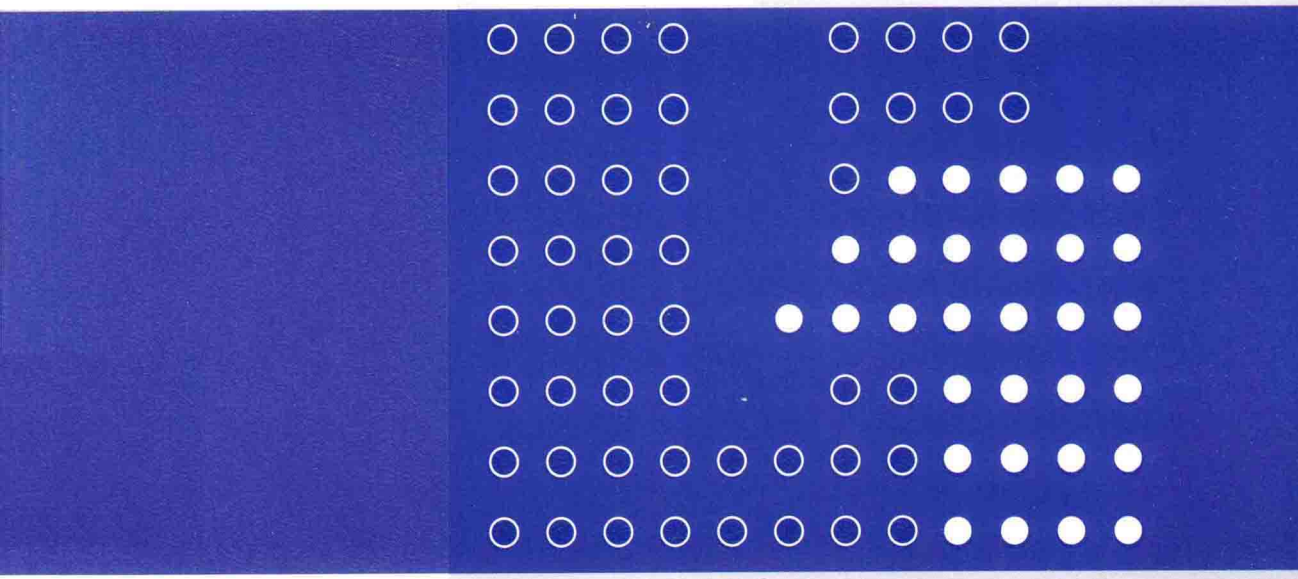
普通高等教育“十一五”国家级规划教材 计算机系列教材

北京市高等教育精品教材立项项目

实用数据结构习题解答与上机指导



林小茶 编著

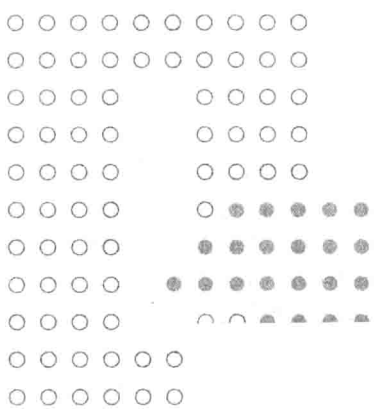
清华大学出版社

普通高等教育“十一五”国家级规划教材 计算机系列教材

北京市高等教育精品教材立项项目

林小茶 编著

实用数据结构习题解答与上机指导



清华大学出版社
北京

内 容 简 介

本书是与北京市精品教材《实用数据结构》(林小茶编著,ISBN:9787302338284)配套的习题解答和上机指导。

习题解答与主教材的内容完全对应。包括概述、栈与队列、线性表及线性表的顺序存储、线性表的链式存储、哈希表与索引表、内排序、树与二叉树和图等内容。

上机指导部分包括栈与队列、线性表、线性表的链式存储、树与二叉树和图等内容。

本书既可以作为计算机相关专业本科学生学习数据结构的教材,也可作为自学者的参考书。

图书在版编目(CIP)数据

实用数据结构习题解答与上机指导/林小茶编著. --北京:清华大学出版社,2015
计算机系列教材
ISBN 978-7-302-39959-9

Ⅰ. ①实… Ⅱ. ①林… Ⅲ. ①数据结构-高等学校-教学参考资材 Ⅳ. ①TP311.12

中国版本图书馆 CIP 数据核字(2015)第 085934 号

责任编辑:张 民
封面设计:常雪影
责任校对:焦丽丽
责任印制:王静怡

出版发行:清华大学出版社
网 址:http://www.tup.com.cn, http://www.wqbook.com
地 址:北京清华大学学研大厦 A 座 **邮 编**:100084
社 总 机:010-62770175 **邮 购**:010-62786544
投稿与读者服务:010-62776969, c-service@tup.tsinghua.edu.cn
质量反馈:010-62772015, zhiliang@tup.tsinghua.edu.cn
课件下载:http://www.tup.com.cn,010-62795954
印 刷 者:北京四季青印刷厂
装 订 者:三河市吉祥印务有限公司
经 销:全国新华书店
开 本:185mm×260mm **印 张**:8.25 **字 数**:187 千字
版 次:2015 年 7 月第 1 版 **印 次**:2015 年 7 月第1次印刷
印 数:1~2000
定 价:22.00 元

产品编号:055675-01

《实用数据结构习题解答与上机指导》 前 言

“数据结构”是一门对实践环节要求很高的课程，通过做题和上机练习能够很好地巩固所学知识。为了帮助学生顺利地上机调试程序以及完成相关的书面练习而编写了此书。建议学生在经过独立思考之后再参考本书的答案，程序的答案也并不是唯一的，希望大家将本书作为参考，同时开拓思路，尽量自己编写相关的程序，才能真正学到知识。“Ctrl＋C”和“Ctrl＋V”对于学习程序设计的人来说是非常有害的。出于这种考虑，本书对于主教材中列出的实习题只给出了部分答案，没有给出答案的题目可以由学生自己完成或作为课程设计的题目。

按照“数据结构”的教学要求，习题解答部分的大部分算法不是完整的程序，而是算法描述。因此笔者编写了上机指导，上机指导中的程序是完整的，可以调试。但是要注意读懂所有的函数，当然还包括主函数以及宏定义，否则，上机调试有可能失败。

上机指导中还包括了程序阅读题目的完整程序，目的是提示读者如何得到待阅读程序的正确结果。由于部分章节的程序比较简单，上机指导部分对应于主教材只包含了第2,3,4,7章和第8章。

本书包含的大部分函数都有比较详尽的注释，而主函数和阅读程序练习中的程序几乎不包含注释，这是编者有意为之，希望读者得到锻炼自己阅读程序能力的机会。

由于编者水平有限，错误在所难免，请广大读者批评指正。

作　者

2015年3月

FOREWORD

第1部分　习题解答

第1章　概述

【1-1】 请解释以下术语，并尽可能地举一些例子。

解：（例子略，请读者参考主教材举例，尽量与教材不相同）

（1）数据

数据（Data）是能够输入到计算机中，并被计算机识别、存储和处理的符号集合。数据可以分为数值数据和非数值数据。

（2）数据元素

数据元素（Data Element）是组成数据的基本单位，每个数据元素是具有独立意义的个体，在程序设计时，将其作为一个整体来对待。

（3）数据对象

数据对象（Data Object）是一组具有相同性质的数据集合。

（4）数据结构

数据结构研究各数据元素之间存在的关系。数据结构包括数据的逻辑结构、存储结构（物理结构）和数据的运算三方面的内容。

数据元素之间的逻辑关系称为数据的逻辑结构。

数据结构在计算机中的表示称为数据的物理结构（或存储结构）。它包括数据元素的表示和关系的表示。

数据的运算是在数据的逻辑结构上定义的操作。

（5）算法

用比较通俗的语言说，算法是解题的步骤。严格地讲，算法是一个有穷的规则集合，这些规则为解决某一特定任务规定了一个运算序列。

【1-2】 算法的特性有哪些？举例说明。

解：（例子略，请读者参考主教材举例，尽量与教材不相同）

算法具有五大特性：

① 输入。一个算法具有零个或多个输入，这些输入取自特定的数据对象集合。

② 输出。一个算法具有一个或多个输出，这些输出同输入之间存在某种特定的关系。

③ 有穷性。一个算法必须在有穷步之后结束，即必须在有限时间内完成。

④ 确定性。算法的每一步必须有确切的定义，无二义性。

⑤ 可行性。算法中的每一步都可以通过已经实现的基本运算的有限次执行得以实现。

【1-3】 单选题

(1) 从逻辑上可以将数据结构分为(　　)两大类。

A. 动态结构、静态结构　　B. 顺序结构、链式结构

C. 线性结构、非线性结构　　D. 初等结构、构造型结构

解：答案是 C。

(2) 数据结构中讨论有关数据的最小单位是(　　)。

A. 数据对象　　B. 数据元素　　C. 数据项　　D. 以上都不对

解：答案是 C。

(3) 数据结构中组成数据的基本单位是(　　)。

A. 数据对象　　B. 数据元素　　C. 数据项　　D. 以上都不对

解：答案是 B。

(4) 下面关于算法说法错误的是(　　)。

A. 算法最终必须由计算机程序实现

B. 为解决某问题的算法与为该问题编写的程序含义是相同的

C. 算法的可行性是指指令不能有二义性

D. 算法的确定性是指指令不能有二义性

解：答案是 C。

(5) 下面算法程序段的时间复杂度是(　　)。

```
for (int i=0; i<m; i++)
    for (int j=0; j<n; j++)
        a[i][j]=i * j;
```

A. $O(m^2)$　　B. $O(n^2)$　　C. $O(m*n)$　　D. $O(m+n)$

解：答案是 C。

【1-4】 判断题

(1) 程序不等于算法。

解：正确。

(2) 算法可以用不同的语言进行描述。

解：正确。

(3) 线性结构只能用顺序存储结构存储。

解：错误。

(4) 非线性结构只能用链式存储结构存储。

解：错误。

(5) 数据元素是数据的最小单位。

解：错误。

(6) 算法的优劣与算法描述语言无关。

解：正确。

【1-5】 分析下列算法的效率。

(1)

```
int MaxValu_3(int a[][3],int n,int m)
{   max=a[0][0];
    for (i=0;i<n;i++)
        for (j=0;j<m;j++)
            if (a[i][j]>max)
        max=a[i][j];
    return max;
}
```

解：O(m * n)。

(2)

```
int MaxValu_4(int a[ ][3],int n,int m)
{   max_i=0;
    max_j=0;
    for (i=0;i<n;i++)
        for (j=0;j<m;j++)
            if (a[i][j]>a[max_i][max_j])
            {   max_i=i;
                max_j=j;
            }
    return a[max_i][max_j];
}
```

解：O(m * n)。

【1-6】 实习题

(1) 编写主函数调用例 1.8 和例 1.9 中的算法，并对程序进行调试。注意例 1.8 和例 1.9 中的算法并不是完整的程序。

调用例 1.8 的主函数：

```
#include "sidio,h"
int MaxValu_1(int a[],int size);
int main()
{
    int a[]={10,9,8,7,6,5};
    printf("最大值是", MaxValu_1(a,6);
}
int MaxValu_1(int a[],int size)
{   max=a[0];                          /*假设下标为 0 的元素最大,赋值给 max*/
    for (i=1;i<size;i++)               /*i 从下标为 1 到下标 size-1*/
        if (a[i]>max)                  /*判断当前元素 a[i]是否大于 max*/
```

```
            max=a[i];                    /*是,当前元素值覆盖 max*/
    return max;                          /*返回最大值*/
}
```

调用例 1.9 的主函数:

```
#include "stdio,h"
int MaxValu_2(int a[],int size);
int main()
{
    int a[]={10,9,8,7,6,5};
    printf("最大值是",a[ MaxValu_2(a,6)]);
}
int MaxValu_2(int a[],int size)
{   max_loca=0;              /*假设下标为 0 的元素最大,max_loca 记录最大元素的位置*/
    for (i=1;i<size;i++)     /*i 从下标为 1 到下标 size-1*/
        if (a[i]>a[max_loca]) /*判断当前元素 a[i]是否大于 max_loca 位置的元素*/
            max_loca=i;      /*是,当前元素的位置覆盖 max_loca*/
    return max_loca;         /*返回最大元素的位置*/
}
```

(2) 编写主函数调用习题 1-5 的两个算法。

本题属于读者自己练习的题目。可参照上一题。

第 2 章　栈与队列

【2-1】 单选题

(1) 已知一个栈的入栈序列是 1,2,3,4,5,6,则不可能的输出序列是(　　)。

A. 1 2　3 4　5 6　　B. 5 6 4 1 2 3

C. 2 4　3 5　6 1　　D. 5 6 4 3 2 1

解:答案是 B。因为若要得到输出序列 5,6,4,1,2,3,5,必须最先出栈,那么就需要将 1,2,3,4 和 5 依次进栈,然后 5 出栈,6 再进栈,6 出栈,4 出栈,这时的栈顶元素是 3,1 是不可能出栈的。

其他的答案都是能够得到的输出序列,以答案 A 为例:1 进栈、1 出栈、2 进栈、2 出栈、3 进栈、3 出栈、4 进栈、4 出栈、5 进栈、5 出栈、6 进栈、6 出栈。

(2) 栈和队列的共同特点是(　　)。

A. 都是操作受限的线性表　　B. 都是先进后出

C. 都是后进后出　　D. 无共同点

解:答案是 A。栈和队列是特殊的线性表,栈只能在栈顶一端进行插入和删除,队列只能在队尾做插入和在队头做删除。

(3) 如果以链表作为栈的存储结构,则出栈操作时,需要做的是(　　)。

A. 必须判别栈是否已满　　B. 必须判别栈是否为空

C. 必须判别栈元素的类型　　D. 必须判别是否还有可用空间

解：答案是B，不论是链表还是顺序表，出栈操作时都要判断栈是否为空。

(4) 设一个链栈的栈顶指针是st，栈中结点类型为NODE(info,link)，如果栈不为空，则退栈的操作是(　　)。

A. q=st; st=st->link; free(q);

B. free(st);

C. st=st->link; free(q)

D. st=st->link; free(st);

解：答案是A，退栈操作。原因是：st的内容指向原来的栈顶，先用q取出，修改st，指向新的栈顶，然后释放原来的栈顶元素。

(5) 设InitQueue(Q)、EnQueue(Q,e)和DeQueue(Q,e)分别表示队列初始化、入队和出队操作。经过以下队列操作后，队头的值是(　　)。

InitQueue(Q); EnQueue(Q,a);EnQueue(Q,b);EnQueue(Q,c);DeQueue(Q,x)

A. a　　B. b　　C. NULL　　D. x

解：答案是B。原因是：对队列的操作是初始化队列、a进队列、b进队列，c进队列，然后是a出队列。队头的值是b。

(6) Push(e)表示e进栈，Pop(e)表示退栈并将栈顶元素存入e。那么，下面的程序段正好可以将A和B的内容交换的操作序列是(　　)。

A. Push(A) Push(B) Pop(A) Pop(B)

B. Push(A) Push(B) Pop(B) Pop(A)

C. Push(A) Pop(B) Push(B) Pop(A)

D. Push(B) Pop(A) Push(A) Pop(B)

解：答案是A。假设A单元内容是10和B单元内容是20，先将A单元内容进栈，10进栈，再将20进栈，栈顶元素和次栈顶元素是20和10，此时将栈顶元素20出栈并覆盖A，因此A中是20，新的栈顶元素是10，出栈覆盖B，B修改为10。A和B的原始值得到了互换。

(7) Push表示进栈，Pop表示退栈，输入序列为ABC，可以变为CBA时，经过的栈操作为(　　)。

A. Push, Pop, Push, Pop, Push, Pop

B. Push, Push, Push, Pop, Pop, Pop

C. Push, Push, Pop, Pop, Push, Pop

D. Push, Pop, Push, Push, Pop, Pop

解：答案是B。输入序列和输出序列完全相反。

(8) 一个栈的输入序列为1,2,3,…,n，若输出序列的第一个元素是n，输出第i(1≤i≤n)个元素是(　　)。

A. 不确定　　B. n-i+1　　C. i　　D. n-i

解：答案是B，第一个输出的是n，第i个输出的是n-(i-1)，即n-i+1。

(9) 用不带头结点的单链表存储队列时，其队头指针指向队头结点，其队尾指针指向

队尾结点，则在进行删除操作时，正确的描述是(　　)。

A. 仅修改队头指针　　B. 仅修改队尾指针

C. 队头、队尾指针都要修改　　D. 队头、队尾指针都可能要修改

解：答案是D，如果队列中只剩一个元素，删除这个元素后，队头、队尾指针要修改，但是，其他情况就不需要修改，因此准确的答案是D。

(10) 若用一个大小为6的数组来实现环形队列，且当前rear和front的值分别为0和3，当从队列中删除一个元素，再加入两个元素后，rear和front的值分别是(　　)。

A. 1和5　　B. 2和4　　C. 4和2　　D. 5和1

解：答案是B。

原因是：初始情况的示意图如图2-1所示，队列中有3个元素，rear和front的值分别为0和3；删除一个元素，再加入两个元素以后的队列如图2-2所示。

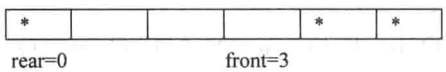


图2-1　队列的初始情况

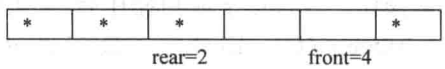


图2-2　删除元素和插入元素之后队列的情况

注意：队列的操作与主教材一致。

(11) 在链队列中执行入队操作，正确的是(　　)。

A. 需判断队列是否为空　　B. 需判断队列是否为满

C. 限制在链表头进行　　D. 限制在链表尾进行

解：答案是D。

(12) 队列是(　　)。

A. 先进先出的线性表　　B. 先进后出的线性表

C. 后进先出的线性表　　D. 随意进出的线性表

解：答案是A。基本概念！

(13) 设栈S和队列Q的初始状态为空，元素e1,e2,e3,e4,e5和e6依次通过栈S，元素退栈后即进入队列Q，若6个元素的出队序列是e2,e4,e3,e6,e5,e1，则栈S的容量至少为(　　)。

A. 2　　B. 3　　C. 4　　D. 6

解：若想得到出队序列e2,e4,e3,e6,e5,e1，进队的序列与之相同，也就是从S出栈的序列。进出栈的顺序是：e1、e2进栈，e2出栈，e3、e4进栈(此时栈中的元素是3个)，e4、e3出栈，e5、e6进栈(此时栈中的元素还是3个)，e6、e5出栈，最后e1出栈。因此栈的容量至少为3。

(14) 已知一个栈的入栈序列是1,2,3,…,n，其输出序列为p1,p2,p3,…,pn，若p1是n，则pi是(　　)。

A. i　　B. n−i　　C. n−i+1　　D. 不确定

解：答案是C。此题与第(8)小题大同小异。

(15) 设环形队列的元素存放在一维数组Q[0..30]中，队列非空时，front指示队头元素的前一个位置，rear指示队尾元素。如果队列中元素的个数为11，front的值为25，

则 rear 应指向的元素是(　　)。

A. Q[4]　　B. Q[5]　　C. Q[14]　　D. Q[15]

解：答案是 B。根据图 2.3 进行计算。

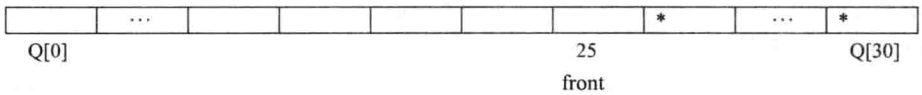


图 2-3　环形队列

30－25＝5，因此从 25 到 30 可以存储 5 个元素，还有 11－5＝6 个元素，需要存储在 0 号空间到 5号空间。rear 应指向的元素是 Q[5]。

(16) 下面是在顺序栈上实现的一个栈基本操作，该操作的功能是(　　)。

```
typedef struct{
    DataType data[100];
    int top;
}SeqStack;
DataType fun(SeqStack * S)
{   if  (StackEmpty(S))
        Error("Stack is empty ");
    return S->data[S->top];
}
```

A. 进栈　　B. 退栈　　C. 取栈顶元素　　D. 判栈空

解：答案是 C。取栈顶元素。S 是指向一个顺序栈的指针。

【2-2】 运算题

(1) 一个环形队列的容量是 23(序号从 0 到 22)，经过一系列的入队运算与出队运算，若有 front＝8，rear＝3，环形队列中有多少个元素？

解：18 个元素，见图 2-4。

23－(8－3)＝18

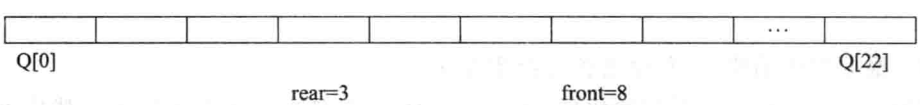


图 2-4　环形队列

(2) 有 4 个元素 1234 依次进栈，任何时候都可以出栈，请写出所有可能的出栈顺序和所有不可能的序列。

解：可能的出栈顺序：

1234　1243　1342　1324　1432　2134　2143　2431　2314　2341

3214　3241　3421　4321

不可能的出栈顺序：

1423　2413　3124　3142　3412　4123　4132　4213　4231　4312

(3) 假定一维数组 a[8]顺序存储一个环形队列，队列非空时，front 指示队头元素的前一个位置，rear 指示队尾元素。假设队列中已经有 6 个元素：13、24、45、26、78、51，其中 13 是队头元素，front 的值为 4，请画出此时的存储状态，当再连续做 3 次出队运算以后，再让 19、42 和 77 进队列，再画出对应的存储状态。

解：见图 2-5 和图 2-6。

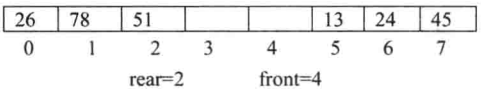


图 2-5 题目要求的开始状态

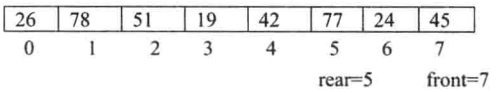


图 2-6 题目要求的结束状态

注意：24 和 45 看上去还在，实际上已经不属于队列元素了。

【2-3】 算法分析题(判断下列算法的执行结果，其中包含对本章所学算法的调用)

(1)

```
void test1()
{   int i;
    int a[]={10,9,8,7,6,5,4,3,2,1};
    ElemType temp;
    LinkStack L_Stack;
    Init_LinkStack(&L_Stack);
    for (i=0;i<=9;i++)
       if (Push_LinkStack(&L_Stack,a[i])==False)
          break;
    while (Empty_LinkStack(L_Stack)==False)
    {   Pop_LinkStack (&L_Stack,&temp);
        printf("  %d",temp);
        if (temp==5)
            SetNull_LinkStack(&L_Stack);
    }
}
```

解：根据算法的描述，程序的执行结果是 1 2 3 4 5。

原因是：算法首先进行初始化栈，然后将数组 a 的内容依次推入栈中。栈中元素从栈顶依次是 1、2、3、4、5、6、7、8、9、10。然后循环退栈输出，直到输出的元素值为 5 时清空栈。

(2)

```
void test2()
{   int i;
    int e=7;
    int a[]={8,7,6,5,4,7,3,2,7,1};
    ElemType temp;
```

```
    LinkStack L_Stack1,L_Stack2;
    Init_LinkStack(&L_Stack1);
    Init_LinkStack(&L_Stack2);
    for (i=0;i<=9;i++)
      if (Push_LinkStack(&L_Stack1,a[i])==False)
         break;
    while (Empty_LinkStack(L_Stack1)==False)
    {   Pop_LinkStack (&L_Stack1,&temp);
        if (temp!=e)
            if (Push_LinkStack(&L_Stack2,temp)==False)
                break;
    }
    while (Empty_LinkStack(L_Stack2)==False)
    {   Pop_LinkStack (&L_Stack2,&temp);
        printf("%d ",temp);
    }
}
```

解：根据算法的描述，程序的执行结果是 1 2 3 4 5 6 8。

原因是：算法首先进行初始化两个栈，然后将数组 a 的内容依次推入栈 1 中。栈中元素从栈顶依次是 1、7、2、3、7、4、5、6、7、8。然后循环从栈 1 中退栈，如果退栈的元素不是 7 则压入栈 2 中，最后从栈 2 中退栈的结果就不含有 7。

【2-4】 算法设计题

(1) 假设以带表头结点的循环链表表示队列，并且只设一个指针指向队尾元素(不设头指针)，请编写相应的队列操作算法：创建一个空队列、判队空、进队列、出队列、取队头元素和清空一个队列等。

解：

```
typedef  struct  node {                          /*链表结点的定义*/
    ElemType data;
    struct node *next;
} Node;
typedef  struct  {
    Node   *rear;                                /*队尾指针*/
} LinkQueue;
int InitQueue(LinkQueue *Q)
{                                                /*创建一个空队列*/
    Q->rear=(struct node *)malloc(sizeof(Node));  /*申请空间作为表头结点*/
    if (!Q->rear) exit (OverFlow);
    Q->rear->next=Q->rear;                  /*表头结点的指针指向自己*/
    Q->rear->data=99;                       /*表头结点的值设置为 99*/
    return OK;
```

```
}
int EmptyQueue(LinkQueue * Q)
{                                          /* 判队列空 */
    if (Q->rear->next==Q->rear) return True;
    return False;
}
int EnQueue(LinkQueue * Q,ElemType e)
{                                          /* 插入元素 e 为 Q 的新的队尾元素 */
    Node * p;
    p=(Node * )malloc(sizeof(Node));       /* 申请空间 */
    if (!p) exit (OverFlow);
        p->data=e;                         /* 设置数据域 */
    p->next=Q->rear->next;                 /* 设置指针域 */
    Q->rear->next=p;                       /* 设置队尾元素指针域 */
    Q->rear=p;                             /* 设置队尾指针 */
    return OK;
}
int DeQueue(LinkQueue * Q,ElemType * e)
{         /* 若队列不空,则删除 Q 的队头元素,用 e 返回其值,并返回 OK;否则返回 ERROR */
    Node * p;
    if (Q->rear->next==Q->rear) return Error;
                                           /* 若队列空返回错误信息 */
    p=Q->rear->next->next;                 /* 取队列的第一个元素位置 */
    * e=p->data;                           /* 取队列的第一个元素值 */
    if (p==Q->rear)                        /* 删除的为队尾元素 */
    {  Q->rear=Q->rear->next; Q->rear->next=Q->rear;}
    else                                   /* 删除其他元素 */
      Q->rear->next->next=p->next;
    return OK;
}
int GetFrontQueue(LinkQueue * Q,ElemType * e)
{             /* 若队列不空,取 Q 的队头元素,用 e 返回其值,并返回 OK;否则返回 ERROR */
    Node * p;
    if (Q->rear->next==Q->rear) return Error;        /* 若队列空返回错误信息 */
    p=Q->rear->next->next;                 /* 取队列的第一个元素位置 */
    * e=p->data;                           /* 取队列的第一个元素值 */
    return OK;
}
void SetNullQueue(LinkQueue * Q)
{                                          /* 清空队列,只剩一个表头结点 */
    Node * p, * q;
    p=Q->rear->next->next;                 /* 取表头结点的下一个结点 */
    while(p!=Q->rear->next)                /* 未到表尾结点 */
    {   q=p;                               /* q 取 p 的前驱 */
```

```
        p=p->next;                          /*走链*/
        free(q);                            /*释放 q*/
    }
}
```

注意题目的要求是带表头结点的循环链表表示队列，并且只设一个指针指向队尾元素(不设头指针)，例如清空队列时，需要剩一个表头结点。

(2) 要求环形队列不损失一个空间就能够得到全部的利用，方法是设置一个标志域tag，tag为0或1来区分队头与队尾位置相等时表示的是队列满还是队列空，请编写基于此数据结构的入队和出队的算法。

解：

```
typedef int ElemType;
typedef  struct
{   ElemType Element[MaxSize];          /*存储队列元素的数组*/
    int  rear, front,tag;               /*队头、队尾位置*/
}SeqQueue;
SeqQueue ListQueue;
int InitSeqQueue(SeqQueue *Q)
{   /*构造一个空队列*/
    Q->front=Q->rear=0;
    Q->tag=0;
    return OK;
}
int EmptyQueue(SeqQueue *Q)
{   /*判队列空*/
    if (Q->rear==Q->front&&Q->tag==0) return True;
    return False;
}
int InQueue(SeqQueue *Q,ElemType e)
{   /*插入元素 e 为 Q 的新的队尾元素*/
    if (Q->tag==1&&Q->rear==Q->front) return Error;      //队列满
    Q->Element [Q->rear]=e;
    if (Q->tag==0) Q->tag=1;
    Q->rear=(Q->rear+1)%MaxSize;
    return OK;
}
int  OutQueue(SeqQueue *Q,ElemType *e)
{   /*若队列不空,则删除 Q 的队头元素,用 e 指向返回值,并返回 OK;否则返回 ERROR*/
    if (Q->tag==0&&Q->rear==Q->front) return Error;
     *e=Q->Element [Q->front];
    Q->front=(Q->front+1)%MaxSize;
    if (Q->rear==Q->front)
        Q->tag=0;
```

```
    return OK;
}
```

【2-5】 实习题

(1) 修改进制转换的问题,要求可以对一个整数做任意小于 9 的进制转换。

解:完整程序如下:

```
#include "stdio.h"
#define MaxSize 10
#define True 1
#define False 0
#define OK 1
#define OverFlow 0
#define UnderFlow 0
typedef int ElemType;
typedef struct
{   ElemType Element[MaxSize];
    int Top;                                          /*栈顶*/
} SeqStack;                                           /*说明 SeqStack*/
void Init_SeqStack(SeqStack *S_pointer)               /*构造一个空栈*/
{   S_pointer->Top=-1;
}
int Empty_SeqStack(SeqStack *S_pointer)               /*判栈空*/
{   if (S_pointer->Top==-1) return True;
    else  return False;
}
int Full_SeqStack(SeqStack *S_pointer)                /*判栈满*/
{   if (S_pointer->Top==MaxSize-1) return True;       /*返回栈满标记*/
    else  return False;                               /*返回栈不满的标记*/
}
int Push_SeqStack(SeqStack *S_pointer,ElemType x)     /*进栈*/
{   if (Full_SeqStack(S_pointer)==True)               /*判栈满*/
        return OverFlow;                              /*栈满则返回操作失败标记*/
    else
    {   S_pointer->Top++;                             /*栈顶位置加 1*/
        S_pointer->Element[S_pointer->Top]=x;         /*元素 x 赋值到栈顶位置*/
        return OK;                                    /*返回操作成功标记*/
    }
}
int Pop_SeqStack(SeqStack *S_pointer,ElemType *x_pointer)    /*退栈*/
{   if (Empty_SeqStack(S_pointer)==True)              /*判栈空*/
        return UnderFlow;                             /*栈空则操作失败*/
    else
    {   *x_pointer=S_pointer->Element[S_pointer->Top];
```