清华大学计算机系列教材

数据结构题集

（第二版）

严蔚敏 吴伟民

清华大学出版社

数据结构题集

（第二版）

严蔚敏
吴伟民 编著

清华大学出版社

(京)新登字 158 号

内 容 简 介

本题集和清华大学出版社出版的教科书《数据结构》(第二版)相配套,该教材获第二届全国高校优秀教材国家级特等奖,电子部电子类专业优秀教材特等奖。本题集主要内容有:习题与学习指导,实习题和部分习题的提示或答案等三大部分,书末有一个附录("数据结构算法(DSDEMO)演示"使用手册)。

其中习题篇的内容和教科书相对应,也分为 12 章,每一章大致由基本内容、学习要点、算法演示内容及基础知识题、算法设计题和算法练习的规格说明等 6 部分组成。实习题分成 6 组,每一组都有鲜明的主题,围绕 1 至 2 种数据结构安排 4 至 9 个题,每个题都有明确的练习目的和要求,在每一组中还都给出一个实习报告的范例,以供读者参考。

本书内容丰富,程序设计观点新颖,在内容的详尽程度上接近于课程辅导材料,不仅可作为大专院校的配套教材,也是广大工程技术人员和自学读者的颇有帮助的辅助教材。

图书在版编目(CIP)数据

数据结构题集/严蔚敏,吴伟民编著. —2 版. —北京:清华大学出版社,1998
ISBN 7-302-03238-6

Ⅰ. 数… Ⅱ. ①严… ②吴… Ⅲ. 数据结构-习题-高等学校-教学参考资料
Ⅳ. TP311.12-44

中国版本图书馆 CIP 数据核字(98)第 36153 号

出 版 者:清华大学出版社(北京清华大学学研楼,邮编 100084)
http://www.tup.tsinghua.edu.cn
责任编辑:贾仲良
印 刷 者:北京国马印刷厂
发 行 者:新华书店总店北京发行所
开 本:787×1092 1/16 印张:16.5 字数:408 千字
版 次:1998 年 12 月第 2 版 2000 年 3 月第 5 次印刷
书 号:ISBN 7-302-03238-6/TP·1733
印 数:473001~513000
定 价:15.80 元

前　言

“数据结构”是计算机学科的基础理论知识，也是软件设计的技术基础，主要研究信息在计算机中的组织和表示方法。“数据结构”课程的教学要求之一是训练学生进行复杂程序设计的技能和培养良好程序设计的习惯，其重要程度决不亚于知识传授。因此在数据结构的整个教学过程中，习题作业和上机实习是两个至关重要的环节。为了帮助读者学好这门课程，我们编写了这本指导性题集。

这本题集与清华大学出版社，1992 年出版的《数据结构》(第二版)一书是配套教材，习题和实习都是按教科书的内容顺序编排的，很多习题涉及教科书上的内容或算法，因此最好读者手边能有这本教科书，以便随时查阅。

习题作业的作用在于帮助学生深入理解教材内容，巩固基本概念，达到培养良好程序设计习惯的目的。其中部分题目可作为检查对授课内容理解和掌握程度的手段，大部分题是作为掌握算法设计技能的基本训练，还有少量题用以提高程序设计能力。本题集的第一篇含有全部 400 余个习题，组织成 12 章，分别与教科书中各章一一对应，在每一章之前给出这一章的内容提要和学习要求。这些习题是在编者多年教学过程中所积累资料的基础上，参考大量国外教材之后精心设计而成的。书中对特别推荐的题作了标记，并对每道习题的难度按五级划分法给出了难度系数，仅供参考。

涉及算法的习题侧重于局部程序设计，即如何编好“小程序”(Programming-in-the-small)。但仅有这方面的训练还是不够的。本题集的第二部分分别以线性表、栈和队列、串、数组和广义表、树和图以及查找和排序为核心设置了 6 组上机实习题，每组有 4 至 9 个题可供读者自由选择。希望这些实习题能对习题起到良好的补充作用，使读者受到涉及程序系统设计的完整过程的综合训练，体会系统结构设计(Programming-in-the-large)与“小程序”设计的区别，培养合作能力，因此，可以说是为将来进行软件开发和研究工作的一次“实践演习”。

在本书的第三篇给出了部分习题的提示或解答。对于多数有唯一确定解的题给出答案，而对算法题则有选择性地作了示范解答或提示。但算法的解答都不是唯一的，我们的解答也不一定是完美的。希望答案或提示能起到抛砖引玉的作用，愿读者开发出更多更好的解法。热忱欢迎读者将这些好的算法寄给我们，在此预先表示感谢。然而，在此我们仍想特别强调的是，本题集主要是为配合高等院校的教学而编写的，因此，为了培养学生独立思考和解决问题的能力，我们诚恳希望不要出版或编印本题集的更详尽的解答，以免干扰学校正常的教学和本书的训练意图，敬请谅解。

本题集的一个特点是强调规范化在算法设计基本训练中的重要地位。在题集的习题篇中给出了算法书写规范，在实习题篇中给出了实习步骤和实习报告的规范。教学经验表明，严格实施这些虽然比较繁琐的规范，对于学生基本程序设计素养的培养和软件工作者工作作风的训练能起到显著的促进作用。

“算法动态模拟演示系统”是为学习并掌握数据结构中各类典型算法而开发的一个辅助

教学软件，它的主要特点是让计算机执行算法，同时增加透明度，令算法的每一步在机内执行的情况都显示在读者面前，就此获得单从文字说明中无法获得的直观知识。为便于读者参考，在习题篇的每一章中列举了与该章相关的算法清单，并在附录中提供该软件完整的使用说明。

本题集的旧版曾在计算机和计算机应用专业的本科生教学中得到广泛使用，效果良好。这本新版题集在力求反映程序设计和软件工程新思想方面作了一些探索，如：模块化抽象和信息隐蔽、软件生命周期等。然而，书中不可避免地存在着谬误和有争议之处，编者诚恳地欢迎广大读者提出批评意见和建议，在此谨向热情的读者致以衷心感谢。

严蔚敏　清华大学计算机技术与科学系
吴伟民　广东工业大学计算机一系

1998 年 7 月

目　录

第一篇　习题与学习指导

第0章　本篇提要与作业规范

1. 本篇提要

本篇内容是按照作者编著的教科书《数据结构》(第二版)的内容和课程教学要求组织的。各章均由基本内容、学习要点、算法演示内容(第1章和第12章除外)及基础知识题、算法设计题和算法设计题的规格说明(第1、8、11和12章除外)等6部分组成。其中：

“基本内容”列举了该章的内容提要，提醒读者把握该章主要内容；

“学习要点”指出了该章的教学重点和难点，以供读者在组织教学或自学时选择习题作参考；

“算法演示内容”提供了“数据结构算法演示DSDEMO”软件中包含的与该章相关的算法清单，通过观察算法执行过程的演示，将有助于深刻理解算法的本质和提高教学效果(在第三篇中提供有DSDEMO软件的使用手册)。

数据结构的练习题大致可分为“基础知识题”和“算法设计题”两类。

“基础知识题”主要供读者进行复习自测之用，目的是帮助读者深化理解教科书的内容，澄清基本概念、理解和掌握数据结构中分析问题的基本方法和算法要点，为完成算法设计题做准备。

“算法设计题”则侧重于基本程序设计技能的训练，相对于上机实习题而言，这类编程习题属于偏重于编写功能单一的“小程序”的基础训练，然而，它是进行复杂程序设计的基础，是本课程习题作业的主体和重点。

各章的题量根据教学内容的多少和重要程度而定，几乎对教科书的每一小节都安排了对应的习题。但对每个读者来说，不必去解全部习题，而只需根据自己的情况从中选择若干题求解即可。为了表明题目的难易程度，便于读者选择，我们在每个题的题号之后注了一个难度系数，难度级别从①至⑤逐渐加深，其区别大致如下：难度系数为①和②的习题以基础知识题为主；难度系数为③的习题以程序设计基础训练为主要目的，如强化对“指针”的基本操作的训练等；习题中也收纳了不少难题，其难度系数设为④和⑤，解答这些题可以激起学习潜力较大的读者的兴趣，对广泛开拓思路很有益。但习题的难度系数也只是一个相对量，读者的水平将随学习的进展而不断提高，因此没有必要去比较不同章节的习题的难度系数，此外，该难度系数值的假设是以读者没有参照习题的解答或提示为前提的。

“循序渐进”是最基本的学习原则。学习者不应该片面追求难题。对于解难度系数为i的习题不太费力的读者，应试试难度系数为i+1的习题，但不要把太多的时间浪费在难度系数为i+2的习题上。“少而精”和“举一反三”是实践证明行之有效的。解答习题应注重于“精”，而不要求“多”。为此，编者在一些自认为值得向读者推荐的“好题”题号前加注了标记◆。把握住这些“关键点”，就把握住了数据结构习题乃至数据结构课程的总脉络。

对算法设计习题的基本要求是使用本章提供的类 **Pascal** 语言和算法书写规范写出书面作业的算法。如能上机实现自己设计的算法，将更有助于程序设计能力的提高。但为了验证“算法”正确与否，往往需要编写更多的程序，它将掩盖这类算法设计题的训练目的。为此，作者正在开发一个可进行算法设计练习的软件 **DSEXMO**，该软件为读者提供一个完成算法作业的集成环境，它可以对读者编写的过程或函数进行自动测试和判断，但读者必须按照该软件事先已定义好的 **Turbo Pascal** 过程或函数首部的要求，并用 **Turbo Pascal** 语言按照事先写好的算法编出程序。本篇中每一章的最后一部分“算法设计题的规格说明”提供了每道算法设计题的 **Turbo Pascal** 过程或函数首部、算法功能和形式参数的说明，以及各算法设计题涉及的数据结构的类型说明。

需要强调的是“算法的可读性”。算法是为了让人来读的，而不是供机器读的。初学者总是容易忽视这一点。算法的真正意图主要在于提供一种在程序设计者之间交流解决问题方法的手段。因此，可读性具有头等的重要性。不可读的算法是没有用的，由它得到的程序极容易产生很多隐藏很深的错误，且难以调试正确。一般地说，宁要一个可读性好、逻辑清晰简单、但篇幅较长的算法，也不要篇幅较小但晦涩难懂的算法。算法的正确性力求在设计算法的过程中得到保证，然而一开始做不到这一点也没有多大的关系，可以逐步做到。

算法设计的正确方法是：首先理解问题，明确给定的条件和要求解决的问题；然后按照自顶向下，逐步求精，分而治之的策略逐一地解决子问题；最后严格按照和使用本章后面提供的算法书写规范和类 **Pascal** 语言完成算法的最后版本。

按照规范书写算法是一个值得高度重视的问题。在基础训练中就贯彻这一规范，不但能够有助于写出“好程序”，避免形成一系列难以纠正且遗害无穷的程序设计坏习惯，而且能够培养软件工作者应有的严谨的科学工作作风。

2. 类 Pascal 语言语法概要

类 **Pascal** 语言是为了方便算法描述而定义的一个语言。实际上，它是对 **Pascal** 语言的一个简单扩充，且与标准 **Pascal** 语言兼容。这就是说，任何一个用标准 **Pascal** 语言写的过程或函数，都合乎类 **Pascal** 语言的语法。一个好的算法描述语言应该在保留程序设计语言的精华(**Pascal** 语言集中地体现了这些精华)的同时，比通常的高级语言抽象一些，以便算法描述能够回避细节，更加明显简洁地转达算法思想。扩充部分就是为了实现这一目的而设计的。

为了算法简明起见，类型和变量说明没有作为算法的一个必不可少的部分，只要算法书写者和阅读者都不会引起误解就可以略去，当然也可在算法说明中给出。这个语言并未实现，但将用它书写的算法转化为一个标准 **Pascal** 程序是一件轻而易举的事情。因此，如何实现它并不重要。

为了便于读者将它同标准 **Pascal** 语言比较，这里只给出扩充部分的语法图。对于其中的任何一个语法单位，如“语句”，只要标准 **Pascal** 语法图中有同名语法单位，读者就应将这相应的两个图“并联”起来看，从而可以得到完整的类 **Pascal** 语法图。

扩充的保留字、标识符和标点符号不再单独列出，读者可以很容易地从扩充部分的语法图中发现。可以直接使用的函数共 4 个：下取整“ ⌊ ⌋ ”，上取整“⌈ ⌉”，求两整数的较大者的值 max(a,b)以及求两整数较小者的值 min(a,b)。其中前两个已嵌在语法中了。

下面的例子直观地介绍了扩充的语法部分的一些典型用法及其语义解释。

(1) COR 与 CAND

OR 与 **COR** 的差别在于：前者不限制两操作数的求值顺序，且满足交换率，而后者限定先对第一操作数求值，如果为真，则表达式的值为真，不再对第二操作数求值。仅当第一操作数的值为假时才对第二操作数求值，并将其值作为表达式的值。

AND 与 **CAND** 的情形完全类似，下面对两种常用的形式给出等价定义。b1,b2 为布尔量。

```
①    IF b1 CAND b2 THEN S; {S 是某个语句}
  ≡  IF b1 THEN IF b2 THEN S;
②    WHILE b1 CAND b2 DO S;
  ≡  endloop :=false; {endloop 是一个新引入的布尔量}
     WHILE b1 AND NOT endloop DO
       IF b2 THEN S ELSE endloop :=true;
```

CAND 用于 **WHILE** 语句的循环条件之中是最常用的形式。在下面的例子中，**CAND** 有效地防止了数组下标越界：

```
{在数组 a[1..max]的前 n 个元素中查找值为 x 的元素}
i :=n;  WHILE (i<>0) CAND (a[i]<>x) DO i :=i-1;
```

(2) 赋值操作

·“串联”赋值：

```
  i :=j :=k :=0;
≡ i :=0;  j :=0;  k :=0;
```

·成组赋值：

```
①  (i, j, k) :=(i-j, j-k, k-i);
  ≡i :=i-j;  j :=j-k;  k :=k-i;   {不是同时赋值}
②  a[1..n] :=a[2..n+1]; {执行时间计为 n 次}
  ≡FOR i :=1 TO n DO a[i] :=a[i+1];
   {  注意，若写 a[2..n+1] :=a[1..n],
            将执行 a[2] :=a[1];  a[3] :=a[2]; …; a[n+1] :=a[n];
            往往非程序员本意。后移一个元素应像下例这样写：
            (a[n+1], …,a[2]) :=(a[n], …, a[1])  }
③  c[0..1,0..1] :=((1,2), (3,4));  {矩阵整体赋值}
```

≡c[0,0] :=1; c[0,1] :=2; c[1,0] :=3; c[1,1] :=4;

④ { 设 r : **RECORD** s : **RECORD** p : integer; q : boolean **END**; x : real **END** }

r :=((i−j, true), 3.1416);

≡r.s.p :=i−j; r.s.q :=true; r.x :=3.1416;

(3) 条件表达式

① a[4..6]≠a[6..8] {执行时间计为 3 次比较}

≡(a[4]≠a[6]) **OR** (a[5]≠a[7]) **OR** (a[6]≠a[8])

② a[4..6]≤a[6..8]≠0

≡a[4..6]中任一元素均小于 a[6..8]中的任一元素且

(a[6]≠0) **AND** (a[7]≠0) **AND** (A[8]≠0)

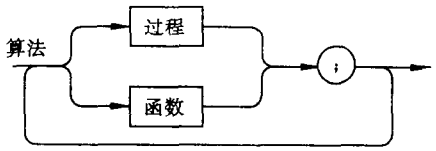
上面(2)中赋值号两边的对象都可以比较。

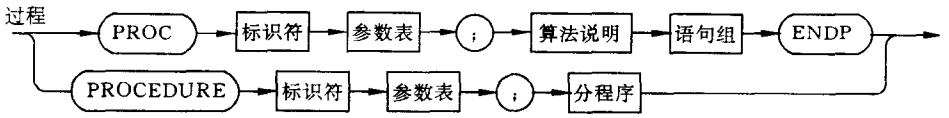
(4) 变量值交换

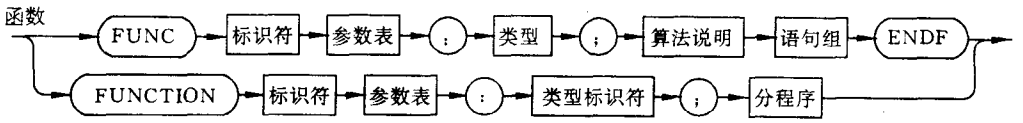
a[i]←→a[j];

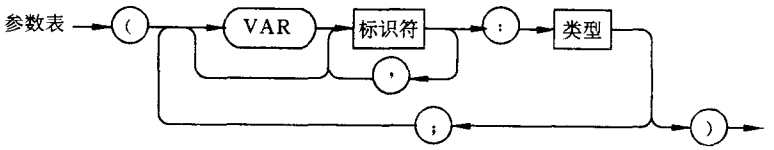
≡temp :=a[i]; a[i] :=a[j]; a[j] :=temp;

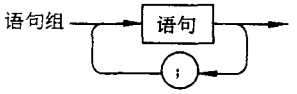
类 **Pascal** 语言扩充部分语法图如下：

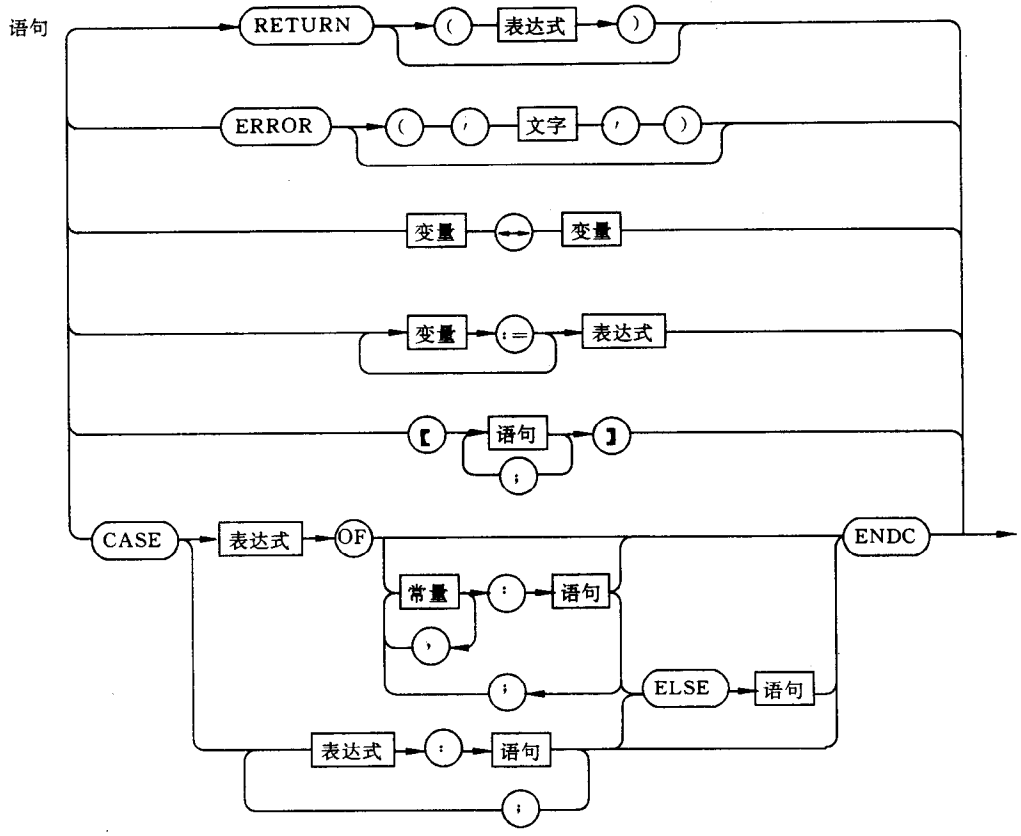


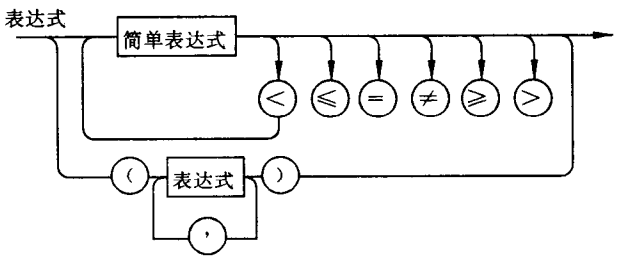


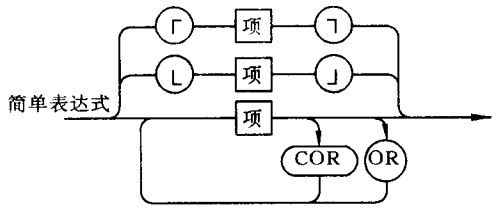


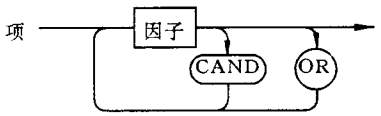


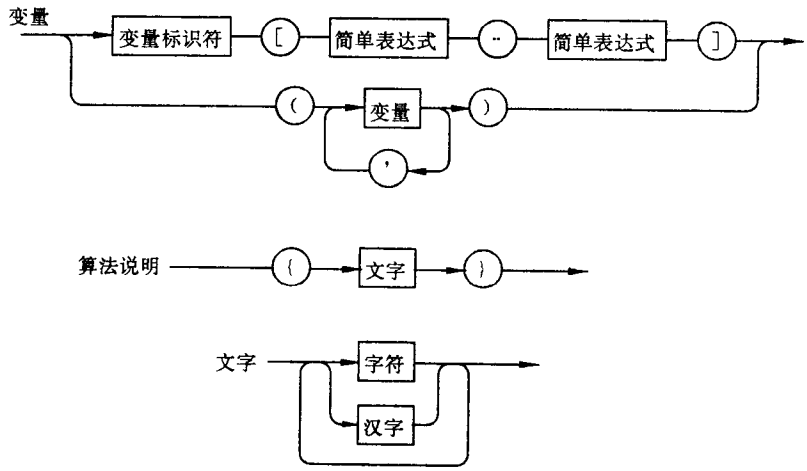












3. 算法书写规范

(1) 算法说明

算法说明,也称为(算法)规格说明,是一个完整算法不可缺少的部分,应该在算法头(即过程或函数首部)之下以注释的形式写明如下内容:指明算法的功能;参数表中各参量的含意和输入输出属性;算法中引用了哪些全局变量或外部定义的变量,它们的作用、入口初值、以及应满足哪些限制条件,例如链表是否带头结点、表中元素是否有序、按递增还是递减方式有序等。必要时,算法说明还可用来陈述算法思想、采用的存储结构等等。递归算法的说明特别重要,读者应该力求将它写为算法的严格定义。

算法说明应该在开始设计算法时就写明,可以在算法设计过程中作一些补充和修改,但是切忌最后补写。对于递归算法的情况,这一点尤其重要。这样做也是递归算法设计的正确而有效的途径,在算法设计(即解决一个问题)的过程中,能否利用自身的处理能力来解决所划分出的一个或几个子问题,全凭检查自身的规格说明而定。书写(递归)算法的规格说明时,应该忽略它如何实现或者假定它能够实现。如何实现的问题正是接下去要做的事。

算法说明书写得不好或不完全时,往往失去了评判一个算法正确与否的标准。书写恰当而又简洁的算法说明是一项具有很强技巧性的活动,通常要经过不断的练习才能达到。在本节的末尾将列出一些规格说明的例子。

(2) 子表表示法

为了使算法说明、注释和断言能够简洁明确地书写,这里规定了一套子表表示法,但仅限于本课程的范围内使用。下述的子表表示形式中有些已被纳入了类 **Pascal** 语言,成为符合语法的变量,其余的只能在"{…}"中出现。

设有一个数组 a: **ARRAY**[0..m-1]**OF** ElemType;以及一个由指针变量 h 指向其头

结点的单链表。变量 i 和 j 作为数组 a 的下标变量；指针变量 p 和 q 指向上述单链表中的两个结点。

· 当 $-1 \leqslant i \leqslant m$ 且 $-1 \leqslant j \leqslant m$ 时，

(a) 若 $i > j+1$，则 a[i..j]是非法的表示；

(b) 若 $i = j+1$，则 a[i..j]表示空子表部分；

(c) 若 $0 \leqslant i < j+1 \leqslant m-1$，则 a[i..j]表示数组 a 的一个非空子表部分。

① 当 a 作为栈使用时，设 i 为栈顶指针，则可以表达为：a[1..i]为栈，$0 \leqslant i \leqslant m-1$。

② 当 a 作为队列使用时，设 i 为队首元素下标，j 为队尾元素下标，则可以表达为：a[i..j]为队列，$1 \leqslant i \leqslant m$，$0 \leqslant j \leqslant m-1$。

③ 当 a 作为循环队列使用时，设 i 为队首元素下标，j 为队尾元素下标，则可以表达为：a[i·(m)·j]为循环队列，$0 \leqslant i \leqslant m-1$，$0 \leqslant j \leqslant m-1$。其中 m 是循环队列的模数，它可以是一个具体的正整数。与此相适应，引入模加和模减算符，定义如下：(同样，m 可以换为常数)

$A \oplus_m B \equiv (A+B)$ **MOD** m

$A \ominus_m B \equiv ((A+m)-B)$ **MOD** m

这两个算符没有纳入类 **Pascal** 语言，因为它们不是单一的符号。读者不仅可以在注释中使用它们，而且可以(并鼓励)在算法逐步求精过程中的中间版本中尽可能使用。

· 若 p≠**NIL**≠q，

(a) 若 q↑.next≠**NIL** 且 q↑.next↑在 p↑之前，则(p↑..q↑)非法；

(b) 若 q↑.next=p，则(p↑..q↑)为一个空子表；

(c) 若 p≠h 且 p↑不在 q↑之后，则(p↑..q↑)表示由 h 指其头结点的链表中的一个非空子表部分。

(d) (p↑…)表示由 p 所指结点起到单链表表尾元素结点上的子表部分。

(e) (…q↑)表示 h 指其头结点的链表中 q 所指结点之前(包括 q↑)的链表部分。

(3) 注释与断言

在难懂的语句和关键的语句(段)之后加以注释可以大大提高程序的可读性。注释要恰当，并非越多越好；此外，注释句的抽象程度应略高于语句(段)，例如，应避免用“{i 增加 1}”来注释语句“i :=i+1;”。

断言是注释的一种特殊写法，是一类特别重要的注释。它是一个逻辑谓词，陈述算法执行到这点时应满足的条件。多写断言式的注释，甚至以断言引导算法段的设计，是提高算法的结构良好性、避免错误和增强可读性的有效手段，是特别值得提倡的。其中最重要的是算法的入口断言和 **ELSE** 分支断言。注意，正确的算法也只能在输入参数值合法的前提下得出正确的结果。如果算法不含参数合法性检测代码段，书写入口断言是最低限度的要求。

(4) 输入和输出

算法的输入和输出可以通过三种途径实现。第一种是通过 read 和 write 语句实现，其特点是实现了算法与计算环境外部的信息交换；第二种是以算法头中参数表里显式列出的参量作为输入/输出的媒介；第三种是通过全局变量或外部变量隐式地传递信息。后两种方法的特点是实现了一个算法与其调用者之间的信息交换。

如果一个算法是定义在某个数据结构之上的几个操作之一，该数据结构可以不列在算法的参数表中。在其他情况下，应尽量避免使用第三种方法。

(5) 错误处理

可以使用 ERROR 语句处理错误，它的语义是停止算法的执行(如果带参数，还要印出参数)。在某些情况下，这种方法可以减少 **IF** 语句的嵌套层次。但是利用变参或函数值返回算法的执行状态(正确/错误，或是错误号等)，便于调用者处理异常情况，是更值得提倡的错误处理方法，有利于培养良好的程序设计习惯。

(6) 语句选用和算法结构

赋值语句、**IF** 分支语句和 **WHILE** 循环语句是最基本的三种语句，仅用此三种语句就足以对付一切算法的设计了。实际上，不仅是"足够"，而且是"最好"。这样做对于提高结构良好性和可读性、避免逻辑错误是有益和有效的。**WHILE** 语句比其他循环语句更容易验证算法的正确性。**CASE** 分支语句是广义的 **IF** 分支语句，在分支条件复杂时选用可以避免 **IF** 语句的多重嵌套，有助于提高算法的可读性，也是一个鼓励使用的语句。在绝对合适的情况下选用 **FOR** 循环语句也是合理的，不准使用 **GOTO** 语句(涉及 3.3 节的习题除外)。

算法设计过程中应尽量避免下列所示的语句结构：

```
REPEAT
  REPEAT
    …
  UNTIL…
  …
UNTIL…
```

或者

```
IF…
THEN IF…
```

对于第二种情形，如果难以改变，应该对第二个 **IF** 语句加上一对语句括号，以便明确 **THEN** 的作用范围。此外，语句的开/闭括号应对齐。

(7) 基本运算

如果题目中未明确要求用某种数据结构上的基本运算编写算法，不得直接利用教科书中给出的基本运算。如果非用不可，则要求将所用到的所有基本运算同时实现。

(8) 几点建议

- 建议以图说明算法；
- 建议在算法书写完毕后，用边界条件的输入参数值验证一下算法能否正确执行。例如，对于顺序表插入算法，空表是一个边界条件。

附例:算法规格说明例释

例 1 第 2.10 题

见原题。

例 2 第 2.39 题

```
FUNCTION Evaluate( pn : SqPoly;  x : real ) : real;
{多项式 pn.data[i].coef 存放 ai,pn.data[i].exp 存放 ei (i=1,2,…,m)。本算法}
{计算并返回 ∑aᵢx^eᵢ。不判别溢出。此外,入口时要求 0≤e1<e2<…<em,算法内}
{不对此再作验证。}
```

例 3 第 3.15 题的入栈 push 算法

```
PROC push( VAR tws : TwoWayStack; i : 0..1;  x : elemtype;
                                        VAR overflow : boolean);
{两栈(标号 0、1)共享空间 tws.elem[1..m],栈底在两端。tws.top[0]和 tws.top[1]}
{分别为两栈的栈顶指针。tws.data[1..tws.top[0]]为栈 0,tws.data[tws.top[1]..m]}
{为栈 1。本算法将 x 推入栈 i。入口时栈满则不改变栈且返回 overflow 为 true。   }
```

例 4 第 3.30 题的出列 del_queue 算法

```
FUNC dequeue (VAR x : elemtype) : boolean;
{循环队列 squeue[(m+rear-quelen+1) MOD m.(m).rear]之上的出列操作。   }
{quelen 为队列长度。当 quelen=0 时,返回 false;否则返回 true,        }
{且 x 将出列元素值带回。                                            }
```

例 5 递归算法

见题 5.35 答案。

第1章　绪论(预备知识)

1. 基本内容

数据、数据元素、数据对象、数据结构、存储结构和数据类型等概念术语的确定含义；抽象数据类型的定义、表示和实现方法；描述算法的类 **Pascal** 语言；算法设计的基本要求以及从时间和空间角度分析算法的方法。

2. 学习要点

(1) 熟悉各名词、术语的含义，掌握基本概念，特别是数据的逻辑结构和存储结构之间的关系。分清哪些是逻辑结构的性质，哪些是存储结构的性质。

(2) 了解抽象数据类型的定义、表示和实现方法。

(3) 熟悉类 **Pascal** 语言的书写规范，特别要注意值参和变参的区别，输入、输出的方式以及错误处理方式。

(4) 理解算法5个要素的确切含义：①动态有穷性(能执行结束)；②确定性(对于相同的输入执行相同的路径)；③有输入；④有输出；⑤可行性(用以描述算法的操作都是足够基本的)。

(5) 掌握计算语句频度和估算算法时间复杂度的方法。

3. 基础知识题

1.1① 简述下列术语：数据、数据元素、数据对象、数据结构、存储结构、数据类型和抽象数据类型。

1.2② 试描述数据结构和抽象数据类型的概念与程序设计语言中数据类型概念的区别。

◆1.3② 设有数据结构(D,R)，其中

D={d1,d2,d3,d4}，R={r}，r={(d1,d2),(d2,d3),(d3,d4)}。

试按图论中图的画法惯例画出其逻辑结构图。

◆1.4② 试仿照复数的抽象数据类型写出抽象数据类型有理数的定义(有理数是其分子、分母均为自然数且分母不为零的分数)。

1.5② 试画出与下列程序段等价的框图。

```
(1) product :=1;  i :=1;
    WHILE i≤n DO
        [ product :=product * i;
          i :=i+1
```

```
    】;
(2) i :=0;
   REPEAT
      i :=i+1
   UNTIL (a[i]=x) OR (i=n);
(3) CASE
      x<y :   z :=y-x;
      x=y :   z :=ABS(x * y);  {ABS 为取绝对值函数}
      ELSE z :=(x-y)/ABS(x) * ABS(y)
   ENDC;
```

◆1.6③ 在程序设计中,常用下列 3 种不同的出错处理方式:

(1) 用 **ERROR** 语句终止执行并报告错误;

(2) 用布尔函数实现算法以区别正确返回或错误返回;

(3) 在过程的参数表中设置一整型变量以区别正确返回或某种错误返回。

试讨论这三种方法各自的优缺点。

◆1.7③ 在程序设计中,可采用下列 3 种方法实现输出和输入:

(1) 通过 read 和 write 语句;

(2) 通过参数表中的参数显式传递;

(3) 通过全局变量隐式传递。

试讨论这三种方法的优缺点。

1.8④ 设 n 为正整数。试确定下列各程序段中前置以记号@的语句的频度:

```
(1) i :=1;  k :=0;
     WHILE i≤n-1 DO
     【@  k := k +10 * i;
          i := i +1
     】;
(2) i :=1;  k :=0;
     REPEAT
   @  k := k +10 * i;
      i := i+1
    UNTIL NOT(i≤n-1);
(3) i := 1;  k := 0;
    WHILE  i≤n-1  DO
     【i : = i + 1;
     @  k := k + 10 * i】;
(4) k :=0;
     FOR i :=1 TO n DO
       FOR j :=i TO n DO
     @  k := k + 1;
```