21世纪高等学校计算机规划教材
21st Century University Planned Textbooks of Computer Science

软件技术基础

Software Technology

张选芳 傅茂洛 王欣 主编
李廷元 何元清 袁小珂 朱建刚 宋劲 副主编

- 简明实用
- 实例丰富
- 图文并茂

高校系列

人民邮电出版社
POSTS & TELECOM PRESS

21世纪高等学校计算机规划教材
21st Century University Planned Textbooks of Computer Science

软件技术基础

Software Technology

张选芳 傅茂洛 王欣 主编
李廷元 何元清 袁小珂 朱建刚 宋劲 副主编

高校系列

人民邮电出版社
北京

图书在版编目（CIP）数据

软件技术基础 / 张选芳，傅茂洺，王欣主编. -- 北京 : 人民邮电出版社，2010.9
21世纪高等学校计算机规划教材
ISBN 978-7-115-23316-5

Ⅰ. ①软… Ⅱ. ①张… ②傅… ③王… Ⅲ. ①软件－高等学校－教材 Ⅳ. ①TP31

中国版本图书馆CIP数据核字(2010)第158185号

内 容 提 要

本书按照国家教育部工科计算机基础课程指导委员会提出的“三个层次五门课”的系列课程体系设置的第二层次的一门基础理论课教学大纲编写，系统地介绍了计算机软件技术基础的基本内容，包括数据结构，计算机操作系统，软件工程，数据库技术。本书体系结构和内容选取强调基础性和实用性，符合理工科学生的认知规律，各章配有选择题、填空题和问答题等，供学生练习使用。

本书可作为高等院校理工科非计算机专业学生教材，也可供科技人员及电脑爱好者阅读。

21 世纪高等学校计算机规划教材

软件技术基础

◆ 主　　编　张选芳　傅茂洺　王　欣
副 主 编　李廷元　何元清　袁小珂　朱建刚　宋　劲
责任编辑　刘　博

◆ 人民邮电出版社出版发行　　北京市崇文区夕照寺街 14 号
邮编　100061　　电子函件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
大厂聚鑫印刷有限责任公司印刷

◆ 开本：787×1092　1/16
印张：17.5　　2010 年 9 月第 1 版
字数：455 千字　　2010 年 9 月河北第 1 次印刷

ISBN 978-7-115-23316-5

定价：32.00 元

读者服务热线：(010)67170985　印装质量热线：(010)67129223
反盗版热线：(010)67171154

本书编委会

主　编　张选芳　傅茂洺　王　欣

副主编　李廷元　何元清　袁小珂　朱建刚　宋　劲

委　员（按拼音排序）

陈华英　戴　敏　戴　蓉　高大鹏　华　漫

李尚俊　林瑞春　路　晶　罗银辉　潘　磊

宋海军　徐国标　严　宏　张　欢　张中浩

张建学　张娅岚　郑　涛　钟　晓　周　敏

出版者的话

现今社会对人才的基本要求之一就是应用计算机的能力。在高等学校，培养学生应用计算机的能力，主要是通过计算机课程的体制改革，即计算机教学分层、分类规划与实施；密切联系实际，恰当体现与各专业其他课程配合；教学必须以市场需求为导向，目的是培养高素质创新型人才。

人民邮电出版社经过对教学改革新形势充分的调查研究，依据目前比较成熟的教学大纲，组织国内优秀的有丰富教学经验的教师编写一套体现教学改革最新形势的“高校系列计算机教材”。在本套教材的出版过程中，我社多次召开教材研讨会，广泛听取了一线教师的意见，也邀请众多专家对大纲和书稿做了认真的审读与研讨。本套教材具有以下特点。

1. 覆盖面广，突出教改特色

本套教材主要面向普通高等学校（包括计算机专业和非计算机专业），是在经过大量充分的调研基础上开发的计算机系列教材，涉及计算机教育领域中的所有课程（包括专业核心骨干课程与选修课程），适应了目前经济、社会对计算机教育的新要求、新动向，尤其适合于各专业计算机教学改革的特点特色。

2. 注重整体性、系统性

针对各专业的特点，同一门课程规划了组织结构与内容不同的几本教材，以适应不同教学需求，即分别满足不同层次计算机专业与非计算机专业（如工、理、管、文等）的课程安排。同时本套教材注重整体性的策划，在教材内容的选择上避免重叠与交叉，内容系统完善。学校可根据教学计划从中选择教材的各种组合，使其适合本校的教学特点。

3. 掌握基础知识，侧重培养应用能力

目前社会对人才的需要更侧重于其应用能力。培养应用能力，须具备计算机基础理论、良好的综合素质和实践能力。理论知识作为基础必须掌握，本套教材通过实践教学与实例教学培养解决实际问题的能力和知识综合运用的能力。

4. 教学经验丰富的作者队伍

高等学校在计算机教学和教材改革上已经做了大量的工作，很多教师在计算机教育与科研方面积累了相当多的宝贵经验。本套教材均由有丰富教学经验的教师编写，并将这些宝贵经验渗透到教材中，使教材独具特色。

5. 配套资源完善

所有教材均配有 PPT 电子教案，部分教材配有实践教程、题库、教师手册、学习指南、习题解答、程序源代码、演示软件、素材、图书出版后要更新的内容等，以方便教与学。

我社致力于优秀教材的出版，恳请大家在使用的过程中，将发现的问题与提出的意见反馈给我们，以便再版时修改。

人民邮电出版社

前　言

软件技术基础是按照国家教育部工科计算机基础课程指导委员会提出的“三个层次五门课”的系列课程体系设置的第二层次的一门基础理论课程。通过本课程的学习，学生能对计算机软件设计所需的基本知识和技巧有一个全面的认识，为软件设计开发工作打下坚实的基础。

全书着重介绍了数据结构与算法、软件工程、操作系统和数据库技术方面的基础理论知识。全书分 4 章，第 1 章数据结构，主要讲述算法与数据结构的基本概念及常用的典型数据结构与算法，包括链表、队列、栈、数组等线性数据结构，二叉树、哈夫曼树等树型数据结构和简单的图型数据结构。在算法方面，结合数据结构讲述了查找与排序算法。第 2 章计算机操作系统，主要介绍操作系统的 5 大管理功能：进程管理、存储管理、作业管理、设备管理与文件管理。第 3 章软件工程，介绍软件工程的概念和常用开发模型以及新型软件工程技术。第 4 章数据库技术，主要介绍数据库的基本概念与技术，包括数据库的基础知识、数据库的数据模型、结构化查询语言以及新型数据库技术。

本书内容简明清晰、重点突出、实例丰富、图文并茂，并结合每章内容给出了填空题、选择题、判断题、问答题，以达到通过练习，巩固每章所学知识的目的。

本书由张选芳、李廷元编写第 1 章，傅茂洺、袁小珂、朱建刚和宋劲编写第 2 章，王欣编写第 3 章，何元清编写第 4 章，由张选芳老师统编全稿。本书在编写过程中得到许多老师的热情支持和帮助，在此对他们一并表示诚挚的谢意！

由于作者水平有限，加之时间仓促，书中错误和疏漏之处在所难免，恳请广大读者不吝赐教。

编者

2010 年 6 月

目　录

第1章 数据结构

1.1 数据结构的基本概念

1.1.1 数据结构的研究内容及其重要性

数据结构是组织和访问数据的系统方法。自从1946年第一台计算机ENIAC问世以来，计算机已经在各个领域得到了广泛的应用。用计算机处理问题，首先需要把客观对象抽象为某种形式的数据，然后设计对这些数据进行处理的算法，由计算机执行设计好的算法，最后获得问题的处理结果。著名计算机科学家Niklaus Wirth在其经典著作《数据结构+算法=程序》中，认为程序其实就是数据在特定的表示方式和结构的基础上对抽象算法的具体描述，说明了数据结构的重要性。

在计算机应用的早期，计算机主要应用于科学计算，解决的问题侧重于数值计算，处理的对象相对较为简单，如对整数、实数进行算术运算、逻辑运算等，此时用一些变量或数组等足以表示要解决的问题。但随着计算机从早期的数值计算扩展到非数值计算领域，如管理信息系统、受控热核聚变、人工智能模拟、工业过程实时控制、卫星遥感遥测及气象等领域，数据的处理对象日益复杂和多样化，处理的数据呈海量式增长，有关数据结构的研究内容已经成为编译系统、操作系统、数据库管理系统及其他系统程序和一些应用系统的重要基础。

1968年美国的唐纳德·克努特（Donald E. Kunth）教授出版了其名著《计算机程序设计艺术》第一卷《基本算法》，首次系统地阐述了数据结构的主要内容，即数据的逻辑结构、存储结构及对数据进行操作的各种算法。到20世纪70年代中期和80年代初各种有关数据结构的著作大量问世，我国于20世纪70年代末开始开设数据结构的有关课程，现在数据结构不仅是计算机科学与技术专业的核心课程，同时也是很多与计算机应用有关的专业的一门重要的选修课或必修课，因此掌握数据结构的知识对于进一步进行高效率的计算机程序开发非常重要。

1.1.2 数据结构的基本概念和术语

数据（data）：是信息的载体，是描述客观事物的数、字符，以及所有能输入到计算机中、能被计算机程序识别和处理的符号的集合。数据又可以分为数值型数据和非数值型数据两大类。数值型数据如整数、实数、复数等，一般用于工程和科学计算等；非数值型数据如字符、字符串、文字、图形、语音等。

数据元素（data element）：是数据的基本单位，在计算机程序中通常作为一个整体进行考虑

和处理。

数据项（data item）：是具有独立含义的最小标识单位。有时，一个数据元素可由若干数据项（data item）组成，如整数这个集合中，10 这个数就可以称为是一个数据元素；又比如在一个关系型数据库中，一个记录可称为一个数据元素，而这个元素中的某一字段就是一个数据项。

数据对象（data object）：数据的子集。数据对象是具有相同性质的数据成员（数据元素）的集合，如整数数据对象 N = { 0，1，2，…}，英文字母数据对象 LETTER ={'A'，'B'，…，'Z'}。

数据类型（data type）：是一个值的集合及在这些值上定义的一组操作的总称。

结构（structure）：数据元素相互之间的关系称为结构。有 4 种类型的基本结构。

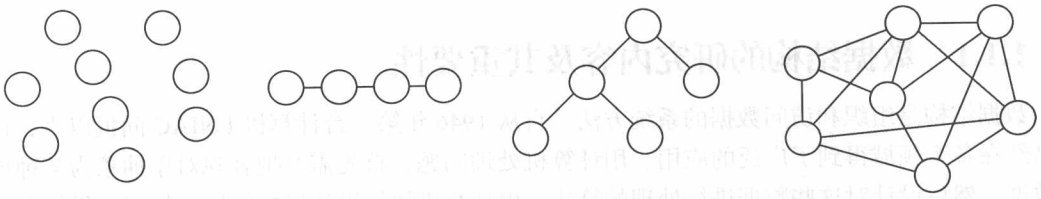
（1）集合：结构中的数据元素之间除了“同属于一个集合”的关系外，别无其他关系。

（2）线性结构：结构中的数据元素之间存在着一个对一个的关系。

（3）树形结构：结构中的数据元素之间存在着一个对多个的关系。

（4）图状结构或网状结构：结构中的数据元素之间存在着多个对多个的关系。

以上 4 种类型的基本结构如图 1-1 所示。



（a）集合　（b）线性结构　（c）树形结构　（d）网状结构

图 1-1　4 种类型的基本结构

数据结构（data structure）：相互之间存在着一种或多种特定关系的数据元素的集合。

数据结构的定义虽然没有统一的标准，但是它包括以下 3 方面的内容：逻辑结构、存储结构和对数据的操作。

为了增加对数据结构的认识，笔者给出一个实例，见表 1-1。

表 1-1　学生成绩表

学　号	姓　名	语　文	数　学	物　理
021001	王强	87	90	96
021002	李一龙	69	91	89
021003	张映月	87	79	71
021004	何一端	84	88	68
…	…	…	…	…

这是一个班级的学生成绩表（也称为成绩数据库），在数据库中称它为一个数据结构，它由很多记录（数据元素）组成，每个元素又包括多个数据列（字段、数据项）。那么这张表的逻辑结构是怎么样的呢？通常分析数据结构都是从结点（其实也就是元素、记录、顶点，虽然在各种情况下所用名称不同，但说的是同一个概念）之间的关系来分析的，对于这个表中的任一个记录（结点），它只有一个直接前趋和一个直接后继（前趋后继就是前相邻后相继的意思），整个表只有一个开始结点和一个终端结点。所有这些就构成了这个表的逻辑结构，即逻辑结构就是数据元素之间的逻辑关系。

数据的逻辑结构通常分两大类：线性结构和非线性结构。

线性结构：线性结构的特征是若结构为非空集，则该结构有且只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。

非线性结构：非线性结构的逻辑特征是该结构中一个数据元素可能有多个直接前趋和直接后继，非线性结构中最普遍的就是图的结构。

数据的存储结构是指数据的逻辑结构在计算机存储空间中的存放形式。在数据的存储结构中，不仅要存放各数据元素的信息，还需要存放各数据元素之间的前后关系的信息。数据的存储结构有时也称为数据的物理结构，它是逻辑结构在计算机存储器里的物理实现。

如表 1-1 所示的数据元素，在计算机中可以采取不同的存储方式。一种方式是将数据元素连续存放在一片内存单元中，另一种方式是将数据元素随机地存放在不同的内存单元中，再用指针将这些数据元素链接在一起。这两种存储方式形成两种不同的存储结构。

常用的数据存储结构有以下 4 种基本形式。

1. 顺序存储

顺序存储把逻辑上相邻的数据元素存储在物理位置上相邻的存储单元里，数据元素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），通常借助程序语言的数组来实现。

顺序存储方法主要应用于线性的数据结构。非线性的数据结构也可通过某种线性化的方法来实现顺序存储。

2. 链式存储

链式存储不要求逻辑上相邻的数据元素在物理位置上也相邻，数据元素间的逻辑关系由附加的指针字段表示。由此得到的存储表示称为链式存储结构（Linked Storage Structure），通常借助于程序语言的指针类型来实现。

3. 索引存储

索引存储通常在存储数据元素信息的同时，还建立附加的索引表。

索引表由若干索引项组成。若每个数据元素在索引表中都有一个索引项，则该索引表称为稠密索引（Dense Index）；若一组数据元素在索引表中只对应一个索引项，则该索引表称为稀疏索引（Spare Index）。索引项的一般形式是：（关键字，地址）。关键字是能唯一标识一个数据元素的数据项。稠密索引中索引项的地址表示数据元素所在的存储位置；稀疏索引中索引项的地址表示一组数据元素的起始存储位置。

4. 散列存储

散列存储的基本思想是根据数据元素的关键字直接计算出该数据元素的存储地址。

以上 4 种基本存储方法既可单独使用，也可组合起来对数据结构进行存储。同一逻辑结构采用不同的存储方法，可以得到不同的存储结构。选择何种存储结构来表示相应的逻辑结构，视具体要求而定，主要考虑运算方便及算法的时间和空间要求。

除了逻辑结构和物理结构外，数据结构的第三个内容是对数据的操作（运算），比如一张表格，用户需要进行查找、增加、修改、删除记录等工作，而怎么样才能进行这样的操作呢？这就是数据的运算，它不仅仅是指加减乘除这些算术运算，在数据结构中，这些运算常常涉及算法问题。

数据结构不仅仅是存储数据，它还包括对数据的操作。这些操作的设计和实现需要使用算法。算法（algorithm）是对特定问题求解步骤的一种描述，它由有限的指令序列组成，可完成某项特定的任务。为了保证正确地处理数据，学习数据结构必然要学习算法。要理解数据结构本身，

重要的是理解实现数据结构操作的算法。算法和数据结构是相辅相成、缺一不可的两个方面。数据结构是算法处理的对象，也是设计算法的基础。一个具体问题的数据在计算机中往往可以采用多种不同的数据结构来表示；一个计算过程的实现又常常有多种可用的算法。因此，选择什么样的算法和数据结构就成为实现程序过程中最重要的一个课题。

一个算法具有 5 个重要的特性。

（1）有穷性：一个算法必须在执行有穷步之后结束。

（2）确定性：算法的每一个步骤，必须是确切地定义的。

（3）输入：一个算法有 0 个或多个输入。

（4）输出：一个算法有 1 个或多个输出。

（5）可行性：一个算法的每一个步骤都应当能有效地执行，并且在执行有限的步骤之后能得到确定的结果。

在使用算法对特定的问题进行求解时，往往涉及对算法的分析。算法分析（algorithm analysis）是指对算法的执行时间和所需内存空间的估算。同一问题的求解往往可以使用不同的算法，通过算法分析，可以比较两个算法的效率高低。对于算法所需时间和空间的估算，一般不需将算法写成程序在实际的计算机上运行。

算法的时间复杂度是指执行一个算法所花费的时间代价。当要解决的问题的规模以某种单位由 1 增至 n 时，对应算法所耗费的时间也以某种单位由 $f(1)$增至 $f(n)$，此时称该算法的时间复杂度为 $f(n)$。

算法的空间复杂度是指执行一个算法所需占用的空间代价。当要解决的问题的规模以某种单位由 1 增至 n 时，对应算法所需占用的空间也以某种单位由 $g(1)$增至 $g(n)$，此时称该算法的空间复杂度为 $g(n)$。

问题的规模是指算法要解决的问题所要处理的数据量的大小。例如，对 n 个记录进行排序，n 就是问题的规模；又如，求 n 阶矩阵的转置矩阵，n 也可以看做是问题的规模。时间单位一般规定为一个简单语句（如赋值语句、条件判断语句等）所用的时间。空间单位一般规定为一个简单变量（如整型、字符型、浮点型等）所占用的存储空间大小。

要全面地分析一个算法，应考虑该算法在最坏情况下的代价、最好情况下的代价、平均情况下的代价。然而，要全面准确地分析每一个算法是相当困难的，一般考虑这个算法在最坏情况下的代价。在多数情况下，只要得到一个估计值就足够了。

在描述算法分析的结果时，通常采用大 O 表示法：说某个算法的时间代价（或空间代价）为 $O(f(n))$，如果存在正的常数 c 和 n_0。当问题的规模 $n \geqslant n_0$ 时，该算法的时间代价（或空间代价）$T(n) \leqslant c \cdot f(n)$。这时称该算法的时间（或空间）代价的增长率为 $f(n)$。这种说法意味着当 n 充分大时，该算法的复杂度不大于 $f(n)$的一个常数倍。

采用大O表示法简化了时间复杂度和空间复杂度的度量，其基本思想是关注复杂性的数量级，而忽略数量级的系数，这样在分析算法的复杂度时，可以忽略零星变量的存储空间和个别语句的执行时间，重点分析算法的主要代价。

常见的时间复杂度，按数量级递增排列依次为：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 $O(2^n)$。

理解了逻辑结构、存储结构和对数据的操作 3 个概念，就可以理解数据结构这个概念。

在不引起混淆的情况下，通常将数据的逻辑结构简称为数据结构，但读者要清楚的是，数据结构这门科学研究的不仅仅是数据的逻辑结构，它还包含对数据的存储结构及对数据的操作的研究。

1.1.3　数据结构、数据类型和抽象数据类型

数据结构用来反映一个数据的内部构成，即一个数据由哪些数据元素构成，以什么方式构成，呈什么结构。数据结构包括逻辑上的数据结构和物理上的数据结构。逻辑上的数据结构反映数据元素之间的逻辑关系，物理上的数据结构反映数据元素在计算机内的存储方式。数据结构是数据存在的形式。

数据是按照数据结构分类的，具有相同数据结构的数据属同一类。同一类数据的全体称为一个数据类型。在高级程序设计语言中，数据类型用来说明一个数据在数据分类中的归属。它是数据的一种属性。这个属性限定了该数据的变化范围。为了解题的需要，根据数据结构的种类，高级语言定义了一系列的数据类型。不同的高级语言所定义的数据类型不完全相同。例如，C++语言所定义的数据类型如图 1-2 所示。

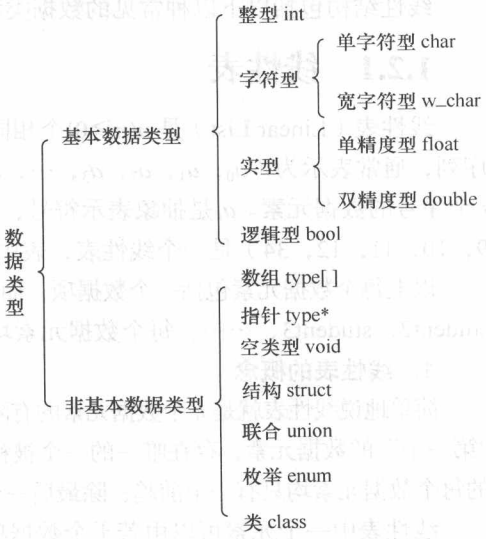


图 1-2　C++的数据类型

其中，基本数据类型对应于简单的数据结构，非基本数据类型对应于复杂的数据结构。在复杂的数据结构里，允许数据元素本身具有复杂的数据结构，因而非基本数据类型允许复合嵌套。指针类型对应于数据结构中数据元素之间的关系，表面上属基本数据类型，实际上都指向复杂的数据元素（即构造数据类型）中的数据，因此这里没有把它划入基本数据类型，而是把它划入非基本数据类型。

数据结构反映数据内部的构成方式，它常常用一个结构图来描述数据中的每一项，数据元素被看做一个结点，并用方框或圆圈表示，数据元素之间的关系用相应的结点之间带箭号的连线表示。如果数据元素本身又有它自身的结构，则结构出现嵌套。在这里嵌套还允许是递归的嵌套。

由于指针数据的引入，使构造各种复杂的数据结构成为可能。按数据结构中的数据元素之间的关系，数据结构有线性与非线性之分。在非线性数据结构中又有层次与网状之分。由于数据类型是按照数据结构划分的，因此，一类数据结构对应着一种数据类型。数据类型按照该类型中的数据所呈现的结构也有线性与非线性之分，层次与网状之分。一个数据变量，在高级语言中的类型说明必须是该变量所具有的数据结构所对应的数据类型。

抽象数据类型（Abstract Data Type，ADT）是带有一些操作的数据元素的集合，它是一种描述用户和数据之间接口的抽象模型，ADT 的主要功能是简单而明确地描述数据结构的操作。此处的“抽象”意味着用户应该从与实现方法无关的角度去研究数据结构。抽象数据类型为用户提供了一种定义数据类型的手段，其关键的要素为数据的结构及在该结构上相应的操作的集合。

引入抽象数据类型的目的是把数据类型的表示和数据类型上运算的实现与这些数据类型和运算在程序中的应用隔开，使它们相互独立。对于抽象数据类型的描述，除了必须描述它的数据结构外，还必须描述定义在它上面的运算（过程或函数）。抽象数据类型上定义的过程和函数以该抽象数据类型的数据所应具有的数据结构为基础。

下面的各节将讨论一些基本抽象数据类型。所谓基本，只是相对而言，这些数据类型是最基本、最简单的，并且是实现其他抽象数据类型的基础。

在后面的内容中将介绍表、栈、队列、串、树、图等常见的基本抽象数据类型的 ADT 操作，

以及这些操作用于不同数据描述方法的具体实现。

1.2 线性结构

线性结构是最简单且最常用的一种数据结构。线性结构的基本特点是数据元素有序并且是有限的。

线性结构包括以下以种常见的数据类型：线性表、栈、队列、数组、串等。

1.2.1 线性表

线性表（Linear List）是 $n(n\geq 0)$个相同类型的元素 a_0，a_1，a_2，a_3，…，a_n所构成的有限线性序列，通常表示为（a_0，a_1，a_2，a_3，…，a_n），其中 n 为线性表的长度。$a_i(0\leq i\leq n)$是线性表中第 i 个序号的数据元素。a_i是抽象表示符号，在不同的情况下含义不同。例如，一个整数序列（7，8，9，10，11，12，34）是一个线性表，表中每一个元素 a_i均为整数，表长为 7。

以上每个数据元素包括一个数据项，而表 1-1 中的学生成绩表数据，对于每一位学生 student1，student2，student3、……，每个数据元素均包括多个数据项。

1. 线性表的概念

简单地说线性表就是 n 个数据元素的有限序列。线性表具有这样一些特点：存在唯一的一个被称为“第一个”的数据元素；存在唯一的一个被称为“最后一个”的数据元素；除第一个之外，线性表中的每个数据元素均只有一个前趋；除最后一个之外，线性表中的每个数据元素均只有一个后继。

线性表中一个元素可以由若干个数据项组成，在这种情况下的数据元素为记录，而含有大量记录的线性表又称为文件。

线性表中元素的个数 n 定义为线性表的长度（$n\geq 0$）。

同一线性表中的元素必定具有相同特性，即属同一数据对象，相邻数据元素之间存在着序偶关系，即数据元素是“一个接一个地排列在一起”：

$$(a_0,\ a_1,\ \cdots,\ a_{i-1},\ a_i,\ a_{i+1},\ \cdots,\ a_n)$$

其中，a_{i-1}为 a_i的直接前驱，a_{i+1}为 a_i的直接后继，a_i是第 i 个元素，称 i 为数据元素 a_i在线性表中的位序。

线性表是相当灵活的一种数据结构，如长度可根据需要增减，元素也可以增删。

线性表的基本操作有以下几种。

（1）Initiate(L){初始化}：设定一个空的线性表。

（2）Length(L){求表长}：对给定的线性表 L，函数返回值为其数据元素的个数。

（3）Get(L,i){取元素}：若给定的数据元素序号 i 满足 $1\leq i\leq$Length(L)，则函数返回值为给定线性表 L 的第 i 个元素 a_i，否则返回空元素。

（4）Locate(L,x){定位}：对给定值 x，若线性表 L 中存在某个数据元素 a_i等于 x，则函数返回索引号最小的 i 的值；若 L 中不存在等于 x 的数据元素，则函数返回一个特殊值（比如−1），以说明不存在的位置。

（5）Insert(L,i,x){插入}：对给定的线性表 L，若索引号 i 满足 $1\leq i\leq$Length(L)+1，则在线性表的第 i 个位置上插入一个新的数据元素 x，使原来表长度为 n 的线性表变为表长为 n+1 的线性表，并使函数返回值为 true，否则函数返回值为 false。

（6）Delete（L,i）{删除}：对给定的线性表，若索引号 i 满足 $1\leq i\leq$Length(L)，则把线性表

中第 i 个元素 a_i 删除，使原来表长为 n 的线性表变成表长为 $n-1$ 的线性表，并使函数返回值为 true，否则函数返回值为 false。

对于线性表的其他有关操作，如线性表的合并、排序等操作，都可以通过以上的基本操作来实现。

2. 线性表的抽象数据类型的定义

```
ADT List{
    数据对象：D={a_i | a_i∈Elemset, i=1,2, …,n, n≥0}
    数据关系：R1={<a_i-1, a_i> | a_i-1, a_i∈D,i=2, …,n}
    基本操作：
    InitList(&L)
    操作结果：构造一个空的线性表 L
    DestroyList(&L)
    初始条件：线性表已存在
    操作结果：销毁线性表 L
    ClearList(&L)
    初始条件：线性表已存在
    操作结果：置线性表 L 为空表
    ListEmpty(L)
    初始条件：线性表已存在
    操作结果：若线性表 L 为空表，则返回 TURE，否则返回 FALSE
    ListLenght(L)
    初始条件：线性表已存在
    操作结果：返回线性表 L 数据元素个数
    GetElem(L,i,&e)
    初始条件：线性表已存在（1≤i≤ListLenght(L)）
    操作结果：用 e 返回线性表 L 中第 i 个数据元素的值
    locatElem(L,e,compare())
    初始条件：线性表已存在，compare()是数据元素判定函数
    操作结果：返回线性表 L 中第 1 个与 e 满足关系 compare()的数据元素的位序
    PriorElem(L,cur_e,&pre_e)
    初始条件：线性表已存在
    操作结果：若 cur_e 是线性表 L 的数据元素，且不是第一个，则用 pre_e 返回它的前驱，否则操作失败，
pre_e 无定义
    NextElem(L,cur_e,&)
    初始条件：线性表已存在
    操作结果：若 cur_e 是线性表 L 的数据元素，且不是第最后一个，则用 next_e 返回它的后继，否则操作失
败，next_e 无定义
    ListInsert(&L,i,e)
    初始条件：线性表已存在（1≤i≤ListLenght(L)+1）
    操作结果：在线性表 L 中第 i 个数据元素之前插入新元素 e，L 长度加 1
    ListDelete(&L,i,&e)
    初始条件：线性表已存在（1≤i≤ListLenght(L)）
    操作结果：删除线性表 L 中第 i 个数据元素，用 e 返回其值，L 长度减 1
    ListTraverse(L,visit())
    初始条件：线性表已存在
    操作结果：依次对线性表 L 的每个数据元素调用 visit()函数，一旦 visit()失败，则操作失败
}ADT List
```

上述是线性表抽象数据类型的定义，其中只是一些基本操作，还可以更复杂。例如，将两个线性表合并等。

3. 线性表的顺序存储结构及其实现——顺序表

线性表的存储结构有多种类型，如顺序存储结构和链式存储结构，因此线性表可以采用使用顺序存储结构的顺序表和有序顺序表来实现，也可以采用使用链式存储结构的单链表、双链表和循环链表等来实现。

（1）顺序表的定义。

① 顺序存储结构：把线性表的数据元素按逻辑次序依次存放在一组地址连续的存储单元里的方法。

② 顺序表（Sequential List）：用顺序存储结构实现的线性表简称为顺序表（Sequential List）。按数据元素的值无序或有序存放，顺序表又可以分为无序顺序表和有序顺序表两种。

（2）数据元素的存储地址。

不失一般性，设线性表中所有数据元素的类型相同，则每个数据元素所占用的存储空间大小也相同。当把线性表的数据元素存入存储空间后，称这样的存储空间为一个“结点”。假设表中每个结点占用 c 个存储单元，并设表中第一个结点 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：

$$\text{Loc}(a_i) = \text{Loc}(a_1) + (i-1) * c \quad (1 \leqslant i \leqslant n)$$

在顺序表中，每个结点 a_i 的存储地址是该结点在表中的位置 i 的线性函数。只要知道基地址和每个结点的大小，就可在相同时间内求出任一结点的存储地址，因此顺序表是一种随机存取存储结构。

（3）顺序表类型定义。

```
#define ListSize 100          //表空间的大小可根据实际需要而定，这里假设为 100
struct SeqList {
   int data[ListSize];        //数组 data 用于存放表结点，结点中元素的类型假设为 int
   int length;                //当前的表长度
};
```

① 除了用数组这种顺序存储的类型存储线性表的元素外，顺序表还应该用一个变量来表示线性表的长度属性，因此用结构体类型来定义顺序表类型。

② 存放线性表结点的数组空间的大小 ListSize 应仔细选值，使其既能满足表结点的数目动态增加的需求，又不至于定义过大而浪费存储空间。

③ 由于 C 语言中数组的下标从 0 开始，所以若 L 是 SeqList 类型的顺序表，则线性表的开始结点 a_1 和终端结点 a_n 分别存储在 L.data[0]和 L.Data[L.length−1]中。

④ 若 L 是 SeqList 类型的指针变量，则 a_1 和 a_n 分别存储在 L->data[0]和 L->data[L->length−1]中。

（4）顺序表的特点。

顺序表是用顺序存储结构实现的线性表，其具有以下优点：

① 存储方式简单。几乎所有的程序设计语言都支持数组，因此用一维数组表示顺序表是最简单的实现方式。数组的下标可以看做数据元素的相对地址，因此在顺序表中逻辑上相邻的数据元素，其物理位置亦相邻。

② 顺序表便于随机访问，访问效率高。顺序表第 i 个元素的地址可以表示为：

$$\text{Loc}(a_i) = \text{Loc}(a_1) + (i-1) * c \quad (1 \leqslant i \leqslant n)$$

因此，只要知道顺序表的首地址和每个数据元素所占存储单元的大小就可以求出第 i 个数据元素的地址。

但顺序表也具有以下一些缺点：

① 扩充不方便。顺序表一般采用数组实现，而定义数组时必须指明大小，该大小一旦确定，在程序运行过程中一般不允许修改，因此对于表中数据元素个数需要增加的情况，存储空间不易扩充。

② 插入和删除操作不方便。由于顺序表一般采用数组实现，因此在顺序表中插入或删除某个数据元素时，需要移动数组中的元素，从而占用较大的存储空间和较多的运行时间，致使插入和删除效率低。

（5）顺序表上实现的基本运算。

① 表的初始化。

```
void InitList(struct SeqList *L)
{  //顺序表的初始化即将表的长度置为 0
   L->length=0;
}
```

② 求表长。

```
int ListLength(struct SeqList *L)
{  //求表长只需返回 L->length
   return L->length;
}
```

③ 取表中第 i 个结点。

```
int GetNode(struct SeqList *L, i)
{  //取表中第 i 个结点只需返回 L->data[i-1]即可
   if (i<1||i> L->length-1)
   Error("position error");
   return L->data[i-1];
}
```

④ 插入。

a. 插入运算的逻辑描述。

线性表的插入运算是指在表的第 i（$1 \leqslant i \leqslant n+1$）个位置上，插入一个新结点 x，使长度为 n 的线性表（a_1，…，a_{i-1}，a_i，…，a_n）变成长度为 $n+1$ 的线性表（a_1，…，a_{i-1}，x，a_i，…，a_n）。

由于向量空间大小在声明时确定，当 L->length≥ListSize 时，表空间已满，不可再做插入操作；当插入位置 i 的值为 $i>n$ 或 $i<1$ 时为非法位置，不可做正常插入操作。

b. 顺序表插入操作过程。

在顺序表中，结点的物理顺序必须和结点的逻辑顺序保持一致，因此必须将表中位置为 n，$n-1$，…，i 上的结点，依次后移到位置 $n+1$，n，…，$i+1$ 上，空出第 i 个位置，然后在该位置上插入新结点 x。仅当插入位置 $i=n+1$ 时，才不需移动结点，直接将 x 插入表的末尾。

c. 具体算法描述。

```
void InsertList(struct SeqList *L, int x, int i)
{  //将新结点 x 插入 L 所指的顺序表的第 i 个结点的位置上
   int j;
   if (i<1||i>L->length+1)
```