计算机及应用专业 专科

全国高等教育自学考试

数据结构导论习题详解

黄明 梁旭 编著

机械工业出版社
CHINA MACHINE PRESS

全国高等教育自学考试

数据结构导论习题详解

（计算机及应用专业　专科）

黄　明　梁　旭　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 专科)考试大纲”以及历年考题编写的。本书分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,并附有答案,以检查读者对考点的掌握程度。

本书适用于准备参加全国自学考试(计算机及应用专业 专科)的考生。

图书在版编目（CIP）数据

数据结构导论习题详解/黄明，梁旭编著. —北京：机械工业出版社，2004.6
（全国高等教育自学考试）
ISBN 7-111-14393-0

Ⅰ. 数… Ⅱ. ①黄…②梁… 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（2004）第038284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：李　妍

北京蓝海印刷有限公司印刷·新华书店北京发行所发行

2004年6月第1版·第1次印刷
787mm×1092mm 1/16·11印张·268千字
0001－5000册
定价：17.00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。

全书共分四部分，即笔试应试指南、笔试题解、模拟试卷及参考答案和附录。书中所选试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，且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 专科）数据结构导论考试大纲”，以及“2002 年下半年全国自学考试数据结构导论试卷及参考答案”。

本书由大连铁道学院黄明、梁旭编写。

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

附录

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 专科）数据结构导论考试大纲涵盖了数据结构概论、线性表、栈、队列和数组、树、图、查找表、文件、排序八部分内容。使用的教材是由全国高等教育自学考试指导委员会组织，陈小平编著的《数据结构导论》，1999 年 11 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，首先应熟练掌握大纲涉及的各章基本概念。

第 1 章为概论。要求：理解数据、数据元素和数据项的概念及其相互关系；理解逻辑结构、基本运算和数据结构的概念、意义和分类；理解存储结构与逻辑结构的关系；理解机内表示和四种基本存储方式；理解算法的概念；了解算法分析的基本概念、时间复杂性及其量级的概念。占分量约为 4 分。

第 2 章为线性表。要求：深刻理解线性结构的定义和特点；理解线性表的概念；熟练掌握顺序表和单链表的组织方法及实现基本运算的算法；了解顺序表与链表的优缺点以及串的概念、运算和存储方法。重点：线性结构的定义和特点；线性表的运算；顺序表和单链表的组织方法和算法设计。占分量约为 13 分。

第 3 章为栈、队列和数组。要求：理解栈和队列的定义、特点及与线性表的异同；熟悉顺序栈和链栈的组织方法，队满、队空的判断条件及其描述。重点：栈和队列的特点；顺序栈和链栈上基本运算的实现和简单算法；链队上基本运算的实现和简单算法设计。难点：循环队的组织，队满、队空的条件及算法设计。占分量约为 15 分。

第 4 章为树。要求：理解树形结构的基本概念和术语；深刻领会二叉树的定义和存储结构，熟悉二叉树的遍历次序并熟练掌握遍历算法；了解树和森林的定义、树的存储结构并掌握树、森林与二叉树之间的相互转换方法。重点：树形结构的概念；二叉树的定义、存储结构和遍历算法。难点：二叉树的遍历算法。占分量约为 14 分。

第 5 章为图。要求：理解图的概念并熟悉有关术语；熟练掌握邻接矩阵表示法和邻接表表示法；深刻理解连通图遍历的基本思想和算法；理解最小生成树的有关概念和算法；了解拓扑排序的概念、步骤和背景。重点：图的存储结构和连通图的遍历。占分量约为 12 分。

第 6 章为查找表。要求：了解集合的基本概念；理解查找表的定义、分类和各类的特点；掌握顺序查找和二分查找的思想和算法；理解二叉排序树的概念和有关运算的实现方法；掌握散列表、散列函数的构造方法，以及处理冲突的方法；掌握散列存储和散列查找的基本思想及有关方法、算法。重点：二分查找；二叉排序树的查找；散列表的查找。占分量约为 12 分。

第 7 章为文件。要求：熟悉文件的基本概念；熟悉顺序文件、索引文件和散列文件的组织方式及操作方式。占分量约为 10 分。

第 8 章为排序。要求：深刻理解各类内部排序方法的指导思想和特点；熟悉各种内部排序算法并理解其基本思想；了解各种内部排序算法的优缺点、时空性能和适用场合。占分量约为 20 分。

在复习时根据大纲里提供的考核知识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。

数据结构导论的考试题型有：单选题、填空题、应用题和设计题等题型。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、推理分析能力，综合比较，评分客观。在答题时，如果有把握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：这种题型常用于考查考生观察能力与运用有关概念、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。

应用题：这种题型灵活性比较大，着重考核考生对概念、知识、原理的掌握和逻辑思维的能力。

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和创见性。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. 数据、数据元素和数据项的概念

(1) 数据的含义

随着计算机科学的发展和计算机应用的普及，计算机加工处理的对象已从早期的数值、布尔值等扩展到字符串、表格、图像甚至语音等。凡能被计算机存储、加工的对象通称为数据。

(2) 数据元素和数据项的概念

1) 数据元素是数据的基本单位，在程序中作为一个整体而加以考虑和处理。换句话说，数据元素被当作运算的基本单位，并且通常具有完整确定的实际意义。

2) 数据项是数据的不可分割的最小标识单位。在很多情况下，数据元素是由若干个数据项组成的。

2. 数据的逻辑结构

(1) 逻辑结构的意义

数据元素之间逻辑关系的整体称为逻辑结构。数据的逻辑结构就是数据的组织形式。一些表面上不同的数据可以有相同的逻辑结构，因此，逻辑结构是数据组织的某种“本质性”的东西。抓住这种本质，不仅有利于分析数据在机外表示的组织形式，更重要的是使程序设计人员可以根据解题需要重新组织数据，即把数据中的所有数据元素按所需的逻辑结构重新加以安排。

(2) 数据元素间逻辑关系的含义

所谓逻辑关系是指数据元素之间的关联方式或称“邻接关系”。例如，下表所示的第一

条档案与第二条档案是邻接的，即相互关联的，因此这两个数据元素之间有逻辑关系。但是，第一条档案与第三条档案不相邻接，即相互不直接关联，因此它们之间没有逻辑关系。

数据示例表

编　号	姓　名	性　别	出生日期	职　务
0001	刘建国	男	19491001	老　师
0002	张　华	女	19682235	助　工
0003	王　军	男	19467778	技术员
0004	王　红	女	19622147	工程师
…	…	…	…	…

(3) 四类基本逻辑结构（集合、线性结构、树形结构和图状结构）的不同特点

1) 集合。集合中任何两个结点之间都没有逻辑关系，组织形式松散。

2) 线性结构。线性结构中结点按逻辑关系依次排列形成一条“锁链”。

3) 树形结构。树形结构具有分支、层次特性，其形态有点像自然界中的树。

4) 图状结构。图状结构最复杂，其中的各个结点按逻辑关系互相缠绕，任何两个结点都可以邻接。

3. 运算的概念

(1) 运算是在逻辑结构层次上对处理功能的抽象

运算是指在任何逻辑结构上施加的操作，即对逻辑结构的加工。这种加工以一个或多个逻辑结构及其他有关参数为对象，以经过修改的逻辑结构或从原逻辑结构中提取的有关信息为结果。根据操作的结果，可将运算分成以下两种基本类型：

1) 加工型运算，其操作改变了原逻辑结构的“值”，如结点个数、某些结点的内容等；

2) 引用型运算，其操作不改变原逻辑结构，只从中提取某些信息作为运算的结果。

(2) 基本运算的含义和作用

一般地，可能存在同一逻辑结构 S 上的两个运算 A 和 B，A 的实现需要或可以利用 B，而 B 的实现不需要利用 A。在这种情况下，称 A 可以“归约”为 B。假如 Γ 是 S 上的一些运算的集合，△是 Γ 的一个子集，使得 Γ 中每一运算都可以“归约”为△中一个或多个运算，而△中任一运算不可“归约”为别的运算，则称△中运算为（相对于 Γ 的）基本运算。相对而言，基本运算比非基本运算简单。在很多情况下，一旦基本运算得到实现（即写出了实现其功能的程序），非基本运算的实现就十分容易。一般地，将较复杂的运算分解为（“归约”为）较简单的运算，有利于降低程序设计的难度，同时也有利于提高程序设计的“效率”。例如多个复杂运算可能都需要利用同一个基本运算，这时，只要写出该基本运算的一个实现，让这些复杂运算分别“调用”该基本运算的实现程序即可。因此，对于任何具体应用问题，可以考虑将其处理要求“整理”为一组运算 Γ，并经适当分解将 Γ 归约为一组基本运算△。

4. 数据结构的概念

(1) 数据结构的含义

作为一个科学概念，数据结构这一术语目前还没有一致公认的定义。比较流行的观点有

两种：一个观点认为，一个数据结构是由一个逻辑结构S、一个定义在S上的基本运算集△和S的一个存储实现D所构成的整体（S，△，D）；另一观点认为，一个数据结构是由一个逻辑结构S和S上的一个基本运算集△构成的整体（S，△）。

（2）数据结构涉及数据表示和数据处理两个方面

数据结构包括逻辑结构和基本运算两部分，逻辑结构是用来完成数据表示的，运算是用来完成数据处理的。所以说数据结构涉及数据表示和数据处理两个方面。

5．存储结构和运算实现

（1）存储结构的含义和四种基本存储方式

存储实现的基本目标是建立数据的机内表示。由于已经建立的逻辑结构是设计人员根据解题需要选定的数据组织形式，因此存储实现建立的机内表示应遵循选定的逻辑结构。另一方面，由于逻辑结构不包括结点内容即数据元素本身的表示，因此存储实现的另一主要内容是建立数据元素的机内表示。按上述思路所建立的数据的机内表示称为数据的存储结构。一般地，一个存储结构包括以下三个主要部分：

1）存储结点（在不致混淆时简称为结点），每个存储结点存放一个数据元素；

2）数据元素之间关联方式的表示，也就是逻辑结构的机内表示；

3）附加设施，如为便于运算实现而设置的“哑结点”等。

通常，存储结点之间可以有四种关联方式，称为四种基本存储方式。

1）顺序存储方式：每个存储结点只含一个数据元素。所有存储结点相继存放在一个连续的存储区里。用存储结点间的位置关系表示数据元素之间的逻辑关系。

2）链式存储方式：每个存储结点不仅含有一个数据元素，还包含一组指针。每个指针指向一个与本结点有逻辑关系的结点，即用附加的指针表示逻辑关系。

3）索引存储方式：每个存储结点只含一个数据元素，所有存储结点连续存放。此外增设一个索引表，索引表中的索引指示各存储结点的存储位置或位置区间端点。

4）散列存储方式：每个结点含一个数据元素，各个结点均匀分布在存储区里，用散列函数指示各结点的存储位置或位置区间端点。

（2）运算实现与运算的联系及区别

一个运算的实现是指一个完成该运算功能的程序。运算只描述处理功能，不包括处理步骤和方法，运算实现的核心是处理步骤的规定，即算法设计。

6．算法及其描述

算法规定了求解给定类型问题所需的所有“处理步骤”及其执行顺序，使得给定类型的任何问题能在有限时间内被机械地求解。

任何算法都必须用某种语言加以描述。根据描述算法的语言的不同，可将算法分为以下三类：

1）运行终止的程序可执行部分。运行终止的程序可执行部分是用程序设计语言描述的算法，这种算法可直接在计算机上运行，从而使给定问题在有限时间内被机械地求解。为了叙述方便，有时又将这种算法称为程序。

2）伪语言算法。采用某种“伪程序设计语言”描述的算法称为伪语言算法。伪语言算法不可直接在计算机上运行（可在某种“抽象机”上运行），但容易编写和阅读。

3）非形式算法。用自然语言（如汉语），同时可能还使用了程序设计语言或伪程序设计

语言描述的算法称为非形式算法。

7. 算法分析

(1) 算法在给定输入下的计算量的含义和估算的方法

算法的计算量是指算法的时间性能（或时间效率）。评价算法的时间性能，并不需要知道计算量的精确值，只要能反映出求解同一问题的不同算法的计算量之间的差异即可。通常采用下述办法来估算求解某类问题的各个算法在给定输入下的计算量：

1) 根据该类问题的特点合理地选择一种或几种操作作为“标准操作”。

2) 确定每个算法在给定输入下共执行了多少次标准操作，并将此次数规定为该算法在给定输入下的计算量。

(2) 最坏情况时间复杂性和平均时间复杂性的概念

1) 以算法在所有输入下的计算量的最大值作为算法的计算量，这种计算量称为算法的最坏情况时间复杂性或最坏情况时间复杂度。

2) 以算法在所有输入下的计算量的加权平均值作为算法的计算量，这种计算量称为算法的平均时间复杂性或平均时间复杂度。

(3) 空间复杂性的含义

空间复杂性的概念和时间复杂性的概念类似，但通常更关心一个算法除输入数据占用存储空间之外所需的附加存储空间的大小。

1.2.2 线性表

1. 线性结构的概念

(1) 线性结构的定义

线性结构是 n（$n \geqslant 0$）个结点的有穷序列。

为了便于讨论，有时将含 n（$n>0$）个结点的线性结构表示成（a_1，a_2，…，a_n），其中每个 a_i 代表一个结点。a_1 称为起始结点，a_n 称为终端结点。i 称为 a_i 在线性表中的序号或位置。对任意一对相邻结点 a_i，a_{i+1}（$1 \leqslant i < n$），a_i 称为 a_{i+1} 的直接前趋，a_{i+1} 称为 a_i 的直接后继。为了满足运算的封闭性，通常允许一种逻辑结构出现不含任何结点的情况。不含任何结点的线性结构记为（）或 Ø。

(2) 线性结构的基本特征

线性结构的基本特征是：若至少含有一个结点，则除起始结点没有直接前趋外，其他结点有且仅有一个直接前趋；除终端结点没有直接后继外，其他结点有且仅有一个直接后继。在线性结构中，这种邻接关系是 1 对 1 的，即每个结点至多只有一个直接前趋并且至多只有一个直接后继。而所有结点按 1 对 1 的邻接关系构成的整体就是线性结构。

2. 线性表的概念

(1) 线性表的逻辑结构

线性表的逻辑结构是线性结构。所含结点的个数称为线性表的长度（简称表长）。表长为 0 的线性表称为空表。

(2) 线性表的基本运算的功能

1) 初始化 INITIATE（L），加工型运算。其作用是建立一个空表 L = Ø（即建立线性表的“表示构架”，但不含任何数据元素）。

2) 求表长 LENGTH (L)，引用型运算。其结果是线性表 L 的长度。

3) 读表元 GET (L, i)，引用型运算。若 1≤i≤LENGTH (L)，其结果是线性表 L 的第 i 个结点；否则，结果为一特殊值。

4) 定位（按值查找）LOCATE (L, X)，引用型运算。若 L 中存在一个或多个值与 X 相等的结点，运算结果为这些结点的序号的最小值；否则，运算结果为 0。

5) 插入 INSERT (L, X, i)，加工型运算。其作用是在线性表 L 的第 i 个位置上增加一个以 X 为值的新结点，使 L 由 $(a_1, \cdots, a_{i-1}, a_i, \cdots, a_n)$ 变为 $(a_1, \cdots, a_{i-1}, X, a_i, \cdots, a_n)$。参数 i 的合法取值范围是 1～n+1（当 n≤LENGTH (L) 时）。

6) 删除 DELETE (L, i)，加工型运算。其作用是撤销线性表 L 的第 i 个结点 a_i，使 L 由 $(a_1, \cdots, a_{i-1}, a_i, a_{i+1}, \cdots, a_n)$ 变为 $(a_1, \cdots, a_{i-1}, a_{i+1}, \cdots, a_n)$。i 的合法取值范围是 1～n。

3. 线性表的顺序存储结构——顺序表

(1) 顺序表表示法的基本思想、特点和类 C 语言的描述

顺序表是线性表的顺序存储结构，即按顺序存储方式构造的线性表的存储结构。按照顺序存储方式，顺序表的一个存储结点存储线性表的一个结点的内容，即数据元素（此外不含其他信息），所有存储结点按相应数据元素间的逻辑关系（即 1 对 1 的邻接关系）决定的次序依次排列。这就是顺序表表示法的基本思想。由此决定了顺序表的特点：逻辑结构中相邻的结点在存储结构中仍相邻。

假定线性表的数据元素的类型为 datatype（在应用中，此类型应根据实际问题中出现的数据元素的特性具体定义），则它的类 C 语言描述如下：

```
const maxsize = 顺序表的容量;
typedef struct
        {datatype data[maxsize];
         int last;
        }sqlist;
sqlist L;
```

数据域 data 是一个一维数组，线性表的第 1，2，…，n 个元素分别存放在此数组的第 0，1，…，last－1 个分量中。数据域 last 表示线性表当前的长度，而 last－1 是线性表的终端结点在顺序表中的位置（因为 C 语言的数组下标从 0 开始）。常数 maxsize 称为顺序表的容量，其值通常根据具体问题的需要取线性表实际可能达到的最大长度。从 last 到 maxsize－1 为顺序表当前的空闲区（或称备用区）。

(2) 区别顺序表的容量与线性表的表长

如上面所述，容量指线性表实际达到的最大长度；表长指当前表的长度，表长小于等于表的容量。

4. 插入、删除和定位运算在顺序表上的实现

(1) 顺序表上实现插入、删除和定位运算的三个算法

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 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 $(a_1, a_2, \cdots, a_{i-1}, X, a_i, \cdots, a_n)$。用顺序表作为线性表的存储结构时，结点的物理顺序必须和结

点的逻辑顺序保持一致。插入算法的基本步骤是：

① 将结点 a_i，…，a_n 各后移一位以便腾出第 i 个位置；

② 将 X 置入该空位；

③ 表长加 1。

步骤②、③非常简单，故只需进一步分析步骤①。这里的关键问题是移动次序。显然只能按 a_n，a_{n-1}，…，a_i 的次序进行，即按从右到左的次序，先将 a_n 右移一位，再将 a_{n-1}右移一位到 a_n 原来的位置上，依次类推，直到将 a_i 右移一位到 a_{i+1}原来的位置上。这样，步骤①可用下述算法的第 3 行实现，第 4、5 行分别是步骤②、③的实现。此外，必须在上述各步之前判断参数 i 即插入位置是否合法，故增加第 1、2 行。完整的插入算法如下：

```
void insert _ sqlist(sqlist L, datatype x, int i)
    /* 将 X 插入到顺序表 L 的第 i-1 个位置 */
    {
第 1 行:  if(L.last == maxsize) error('表满');          /* 溢出 */
第 2 行:  if((i<1) || (i> L.last+1)) error('非法位置');
第 3 行:  for(j = L.last ;j = i ; j --)
          L.data[ j ]= L.data[ j-1 ];                   /* 依次后移 */
第 4 行:  L.data[ i - 1] = X;                           /* 置入 X */
第 5 行:  L.last = L.last + 1;                          /* 修改表长 */
    }
```

2）删除。线性表的删除运算是指将表的第 i（$1 \leqslant i \leqslant n$）个结点删去，使长度为 n 的线性表（$a_1$，$a_2$，…，$a_{i-1}$，$a_i$，$a_{i+1}$，…，$a_n$）变成长度为 n－1 的线性表（$a_1$，$a_2$，…，$a_{i-1}$，$a_{i+1}$，…，$a_n$）。和插入运算类似，在顺序表上实现删除运算也必须移动结点才能够反映出结点间逻辑关系的变化。若 i = n，则只要简单地删除终端结点，无需移动结点；若 $1 \leqslant i \leqslant n-1$，则必须将表中位置 i+1，i+2，…，n 上的结点依次前移到位置 i，i+1，…，n－1 上，以填补删除操作造成的空缺。

删除运算的实现比较简单，其基本步骤是：

① 结点 a_{i+1}，…，a_n 依次前移一位（从而覆盖掉被删结点 a_i）；

② 表长减 1。此外无需考虑溢出，只判断参数 i 是否合法即可。

算法如下：

```
void delete _ sqlist(sqlist L, int i);
    /* 删除顺序表 L 中的 i 个位置上的结点 */
  {
    if ((i < 1) || (i > L.last)) error('非法位置');
    for(j = i + 1; j = L.last; j ++)
        L.data[ j-2 ] = L.data[ j-1 ];  /* 依次前移,注意第一个 L.data[j-2]存放 ai */
        L.last = L.last - 1;            /* 修改表长 */
  }
```

3）定位。定位运算 LOCATE（L，X）的功能是求 L 中值等于 X 的结点序号的最小值，当不存在这种结点时结果为 0。算法如下：

```
int locate_sqlist( sqlist L, datatype X )
    /*在顺序表从前往后查找第一个值等于X的结点。若找到则回传该结点序号;否则回传0 */
    {
     i = 1;                    /* i指示顺序表L中结点序号 */
     while((i≤L.last) && (L.data[i-1] != x))     /* 注意:ai在L.data[i - 1]中 */
           i ++;               /* 从前往后查找 */
     if( i≤L.last ) return(i);
              else return(0);
    }
```

(2) 设计顺序表上的其他简单算法

5. 线性表的链式存储结构——单链表

(1) 单链表表示法的基本思想

单链表表示法的基本思想是用指针表示结点间的逻辑关系。

(2) 单链表的结点形式及其作用

1) 由数据域和指针（链）域两部分组成；

2) 这两部分各自的作用：

数据域用于存储线性表的一个数据元素；指针域（链域）用于存放一个指针，该指针指向本结点所含数据元素的直接后继所在的结点。这样，所有结点通过指针的链接而组织成单链表。

(3) 单链表的类C语言描述

假设数据元素的类型为datatype。单链表的类型定义如下：

```
typedef struct node * pointer;
struct node
        { datatype data;
          pointer next;
        };
typedef pointer lklist;
```

其中，pointer是指向struct node类型变量的指针类型。struct node是结构体类型，规定一个结点是由两个域data和next组成的记录（每个域实际上相当于一个变量），其中data是结点的数据域，next是结点的链域，而next本身又是一个pointer类型的指针型变量。lklist是一个与pointer相同的类型，但名字不同。我们将用lklist来说明头指针变量的类型，因而lklist也就被用来作为单链表的类型。

(4) 头指针和头结点的作用

head称为头指针变量，该变量的值是指向单链表的第一个结点的指针，称为头指针。单链表的每一个结点都被一个指针所指，并且任何结点也只能通过指向它的指针才能引用。因此，对单链表中任一结点的访问必须首先根据头指针变量中存放的头指针找到第一个结点，再按有关各结点链域中存放的指针顺序往下找，直到找到所需的结点。由此可见，头指针变量具有标识单链表的作用。通常在单链表的第一个结点之前增设一个类型相同的结点，称之为头结点，其他结点称为表结点，头结点的作用是便于实现各种运算。

6. 插入、删除和定位运算在单链表上的实现

(1) 实现上述三种运算的三个算法

1) 插入。实现插入运算 INSERT (L, X, i) 的基本步骤如下:

① 在单链表上找到插入位置;

② 生成一个以 X 为值的新结点;

③ 将新结点链入 ("链入"与"插入"的区别在于链入不包括寻找插入位置)。

假定指针 p 指向单链表中的第 i-1 个结点, s 指向已生成的新结点, 链入操作如下: 将结点 *s 的链域指向结点 *p 的后继结点; 将结点 *p 的链域 p→next 改为指向新结点。算法如下:

```
void insert_lklist( lklist head, datatype x, int i)
    /* 在表 head 的第 i 个位置上插入一个以 x 为值的新结点 */
  {
    p = find_lklist( head, i-1 );          /* 先找第 i-1 个结点 */
    if( p == NULL)                         /* 若第 i-1 个结点不存在 */
      error( '不存在第 i 个位置');          /* 退出并给出出错信息 */
    else
      {
       s = malloc(size);s-> data = x;      /* 否则生成新结点 */
       s -> next = p -> next;              /* 结点 *p 的链域值传给结点 *s 的链域 */
       p -> next = s;                      /* 修改 *p 的链域 */
      }
  }
```

注意: 链入操作的两条语句的执行顺序不能颠倒, 否则结点 *p 的链域值即指向原表第 i 个结点的指针将丢失。

2) 删除。实现删除运算 DELETE (L, i) 的基本步骤是:

① 找到第 i 个结点;

② 从单链表上摘除该结点 ("摘除"与"删除"的区别在于前者不包括寻找待删结点)。假定指针 p 指向待删结点的前一个结点, q 指向待删结点, 摘除操作的方法如下: 将 p 所指结点 *p 的链域 p -> next 改为指向待删结点 q 的后继结点, 这一操作可通过将待删结点 *q 的链域 q -> next 的值传给结点 *p 的链域 p -> next 完成: p -> next = q -> next。算法如下:

```
void delete_lklist( lklist head, int i)
    /* 删除表 head 的第 i 个结点 */
  {
    p = find_lklist( head, i -1);                  /* 先找待删结点的直接前趋 */
    if ((p! =NULL) && (p->next! =NULL))            /* 若直接前趋存在且待删结点存在 */
      {
       q = p -> next;                              /* q 指向待删结点 */
       p -> next = q -> next;                      /* 摘除待删结点 */
       free ( q );                                 /* 释放已摘除结点 q */
```