黄明 梁旭 金花 编著

全国高等教育自学考试

数据结构习题详解

计算机及应用专业
计算机信息管理专业
独立本科段

机械工业出版社
CHINA MACHINE PRESS

全国高等教育自学考试

数据结构习题详解

（计算机及应用专业、计算机信息管理专业 独立本科段）

黄 明 梁 旭 金 花 编著

机械工业出版社

本书是根据“全国自学考试（计算机及应用专业、计算机信息管理专业 独立本科段）考试大纲”以及历年考题编写的。全书共分 4 部分：第 1 部分是笔试应试指南；第 2 部分是笔试题解；第 3 部分是模拟试卷及参考答案；最后是附录，包括考试大纲和 2002 年下半年试卷。

本书紧扣考试大纲，内容取舍得当，叙述通俗易懂，附有大量与考试题型类似的习题及答案，以检查对考点的掌握程度。

本书适用于准备参加全国自学考试（计算机及应用专业、计算机信息管理专业 独立本科段）的考生，也可作为大专院校和培训班的教学参考书。

图书在版编目（CIP）数据

数据结构习题详解/黄明等编著. —北京：机械工业出版社，2003.9
（全国高等教育自学考试）
ISBN 7-111-12886-9

Ⅰ. 数… Ⅱ. 黄… Ⅲ. 数据结构—高等教育—自学考试—解题
Ⅳ. TP311.12-44

中国版本图书馆 CIP 数据核字（2003）第 070557 号

机械工业出版社（北京市百万庄大街 22 号 邮政编码 100037）
策　　划：胡毓坚
责任编辑：孙　业
责任印制：闫　焱

北京中加印刷有限公司印刷・新华书店北京发行所发行

2003 年 9 月第 1 版・第 1 次印刷
787mm×1092mm 1/16・11.5 印张・279 千字
0001－5000 册
定价：18.00 元

出 版 说 明

全国高等教育自学考试指导委员会推出面向社会的高等自学考试，经过 10 多年的实践，已建立起一整套较为完善的规章制度和操作程序，考试组织严密规范，考试纪律严格；坚持考试标准，实行教考分离，确保了毕业生的质量。它为没有机会进入高等学校的中国公民提供了接受高等教育的机会，并以严格的国家考试保证了毕业生的质量，获得了普遍赞誉。国家自考中心于 2002 年开始执行新的考试计划。新计划中开设的专业共 224 个，其中专科 141 个占 63%，独立本科段 61 个占 27%，专本衔接专业 22 个占 10%。为帮助、指导广大自学考生深入理解计算机及相关专业考试的基本概念，灵活运用基本知识，掌握解题方法和技巧，熟悉考试模式，进一步提高应试能力和计算机水平，特编写了以下专业的基础课与专业课主要课程的习题详解。

- ✧ 计算机及应用专业 独立本科段
- ✧ 计算机信息管理专业 独立本科段
- ✧ 计算机网络专业 独立本科段
- ✧ 计算机及应用专业 专科

丛书特点：

1．以 2002 年最新考试大纲为基准

本丛书是根据 2002 年最新考试大纲，为参加全国高等教育自学考试考生编写的一套习题详解教材。

2．例题反映了历届考试中的难度和水平

书中对大量的例题进行了分析，所选例题都是在对最近几年考题深入研究的基础上精心筛选的，从深度和广度上反映了历届考试中的难度和水平。

3．作者经验丰富

本丛书的作者都是多年从事全国高等教育自学考试辅导的高等院校的教师。

读者对象：

- ✧ 准备参加全国高等教育自学考试的考生。
- ✧ 计算机及相关专业的本专科生。

前言

自学考试是对自学者进行以学历考试为主的国家高等教育学历考试。本书是为帮助和指导广大考生深入理解考点涉及的基本概念，灵活运用基本知识，掌握解题方法和技巧，熟悉考试模式，进一步提高应试能力和计算机水平而编写的。

全书共分 4 部分，即笔试应试指南、笔试题解、模拟试卷及参考答案和附录。书中所选例题均是在对历年真题深入研究的基础上精心筛选的，从深度和广度上反映了考试的难度和水平。模拟试卷的题型分配与真题一致，这些题目是考试指导教师的多年积累，在辅导班中多次实际使用过。

书中附录给出了“全国自学考试（计算机及应用专业、计算机信息管理专业 独立本科段）数据结构考试大纲”，以及“2002 年下半年全国自学考试数据结构试卷及参考答案”。

本书由黄明、梁旭、金花编写。

由于编者水平有限，编写时间仓促，书中错误和不妥之处在所难免，请读者和专家批评指正。

读者在使用本书的过程中如有问题，可通过 E-mail 与我们联系：

dlhm@263.net

编 者

目　录

第3部分 模拟试卷及参考答案

第1部分

笔试应试指南

笔试应试策略

笔试考点归纳

1.1 笔试应试策略

全国自学考试（计算机及应用专业、计算机信息管理专业 独立本科段）数据结构考试大纲涵盖了概论、线性表、栈和队列、串、多维数组和广义表、树、图、排序、查找、文件等 10 章内容。使用的教材是由全国高等教育自学考试指导委员会组编，黄刘生编著的《数据结构》，2000 年 3 月由经济科学出版社出版。考试复习的过程中要紧紧围绕大纲的知识点，首先对大纲涉及的各章基本概念熟练掌握。

第 1 章概论，是整个课程的导引，集中介绍数据结构的概念、实现方法和算法复杂度等贯穿于课程始终的基本概念和主要工具。本部分的难点在于对算法时间复杂度的分析。占分量约为 5 分。

第 2 章线性表，是课程的基础部分，介绍最简单、最基本的数据结构。本部分的难点是指针、动态链表的概念和链表上的算法设计。占分量约为 10 分。

第 3 章栈和队列，它们的逻辑结构与线性表的逻辑结构相同，都是运算受限的线性表。栈和队列的难点在理解循环队列队满和队空的区分方法和算法设计。占分量约为 10 分。

第 4 章串，串的难点在于串的模式匹配算法。占分量约为 5 分。

第 5 章多维数组和广义表。多维数组和广义表的的难点是确定多维数组元素与一维存储向量的元素之间的对应关系和稀疏矩阵的三元组存储与基于它的一些矩阵算法。占分量约为 5 分。

第 6 章树，是课程的难点，尤其是二叉树部分。树一般来说所占比例较大，占分量为 20 分。

第 7 章图，本部分的内容也是课程的一个难点，考核的重点是图的基本概念和简单应用。图的难点是连通图的两种遍历算法和 Prim 算法。占分量约为 15 分。

第 8 章排序，排序的重点是掌握快速排序、堆排序、归并排序和基数排序的基本思想及排序过程，排序的难点是这四个排序算法的实现。占分量约为 15 分。

第 9 章查找，重点掌握顺序查找、二分查找、二叉查找树上查找以及散列表上查找的基本思想和算法实现，查找的难点是二叉查找树的删除算法以及 B-树上的插入和删除算法。占分量约为 10 分。

第 10 章文件，主要介绍存储在外存上的数据结构（即文件）的有关概念、各种文件的特点、组织方法及查询和更新操作，要求对这些内容做一般性的了解。占分量约为 5 分。

在复习时根据大纲里提供的考核点和考核要求来进行复习，这样就能抓住重点，进行有效复习，在做练习时，要根据考试的题型进行练习，在掌握基本概念的基础上，掌握一定的解题技巧。软件工程的考试题型有：单选题、填空题、简答题、程序填空题和程序设计题等题型。对于不同题型，要采用不同的答题方法。

单选题：这种题型可考查考生的理解、推理分析，综合比较，评分客观。在答题时，如果可以，直接得出正确答案，对于没有太大把握的试题，也可以采用排除法，经过分析比较加以逐步排除错误答案，最终选定正确答案。

填空题：这种题型常用于考核考生观察能力与运用有关公式、原理的能力。在答题时，

无论有几个空，回答都应明确、肯定，考生在复习中最好的应对办法是对学科知识中最基本的知识、概念、原理等要牢记。

简答题：这种题型一般围绕基本概念、原理及其联系进行命题，着重考核考生对概念、知识、原理的掌握、辨别和理解能力。在答题时，答案要有层次性，条理清晰，列出要点，同时加以简要扩展就可以。

程序填空题：这种题型主要考核考生对算法的分析能力和全面理解程序的能力，在全面掌握程序功能的基础上进行填空。

程序设计题：这种题型着重考核考生分析、解决实际问题的能力，考核考生综合应用能力。在答题时，要综合运用所学知识进行分析和设计。

考生在复习时在掌握知识点的同时也应抓住这些题型的特点，这样才能达到好的应试效果。

1.2 笔试考点归纳

1.2.1 概论

1. 基本概念和术语

数据（Data）是信息的载体，它能够被计算机识别、存储和加工处理。它是计算机程序加工的“原料”。

数据元素（Data Element）是数据的基本单位。有些情况下，数据元素也称为元素、结点、顶点、记录。有时一个数据元素可以由若干个数据项（也可称为字段、域、属性）组成，数据项是具有独立含义的最小标识单位。

数据结构（Data Structure）指的是数据之间的相互关系，即数据的组织形式。虽然至今没有一个关于数据结构的标准定义，但它一般包括以下三个方面的内容：

1) 数据元素之间的逻辑关系，也称为数据的逻辑结构（Logical Structure）；

2) 数据元素及其关系在计算机存储器内的表示，称为数据的存储结构（Storage Structure）；

3) 数据的运算，即对数据施加的操作。

数据类型是一个值的集合以及在这些值上定义的一组操作的总称。

抽象数据类型（Abstract Data Type 简称 ADT）是指抽象数据的组织和与之相关的操作。它可以看作是数据的逻辑结构及其在逻辑结构上定义的操作。

ADT 和类的概念实际上反映了程序或软件设计的两层抽象：ADT 相当于是在概念层（或称为抽象层）上描述问题，而类相当于是在实现层上描述问题。

在不会产生混淆的前提下，我们常常将数据的逻辑结构简称为数据结构。数据的逻辑结构有两大类：

1) 线性结构的逻辑特征是：若结构是非空集，则有且仅有一个开始结点和一个终端结点，并且所有结点都最多只有一个直接前趋和一个直接后继。线性表就是一个典型的线性结构。

2) 非线性结构的逻辑特征是一个结点可能有多个直接前趋和直接后继。

数据的存储结构可用以下四种基本的存储方法得到：

(1) 顺序存储方法。

该方法是把逻辑上相邻的结点存储在物理位置上相邻的存储单元里，结点间的逻辑关系由存储单元的邻接关系来体现。由此得到的存储表示称为顺序存储结构（Sequential Storage Structure），通常顺序存储结构是借助于程序语言的数组来描述的。

(2) 链接存储方法。

该方法不要求逻辑上相邻的结点在物理位置上亦相邻，结点间的逻辑关系是由附加的指针字段表示的。由此得到的存储表示称为链式存储结构（Linked Storage Structure），通常要借助于程序语言的指针类型来描述它。

(3) 索引存储方法。

该方法通常是在存储结点信息的同时，还建立附加的索引表。索引表中的每一项称为索引项，索引项的一般形式是：（关键字，地址），关键字是能惟一标识一个结点的那些数据项。若每个结点在索引表中都有一个索引项，则该索引表称之为稠密索引（Dense Index）。若一组结点在索引表中只对应一个索引项，则该索引表称之为稀疏索引（Sparse Index）。稠密索引中索引项的地址指示结点所在的存储位置，而稀疏索引中索引项的地址则是指示一组结点的起始存储位置。

(4) 散列存储方法。

该方法的基本思想是根据结点的关键字直接计算出该结点的存储地址。

2．学习数据结构的意义

数据结构是计算机软件和计算机应用专业的核心课程之一，在众多的计算机系统软件和应用软件中都要用到数据结构。

随着计算机应用领域的扩大和软、硬件的发展，“非数值性问题”越来越显得重要。据统计，当今处理非数值性问题占用了90%以上的机器时间，这类问题涉及到的数据结构更为复杂，数据元素之间的相互关系一般无法用数学方程式加以描述。因此，解决此类问题的关键问题已不再是分析数学和计算方法，而是要设计出合适的数据结构，才能有效的解决问题。

著名的瑞士计算机科学家沃思教授曾提出：算法+数据结构=程序。这里的数据结构是指数据的逻辑结构和存储结构，而算法则是对数据运算的描述。由此可见，程序设计的实质是对实际问题选择一种好的数据结构，加之设计一个好的算法，而好的算法在很大程度上取决于描述实际问题的数据结构。

3．算法的描述与分析

非形式的说，算法是任意一个良定义的计算过程，它以一个或多个值作为输入，并产生一个或多个值作为输出。因此一个算法是一系列将输入转换成输出的计算步骤。一个算法也可以被认为是用来解决一个计算问题的工具。

一个算法所耗费的时间，应该是该算法中每条语句的执行时间之和，而每条语句的执行时间是该语句的执行次数（Frequency Count，也称为频度）与该语句执行一次所需时间的乘积。

一般地，我们将算法求解问题的输入量称为问题的规模（Size），并用一个整数表示。一个算法的时间复杂度（Time Complexity，也称时间复杂性）T(n)则是该算法的时间耗费，

它是该算法所求解问题规模 n 的函数。当问题的规模 n 趋向无穷大时，我们把时间复杂度 T(n) 的数量级（阶）称为算法的渐近时间复杂度。

将常见的时间复杂度，按数量级递增排列，则依次为：常数阶 O(1)、对数阶 $O(\log_2 n)$、线性阶 O(n)、线性对数阶 $O(n\log_2 n)$、平方阶 $O(n^2)$、立方阶 $O(n^3)$、…K 次方阶 $O(n^k)$、指数阶 $O(2^n)$。显然，时间复杂度为指数阶 $O(2^n)$的算法效率极低，当 n 值稍大时就无法应用。

一个算法的空间复杂度（Space Complexity）S(n)定义为该算法所耗费的存储空间，它也是问题规模 n 的函数。渐近空间复杂度也常常简称为空间复杂度。

算法的时间复杂度和空间复杂度合称为算法的复杂度。

1.2.2 线性表

1. 线性表的逻辑结构

线性表（Linear List）是由 n（n≥0）个数据元素（结点）a_1，a_2，…，a_n 组成的有限序列。其中数据元素的个数 n 定义为表的长度。当 n=0 时称为空表，常常将非空的线性表（n>0）记作：（a_1，a_2，…，a_n）

这里的数据元素 a_i（1≤i≤n）只是一个抽象的符号其具体含义在不同情况下可以不同。

常见的线性表的基本运算有如下六种：

1) InitList（L）。构造一个空的线性表 L，即表的初始化。

2) ListLength（L）。求线性表 L 中的结点个数，即求表长。

3) GetNode（L，i）。取线性表 L 中的第 i 个结点，这里要求 1≤i≤ListLength（L）。

4) LocateNode（L，x）。在 L 中查找值为 x 的结点，并返回该结点在 L 中的位置。若 L 中有多个结点的值和 x 相同，则返回首次找到的结点位置；若 L 中没有结点的值为 x，则返回一个特殊值表示查找失败。

5) InsertList（L，x，i）。在线性表 L 的第 i 个位置上插入一个值为 x 的新结点，使得原编号为 i，i+1，…， n 的结点变为编号为 i+1，i+2，…，n+1 的结点。这里 1≤i≤n+1，而 n 是原表 L 的长度。插入后表 L 的长度加 1。

6) Deletelist（L，i）。删除线性表 L 的第 i 个结点 ，使得原编号为 i+1，i+2，…，n 的结点变成编号为 i，i+1，…，n，n-1 的结点。这里 1≤i≤n，而 n 是原表 L 的长度。删除后表 L 的长度减 1。

2. 线性表的顺序存储结构

(1) 顺序表。

将一个线性表存储到计算机中，可以采用许多不同的方法，其中既简单又自然的是顺序存储方法：即把线性表的结点按逻辑次序依次存放在一组地址连续的存储单元里。用这种方法存储的线性表简称为顺序表（Sequential List）。

设线性表中所有结点的类型是相同的，因此每个结点所占用的存储空间的大小亦是相同的。假设表中每个结点占用 c 个存储单元，其中的第一个单元的存储地址则是该结点的存储地址，并设表中开始结点 a_1 的存储地址（简称为基地址）是 $LOC(a_1)$，那么结点 a_i 的存储地址 $LOC(a_i)$可通过下式计算

$$LOC(a_i)=LOC(a_1)+(i+1)*c \qquad 1\leqslant i\leqslant n$$

由于程序语言中的向量（一维数组）也是采用顺序存储表示，故可用向量这种数组类型

来描述顺序表。又因为除了用向量来存储线性表的元素外，顺序表还可以用一个向量来表示线性表长度属性，所以我们用结构类型来定义顺序表类型，描述如下：

```
# define ListSize 100    //表空间的大小可根据实际需要而定，这里假设为 100。
typedef int DataType;    //DataType 的类型可根据实际情况而定，这里假设为 int。
typedef struct{
    DataType data[ListSize];    //向量 data 用于存放表结点
    int length;    //当前的表长度
}Seqlist;
```

(2) 顺序表上实现的基本运算。

1) 插入。线性表的插入运算是指在表的第 i（1≤i≤n+1）个位置上，插入一个新结点 x，使长度为 n 的线性表

$$(a_1, a_2, \cdots, a_{i-1}, a_i, \cdots, a_n)$$

变成长度为 n+1 的线性表

$$(a_1, a_2, \cdots, a_{i-1}, x, a_i, \cdots, a_n)$$

在顺序表上做插入运算，平均要移动表中的一半结点。平均时间复杂度也是 O(n)。

2) 删除。线性表的删除运算是指将表的第 i 个结点删去，使长度为 n 的线性表

$$(a_1, a_2, \cdots, a_{i-1}, a_i, a_{i+1}\cdots, a_n)$$

变成长度为 n-1 的线性表

$$(a_1, a_2, \cdots, a_{i-1}, a_{i+1}\cdots, a_n)$$

在顺序表上做删除运算，平均要移动表中约一半的结点，平均时间复杂度也是 O(n)。

3. 线性表的链式存储结构

(1) 单链表。

链表正是通过每个结点的链域将线性表的 n 个结点按其逻辑顺序链接在一起的。如果链表的每个结点只有一个链域，将这种链表称为单链表（Single Linked List）。

用 C 语言描述的单链表如下：

```
typedef char DataType;      //假设结点的数据域类型是字符
typedef struct node {       //结点类型定义
    DataType data;          //结点的数据域
    Struct node * next;     //结点的指针域
}ListNode;
typedef ListNode * Linklist;
ListNode * P;
LinkList head;
```

(2) 循环链表。

循环链表（Circular Linked List）是一种首尾相接的链表。其特点是无须增加存储量，仅对表的链接方式稍作改变，即可使得表处理更加方便灵活。

(3) 双链表。

单循环链表中，虽然从任一已知结点出发能找到其直接前趋结点，但时间耗费是 O(n)。希望从表中快速确定一个结点的直接前趋，可以在单链表的每个结点里再增加一个指向其直接前趋的指针域 prior。这样形成的链表中有两条方向不同的链，故称之为双（向）链表（Double

Linked List）。形式描述为：

```
typedef struct dlistnode{   //双链表的结点类型
        DataType data;
        Struct dlisnode * prior, * next;
}DlistNode;
typedef DlistNode * DLinkList;
DLinkList head;
```

4. 顺序表和链表的比较

(1) 基于空间的考虑。

顺序表的存储空间是静态分配的，在程序执行之前必须明确规定它的存储规模。链表的存储空间是动态分配的，只要内存空间尚有空闲，就不会产生溢出。因此，当线性表的长度变化较大，难以估计其存储规模时，以采用动态链表作为存储结构为好。

所谓存储密度（Storage Density）是指结点数据本身所占的存储量和整个结点结构所占的存储量之比，即

存储密度=结点数据本身所占的存储量/结点结构所占的存储总量

一般地，存储密度越大，存储空间的利用率就越高。显然，顺序表的存储密度为 1，而链表的存储密度小于 1。当线性表的长度变化不大，易于事先确定其大小时，为了节约存储空间，宜采用顺序表作为存储结构。

(2) 基于时间的考虑。

顺序表是由向量实现的，它是一种随机存取结构，对表中任一结点都可在 O(1)时间内直接地存取，而链表中的结点，需从头指针起顺着链扫描才能取得。因此，若线性表的操作主要是进行查找，很少做插入和删除操作时，采用顺序表做存储结构为宜。

在链表中的任何位置上进行插入和删除，都只需要修改指针。而在顺序表中进行插入和删除，平均要移动表中的近一半的结点，尤其是当每个结点的信息量较大时，移动结点的时间开销就相当可观。因此，对于频繁进行插入和删除的线性表，宜采用链表做存储结构。若表的插入和删除主要发生在表的首尾两端，则采用尾指针表示的单循环链表为宜。

1.2.3 栈和队列

1. 栈

(1) 栈的定义及基本运算。

栈（Stack）是限制仅在表的一端进行插入和删除运算的线性表，通常称插入、删除的这一端为栈顶（Top），另一端称为栈底（Bottom）。当表中没有元素时称为空栈。

栈又称为后进先出（Last In First Out）的线性表，简称为 LIFO 表。

栈的基本运算有六种：

1) InitStack(S),构造一个空栈 S。

2) StackEmpty(S)，判栈空。若 S 为空栈，则返回 TRUE，否则返回 FALSE。

3) StackFull(S)，判栈满。若 S 为满栈，则返回 TRUE，否则返回 FALSE。该运算只适用于栈的顺序存储结构。

4) Push(S，x)，进栈。若栈 S 不满，则将元素 x 插入 S 的栈顶。

5) Pop(S)，退栈。若栈 S 非空，则将 S 的栈顶元素删去，并返回该元素。

6) StackTop(S)，取栈顶元素。若栈 S 非空，则返回栈顶元素，但不改变栈的状态。

(2) 顺序栈。

栈的顺序存储结构简称为顺序栈，它是运算受限的顺序表。类似于顺序表的定义，顺序栈也是用向量来实现。因为栈底位置是固定不变的，所以可以将栈底位置设置在向量两端的任意一个端点；栈顶位置是随着进栈和退栈操作而变化的，故需用一个整型量 top 来指示当前栈顶位置，通常称 top 为栈顶指针。因此，顺序栈的类型定义只需将顺序表的类型定义中的长度属性改为 top 即可。顺序栈的类型定义如下：

```
# define StackSize 100    //假定预分配的栈空间最多为 100 个元素
typedef char DataType;   //假定栈元素的数据类型为字符
typedef struct {
    DataType data[StackSize];
    int top;
} SeqStack;
```

(3) 链栈。

栈的链式存储结构称为链栈，它是运算受限的单链表，其插入和删除操作仅限制在表头位置上进行。由于只能在链表头进行操作，故链栈没有必要像单链表那样附加头结点。栈顶指针就是链表的头指针。

链栈的类型说明如下：

```
typedef struct stacknode   {
        DataType data;
        struct stacknode * next;
} StackNode;
 typedef struct   {
        StackNode * top;     //栈顶指针
} LinkStack;
```

2．队列

(1) 队列的定义及基本运算。

队列（Queue）也是一种运算受限的线性表。它只允许在表的一端进行插入，而在另一端进行删除。允许删除的一端称为队头（Front），允许插入的一端称为队尾（Rear）。

队列又称作先进先出（First In First Out）的线性表，简称为 FIFO 表。

队列的基本运算有以下六种：

1) InitQueue(Q)，置空队。构造一个空队列 Q。

2) QueueEmpty(Q)，判队空。若队列 Q 为空，则返回真值，否则返回假值。

3) QueueFull(Q)，判队满。若队列 Q 为满，则返回真值，否则返回假值。此操作只适用于队列的顺序存储结构。

4) EnQueue(Q，x)，若队列 Q 非满，则将元素 x 插入 Q 的队尾。此操作简称入队。

5) DeQueue(Q)，若队列 Q 非空，则删去 Q 的队头元素，并返回该元素。此操作简称出队。

6) QueueFront(Q)，若队列 Q 非空，则返回队头元素，但不改变队列 Q 的状态。

(2) 顺序队列。

顺序存储结构的队列称为顺序队列，顺序队列实际上是运算受限的顺序表，和顺序表一样，顺序队列也必须用一个向量空间来存放当前队列中的元素。

和栈类似，顺序队列中亦有上溢和下溢现象。此外，顺序队列中还存在“假上溢”现象。因为在入队和出队操作中，头尾指针只增加不减小，致使被删除元素的空间永远无法重新利用。因此尽管队列中实际的元素个数远远小于向量空间的规模，但也可能由于尾指针已超越向量空间的上界而不能做入队操作。该现象称为假上溢。

为充分利用向量空间，克服上述假上溢现象的方法是将向量空间想象为一个首尾相接的圆环，并称这种向量为循环向量，存储在其中的队列称为循环队列（Circular Queue）。

循环队列的类型定义如下：

```
# define QueueSize 100    //应根据具体情况定义该值
typedef char DataType;    //DataType 的类型依赖于具体应具
typedef Sturet {
      int front;              //头指针，队列非空时指向队头元素
      int rear;               //尾指针，队列非空时指向队尾元素的下一位置
      DataType data[QueueSize]
} CirQueue;
```

(3) 链队列。

链式存储结构的队列简称为链队列，它是限制仅在表头删除和表尾插入的单链表。显然仅在单链表的头指针不便于在表尾做插入操作，为此再增加一个尾指针，指向链表上的最后一个结点。于是，一个链队列由一个头指针和一个尾指针惟一地确定。和顺序队列类似，也将这两个指针封装在一起，将链队列的类型 LinkQueue 定义为一个结构类型：

```
typedef struct queuenode   {   //链队列中的结点类型
       DataType data;
       Struct queuenode * next;
}   QueueNode;
typedef struct   {
        QueueNode * front;      //队头指针
        QueueNode * rear;       //队尾指针
} LinkQueue;
```

3. 栈和队列的应用实例

栈和队列的应用非常之广，只要问题满足后进先出和先进先出原则，均可使用栈和队列作为其数据结构。

(1) 数制转换。

将一个非负的十进制整数 N 转换为另一个等价的基为 B 的 B 进制数的问题，很容易用栈来解决。

(2) 栈和递归。

所谓递归是指：若在一个函数、过程或者数据结构定义的内部又直接（或间接）出现有