世界著名大学核心教材(计算机类)

面向对象C++数据结构

(美) Jan Harrington 著
陈博 译

科学出版社
www.sciencep.com

世界著名大学核心教材（计算机类）

面向对象 C++ 数据结构

〔美〕Jan Harrington 著

陈 博 译

科学出版社

北 京

图字：01-2003-5898 号

内 容 简 介

数据结构是一门研究非数值计算的程序设计问题中计算机操作对象以及它们之间的关系和操作等内容的学科。本书从 C++ 编程语言的角度出发，详细介绍了线性表、堆栈、队列、树和二叉树等基本的数据结构，以及在程序设计中经常遇到的两个问题——查找和排序，并且提供了具体的 C++ 实现方法。最后通过两个具体的示例，讲述了如何将数据结构应用到实际的编程工作中。

本书体系合理、结构清晰、实用性强，既可作为大专院校计算机专业的教材，也可作为从事计算机工程与应用的科技工作者的参考书。

图书在版编目(CIP)数据

面向对象 C++ 数据结构/（美）哈林顿（Harrington J.）著；陈博译. —北京：科学出版社，2005

ISBN 7-03-014572-0

Ⅰ.面… Ⅱ. ①哈…②陈… Ⅲ.①C 语言-程序设计②数据结构 Ⅳ. ①TP312②TP311.12

中国版本图书馆 CIP 数据核字（2004）第 116799 号

责任编辑：李佩乾 朱凤成/责任校对：耿 耘

责任印制：吕春珉/封面设计：东方人华平面设计部

科学出版社 出版

北京东黄城根北街16号

邮政编码:100717

http://www.sciencep.com

新蕾印刷厂 印刷

科学出版社发行 各地新华书店经销

*

2005 年 2 月第 一 版 开本：787×1092 1/16

2005 年 2 月第一次印刷 印张：23 1/4

印数：1—4 000 字数：533 000

定价：38.00 元

（如有印装质量问题，我社负责调换（环伟））

前　言

数据结构往往让许多的计算机系学生感到不寒而栗；强调数据结构的任何一门课程注定是一门艰难的课程。数据结构是各种重大计算机程序的中枢和骨架，但是您却永远无法看到它的存在。它们可能并不那么令人激动和兴奋，而且还是所有应用程序或系统程序难于理解的一部分。但与此同时，如果您不满足于仅仅编写一些微不足道的小程序，数据结构对您而言则绝对是一门需要掌握的知识。

数据结构能够为多个数据元素（无论是简单数据类型、结构或者对象）的处理提供方法，它将多个数据元素以一种可预期的方式加以组织，以便这些元素能够被检索、排序和（或）搜索。例如，多任务操作系统会使用一种被称为队列的数据结构，用它来保存等待访问 CPU 的进程。用来管理数据的应用程序——无论这些数据是消费者，还是一组游戏玩家的当前成绩和状态——必须对多个对象加以组织，以便在需要时能够迅速找到它们。

数据结构这一概念的出现已经有相当长的一段时间了。决定数据结构作用方式的基本算法早已广为人知并形成了细致、全面的文档。但是，直到最近几年，有关数据结构的内容才开始大量出现在各类结构化编程语言，例如 Pascal、C 等之中。目前，数据结构也已经应用到面向对象的编程模式之中。

许多经典的数据结构已经成为 C++ 库的一部分，特别是 C++ 标准模板库（Standard Template Library，STL），无需了解它们的具体工作方式即可使用这些数据结构。但是，如果希望为程序选择正确的数据结构，还是应该了解数据结构的工作方式以及它们能够提供哪些功能。在某些情况下，您可能会认为不能使用由库提供的数据结构，因为它们无法提供满足需要的特定功能。此时，唯一的办法便是自己编写一个数据结构。

本书站在面向对象的角度介绍了各种经典的数据结构，同时讨论了这些数据结构的 C++ 实现方式。本书的第一部分介绍了一些基本的数据结构，包括数组、向量、链表、堆栈以及队列。针对这些数据结构开发的类将在本书的第三和第四部分中重复使用，以便创建更加复杂的结构。

本书的第二部分深入考察了树结构，尤其是二叉树、AVL 树和 B 树。本部分的最后一章介绍了堆和优先级队列，它们为观察二叉树提供了一种不同的方法。树从算法上来说非常复杂，但是它们属于使用最广泛的几种数据结构之一，能够快速访问所需的数据。树的遍历使用了本书第一部分中所介绍的数据结构。

本书的第三部分讨论了用于排序、访问和查找数据的其他技术。其中一章专门讲述了各种排序例程，以及被称为二分法查找的一种速度非常快的搜索方法。此外，本部分还介绍了哈希表和字典。第三部分的所有这些排序和查找方法均使用本书先前部分所介绍的基本数据结构构建而成。

第四部分包括两个示例程序，程序综合使用了本书前面介绍的各种数据结构，利用它们来组织和访问数据。第一个程序是基于内存的；第二个程序则基于磁盘数据，因此

它们需要使用不同的数据组织方式。

如果您阅读过其他数据结构教材，一般都会找到有关图的一章内容。图实际上不是一种数据结构；它们是可以通过数据结构得以实现的一种数学概念。数学中的图论是如此复杂，以至于根本无法在一章之中对其进行完整的讨论，所以数据结构教材中对图论的所有介绍仅仅是管中窥豹，略见一斑而已。为了避免就图的问题向大家进行仅仅触及其皮毛的介绍，我做出了一个明智的决定，将其摒弃在本书的内容之外。我们将对其他主题例如树进行更加深入的讨论，它是形成许多数据结构访问方法的基础。

注意：如果希望了解有关图和图论的更多信息，请参考以下书籍：

- Giuseppe Di Battista et al. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, 1998.
- G. Chartrand. *Graphs & Digraphs*. CRC Press, 1996.
- Robin J. Wilson. *Introduction to Graph Theory*. Addison-Wesley, 1997.

据我所知，本书中的示例代码具有平台无关性。虽然这些代码是利用 CodeWarrior for Macintosh 编写的，但是我尽力避免在代码中使用特定于某个平台或编译器的语言要素。所以，这些代码应该可以正确工作在 Windows、Macintosh 和 UNIX 平台上。

阅读须知

为了最大限度理解本书的内容并加以利用，您应该能够熟练进行 C++ 编程，您应该可以轻松编写面向对象的程序，并熟悉有关面向对象的各种概念，例如类、封装和继承。此外，您还应该对指针有很深的理解。

部分数据结构代码中使用了几个比较高级的 C++ 概念（例如，指向函数和句柄的指针）。如果出现这种情况，我们将在您接触到这些概念之前，对这些语言概念做出相关解释。

获取示例代码

可以通过以下 Web 站点下载示例程序的代码以及数据结构类的模板：

http://www.blackgryphonltd.com

请单击“Books”（书籍）按钮，然后单击书名。在本书的专用页面上，您会找到一个包含了源代码的 ZIP 文档（Macintosh 和 UNIX 的用户一般可以使用 unzip，但是 Windows 的用户不能对该文档进行 unstuff 或 untar）。

致谢

每一本书的问世均凝聚着一个集体的心血和智慧。借此机会，我希望向那些为本书的写作提供帮助的人表示我衷心的谢意。

- Diane Cerra，中途接手本书的编辑工作，并引领项目顺利完成。Diane，能够和你一起工作真的感到很高兴。
- 同属于编辑小组的 Belinda Breyer 和 Mona Buehler，感谢两位女士的辛勤工作。您们对于我遇到问题的支持和快速响应令我感动不已。
- 图书制作小组：Victor Curran（出版服务经理）和 Angela Dooley（制作编辑）为

Morgan Kaufmann 制作书籍永远是一件乐事。他们是世界上最出色的书籍制作人！

- Adrienne Rebello，富于幽默感的加工编辑。她漂亮的眼睛是那么的热情。
- 此外，本书还聘请了三位技术顾问对内容进行全面的审阅，我在此对他们表示由衷的感谢：
- Karen Watterson：圣地亚哥的独立数据库顾问。
- Mark Waston：行业顾问。
- Derek Jamison：软件设计（测试）工程师。

（我从来不认为别人在我的代码里发现 bugs 是一件多么光彩的事情，但是让我很高兴的是这三个家伙居然做到了！！！）

谢谢大家！

目　录

第一部分　基本数据结构

第二部分 树

第三部分　排序、访问和查找

第四部分 应 用 示 例

第一部分　基本数据结构

面向对象的程序拥有自己独特的数据结构实现策略。在本书的这一部分中，我将首先介绍有关数组的实现策略。为什么是数组呢？因为数组是大多数程序开发人员所面对的第一种数据结构（是的，它们的确是数据结构……）。您肯定已经了解了 C++ 数组的工作方式。因此，对于演示一种新的代码组织方式来说，数组是一个非常优秀的载体。

本部分还将讨论的一些数据结构是大部分数据结构研究的基础，这些数据结构是向量、链表、堆栈和队列。正如在前面所讲到的，这些数据结构将作为其他更复杂数据结构的基础并重复使用。所以，在我们转入本书的其余部分之前，了解这些基本数据结构的功能很重要。

第1章 数 组

面向对象的编程人员处理数据结构的方式与众不同。虽然操纵数据结构的实现细节一般和其他传统数据结构没有什么不同，但是数据结构类在更高层次上的组织方式则体现了面向对象这一理念。

在本章中，我将站在面向对象的角度介绍审视数据结构的方法。为了让有经验的编程人员和刚刚接触到数据结构的新手都可以更加轻松地完成这一过渡，我们讲解的第一个例子使用了数组，大家应该对它并不感到陌生。

在以前学习数组的时候，您可能并不知道它是一种数据结构。但是，它的确满足了作为一种数据结构的所有条件：数组允许您以一种已知和可预期的方式存储、组织和操作相同数据类型的多个数据元素。数组也因此成为我们学习数据结构的一个绝佳起点。

注意：数组（以及向量，我们将在第2章讨论向量）的最妙之处在于，它提供了对数组元素的直接访问。这意味着在对数据进行排序时，可以使用一种快速的查找技术来定位特定的数值。可用来对数组及向量进行排序和查找的技术有很多，其中的部分技术将在第9章中进行讨论。

1.1 容器类和迭代程序

面向对象编程的基本原则之一便是封装，对象功能的具体实现细节对于其他对象来说是隐藏的。面向对象的数据结构将这一概念扩展到了整个数据结构的功能上。数据结构组织和操作数据的详细信息对于结构外部的所有对象来说都是不可见的。

被称为“容器类”的类负责数据结构的管理工作。一般来说，容器类能够完成以下工作：

- 包含元素或者对元素的引用，这些元素均由数据结构加以管理。
- 接受元素（或者对元素的引用），并且将元素插入到数据结构之中。
- 为数据结构内部的元素定位提供一种或多种手段。
- 返回数据结构中某个元素的值。为了找到所需的元素，可能需要进行一次“查找”操作。
- 从数据结构中删除元素（或者对元素的引用）。

容器类也支持修改数据元素的数值。修改数值需要什么？这完全依赖这个数据结构是如何组织的，以及被存储的元素的性质。你将会在整本书中看到许多这方面的示例。

容器类不执行I/O操作，但是存在一个特例。如果被容器类管理的对象是持久性存在的——在创建对象的程序停止运行后对象依然存在，那么，尽管实际执行读写操作的代码也是持久性对象自身的一部分，容器类可能也需要与文件I/O进行协调。第12章和第13章提供了一些示例程序，展示了容器类与持久性对象进行交互的具体方法。

注意： 容器类不支持按照顺序返回数据结构中的元素（即遍历整个数据结构）。数据遍历由一个被称作迭代类（iterator）的特殊类型的类来完成。迭代对象每次使用迭代程序和数据结构中的一个元素，以某种预定的顺序提供对象。

迭代对象持续跟踪它在数据结构中的位置，它独立于该数据结构本身。这意味着在任何一个给定的时间，每个数据结构可以存在多个迭代对象，每个迭代对象都以不同的顺序从数据结构的不同位置取得元素。

迭代程序极大简化了某些更复杂数据结构（例如二叉树）的遍历工作。在迭代类编写完毕之后，就永远再也不需要担心数据结构的遍历问题了。只需简单地创建一个能够以希望的顺序返回数值的迭代对象，然后要求它从数据结构一个接一个地返回元素就行了。

1.2 处理简单数据类型数组

图 1-1 展示了一个简单的程序用户界面，演示了在简单数据类型（在本例中是整数型数据）的数组上使用容器类的方法。和本书中其他的演示程序一样，本程序不会执行任何有意义的实际工作，它仅仅提供了对容器类所有成员函数的访问，以及展示了与容器相伴的迭代类的使用方法。

```
You can:
  1. Add an item
  2. Delete an item
  3. Retrieve an item by its position in the structure
  4. Find ordinal position of element
  5. List all items
  6. Get size of structure
  9. Exit

Choice:
```

图 1-1 成数组演示程序的用户界面

容器类，即数组管理器的声明可以在程序 1-1 中找到。该数组管理器（array manager）类存储了数组中的所有元素，数据能够存储的最大元素，以及指向数组自身的一个指针。它支持的成员函数包括添加元素、读取元素、删除元素以及查找元素。

【程序 1-1】 简单数据类型数组的数组管理器类的声明。

```
#ifndef MGR
#define MGR 1

class ArrayMgr
{
   private:
      int total_elements, max_elements;
      int * theArray;

   public:
      ArrayMgr (int);  // pass in value for max_elements
      ~ArrayMgr ();
```

```
        bool addElement (int); // pass in a single element
        // pass in ordinal position of element; return element value
        bool getElement (int, int &);
        // pass in ordinal position of element
        bool deleteElement (int);
        // pass in element value; return ordinal value
        bool findElement (int, int &);
        int getSize (); // return total elements in the array
};

#endif
```

该类的具体实现方式可以在程序 1-2 中找到。构造函数（constructor）以输入参数的形式获得数组的大小，并且为该整数数组分配存储空间。虽然数组的存储空间是动态分配的，但是它并不是一个动态数组。换句话说，在数组存储空间分配完毕之后，数据的大小就固定下来，并且无法改变。

【程序 1-2】 简单数据类型数组的数组管理器类。

```
#include "ArrayMgr.h"

ArrayMgr::ArrayMgr (int iMax)
{
    max_elements = iMax;
    total_elements = 0;
    theArray = new int [max_elements];
}
ArrayMgr::-ArrayMgr()
    { delete [] theArray; }

bool ArrayMgr::addElement (int newValue)
{
    bool result;
    if (total_elements < max_elements)
    {
        theArray[total_elements] = newValue;
        total_elements++;
        result = true;
    }
    else
        result = false;
    return result;
}

bool ArrayMgr::getElement (int position, int & value)
{
    int result = true;
    if (position >= total_elements)
        result = false;
    else
        value = theArray [position];
    return result;
}

bool ArrayMgr::deleteElement (int position)
{
    bool result = true;
```

```
    if (position >= total_elements || total_elements == 0)
        result = false;
    else
    {
        for (int i = position; i < total_elements - 1; i++)
            theArray[i] = theArray[i+1];
        total_elements--;
    }
    return result;
}

bool ArrayMgr::findElement (int searchValue, int & position)
{
    int result = true;
    int i = 0;
    while (i < total_elements)
    {
        if (theArray[i] == searchValue)
            position = i;
        i++;
    }
    if (i >= total_elements)
        result = false;
    return result;
}

int ArrayMgr::getSize ()
    { return total_elements; }
```

注意：动态数组一般被人们称作向量。我们将在第 2 章讨论向量问题。

1.2.1 添加元素

为了向数组中添加一个元素，数据管理器期望对象调用 addElement 函数，并提供将要被插入到数组中的值。然后，数组管理器以如下方式完成元素的插入操作：

（1）确定数组中是否存在能够插入新元素的空间。如果没有，返回 false，表明插入操作失败。

（2）将新值插入到数组中。

（3）将数组所包含元素的总数加 1。

（4）返回 true，表明插入操作成功完成。

当然，还需要让调用函数对返回的布尔值进行解释。

1.2.2 读取元素

为了读取一个元素，数据管理器需要知道所读取元素在数组中的次序位置，以便返回该值。然后，它可以以引用参数的形式返回该值。

数组管理器以如下方式执行 getElement 函数：

（1）确定所请求的位置是否在当前数组位置范围之内。如果不是，则返回 false，表明取值操作失败。

（2）将所请求数组位置的值赋给引用参数（在本例中是 value）。

（3）返回 true，表明取值操作成功完成。