高等院校计算机实验与实践系列示范教材

数据结构与算法实验实践教程

乔海燕　蒋爱军　高集荣　刘晓铭　编著

清华大学出版社

北　京

内 容 简 介

《数据结构与算法实验实践教程》是为“数据结构与算法”实验课程设计的教材。

全书共 9 章，内容包括程序测试与运行时间度量、线性表和串的实现及其应用、栈与队列的实现和应用、递归、二叉树的实现和应用、查找的实现与应用、排序的实现与应用、图算法及其应用和标准模板库 STL 简介。每章针对常用的数据结构和算法设计了例题和习题，其中大部分习题可以通过网上在线测评系统 `http://soj.me` 提交。部分习题在书后附有参考答案。

本书是独立于其他数据结构和算法教材的辅导书，可作为高等院校数据结构与算法实验课的教材和参考书，也适用于计算机编程爱好者。

图书在版编目（CIP）数据

数据结构与算法实验实践教程/乔海燕等编著. —北京: 清华大学出版社, 2012.
高等院校计算机实验与实践系列示范教材
ISBN 978-7-302-30066-3

Ⅰ. ① 数… Ⅱ. ① 乔… Ⅲ. ① 数据结构–高等学校–教材 ② 算法分析–高等学校–教材
Ⅳ. ① TP311.12

中国版本图书馆 CIP 数据核字（2012）第 212212 号

责任编辑：索　梅　赵晓宁
封面设计：
责任校对：焦丽丽
责任印制：

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　邮　　编：100084
社 总 机：010-62770175　　邮　　购：010-62786544
投稿与读者服务：010-62776969, c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015, zhiliang@tup.tsinghua.edu.cn
课件下载：http://www.tup.com.cn,010-62795954
印 刷 者：
装 订 者：
经　　销：全国新华书店
开　　本：185mm × 260mm　　**印　张：**14.25　　**字　数：**347 千字
版　　次：2012 年 10 月第 1 版　　**印　次：**2012 年 10 月第 1 次印刷
印　　数：000
定　　价：00 元

产品编号：045598-01

出 版 说 明

当前，重视实验与实践教育是各国高等教育界的发展潮流，我国与国外教学工作的差距也主要表现在实践教学环节上。面对新的形式和新的挑战，完善实验与实践教育体系成为一种必然。为了培养具有高质量、高素质、高实践能力和高创新能力的人才，全国很多高等院校在实验与实践教学方面进行了大力改革，在实验与实践教学内容、教学方法、教学体系、实验室建设等方面积累了大量的宝贵经验，起到了教学示范作用。

实验与实践性教学与理论教学是相辅相成的，具有同等重要的地位。它是在开放教育的基础上，为配合理论教学、培养学生分析问题和解决问题的能力以及加强训练学生专业实践能力而设置的教学环节；对于完成教学计划、落实教学大纲，确保教学质量，培养学生分析问题、解决问题的能力和实际操作技能更具有特别重要的意义。同时，实践教学也是培养应用型人才的重要途径，实践教学质量的好坏，实际上也决定了应用型人才培养质量的高低。因此，加强实践教学环节，提高实践教学质量，对培养高质量的应用型人才至关重要。

近年来，教育部把实验与实践教学作为对高等院校教学工作评估的关键性指标。2005 年 1 月，在教育部下发的《关于进一步加强高等学校本科教学工作的若干意见》中明确指出:“高等学校要强化实践育人的意识，区别不同学科对实践教学的要求，合理制定实践教学方案，完善实践教学体系。要切实加强实验、实习、社会实践、毕业设计 (论文) 等实践教学环节，保障各环节的时间和效果，不得降低要求。”,“要不断改革实践教学内容，改进实践教学方法，通过政策引导，吸引高水平教师从事实践环节教学工作。要加强产学研合作教育，充分利用国内外资源，不断拓展校际之间、校企之间、高校与科研院所之间的合作，加强各种形式的实践教学基地和实验室建设。”

为了配合开展实践教学及适应教学改革的需要，我们在全国各高等院校精心挖掘和遴选了一批在计算机实验与实践教学方面具有潜心研究并取得了富有特色、值得推广的教学成果的作者，把他们多年积累的教学经验编写成教材，为开展实践教学的学校起一个抛砖引玉的示范作用。

为了保证出版质量，本套教材中的每本书都经过编委会委员的精心筛选和严格评审，坚持宁缺毋滥的原则，力争把每本书都做成精品。同时，为了能够让更多、更好的实践教学成果应用于社会和各高等院校，我们热切期望在这方面有经验和成果的教师能够加入到本套丛书的编写队伍中，为实践教学的发展和取得成效做出贡献；也衷心地期望广大读者对本套教材提出宝贵意见，以便我们更好地为读者服务。

清华大学出版社

联系人：索梅 suom@tup.tsinghua.edu.cn

前　言

“数据结构和算法” 课程介绍常用的数据组织方法和常用算法，包括栈、队列、线性表和串等标准数据结构的使用和实现，二叉树和图等结构的表示和使用，常用的查找方法和排序方法。

本课程的教学目标包括两层：

(1) 熟悉常用的数据结构和算法，以便在软件开发中选择合适的现成工具；

(2) 掌握有效组织数据和处理数据的方法和技能，设计高质量的程序。

笔者认为，达到这两层目标的主要手段是动手多编程序。为此，本书提供了大量的程序设计例题和习题，既包括标准数据结构和算法的简单应用和实现，也包括典型的实用程序设计，如银行排队系统的模拟、基于 Huffman 算法的压缩、地铁换乘查询系统等。希望读者通过阅读例题和完成练习达到课程目标。

调查也发现，学生编程序时，在一个程序调试上花的时间越长，学生越容易对编写程序失去信心。为此，本书在如何激励学生多编程序，并且编高质量程序上做了一些尝试，也是本书的特色所在：

- 本书提供的程序设计练习大多提供了目前流行的在线测评, 读者在完成程序后可以提交到在线测评系统检查程序是否通过测试, 见 `http://www.teaching.sist.sysu.edu.cn/course/DataStructure/`。
- 本书设计了一些软装置，这些软装置具有简单、友好的界面，读者只需填写一个函数便可以通过运行软装置观察程序运行效果，或测试程序能否通过软装置设计的测试数据。这里的软装置可以类比于物理或化学实验室的实验装置。软装置的下载和使用见 `http://www.teaching.sist.sysu.edu.cn/course/DataStructure/`。
- 本书程序设计练习设计的一个原则是循序渐进。例如，在学习实现一个数据结构时要求学生一次只实现一个方法，其他方法由软装置或在线测评系统完成。这样使得每个程序都比较容易调试和通过，给学生编程带来一定的信心。
- 本书每个章节中都包含了相关的程序设计练习，并没有将所有练习集中编排在每章的最后面，希望这样能够更好地体现“通过动手编程序学习程序设计”的原则。

本书第 1∼ 第 5、第 7 章由乔海燕编写，第 6 和第 9 章由蒋爱军和乔海燕编写，第 8 章由高集荣和乔海燕编写。刘晓铭编写和设计了部分在线测试题目。李清洋设计了部分软装置。

王若梅教授和郭嵩山教授对本书的编写给予大力支持和鼓励，在此表示衷心感谢!

本书在编写过程中参考了诸多同行的文章和著作 (见参考文献)，在此一并致谢。

由于编者的知识水平有限，书中难免有不足和错误之处，恳请专家和读者批评指正。

编　者

2012 年 6 月

目　录

第 1 章　程序测试与运行时间度量

一个算法或程序首先应该是正确的，即对于算法的任何合法输入，程序应给出符合要求的输出。算法的输入和输出应该满足的这种关系称为算法的规格说明。一个算法或程序是否满足其规格说明往往是通过测试来确定的。本章通过排序算法的例子来学习如何设计算法的规格说明，以及如何测试算法是否满足其规格说明。

评价算法的一个重要指标是其运行时间，即算法从开始运行到结束所需要的时间。本章通过排序的练习学习如何统计算法的运行时间。另外，通过运行解决最大子序列和问题的 3 个不同算法，进一步理解算法复杂度大 O 表示的含义。

1.1　程序的规格说明与测试

一个 **算法 (algorithm)** 是描述解决一个问题的方法的指令序列。这个指令序列说明对于给定的输入，如何一步步地运行, 最后得到期望的输出。这种描述语言可以是自然语言, 程序设计语言, 或介于程序设计语言和自然语言之间的 **伪代码**。一个 **程序 (program)** 通常指用程序设计语言描述的算法。所以, 当强调算法是用程序设计语言描述时, 本书将使用 “程序” 这个术语；否则将混用算法和程序这两个术语。

1.1.1　程序的规格说明

一个 **算法的正确性** 是指，对于算法的任何合法输入，算法总是给出正确的输出。算法应该满足的这种输入和输出之间的关系，称为算法的**规格说明 (specification)**。

例如，排序算法的正确性容易描述为如下的规格说明：

- 输出是按照关键字有序的。
- 输出是输入的一个重新排列。例如, 序列 $1, 2, 2, 4$ 是 $2, 1, 4, 2$ 的一个重新排列, 而序列 $1, 2, 4$ 不是 $2, 1, 4, 2$ 的重新排列。

如果这种规格说明能够用命题表达出来，那么原则上可以用数学的方法来证明算法的正确性。不过，算法的正确性证明往往是很困难的问题，不妨试着证明插入排序是正确的。

因为编写程序是一个非常容易出错的过程, 也就是说, 把一个正确的算法编写成程序时, 程序往往没有正确地表达原算法。所以, 程序的正确性包含两层意思：

(1) 程序所表达的算法的正确性。

(2) 程序是否正确地表达了算法。

算法的正确性是其实现 (程序) 正确性的基础。程序的正确性验证仍然是一个热门的研究课题。

1.1.2　编程练习: 排序函数的规格说明

定义排序函数的规格说明所需的两个函数 (假定待排序元素是整数)：

(1) 首先，排序的输出应该是有序的，如按照关键字递增顺序排列。

(2) 其次，排序的输出应该是输入的重新排列。例如，$(3,2,2,4,3)$ 的排序结果是 $(2,2,3,3,4)$, 而不是 $(2,3,3,4)$。或者说，排序使输入和输出互为置换。

习题 1.1 定义下列函数检查一个序列是否有序:

```
bool sorted(int a[], int n);
/* 如果数组 a 的 n 个元素是按照非递减序排列的，则返回 true，否则返回
false*/
```

习题 1.2 定义下列函数检查两个序列是否互为置换:

```
bool permutation(int a[], int b[], int n);
/*长度为 n 的数组 a 和 b 的元素互为置换(重新排列)。例如，a 的元素为 (3,2,2,
4,3)，b 的元素为 (2,2,3,3,4)，则 permutation(a,b) 为 true。如果 a 的元素为
(2,1,3,2)， b 的元素为 (1,2,3,3)，则 permutation(a,b) 为 false*/
```

现在可以定义如下函数说明一个排序算法输入 a 和输出 b 之间的关系:

```
bool sortSpec(int a[], int b[], int n)
{
  return sorted(b,n) && permutation(a,b,n);
}
```

一个排序函数是正确的, 如果对于排序函数的任意输入 a 和相应的输出 b 满足 sortSpec (a, b) == true。

1.1.3 程序测试

测试 是设法寻找程序错误的过程。一般而言，一个程序的输入集合是无穷的，所以，程序员只能选择尽可能多的输入，包括特殊情况、边界情况和一般情况。检查对于这些输入，程序产生的输出是否正确, 或是否满足程序的规格说明。如果被测试的程序对于一组输入，其输出不正确，则说程序没有通过测试，说明程序有错误。

注 1.1 如果被测试程序对于一组输入，其输出不正确，说明被测试程序有错误，这时也称 **测试成功**，否则，称 **测试失败**。注意，这里的成功和失败或许有悖人们的思维习惯。这是因为程序测试的目的是“设法寻找错误”。

如果对程序进行了“大量”、“充分”的测试仍然没有发现错误，那么并不能说程序是正确的，因为没有测试所有的输入。这就是著名计算机科学家 Dijkstra 所说的:“程序测试可用来证明错误的存在，但是永远不能证明错误不存在。”所以，程序测试是控制程序质量的一个过程，对程序进行充分测试只能增强我们对程序正确性的信心。

1.1.4 编程练习：排序的测试

习题 1.3 编写一个排序程序，如插入排序、冒泡排序或选择排序。排序程序应该是一个独立的、可以被调用的函数。

例如:

```
void sort(int a[], int n)
{
//a是一个长度为n的整数数组，函数sort将数组a的元素按照非递减序排列
//以下填写函数体
}
```

习题 1.4 如果排序输入的数列很长，观察其输出是否正确不现实。可以利用 1.1.2 节练习的函数 sortSpec 测试排序算法是否存在错误。

例如:

```
int a[1000],a0[1000];              //准备输入a;
a0=a;                              //a0表示排序前的序列
sort(a,1000);                      //a为排序输出
bool b = sortSpec(a0, a, 1000);
if (b)
   cout << "排序结果正确."<<endl;
else
   cout << "排序结果不正确."<<endl;
```

如果发现错误，试改正，然后继续进行测试。设法说明进行了充分的测试。

1.1.5 随机数的生成

对程序测试时，经常需要生成一些随机数。随机数的生成可使用函数 rand。

例如：

```
#include <ctime>                //包含 time() 的头文件
#include <cstdlib>              //包含 srand() 和 rand() 的头文件

   srand(time(0));              //初始化随机数生成器

   r = (rand() % 10) + 1;       //生成1至10间的随机数
```

注意：如果没有使用函数srand进行初始化，那么每次运行程序时将得到同样的随机数序列。详情可参考 http://www.cplusplus.com/reference/clibrary/cstdlib/rand.html。

1.1.6 自动化测试

在对排序算法进行大量测试时，可以观察输入和输出是否满足规格说明。但是，这样的人为观测是不现实的，应该设法使测试自动化。例如，可以随机生成大量的整数序列作为程序的输入，并使用规格说明函数判断程序产生的每个输出是否正确。这样便可以实现大量输入的自动测试。如何实现测试的自动化是一个研究课题。

1.1.7 编程练习：排序的自动测试

习题 1.5 请用随机生成整数序列的方法，对排序函数进行大规模测试。例如，设定一个测

试次数 count，每次生成一定数目的随机数，进行排序，并检查排序结果是否正确。

```
for (int n = 0; n< count; n++)
{
   for (int j=0; j<n; j++)
      a[i] = rand();
   a0 = a;
   sort(a, n);                    //对含有n个数的数组排序
   b = sort(a0, a, n);            //检查结果是否正确
   if (!b)                        //如果不正确，输出反例
   {
      cout << "排序错误。以下是一个反例。"<<endl;
      cout << "输入: ";
      for (int j=0; j<n; j++)   //打印输入a0
         cout << a0[j] << " ";
      cout << endl <<"输出: ";
      for (int j=0; j<n; j++)   //打印输出a
         cout << a[j] << " ";
      break;
   }
}
```

1.2 程序的运行时间度量

算法的时间复杂度可以进行理论估算, 也可以通过在计算机上运行程序度量其花费的时间。不过，即使是同一个算法, 它的不同实现的运行时间 (在同一个环境下) 可能有一定的差异。

1.2.1 取得 CPU 时间

要取得进程的 **CPU 时间**，使用 clock()。在一个进程的开始和结束时分别调用函数 **clock**, 其差是两次调用 clock() 之间的时间，单位是时钟数，将时钟数除以 CLOCKS_PER_SEC(每秒时钟数) 便是进程的 **运行时间**(秒)。

例如：

```
#include <time.h>
clock_t start, end;
double cpu_time_used;
start = clock();        //计时开始
  /* 插入待度量的程序 */
end = clock();          //计时结束
cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
```

```
cout << "Running time: " << cpu_time_used<<endl;
```

有关函数 clock 的详情，可以参考 http://www.cplusplus.com/reference/clibrary/ctime/clock.html。

1.2.2 统计排序函数的运行时间

下面程序测试习题 1.3 排序函数sort 对某一组随机数序列输入的运行时间。

```
#include <iostream>
#include <ctime>
using namespace std;
const int N = 10000;
int main()
{
    clock_t start, end;
    double cpu_time_used;
    srand(time(NULL));
    int a[N];
    for (int i=0; i<N; i++)
       a[i] = rand();
    start = clock();      //计时开始
    sort(a, N);
    end = clock();        //计时结束
    cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
    cout << "Running time: " << cpu_time_used<<endl;
    return 0;
}
```

可以对不同的输入检查排序函数的运行时间。例如，输入是有序的数列，输入是逆序的数列等。注意，输入需要有适当大的规模；否则，统计时间可能是 0。

1.2.3 编程练习：排序的运行时间度量

习题 1.6 试对 1.1.4 节编写的排序函数运行时间进行事后统计。

表 1-1

输入长度 n	CUP 时间 (ms)
100 000	···
⋮	⋮

(1) 随机生成一些整数序列，统计程序对不同规模输入所花费的 CPU 时间，并将统计数据制成如表 1-1 所示的表格。

(2) 画出上述表格数据的图像，并与理论时间复杂度进行比较，看看有什么结论。

1.2.4 理解算法的时间复杂度

算法的时间复杂度通常用 O 表示法衡量，这种表示说明了算法的运行时间随着输入规

模增长的模式。例如，如果一个排序算法的时间复杂度是 $O(n^2)$，将 $n = 10\,000$ 个数进行排序需要 1 秒，那么当输入规模增长 10 倍时，运行时间将增长 100 倍，即将 $n = 100\,000$ 数进行排序需要 100 秒。同理，一个复杂度为 $O(n)$ 的算法，当输入规模增加 10 倍时，运行时间大致增加到 10 倍。再如，如果一个查找算法的时间复杂度是 $O(\mathrm{lb}\,n)$，假设在规模是 n 的表中查找一次需要 1 秒，那么当输入规模增长 10 倍时，即输入规模为 10n 时，一次查找的运行时间大致是 $\mathrm{lb}10n = \mathrm{lb}\,n + \mathrm{lb}10$, 即 1 秒加上 lb10 个时间单位, 其增长速度比 $O(n)$ 算法慢得多。换言之，时间复杂度为 $O(\mathrm{lb}\,n)$ 的算法明显优于时间复杂度为 $O(n)$ 的算法，而时间复杂度为 $O(n)$ 的算法优于时间复杂度为 $O(n^2)$ 的算法。

下面的例子说明，解决同一个问题的不同算法的复杂度可能相差很大。

例 1.1 给定一个整数序列 $a_1, a_2, \cdots, a_n$, 求其中最大子序列的和 $\sum_{k=i}^{j} a_k$, 这就是 **最大连续子序列和问题 (Maximum Subsequence Sum Problem)**。例如, 对于序列 $-1, 10, -6, 7, -3, -2$, 正确的答案是 11，即 $a_2 + a_3 + a_4 = 11$。另外，如果整数序列均为负数，则规定最大和为 0。

假定整数序列存储在数组 X[1..N] 中。这个问题的最简单解法是计算所有子序列的和，见算法 1.2.4。不难得出此算法的时间复杂度为 $O(n^3)$。

算法 1.1 时间复杂度为 $O(n^3)$ 的最大子序列算法。

```
MaxSoFar = 0;
for i = 1 to N do
   for  j = i to N do
      Sum = 0
      for  k = i to j do
         Sum = Sum + X[K]
         /* Sum 是数组X[i..j]元素的和 */
      MaxSoFar = max(MaxSoFar, Sum)
```

算法 1.2.4 中计算每一个子序列的和时都使用了一个循环。事实上，算法 1.2.4 中最内层的循环可以用一个加法代替，这样可以得到改进的算法，见算法 1.2，其时间复杂度为 $O(n^2)$。更进一步，最大子序列和问题存在线性时间复杂度解法 1.3。

算法 1.2 时间复杂度为 $O(n^2)$ 的最大子序列算法。

```
MaxSoFar = 0
for i = 1 to N do
   Sum = 0
   for j = i to N do
      Sum = Sum + X[j]
      /* Sum 是数组 X[i..j]的和 */
   MaxSoFar = max(MaxSoFar, Sum)
```

算法 1.3　时间复杂度为 O(n) 的最大子序列算法。

```
MaxSoFar = 0
MaxEndingHere = 0
for i = 1 to N do
   MaxEndingHere  = max(0, MaxEndingHere + X[i])
   MaxSoFar = max(MaxSoFar, MaxEndingHere)
```

1.2.5　编程练习：最大连续子序列和算法运算时间的比较

习题 1.7　实现以上三个算法，并对不同长度的整数序列比较三个算法的运行时间。例如，可以随机生成不同规模的整数序列，然后分别调用三个算法求解。通过本例可以进一步体验不同复杂度算法的效率差异。

习题 1.8　常见的排序算法也有很多，它们的时间性能差别也很大。例如，时间复杂度为 O(nlbn) 的排序方法有快速排序、堆排序和归并排序等，时间复杂度为 $O(n^2)$ 的排序算法有插入排序、冒泡排序和选择排序等。试着在互联网上搜索 “sorting demo”，查看排序算法演示，如 http://www.cs.bu.edu/teaching/alg/sort/demo/。

小　结

在一个大型软件项目中，程序测试人员与开发人员的比例应该是 2:1 或更高，一方面说明测试在开发费用中占用了相当大的比例；另一方面也说明测试的重要性。

程序员将算法用代码实现后，往往自己测试没有问题，但是将程序交给别人测试或提交给在线评判系统时发现了问题。主要原因可能是，程序员做的测试不够充分，或考虑的情况不够全面，或数据量太小。例如，对于排序的测试是否考虑了各种大小的输入，包括输入为空的情况；是否考虑了不同情况的输入，包括输入已经有序，输入是逆序，或输入是随机的。

程序出错更大的可能性是对问题理解不准确或不全面，因此解决问题的算法存在问题或者是错误的。所以，编写正确程序的基础是对问题理解准确，然后设计正确的算法，最后对算法的实现进行充分的测试。对于某些问题，算法的规格说明可以用函数表示，因此算法是否正确可以通过规格说明函数判断，这是实现测试自动化的一种途径。尝试写出规格说明对于理解问题和进行详细测试都是有帮助的。

程序的运行效率往往是评价一个算法的最重要指标。解决一个问题往往有多种方法，对这些方法进行时间性能和空间性能的评价可以帮助我们做出选择。对算法的时间性能一方面可以从理论上估算，另一方面也可以通过实际运行度量其运行效率，以确定算法是否满足特定的应用场合。

通过本章的练习，希望能够达到下列目的：

- 设计一个算法的规格说明；
- 设计测试程序，测试一个算法是否满足其规格说明，特别是设计自动测试程序；
- 统计一个程序的运行时间，理解算法的时间复杂度。

第 2 章　线性表和串的实现及其应用

线性表是同类型元素的线性序列，并且提供任意位置元素的存取、任意位置的插入和删除运算的抽象数据类型。线性表在标准库中的实现包括向量 vector 和双向链表 list 等。string 是表示字符序列及其操作的数据结构。三个数据结构 vector、list 和 string 是最常用的数据结构。

本章目的是熟悉 vector、list 和 string 的应用和实现。

2.1　标准库数据结构 vector 和 list 的使用

如果处理的数据具有线性的逻辑结构，需要在任意位置存取元素、插入元素或删除元素，那么考虑使用模板类 vector 或 list, 又称为 **容器 (container)**。

2.1.1　标准库数据结构 vector

模板类 vector 又称 **向量**，可以理解成将数组打包成类的实现, 其主要特点是通过运算[]提供随机存取任意位置元素的操作。

向量 vector 不同于数组的是可以使用方法 capacity 检查一个向量的当前容量 (其中打包的数组的长度)，而且向量的容量是可变的, 即当一个 vector 中元素个数超出其容量 (capacity) 时，vector 的容量会自动增加。

向量中随机存取任意元素的运行时间是常量，在表的末尾插入或删除元素的运行时间是常量，在表头或表中插入或删除元素需要线性阶时间复杂度，参见 `http://www.sgi.com/tech/stl/Vector.html`。

表 2-1 列出 vector 提供的部分方法。其中，T 表示元素类型；size_type 表示 unsigned int；reference 表示 T&；const_reference 表示 const T &。

表 2-1　vector 的部分方法

方法	说明
vector()	构造一个空向量
vector(size_type n)	构造一个有 n 个元素的向量
vector(size_type n, const T&t)	构造 n 个元素的向量，每个元素初始化为 t
iterator begin()	返回指向第一个元素的迭代器
const_iterator begin() const	返回指向第一个元素的常量迭代器
iterator end()	返回指向向量中最后一个元素下一位置的迭代器
const_iterator end() const	返回指向向量中最后一个元素下一位置的常量迭代器
size_type size() const	返回向量的元素个数
reference operator[](size_type n)	返回第 n 个元素的引用
const_reference operator[](size_type n) const	返回第 n 个元素的常量引用
reference front()	返回第一个元素的引用
const_reference front() const	返回第一个元素的常量引用
reference back()	返回最后一个元素的引用

续表

方法	说明
const_reference back() const	返回最后一个元素的常量引用
void push_back(const T&x)	在向量尾部插入一个元素 x
void pop_back()	删除最后一个元素
iterator insert(iterator pos, const T&x)	在 pos 所指位置前插入元素 x, 返回指向插入元素的迭代器。插入完成后，指向插入位置之后元素的迭代器不再指向插入前的元素
iterator erase(iterator pos)	删除 pos 处的元素, 返回指向被删除元素的下一个位置的迭代器。删除位置之后的迭代器可能不合法，或者不指向删除前的元素
void clear()	删除所有元素

模板类 vector 的第一个参数表示元素类型，第二个参数表示内存分配方式，并有一个默认值。使用 vector 时必须包含 vector 头文件，通常提供第一个类型参数即可。

例 2.1 下列代码显示 vector 部分方法的使用。

```
#include <iostream>
#include <vector>
using namespace std;

 int main()
 {
     vector<int> V0;          //V0 是空的
     cout<<"Vector V0 can hold " << V0.capacity()<<" elements."<<endl;

     V0.push_back(10);        //在尾部添加一个元素，容量自动增加
     cout<<"Vector V0 has  " << V0.size()<<" elements."<<endl;
     cout<<"Vector V0 can hold " << V0.capacity()<<" elements."<<endl;

     vector<int> V(10);       //构造包含10个元素的向量
     vector<int>::iterator itr;
                              //iterator是类vector中定义的一个类型，
                                所以使用限定符::

     cout<<"Vector V can hold " << V.capacity()<<" elements:"<<endl;
     for (vector<int>::size_type i=0; i<V.size(); i++)
       V[i]=i;                //使用运算[]存取向量的元素
     cout<<"Vector V has " << V.size()<<" elements:"<<endl;
     for (itr=V.begin(); itr!=V.end();itr++)
       cout<<(*itr) <<" ";    //使用迭代器存取向量元素
     cout<<endl;
     for (itr=V.begin(); itr!=V.end();itr++)
```

```
        *itr = 2*(*itr);
    //如果不修改向量的元素，也可以使用常量迭代器
    vector<int>::const_iterator citr;
    cout << "Now the elements are:"<<endl;
    for (citr=V.begin(); citr!=V.end(); citr++)
      cout<<(*citr) <<" ";
    cout << endl;

    itr = V.begin();
    itr++;                  //itr指向元素V[1]
    vector<int> :: iterator p;
    p = V.insert(itr, 100); //在V[1]前插入100。插入之后，itr可能不合法

    cout << "After insertion the elements are:"<<endl;
    for (citr=V.begin(); citr!=V.end(); citr++)
      cout<<(*citr) <<" ";
    cout << endl;

    V.erase(p);             //删除p所指元素，即100
    cout << "After deletion the elements are:"<<endl;
    for (citr=V.begin(); citr!=V.end(); citr++)
      cout<<(*citr) <<" ";
    return 0;
}
```

程序运行结果:

```
Vector V0 can hold 0 elements.
Vector V0 can hold 256 elements.
Vector V0 has  1 elements.
Vector V has can hold 10 elements.
Vector V has 10 elements:
0 1 2 3 4 5 6 7 8 9
Now the elements are:
0 2 4 6 8 10 12 14 16 18
After insertion the elements are:
0 100 2 4 6 8 10 12 14 16 18
After deletion the elements are:
0 2 4 6 8 10 12 14 16 18
```