21世纪普通高等教育精品教材

计算机系列

数据结构

主编 孙 涛 黄泳松

WUHAN UNIVERSITY PRESS
武汉大学出版社

21世纪普通高等教育精品教材

计算机系列

数据结构



主　编　孙　涛　黄泳送

副主编　陈忠伟　杨勋华　蓝振师

编　委　陆　涛　彭小英　韦海清

黄　翠

WUHAN UNIVERSITY PRESS

武汉大学出版社

图书在版编目(CIP)数据

数据结构/孙涛主编．—武汉：武汉大学出版社，2014.7
21世纪普通高等教育精品教材.计算机系列
ISBN 978-7-307-13788-2
Ⅰ.①数… Ⅱ.①孙… Ⅲ.数据结构—高等学校—教材 Ⅳ.TP311.12

中国版本图书馆CIP数据核字(2014)第154607号

责任编辑：刘俊杰

出版发行：**武汉大学出版社** (430072 武昌 珞珈山)
(电子邮件：cbs22@whu.edu.cn 网址：www.wdp.com.cn)
印刷：北京泽宇印刷有限公司
开本：787×1092 1/16 印张：16 字数：333千字
版次：2014年7月第1版 2014年7月第1次印刷
ISBN 978-7-307-13788-2 定价：35.00元

内 容 简 介

本书根据本科层次的教学大纲，从方便读者理解的角度出发，使读者能够快速掌握数据结构的基本内容，为后续课程的学习打下良好的基础。

本书分为11章，主要介绍了绪论、线性表、堆栈和队列、串、数组和广义表、树和二叉树、图、检索、排序，以及高级数据结构内容——索引技术，第11章为数据结构案例，用以帮助读者更好地理解本书内容。其中，第1～7章从抽象数据类型的角度讨论了基本的数据结构及应用，第8～10章讨论了检索、排序和索引的实现及分析。

本书采用C语言作为各类程序和算法的描述语言。语言精练，表述通俗易懂，推理严谨，可作为计算机类相关专业或信息类相关专业的普通本科或专科院校教材，也可供从事计算机相关领域的工作人员参考。

前　言

“数据结构”是计算机学科中的一门综合性的专业基础课，主要研究数据的逻辑结构、数据在计算机中的存储结构和对数据进行的各种运算的方法和算法。数据的逻辑结构分为集合、线性、树和网络四种基本结构，它们可以构成任何较为复杂的逻辑结构。其中数据的存储结构可分为顺序、链接、索引、散列等四种基本结构，对数据的运算包括查找、排序、插入、删除、修改、遍历等，同样的数据，采用的逻辑结构和物理存储结构不同，则相对于的运算所采用的方法也不同，得到的算法也不同。这门课程的内容不仅是一般程序设计(特别是非数值性程序设计)的基础，而且是设计和实现编译程序、操作系统、数据库系统及其他系统程序的重要基础。学好该课程，不仅对这些后续课程的学习有很大帮助，而且在实际中也有广泛的用途。

全书由11个章节组成，各部分内容主要有：第1章介绍了数据结构和算法的一些基本概念；第2章是线性表，介绍了线性表的两种存储结构和一些基本操作的实现；第3章堆栈和队列，介绍了这两种线性结构的概念和相关操作；第4章串，介绍了串的概念和模式匹配的问题；第5章数组和广义表，介绍了数组和广义表的定义及使用；第6章树和二叉树，介绍了树和二叉树的概念与各种运算的实现；第7章图，介绍了图的概念和图的各种算法实现及应用；第8、9、10章分别介绍了检索、排序和文件索引常用的各类方法。为了在学完本书后读者能更好地掌握相关内容，编者增加了第11章数据结构案例，用以加深读者的理解。此外，在每一章后面还配有大量的习题供读者练习。

鉴于大部分数据结构教材中的对算法、函数的解释和说明都是伪代码，初学者理解

起来较为困难，在本书中，编者给出的算法和程序都是在C语言运行环境下调试通过的，有助于读者理解。

由于作者水平有限，不足之处在所难免，敬请广大读者朋友批评指正。

编　者

目　录

第1章　绪　论

1.1　概述 …… 1
1.2　数据结构和算法 …… 12
1.3　抽象数据类型的表示和实现 …… 14
1.4　算法的描述和分析 …… 14
本章小结 …… 15
本章习题 …… 15

第2章　线性表

2.1　线性表的定义 …… 17
2.2　线性表的顺序表示和实现 …… 17
2.3　线性表的链式表示和实现 …… 19
2.4　链表的深入学习 …… 31
本章小结 …… 35
本章习题 …… 35

第3章　堆栈和队列

3.1　堆栈 …… 37
3.2　堆栈的应用 …… 40
3.3　队列 …… 52
3.4　队列的应用 …… 54
本章小结 …… 60
本章习题 …… 60

第4章　串

4.1　串的定义 …… 62
4.2　串的表示和实现 …… 63
4.3　串的运算 …… 66
4.4　串的模式匹配算法 …… 71
4.5　汉字串 …… 75
本章小结 …… 77
本章习题 …… 78

第5章　数组和广义表

5.1　数组的定义 …… 79
5.2　数组的顺序表示和实现 …… 80
5.3　矩阵的压缩存储 …… 81
5.4　广义表 …… 86
本章小结 …… 88
本章习题 …… 88

第6章　树和二叉树

6.1　树的基本概念和术语 …… 89
6.2　树的存储结构 …… 91
6.3　二叉树 …… 91
6.4　遍历二叉树 …… 94
6.5　线索二叉树 …… 96
6.6　二叉排序树 …… 99
6.7　堆 …… 107
6.8　哈夫曼树 …… 111
6.9　二叉树的深入学习 …… 120
本章小结 …… 124

本章习题 …… 125

第7章 图

7.1 图的基本概念和术语 …… 128
7.2 图的存储结构 …… 131
7.3 图的遍历 …… 135
7.4 生成树 …… 138
7.5 最短路径 …… 141
7.6 拓扑排序 …… 145
本章小结 …… 149
本章习题 …… 150

第8章 检索

8.1 顺序检索 …… 152
8.2 对半检索 …… 153
8.3 分块检索 …… 161
8.4 哈希检索 …… 162
本章小结 …… 173
本章习题 …… 173

第9章 排序

9.1 排序的概念 …… 174
9.2 交换排序 …… 175
9.3 Shell 排序 …… 180
9.4 快速排序 …… 181
9.5 堆排序 …… 183
9.6 归并排序 …… 185
本章小结 …… 188
本章习题 …… 188

第 10 章　高级数据结构内容——索引技术

10.1　基本概述 …… 190
10.2　线性索引 …… 191
10.3　2-3 树 …… 193
10.4　B＋树 …… 201
本章小结 …… 205
本章习题 …… 205

第 11 章　数据结构案例

11.1　停车场管理 …… 207
11.2　家族关系查询系统 …… 214
11.3　地铁建设问题 …… 230
11.4　教学计划的安排 …… 233
11.5　校园导航系统 …… 236
11.6　电文的编码与译码 …… 241

参考文献

第1章 绪 论

本章导读

* 理解相关概念：数据、数据对象、数据元素、数据结构、数据类型、数据抽象等；
* 理解算法的定义和特性；
* 掌握算法的复杂度计算。

1.1 概述

1.1.1 数据结构的应用对象

计算机应用可以分为两大类，一类是科学计算和工业控制，另一类是商业数据处理。相应的计算机语言也是如此，比如 Fortran 语言、C、汇编语言主要适应于前者，比如 Java、Powerbuilder(关系数据库平台开发工具)、Visual C 等主要适应于后者。面向工业控制与科学计算的内容主要涉及它的计算方法、效率与速度等因素，某一特定的测控对象有特定的算法，在这里我们主要侧重于解决问题的方法研究，比如高次方程的迭代算法，快速富氏变换的蝶形算法等。面向商业管理是要解决海量数据的管理与关联分析，即使是一个特定的对象也有通用的数据管理形式，比如商业数据库系统，无论何种具体应用，它都是大量的表格一类的数据处理形式，在海量数据中检索与查询是一类至关重要的操作工具，于是，数据的逻辑结构与物理组织形式是我们要解决的主要问题，比如表数据的存储形式，索引结构等，也就是数据结构问题。

什么是数据结构？数据结构的研究对象是数据元素，目的是建立数据元素在计算机中的表达方法，简单地说，在一群有限的数据元素集合里，元素与元素之间相互关系的描述，称为它的数据结构。比如，例 1-1 描述了有限个数据元素集合的字典的数据结构关系。

【例 1-1】 字典的数据结构。

D={(able,能干的),(apple,苹果),(bug,虫),(code,代码),(cool,酷),…,(x－ray,X光),(year,年),(zoo,动物园)}

这里，单词是数据元素检索关键字，单词与注释构成数据元素(节点)，元素节点之间所表达的关系是按字母的顺序排列，这就是我们给字典这一特定对象选定的数据结构。另一个例子 1.2 描述了事务处理中经常见到表格的数据结构形式。

【例 1-2】 线性表数据结构。

表 1-1　设备统计清单

序号	设备名称	型号	单价(元)	数量
1	车床	A64	5500	5
2	台钻	C7	3200	29
3	铣床	X－2	4000	14
4	铣床	X－34	6700	1

如表 1-1 所示设备统计表是一种线性结构，为了把一个线性表转换成可以用计算机处理的形式，或者说选择表在计算机中的数据结构形式，需要采取如下步骤：首先，水平方向看表的每一行是一条记录，我们称之为向量 a_i，a_i＝(序号，设备名称，型号，单价，数量)，a_i 的各分量是设备这一客观实体的属性，属性的取值就是实体记录，所以，从纵向看，表是成由一组记录所组成的，记录是表的数据结构元素，定义如下：

```
struct BILL{
        char Facility[20];
        char Type[10];
        int Cost;
        int Number;
        };
```

表结构表达的记录(节点元素)之间的关系是＜a_i，a_{i+1}＞，所以我们称表结构是线性的，可以用 C 语言的数组变量定义相应的数据关系为：

```
struct BILL a[4];
```

同所有的数组变量一样，结构数组的下标也是从 0 开始的。因此，在计算机中可以用 BILL 结构变量型数组 A[]来描述表 1-1 所表达的关系，也就是线性表的数据结构形式：

a_0＝(1，车床，A64，5500，5)

a_1＝(2，台钻，C7，3200，29)

a_2＝(3，铣床，X－2，4000，14)

a_3＝(4，铣，X－34，6700，1)

1.1.2　学习数据结构的基础

数据结构建立在计算机语言之上。学习计算机语言是学习编程方法，我们应该如何用一种具体的计算机语言实现一个算法。学习数据结构，是学习如何描述一个应用对象的数据元素(属性构成)，如何根据应用对象的特点构造数据元素之间的逻辑关系以及内存中的存储实现，这是二者的区别。

设计数据结构的时候要有相应的计算机语言工具支持，在 Basic、Fortran、C 语言中，只有 C 是面向数据结构应用的工具语言。比较一下 C 和其他语言的区别就可以知道原因，因为它有定义数据结构基本单元的能力，并有地址的运算能力，这两点是非常重要的。通过定

义数据结构的基本单元，我们可以把不同数据类型的变量聚集在一个节点内；通过地址运算，我们可以把数据结构的逻辑关系在计算机内存中用不同存储方式实现。在C语言中定义数据结构元素是通过结构体实现的。

在学习C语言时，同学对数组很熟悉，比如一个整型量的数组定义如下：

```
int array[100];
```

它表达了一组整型量的集合，在C语言中基本变量的类型有整型量，浮点变量，字符变量等，将所有基本变量聚合在一起的方法是定义结构体，用结构体作为基本元素描述事物的属性信息，比如表1-1那样，我们称之为数据结构元素，或者节点。关于数据结构元素在C语言中给出了明确定义：

结构元素是一种被命名为一个标识符的各种变量的集合，是提供将各种基本数据类型汇集到一块的手段，它提供了结构变量的格式。

比如一个电话簿的结构元素如下定义：

```
struct ADDER{
        char Name[20];
        char Street[40];
        char City[20];
        char STATE[2];
        unsigned long Zip;
        };
```

通过结构体定义，ADDER结构变量代表了一组基本数据类型的聚合结构，它就是所谓数据结构的基本单元，我们定义，数据结构就是描述这样一组结构变量之间关系的形式，例如：

```
struct ADDER adder_info[100];
```

给出了结构变量ADDER的数组结合形式，是一种线性关系数据结构。

结构化的程序模块和指针的应用是C语言程序设计的基本风格，随着BC和VC的出现，面向对象的程序设计方法以及多线程技术给我们提供了在Windows平台上开发应用软件的多样化风格，但是，指针的应用依然是我们程序设计最基本的特征。

指针是地址，它指向数据变量在内存中的地址，请同学牢牢记住这个概念，我们之所以对指针的理解非常容易混淆，是因为我们没有把指针的概念与变量的存储位置关联在一起进行考虑。请同学清楚下面几点：

- 指针的值是地址；
- 任何一个变量都有一个地址，变量类型不同占用的地址单元数量不同；
- 指针也是一个变量，所以它也有地址；
- 给指针赋值是让指针变量指向与其同类的一个数据变量的地址；
- 没有赋值的空指针其指向不确定，所以绝对不能在程序中使用。

程序中定义任何一个名称的变量都对应着一个物理地址，因为需要保存变量值在内存

该地址单元内，比如：

```
char name[20];          //编译程序分配地址单元
scanf("%s",name);  //给变量 name 赋值
```

一个变量在内存占用的地址单元多少由变量的类型决定，比如，字符型变量占用一个字节，整型量占用 2 个字节，浮点型占用 4 个字节等，而一个结构元素占用的内存字节数由它所聚集的基本变量类型及数量决定。

指针是程序中定义的，因而指针也是一个变量，为了区别数据与地址的关系，我们将元素变量称之为数据变量，称指针为地址变量，所以指针也需要保存，指针本身也有地址：

```
char *cp,name[20];  //编译程序给指针变量 cp 分配地址单元
cp=name;                  //给指针变量 cp 赋值，让它指向数据变量 name
```

1. C 程序中指针的用法

指针变量的基本概念是地址，它用地址运算符取得某一数据变量在内存的地址，从而指向了该变量。指针存储着一个数据变量的地址，既然不同类型的数据变量占用的存储单元数不同，所以，指针变量的类型应该与其所指向的数据变量同类，也就是有整型量指针，字符型指针，浮点数指针和结构体指针，这样，在变量集合内，指针加一操作时所跨过的地址单元数，是该类数据变量占用的内存单元长度，从而能正确地指向下一个变量位置。指针如下操作得以指向一个数据变量：

```
int val=10,y,*p;
p=&val;
y=*p;
*p=20;
```

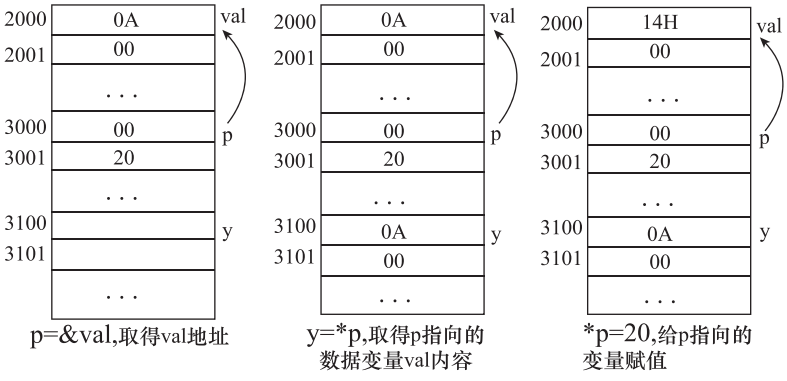


图 1-1　指针应用—指针 p 指向变量，操作指针等于操作变量

我们首先定义了整型数据变量 val、y 和整型指针变量 p，第二条语句让指针 p 取得了 val 的地址，即指针 p 指向了变量 val，第三条语句将指针所指向的变量 val 的值赋给了变量 y，第四条语句将指针 p 指向的变量 val 的值修改为 20，见图 1-1。于是，对指针的操作等价于对变量的操作，程序变得非常简洁，比如下面程序对数组进行线性赋值：

```
int array[100], * p,i;
p=&array[0];
for(i=0;i<100;i++) *(p+i)=i;
```

切记，一定不能给一个没有值的指针，也就是空指针赋值，也不能给指针任意赋一个值，比如零，图 1-2 显示了给一个空指针置数的结果，一般 0000H 是计算机操作系统保留区域，比如是软中断引导程序的入口地址，假设你给指针指向的地址单元赋值，那就是说你破坏了系统程序入口地址，如果编译系统没有检查功能，你的程序运行时候将破坏整个计算机系统运行状态。如果指针的值是任意一个随机数，它可能指向任何可能的应用程序正在使用的数据区或者栈区域，你的赋值操作就破坏了该应用程序，比如说它的返回地址。

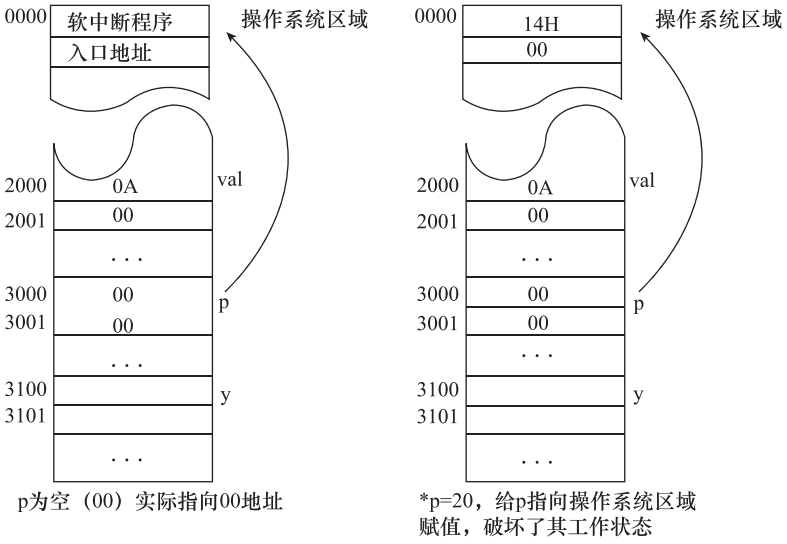


图 1-2 对空指针赋值

指针的另一种用法是地址的传递，数据结构中经常将一个指针的值（某一节点元素的地址）传递给另一个指针，比如，图 1-3 表示了如下程序段的执行结果。

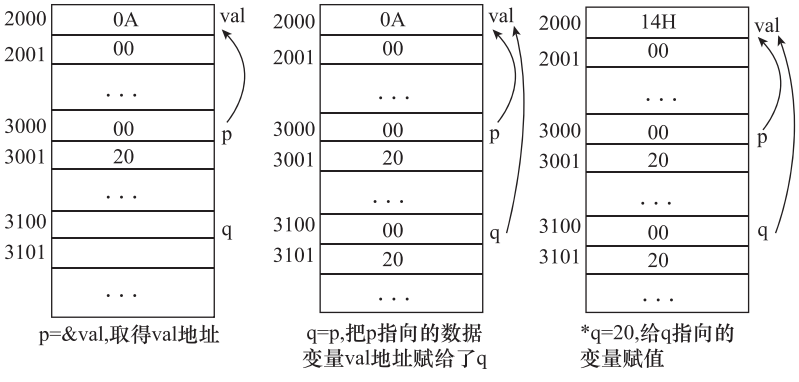


图 1-3 地址传递

```
int val=10, *p, *q;
p=&val;
q=p;
*q=20;
```

另外，为数据节点申请内存空间时，用指针指向调用函数返回的节点地址：

```
p=(struct node *)malloc(sizeof(node));//p 是指针变量
```

2. 指针在数据结构中的关联作用

指针在数据结构起到关联节点的作用，让指针从一个节点元素指向另一个节点元素，换句话说，通过指针连接节点元素之间的存储位置，从而让它们之间关联在一起，进而表达了它们之间的逻辑关系。

让指针从一个节点指向另一(或者是多个)节点，需要在节点定义中加入指针变量，指针在节点内，它指向下一个节点，如果你能找到当前节点位置，就能根据指针找到后续节点所在，这就是节点关联。现在讨论如何用指针关联两个节点元素，我们看例 1-3。

【例 1-3】 用节点内部指针关联两个节点。

如下一个学生数据节点的定义：

```
struct student_node{
                int number;
                char name[40];
                char gender;
                struct node *student_next;
                };
```

在这个结构体内，不但提供了描述学生个体属性的基本变量聚合，而且还有该节点类型的指针变量 next，用 next 可以指向学生集合中的其他个体或者说是节点，从而表达了集合中节点之间的关系，使它们关联在一起。比如，设指针 head 已指向内存里的一个节点 a1，当再申请一个节点比如 a2 时，通过对 a1 的 next 赋值使其指向 a2，从而让 a1 与 a2 关联起来，如图 1-4 所示。方法实例如程序 1.1。

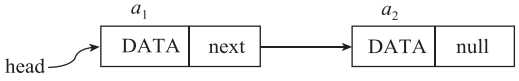


图 1-4 节点关联

【程序 1.1】

```
#include<stdio.h>
#include<malloc.h>
#define null 0
int main(void)
{
```

```
struct student_node{
                    char number[20];
                    char name[40];
                    char gender;
                    struct student_node *next;
                    } *q, *head;
    q=(struct student_node *)malloc(sizeof(student_node));//节点 a1
    head=q;                          //head 指向 a1
    q=(struct student_node *)malloc(sizeof(student_node));//节点 a2
    printf("请输入名字\n");
    scanf("%s",q->name);             //给 name 赋值,输入名字
    printf("请输入学号\n");
    scanf("%s",q->number);           //给 number 赋值,输入学号
    q->next=null;
    head->next=q;                    //给 a1 的指针赋值,a1 的指针指向 a2
    printf("节点 a2 记录内容是:");
    printf("%s   %s\n",head->next->number,head->next->name);//打印
节点 a2 输入的名字
    return(0);
}
```

程序运行结果：

```
请输入名字
张三
请输入学号
2003w1234
节点 a2 记录内容是:2003w1234                张三
```

在链表和树结构中往往要求内部节点与外部(比如树权是内部节点而叶子是外部节点)同构,处理的方法是C语言中的共用体(union)数据类型,这里简要地回顾共用体的概念。

3. 共用体说明和共用体变量定义

共用体是一种数据类型,它是一种特殊形式的变量。共用体说明和共用体变量定义与结构十分相似。其形式为：

```
union 共用体名{
        数据类型 成员名;
        数据类型 成员名;
        …
} 共用体变量名;
```

共用体表示几个变量共用一个内存位置，在不同的时间保存不同的数据类型和不同长度的变量。下例表示说明一个共用体 data：

```
union data {
        int i;
        char ch;
        float f;
        };
```

再用已说明的共用体可定义共用体变量。例如用上面说明的共用体定义一个名为 lgc 的共用体变量可写成：

```
union data lgc;
```

共用体变量 lgc 中整型量 i、字符 ch 以及浮点变量 f 共用同一内存区域，如图 1-5 所示。因此，对三个变量中任何一个变量的赋值操作，都会影响其余变量的值。

图 1-5　共用体内变量共用内存的同一个区域

当一个共用体被说明时，编译程序自动地产生一个变量，其长度为共用体中最大的变量长度。共用体访问其成员的方法与结构相同。同样共用体变量也可以定义成数组或指针，但定义为指针时也要用“－＞”符号，此时共用体访问成员可表示成：

```
共用体名－＞成员名
```

共用体也可以出现在结构内，图 1-6 描述了下述结构定义的变量之间关系：

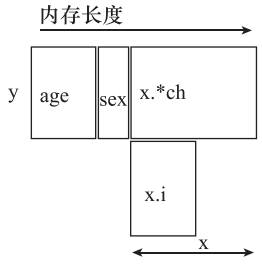


图 1-6　结构与共用体关系

```
struct {
        int age;
        char sex;
        union {
            int i;
            char *ch;
            }x;
        }y;
```

若要访问结构变量 y 中共用体 x 的成员 i，可以写成：

```
y.x.i;
```

若要访问结构变量 y 中共用体 x 的字符指针 ch 所指向的内容，可写成：

```
*y.x.ch;
```

若写成“y.x.*ch;”是错误的。

• 结构和共用体的区别

(1)结构和共用体都是由多个不同的数据类型成员组成,但在任何同一时刻,共用体中只存放了一个被选中的成员,而结构的所有成员都存在。

(2)对于共用体的不同成员赋值,将会对其他成员重写,原来成员的值就不存在了,而对于结构的不同成员赋值是互不影响的。因此,共用体中的指针操作需要特别小心,它很容易被误操作。作为加深对共用体理解的例子,请读者参见程序 1.2。

【程序 1.2】 共用体的应用

```
#include <stdio.h>
int main(void)
{
    char a;
    union {                          //注意和图 1-6 的区别
        int age;
        char sex;
        struct {
            int i;
            char *ch;
            }x;
        }y;
    y.age=10;
    y.x.i=20;                        //覆盖了 y.age,注意高位是 0
    y.sex='b';                       //覆盖了 y.x.i 的低位,'b'=98
    y.x.ch=&a;                       //y.x.ch 在地址上与共用体内其他变量无关
    *(y.x.ch)='a';
    printf("y.x.i=%d\n",y.x.i);
    printf("y.age=%d\n",y.age);
    printf("y.sex=%c\n",y.sex);
    printf("*(y.x.ch)=%c\n",*(y.x.ch));
    return (0);
}
```

运行结果:

```
y.x.i=98
y.age=98
y.sex=b
*(y.x.ch)=a
```

从程序 1.2 的结果可以看出，当给 y.sex 赋值后，也就是 y.age 和 y.x.i 的低八位值置成字符“b”的 ASCII 码 98，这两个字的高八位都是零。

1.1.3 数据结构的定义

不仅指针，数组也是 C 语言中提供的聚合某类元素的工具，它所表达的关系是相邻元素之间只存在一种线性有序关系$<a_i, a_{i+1}>$，如例 1-2 的表格数据结构，就使用了数组形式。因为数组提供的数据元素之间的关系只能是线性相邻的，往往不能满足现实中具有非线性关系的对象需要，因此，必须借助指针来表达复杂逻辑结构的数据关系。比如，现实中还可以举出含有多种关系存在的数据结构，设一批数据元素的逻辑结构如下：

$K=\{K_1, K_2, \cdots, K_{10}\}$ //一组元素

$R=\{r_1, r_2\}$ //一组关系

$r_1=\{<k_1,k_2>,<k_1,k_8>,<k_2,k_3>,<k_2,k_7>,<k_3,k_4>,<k_3,k_5>,<k_3,k_6>,<k_8,k_9>,<k_8,k_{10}>\}$

$r_2=\{<k_{10},k_4>,<k_4,k_2>\}$

这是有二元数据关系的逻辑结构，分别用实、虚线表示如图 1-7 所示。关系 r_1 表示一种树形逻辑关系，r_2 表示了(k_{10}，k_4，k_2)的顺序相邻关系。树形结构是非线性的，当然不能用线性的数组关系描述，所以必须借助指针。指针既能表达线性相邻也能表达非线性分支，这是数据结构使用指针的原因所在。现在我们给出数据结构的定义：

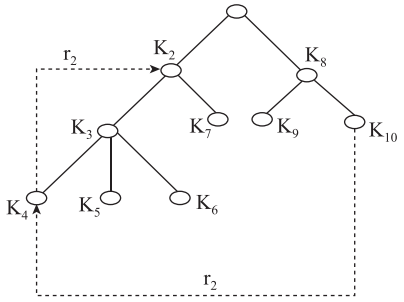


图 1-7 二元关系的逻辑结构图示法

【定义 1.1】 数据结构是一个二元组集合 S=(D,R)，其中 D 是结构变量的非空有限集合，R 是描述在 D 上的有限个关系的集合。

所以说数据结构研究的是客观事物个体属性在计算机中的表达及描述方法。在节点元素中，用计算机语言的基本变量的聚合，刻画了事物的客观属性、指针或数组的地址连接，则描述了节点之间的关联关系。数据结构的内容主要有以下 3 点。

①数据结构的逻辑结构。根据应用对象设计有限元素集合中节点之间的逻辑关系，如线性表、树、图等。

②逻辑结构在计算机中的物理实现，如顺序表、链表、二叉树等。

③数据结构中节点的操作运算，如插入、删除、检索等。

图 1-8 给出了几种基本的数据结构形式。要设计应用于计算机处理的数据结构形式，上述的定义必须与计算机的物理实现相连才有实际意义。

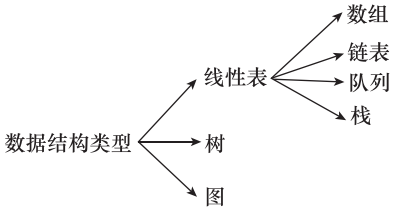


图 1-8　基本的数据逻辑结构类型

数据结构在计算机内存中的表示方法称为数据结构的物理结构，以区别前者的逻辑结构形式。物理结构有四种基本的形式，如图 1-9 所示，其中，索引存储结构用于文件操作，散列存储结构用于对数据检索。

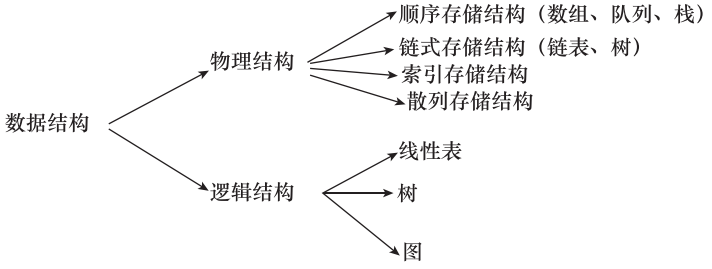
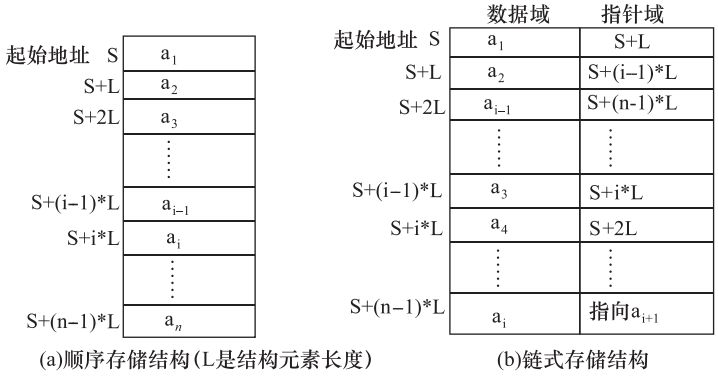


图 1-9　基本的数据物理结构类型

所谓顺序存储结构，是指将数据元素顺序地存放于计算机内存中一个连续的存储区域里，借助元素在存储器中的相对位置来表示元素之间的逻辑关系，也就是用数组描述的一群有限数据元素集合。

链式存储结构的特点，是在每个元素中加入一个指针域，它指向逻辑上相邻接的元素的存放地址。而数据元素在内存中的存放顺序与逻辑关系无关。即链式存储结构是用指针的指向来表达节点的逻辑关系，这也是 C、Pascal 语言适用于数据结构设计的原因。图 1-10 给出了顺序、链式存储结构。它们都是描述，或者说存储了线性关系$<a_i, a_{i+1}>$，但方式不同。



(a)顺序存储结构(L是结构元素长度)　　(b)链式存储结构

图 1-10　向量的顺序存储结构与链式存储结构

数据结构有线性与非线性之分。一个数据结构的关系里，除去端点外，每一个节点有且仅有一个前趋和后继时，这个数据结构就是线性的，如数组、链表。如果数据结构关系中，其节点有一个以上的前驱或后继，则称为非线性的数据结构，如树、图。一般情况下，讨论非空有限集合 D 上只有单一关系 r 的数据结构。但是，在关系数据库设计时，讨论的则是非空有限集合 D 上的一组关系 R 的数据结构设计问题。

数据结构的物理表达问题在相关参考书中已经明确给出，可以仔细阅读理解，对 C 语言不熟悉的读者尤其要注意指针和链式存储结构。本书中线性表、树是学习的重点，而线性表中的链表设计是重点内容，它应用了指针的概念。在 C 语言设计中，掌握了指针的应用方法就掌握了 C 语言的设计风格。图的内容不在本教材讨论范围，排序、检索将在数据结构之后介绍。

在结束有关数据结构的概念讨论之前，再次明确地给出数据结构内容的三要素，即：

数据结构＝数据逻辑结构＋物理结构＋数据运算

数据运算是指对数据结构的检索、插入、排序、删除、更新等操作。此外，不同数据结构之间的运算效率也是要重点考虑的内容。

1.2 数据结构和算法

1.2.1 算法的概念和特性

算法(algorithm)的意义非常类似于处方、方法和规程。通俗地说，所谓算法是指为解决给定问题而需要执行的有穷操作过程的描述。在计算机科学中，算法用来描述计算机解决给定问题的操作过程。

通常，一个算法必须具备以下五个重要特征。

1. 有穷性

有始有终是算法最基本的特征。一个算法必须在它所涉及的每种情形下，都能在执行有穷步操作之后结束。有的运算过程，操作步骤看似有限，但不能保证它在动态环境下实行执行次数的有穷性。因此，判断一个算法的有穷性应对算法所涉及的各种情形作动态分析，从而进行判断。下面介绍两个例子。

【例 1-4】 一个非算法的计数过程。

(1)开始

(2)n＜＝0

(3)n＜＝n＋1

(4)重复(3)

(5)结束

粗略地看，这个过程仅有五个步骤，是有穷的，但事实上，该过程一开始，就永不休止。

因而，在动态执行中它并不具备有穷性，它不是一个算法。当然，只要对上述过程稍加修改，限定其计数上限，即可使之成为一个算法。例 1-5 将对此进行介绍。

【例 1-5】 不超过一万次的计数算法。

(1)开始

(2)n＜＝0

(3)n＜＝n＋1

(4)若 n＝10000，则顺次执行(5)，否则重复(3)

(5)输出 n

(6)结束

2. 确定行

算法的每一步操作，其顺序和理论都必须确切定义，且不得有任何歧义性。

3. 数据输入

一个算法有 n(n≥0)个初始数据的输入。

4. 信息输出

算法是用来解决给定问题的，所以，一个算法必须将人们所关心的信息输出。也就是说，一个算法必须有一个或多个有效的输出，它是同输入数据有某种特定关系的信息。

5. 可行性

一个算法的任何一步操作都必须是可行的，即必须是可以执行并能具体实现的基本操作。也就是说，每步操作都能由计算机(或人用纸和笔)操作有限次即可完成。

1.2.2 数据结构和算法的关系

数据结构和算法是计算机科学的两个重要支柱，它们是一个不可分割的整体。

计算机求解问题的过程，从某个角度来讲，无非是：

初始数据输入→计算机处理→结果输出

对于一个给定问题的初始数据，如何组织及在计算机内如何存储，以节约存储空间和便于计算机处理；如何选择合适的算法以提高求解问题的可靠性和效率，这都是至关重要的。

例如前面列举的计算机管理问题，可用一个表来存储初始数据，那么现在要查询《数据结构》教材的库存量以决定是否要征订，应如何进行呢？如果初始数据在表内的存放是随机的，即没有什么规律的，那么查询算法只能是在表中顺序地执行逐行地查找，显然其效率是不高的。如果想提高效率，则必须对数据的组织和存储作相应的调整与改进。例如，可以让教材按教材名的字典顺序排列或按专业归类存储，并建立索引表，同时选择更有效的算法。这样，就可以极大提高查询的速度。

由此可知，不同的算法必须选用相适应的数据结构才能发挥作用，也就是说，数据结构的不同，直接影响算法的选择效率。

所以，对于一个特定问题，究竟采用何种数据结构及选用什么算法，不能一概而论，需要

具体问题具体分析。即要看问题的具体要求和现有的各种条件，把数据结构和算法这两者有机地结合起来考虑，这样才能设计出较好的计算机程序。

1.3 抽象数据类型的表示和实现

抽象数据类型可通过固有数据类型来表示和实现，即利用处理器中已存在的数据类型来说明新的结构，用已经实现的操作来组合新的操作。由于本书在高级程序设计语言的虚拟层次上讨论抽象数据类型的表示和实现，并且讨论的数据结构及其算法主要是面向读者，故采用介于伪码和C语言之间的类C语言作为描述工具，有时也用伪码描述一些只含抽象操作的抽象算法。这使数据结构与算法的描述和讨论简明清晰，不拘泥于C语言的细节，又容易转换成C或者C++程序。

1.4 算法的描述和分析

数据结构是一门具有较强实践性的学科。通过该课程的学习，学生应能运用数据结构的知识和技巧更好地进行算法与程序的设计。所以，在讨论各种数据结构的基本运算时，都给出了相应的算法。对于算法的描述，力求做到通俗易懂和便于自学，所以采用文字框图进行图示。在掌握和理解了框图所示的设计思想后，可以方便使用熟悉的算法语言来编写程序。另外，考虑到C语言是当前国际上广泛流行和很有发展前途的计算机高级语言，它能较好地体现结构化程序设计的原则，并且简洁明了，便于教学，所以本书中大部分算法在给出文字框图的同时还给出了用C语言编写的源程序片段。

另外，对算法（或程序）的分析和评价通常较为复杂。一般需考虑正确性、最优性、可读性、可维护性、运算量及占用存储空间等诸多因素。为了简化讨论，本书主要考虑其中的两条标准：一是算法实现所需的存储量，二是算法实现所需的运算量。用问题的某个参数的函数来加以估算，假设问题规模的参数为n，此参数可以使矩阵的阶、线性表的长度、图的顶点数等显示该问题规模大小的参数。那么，所选数据结构上执行某种操作所需要的存储空间及运算量是n的什么函数呢？这里引进记号“O”（读作“大O”），对这些函数作数量级的估算。下面有三个简单程序段：

```
(1)x=x+1;
(2)for(i=1;i<=n;i++)
     x=x+1;
(3)for(i=1;i<=n;i++)
     for(j=1;j<=n;j++)
     x=x+1;
```

在程序段(1)中,语句 x=x+1 不包含在任何循环体中,则此语句只执行一次,运算量可记为 O(1)。在程序段(2)中,上述赋值语句在 for 循环中,所以要执行 n 次,其执行时间和 n 成正比,运算量可记为 O(n)。在程序段(3)中,x=x+1 语句要执行 $n\times n$ 次,其执行时间和 n^2 成正比,故运算量可记为 O(n^2)。然而,要对一个算法的运算量作仔细分析是很复杂的,而且也不是本课程的主要内容,所以书中对一些算法的性能评价只根据算法中执行次数最多的语句来估算该算法的运算量的数量级。对于算法或程序所需的存储空间,本书也作了类似处理。

本章小结

本章主要介绍了数据、数据对象、数据元素、数据结构、数据类型等基本概念和有关算法、算法性能分析方法等内容。通过本章的学习,读者能够较好地理解相关概念和算法的定义及特征,掌握简单的时间复杂度和空间复杂度的计算,为后续章节的学习打下良好的基础。

本章习题

1. 什么是数据? 数据与信息是什么关系?

2. 什么是数据结构? 有关数据结构的讨论一般都涉及哪几个方面?

3. 数据的逻辑结构分为线性结构和非线性结构两大类。线性结构包括数组、链表、栈、队列、优先级队列等;非线性结构包括树、图等,这两类结构各自的特点是什么?

4. 什么是算法? 算法的 5 个特性是什么? 试根据这些特性解释算法与程序的区别。

5. 指出下列各算法的功能并求出其时间复杂度。

(1)
```
int Prime ( int n ){
    int i = 2,x = (int)sqrt ( n );
    while ( i <= x ){
      if ( n % i == 0 )break;
      i++;
    }
    if ( i > x )return 1;
    else return 0;
  }
```

(2)
```
int sum1 ( int n ){
    int p = 1,s = 0;
    for ( int i = 1; i <= n; i++ )
```

```
        { p *= i; s += p; }
    return s;
}
(3)int sum2 ( int n ){
    int s = 0;
    for ( int i = 1; i <= n; i++ ){
      int p = 1;
      for ( int j = 1; j <= i; j++ )p *= j;
      s += p;
    }
    return s;
}
(4)int fun ( int n ){
    int i = 1,s = 1;
    while ( s < n )s += ++i;
    return i;
}
(5)void UseFile ( ifstream& inp,int c[ ] ){
    //假定 inp 所对应的文件中保存有 n 个整数
    for ( int i = 0; i < 10; i++ )c[i] = 0;
    int x;
    while ( inp >> x )
      { i = x%10; c[i]++; }
}
(6)void mtable ( int n ){   for ( int i = 1; i <= n; i++ ){
      for ( int j = i; j <= n; j++ )
        cout << i << "*" << j << "=" << setw(2)<< i * j << " ";
    cout << endl;
    }
}
(7)void cmatrix ( int a[ ][ ],int M,int N,int d ){
  //M 和 N 为全局整型常量
    for ( int i = 0; i < M; i++ )
      for ( int j = 0; j < N; j++ )
        a[i][j] *= d;
}
```