跨考教育
www.kuakao.com

跨考教育计算机教研室 编

全面贯彻“跨越135分”辅导理念

全国硕士研究生入学统一考试

计算机基础综合辅导

QUANGUO SHUOSHI YANJIUSHENG RUXUE TONGYI KAOSHI
JISUANJI JICHU ZONGHE FUDAO LANBAOSHU

蓝宝书

★历年考试真题分值分布表　★复习技巧
★重点、难点、考点梳理　★备考说明
★真题演练及答案解析　★自我检测

北京邮电大学出版社
www.buptpress.com

最新版

全国硕士研究生入学统一考试
计算机基础综合辅导·蓝宝书

跨考教育计算机教研室　编

北京邮电大学出版社
·北京·

内容简介

“计算机考研跨越135分必备”系列包括四个分册：(1)《全国硕士研究生入学统一考试计算机基础综合辅导·蓝宝书》；(2)《全国硕士研究生入学统一考试计算机考研核心习题集·绿宝书》；(3)《全国硕士研究生入学统一考试计算机历年真题全真解析·黄宝书》；(4)《全国硕士研究生入学统一考试计算机全真模拟题及答案·红宝书》。每一个分册的编写都凝聚了跨考教育教授们多年的研究心血。

本书分为四个部分：数据结构、计算机组成原理、操作系统和计算机网络。每个章节均按照教育部新大纲的结构编写而成，分为知识点精讲、典型例题、习题练习以及习题答案。除选择题外，我们对于每一道习题都给出了详尽的解答。本书的知识面完全契合计算机统考新大纲，不缺不溢，而且难度适中，适合广大计算机专业考研学子作为备考复习全书使用。

本书不仅特别适合在硕士研究生入学考试中参加理工类科目考试的考生，也适合各大院校学习理工类高级课程的师生，对于参加高级职称考试及其他相关专业人员来说，本书也是一本宝贵的学习和了解计算机课程的参考资料。

图书在版编目(CIP)数据

全国硕士研究生入学统一考试计算机基础综合辅导：蓝宝书/跨考教育计算机教研室编．--北京：北京邮电大学出版社，2011.7

ISBN 978-7-5635-2651-2

Ⅰ.①全… Ⅱ.①跨… Ⅲ.①电子计算机—研究生—入学考试—自学参考资料 Ⅳ.①TP3

中国版本图书馆CIP数据核字(2011)第118861号

书　　名：全国硕士研究生入学统一考试计算机基础综合辅导·蓝宝书

作　　者：跨考教育计算机教研室

责任编辑：满志文

出版发行：北京邮电大学出版社

社　　址：北京市海淀区西土城路10号(邮编：100876)

发 行 部：电话：010-62282185　传真：010-62283578

E-mail：publish@bupt.edu.cn

经　　销：各地新华书店

印　　刷：北京忠信诚胶印厂

开　　本：787 mm×1 092 mm　1/16

印　　张：26.5

字　　数：657千字

版　　次：2011年7月第1版　2011年7月第1次印刷

ISBN 978-7-5635-2651-2　　定　价：48.80元

前　言

随着2009年教育部的一纸通文下来，全国的计算机专业研究生入学考试采取了计算机专业基础综合考试的形式，也就是我们通常说的计算机统考。为了帮助同学们掌握计算机学科的知识迎接统考，我们特地精心编写了这本书。

本书是编者在对全国多所著名院校研究生入学考试试题分析的基础上，结合目前考研的发展动向编写而成的。全书共包括四个部分，第一部分讲述了数据结构，根据《数据结构C语言版》(严蔚敏主编，清华大学出版社)编写；第二部分讲述了计算机组成原理，主要根据《计算机组成原理》(唐朔飞编写，高等教育出版社)编写；第三部分讲述了操作系统；第四部分讲述了计算机网络，在使用的过程中需要注意部分概念在不同教材体系下描述上的差异。

每章由六部分构成，即核心考点、考纲解析与应试指导、知识点精讲、典型例题、习题练习及习题答案组成。知识点精讲部分详细讲述了本章的知识点，提取出真正的精华部分给大家；典型例题部分通过对典型例题剖析解答，融每章的重点、难点和常用方法于典型例题之中；习题练习部分收集了大量的相关试题，并给出了相应的参考答案。

本书中的绝大多数题目是从近几年来全国多所高校有关课程考研试题中精选出的，并给出了详解和参考答案，有一小部分题目虽然不是考研试题，但很有代表性。不少研究生的入学考试试题来自国内外著名教材和辅导书的习题或习题的变形，这些题目或者思路新颖，或者涉及十分重要的知识点，或者解题方法独到、代表性很强，或者直接或以某种变形的方式出现在考研试题中，本书对于此类典型的试题均安排在典型例题部分。

本书的特点是概念清晰，知识点讲解简洁明了，所有题目都给出了详细的解答，以便于读者在短时间内掌握解题要点。

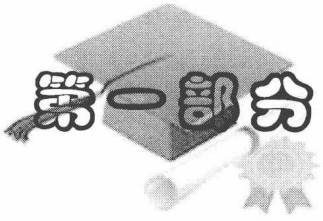
由于作者水平有限，书中难免存在不足之处，敬请有关专家和广大读者不吝指正，如遇到疑难问题，可通过以下方式与我们联系：bjbaba@263.net。

跨考教育计算机教研室

目　录

第一部分　数据结构

第四部分　计算机网络



数据结构

复习方法

统考大纲对数据结构的考查目标定位为理解数据结构的基本概念，掌握数据的逻辑结构、存储结构及其差异，以及各种基本操作的实现；在掌握基本的数据处理原理和方法的基础上，能够对算法进行设计与分析；能够选择合适的数据结构和方法进行问题求解。这个考查目标跟以往各个学校的考研大纲的考查目标并没有什么实质性的区别，这说明数据结构科目考查的指导思想并没有发生变化，同学们可以在不影响已有复习成果的基础上继续进行复习计划，但在数据结构的考点上注意有些调整。

从考试大纲来看，所要求的知识在一般的大学数据结构教材中都已经包含，所以，选择哪本书并不是最重要的事情。建议对于数据结构的复习，可以选择清华大学出版社的《数据结构(第二版)》(严蔚敏主编)。这本书有多种语言的版本，建议选择C语言的版本，在复习的过程中，还可以配以相应的习题集进行练习，来加深对知识点的理解。

对于数据结构的学习，难点在其中的算法及实现上。有条件的考生，可以在计算机上编写程序，自己实现教材上的算法。(要注意，书上的算法通常都采用伪代码编写，需要我们自己用某种程序设计语言去具体实现)数据结构的核心就是算法，首先必须理解经典算法，然后才能创造性地发明简单的算法以解决遇到的问题。所以不管是上机实现，还是在纸上写出来，都一定要注意程序的一些基本规范，这个对于应试是非常重要的。

○ 考试大纲 ○

1. 理解数据结构的基本概念；掌握数据的逻辑结构、存储结构及其差异，以及各种基本操作的实现。

2. 掌握基本的数据处理原理和方法的基础上，能够对算法进行基本的时间复杂度与空间复杂度的分析。

3. 能够选择合适的数据结构和方法进行问题的求解；具备采用C、C++或Java语言设计与实现算法的能力。

一、线性表

（一）线性表的定义和基本操作

（二）线性表的实现

1. 顺序存储结构

2. 链式存储结构

3. 线性表的应用

二、栈、队列和数组

（一）栈和队列的基本概念

（二）栈和队列的顺序存储结构

（三）栈和队列的链式存储结构

（四）栈和队列的应用

（五）特殊矩阵的压缩存储

三、树与二叉树

（一）树的基本概念

（二）二叉树

1. 二叉树的定义及其主要特性

2. 二叉树的顺序存储结构和链式存储结构

3. 二叉树的遍历

4. 线索二叉树的基本概念和构造

5. 二叉排序树

6. 平衡二叉树

（三）树、森林

1. 树的存储结构

2. 森林与二叉树的转换

3. 树和森林的遍历

（四）树与二叉树的应用

1. 等价类问题

2. 哈夫曼(Huffman)树和哈夫曼编码

四、图

（一）图的概念

（二）图的存储及基本操作

1. 邻接矩阵法

2. 邻接表法

（三）图的遍历

1. 深度优先搜索

2. 广度优先搜索

（四）图的基本应用

1. 最小(代价)生成树

2. 最短路径

3. 拓扑排序

4. 关键路径

五、查找

（一）查找的基本概念

（二）顺序查找法

（三）折半查找法

（四）B-树及其基本操作，B+树的基本概念

（五）散列(Hash)表

（六）查找算法的分析及应用

六、内部排序

（一）排序的基本概念

（二）插入排序

1. 直接插入排序

2. 折半插入排序

（三）气泡排序(Bubble Sort)

（四）简单选择排序

（五）希尔排序(Shell Sort)

（六）快速排序

（七）堆排序

（八）二路归并排序(Merge Sort)

（九）基数排序

（十）各种内部排序算法的比较

（十一）内部排序算法的应用

第1章

线性表

核心考点

★ 线性关系、线性表的定义，线性表的基本操作。

★ 线性表的顺序存储结构与链式存储结构（包括单链表、循环链表和双向链表）的构造原理。在以上两种存储结构上对线性表实施的最主要的操作（包括三种链表的建立、插入和删除、检索等）的算法设计。

★ 线性表的应用。

考纲解析与应试指导

线性表在线性结构的学习乃至整个数据结构学科的学习中，其作用都是非常重要的。在这一章，第一次系统性地引入链式存储的概念，链式存储概念将是整个数据结构学科的重中之重，无论哪一章都涉及了这个概念，所以一定要学习透彻了。线性表是线性结构的基础，紧接着的栈、队列和数组等都是线性表在运算或者存储对象上的扩展，树和图是在线性结构一对一关系的基础上变化成一对多和多对多的关系。

线性表一般考题以选择题和小分值的综合应用题居多，建议大家注意线性表的两种存储结构（顺序存储和链式存储）以及两种结构之间的对比，尤其是一些线性表的基本操作在单链表或其变形上的实现步骤，要求考生能够用线性表的常用操作解决线性表的应用问题。

知识点精讲

1.1 线性表的定义

线性表是具有相同数据类型的 $n(n\geqslant 0)$ 个数据元素的有限序列，通常记为：$(a_1, a_2, \cdots,$

$a_{i-1}, a_i, a_{i+1}, \cdots, a_n$)。

其中 n 为表长，$n=0$ 时称为空表。

表中相邻元素之间存在着顺序关系。将 a_{i-1} 称为 a_i 的直接前驱，a_{i+1} 称为 a_i 的直接后继。就是说：对于 a_i，当 $i=2,\cdots,n$ 时，有且仅有一个直接前驱 a_{i-1}；当 $i=1,2,\cdots,n-1$ 时，有且仅有一个直接后继 a_{i+1}。而 a_1 是表中第一个元素，它没有前驱，a_n 是最后一个元素，无后继。

知识点精讲 数据结构

1.2 线性表的基本操作

数据结构的运算是定义在逻辑结构层次上的，而运算的具体实现是建立在存储结构上的，因此下面定义的线性表的基本运算作为逻辑结构的一部分，每一个操作的具体实现只有在确定了线性表的存储结构之后才能完成。

线性表上的基本操作如下：

(1) 线性表初始化：Init_List(L)

初始条件：表 L 不存在。

操作结果：构造一个空的线性表。

(2) 求线性表的长度：Length_List(L)

初始条件：表 L 存在。

操作结果：返回线性表中的所含元素的个数。

(3) 取表元：Get_List(L,i)

初始条件：表 L 存在且 $1\leqslant i\leqslant$Length_List(L)。

操作结果：返回线性表 L 中的第 i 个元素的值或地址。

(4) 按值查找：Locate_List(L,x)，x 是给定的一个数据元素。

初始条件：线性表 L 存在。

操作结果：在表 L 中查找值为 x 的数据元素，其结果返回在 L 中首次出现的值为 x 的那个元素的序号或地址，称为查找成功；否则，在 L 中未找到值为 x 的数据元素，返回一特殊值表示查找失败。

(5) 插入操作：Insert_List(L,i,x)

初始条件：线性表 L 存在，插入位置正确($1\leqslant i\leqslant n+1$，$n$ 为插入前的表长)。

操作结果：在线性表 L 的第 i 个位置上插入一个值为 x 的新元素，这样使原序号为 $i,i+1,\cdots,n$ 的数据元素的序号变为 $i+1,i+2,\cdots,n+1$，插入后表长＝原表长＋1。

(6) 删除操作：Delete_List(L,i)

初始条件：线性表 L 存在，$1\leqslant i\leqslant n$。

操作结果：在线性表 L 中删除序号为 i 的数据元素，删除后使序号为 $i+1,i+2,\cdots,n$ 的元素变为序号为 $i,i+1,\cdots,n-1$，新表长＝原表长－1。

知识点精讲　数据结构

1.3 线性表的顺序存储及运算实现

1.3.1 顺序表

线性表的顺序存储是指在内存中用地址连续的一块存储空间顺序存放线性表的各元素，用这种存储形式存储的线性表称为顺序表。因为内存中的地址空间是线性的。因此用物理上的相邻实现数据元素之间的逻辑相邻关系既简单又自然。设 a_1 的存储地址为 $\mathrm{Loc}(a_1)$，每个数据元素占 d 个存储地址，则第 i 个数据元素的地址为：

$$\mathrm{Loc}(a_i)=\mathrm{Loc}(a_1)+(i-1)\times d \qquad 1\leqslant i\leqslant n$$

这就是说只要知道顺序表首地址和每个数据元素所占地址单元的个数就可求出第 i 个数据元素的地址来，这也是顺序表具有按数据元素的序号随机存取的特点。

类型定义：

```
#define  MAXSIZE 100
struct   LIST {
     elementtype elements[MAXSIZE];
     int last;}
```

位置类型：

```
typedef  int  position;
```

线性表 L：

```
LIST  L;
```

1.3.2 顺序表上基本运算的实现

1. 顺序表的初始化

顺序表的初始化即构造一个空表，这是对表的一个加工型的运算。因此，将 L 设为指针参数，首先动态分配存储空间，然后将表中 last 指针置为 -1，表示表中没有数据元素。

2. 插入运算

线性表的插入是指在表的第 i 个位置上插入一个值为 x 的新元素，插入后使原表长为 n 的表长度为 $n+1$（其中 i 的取值范围为 $1\leqslant i\leqslant n+1$）。

(1) 顺序表上完成这一运算的步骤如下：

① 将 $a_i\sim a_n$ 顺序向下移动，为新元素让出位置；

② 将 x 置入空出的第 i 个位置；

③ 修改 last 指针（相当于修改表长），使之仍指向最后一个元素。

(2) 本算法中注意以下问题：

① 顺序表中数据区域有 MAXSIZE 个存储单元，所以在向顺序表中作插入运算前应先

检查表空间是否满了，在表满的情况下不能再作插入，否则产生溢出错误。

② 要检验插入位置的有效性，这里 i 的有效范围是：$1 \leqslant i \leqslant n+1$，其中 n 为原表长。

③ 注意数据的移动方向。

(3) 插入算法的时间性能分析：

顺序表上的插入运算，时间主要消耗在数据的移动上，在第 i 个位置上插入 x，从 a_i 到 a_n 都要向下移动一个位置，共需要移动 $n-i+1$ 个元素，而 i 的取值范围为：$1 \leqslant i \leqslant n+1$，即有 $n+1$ 个位置可以插入。设在第 i 个位置上作插入的概率为 P_i，则平均移动数据元素的次数：$E_{in} = \sum_{i=1}^{n+1} P_i(n-i+1)$。

设：$P_i = 1/(n+1)$，即为等概率情况，则：

$$E_{in} = \sum_{i=1}^{n+1} P_i(n-i+1) = \frac{1}{n+1}\sum_{i=1}^{n+1}(n-i+1) = \frac{n}{2}$$

这说明：在顺序表上作插入操作需移动表中一半的数据元素。显然时间复杂度为 $O(n)$。

3. 删除运算 DeleteList(L,i)

线性表的删除运算是指将表中第 i 个元素从线性表中去掉，删除后使原表长为 n 的线性表的长度为 $n-1$，其中 i 的取值范围为：$1 \leqslant i \leqslant n$。

(1) 顺序表上完成这一运算的步骤如下：

① 将 $a_{i+1} \sim a_n$ 顺序向上移动。

② 修改 last 指针(相当于修改表长)使之仍指向最后一个元素。

(2) 本算法注意以下问题：

① 删除第 i 个元素，i 的取值为 $1 \leqslant i \leqslant n$，否则第 i 个元素不存在，因此要检查删除位置的有效性。

② 当表空时不能作删除，因表空时 $L->$last 的值为 -1，条件($i<1$||$i>L->$last$+1$)也包括了对表空的检查。

③ 删除 a_i 之后，该数据已不存在，如果需要，先取出 a_i，再作删除。

(3) 删除算法的时间性能分析：

与插入运算相同，该算法的时间复杂度为 $O(n)$。

4. 按值查找

线性表中的按值查找是指在线性表中查找与给定值 x 相等的数据元素。在顺序表中完成该运算最简单的方法是：从第一个元素 a_1 起依次和 x 比较，直到找到一个与 x 相等的数据元素，则返回它在顺序表中的存储下标或序号(二者差一)；或者查遍整个表都没有找到与 x 相等的元素，返回 -1。

本算法的主要运算是比较。显然比较的次数与 x 在表中的位置有关，也与表长有关。当 $a_1 = x$ 时，比较 1 次成功；当 $a_n = x$ 时，比较 n 次成功。平均比较次数为 $(n+1)/2$，时间性能为 $O(n)$。

知识点精讲　数据结构

1.4 线性表的链式存储及运算实现

1.4.1 单链表

单链表是由一个个结点构成的，如图 1-1 所示，结点定义如下：

```
typedef struct node
            {    datatype data;
                 struct node * next;
  }LNode,   * LinkList;
```

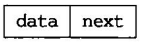


图 1-1　单链表结点结构

定义头指针变量：

```
LinkList H;
```

通常我们用“头指针”来标识一个单链表，如单链表 L、单链表 H 等，是指某链表的第一个结点的地址放在了指针变量 L、H 中，头指针为“NULL”则表示一个空表。

1.4.2 单链表上基本运算的实现

1. 建立单链表

(1) 在链表的头部插入结点建立单链表。

注意：在链表的头部插入，读入数据的顺序和线性表中的逻辑顺序是相反的。

算法如下：

算法 1-1

```
LinkList  Creat_LinkList1()
{    LinkList L = NULL;          //空表
     Lnode * s;
     int x;                      //设数据元素的类型为 int
   scanf(" % d",&x);
   while(x! = flag)
     { s = malloc(sizeof(LNode));
       s ->data = x;
       s ->next = L;L = s;
         scanf(" % d",&x);
     }
   return L;
}
```

(2) 在单链表的尾部插入结点建立单链表。

若希望次序一致,则用尾插入的方法。

算法如下:

算法 1-2

```
LinkList  Creat_LinkList2()
  { LinkList L = NULL;
     Lnode * s, * r = NULL;
    int x;                                  //设数据元素的类型为 int
    scanf("%d",&x);
    while(x! = flag)
      { s = malloc(sizeof(Lnode));  s ->data = x;
        if  (L == NULL)  L = s;             //第一个结点的处理
        else  r ->next = s;                 //其他结点的处理
        r = s;                              //r 指向新的尾结点
        scanf("%d",&x);
      }
if(r! = NULL)r ->next = NULL;               //对于非空表,最后结点的指针域放空指针
    return L;
}
```

在上面的算法中,第一个结点的处理和其他结点是不同的,原因是第一个结点加入时链表为空,它没有直接前驱结点,它的地址就是整个链表的指针,需要放在链表的头指针变量中;而其他结点有直接前驱结点,其地址放入直接前驱结点的指针域。"第一个结点"的问题在很多操作中都会遇到,如在链表中插入结点时,将结点插在第一个位置和其他位置是不同的,在链表中删除结点时,删除第一个结点和其他结点的处理也是不同的。为了方便操作,有时在链表的头部加入一个"头结点",头结点的类型与数据结点一致,标识链表的头指针变量 L 中存放该结点的地址,这样即使是空表,头指针变量 L 也不为空。头结点的加入使得"第一个结点"的问题不再存在,也使得"空表"和"非空表"的处理成为一致。

头结点的加入完全是为了运算的方便,它的数据域无定义,指针域中存放的是第一个数据结点的地址,空表时为空。

2. 求表长

(1) 设 L 是带头结点的单链表(线性表的长度不包括头结点)。

算法如下:

算法 1-3

```
int Length_LinkList1(LinkList L)
  { Lnode * p = L;                  //p 指向头结点
    int j = 0;
    while(p ->next)
    { p = p ->next;j ++ }           //p 所指的是第 j 个结点
```

```
    return j;
  }
```

（2）设 L 是不带头结点的单链表。

算法如下：

算法 1-4

```
int Length_LinkList2(LinkList L)
    {  Lnode *p=L;
       int j;
       if(p==NULL)return 0;          //空表的情况
       j=1;                          //在非空表的情况下,p所指的是第一个结点
       while(p->next)
        {p=p->next; j++;}
       return j;
    }
```

从上面两个算法中看到，不带头结点的单链表空表情况要单独处理，而带上头结点之后则不用。在以后的算法中若不加说明则认为单链表是带头结点的。算法 1-3 和算法 1-4 的时间复杂度均为 $O(n)$。

3. 查找操作

（1）按序号查找 Get_LinkList(L,i)

算法思路：从链表的第一个元素结点起，判断当前结点是否是第 i 个。若是，则返回该结点的指针，否则继续后一个，表结束为止，没有第 i 个结点时返回空。

（2）按值查找即定位 Locate_LinkList(L,x)

算法思路：从链表的第一个元素结点起，判断当前结点其值是否等于 x。若是，返回该结点的指针，否则继续后一个，表结束为止，找不到时返回空。

按序号查找和按值查找的时间复杂度均为 $O(n)$。

4. 插入

（1）后插结点：设 p 指向单链表中某结点，s 指向待插入的值为 x 的新结点，将 $*s$ 插入到 $*p$ 的后面，插入示意图如图 1-2 所示。

操作如下：

① s->next=p->next;

② p->next=s;

注意：两个指针的操作顺序不能交换。

（2）前插结点：设 p 指向链表中某结点，s 指向待插入的值为 x 的新结点，将 $*s$ 插入到 $*p$ 的前面，插入示意图如图 1-3 所示。与后插不同的是：首先要找到 $*p$ 的前驱 $*q$，然后再完成在 $*q$ 之后插入 $*s$，设单链表头指针为 L，操作如下：

```
q=L;
while(q->next!=p)
  q=q->next;          //找*p的直接前驱
```

```
s->next = q->next;
q->next = s;
```

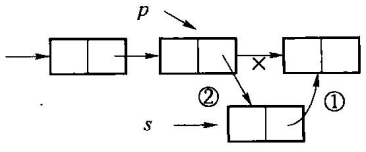


图 1-2　在 * p 之后插入 * s

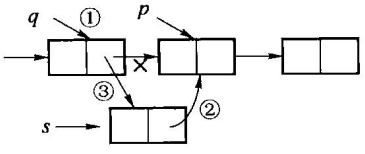


图 1-3　在 * p 之前插入 * s

后插操作的时间复杂性为 $O(1)$，前插操作因为要找 * p 的前驱，时间性能为 $O(n)$；其实我们关心的更是数据元素之间的逻辑关系，所以仍然可以将 * s 插入到 * p 的后面，然后将 p->data 与 s->data 交换即可，这样既满足了逻辑关系，也能使得时间复杂性为 $O(1)$。

(3) 插入运算 Insert_LinkList(L,i,x)

算法思路：① 找到第 $i-1$ 个结点；若存在继续②，否则结束；

② 申请、填装新结点；

③ 将新结点插入，结束。

5. 删除

(1) 删除结点：设 p 指向单链表中某结点，删除 * p。操作示意图如图 1-4 所示。通过示意图可见，要实现对结点 * p 的删除，首先要找到 * p 的前驱结点 * q，然后完成指针的操作即可。指针的操作由下列语句实现：

```
q->next = p->next;
free(p);
```

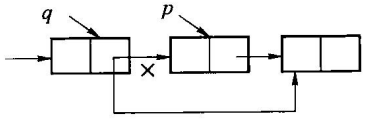


图 1-4　删除 * p

显然找 * p 前驱的时间复杂性为 $O(n)$。

若要删除 * p 的后继结点(假设存在)，则可以直接完成：

```
s = p->next;
p->next = s->next;
free(s);
```

该操作的时间复杂性为 $O(1)$。

(2) 删除运算：Del_LinkList(L,i)

算法思路：① 找到第 $i-1$ 个结点；若存在继续②，否则结束；

② 若存在第 i 个结点则继续③，否则结束；

③ 删除第 i 个结点，结束。

通过上面的基本操作我们得知：

在单链表上插入、删除一个结点，必须知道其前驱结点。单链表不具有按序号随机访问的特点，只能从头指针开始一个个顺序进行。

1.4.3　循环链表

对于单链表而言，最后一个结点的指针域是空指针，如果将该链表头指针置入该指针域，则使得链表头尾结点相连，就构成了单循环链表。如图 1-5 所示。