信息与计算科学教学丛书

算法与数据结构实验指导

江世宏　编著

科学出版社

信息与计算科学教学丛书

算法与数据结构实验指导

江世宏　编著

科学出版社

北京

内 容 简 介

本书为《算法与数据结构》配套的实验指导书。

本书按照课程的讲授顺序，提供了数组、链表、算法与数据结构、线性表、栈、递归、队列、树、图、排序、搜索、散列、模板的相关实验。每个实验，给出了实验内容与实验目的，对解决实验内容所提出的问题，进行了详细分析，并给出解决方案和完整程序。

通过亲手做实验，读者能更好地掌握算法与数据结构的思想和方法，深化对基本概念、基本性质和基本结论的理解，提高分析与解决问题的能力。

本书可作为信息专业与计算机专业学习算法与数据结构的参考书，也可作为任课教师的教学参考书，还可以作为编程爱好者学习编程的参考书。

图书在版编目(CIP)数据

算法与数据结构实验指导/江世宏编著. —北京：科学出版社，2016.5

（信息与计算科学教学丛书）

ISBN 978-7-03-048191-7

Ⅰ.①算… Ⅱ.①江… Ⅲ.①算法分析—实验—高等学校—教材 ②数据结构—实验—高等学校—教材 Ⅳ.①TP301.6-33 ②TP311.12-33

中国版本图书馆 CIP 数据核字(2016)第 093763 号

责任编辑：王雨舸/责任校对：董艳辉

责任印制：彭 超/封面设计：蓝 正

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

武汉市首壹印务有限公司印刷

科学出版社发行 各地新华书店经销

*

开本：787×1092 1/16

2016 年 5 月第 一 版 印张：9 1/2

2016 年 5 月第一次印刷 字数：221 000

定价：26.80 元

（如有印装质量问题，我社负责调换）

前　言

在《算法与数据结构》一书中，我们讨论了一些常见的数据结构，即数据的组织形式和与之绑定的相关运算，给出了标准的程序代码，为开发求解实际问题的应用程序，奠定了初步基础。

欲开发求解实际问题的应用程序，首先要确定数据应采用哪种已知的数据结构进行存储，然后根据选定的数据结构，去寻找实际问题的求解方法，并将它转化成面向计算机编程的算法，即

程序＝数据结构＋算法

这样做有利于将应用程序的开发变成对已知数据结构的复用，减少程序代码的撰写量，使程序的组织形式更趋一致，将应用程序的开发重点，集中到为求解实际问题的方法寻找与算法描述上。

一般地，求解实际问题的算法依附于数据结构的选取，不同的数据结构会导致不同的算法。不恰当的数据结构，往往不利于求解实际问题的算法设计与实现，甚至导致算法设计与实现陷入困境。反之，当求解实际问题的算法难以设计与实现时，应思考所选择的数据结构是否恰当，试着去尝试其他的数据结构。

学习“算法与数据结构”这门课程的目的是提升我们利用计算机编程解决实际问题的能力，而这种能力的获取与提升，主要靠自己亲自动手做实验。在做每个实验时，应特别关注求解实际问题的方法探索与算法描述，撰写程序代码应尽量“多抄少写”。“多抄”是指，对已知数据结构、经典算法和常规函数应尽可能地复用（即拷贝）；“少写”是指，仅为求解问题所需的自定义函数编写代码。

第 1～13 章的实验，是与教材《算法与数据结构》相配套的实验，主要包括以下几类实验：一是对教材中所讨论过的数据结构的应用举例；二是教材中尚未讨论的数据结构，如带附加头结点的链表、优先级链队、二叉树的广义表生成法等；三是具有实用价值的算法介绍，如求全排列、随机排队、轮盘抽取、贪心法和关键路径等；四是求数学问题的数值解，以及数学结论的数值模拟证明，如求极限、行列式、数学期望、定积分等，证明伯努利大数定律、棣莫弗-拉普拉斯中心极限定理等；五是对算法与数据结构中典型问题的进一步研讨，如时间复杂度估计、哈夫曼树的 WPL 计算、图的 DFS 与 BFS 算法、二叉搜索树的 ASL 估计、排序中的指针移动、哈希函数地址的均匀性、面向过程设计与面向对象设计的关系等。设置这些实验的目的是，介绍已知数据结构的应用实例，向读者开启一扇“计算机应用窗口”，拓宽视野；了解一些常见算法；学会利用计算机辅助学习，建立起所学知识之间的横向联系；深化对算法与数据结构知识点的理解。

第 14 章的实验是专门为《算法与数据结构》课程设计（论文）而设置的。通过两个实验介绍课程设计（论文）的撰写内容与大致格式。配置练习为课程设计（论文）拟定选题。选题取自数学分析、概率统计、运筹学和图论，以及现代颇为流行的模拟算法。

希望广大编程爱好者，能通过“算法与数据结构”这门课程的学习，以及亲自动手做实验，领悟到编程的“真谛”，成为一名真正的程序高手。

本书在编写过程中的主要参考资料一并在参考文献中列出，对这些专家教授给予本书的帮助，作者表示衷心的感谢。鉴于作者的水平有限，疏漏与不妥之处在所难免，恳请专家和读者批评指正。

编　者

2016 年 3 月

目　　录

第1章 数 组

实 验 1.1

已知数列

$$x_n=\begin{cases}1 & n=1\\ 1+\dfrac{1}{2}+\cdots+\dfrac{1}{n}-\ln n & n=2,3,\cdots\end{cases}$$

对于任意给定 $\varepsilon=10^{-5}$，用柯西收敛准则判断其敛散性；若该数列收敛，给出所需的 N 以及极限的近似值。

【实验目的】

了解如何将一个数学问题转化成一个计算机问题。掌握使用数组或不使用数组解决此问题的方法，体会各自的优缺点。掌握在数列收敛的条件下，求其极限近似值的方法。

【参考解答】

1. 解决方案的分析与设计

柯西准则的数学描述是这样的：

数列$\{x_n\}$收敛的充要条件是任给充分小的正数$\varepsilon>0$，存在正整数$N>0$，当$n\geqslant N$时，对任意给定的正整数p，恒有不等式$|x_{n+p}-x_n|<\varepsilon$成立。

从这一准则可以看出，对于任给充分小的正数ε，只要能找到正整数N，使得

$$|x_{n+p}-x_n|<\varepsilon \quad (n=N,N+1,\cdots;\ p=1,2,\cdots)$$

恒成立，就可以断定数列$\{x_n\}$收敛。

用此准则，靠计算机来判定数列的收敛性，并不方便。为此，对它稍作修改

$$x_{n+p}-x_n=\sum_{k=0}^{p-1}(x_{n+k+1}-x_{n+k})$$

若$|x_{n+k+1}-x_{n+k}|<\dfrac{\varepsilon}{p}$，则

$$|x_{n+p}-x_n|\leqslant\sum_{k=0}^{p-1}|x_{n+k+1}-x_{n+k}|<\sum_{k=0}^{p-1}\frac{\varepsilon}{p}=\varepsilon$$

因此，准则可修改为以下形式：

任给充分小的正数$\varepsilon>0$，存在正整数$N>0$，当$n\geqslant N$时，对任意给定的正整数p，以下不等式

$$|x_{n+k+1}-x_{n+k}|<\frac{\varepsilon}{p} \quad (k=0,1,\cdots,p-1)$$

均成立，则数列$\{x_n\}$收敛。

当数列$\{x_n\}$收敛时，则极限$\lim\limits_{n\to\infty}x_n=x$存在，即当$n\geqslant N$时，对任意的正整数$p$恒有$|x_{n+p}-x_n|<\varepsilon$，如果令$p\to\infty$，则有$|x-x_n|\leqslant\varepsilon$成立。

这表明，如果取x_N作为其极限值x的近似，其误差不超过ε。

若用双精度实型数组 x[]来存储该数列的各项，而数列项数在理论上是无穷的，为了尽可能多地存储数列项，必须将数组的最大容量 MaxSize 设置得尽可能大。而 C/C++允许定义的双精度实型数组的最大容量总是有限的，这里，我们预设 MaxSize=120000。

数列可通过以下递推式

$$\begin{cases} x_1=1 \\ x_{n+1}=x_n+\dfrac{1}{n+1}-\ln\left(1+\dfrac{1}{n}\right) & n=1,2,\cdots \end{cases}$$

存储到数组 x[MaxSize]（需先定义 x[1]=1）。

关于任取正整数p，为了明确起见，不妨假定$1\leqslant p\leqslant 100$。通过生成 1～100 之间的随机数来实现。

2. 解决方案的程序实现

```
#include<iostream>
#include<cmath>
#include<ctime>
using namespace std;
#define MaxSize 120000                          //数组的最大容量
//任取 1～100 间的正整数
int Any()
{   return rand()%100+1;}
//主函数
void main()
{   double x[MaxSize],epsilon=0.00001;
    int p,k,n,N,flag;
    srand(time(NULL));                          //设置随机数发生器种子
    p=Any();                                    //任取正整数 p
    x[1]=1;                                     //数组第 1 项赋初值
    for(n=1;n<MaxSize;n++)
    {   for(k=0;k<p;k++)
        {   //由 x[n+k]生成 x[n+k+1]
            x[n+k+1]=x[n+k]+1.0/(n+k+1)-log(1+1.0/(n+k));
            if(fabs(x[n+k+1]-x[n+k])>=epsilon/p)  //如果不满足收敛条件
```

```
            {   flag=0;                          //置标识变量 flag=0
                break;                           //退出内循环,再进入外循环
            }
            else                                 //如果满足收敛条件
            {   flag=1;                          //置标识变量 flag=1
                continue;                        //继续内循环
            }
        }
        if(flag==0)
            continue;                            //继续外循环
        else
        {   N=n;                                 //取 N
            break;                               //退出外循环
        }
    }
    if(flag==0)
        cout<<"数列不一定收敛"<<endl;
    else
        cout<<"数列收敛,极限约为 x="<<x[N]
        <<",epsilon="<<epsilon<<",p="<<p<<",N="<<N<<endl;
}
```

程序运行的结果如图 1.1 所示。

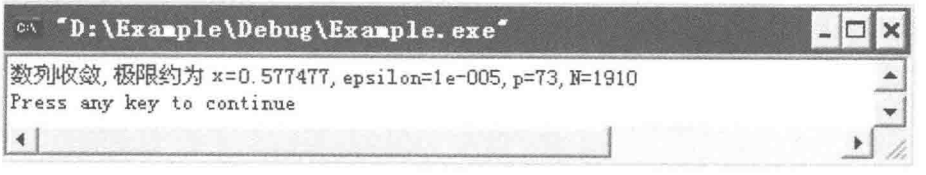

图 1.1 实验 1.1 程序运行结果

需特别说明的是,该数列的极限是著名的欧拉常数 $x^*=0.57721\cdots$。该数列极限的存在性,说明等价关系式 $1+\frac{1}{2}+\cdots+\frac{1}{n}=O(\ln n)$ 是成立的。

3. 解决方案的完善

用大维数数组 x[MaxSize]来存储数列各项,既不完美,也没有必要。如果用 MaxSize 表示数列的最大项数,可用 4 个变量 x1,x2,y1,y2 替代数组 x[MaxSize],改进上述程序。

```
#include<iostream>
#include<cmath>
```

```
#include<ctime>
using namespace std;
#define MaxSize 120000                    //数列的最大项数
//任取 1～100 间的正整数
int Any()
{  return rand()%100+1;}
//主函数
void main()
{  double x1,x2,y1,y2,epsilon=0.00001;
   int p,k,n,N,flag;
   srand(time(NULL));                     //设置随机数发生器种子
   p=Any();                               //任取正整数 p
   x1=1;                                  //数组第 1 项赋初值
   for(n=1;n<MaxSize;n++)
   {  y1=x1;
      for(k=0;k<p;k++)
      {  //由 y[n+k]生成 y[n+k+1]
         y2=y1+1.0/(n+k+1)-log(1+1.0/(n+k));
         if(fabs(y2-y1)>=epsilon/p)
         {  flag=0;
            break;                        //退出内循环
         }
         else
         {  flag=1;
            y1=y2;                        //更新内循环迭代初值
            continue;
         }
      }
      if(flag==0)
      {  //由 x[n]生成 x[n+1]
         x2=x1+1.0/(n+1)-log(1+1.0/n);
         x1=x2;                           //更新外循环迭代初值
         continue;
      }
      else
      {  N=n;//取 N
         break;
      }
```

```
    }
    if(flag==0)
        cout<<"数列不一定收敛"<<endl;
    else
        cout<<"数列收敛,极限约为 x="<<x2
        <<",p="<<p<<",N="<<N<<endl;
}
```

实 验 1.2

给出十进制与十六进制之间的转换程序。

【实验目的】

了解十进制与十六进制之间转换算法,掌握字符数组的使用方法。

【参考解答】

1. 解决方案分析与设计

1) 十六进制数转换为十进制数

十六进制数字与十进制数字的对应关系如表 1-1 所示。

表 1-1

十六进制	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
十进制	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

在 C/C++中,一般用 8 位的十六进制数来表示内存地址。根据表 1-1,表示内存地址的十六进制字符串 str16=001A2B3C,对应着八维整型数组 n[8]={0,0,1,10,2,11,3,12},而该整型数组所表示的十进制数为

$$x=0\times16^7+0\times16^6+1\times16^5+10\times16^4+2\times16^3+11\times16^2+3\times16^1+12\times16^0$$

数 x 的计算可采用秦九韶算法。即

取 x=n[0],对于 $i=1,2,\cdots,7$,执行迭代 x=16 * x+n[i]

因此,十六进制数转换为十进制数的算法如下:

第一步 输入 9 维十六进制字符数组 str16[9]。

第二步 据表 1-1,将 str16[9]转化成为十进制整数数组 n[8]。

第三步 据秦九韶算法,将 n[8]转换成十进制数 x。

第四步 输出 x。

2) 十进制数转换为十六进制数

第一步 输入十进制数 x,置 n[8]={0,0,0,0,0,0,0,0},k=0。

第二步　当 x>15 时，反复做以下操作：

1　r=x%16

2　n[k]=r

3　x=(x−r)/16

4　k++

第三步　n[k]=r

第四步　据表 1-1，将 n[8]转化为十六进制字符数组 str16[9]。

第五步　输出 str16[9]。

2. 解决方案的程序实现

```
#include<iostream>
using namespace std;
//十六进制转换为十进制
void Hex2Dec(char str16[9])
{  int i,n[8],x;
   for(i=0;i<8;i++)
   {  //将字符数组 str16[9]转化为整数数组 n[8]
      switch(str16[i])
      {  case 'A':
            n[i]=10;break;
         case 'B':
            n[i]=11;break;
         case 'C':
            n[i]=12;break;
         case 'D':
            n[i]=13;break;
         case 'E':
            n[i]=14;break;
         case 'F':
            n[i]=15;break;
         default:
            n[i]=str16[i]-48;break;
      }
   }
   //用秦九韶算法求十进制数
   x=n[0];
   for(i=1;i<8;i++)
      x=16 * x+n[i];
```

```
    cout≪"十六进制数"≪str16≪"的十进制数为"≪x≪endl;
}
//十进制转换为十六进制
void Dec2Hex(int x)
{   char str16[9];              //定义一个9维十六进制字符数组
    int xbak,n[8]={0,0,0,0,0,0,0,0},i,r,k=0;
    xbak=x;                     //x备份
    while(x>15)
    {   r=x%16;
        n[k]=r;
        x=(x-r)/16;
        k++;
    }
    n[k]=x;
    for(i=7;i>=0;i--)
    {   //十进制整数组n[8]转化为十六进制字符数组str16[9]
        switch(n[i])
        {   case 10:
                str16[7-i]='A';break;
            case 11:
                str16[7-i]='B';break;
            case 12:
                str16[7-i]='C';break;
            case 13:
                str16[7-i]='D';break;
            case 14:
                str16[7-i]='E';break;
            case 15:
                str16[7-i]='F';break;
            default:
                str16[7-i]=n[i]+48;break;
        }
    }
    str16[8]='\0';                          //添加字符数组的结束符
    cout≪"十进制数"≪xbak≪"的十六进制数为"≪str16≪endl;
}
//主函数
void main()
```

```
{   char str16[9]="001A2B3C";          //定义一个十六进制字符数组
    Hex2Dec(str16);                    //调用十六进制转换为十进制函数
    int x=1715004;                     //定义一个十进制数
    Dec2Hex(x);                        //调用十进制转换为十六进制函数
}
```

程序运行的结果如图 1.2 所示。

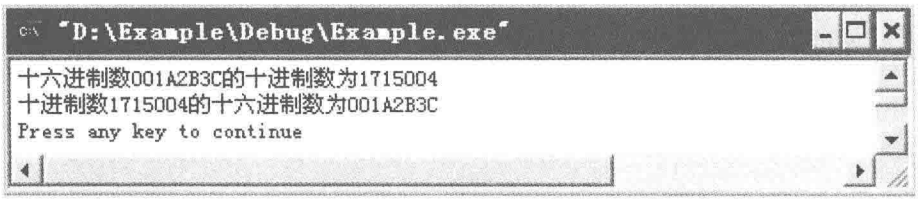

图 1.2　实验 1.2 程序运行结果

练　习　1

1. 已知数列

$$x_n=\begin{cases}2 & n=1\\ \dfrac{1}{2}\left(x_{n-1}+\dfrac{2}{x_{n-1}}\right) & n=2,3,\cdots\end{cases}$$

对于任意给定 $\varepsilon=10^{-5}$，用柯西收敛准则判断该数列收敛，给出其极限的近似值，用数学方法证明：$\lim\limits_{n\to\infty}x_n=\sqrt{2}$。

2. 给出二进制与十进制的转换程序。

3. 在一个 4×5×6 的三维数组中，有哪些元素“以行优先存储”与“以列优先存储”被安排在相同的位置，试用枚举法进行求解。

第2章　链　　表

实　验　2.1

对于表 2-1 所给出的学生信息，试用前插法将它们存放到一个单链表中。

表 2-1

编号	1	2	3	4	5
姓名	黄小华	方小源	林大辉	孙阿毛	王小明
成绩	85	95	68	72	79

【实验目的】

掌握单链表的前插创建法。

【参考解答】

```
#include<iostream>
using namespace std;
//结点定义
struct Node
{   int num,score;              //编号,成绩
    char name[8];               //姓名
    Node *next;                 //指向下一个结点的指针
};
Node *head;                     //定义指向单链表头结点的全局指针变量
//前插法创建单链表
void Create()
{   int num,score;
    char name[8];
    Node *NewNode;              //新结点
    cout<<"请输入学生编号、姓名、成绩"<<endl;
    cout<<"在编号输入时,若键入-1,则中止输入"<<endl;
    while(1)
    {   cout<<"编号:";cin>>num;
        if(num==-1)
```

```
            break;
        else
        {   cout<<"姓名:";cin>>name;
            cout<<"成绩:";cin>>score;
            NewNode=new Node;                    //新结点申请内存
            NewNode->num=num;                    //新结点的编号域赋值
            strcpy(NewNode->name,name);//新结点的姓名域赋值
            NewNode->score=score;                //新结点的成绩域赋值
            NewNode->next=NULL;                  //新结点的指针域赋值
            if(head==NULL)                       //如果单链表为空
                head=NewNode;                    //让新结点成为单链表头结点
            else//如果单链表非空
            {   NewNode->next=head;
                head=NewNode;
            }
        }
    }
}
//输出单链表
void Display()
{   Node *p=head;                                //定义探测指针 p 并指向头结点
    cout<<"编号"<<"\t"<<"姓名"<<"\t"<<"成绩"<<endl;
    while(p!=NULL)
    {   cout<<p->num<<"\t"<<p->name<<"\t"<<p->score<<endl;
        p=p->next;//p 后移
    }
}
//主函数
void main()
{   head=new Node;                               //头结点申请内存
    head=NULL;                                   //创建空链表
    Create();                                    //创建单链表
    Display();                                   //输出单链表
}
```

程序运行的结果如图 2.1 所示。