普通高等院校计算机专业（本科）实用教程系列

数据结构课程实验

徐孝凯　编著

清华大学出版社
http://www.tup.tsinghua.edu.cn

普通高等院校计算机专业(本科)实用教程系列

数据结构课程实验

徐孝凯　编著

清华大学出版社

(京)新登字158号

内 容 简 介

本书是一本学习“数据结构”课程的实验教材，可以与采用C或C++语言进行算法描述的所有“数据结构”教材配套使用。本书给出了九个实验和三个大作业，是对“数据结构”课程内容的应用和深化。在每个实验程序和大作业程序中，既涉及到面向过程的类型定义和算法描述，又涉及到面向对象的类型定义和算法描述，读者可根据需要任选。本书在内存实现数据结构和算法的同时，更注重在外存磁盘文件上的实现，使得内外存操作融为一体。读者通过对书中算法的理解、分析、补充和改进，以及上机输入、修改、调试和运行，能够加深理解和掌握课程所学的理论知识，提高在软件开发中的算法分析和设计能力。

书　　名：数据结构课程实验
作　　者：徐孝凯　编著
出 版 者：清华大学出版社(北京清华大学学研大厦，邮编 100084)
　　http://www.tup.tsinghua.edu.cn
责任编辑：郑寅堃
印 刷 者：世界知识印刷厂
发 行 者：新华书店总店北京发行所
开　　本：787×1092 1/16　印张：11.75　字数：266千字
版　　次：2002年1月第1版　2002年1月第1次印刷
书　　号：ISBN 7-302-05007-4/TP·2954
印　　数：0001～6000
定　　价：16.00元

前　言

“数据结构”课程是计算机和信息管理等相关专业的一门核心课程，具有承上起下的地位和作用，“程序设计语言”和“计算机组成原理”是它的先修课，“操作系统”、“数据库系统”、“软件工程”等是它的后续课。进行程序设计和软件开发的基础是算法设计，即如何对数据进行组织、存储和运算，并且要具有较高的效率，这正是数据结构课程要解决和研究的问题，也是学习“数据结构”课程的目的所在。

要学好“数据结构”课程仅仅通过课堂教学或自学获取理论知识是远远不够的，还必须加强实践，亲自动手上机输入、编辑、检查、修改、调试和运行已有的各种典型算法和自己编写的算法，从成功和失败的经验中得到锻炼，才能够熟练掌握和运用理论知识解决软件开发中的实际问题，达到学以致用的目的。

本书是专门为学习“数据结构”课程而编写的一本实验教材，它能够与社会上所有采用 C 或 C ++ 语言进行描述算法的数据结构教材配套使用。本书给出了九个实验和三个大作业。九个实验题目依次为：线性表操作、单链表操作、表达式计算、二叉树操作、二叉搜索树操作、图的运算、散列表操作、外存文件的排序操作、二叉搜索树与文件操作等。三个大作业课题分别为索引文件的插入、删除与查找操作，散列文件的插入、删除与查找操作，B_树索引文件的插入、删除和查找操作。每个实验程序和大作业程序都包含有典型的算法，都具有实用价值。更重要的是，本书不仅把数据结构的知识应用于内存，而且应用到外存磁盘文件的数据组织和处理上，这是所有数据结构教材尚未涉及到的，是确实有用的知识。采用本书作为“数据结构”课程的实验教材时，要求学生至少独立完成七个实验和一个大作业的上机实践任务，每个实验应安排 3 ~ 4 个机时，每个大作业应安排 6 ~ 8 个机时。若有上机时间保障，请尽量多上机做一些实验内容。

在本书的多数实验中，既给出了面向过程的程序设计算法，又给出了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，可都运行。

本书中的所有程序都在 Microsoft Visual C ++ 6.0 软件开发环境下调试运行并通过，确保所给算法和程序的正确性和有效性。

为了方便读者，本实验教材中每个程序的主要内容将放在 www. tupwg. com 网页上供读者访问。

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，敬请广大专家和读者批评指正，本人不胜感激。

本人电子邮件地址：xuxk@ crtvu. edu. cn 联系电话：010- 66052930

徐孝凯

2001 年 9 月

序　言

时光更迭、历史嬗递。中国经济带着她足以令世人惊叹的持续高速发展驶入了一个新的世纪，一个新的千年。世纪之初，以微电子、计算机、软件、通信技术为主导的信息技术革命给我们生存的社会所带来的变化令人目不暇接。软件是优化我国产业结构、加速传统产业改造和用信息化带动工业化的基础产业，是体现国家竞争力的战略性产业，是从事知识的提炼、总结、深化和应用的高智型产业；软件关系到国家的安全，是保证我国政治独立、文化不受侵蚀的重要因素；软件也是促进其他学科发展和提升的基础学科；软件作为20世纪人类文明进步的最伟大成果之一，代表了先进文化的前进方向。美国政府早在1992年“国家关键技术”一文中提出“美国在软件开发和应用上所处的传统领先地位是信息技术及其他重要领域竞争能力的一个关键因素”，“一个成熟的软件制造工业的发展是满足商业与国防对复杂程序日益增长的要求所必需的”，“在很多国家关键技术中，软件是关键的起推动作用（或阻碍作用）的因素”。在1999年1月美国总统信息技术顾问委员会的报告“21世纪的信息技术”中指出“从台式计算机、电话系统到股市，我们的经济与社会越来越依赖于软件”，“软件研究为基础研究方面最优先发展的领域”。而软件人才的缺乏和激烈竞争是当前国际的共性问题。各国、各企业都对培养、引进软件人才采取了特殊政策与措施。

为了满足社会对软件人才的需要，为了让更多的人可以更快地学到实用的软件理论、技术与方法。我们编著了《普通高等院校计算机专业（本科）实用教程系列从书》。本套丛书面向普通高等院校学生，以培养面向21世纪计算机专业应用人才（以软件工程师为主）为目标，以简明实用、便于学习、反映计算机技术最新发展和应用为特色，具体归纳为以下几点：

1. 讲透基本理论、基本原理、方法和技术，在写法上力求叙述详细，算法具体，通俗易懂，便于自学。

2. 理论结合实际。计算机是一门实践性很强的科学，丛书贯彻从实践中来到实践中去的原则，许多技术理论结合实例讲，以便于学习和理解。

3. 本丛书形成完整的体系，每本教材既有相对独立性，又有相互衔接和呼应，为总的培养目标服务。

4. 每本教材都配以习题和实验，在各教学阶段安排课程设计或大作业，培养学生的实战能力与创新精神。习题和实验可以制作成光盘。

新世纪曙光激人向上，催人奋进。江总书记在十五届五中全会上的讲话：“大力推进国民经济和社会信息化，是覆盖现代化建设全局的战略举措。以信息化带动工业化，发挥优势，实现社会生产力的跨越式发展。”指明了我国信息界前进的方向。21世纪日趋开放的国策与更加迅速发展的科技会托起祖国更加辉煌灿烂的明天。

孙家广

2001年3月

普通高等院校计算机专业(本科)实用教程系列丛书

编　委　会

目　　录

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的作用。

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。

实验要求

1. 认真阅读和掌握本实验内容所给的全部程序。

2. 从本实验内容中任选一个程序上机运行。

3. 保存和打印出程序运行结果，并结合程序进行分析。

4. 按照你对线性表操作的需要，重新改写主文件并运行，打印出主文件清单和运行的结果。

注意事项

在做第一次“数据结构”课程实验之前，要在硬盘上建立好自己的工作目录，专门用来存储你所做的实验程序及相关信息，以后每次做实验最好仍采用这个目录。

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，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   //向表尾插入元素
        if(mark <0) L.list[L.size] = item;
    else {  //有序插入元素
        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   //删除表尾元素
        if(mark <0) item =L.list[L.size -1];
    else {  //删除值为 item 的元素
        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,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;
}
```