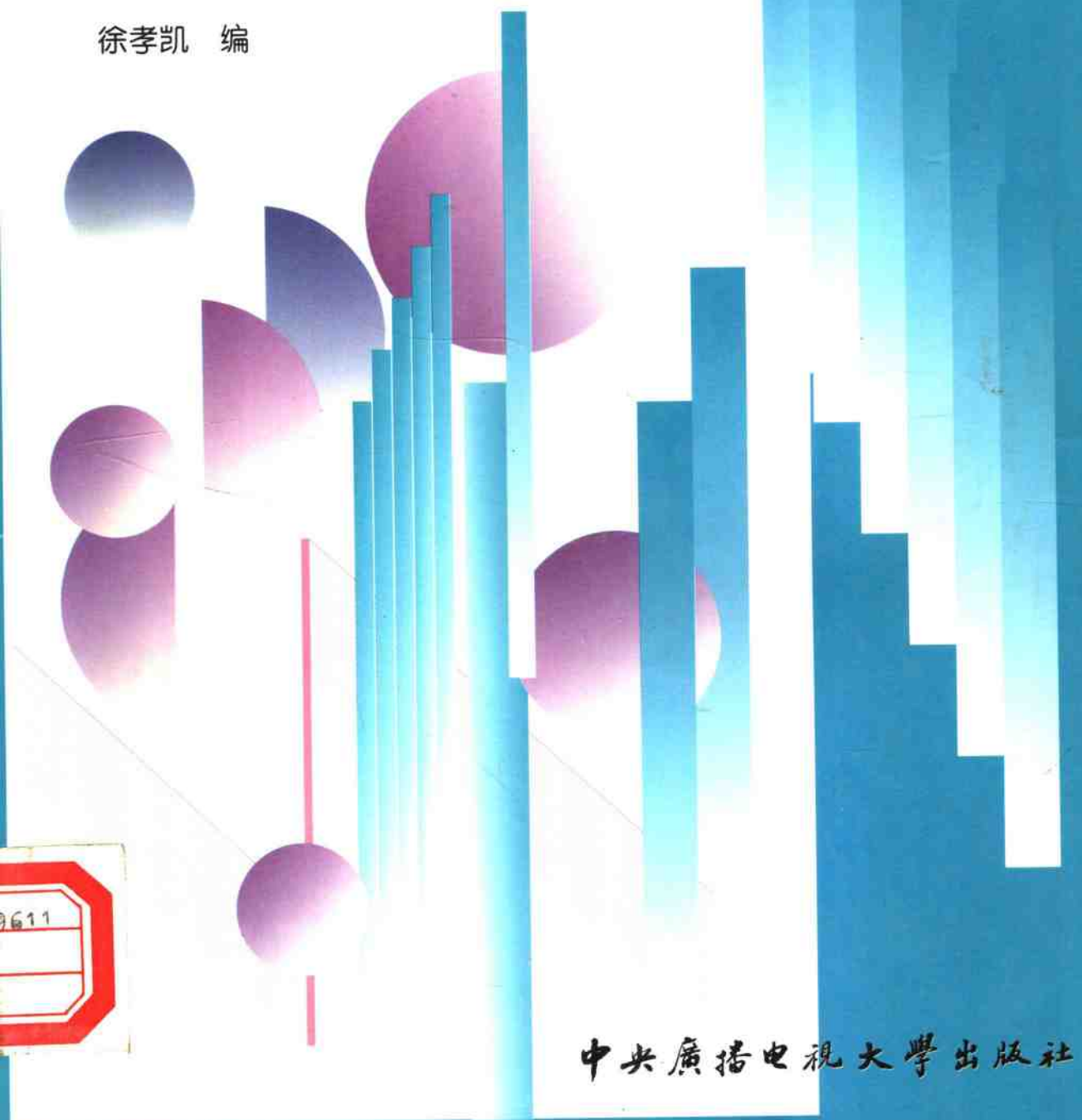
教育部人才培养模式改革和开放教育试点教材

计算机应用专业系列教材

数据结构实验

徐孝凯 编



中央广播电视大学出版社

计算机应用专业系列教材

数据结构实验

徐孝凯　编

中央广播电视大学出版社

图书在版编目(CIP)数据

数据结构实验 / 徐孝凯编. —北京：中央广播电视大学出版社，2001.10

计算机应用专业系列教材

ISBN 7-304-02166-7

Ⅰ.数… Ⅱ.徐… Ⅲ.数据结构-实验-电视大学-教材 Ⅳ.TP311.12-33

中国版本图书馆 CIP 数据核字(2001)第 076527 号

计算机应用专业系列教材

数据结构实验

徐孝凯 编

出版·发行/中央广播电视大学出版社

经销/新华书店北京发行所

印刷/北京集惠印刷有限公司

开本/787×1092 1/16 印张/13 字数/320 千字

版本/2001 年 9 月第 1 版 2002 年 12 月第 4 次印刷

印数/46501-77500

社址/北京市复兴门内大街 160 号 邮编/100031

电话/66419791 68519502 (本书如有缺页或倒装，本社负责退换)

书号：ISBN 7-304-02166-7/TP·175

定价：19.00 元

前　言

数据结构研究的是数据的各种逻辑结构、存储结构和相应运算的方法。实验是该课程教学必不可少的重要组成部分。通过上机实验，即调试和运行已经学习到的算法，或开发新的算法，不仅能够加深理解和巩固所学的知识，而且更重要的是有助于提高自己根据实际问题分析、设计、开发和调试算法的实际能力和水平。

本书是全国广播电视大学系统计算机应用专业数据结构课程的实验教材，与中央广播电视大学出版社出版的、许卓群主编的《数据结构》主教材配套使用。该实验教材共包含9个实验和两个大作业。9个实验依次为：线性表操作、单链表操作、表达式计算、二叉树操作、二叉搜索树的操作、图的运算、散列表操作、外存文件的排序操作，以及二叉搜索树与文件操作等。两个大作业分别为带索引文件的插入、删除和查找操作及散列文件的插入、删除和查找操作。要求学生至少要完成7个实验和一个大作业的实验任务，否则为实验考查不通过，不具备参加期末卷面考试的资格。

本书中的每个实验都由实验目的、实验内容和实验要求三部分组成，在实验内容中通常既给出面向过程的程序设计算法，又给出面向对象的程序设计算法，同学们可从中选择一个运行。当然，若条件允许，最好都能够运行，如此将受益匪浅。

由于本人水平有限，加之时间仓促，错误和不当之处在所难免，敬请广大师生批评指正。

电子邮件地址：xuxk@crtvu.edu.cn

联系电话：010－66052930

徐孝凯

2001年9月4日

目　录

实验部分

大作业部分

附　录

实验部分

实验一　线性表操作

实验目的

1. 会定义线性表的顺序存储类型。

2. 熟悉 C++ 程序的基本结构，掌握程序中的用户头文件、实现文件和主文件之间的相互关系及各自作用。

3. 熟悉对线性表的一些基本操作和具体的函数定义。

4. 熟悉 C++ 操作环境的使用以及多文件程序的输入、编辑、调试和运行的全过程。

实验内容

程序 1：

该程序的功能是对元素类型为整型的顺序存储的线性表进行一些操作。该程序包含三个文件，第一个为头文件，用来存储定义的线性表结构类型以及对线性表进行的各种操作的函数声明；第二个为线性表操作的实现文件，用来存储每一种线性表操作的具体函数定义；第三个为主文件，用来定义线性表和调用线性表的一些操作，实现对给定线性表的具体运算。整个程序如下：

```
//头文件 linearlist1.h

//定义 ElemType 为 int 类型
typedef int ElemType;

//线性表顺序存储类型
struct LinearList
{
    ElemType * list;   //存线性表元素
    int size;          //存线性表长度
```

```
    int MaxSize;        //存 list 数组长度
};

//初始化线性表
  void InitList(LinearList& L, int ms);
//清空线性表
  void ClearList(LinearList& L);
//求线性表长度
  int ListSize(LinearList& L);
//检查线性表是否为空
  bool ListEmpty(LinearList& L);
//检查线性表是否为满
  bool ListFull(LinearList& L);
//遍历线性表
  void TraverList(LinearList& L);
//从线性表中查找元素
  bool FindList(LinearList& L, ElemType& item);
//更新线性表中的给定元素
  bool UpdateList(LinearList& L, const ElemType& item);
//向线性表插入元素
  bool InsertList(LinearList& L, const ElemType& item, int mark);
//从线性表中删除元素
  bool DeleteList(LinearList& L, ElemType& item, int mark);
//对线性表进行有序输出
  void OrderOutputList(LinearList& L, int mark);

//实现文件 linearlist1.cpp

#include<iomanip.h>
#include<stdlib.h>

#include"linearlist1.h"

//初始化线性表
void InitList(LinearList& L, int ms)
{
    L.list=new ElemType[ms];
    if(!L.list){
        cerr<<"Memory allocation failure!"<<endl;
```

```
        exit(1);
    }
    L.size=0;
    L.MaxSize=ms;
}

//清空线性表
void ClearList(LinearList& L)
{
    L.size=0;
{

//求线性表长度
int ListSize(LinearList& L)
{
    return L.size;
}

//检查线性表是否为空
bool ListEmpty(LinearList& L)
{
    return L.size==0;
}

//检查线性表是否为满
bool ListFull(LinearList& L)
{
    return L.size==L.MaxSize;
}

//遍历线性表
void TraverList(LinearList& L)
{
    for(int i=0; i<L.size; i++) cout<<L.list[i]<<' ';
    cout<<endl;
}

//从线性表中查找元素
bool FindList(LinearList& L, ElemType& item)
```

```
}
    for(int i=0; i<L.size; i++)
        if(L.list[i]==item) {
            item=L.list[i];
            return true;
        }
    return false;
}

//更新线性表中的给定元素
bool UpdateList(LinearList& L, const ElemType& item)
{
    for(int i=0; i<L.size; i++)
        if(L.list[i]==item) {
            L.list[i]=item;
            return true;
        }
    return false;
}

//向线性表的表头、表尾或合适位置插入元素
bool InsertList(LinearList& L, const ElemType& item, int mark)
{
    if(ListFull(L)) return false;
    if(mark>0) {  //向表头插入元素
    for(int i=L.size-1; i>=0; i--)
        L.list[i+1]=L.list[i];
    L.list[0]=item;
    }
    else if(mark<0) L.list[L.size]=item;  //向表尾插入元素
    else {for(int i=0; i<L.size; i++)     //向有序表插入元素
            if(item<L.list[i]) break;
        for(int j=L.size-1; j>=i; j--)
            L.list[j+1]=L.list[j];
        L.list[i]=item;
    }
    L.size++;
    return true;
}
```

```
//从线性表中删除表头、表尾或等于给定值的元素
bool DeleteList(LinearList& L, ElemType& item, int mark)
{
    if(ListEmpty(L)) return false;
    if(mark>0) {  //删除表头元素
        item=L.list[0];
        for(int i=1; i<L.size; i++)
            L.list[i-1]=L.list[i];
    }
    else if(mark<0) item=L.list[L.size-1];  //删除表尾元素
    else { for(int i=0; i<L.size; i++)      //删除值为 item 的元素
            if(L.list[i]==item) break;
        if(i>=L.size)
            return false;
        else item=L.list[i];
        for(int j=i+1; j<L.size; j++)
            L.list[j-1]=L.list[j];
    }
    L.size--;
    return true;
}

//对线性表按升序或降序输出
void OrderOutputList(LinearList& L, int mark)
{
    int * b=new int[L.size];
    int i,k;
    for(i=0; i<L.size; i++) b[i]=i;
    for(i=1; i<L.size; i++) {
        k=i-1;
        for(int j=i; j<L.size; j++) {
            if(mark==1 && L.list[b[j]]<L.list[b[k]]) k=j;
            if(mark!=1 && L.list[b[k]]<L.list[b[j]]) k=j;
        }
        if(k!=i-1) {int x=b[i-1]; b[i-1]=b[k]; b[k]=x;}
    }
    for(i=0; i<L.size; i++)
        cout<<L.list[b[i]]<<' ';
```

```
    cout<<endl;
}

//主文件 listmain1.cpp

#include<iomanip.h>

const int ML=10;

#include"linearlist1.h"
void main()
{
    LinearList a;
    InitList(a, ML);
    int i;
    ElemType x;
  //依次向线性表 a 表尾插入 5 个整数元素
    cout<<"从键盘输入 5 个整数:";
    for(i=0; i<5; i++) {
        cin>>x;
        InsertList(a, x, -1);
    }
  //依次向线性表 a 表头插入 2 个整数元素
    cout<<"从键盘输入 2 个整数:";
    cin>>x;  InsertList(a, x, 1);
    cin>>x;  InsertList(a, x, 1);
  //按不同次序遍历输出线性表 a
    TraverList(a);
    OrderOutputList(a, 1);
    OrderOutputList(a, 0);
  //把线性表 a 中的所有元素依次有序插入到一个新线性表 b 中
    LinearList b;
    InitList(b, ML);
    for(i=0; i<a.size; i++)
        InsertList(b, a.list[i], 0);
  //输出线性表 b
    TraverList(b);
  //从线性表 a 中分别删除表头、表尾、给定值元素
    if(DeleteList(a, x, 1)) cout<<"Delete success!"<<endl;
```

```
    else cout<<"Delete fail!"<<endl;
    if(DeleteList(a, x, -1)) cout<<"Delete success!"<<endl;
    else cout<<"Delete fail!"<<endl;
    cout<<"从键盘上输入一个待删除的整数:";
    cin>>x;
    if(DeleteList(a, x, 0)) cout<<"Delete success!"<<endl;
    else cout<<"Delete fail!"<<endl;
  //输出线性表 a
    TraverList(a);
}
```

程序 2:

该程序的功能是对元素类型为 student 结构类型的顺序存储的线性表进行一些操作。该程序同样包含三个文件,第一个为头文件,用来存储 student 学生结构类型,LinearList 线性表结构类型,对线性表进行的各种操作的函数声明,对 student 类型进行等于、小于、插入、提取等操作符重载的函数声明;第二个为线性表操作的实现文件,用来存储每一种线性表操作的函数定义和操作符重载的函数定义;第三个为主文件,用来定义线性表和调用线性表的一些操作,实现对给定线性表的具体运算。整个程序如下:

```
//头文件 linearlist2.h

//定义学生记录
struct student
    char name[10];   //姓名
    short grade;     //分数
};

//定义 ElemType 为 student 类型
typedef student ElemType;

//线性表顺序存储类型
struct LinearList
{
    ElemType * list;   //存线性表元素
    int size;          //存线性表长度
    int MaxSize;       //存 list 数组长度
};

//初始化线性表
```

```
  void InitList(LinearList& L, int ms);
//清空线性表
  void ClearList(LinearList& L);
//求线性表长度
  int ListSize(LinearList& L);
//检查线性表是否为空
  bool ListEmpty(LinearList& L);
//检查线性表是否为满
  bool ListFull(LinearList& L);
//遍历线性表
  void TraverList(LinearList& L);
//从线性表中查找元素
  bool FindList(LinearList& L, ElemType& item);
//更新线性表中的给定元素
  bool UpdateList(LinearList& L, const ElemType& item);
//向线性表插入元素
  bool InsertList(LinearList& L, const ElemType& item, int mark);
//从线性表中删除元素
  bool DeleteList(LinearList& L, ElemType& item, int mark);
//对线性表进行有序输出
  void OrderOutputList(LinearList& L, int mark);
//比较两个元素是否相等
bool operator==(const ElemType& r1, const ElemType& r2);
//比较两个元素的大小
  bool operator<(const ElemType& r1, const ElemType& r2);
//输出一个元素
  ostream& operator<<(ostream& ostr, const ElemType& r);
//输入一个元素
  istream& operator>>(istream& istr, ElemType& r);

//实现文件 linearlist2.cpp

#include<iomanip.h>
#include<stdlib.h>
#include<string.h>

#include"linearlist2.h"

//初始化线性表
```

```
void InitList(LinearList& L, int ms)
{
    L.list = new ElemType[ms];
    if(!L.list) {
        cerr<<"Memory allocation failure!"<<endl;
        exit(1);
    }
    L.size = 0;
    L.MaxSize = ms;
}

//清空线性表
void ClearList(LinearList& L)
{
    L.size = 0;
}

//求线性表长度
int ListSize(LinearList& L)
{
    return L.size;
}

//检查线性表是否为空
bool ListEmpty(LinearList& L)
{
    return L.size == 0;
}

//检查线性表是否为满
bool ListFull(LinearList& L)
{
    return L.size == L.MaxSize;
}

//遍历线性表
void TraverList(LinearList& L)
{
    for(int i = 0; i < L.size; i ++ ) cout<<L.list[i]<<' ';
```

```
    cout<<endl;
}

//从线性表中查找元素
bool FindList(LinearList& L, ElemType& item)
{
        for(int i=0; i<L.size; i++)
            if(L.list[i]==item) {
                item=L.list[i];
                return true;
            }
        return false;
}

//更新线性表中的给定元素
bool UpdateList(LinearList& L, const ElemType& item)
{
    for(int i=0; i<L.size; i++)
        if(L.list[i]==item) {
            L.list[i]=item;
            return true;
        }
    return false;
}

//向线性表插入元素
bool InsertList(LinearList& L, const ElemType& item, int mark)
{
    if(ListFull(L)) return false;
    if(mark>0) {
        for(int i=L.size-1; i>=0; i--)
            L.list[i+1]=L.list[i];
        L.list[0]=item;
    }
    else if(mark<0) L.list[L.size]=item;
    else {for(int i=0; i<L.size; i++)
            if(item<L.list[i]) break;
        for(int j=L.size-1; j>=i; j--)
            L.list[j+1]=L.list[j];
```

```
            L.list[i] = item;
        }
    L.size++;
    return true;
}

//从线性表中删除元素
bool DeleteList(LinearList& L, ElemType& item, int mark)
{
    if(ListEmpty(L)) return false;
    if(mark>0) {
        item = L.list[0];
        for(int i=1; i<L.size; i++)
            L.list[i-1] = L.list[i];
    }
    else if(mark<0) item = L.list[L.size-1];
    else {for(int i=0; i<L.size; i++)
            if(L.list[i] == item) break;
        if(i>=L.size)
            return false;
        else item = L.list[i];
        for(int j=i+1; j<L.size; j++)
            L.list[j-1] = L.list[j];
    }
    L.size--;
    return true;
}

//对线性表进行有序输出
void OrderOutputList(LinearList& L, int mark)
{
    int * b=new int[L.size];
    int i,k;
    for(i=0; i<L.size; i++)  b[i]=i;
    for(i=1; i<L.size; i++) {
        k=i-1;
        for(int j=i; j<L.size; j++) {
            if(mark==1 && L.list[b[j]]<L.list[b[k]]) k=j;
            if(mark!=1 && L.list[b[k]]<L.list[b[j]]) k=j;
```

```
        }
        if(k!=i-1) {int x=b[i-1]; b[i-1]=b[k]; b[k]=x;}
    }
    for(i=0; i<L.size; i++)
        cout<<L.list[b[i]]<<' ';
    cout<<endl;
}

//比较两个元素是否相等
bool operator==(const ElemType& r1, const ElemType& r2)
{
    return strcmp(r1.name, r2.name)==0;
}

//比较两个元素的大小
bool operator<(const ElemType& r1, const ElemType& r2)
{
    return strcmp(r1.name, r2.name)==-1;
}

//输出一个元素
ostream& operator<<(ostream& ostr, const ElemType& r)
{
    ostr.setf(ios::left);
    ostr<<setw(10)<<r.name<<setw(5)<<r.grade<<"    ";
    return ostr;
}

//输入一个元素
istream& operator>>(istream& istr, ElemType& r)
{
    istr>>r.name>>r.grade;
    return istr;
}

//主文件 listmain2.cpp

#include<iomanip.h>
const int ML=10;
```