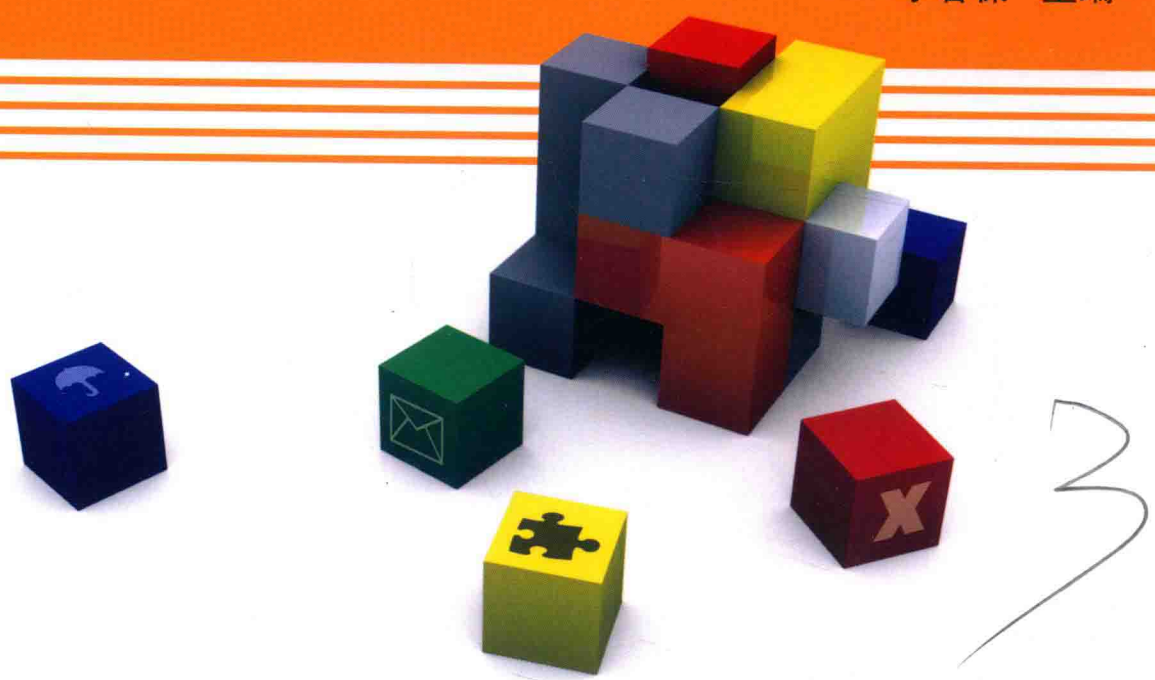
“十二五”普通高等教育本科国家级规划教材
高等学校数据结构课程系列教材

（第5版）

数据结构教程
上机实验指导

李春葆　主编



- 根据相关知识点分为验证性、设计性和综合性实验，包含著名软件公司的面试题。
- 在解答实验过程中，给出算法设计原理、算法调用关系、各函数的功能说明和实验结果。
- 在实验题的设计中，采用结构化编程方法，体现了数据结构中数据组织和数据处理的思想。

清华大学出版社

31=4C2

“十二五”普通高等教育本科国家级规划教材
高等学校数据结构课程系列教材

（第5版）

数据结构教程上机实验指导

李春葆 主编

尹为民 蒋晶珏 喻丹丹 蒋林 编著

清华大学出版社
北 京

内 容 简 介

本书是《数据结构教程(第5版)》(李春葆主编,清华大学出版社出版)的配套上机实验指导。两书章次一一对应,内容包括绪论、线性表、栈和队列、串、递归、数组和广义表、树和二叉树、图、查找、内排序、外排序和文件的实验题解析。每章的实验根据相关知识点分为验证性实验、设计性实验和综合性实验。书中所有程序都在 VC++ 6.0 和 Dev C++ 5 环境下调试通过,读者可以从清华大学出版社网站(http://www.tup.com.cn)免费下载。书中列出了主教材中的全部上机实验题目,其自成一体,也可以脱离主教材单独使用。

本书适合高等院校计算机及相关专业本科生及研究生使用。

本书封面贴有清华大学出版社防伪标签,无标签者不得销售。
版权所有,侵权必究。侵权举报电话:010-62782989 13701121933

图书在版编目(CIP)数据

数据结构教程(第5版)上机实验指导/李春葆主编. —北京:清华大学出版社,2017 (2017.8 重印)
(高等学校数据结构课程系列教材)
ISBN 978-7-302-45586-8

Ⅰ. ①数… Ⅱ. ①李… Ⅲ. ①数据结构—教学参考资料 Ⅳ. ①TP311.12

中国版本图书馆 CIP 数据核字(2017)第 061839 号

责任编辑:魏江江 王冰飞
封面设计:杨 兮
责任校对:时翠兰
责任印制:宋 林

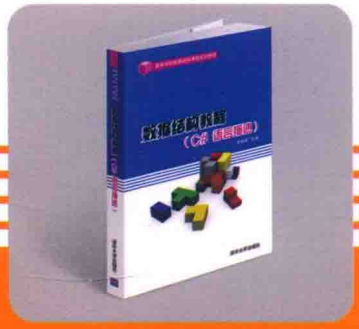
出版发行:清华大学出版社
网 址:http://www.tup.com.cn, http://www.wqbook.com
地 址:北京清华大学学研大厦 A 座 邮 编:100084
社 总 机:010-62770175 邮 购:010-62786544
投稿与读者服务:010-62776969, c-service@tup.tsinghua.edu.cn
质 量 反 馈:010-62772015, zhiliang@tup.tsinghua.edu.cn
课 件 下 载:http://www.tup.com.cn,010-62795954

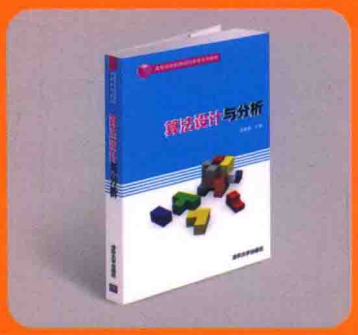
印 装 者:清华大学印刷厂
经 销:全国新华书店
开 本:185mm×260mm **印 张**:21 **字 数**:510 千字
版 次:2017 年 8 月第 1 版 **印 次**:2017 年 8 月第2次印刷
印 数:2001～5000
定 价:45.00 元

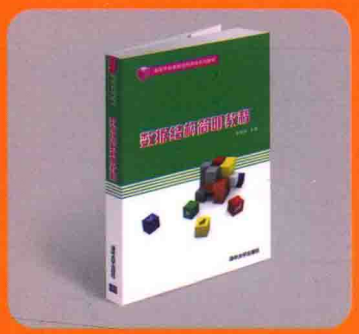
产品编号:072426-01

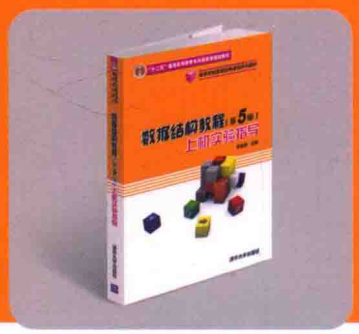
李春葆

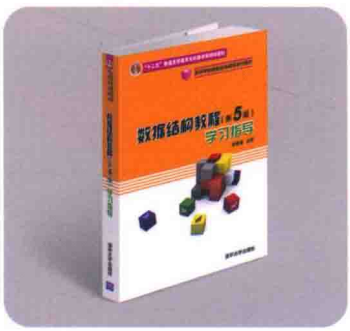
武汉大学计算机学院教授，主要研究方向为数据挖掘和算法设计，先后主持和参加多个大型研究项目。主要为本科生讲授数据结构（15年以上）和软件工程等课程，为研究生讲授软件开发新技术、数据仓库与数据挖掘等课程，并出版十多部精品著作。











试读结束：需要全本请在线购买： www.ertongbook.com

前言 Preface

数据结构实验教学不仅可以使学生巩固对课程中基本原理、基本概念和相关算法的理解和掌握，而且可以改变学生实践教学环节薄弱、动手能力不强的现状，有利于形成以学生为主的学习氛围。该课程实验教学内容以应用型本科教学计划为依据，形成实验体系，涵盖验证性、设计性和综合性的实验，注重学生能力培养。

验证性实验主要是上机实现课程中涉及的相关算法，使学生进一步领会其原理和验证算法的正确性；设计性实验是采用数据结构的基本方法求解问题，学生可以自行设计实验方案并加以实现；综合性实验是综合运用数据结构课程中一章或者多章的内容求解比较复杂的问题，或者同一个问题用多种方法求解。

本书是《数据结构教程（第5版）》（清华大学出版社，以下简称《教程》）的配套上机实验指导。

全书分为12章，第1章绪论，第2章线性表，第3章栈和队列，第4章串，第5章递归，第6章数组和广义表，第7章树和二叉树，第8章图，第9章查找，第10章内排序，第11章外排序，第12章文件。各章次与《教程》的章次相对应。附录中给出了学生提交的实验报告的格式。

每章的实验根据相关知识点分为验证性实验、设计性实验和综合性实验，后两部分中包含一些国内著名软件公司的面试题。在解答实验时给出了算法设计原理、算法调用关系（程序结构图）、各函数的功能说明和实验结果。在实验题的设计中，采用结构化编程方法，体现了数据结构中数据组织和数据处理的思想。

书中所有程序都在VC++ 6.0和Dev C++ 5环境下调试通过，读者可以从清华大学出版社网站（http://www.tup.com.cn）免费下载。

本书列出了《教程》中的全部上机实验题目，其自成一体，也可以脱离《教程》单独使用。

由于水平有限，尽管编者不遗余力，仍可能存在错误和不足之处，敬请教师和同学们批评指正。

编　者

2017年1月

图书资源支持

感谢您一直以来对清华版图书的支持和爱护。为了配合本书的使用，本书提供配套的素材，有需求的用户请到清华大学出版社主页（http://www.tup.com.cn）上查询和下载，也可以拨打电话或发送电子邮件咨询。

如果您在使用本书的过程中遇到了什么问题，或者有相关图书出版计划，也请您发邮件告诉我们，以便我们更好地为您服务。

我们的联系方式：

地　　址：北京海淀区双清路学研大厦 A 座 707

邮　　编：100084

电　　话：010－62770175－4604

资源下载：http://www.tup.com.cn

电子邮件：weijj@tup.tsinghua.edu.cn

QQ：883604（请写明您的单位和姓名）

扫一扫
资源下载、样书申请
新书推荐、技术交流

用微信扫一扫右边的二维码，即可关注清华大学出版社公众号“书圈”。

目录 Contents

第1章 绪论

1.1 验证性实验

实验题 1：求 1～*n* 的连续整数和

目的：通过对比同一问题不同解法的绝对执行时间，体会不同算法的优劣。

内容：编写一个程序 exp1-1.cpp，对于给定的正整数 n，求 1+2+…+n，采用逐个累加和 $n(n+1)/2$（高斯法）两种解法。对于相同的 n，给出这两种解法的求和结果和求解时间，并用相关数据进行测试。

程序 exp1-1.cpp 的结构如图 1.1 所示，add1（逐个累加）和 add2（高斯法）函数采用两种解法计算 1+2+…+n，AddTime1 和 AddTime2 分别调用它们求 1+2+…+n，并统计求解时间。

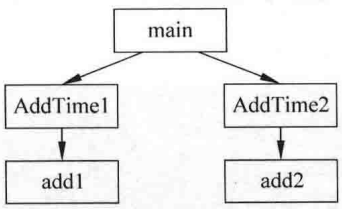


图 1.1 exp1-1.cpp 程序结构

其中，clock_t 是时钟数据类型（长整型数），clock()函数返回 CPU 时钟计时单元数（以毫秒为单位），而 CLOCKS_PER_SEC 是一个常量，表示 1 秒包含的毫秒数。表达式((float)t)/CLOCKS_PER_SEC 返回 t 转换成的秒数。clock_t 类型、clock()函数和 CLOCKS_PER_SEC 常量均在 time.h 头文件中声明。

为了计算一个功能的执行时间，其基本方式如下：

```
clock_t t;                                    //定义时钟变量 t
t = clock();                                  //求调用前的时间
```

调用要计算执行时间的功能函数；

```
t = clock() - t;                                          //求时间差，即该功能的执行时间
printf("用时：%lf 秒\n",((float)t)/CLOCKS_PER_SEC);   //转换为秒单位，再输出
```

exp1-1.cpp 程序的代码如下：

```
#include <stdio.h>
#include <time.h>                          //clock_t, clock, CLOCKS_PER_SEC
#include <math.h>
//------- 方法 1 ------------------------------------------------
long add1(long n)                                  //方法 1：求 1 + 2 + … + n
{   long i,sum = 0;
    for (i = 1;i <= n;i++)
        sum += i;
    return sum;
}
void AddTime1(long n)                              //采用方法 1 的耗时统计
{   clock_t t;
    long sum;
```

```
    t = clock();
    sum = add1(n);
    t = clock() - t;
    printf("方法 1:\n");
    printf("  结果:1～ %d 之和: %ld\n",n,sum);
    printf("  用时: %lf 秒\n" ,((float)t)/CLOCKS_PER_SEC);
}
//------ 方法 2 ------------------------------------------------
long add2(long n)                    //方法 2: 求 1 + 2 + … + n
{
    return n * (n + 1)/2;
}
void AddTime2(long n)                //采用方法 2 的耗时统计
{   clock_t t;
    long sum;
    t = clock();
    sum = add2(n);
    t = clock() - t;
    printf("方法 2:\n");
    printf("  结果:1～ %d 之和: %ld\n",n,sum);
    printf("  用时: %lf 秒\n" ,((float)t)/CLOCKS_PER_SEC);
}
//-----------------------------------------------------------------
int main()
{   int n;
    printf("n(大于 1000000):");
    scanf("%d",&n);
    if (n<1000000) return 0;
    AddTime1(n);
    AddTime2(n);
    return 1;
}
```

其中，add1(n)采用逐个累加法求和，对应算法的时间复杂度为 $O(n)$；而 add2(n)采用高斯法求和，对应算法的时间复杂度为 $O(1)$。

exp1-1.cpp 程序的一次执行结果如图 1.2 所示。对于 $n=99999999$，采用逐个累加法花费 0.234 秒，而高斯法花费的时间几乎可以忽略不计。

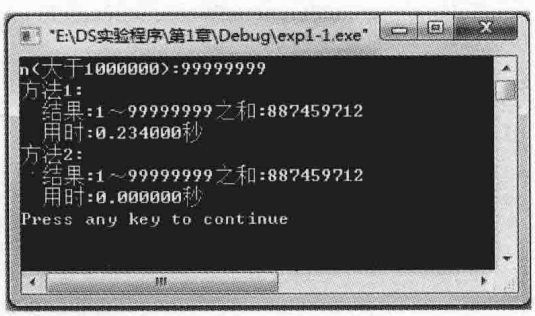


图 1.2 exp1-1.cpp 程序执行结果

实验题 2：常见算法时间函数的增长趋势分析

目的：理解常见算法时间函数的增长情况。

内容：编写一个程序 exp1-2.cpp，对于 1～n 的每个整数 n，输出 $\log_2 n$、$\sqrt{n}$、n、$n\log_2 n$、n^2、n^3、2^n 和 $n!$ 的值。

程序 exp1-2.cpp 的结构如图 1.3 所示，输出 1～10 的 $\log_2 n$、$\sqrt{n}$、n、$n\log_2 n$、n^2、n^3、2^n 和 $n!$ 的值。

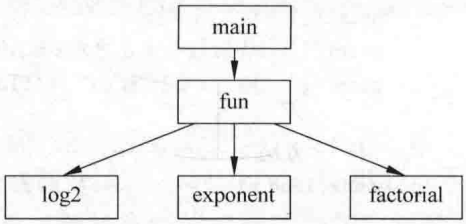


图 1.3 exp1-1.cpp 程序结构

exp1-2.cpp 程序的代码如下：

```
#include<stdio.h>
#include<math.h>
double log2(double n)                          //求 log2(n)
{
    return log10(n)/log10(2);
}
long exponent(int n)                           //求 2^n
{   long s=1;
    for (int i=1;i<=n;i++)
        s*=2;
    return s;
}
long factorial(int n)                          //求 n!
{   long s=1;
    for (int i=1;i<=n;i++)
        s*=i;
    return s;
}
void fun(int n)
{   printf("log2(n) sqrt(n)  n     nlog2(n)   n^2      n^3      2^n      n!\n");
    printf("==========================================================\n");
    for (int i=1;i<=n;i++)
    {   printf("%5.2f\t",log2(double(i)));
        printf("%5.2f\t",sqrt(i));
        printf("%2d\t",i);
        printf("%7.2f\t",i*log2(double(i)));
        printf("%5d\t",i*i);
        printf("%7d\t",i*i*i);
        printf("%8d\t",exponent(i));
        printf("%10d\n",factorial(i));
    }
}
int main()
{   int n=10;
```

```
    fun(n);
    return 1;
}
```

exp1-2.cpp 程序的执行结果如图 1.4 所示。从中看到，时间函数按照 $\log_2 n$、$\sqrt{n}$、n、$n\log_2 n$、n^2、n^3、2^n和 $n!$ 的顺序随 n 增长越来越快。

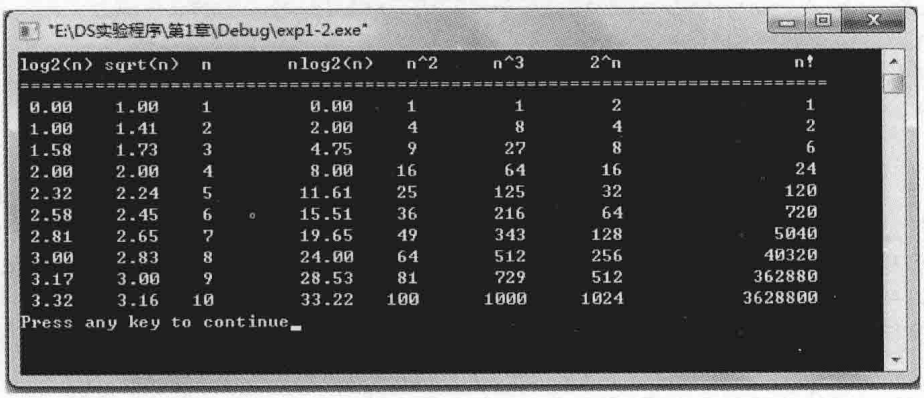


图 1.4 exp1-2.cpp 程序执行结果

1.2 设计性实验

实验题 3：求素数个数

目的：通过对比同一问题不同解法的绝对执行时间，体会如何设计“好”的算法。

内容：编写一个程序 exp1-3.cpp，求 1～n 的素数个数。给出两种解法。对于相同的 n，给出这两种解法的结果和求解时间，并用相关数据进行测试。

程序 exp1-3.cpp 的结构如图 1.5 所示，prime1(n)函数采用传统方法判断 n 是否为素数，称为方法 1，对应的时间复杂度为 $O(n)$。prime2(n)函数采用改进方法判断 n 是否为素数，称为方法 2，对应的时间复杂度为 $O(\sqrt{n})$。PrimeTime1 和 PrimeTime2 分别调用上述两个函数求 1～n 的素数个数，并统计求解时间。

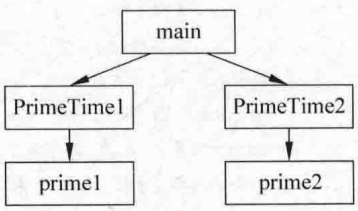


图 1.5 exp1-3.cpp 程序结构

有关程序执行时间的计算方法参见本章实验题 1。

exp1-3.cpp 程序的代码如下：

```
#include <stdio.h>
#include <time.h>                    //clock_t、clock、CLOCKS_PER_SEC
#include <math.h>
//------ 方法 1 ------------------------------------------------
bool prime1(long n)                  //方法 1：判断正整数 n 是否为素数
```

```
{   long i;
    for (i = 2;i < n;i++)
        if (n % i == 0)
            return false;                          //若 n 不是素数,则退出并返回 false
    return true;
}
void PrimeTime1(long n)                            //采用方法 1 的耗时统计
{   clock_t t;
    long sum = 0,i;
    t = clock();
    for (i = 2;i <= n;i++)
        if (prime1(i))
            sum++;
    t = clock() - t;
    printf("方法 1:\n");
    printf("  结果:2～ %d 的素数个数: %d\n",n,sum);
    printf("  用时: %lf 秒\n" ,((float)t)/CLOCKS_PER_SEC);
}
//------ 方法 2 ------------------------------------------------
bool prime2(long n)                                //方法 2: 判断正整数 n 是否为素数
{   long i;
    for (i = 2;i <= (int)sqrt(n);i++)
        if (n % i == 0)
            return false;                          //若 n 不是素数,则退出并返回 false
    return true;
}
void PrimeTime2(long n)                            //采用方法 2 的耗时统计
{   clock_t t;
    long sum = 0,i;
    t = clock();
    for (i = 2;i <= n;i++)
        if (prime2(i))
            sum++;
    t = clock() - t;
    printf("方法 2:\n");
    printf("  结果:2～ %d 的素数个数: %d\n",n,sum);
    printf("  用时: %lf 秒\n" ,((float)t)/CLOCKS_PER_SEC);
}
//-----------------------------------------------------------------
int main()
{   long n;
    printf("n(大于 100000):");
    scanf("%d",&n);
    if (n<10000) return 0;
    PrimeTime1(n);
    PrimeTime2(n);
    return 1;
}
```

exp1-3.cpp 程序的一次执行结果如图 1.6 所示。对于 $n=234567$，采用方法 1 花费了 7.109 秒，而采用方法 2 仅仅花费了 0.097 秒。

图 1.6 exp1-3.cpp 程序执行结果

实验题 4：求连续整数阶乘的和

目的：体会如何设计“好”的算法。

内容：编写一个程序 exp1-4.cpp，对于给定的正整数 n，求 $1!+2!+3!+\cdots+n!$。给出一种时间复杂度为 $O(n)$ 的解法。

程序 exp1-4.cpp 的结构如图 1.7 所示，Sum(n)函数用于计算 $1!+2!+\cdots+n!$，其思路是：i 从 1 开始，fact=$i!$，sum 用于累加 fact；当 i 增 1 时，fact=fact×i，再将 fact 累加到 sum 中，以此类推，直到 $i=n$ 为止，算法的时间复杂度为 $O(n)$。如果对于每个 $m!$（$1\leqslant m\leqslant n$），都从 1 到 m 做连乘，对应算法的时间复杂度为 $O(n^2)$。

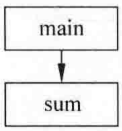


图 1.7 exp1-4.cpp 程序结构

exp1-4.cpp 程序的代码如下：

```
#include <stdio.h>
long Sum(int n)
{   long sum = 0,fact = 1;
    for (int i = 1;i <= n;i++)
    {   fact * = i;                          //求出 fact = i!
        sum += fact;                         //求出 sum = 1! + 2! + … + i!
    }
    return sum;
}
int main()
{   int n;
    printf("n(3 - 20):");
    scanf("%d",&n);
    if (n<3 || n>20) return 0;
    printf("1! + 2! + … + %d!= %ld\n",n,Sum(n));
    return 1;
}
```


试读结束：需要全本请在线购买：www.ertongbook.com