21世纪高等教育计算机规划教材

"十三五"江苏省高等学校重点教材

COMPUTER

数据结构（C语言）

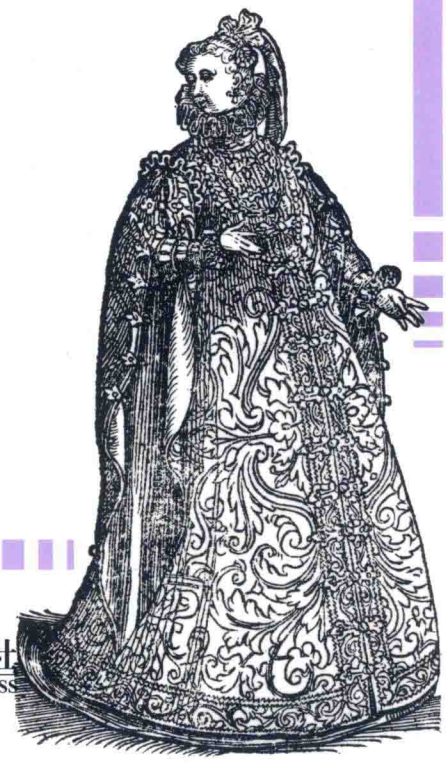
Data Structures in C

王海艳 主编

骆健 朱洁 邹志强 戴华 徐鹤 王甦 副主编

微课版

- 现代信息技术与教育教学紧密融合
- 突破传统教学模式，通过微课形式全面阐述知识点
- 一套完整的立体化教学资源



中国工信出版集团

人民邮电出版社
POSTS & TELECOM PRESS

“十三五”江苏省高等学校重点教材
（编号：2016-2-032）

21世纪高等教育

COMPUTER

数据结构（C语言）

Data Structures in C

王海艳 主编
骆健 朱洁 邹志强 戴华 徐鹤 王甦 副主编

人 民 邮 电 出 版 社
北 京

图书在版编目（CIP）数据

数据结构 : C语言 / 王海艳主编. -- 北京 : 人民邮电出版社, 2017.7
21世纪高等教育计算机规划教材
ISBN 978-7-115-45825-4

Ⅰ. ①数… Ⅱ. ①王… Ⅲ. ①数据结构－高等学校－教材②C语言－程序设计－高等学校－教材 Ⅳ. ①TP311.12②TP312.8

中国版本图书馆CIP数据核字(2017)第181424号

内 容 提 要

本书将现代信息技术与教学紧密融合，突破了传统教学模式，通过微课的形式全面阐述数据结构课程中的重点、难点，涵盖线性表、树、图等内容，形成一套完整的包含知识点、习题、实验、微课视频等的立体化教学资源，帮助学生进行自主式和研究性学习，同时为教师的传统课堂教学提供辅助。

本书系统地讲解了数据结构的相关知识。全书共有 10 章，系统地讲述了线性表、堆栈、队列、数组、树、查找、图、排序等内容，还安排了习题和实验。本书重视算法及其实践性，书中算法都有完整的 C 语言程序，程序代码注释详细。为了让读者能够及时地检验学习效果、把握学习进度，每章都附有丰富的习题。

本书可作为计算机、电子信息、管理信息系统、电子商务、教育技术等相关专业数据结构课程的本科教材，也可以作为计算机软件及应用的工程技术人员参考资料。

◆ 主　　编　王海艳
副 主 编　骆健　朱洁　邹志强　戴华　徐鹤　王甦
责任编辑　李　召
责任印制　陈　犇

◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路 11 号
邮编　100164　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京市艺辉印刷有限公司印刷

◆ 开本：787×1092　1/16
印张：13.25　　2017 年 7 月第 1 版
字数：353 千字　　2017 年 7 月北京第 1 次印刷

定价：38.00 元

读者服务热线：(010)81055256　印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京东工商广登字 20170147 号

前 言

“数据结构”是设计系统软件及大型应用软件的重要基础，对于构建高效的算法起着决定性的作用。“数据结构”课程作为计算机专业的重要核心课程，对培养学生的计算思维和系统分析与设计、算法设计与分析、程序设计与实现等学科基本能力起着至关重要的作用。本书将现代信息技术与传统教育教学方法紧密融合，通过微课的形式阐述数据结构课程中的重点、难点，突破传统教学模式的束缚，建立立体化的教学资源，形成新的知识传授模式。本书在促进教学理念更新，推动高等教育教学方式方法和学习方式的创新方面做了重要尝试。本书旨在使教学与学习过程朝着教学方式混合化、教学资源开放化、学生学习个性化、学习过程社会化方向转变，对于帮助学生进行自主式和研究性学习具有重要意义。

本书以计算机类和信息类学科人才为培养目标，以教学认知规律为编写依据，以“理论、实践、理论与实践相结合”为编写原则。全书共分 10 章，第 1 章介绍数据结构课程的研究内容和关键问题；第 2 章至第 4 章介绍线性表、堆栈和队列、数组和字符串等线性结构的基本概念及常用算法；第 5 章至第 9 章介绍非线性结构的树、散列表、图等数据结构以及相关算法和应用；第 10 章主要讨论排序的各种实现方法及其综合分析比较。附录部分为上机实验内容，指导学生按软件工程的方法设计与编写程序。

本书条理清楚，内容翔实，用词达意，深入浅出，并且配有大量的实例和图示。本书把数据结构的基本概念和常用算法的设计、应用与程序紧密结合，书中算法都有完整的 C 语言程序，程序代码注释详细。本书提供了丰富的习题，并在各章中对重点难点内容配备了微课视频，使读者更具体、更深刻地理解各种常用的数据结构及它们与算法之间的关系，以达到学以致用的目的。

本书的参考学时为 48 ~ 64 学时，以 56 学时为例给出各章的参考教学课时，课时分配如下表。

章　节	课 程 内 容	课 时 分 配	
		讲授	实践训练
第 1 章	绪论	2	
第 2 章	线性表	4	2
第 3 章	堆栈和队列	4	
第 4 章	数组和字符串	4	
第 5 章	树和二叉树	10	2
第 6 章	集合和搜索	2	
第 7 章	搜索树	4	
第 8 章	跳表和散列表	2	

续表

章　节	课程内容	课时分配	
		讲授	实践训练
第 9 章	图	10	2
第 10 章	排序	6	2
课时总计		48	8

本书可作为计算机类专业或信息类相关专业的本科教材，也可以作为报考相关专业硕士研究生入学考试的复习用书，还可以供从事计算机类工程与应用工作的科技工作者学习参考。

本书由王海艳主编，并编写第 1、6 章，骆健编写第 2、9 章，朱洁编写第 4、10 章，邹志强编写第 7、8 章，戴华编写第 3 章，徐鹤编写第 5 章，王甦编写附录综合实验。此外，本书的编写得到南京邮电大学教务处及计算机学院的支持和帮助，在此对学校的支持和同事特别是《数据结构》课程组的各位前辈、各位课程组成员的鼓励表示衷心的感谢。

由于编者水平和经验有限，书中难免有欠妥和错误之处，恳请读者批评指正。

编　者

2017 年 7 月

目 录

第 1 章 绪论

本章首先介绍数据结构的基本概念及相关术语；然后讨论数据结构、数据类型和抽象数据类型之间的关系，介绍数据结构描述方法；最后阐述算法分析的方法。

1.1 数据结构起源

“数据结构”的概念起源于 1968 年美国计算机科学家唐纳德·克努特（Donald Ervin Knuth）教授所著的《计算机程序设计艺术》（The Art of Computer Programming），如图 1.1 所示。在该书的第一卷《基本算法》中，他开创了数据结构的最初体系，较系统地阐述了数据的逻辑结构和存储结构及其操作。

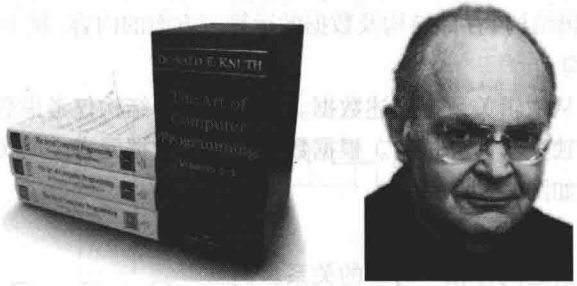


图 1.1 Donald Ervin Knuth 及其著作《The Art of Computer Programming》

在计算机科学中，研究数据结构对设计出高性能的算法和高性能软件至关重要。“数据结构”课程不仅是程序设计的基础，而且是设计和实现编译程序、操作系统、数据库系统及其他应用程序的重要基础。

1.2 基本概念和术语

1.2.1 基本概念

1. 数据

数据是可被计算机识别并加工处理的对象。数据不仅包括整型、实型等数值数据，还包括声

音、图像、视频等非数值数据。例如，MP3 格式的文件是常见的音频声音数据，BMP 格式的文件是典型的图像数据，RMVB 格式的文件是视频数据。

2. 数据元素

数据元素是由数据组成的具有一定意义的基本单位，在计算机中通常作为一个整体来处理。有些情况下，数据元素也称为元素、记录。

3. 数据项

数据项是组成数据元素的、不可分割的最小单位。

表 1.1 为学生信息表，其中每个学生的信息可看作一个数据元素，它由学号、姓名、性别、籍贯等**数据项**组成。

表 1.1　　学生信息表

学　号	姓　名	性　别	籍贯
B16040101	丁小雨	女	江苏省南京市
B16040102	张萌	女	山东省烟台市
B16040103	李强	男	福建省福州市
B16040104	王健	男	广东省汕头市
…	…	…	…

1.2.2 数据结构

数据结构（Data Structure）是由某一数据对象及该对象中所有数据元素之间的关系组成的。数据结构包括数据的逻辑结构、存储结构及数据的运算三方面的内容。接下来从这三方面分别阐述。

1. 数据的逻辑结构

数据的逻辑结构是从逻辑关系上描述数据。数据的逻辑结构仅考虑数据之间的内在关系，是面向应用问题的，它是独立于计算机的。根据数据结构中数据元素之间关系的不同特征，可划分为四种基本逻辑结构，如图 1.2 所示。

（1）线性结构

线性结构中数据元素之间存在一对一的关系。例如，表 1.1 所示的学生信息表可看成一个线性结构，学生信息按照学号依次排列。

（2）树形结构

树形结构中数据元素之间存在一对多的关系。例如，一所学校有多个学院，一个学院有多个专业，构成树形结构。

（3）图结构

图结构中数据元素之间存在多对多的关系。例如，微信朋友圈中，“我”的朋友们彼此之间可能是相识关系，也可能是不相识关系，他们之间存在多对多的朋友关系，从而构成图结构。

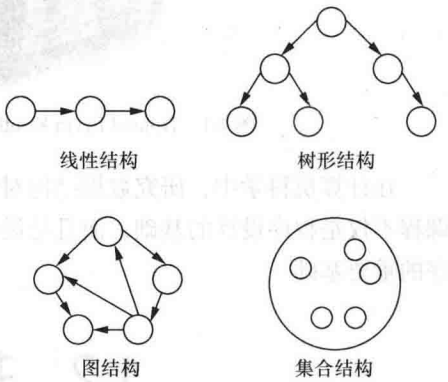


图 1.2　四种基本的逻辑结构示例

（4）集合结构

数据元素之间除了“属于同一个集合”的联系之外没有其他关系。例如，学生存在于某个班

级中，若不考虑学生之间的其他关系，则可视班级为一个集合结构。

以上四种基本的逻辑结构还可进一步分成两类：**线性结构**和**非线性结构**。除了线性结构以外，树形、图和集合结构可统一归入非线性结构一类。

2. 数据的存储结构

数据的**存储结构**是数据及数据之间的关系在计算机内的表示形式。它是面向计算机的，是数据的逻辑结构在计算机存储中的影像。数据的存储结构中，最常见的两种基本存储结构分别是顺序存储结构和链式存储结构。

（1）顺序存储结构

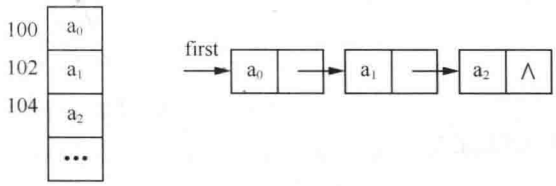
顺序存储结构是将逻辑上相关的数据元素依次存储在地址连续的存储空间中。这种存储结构是借助数据元素在存储空间中的相对位置来表示它们之间的逻辑关系。假定有一线性结构的数据（a_0,a_1,a_2），每个元素占 2 个存储单元，连续存储空间的起始地址是 100，则其顺序存储表示如图 1.3（a）所示。

顺序表示方法并不仅限于存储线性结构的数据。例如，树形结构的数据对象有时也可采用顺序存储的方法表示。这将在以后章节中详细阐述。

（2）链式存储结构

链式存储结构中，数据元素可以存储在任意的存储空间中，可以是连续的存储空间，也可以是不连续的存储空间。在这种情况下，数据元素的存储位置并不能体现它们之间的逻辑关系，需要用指针域存储逻辑上相关的数据元素的地址。因此，为了存储一个元素，需要存放数据元素本身和与该元素逻辑上相关的其他元素的地址，这两部分信息组成一个**结点**。

假定有一线性结构的数据（a_0,a_1,a_2），其链式存储表示如图 1.3（b）所示。其中，每个结点存储了数据元素和该元素逻辑上的后继结点的地址。注意：一个结点的存储地址通常是指存放该结点的存储块的起始存储单元地址。



（a）顺序存储结构　　（b）链式存储结构

图 1.3　两种基本的存储表示方法

在此约定，在不会引起混淆的场合，本书将不明确区分结点和元素这两个术语。但必要时，将包括数据元素和地址在内的整个存储块称为**结点**，而将其中的数据元素称为该结点的**元素**。

3. 数据的运算

如果说数据的逻辑结构描述了数据的静态特性，那么在数据的逻辑结构上定义的一组运算给出了数据被使用的方式，即数据的动态特性。使用数据结构上定义的运算，用户可对数据结构的实例或组成实例的数据元素实施相应的操作。运算的结果可使数据改变状态。

数据结构最常见的运算有：

（1）搜索运算——在数据结构中搜索满足一定条件的元素；

（2）插入运算——在数据结构中插入新元素；

（3）删除运算——将数据结构中指定元素删除；

（4）更新运算——将数据结构中指定元素更新为新的元素。

1.3 抽象数据类型

1. 数据类型

数据类型是指性质相同的值的集合以及定义在该**值集**上的运算集合。C 语言常用的基本数据类型有整型、字符型、指针类型等。数据类型规定了数据的取值范围和允许执行的运算。例如，若在 C 语言中声明 int a,b，则可以给变量 a 和 b 赋值 0，但不可以赋值 2.5，这超出了 int 型变量的取值范围。a 和 b 之间可以执行加法运算，但不可以执行求交集运算，这也超出了 int 型变量所允许的运算范围。

2. 抽象数据类型

了解一个数据类型的对象（变量、常量）在计算机内的表示是有一定用处的，但如果每个数据使用者都要考虑基本数据类型的实现细节，这将给数据使用者增加一项繁杂的工作，并且使用者一旦随意改变数据存储表示，也会滋生不可预知的错误。目前普遍认为对数据类型进行抽象，对使用者隐藏一个数据类型的实现是一个好的设计策略，由此产生了抽象数据类型。

抽象数据类型（Abstract Data Type，ADT）是一个数学模型以及在其上定义的运算集合。其最主要的两个特征是**数据封装**和**信息隐蔽**。**数据封装**是指把数据和操纵数据的运算组合在一起的机制。**信息隐蔽**是指数据的使用者只需知道这些运算的定义（也称规范）便可访问数据，而无须了解数据的存储以及运算算法的实现细节。通过实行数据封装和信息隐蔽，可使数据的使用和实现相分离。

本书中，抽象数据类型的定义格式如下：

```
ADT 抽象数据类型名
{
数据：
    数据元素及其之间关系的定义

运算：
    运算 1(参数表)：运算功能描述
     ......
    运算 n(参数表)：运算功能描述
}
```

3. 数据结构与抽象数据类型

本书将一种数据结构视为一个抽象数据类型，从规范和实现两方面来讨论数据结构。规范是对数据结构中数据元素及其关系、运算给出定义，即逻辑结构和运算的定义组成了数据结构的**规范**。规范指明了一个数据结构可以“做什么”。数据结构的使用者通过规范中的说明使用一个数据结构，不必了解具体的实现细节。数据的存储表示和运算算法的描述构成数据结构的**实现**，它解决了“怎样做”的问题。

1.4 算法和算法分析

1.4.1 算法

算法是计算机科学中的基本概念，是对特定问题的求解步骤。算法须具有下列五个特征。

（1）输入：一个算法有 0 个或多个输入。

（2）输出：一个算法产生一个或多个输出，作为算法运算的结果。

（3）可行性：算法的每一个步骤都可以通过已经实现的基本运算来实现。

（4）确定性：算法的每一个步骤都必须有确切的含义，不会产生二义性。

（5）有穷性：算法必须能在执行有穷步之后终止。

算法的描述方式多种多样，可以用自然语言、流程图、程序设计语言或伪代码来描述。为了方便读者理解算法和上机验证算法，本书采用自然语言描述算法思想，并使用 C 语言描述算法的实现。

衡量一个算法的优劣，主要有以下几个基本标准。

（1）正确性

在合理的数据输入下，算法能够在有限的时间内产生满足预先规定的功能和性能要求。

（2）可读性

一个好的算法应当思路清晰、简单明了。可读性高的算法便于人们阅读、理解和交流；晦涩难懂的算法容易隐藏错误，不易被发现和调试。

（3）健壮性

一个好的算法应在输入不合法数据时，能做出适当处理，而不至于产生异常或是出现崩溃等严重后果。

（4）高效性

评价一个算法的效率主要包括时间和空间两方面。好的算法应具备执行效率高和占用存储空间少的特点。时间复杂度和空间复杂度是衡量算法效率的两个重要指标。

1.4.2 算法的时间复杂度

算法执行时间需通过依据该算法编制的程序在计算机上运行时所消耗的时间来度量。算法的时间复杂度一般是指程序运行从开始到结束所需的时间。而度量一个程序的执行时间通常有两种方法：事后统计法和事前估算法。事后统计法是将算法实现后计算其时间和空间开销，从而确定算法的效率。然而，时间和空间开销的计算与计算机软硬件环境相关，同一个算法在不同的机器上执行所花的时间也不一样，这种方法存在明显的缺陷，因此，不予采纳。本书采用事前估算法评估算法效率。

抛开与计算机软硬件相关的因素，影响算法时间效率最主要的因素是**问题规模**。问题规模通常是指算法的输入量，一般用整数 n 表示。例如，采用相同的排序算法对 10 个元素进行排序与对 100 000 个元素进行排序所需的时间显然是不同的。

一个算法时间花销与算法中语句的执行次数成正比例。算法中语句执行次数多，它的时间花销就多。一个算法中的语句执行次数称为**语句频度**。

一般情况下，算法中基本运算执行次数用 T(n)表示，若有问题规模 n 的某个函数 f(n)，使存在自然数 n_0，正常数 c，当 n 大于等于 n_0 时，$T(n) \leqslant cf(n)$，则称 f(n)是 T(n)的渐近上界。记为

$$T(n)=O(f(n))$$

大 O 记号表示算法的一种**渐近时间复杂度**。渐近时间复杂度也常简称为时间复杂度。大 O 记号用以表达一个算法运行时间的上界，估计算法的执行时间的数量级。

下面举例说明算法的渐近时间复杂度的求解。

程序 1.1　简单求和程序

```
int i=50;              //执行 1 次
int j=200;             //执行 1 次
int sum=0;             //执行 1 次
sum=i+j;               //执行 1 次
printf("%d", sum);     //执行 1 次
```

程序 1.1 语句执行次数为 5，算法的渐近时间复杂度为 O(1)，属于常数级。

程序 1.2　累加求和程序

```
int i;   //执行 1 次
int sum=0;                  //执行 1 次
for (i=0; i<n; i++ )        //执行 n+1 次
{
   sum=sum+1;               //执行 n 次
}
printf("%d",sum);           //执行 1 次
```

程序 1.2 语句执行次数为 2n+4，算法的渐近时间复杂度 T(n)=O(n)。

程序 1.3　矩阵求和程序

```
int i;                                  //执行 1 次
int j;                                  //执行 1 次
int n=100;                              //执行 1 次
for (i=0; i<n; i++ )                    //执行 n+1 次
 {
   for (j=0; j<n; j++ )                 //执行 n(n+1)次
     {
       c[i][j]=a[i][j]+b[i][j];         //执行 n² 次
     }
 }
```

程序 1.3 语句执行次数为 $3+n+1+n(n+1)+ n^2= 2n^2+2n+4$，算法的渐近时间复杂度 $T(n)=O(n^2)$。

一般情况下，可以通过考察一个程序中的关键操作的执行次数来计算算法的渐近时间复杂度。所谓关键操作是对算法执行时间贡献最大的操作。例如，程序 1.2 中，语句 sum=sum+1 可被认为是关键操作，其执行次数为 n，由此计算可得算法的渐近时间复杂度也是 O(n)。

常见的渐近时间复杂度从小到大依次是 $O(1) < O(\log_2 n) < O(n) < O(n\log_2 n) < O(n^2) < O(n^3)$ 。

1.4.3　最坏、最好和平均情况时间复杂度

算法的时间复杂度不仅与问题规模相关，如果输入数据不同，算法所需的时间开销也会不同。例如，在包含 n 个元素的数组中找给定元素 x，设算法从左向右搜索，如果待搜索的元素 x 正好是第一个元素，则所需的查找时间最短，这就是算法的**最好情况**。如果待搜索的元素 x 是最后一

个元素或是不在数组中，则是算法的**最坏情况**。如果需要多次在数组中查找元素，并且假定以相等的概率查找每个数组元素，则是算法时间代价的**平均情况**。

对于算法的时间复杂度分析，存在三种情况，对应三种时间复杂度，即最好情况、最坏情况和平均情况时间复杂度。相关示例将在后续章节中给出。

1.4.4 算法的空间复杂度

算法的**空间复杂度**往往是指对应的程序从运行开始到结束所需的存储量。

程序运行所需的存储空间包括两部分。

（1）**固定部分**。这部分空间与问题规模无关，主要包括程序代码、常量、简单变量等所占的空间。

（2）**可变部分**。这部分空间大小与问题规模有关。例如，长度为 1 000 的两个数组相加，与长度为 10 的两个数组相加，所需的存储空间是不同的。这部分存储空间除了包括数据元素所占的空间外，还包括算法执行所需的额外空间，如递归栈所用的空间。

算法的空间复杂度的讨论类似于时间复杂度，但空间复杂度一般按最坏情况来分析。

1.5 微课（一）

算法的时间复杂度是本章的重点和难点。这部分内容已制作成微课视频以供学习。

扫描以下二维码，可观看数据结构绪论的相关微课视频。

算法和算法分析

习 题

一、基础题

1. 对包含 n 个元素的数组进行顺序搜索时，若搜索每个元素的概率相同，则平均搜索长度为_____。

A. n/2　　B. n

C. (n−1)/2　　D. (n+1)/2

2. 下面说法正确的是_____。

A. 健壮的算法不会因非法的输入数据而出现莫名其妙的状态

B. 算法的优劣与算法描述语言无关，但与所用计算机环境因素有关

C. 数据的逻辑结构依赖于数据的存储结构

D. 以上几个都是错误的

3. 从逻辑上可以把数据结构分为_____两大类。

A. 初等结构、构造型结构　　B. 顺序结构、链式结构

C. 线性结构、非线性结构　　D. 动态结构、静态结构

4. 数据采用链式存储时，存储单元的地址_____。

A. 一定连续　　B. 一定不连续

C. 不一定连续　　D. 部分连续，部分不连续

5. 算法的时间复杂度取决于_____。

A. 问题规模　　B. 计算机的软硬件配置

C. 两者都是　　D. 两者都不是

6. 下面的程序段的时间复杂度为_____。

```
for (i=0; i<n; i++ )
    for (j=0; j<n; j++ )
      x=x+1;
```

A. O(2n)　　B. O(n)

C. $O(n^2)$　　D. $O(\log_2 n)$

二、扩展题

1. 简述下列概念：数据、数据元素、数据项。
2. 什么是数据结构？
3. 简述逻辑结构的四种基本关系。
4. 最常见的存储结构有哪两种？
5. 算法有哪些特征？
6. 算法与程序的区别与联系是什么？
7. 简述衡量算法优劣的基本标准。
8. 对于下列程序段，分析带下划线语句的执行次数，并给出它们的时间复杂度。

（1）
```
i=1; k=0,n=100;
  do
   {
     k=k+10*i; i++;
   } while(i<=n)
```

（2）
```
i=1; x=0;
 do
 {
  x++; i=3*i;
 } while( i<n);
```

（3）
```
for(i=0;i<n;i++)
     for( j=0;j<n;j++)
       a[i][j]=0;
```

（4）
```
y=0;
while(n>=y*y)
y++;
```

第2章 线性表

线性表是最基础、最常用的一种线性数据结构。线性表被广泛应用于信息存储与管理、网络、通信等诸多领域。本章将给出线性表的定义和抽象数据类型描述，讨论线性表的逻辑结构、存储结构及相关运算，并以一元整系数多项式的算术运算为实例介绍线性表的简单应用。

2.1 线性表定义

线性表是零个或多个数据元素构成的线性序列，记为$(a_0, a_1, \cdots, a_{n-1})$。线性表中的数据元素个数 n 称为线性表的长度。当 n=0 时，此线性表为空表。

线性表$(a_0, \cdots, a_{i-1}, a_i, a_{i+1}, \cdots, a_{n-1})$中，$a_i$ 表示下标为 i 的元素，a_{i-1} 是 a_i 的直接前驱元素，a_{i+1} 是 a_i 的直接后继元素。线性表除第一个数据元素 a_0 没有直接前驱元素，最后一个数据元素 a_{n-1} 没有直接后继元素之外，其他数据元素都有唯一一个直接前驱元素和直接后继元素。在不会引起混淆的情况下，我们简称直接前驱元素为“前驱”，直接后继元素为“后继”。线性表中数据元素之间存在着一对一关系，因此，线性表的逻辑结构为线性结构。

线性表是一种非常灵活的数据结构，可在线性表的任意位置执行插入、删除元素的运算，也可执行搜索、修改等运算。本章采用第 1 章给出的抽象数据类型格式对线性表进行描述，其中包括线性表最常见的运算，如 ADT 2.1 所示。

ADT 2.1 线性表 ADT

```
ADT List{
```

数据：

零个或多个数据元素构成的线性序列$(a_0, a_1, \cdots, a_{n-1})$。数据元素之间的关系是一对一关系。

运算：

Init(L)：初始化运算。构造一个空的线性表 L，若初始化成功，则返回 OK，否则返回 ERROR。

Destroy(L)：撤销运算。判断线性表 L 是否存在，若已存在，则撤销线性表 L；否则，返回 ERROR。

IsEmpty(L)：判空运算。判断线性表 L 是否为空，若为空，则返回 OK；否则返回 ERROR。

Length(L)：求长度运算。若线性表 L 已存在，返回线性表 L 的元素个数；否则返回 ERROR。

Find(L,i,x)：查找运算。若线性表 L 已存在且 $0 \leqslant i \leqslant n-1$，则查找线性表 L 中元素 a_i 的值并通过 x 返回；否则，返回 ERROR。

Insert(L,i, x)：插入运算。若线性表 L 已存在且 $-1 \leqslant i \leqslant n-1$，则在元素 a_i 之后插入新元素 x，插入成功后返回 OK，否则返回 ERROR。

Delete(L,i)：删除运算。若线性表 L 非空且 $0 \leqslant i \leqslant n-1$，则删除元素 a_i，删除成功后返回 OK，否则返回 ERROR。