部使用,与外界环境隔离。数据通常定义在类的私有部分以防止外界不必要的访问。它只提供其公共成员与外部环境打交道,并供用户使用。

例如,在类 Circle 中,半径是一私有成员,它仅供类 Circle 的 3 种方法使用。构造函数

可对其赋初值,其它两种方法用它来进行计算,如面积 = (π * radius^2)。这些方法是公共成员,可被外部程序调用。

消息传递

在应用中,对象的公共成员可由外部程序调用。这种调用由控制各对象相互作用的**主控模块**(主程序或子程序)来完成,控制码指挥对象用某种方法或运算访问数据。这种指挥每个对象活动的过程称为**消息传递**。消息的发送者将消息传递给接收对象请求它完成一项任务。

必要时,发送者也发送接收对象要用到的信息。这些信息作为运算的输入数据与消息一起传递。任务完成后,接收对象返回信息给发送者(输出结果)或给其它对象传递消息,要求执行其它任务。在接收对象完成运算时,它也可能修改某些内部数据值,此时,该对象发生**状态转换**,产生新的结果。

1.3 C++应用中的对象*

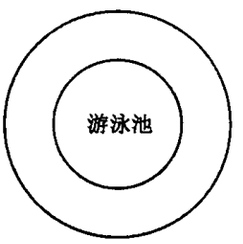
抽象数据类型给出了数据及可对数据进行的操作的综合描述。在声明C++类时一般不定义成员函数,这叫**类声明**(class declaration),是ADT的一种具体表示,方法的具体定义在独立于声明之外的**类实现**(class implementation)中给出。

我们通过一个完整的程序来说明C++类的实现和对象的应用。该程序用来计算一个圆形水池的池壁造价。程序定义了类Circle,并给出了定义和使用对象的方法、类中公共和私有部分的定义及如何使用C++函数来定义操作。主程序首先声明了对象,然后调用其操作来完成计算。另外,主程序还负责应用中的所有消息传递。

应用:类 Circle

我们用类Circle来描述一个圆形游泳池及其边上的过道。通过计算周长和面积,可以求出建造过道及围上栅栏的造价。题目如下:

一圆形游泳池如下图所示。现需在其周围建一圆形过道,并在其四周围上栅栏。栅栏价格为每英尺3.5美元,过道造价为每平方英尺0.5美元。过道宽度为3英尺,游泳池半径由键盘输入。要求编程计算并输出过道和栅栏的造价。



游泳池外环

我们声明对象Pool为游泳池本身,PoolRim为池及其周围的过道。定义好对象后,调用构造函数为其赋初值。对Pool来说,主程序将读入的半径作为参数传递给它,而PoolRim的半径为Pool的半径加3。

在对象名称后加点(.)和操作名称可调用类运算。例如,Pool.Area()计算游泳池的面.

积，PoolRim.Circumference()计算过道的周长。

栅栏安装在过道的周围，可调用计算过道周长的运算来计算栅栏的造价。

```
FenceCost = PoolRim.Circumference() * 3.50
```

过道面积为外环面积减去游泳池面积，则其造价为：

```
ConcreteCost = (PoolRim.Area()-Pool.Area()) * 0.50
```

程序 1.1　类 Circle 的创建及使用

程序 1.1 是上述问题的具体实现。程序中提供了注释以帮助读者理解程序。类 Circle 的声明给出了 ADT Circle 的表示以及可控制对成员的访问的私有及公共调用。

主程序要求用户输入游泳池的半径，并赋值给对象 Pool，对象 PoolRim 的半径为该值加 3 英尺。最后，程序输出栅栏和过道的造价。

类 Circle 在主程序外部定义。读者也许注意到限定词 const，它限定的函数成员不改变数据成员的值，而且在函数的声明和定义中都要用到。建筑材料的价格用常量给出。

```
# include <iostream.h>
const float PI = 3.14152;
const float FencePrice = 3.50;
const float ConcretePrice = 0.50;
// 声明类 Circle 及其数据和方法
class Circle
{
    private:
        // 定义数据成员 radius 为浮点数
        float radius;
    public:
        // 构造函数
        Circle(float r);
        // 计算圆的周长和面积的函数
        float Circumference(void) const;
        float Area(void) const;
};
// 类的实现
// 构造函数用类初始化数据成员 radius
Circle::Circle(float r): radius(r)
{ }
// 计算圆的周长
float Circle::Circumference(void) const
{
    return 2 * PI * radius;
}
// 计算圆的面积
float Circle:Area(void) const
{
```

```
    return PI * radius * radius;
}

void main()
{
    float radius;
    float FenceCost, ConcreteCost;
    // 设定浮点数输出时只显示小数点后两位
    cout.setf(ios::fixed);
    cout.setf(ios::showpoint);
    cout.precision(2);
    // 提示用户输入半径 radius
    cout < < "Enter the radius of the pool:";
    cin > > radius;
    // 定义 Circle 对象
    Circle Pool(radius)
    Circle PoolRim(radius + 3);
    // 计算栅栏造价并输出
    FenceCost = PoolRim.Circumference() * FencePrice;
    cout < < "Fencing Cost is $" < < FenceCost < < endl;
    // 计算过道造价并输出
    ConcreteCost = (PoolRim.Area() - Pool.Area()) * ConcretePrice;
    Cout < < "Concrete Cost is $" < < ConcreteCost < < endl;
}

/*
< 程序 1.1 的运行结果 >

Enter the radius of the pool: 40
Fencing Cost is $ 945.60
Concrete Cost is $ 391.12
*/
```

1.4 对象设计

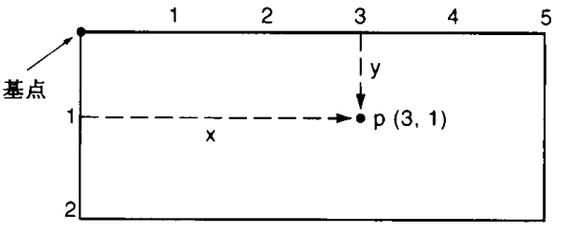
本书用类和对象讲述数据结构。我们从由简单数据成员和操作所定义的类开始。为描述更复杂的结构,类可包含本身就是对象的数据成员。这种通过**复合**方法产生的类,可以访问组成它的对象的成员函数。对象复合扩充了数据封装和信息隐藏的概念,实现了代码复用。面向对象程序设计语言也支持从其它类通过**继承**派生出新的类,这就使设计者可通过细化某个类来创建新类,并复用已开发的代码。继承是面向对象程序设计语言 C + + 的一个基本工具,我们将在第 12 章详细介绍,并用它来强化高级数据结构的设计和实现。

对象及其复合

几何图形由构成线、长方形等的点集组成。点和一系列公理构成了几何学的基础。本节中,我们把点定义成为一个原始的对象以描述线和长方形,并用它们来说明对象及其复合。

点是平面上的位置。我们用点在坐标系里的坐标(x,y)来表示点这个对象,x 和 y 分

别表示点到原点的水平和垂直距离。例如，点 P(3,1)表示从原点往右 3 个单位、往下 1 个单位。



线由点组成，两点决定一条直线。根据这一性质，可给出由起点 P1 和终点 P2 定义的线段的模型，如图 1.1(A)所示。

矩形是邻边正交的四边形。它也可用两点，即其左上点(UL)和右下点(LR)决定。如图 1.1(B)所示。

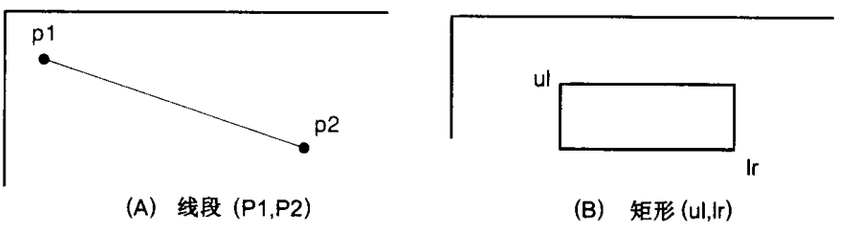
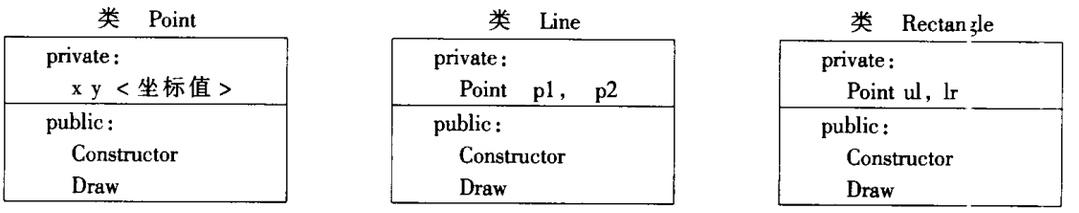


图 1.1 线段和矩形

我们用上述性质来定义类点(Point)、线段(Line)和矩形(Rectangle)。类线段和矩形的数据成员是点的对象。在使用其它类的对象创建类时，复合是一重要的工具。请注意每个类都定义了方法 Draw 来在平面上显示该图形。类 Point 还有取得该点坐标的两个函数 Getx，Gety。

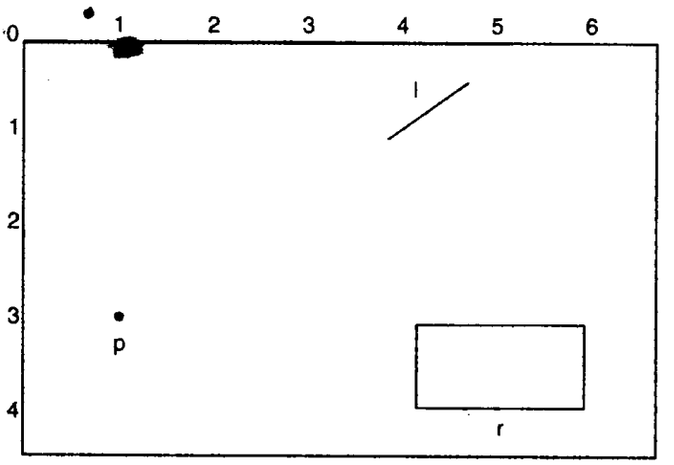


例 1.3

定义几何对象，并根据指定参数，画出几何图形。

1. `Point p(1,3);` // 定义点 p(1,3)
2. `Point p1(4,2), p2(5,1);`
   `Line l(p1,p2);` // 定义线段，起点为 p1(4,2)，终点为 p2(5,1)
3. `Point p1(4,3), p2(6,4);`
   `Rectangle r(p1,p2);` // 定义矩形，左上点为 p1(4,3)，右下点 p2(6,4)
4. 用每个类中的方法 Draw 可在平面上画出如下图形。

   `p.Draw();` `l.Draw();` `r.Draw();`



Draw 方法

C++的几何类*

下面是 C++对类 Point 和 Line 的定义。注意类 Line 的构造函数的参数是决定线段的两点的坐标值。每个类都有 Draw 函数,它可在作图区域上画出图形。

类 Point 的说明

声明

```
class Point
{
    private:
        float x, y;                    // 点的水平及垂直位置
    public:
        Point ( float h, float v);     // 将 h 赋值给 x,v 赋值给 y
        float GetX(void) const;        // 返回 x 座标(水平位置)
        float GetY(void) const;        // 返回 y 座标(垂直位置)
        void Draw(void) const;         // 在(x,y)处画一个点
};
```

类 Line 通过复合引用了两个 Point 的对象。两点均由构造函数初始化。

类 Line 的说明

声明

```
class Line
{
    private:
        Point   p1,   p2;              // 线段的两个端点
    public:
        Line (Point a, Point b);       // 将 a 赋值给 p1,b 赋值给 p2
        void Draw(void) const;         // 画出该线段
};
```