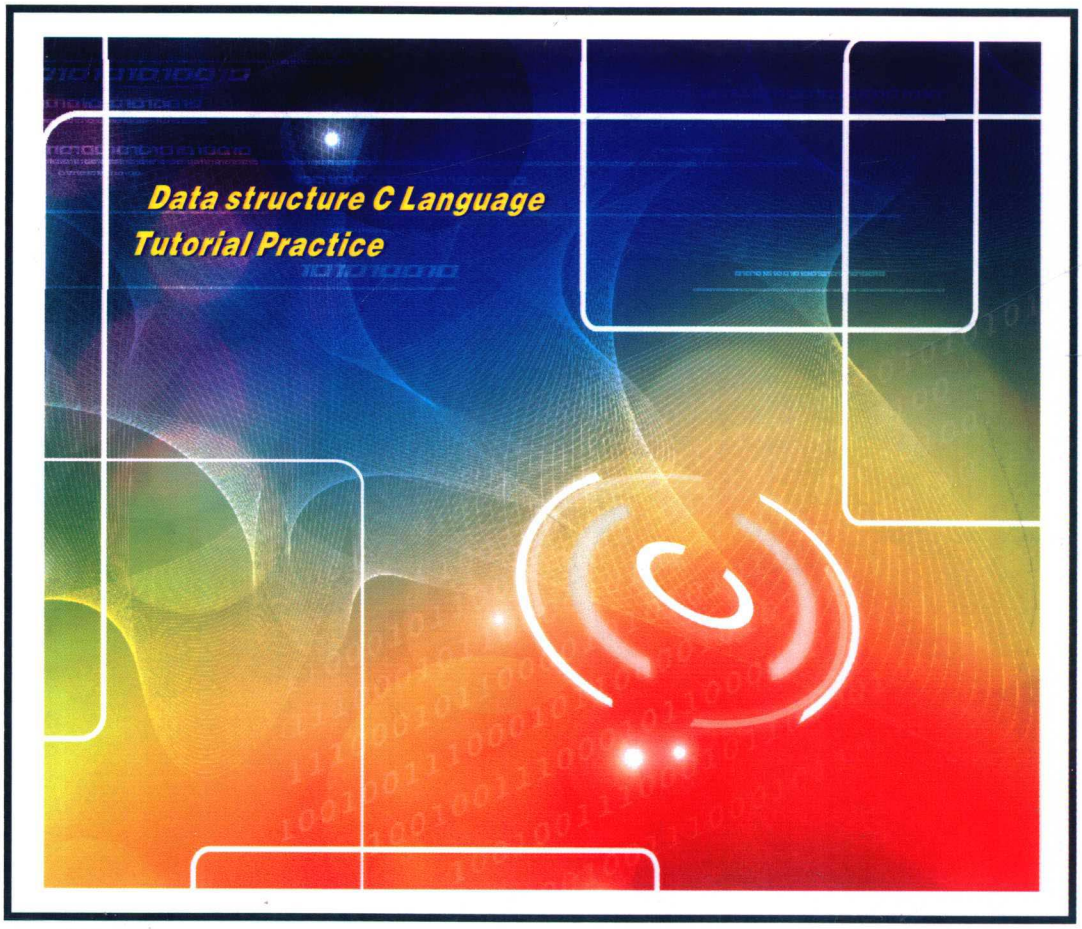
XSJ 新世纪计算机类本科规划教材 COMPUTER

数据结构（C语言）实践教程

（第二版）

主编　胡元义



西安电子科技大学出版社
http://www.xduph.com

新世纪计算机类本科规划教材

数据结构(C语言)实践教程

(第二版)

主　编　胡元义
副主编　王　磊　黑新宏　邓亚玲　段敬红
　　谈姝辰　何文娟　梁　琨
主　审　崔俊凯

西安电子科技大学出版社

内 容 简 介

本书是作者积多年讲授与研究数据结构课程及指导学生上机实践的经验编写而成的。作者力求通过实践的角度，帮助学生深入学习、理解、掌握，并灵活应用数据结构知识。全书涵盖了数据结构课程的全部上机实践内容，对数据结构所有的理论知识均对应给出了程序实现，并且这些程序都在 VC++ 6.0 环境下调试通过。

本书可以配合目前各类数据结构（C 语言）教材使用，可起到衔接教学与实践以及帮助读者开拓学习和应用视野的作用。本书实践内容丰富、程序设计独到、编程方法全面，因而也可以作为计算机应用人员的参考书。

图书在版编目(CIP)数据

数据结构(C 语言)实践教程/胡元义主编. —2 版.
—西安：西安电子科技大学出版社，2014.2
新世纪计算机类本科规划教材
ISBN 978-7-5606-3318-3

Ⅰ. ① 数… Ⅱ. ① 胡… Ⅲ. ① 数据结构—高等学校—教材 ② C 语言—程序设计—高等学校—教材 Ⅳ. ① TP311.12 ② TP312

中国版本图书馆 CIP 数据核字(2014)第 018481 号

策　　划　陈宇光
责任编辑　阎　彬
出版发行　西安电子科技大学出版社（西安市太白南路 2 号）
电　　话　(029)88242885　88201467　邮　　编　710071
网　　址　www.xduph.com　电子邮箱　xdupfxb001@163.com
经　　销　新华书店
印刷单位　北京京华虎彩印刷有限公司
版　　次　2014 年 8 月第 1 版　2014 年 8 月第 1 次印刷
开　　本　787 毫米×1092 毫米　1/16　印张 19.5
字　　数　459 千字
印　　数　8001～8510 册
定　　价　34.00 元

ISBN 978 – 7 – 5606 – 3318 – 3/TP
XDUP 3610002-3

前　言

如果把程序设计比作一棵大树，数据结构无疑是大树的躯干；可以说程序设计的精髓就是数据结构。计算机各个领域的应用都要用到各种数据结构，只有较好地掌握了数据结构知识才能在程序设计的编程中游刃有余，进而在计算机应用领域的研制和开发中做到胸有成竹。

数据结构课程是计算机专业的一门核心课程，它是在长期的程序设计实践中提炼、升华而成的，反过来又应用于程序设计。数据结构课程同时又是操作系统、编译原理等计算机核心课程的基础，在计算机专业课程的学习中起着承上启下的作用。同时，数据结构课程又是一门应用广泛且最有实用价值的课程，不掌握数据结构知识就难以成为一名合格的软件工程师或计算机工作者。

数据结构课程对理论与实践的要求都相当高，并且内容多、难度大。虽然多数数据结构教材都强调了实践的重要性，但还是缺乏供实践练习的材料，很多教材对算法的描述只是扼要和概述性的，大多数算法都采用类 C、类 C++ 或类 PASCAL 语言描述，无法直接上机实现。由于数据结构算法的上机实现要涉及栈、队列、树和图的具体存储结构，并且一个算法的实现往往要涉及其中几种结构，这就给程序的编写增加了难度；特别是初学者，更是感到无从下手。针对这种情况，我们编写了本书。在编写过程中，我们坚持“以理论知识为纲，以实践应用为线，侧重内容创新”，以达到开拓学生思维空间和提高其实际动手能力的目的。目前国家所倡导的“卓越工程师计划”就是要培养实践能力强并具有创新精神的高素质人才，我们所做的工作，可为实现这一目标打下坚实的基础。

本书可作为数据结构课程的辅助教材，供计算机专业或其它相关专业的学生学习数据结构课程时使用，以帮助学生在尽可能短的时间内对数据结构知识的实践与应用有一个比较全面、深入和系统的认识，达到理论与实践相结合的目的。

本书相对第一版，书中的内容全部都进行了重新编写，并且百分之五十以上的内容都是新增加的，原来那些未用程序实现的难度较大的算法，或算法本身不够完善而难以实现的内容，这次都在新增内容中用程序实现。对其余的实践内容这次也都重新编写了程序，其原因之一是使程序更加统一和完善，原因之二则是使其更加适应教学和实践的需要。此外，书中的许多程序都是作者独创的。本版中的所有程序全都在 VC++ 6.0 环境下调试通过，但从篇幅上考虑仅对那些比较复杂且难度较大的程序给出了运行结果。本版的另一个重要特点是对书中的所有程序都添加了详细的注释，这对读者阅读程序、理解程序如何实现算法的功能将起到事半功倍的作用。

本书分为 9 章。第 1 章～第 8 章与数据结构相应章节的理论知识衔接，各章先对该章的理论知识作简要介绍，然后给出该章涉及理论知识的全部实践内容；这些实践内容可作

为算法学习的拓展和补充，也可作为该章的实验上机内容。第 9 章是数据结构实践的扩展，给出了数据结构知识更多的应用，可起到拓展学生思维及提高学生灵活运用数据结构知识能力的作用。

各实验后的思考题大部分都可作为数据结构课程设计题目，第 9 章的内容也可作为数据结构课程设计的参考。

在本书的出版过程中，得到了西安电子科技大学出版社的热情帮助和大力支持，在此表示衷心的感谢。

由于作者水平有限，书中难免存在不妥之处，敬请广大读者批评指正。

作　者

2013 年 8 月

第一版前言

数据结构是计算机专业的主干课程之一，其目的是让读者学习、分析和研究数据对象的特性及数据的组织方法，以便选择合适的数据逻辑结构和存储结构，设计相应的运算操作，把现实世界中的问题转化为计算机内部的表示与处理的方法。在计算机科学领域，尤其是在系统软件和应用软件的设计和应用中要用到各种数据结构，因此，掌握数据结构对提高软件设计和程序编制水平有很大的帮助。

“数据结构”课对理论与实践的要求都相当高，并且内容多难度大。虽然多数数据结构教材都强调了实践的重要性，但比较缺乏供实践练习的材料，很多教材对算法的描述也只是扼要的和概述性的，很多算法都采用类 C 或类 PASCAL 语言描述，无法直接上机实现。针对这种情况，我们编写了这本《数据结构(C 语言)实践教程》。本书可作为“数据结构”课程的辅助教材，供计算机专业的学生在“数据结构”课程实习时使用，以帮助学生在尽可能短的时间内对数据结构知识的实践与应用有一个比较全面、深入和系统的认识，达到理论与实践相结合的目的。

本书分为两篇。

第一篇为数据结构的实践，共七章。该篇从实践角度论述了数据结构的有关知识，并给出了全部实验。每一个实验均给出了本次实验的目的、实验内容，对实验中的要点和难点进行了说明，并给出了相应的参考程序以供学习使用；此外，每个实验后还附有思考题，它使读者能够在此实验的基础上做进一步的思考与提高，从而达到举一反三、触类旁通的目的。第一章：线性表，重点介绍了两种存储结构——顺序表和单链表的操作，特别是单链表，它是贯穿全书的其它逻辑结构——树、图的基础；第二章：栈和队列，对常用的栈和队列进行了介绍；第三章：串与数组，对串的运算特别是串的模式匹配以及数组中的稀疏矩阵做了重点介绍；第四章：树和二叉树，它是本书的重点内容之一，重点介绍了递归和非递归遍历二叉树算法，此外，还介绍了如何由遍历序列恢复二叉树的实现方法；第五章：图，也是本书的重点，包括图的搜索算法、最小生成树构造方法和最短路径；第六章，排序，也是本书的重点，介绍了各种排序方法的实现；第七章：查找，介绍了各种查找算法。

第二篇为数据结构的应用与提高，共两章。第八章：数据结构应用实例，为开拓学习视野给出了数据结构的具体应用，即(1) 线性表应用——仓库管理；(2) 栈的应用——表达式转换；(3) 队列应用——一个简单事件的规划问题；(4) 二叉树应用——银行财务实时处理系统；(5) 图的应用——工程工期控制问题；(6) 查找应用——学生档案管理。第九章：数据结构典型问题研究，为进一步深入学习、研究数据结构知识给出了(1) 最短路径输出

问题研究；(2) 递归转换为非递归问题研究；(3) 人工智能应用研究。

为了阅读方便，在表述串、顶点、结点等时，语言叙述部分的外文字符的大小写，以及采用的下标符等与程序有所差异。

在本书的出版过程中，得到了西安电子科技大学出版社，尤其是陈宇光老师的热情帮助和大力支持，在此表示衷心的感谢。

由于作者水平有限，书中难免存在错误和不妥之处，敬请广大读者批评指正。

编　者

2004 年 1 月

目　录

第 1 章　线性表 1
1.1　内容与要点 1
1.1.1　线性表的定义 1
1.1.2　线性表的顺序存储——顺序表 1
1.1.3　线性表的链式存储 2
1.2　线性表实践 4
实验 1　顺序表及基本运算 4
实验 2　在表头插入生成单链表 7
实验 3　在表尾插入生成单链表 10
实验 4　单链表及基本运算 11
实验 5　双向链表及基本运算 15
实验 6　静态链表 19

第 2 章　栈和队列 23
2.1　内容与要点 23
2.1.1　栈 23
2.1.2　队列 24
2.2　栈和队列实践 26
实验 1　顺序栈及基本运算 26
实验 2　链栈及基本运算 29
实验 3　循环队列及基本运算 31
实验 4　链队列及基本运算 33

第 3 章　串 36
3.1　内容与要点 36
3.2　串实践 37
实验 1　顺序串及基本运算 37
实验 2　链串及基本运算 40
实验 3　链串中求子串运算 42
实验 4　链串中串插入运算 44
实验 5　串的简单模式匹配 46
实验 6　串的无回溯 KMP 匹配 48

第 4 章　数组与广义表 52
4.1　内容与要点 52
4.1.1　数组 52
4.1.2　特殊矩阵 54
4.1.3　稀疏矩阵 54
4.1.4　广义表 57
4.2　数组与广义表实践 58
实验 1　矩阵转置 58
实验 2　矩阵的快速转置 61
实验 3　稀疏矩阵的十字链表存储 65
实验 4　广义表及基本运算 68

第 5 章　树与二叉树 73
5.1　内容与要点 73
5.1.1　树 73
5.1.2　二叉树 73
5.1.3　二叉树的性质 74
5.1.4　二叉树的存储结构 74
5.1.5　二叉树的遍历方法 75
5.1.6　线索二叉树 76
5.1.7　哈夫曼树 77
5.1.8　哈夫曼编码 78
5.2　树与二叉树实践 79
实验 1　二叉树的遍历 79
实验 2　二叉树的非递归遍历 81
实验 3　另一种非递归后序遍历
二叉树的方法 85
实验 4　二叉树遍历的应用 87
实验 5　由二叉树遍历序列恢复二叉树 91
实验 6　按层次遍历二叉树 94
实验 7　中序线索二叉树 97
实验 8　哈夫曼树与哈夫曼编码(1) 101
实验 9　哈夫曼树与哈夫曼编码(2) 107

第 6 章　图 113
6.1　内容与要点 113
6.1.1　图 113
6.1.2　邻接矩阵 113
6.1.3　邻接表 114
6.1.4　图的遍历 115
6.1.5　图的连通性问题 115
6.1.6　生成树与最小生成树 115
6.1.7　最短路径 116
6.1.8　AOV 网与拓扑排序 117
6.1.9　AOE 网与关键路径 118
6.2　图实践 120
实验 1　建立无向图的邻接矩阵 120
实验 2　图的深度优先搜索 122
实验 3　图的广度优先搜索 125
实验 4　图的连通性 129
实验 5　深度优先生成树 132
实验 6　广度优先生成树 135
实验 7　最小生成树的 Prim 算法 138
实验 8　最小生成树的 Kruskal 算法 142
实验 9　单源点最短路径的 Dijkstra 算法 145
实验 10　每一对顶点间最短路径的 Floyd 算法 148
实验 11　拓扑排序 151
实验 12　关键路径 155

第 7 章　查找 162
7.1　内容与要点 162
7.1.1　顺序查找 162
7.1.2　有序表的查找 162
7.1.3　二叉排序树与平衡二叉树 163
7.1.4　哈希表与哈希方法 163
7.1.5　哈希函数的构造方法 164
7.1.6　处理冲突的方法 165
7.2　查找实践 165
实验 1　顺序查找 165
实验 2　折半(二分)查找 167
实验 3　分块查找 169
实验 4　二叉排序树 171
实验 5　平衡二叉树 176
实验 6　哈希(Hash)查找 186

第 8 章　排序 190
8.1　内容与要点 190
8.1.1　插入排序 190
8.1.2　交换排序 191
8.1.3　选择排序 191
8.1.4　归并排序 193
8.1.5　基数排序 193
8.2　排序实践 194
实验 1　插入排序 194
实验 2　折半插入排序 196
实验 3　希尔(Shell)排序 198
实验 4　冒泡排序 200
实验 5　快速排序 202
实验 6　选择排序 208
实验 7　堆排序 210
实验 8　归并排序 213
实验 9　基数排序 218

第 9 章　数据结构实践应用 222
9.1　顺序表的应用 222
9.1.1　顺序表的逆置 222
9.1.2　将两个升序的顺序表 A 和 B 合并为一个升序的顺序表 C 223
9.1.3　单链表的逆置 225
9.1.4　将递增有序的单链表 A 和 B 合并成递减有序的单链表 C 227
9.1.5　删除单链表中值相同的结点 229
9.1.6　按递增次序输出单链表中各结点的数据值 231
9.2　栈和队列应用 233
9.2.1　用栈判断给定的字符序列是否为回文 233
9.2.2　循环链表中只有队尾指针的入队和出队算法 235

9.2.3 算术表达式中的括号匹配 237
9.2.4 将队列中所有元素逆置 240
9.2.5 用两个栈模拟一个队列 243
9.3 串的应用 246
9.3.1 将串 s1 中一字符串用串 s2 替换 246
9.3.2 计算一个子串在字符串中出现的次数 248
9.3.3 输出长度最大的等值子串 249
9.3.4 将链串 s 中首次与链串 t 匹配的子串逆置 250
9.4 数组与广义表应用 253
9.4.1 将所有奇数放到数组前半部分，所有偶数放到数组后半部分 253
9.4.2 求出字符数组中连续相同字符构成的子序列长度 254
9.4.3 求广义表的表头和表尾 255
9.4.4 另一种广义表生成方法 259
9.5 树与二叉树应用 263
9.5.1 交换二叉树的左右子树 263
9.5.2 统计二叉树叶子个数的非递归算法实现 265
9.5.3 判定一棵二叉树是否为完全二叉树 267
9.5.4 求二叉树中第一条最长的路径并输出此路径上各结点的值 270
9.6 图的应用 273
9.6.1 邻接矩阵转换为邻接表 273
9.6.2 深度优先搜索的非递归算法实现 275
9.6.3 求无向连通图中距顶点 v_0 路径长度为 k 的所有结点 278
9.6.4 用深度优先搜索对图中所有顶点进行拓扑排序 281
9.7 查找的应用 284
9.7.1 判定一棵二叉树是否为二叉排序树 284
9.7.2 另一种平衡二叉树的生成方法 287
9.8 排序的应用 293
9.8.1 用双向循环链表表示的插入排序 293
9.8.2 双向冒泡排序 295
9.8.3 单链表存储下的选择排序 297
9.8.4 归并排序的迭代算法实现 299

参考文献 302

第1章 线 性 表

1.1 内容与要点

1.1.1 线性表的定义

线性表是最基本、最常用的一种线性结构。线性结构的特点是：数据元素之间是线性关系，数据元素“一个接一个地排列”；并且，所有数据元素的类型都是相同的。简单地说，一个线性表是 n 个元素的有限序列，其特点是在数据元素的非空集合中：

(1) 存在唯一一个称为“第一个”的元素；

(2) 存在唯一一个称为“最后一个”的元素；

(3) 除第一个元素之外，序列中的每一个元素都只有一个直接前驱；

(4) 除最后一个元素之外，序列中的每一个元素都只有一个直接后继。

1.1.2 线性表的顺序存储——顺序表

线性表的顺序存储是用一组地址连续的存储单元按顺序依次存放线性表中的每一个数据元素。在这种顺序存储结构中，逻辑上相邻的两个元素在物理位置上也相邻；无需增加额外的存储空间来表示线性表中元素之间的逻辑关系。这种顺序存储结构即为顺序表。

由于顺序表中每个元素具有相同的类型，即其长度相同，故顺序表中第 i 个元素 a_i 的存储地址为

$$Loc(a_i) = Loc(a_1) + (i - 1) \times L \qquad 1 \leqslant i \leqslant n$$

其中：$Loc(a_1)$为顺序表的起始地址(即第一个元素的地址)；L 为每个元素所占存储空间的大小。由此可知，顺序表中的任意一个元素都可以随机存取。

从结构上考虑，我们将 data 和 len 组合在一个结构体里来作为顺序表的类型：

```
typedef struct
{
   datatype data[MAXSIZE];      //存储顺序表中的元素
   int len;                     //顺序表的表长
}SeqList;                       //顺序表类型
```

其中：datatype 为顺序表中元素的类型，在具体实现中可为 int、float、char 类型或其它结构类型；len 为顺序表的表长。顺序表中的元素可存放在 data 数组中下标为 1～MAXSIZE 的任何一个位置。第 i 个元素的实际存放位置就是 i。

1.1.3 线性表的链式存储

线性表的链式存储方式可用连续或不连续的存储单元来存储线性表中的元素，在这种方式下元素之间的逻辑关系已无法再用物理位置上的邻接关系来表示。因此，需要用“指针”来指示元素之间的逻辑关系，而这种“指针”是要占用额外存储空间的。链式存储方式失去了顺序表可以随机存取数据元素的功能，但却换来了存储空间操作的方便性：进行插入和删除操作时无需移动大量的元素。

1. 单链表

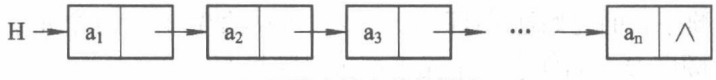
采用链式存储最简单也最常用的方法是：在每个元素中除了含有数据信息外，还要有一个指针，用来指向它的直接后继元素，即通过指针建立起元素之间的线性关系。我们称这种元素为结点，结点中存放数据信息的称为数据域，存放指向后继结点指针的称为指针域(如图 1-1 所示)。因此，线性表中的 n 个元素通过各自结点的指针域“链”在一起而被称为链表；因为每个结点中只有一个指向后继结点的指针，故称其为单链表。

图 1-1 单链表结点结构

链表是由一个个结点构成的。单链表结点的定义如下：

```
typedef struct node
{
   datatype data;              // data 为结点的数据信息
   struct node *next;          // next 为指向后继结点的指针
}LNode;                        //单链表结点类型
```

通常我们用“头指针”来标识一个单链表，如单链表 L、单链表 H 等是指单链表中的第一个结点的地址存放在指针变量 L 或 H 中；当头指针为“NULL”时，则表示单链表为空(见图 1-2)。

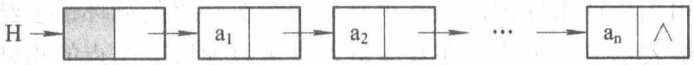


(a) 不带头结点的单链表

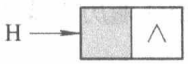
(b) 不带头结点的空单链表

图 1-2 不带头结点的单链表示意

在线性表的链式存储中，为了便于插入和删除算法的实现且使单链表的操作在各种情况下统一，在单链表的第一个结点之前添加了一个头结点；该头结点不存储任何数据信息，只是用其指针域中的指针指向单链表的第一个数据结点，即通过头指针指向的头结点，可以访问到单链表中的所有数据结点(见图 1-3)。



(a) 带头结点的单链表



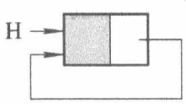
(b) 带头结点的空单链表

图 1-3 带头结点的单链表

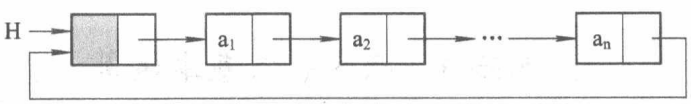
添加头结点后，无论单链表中的结点如何变化，比如插入新结点、删除单链表中任意一个数据结点，头结点将始终不变，这使得单链表的运算变得更加简单。

2. 循环链表

所谓循环链表，就是将单链表中最后一个结点的指针值由空改为指向单链表的头结点，使整个链表形成一个环。这样，从链表中的任意一个位置出发都可以找到链表的其它结点(如图 1-4 所示)。在循环链表上的操作基本上与单链表相同，只是将原来判断指针是否为NULL 改为判断是否为头指针即可。



(a) 循环链表为空



(b) 循环链表非空

图 1-4 带头结点的循环链表示意

3. 双向链表

所谓双向链表，就是指链表的每一个结点中除了数据域之外，还设置了两个指针域，一个用来指向该结点的直接前驱结点，另一个用来指向该结点的直接后继结点。每个结点的结构如图 1-5 所示。

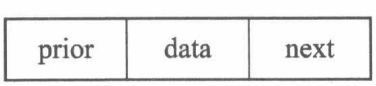
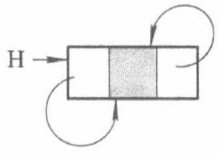


图 1-5 双向链表的结点结构

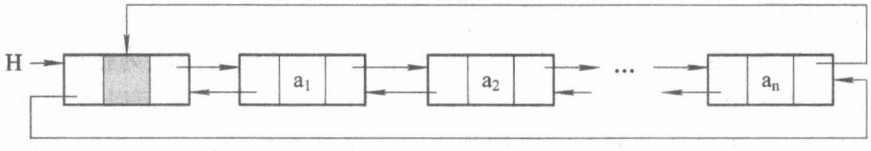
双向链表的结点定义如下：

```
typedef struct dlnode
{
   datatype data; // data 为结点的数据信息
   struct dlnode *prior,*next;
                  // prior 和 next 分别为指向直接前驱和直接后继结点的指针
}DLNode;
```

双向链表也用头指针来标识，通常也是采用带头结点的循环链表结构。图 1-6 是带头结点的双向循环链表示意；也即，在双向链表中可以通过某结点的指针 p 直接得到指向它的后继结点指针 p->next，也可直接得到指向它的前驱结点指针 p->prior。这样，在需要找到前驱结点的操作时就无需再进行循环遍历查找了。



(a) 双向循环链表为空



(b) 双向循环链表非空

图 1-6　带头结点的双向循环链表示意

1.2　线性表实践

实验 1　顺序表及基本运算

1. 实验目的

了解顺序表的结构特点及有关概念，掌握顺序表的基本运算。

2. 实验内容

建立一个顺序表，对顺序表进行初始化、插入、删除和查找运算。

3. 参考程序

```
#include<stdio.h>
#include<stdlib.h>
#define MAXSIZE 20
typedef struct
{
   int data[MAXSIZE];                    //存储顺序表中的元素
   int len;                              //顺序表的表长
}SeqList ;                               //顺序表类型
SeqList *Init_SeqList()
{                                        //顺序表初始化
   SeqList *L;
```

```
    L=(SeqList*)malloc(sizeof(SeqList));
    L->len=0;
    return L;
}
void CreatList(SeqList *L)
{                                   //建立顺序表
    int i;
    printf("Input length of List:");
    scanf("%d",&L->len);
    printf("Input elements of List:\n");
    for(i=1;i<=L->len;i++)
        scanf("%d",&L->data[i]);
}
void Insert_SeqList(SeqList *L,int i,int x)
{                                   //在顺序表中插入元素
    int j;
    if(L->len==MAXSIZE-1)           //表满
        printf("The List is full!\n");
    else
        if(i<1||i>L->len+1)         //插入位置非法
            printf("The position is invalid !\n");
        else
        {
            for(j=L->len;j>=i;j--) //将 an~ai 顺序后移一个元素位置
                L->data[j+1]=L->data[j];
            L->data[i]=x;           //插入 x 到第 i 个位置
            L->len++;               //表长增 1
        }
}
void Delete_SeqList(SeqList *L, int i)
{                                   //在顺序表中删除元素
    int j;
    if(L->len==0)                   //表为空
        printf("The List is empt !\n");
    else
        if(i<1 || i>L->len)         //删除位置非法
            printf("The position is invalid !\n");
```

```
        else
        {
           for(j=i+1;j<=L->len;j++)//将a[i+1]~a[n]顺序前移一个位置实现对a[i]的删除
              L->data[j-1]=L->data[j];
           L->len--;                    //表长减 1
         }
}
int Location_SeqList(SeqList *L,int x)
{                                       //在顺序表中查找元素
   int i=1;                             //从第一个元素开始查找
   while(i<L->len&&L->data[i]!=x) //顺序表未查完且当前元素不是要找的元素
      i++;
   if(L->data[i]==x)
      return i;                         //找到则返回其位置值
   else
      return 0;                         //未找到则返回 0 值
}
void print(SeqList *L)
{                                       //顺序表的输出
   int i;
   for(i=1;i<=L->len;i++)
    printf("%4d",L->data[i]);
   printf("\n");
}
void main()
{
   SeqList *s;
   int i,x;
   s=Init_SeqList();                    //顺序表初始化
   printf("Creat List:\n");
   CreatList(s);                        //建立顺序表
   printf("Output list:\n");
   print(s);                            //输出所建立的顺序表
   printf("Input  element and site of insert:\n");
   scanf("%d%d",&x,&i);                 //输入要插入的元素 x 值和位置值 i
   Insert_SeqList(s,i,x);               //将元素 x 插入到顺序表中
   printf("Output list:\n");
   print(s);                            //输出插入元素 x 后的顺序表
```

```
    printf("Input  element site of delete:\n");
    scanf("%d",&i);                    //输入要删除元素的位置值 i
    Delete_SeqList(s,i);               //删除顺序表第 i 个位置上的元素
    printf("Output list:\n");
    print(s);                          //输出删除元素后的顺序表
    printf("Input  element value of location:\n");
    scanf("%d",&x);                    //输入要查找的元素 x 值
    i=Location_SeqList(s,x);      //定位要查找的元素 x 在顺序表中的位置
    printf("element %d site is %d\n",x,i);  //输出该位置的元素值
}
```

4. 思考题

(1) 如按由表尾至表头的顺序输入顺序表元素，则顺序表应如何建立？

(2) 每次删除操作都会使大量的元素移动，删除多个元素就要多次移动大量的元素，能否一次进行多个元素的删除操作，并使元素的移动只进行一次？

(3) 如何实现顺序表的逆置？

实验 2 在表头插入生成单链表

1. 概述

单链表建立是从空表开始的，每读入一个数据就申请一个结点，然后插在头结点之后。图 1-7 给出了存储线性表('A', 'B', 'C', 'D')的单链表建立过程，因为是在单链表头部插入，故读入数据的顺序与线性表中元素的顺序正好相反。

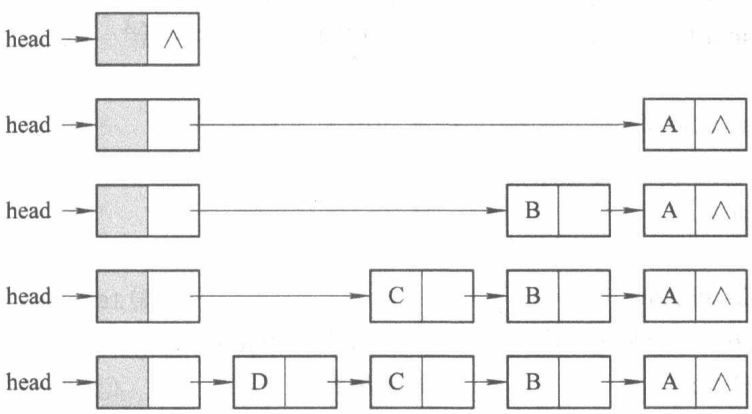


图 1-7 在头部插入建立单链表

2. 实验目的

了解单链表的结构特点及有关概念，掌握在表头插入元素生成单链表的方法。

3. 实验内容

在表头插入元素建立一个单链表。