面向21世纪课程教材
Textbook Series for 21st Century

数据结构学习指导与习题解析

刘大有　杨　博　刘亚波　姜　丽　孙成敏

高等教育出版社
HIGHER EDUCATION PRESS

面向21世纪课程教材
Textbook Series for 21st Century

数据结构
学习指导与习题解析

刘大有　杨　博　刘亚波　姜　丽　孙成敏

高等教育出版社
HIGHER EDUCATION PRESS

内容提要

数据结构课程具有难度大、学习困难和难于掌握等特点。本书配合我社出版的面向21世纪课程教材《数据结构》的使用，为读者学习数据结构课程给予指导。全书共11章，涵盖了主教材的基本内容，共汇集了134道例题和599道习题，每章（除一、二章外）包括重点内容简介、典型例题解析和习题等三部分。各章显式地、并力求准确地给出数据结构基本概念的定义，对大部分例题中的算法都分别采用ADL算法语言和C++语言进行了描述，并通过对一组相关算法的比较分析、阐明新算法产生的关键思想、设置习题的难度级别、给出较难习题的分级提示等手段达到一定的启发式教学和自学辅导的目的。

通过阅读本书，可使读者深入理解数据结构概念，熟练掌握基于数据结构设计和分析算法的技巧，提高理论结合实际的能力。

本书还可与作者承担的教育部新世纪网络课程“数据结构”配合使用。可作为高等院校计算机专业学生的学习辅导书，也可作为研究生入学考试的复习参考书，还可供计算机专业人员参考阅读。

图书在版编目（CIP）数据

数据结构学习指导与习题解析/刘大有等.—北京：
高等教育出版社，2004.6
ISBN 7-04-014617-7

Ⅰ.数… Ⅱ.刘… Ⅲ.数据结构-高等学校-
自学参考资料 Ⅳ.TP311.12

中国版本图书馆CIP数据核字（2004）第029688号

策划编辑 倪文慧 **责任编辑** 倪文慧 **封面设计** 张 楠
版式设计 马静如 **责任校对** 康晓燕 **责任印制** 宋克学

出版发行	高等教育出版社	购书热线	010-64054588
社 址	北京市西城区德外大街4号	免费咨询	800-810-0598
邮政编码	100011	网 址	http://www.hep.edu.cn
总 机	010-82028899		http://www.hep.com.cn
经 销	新华书店北京发行所		
印 刷	北京中科印刷有限公司		
开 本	787×960 1/16	版 次	2004年6月第1版
印 张	17.75	印 次	2004年6月第1次印刷
字 数	330 000	定 价	22.40元

本书如有缺页、倒页、脱页等质量问题，请到所购图书销售部门联系调换。

前　言

为计算机编写程序的过程是饶有趣味的，因为它不但具有科学价值与经济意义，而且也是犹如赋诗或作曲那样的美学实践。图灵奖获得者、杰出的计算机科学家 D. E. knuth（唐纳德·克努特）在他的传世之作《计算机程序设计的艺术》的前言中，阐明了这样的观点。著名的计算机科学家 N. Wirth（尼克劳斯·沃思）指出："算法 + 数据结构 = 程序"。实际上，数据结构与算法是一对孪生兄弟，数据结构课程充分体现了数据结构与算法的有机结合。因此，如果我们将程序设计比作计算机软件与理论领域中的一棵大树，那么将数据结构看作大树的躯干无疑是合情入理的。

同时，数据结构课程是计算机（科学与技术）专业的一门骨干课程，也是操作系统和编译原理等计算机专业核心课程的基础，不透彻掌握数据结构知识，就很难成为一名高水平的软件工程师和计算机科学技术工作者。然而数据结构课程具有难度大、学习困难和难于掌握等特点，因此，配合《数据结构》教材编写《数据结构学习指导与习题解析》就显得十分必要，本书正是在这一背景下展开撰写工作的。

鉴于 OOP 的先进性和 C ++ 语言使用的广泛性，我们对本书的算法用 C ++ 语言进行了描述；另一方面，ADL 是一种继承了 D. E. Knuth 算法语言风格，很容易掌握又便于递归算法设计的算法语言，它不但避免了 C ++ 语言过多涉及数据类型的定义，而且能在较高层面上清晰地勾画出算法的轮廓，因此对大多数算法同时又给出了 ADL 语言描述。

全书共 11 章，第一章和第二章分别对 C ++ 和算法分析作了简单介绍；第三章至第十一章阐明了线性表、堆栈与队列，数组与字符串，树与二叉树，图，递归，排序，查找，复杂数据结构和应用等重点内容。第三章至第十一章具体包括：线性表、堆栈与队列的逻辑结构、顺序存储和链接存储结构；数组和字符串的基本概念、存储结构及主要操作；树、二叉树、线索二叉树、森林和哈夫曼树的基本概念和性质，树、二叉树的顺序及链接存储结构，树、二叉树的遍历，森林与二叉树的转换和哈夫曼算法；图的基本概念、存储结构及主要操作；递归概念，递归问题求解方法及递归与非递归转换方法；排序的基本概念，插入、交换、快速、选择、合并和分布等内排序方法，磁带和磁盘等外排序方法；查找的基本概念，线性表、二叉树和杂凑等查找方法；优先级队列、不相交集合类的概念和应用；事件驱动

模拟、在线等价类、残缺棋盘和图像压缩等应用实例。此外，在第三章至第十一章中，还包括了我们的如下工作：显式地、力求更准确地（或形式化地）给出重要概念的定义（并与概念例题和概念习题相配合），期望改善相当数量的学生和一些读者轻视对概念的深入理解和对概念掌握比较模糊的状况；用定理、引理和推论阐明了重要结论，向读者强调正确性和严格性对算法是非常重要的；对相关算法做出比较，尽可能给出产生新算法（两个相关的算法，一个相对于另一个为更新，这个为更新的算法被称为新算法）的关键思想，以培养读者的创新能力；提出了典型例题的解题思路和答案；提供了比较丰富的习题，对每一道习题都标明了难度级别（共分5级。难度级别用放在习题编号后面的方括号内的自然数表示，5、4、3、2、1分别表示第5级、第4级、第3级、第2级、第1级。第5级〔很难〕，完成解答大约平均需要花费数小时以上；第4级〔难〕，完成解答大约平均需要花费1至2小时；第3级〔较难〕，完成解答大约平均需要花费20分钟至40分钟；第2级〔比较简单〕，完成解答大约平均需要花费10分钟；第1级〔简单〕，完成解答大约平均需要花费5分钟），同时对概念题和级别为较难、难和很难的习题分别给出了答案和分级提示（第$i+1$级提示是相对于第i级提示的进一步提示，比如，某习题共有3个提示，第1级提示指第1个提示，第2级提示指第1个提示加上第2个提示，第3级提示指3个提示的总和）。通过对一组相关算法的比较分析，阐明新算法产生的关键思想，设置习题的难度级别，给出难度级别大于等于较难的习题的分级提示来达到一定的启发式教学和因材施教的目的。

本书包括134道例题、599道习题。所选例题、习题的概念性强，覆盖面广，并具有典型性和代表性。学习数据结构课程的读者，通过阅读本书的重点知识、例题，解答本书的习题，可达到深入理解数据结构概念，熟练掌握数据结构原理和知识以及基于数据结构设计和分析算法的技巧，提高将所学数据结构理论知识与实际问题相结合的能力。

参加本书撰写工作的人员主要包括：刘大有、杨博、刘亚波、姜丽和孙成敏。其中第二章由刘大有执笔，第三章、第五章和第十章由姜丽执笔，第一章、第四章、第六章和第七章由刘亚波执笔，第八章、第九章和第十一章由孙成敏执笔，刘大有和杨博对各章进行了仔细修改，并对全书进行了统稿。

由于编者水平和撰写时间所限，书中难免存在差错和不足，敬请广大读者批评指正。本书责任编辑的电子邮件地址为 niwh@hep.com.cn，衷心希望与各位读者进行交流。

编 者

2004年3月

目　录

第一章 C++语言概述

本书使用面向对象程序设计语言C++描述各种数据结构和相关算法。本章主要介绍C++的一些基本概念,包括:指针和动态存储、数组、引用、参数传递、类、多态性、友元函数、模板和继承。

1.1 指针和动态存储

1.1.1 指针

指针是C++语言的一个重要特性。在介绍指针之前,首先来看以下三个概念。

变量的名称 是标识变量在程序中身份的一个符号标签。

变量的地址 是变量所占存储单元的第一个字节地址。

变量的值 是指变量所占存储单元中的内容。

所谓**指针**是一个变量,它的值或者为0,或者为另一变量或对象的内存地址。如果它的值是0,则称为空指针(NULL);否则,称该指针指向一个变量或者对象,它存储了此变量或对象的地址。指针的声明格式为:

数据类型 * 变量;

C++语言中,如果 x 是一个T类型变量或者对象的名称,则T * px = &x表示px的值为 x 在内存中的地址,因此px是指向 x 的一个指针。例如:

```
int n = 44;                    // 将名字为 n 的整型变量初始化为 44,
string s = "hello";            // 将名字为 s 的字符串变量初始化为"hello"
int * pn = &n;                 // pn 是指向 n 的指针
string * ps = &s;              // ps 是指向 s 的指针
```

例1.1 指针。

```
#include < iostream. h >
int main( )
{
    int n = 44;
```

```
    string s = "hello";
    cout << "&n = " << &n << "\n";
    cout << "&s = " << &s << "\n";
    int *p;
    int *q = 0;              // 初始化 q 是一个空指针
    int *pn = &n;            // 初始化 pn 是指向 n 的指针
    string *ps = &s;         // 初始化 ps 是指向 s 的指针
    cout << "p = " << p << "\n";
    cout << "q = " << q << "\n";
    cout << "pn = " << pn << "\n";
    cout << "ps = " << ps << "\n";
}
```

程序运行结果为：

```
&n = 0x18ff24b8
&s = 0x18ff24bc
p = 0x2007e9a4
q = 0x00000000
pn = 0x18ff24b8
ps = 0x18ff24bc
```

说明：例 1.1 第一条语句中的 #include 是预处理指令，语句 #include <iostream.h> 将尖括号中的库文件作为头文件包含到程序中。在库文件 iostream.h 中，有两个预定义的类 istream 和 ostream，它们分别定义了输入流类和输出流类。输入语句的形式是：在“>>”前面给出输入流对象 cin，后面给出输入对象，对于一系列输入对象，可用“>>”分开；输出语句的形式是：在“<<”前面给出输出流对象 cout，后面给出输出对象，对于一系列输出对象，可用“<<”分开；输出语句最后的“\n”是 C ++ 的 I/O 操作符，其作用是执行换行和对流进行清空操作。

1.1.2　动态存储分配

C ++ 用指令 new 按指定类型自动分配该类型需要的足够空间，并自动返回指向该类型的指针。如果没有足够的存储空间可以分配，则返回空指针(NULL)。用 new 分配的存储空间必须用 delete 释放。使用 new 和 delete 操作符的方法是：

```
T *p;              // 定义 p 为指向 T 类型的指针
p = new T;         // p 为类型 T 数据的内存地址
...
```

```
delete p ;          // 释放由 p 指向的内存空间
```

例如：

```
int * point1 = new int;// 分配一个整型数据所需的空间,并令指针 point1 指向它
if( point1 = = NULL)   cerr << " Memory not allocated ! " << endl;
...
delete point1 ;      // 释放指针 point1 所指向的存储空间
```

若释放用 new 分配的数组空间，则必须令 delete 知道被释放的是数组。否则，将只释放第一个数组元素所占的空间，而无法再对该数组其他元素所占的空间进行使用。

```
int * point2 = new int[10];// 指针 point2 指向新分配的数组空间的首地址
...
delete[ ]point2;            // 释放 point2 所指数组的占用空间
```

例 1.2 使用 new 与 delete 操作符。

```
int main( )
{
    string * p = new string( " Goodbye" );
    cout << " p = " << p << " \n";
    cout << " * p = " << * p << " \n";
    delete p;
    cout << " p = " << p << " \n";
    cout << " * p = " << * p << " \n";
}
```

程序运行结果为：

```
p = 0x004307a0
* p = Goodbye
p = 0x004307a0
* p =
```

1.1.3 数组

数组是存储在连续存储空间内的具有相同数据类型的一组数据。在 C ++ 语言中，一个数组的名字实际上是一个指向其第一个元素的指针，因此，可以使用数组名和下标来访问数组元素，并且数组第一个元素的下标为 0。例如，a 是数组的名字，则可以用 a[0]、a[1]、a[2]、a[3]等来访问其中的元素。数组的声明格式为：

数据类型 数组名称[长度]

例如：

```
int intarray[100];      // 声明一个含有 100 个元素的整型数组
char chararray[20];     // 声明一个含有 20 个元素的字符数组
```

例 1.3　字符串数组。

```
int main( )
{   string a[4];
    a[0] = "One";
    a[1] = "Two";
    a[2] = "Three";
    a[3] = "Four";
    for(int i = 0;i < 4;i ++ )
        cout << "a[" << i << "] = " << a[i] << "\n";
}
```

程序运行结果为：

```
a[0] = One
a[1] = Two
a[2] = Three
a[3] = Four
```

数组的初始化也可以用一个初始化列表完成。例如：a = {"One","Two","Three","Four"}。

例 1.3 中声明的叫做静态数组，数组的大小是一个整型常量，在编译时分配存储空间。使用 new 操作符可以创建一个动态数组，它的大小可以是一个整型变量，在运行时分配存储空间。动态数组的名字是一个指针，其声明格式如下：

```
T *a = new T[n];      // T 是元素的类型，n 是数组的大小，它也可以是一个整型
                         表达式
```

例 1.4　使用动态数组。

```
int main
{   int n;
    cout << "How many city do you know?";
    cin >> n;
    string *city = new string[n];
    cout << "Please give me the names of the " << n << "cities:\n";
    for(int i = 0;i < n;i ++ )
    {  cout << "\t" << i + 1 << ":";
       cin >> city[i];
    }
    cout << "They are" << child[0];
    for(int i = 1;i < n;i ++ )
```

```
    { cout << " , " << city[i];}
    cout << " \n" ;
}
```

程序运行结果为：

```
How many city do you know? 4
Please give me the name of the 4 cities:
        1:beijing
        2:shanghai
        3:dalian
        4:changchun
They are beijing,shanghai,dalian,changchun
```

1.2 引 用

引用是一个已经存在的变量或者对象的别名。引用的定义看起来非常像变量的定义。但是，引用不是变量。例如，代码序列：

```
int x = 100;
int &y = x;
```

int &y 表示 y 是对 int 类型变量 x 的一个引用。在 C ++ 语言中，所有的引用都必须被初始化。

例 1.5 使用引用。

```
int main( )
{
    int n = 44;
    int &m = n;          // 声明 m 是 n 的引用
    cout << " n = " << n << " \n" ;
    cout << " m = " << m << " \n" ;
    n * = 2;             // n 和 m 都乘以 2
    cout << " n = " << n << " \n" ;
    cout << " m = " << m << " \n" ;
    m/ = 2;              // n 和 m 都除以 2
    cout << " n = " << n << " \n" ;
    cout << " m = " << m << " \n" ;
}
```

程序运行结果为：

```
n = 44
```

```
m = 44
n = 88
m = 88
n = 44
m = 44
```

这个程序在第一行声明 n 为 int 类型的变量,第二行声明 n 的一个引用 m,说明存在 n 的另外一个名字。因此,当 n 被乘以 2 时,m 也被乘以 2;同样,当 m 被除以 2 时,n 也被除以 2。

1.3 参数传递

在调用函数时,实参必须与形参在类型、个数和顺序上都保持一致。参数传递有如下几种方式。

1.3.1 值传递

值传递是缺省的参数传递方式。它的过程是把实参的值传送给函数局部工作区相应的副本,函数使用这个副本执行必要的操作。如此一来,函数修改的是副本的值,而实参的值没有被改变。

例 1.6 值传递。

```
void swap (int a,int b)
{
    int c = a;
    a = b;
    b = c;
}
main ( )
{
    int x = 1 ,y = 2;
    cout << " x = " << x << " y = " << y << endl ;
    swap (x,y) ;
    cout << " x = " << x << " y = " << y << endl ;
}
```

程序运行结果为:

```
x = 1   y = 2
x = 1   y = 2
```

因为 swap 的函数定义采用的是值调用方式，所以调用 swap 函数没有达到交换 x 和 y 的值的目的。

1.3.2 按引用调用

如果形参声明的参数名前加一个“&”，则该参数的调用方式是按引用调用，形参是引用类型。当一个实参与一个引用型形参相结合时，被传递的不是实参的值，而是实参的地址，函数通过地址存取被引用的实参。因此，如果在函数体中对形参进行修改，那么调用函数时实参亦相应被修改。将例 1.6 的 swap 函数略做改动，主程序不变，则得到例 1.7。

例 1.7 按引用调用。

```
void swap ( int &a, int &b )
{
    int c = a ;
    a = b ;
    b = c ;
}
void main ( )
{
    ...     // 主程序同例 1.6
}
```

程序输出为：

```
x = 1   y = 2
x = 2   y = 1
```

值传递比引用传递更加安全，因为它防止了实参的意外改变，而且不用复制实参。但是它的缺点是必须要辅之实参。当把一个对象传递给一个不改变它的函数的时候，可以使用常值引用。

1.3.3 常值调用

这是一种特殊的引用调用方式，其格式为 const. Type & a，Type 为参数的数据类型，a 为常值参数，这种方式要求在函数体中不能修改常值参数，否则将导致编译错误。

1.3.4 数组参数传递

在函数定义中，若形参为数组，函数调用是通过传递数组名作为实参的，参数前可以不用加“&”，例如：max(a,100)，这样表面上是值调用，实际上采用的

是按引用调用，这是因为传递的是数组第一个元素的地址，从而对形参的修改必导致对实参的修改。另外，在参数表中声明数组的形式是 int a[ ]，因此还必须声明数组的大小。

```
void max(double a[ ],int size);
void max(double *a,int size);
```

这两种形式是等价的，因为一个数组的名字实际上是指向其第一个元素的指针。

例 1.8　找数组中的最大元素。

```
int main( )
{ int a[ ] = {44,77,33,66,55,88,22};
  cout << "max(a,7) = " << max(a,7) << "\n";
}
int max(int *a,int n)
{ int m = a[0];
  for(int i = 1;i < n;i ++ )
    if(a[i] > m) m = a[i];
  return m;
}
```

程序运行结果为：

```
max(a,7) = 88
```

1.4　类

类是一种用户自定义的数据类型，完整的类定义包括类声明和类实现。

1.4.1　类声明

用 C++语言对类进行声明，其一般形式为：

```
class classname
{
    private:
      私有数据成员
      私有函数成员
    public:
      公有数据成员
      公有函数成员
```