高等教育自学考试计算机及应用专业(独立本科段)自学辅导丛书

数据结构自学辅导

苏仕华　主编

清华大学出版社

(京)新登字 158 号

内 容 简 介

本书依据数据结构自学考试大纲编写，全书共分 12 章。第 1 章至第 10 章的章节结构与考试大纲对应，每章均先介绍知识点和学习方法，再进行重点与难点分析，最后给出自测练习及其答案；第 11 章是复习应试指南，对全书进行系统复习；第 12 章是模拟试题及参考答案。

本书是高等教育自学考试计算机及应用专业(独立本科段)自学辅导丛书之一，旨在帮助考生充分把握考试内容和题型，做好应试准备，考取理想成绩。本书不仅可供参加自学考试的考生学习课程使用，也可作为大专院校及社会上数据结构学习者的参考用书。

图书在版编目(CIP)数据

数据结构自学辅导/苏仕华主编. —北京：清华大学出版社，2002
(高等教育自学考试计算机及应用专业(独立本科段)自学辅导丛书)
ISBN 7-302-05547-5

Ⅰ. 数… Ⅱ. 苏… Ⅲ. 数据结构-高等教育-自学考试-自学参考资料
Ⅳ. TP311.12

中国版本图书馆 CIP 数据核字(2002)第 040547 号

出 版 者：清华大学出版社(北京清华大学学研大厦，邮编 100084)
http:// www. tup. tsinghua. edu. cn
责任编辑：张 民
印 刷 者：北京市人民文学印刷厂
发 行 者：新华书店总店北京发行所
开 本：787×1092 1/16 **印张**：10 **字数**：229 千字
版 次：2002 年 8 月第 1 版 2002 年 8 月第 1 次印刷
书 号：ISBN 7-302-05547-5/TP・3270
印 数：0001～5000
定 价：15.00 元

前　言

“数据结构”是计算机专业的必修、主干课程之一，它旨在使读者学会分析计算机加工的数据对象的特性，学会数据的组织方法，以便选择合适的数据逻辑结构和数据存储结构，以及相应的运算（操作），能把现实世界中的问题转化为在计算机内部的表示和处理，这是一个良好的程序设计技能进一步训练的过程。在整个教学或自学过程中，解题能力和技巧的训练是一个重要的环节。为了帮助学习这门课程特别是参加自学考试的读者更好地掌握知识，应对考试，特编写本书。

作者在长期讲授“数据结构与算法”这门课的过程中体会到，读者对书本中知识的理解和掌握并不困难，因为书中有一些内容与日常生活中的现象很接近，如顺序表、队列、某些查找和排序方法等，但对于如何利用这些基本知识和方法解答一些问题以及对问题的求解进行算法设计则感到难以下手。实践证明，从理解课程内容到能够较好地解答习题还有很长的一段路要走，而算法设计完成的质量与基本的程序设计的素质培养是密切相关的。要想理解和巩固所学的基本概念、原理和方法，牢固地掌握所学的基本知识、基本技能，达到融会贯通、举一反三的目的，就必须多做、多练、多见（见多识广），有条件的读者应尽可能多地上机实践，将本书中的一些算法作适当改变就可以成为能够运行的程序。经过循序渐进的训练，就可以掌握许多程序设计的技巧和方法。

为了提高应试能力，我们按照大纲的要求，对教材中的重点、难点进行了分析，收集、整理和编写了大量的例题、习题，这些习题内容丰富、涉及面广、难易适当，很有实用价值；另外还编写了包含各种题型的模拟试题，这些题目具有很强的针对性。学生或考生不仅要掌握辅导材料的题型变化，还要掌握分析问题的思想方法。只要考生能按照要求理解并掌握各种题型的分析解题方法，就能提高应变能力。为了帮助自学“数据结构”这门课程的读者，启发其思路，开阔其视野，对大部分题目都做了解析，所有习题及试题都给出了参考答案。

本书从第 1 章至第 10 章的结构均与教材和大纲一致，每章先介绍知识点和学习方法，再进行重点与难点分析，最后给出自测练习及其答案；第 11 章是复习应试指南，对全书进行了系统的总结和复习；第 12 章是模拟试题及参考答案。

本书由苏仕华主编，参加本书编写工作的还有郭草敏、吴河辉、经纶、贾伯琪、黄学俊、余华敏等。

在本书的编写过程中，得到了中国科学技术大学计算机系主任黄刘生教授和自动化系刘振安教授的支持和帮助，并对本书的编写提出了许多宝贵的意见。另外本书还参考了大量的书籍和资料。笔者在此一并致以诚挚的谢意。

由于作者水平有限、时间仓促，书中难免存在一些缺点和错误，殷切希望广大读者及同行们批评指正。

编　者

2002 年 2 月于合肥

目　录

第1章 概 论

本章主要介绍了数据、数据元素、数据对象、数据结构、存储结构和数据类型等概念的含义；算法设计的基本要求；算法描述、算法分析以及估算算法时间复杂度的方法。

1.1 知识点和学习方法

大纲对本章内容的要求不是很高，除了“识记”就是“领会”。具体要求如下：

“识记”层次：(1) 数据、数据元素、数据项、数据结构等基本概念；(2) 数据结构的逻辑结构、存储结构及数据运算的含义及其相互关系；(3) 数据结构的两大类逻辑结构和四种常用的存储表示方法；(4) 数据结构在各种软件系统中所起的作用；(5) 选择合适的数据结构是解决应用问题的关键步骤。

“领会”层次：(1) 算法、算法的时间复杂度和空间复杂度、最坏情况下的时间复杂度和平均时间复杂度等概念；(2) 算法的时间复杂度不仅仅依赖于问题的规模，也取决于输入实例的初始状态；(3) 算法描述和算法分析的方法，对于一般算法能分析出时间复杂度。

对本章的学习，主要是要熟悉各名词和术语的含义；掌握各种基本概念，特别是数据结构的逻辑结构、存储结构、数据运算3方面的内容以及这3方面的相互关系；熟悉C语言的书写规范，理解算法的5个要素的确切含义，即有穷性、确定性、可行性及有输入、有输出，从而掌握计算语句频度和估算算法时间复杂度的方法等，为学习数据结构打下基础。

1.2 重点与难点分析

本章的重点是熟悉各种基本概念，了解数据结构的逻辑结构、存储结构及数据运算3方面的概念及相互关系；其难点是算法复杂度的分析方法。下面将通过例题来加深对算法分析和算法复杂度计算的理解。

对于较复杂的算法，可以将它分成几个容易估算的部分，然后利用“O”的求和原则和乘法原则计算整个算法的时间复杂度。

大“O”下的求和准则：若算法的两部分的时间复杂度为 $T_1(n)=O(f(n))$ 和 $T_2(n)=O(g(n))$，则 $T_1+T_2=O(\max(f(n),g(n)))$。又若 $T_1(m)=O(f(m))$，$T_2(n)=O(g(n))$，则 $T_1+T_2=O(f(m)+g(n))$。

大“O”下的乘法准则：若算法的两部分的时间复杂度为 $T_1(n)=O(f(n))$，$T_2(n)=O(g(n))$，则 $T_1\cdot T_2=O(f(n)\cdot g(n))$。

【例 1.1】 分析计算下面程序段的时间复杂度。

```
(1)   s=0;
(2)   for (i=1;i<=n;i++)
(3)     for (j=1;j<=n;j++)
(4)       s=s+1;
```

分析：按常规求算法时间复杂度，该算法的(1) 行频度为 1；(2)、(3)、(4)行的频度分别为 $n+1$，$n(n+1)$和 n^2，因此算法中所有频度之和为：$T(n)=n^2+(n+1)+n(n+1)+1=2n^2+2n+2$，即 $f(n)=n^2$，所以，该算法的时间复杂度为 $T(n)=O(f(n))=O(n^2)$。

我们知道，执行一条赋值语句与 n 无关，时间复杂度为 O(1)，因此 $T_4(n)=O(1)$，对于第(3)条语句，$T_3(n)=O(n)$，如果利用上面的求和准则，有 $T_4(n)+T_3(n)=O(n)$，第(2)条语句 $T_2(n)=O(n)$，而(2)与(3)、(4)是循环嵌套，故有 $T_2(n)*(T_3(n)+T_4(n))=O(n^2)$，对第(1)条语句 $T_1(n)=O(1)$，它与其他语句之间的关系是顺序执行，所以该程序段的算法时间复杂度是：$T_1(n)+T_2(n)*(T_3(n)+T_4(n))=O(n^2)$，与前面常规方法求的值完全一样。

【例 1.2】 求下面程序段的算法时间复杂度。

```
x=0;
for (i=2;i<=n;i++)
  for (j=2;j<=i-1;j++)
    x=x+1;
```

分析：由于算法的时间复杂度考虑的只是对于问题规模 n 的增长率，则在难精确计算基本操作执行次数(或语句频度)的情况下，只需要求出它关于 n 的增长率或阶即可。因此，上述语句 x=x+1 执行次数关于 n 的增长率为 n^2，它是语句频度表达式 $(n-1)(n-2)/2$中增长最快的项，所以该程序段的算法时间复杂度为 $O(n^2)$。

【例 1.3】 求下面的冒泡排序算法的时间复杂度。

```
change=1;
for (i=1;i<=n&&change;i++)  {
  change=0;
  for (j=1;j<=n-i;j++)
    if (a[j]>a[j+1]) { a[j]与 a[j+1]交换;change=1;}
  }
```

分析：冒泡排序是以“交换序列中相邻两个整数”为基本操作。当 a 中元素的初始序列为自小至大有序，基本操作的执行次数为 0，但循环还是执行了 n-1 次；当初始序列为自大至小有序时，基本操作的执行次数为 $n(n-1)/2$。对这类算法的分析，一种解决的办法是计算它的平均值，即考虑它对所有可能的输入数据集的期望值，此时相应的时间复杂度为算法的平均时间复杂度。如假设 a 中初始输入数据可能出现 n!种的排列情况的概率相等，则冒泡排序的平均时间复杂度 $T_{avg}(n)=O(n^2)$；然而，在很多情况下，各种输入数据集出现的概率难以确定，算法的平均时间复杂度也就难以确定。因此，另一种更可行的也是更常用的办法是讨论算法在最坏情况下的时间复杂度。例如，上述冒泡排序的最坏情况为 a 中初始序

列为自大至小有序，则冒泡排序算法在最坏情况下的时间复杂度为 $T(n)=O(n^2)$。在本书以后各章中讨论的时间复杂度，除特别指明外，都是指最坏情况下的时间复杂度。

【例 1.4】 计算求一个正整数是否为素数的算法的时间复杂度。

```
i=2;
while ( ( n / i !=0 ) && ( i < sqrt ( n ) ) )
  i ++;
if ( i > sqrt ( n ) )
  printf("%d is prime", n) ;
else
  printf("%d is not prime",n) ;
```

分析：算法的时间复杂度是由嵌套最深层语句的频度决定的，而求素数算法的嵌套最深层语句(即 i++)的频度则由条件((n/i!=0)&&(i<sqrt(n))决定，显然 i<sqrt(n)，即执行频度小于 sqrt(n)，所以其时间复杂度是 $O(\sqrt{n})$。

【例 1.5】 分析计算求 n! 的递归函数的时间复杂度。

```
float fact (int n)
  {
    if (n<=1)
      return 1;                ①
    else
      return n * fact(n-1);    ②
  }
```

分析：设 fact(n)的运行时间函数是 T(n)。该算法中语句①的运行时间是 O(1)，语句②的运行时间是 T(n−1)+O(1)。因此有：

$$T(n)=\begin{cases}O(n) & n\leqslant 1\\ T(n-1)+O(1) & n>1\end{cases}$$

所以有：

$$\begin{aligned}T(n)&=O(1)+T(n-1)\\&=2*O(1)+t(n-2)\\&\cdots\\&=(n-1)*O(1)+T(1)\\&=n*O(1)\\&=O(n)\end{aligned}$$

即算法 fact(n)的时间复杂度为 O(n)。

1.3 自测练习

1. 选择题。在下列各题的被选答案中选出一个正确答案，并将其代号(A,B,C,D 中的一个)填写在题目后面的括号内。

(1) 在数据结构中，从逻辑上可以把数据结构分为(　)。

A. 紧凑结构和非紧凑结构　　B. 线性结构和非线性结构

C. 内部结构和外部结构　　D. 动态结构和静态结构

(2) 线性表的顺序存储结构是一种(　)的存储结构。

A. 顺序存取　　B. 索引存取　　C. 随机存取　　D. 散列存取

(3) 算法分析的两个主要方面是(　)。

A. 正确性和简明性　　B. 数据复杂性和程序复杂性

C. 可读性和可维护性　　D. 时间复杂性和空间复杂性

(4) 线性表若采用链式存储结构存储时，要求内存中可用存储单元地址(　)。

A. 不一定连续的　　B. 部分地址必须是连续的

C. 必须是连续的　　D. 一定是不连续的

2. 填空题。在题目的空处，按标号填上适当的内容。

(1) 数据结构一般包括__①__、__②__和数据运算三个方面的内容。

(2) 数据的逻辑结构可分为__①__、__②__两大类。

(3) 数据的存储结构(物理结构)可以用__①__、__②__、__③__及散列存储等四种存储方法表示。

(4) 选用算法除了首先考虑“正确的”外，主要考虑__①__、__②__、__③__等三点。

3. 设 n 为正整数，分别写出下面各程序段的时间复杂度。

(1)
```
for (i=1;i<=n;i++)
  {
    y=y+1;
    for (j=1;j<=2*n;j++)
      x=x+1;
  }
```

(2)
```
s=0;
while (n>=(s+1)*(s+1))
    s=s+1;
```

(3)
```
x=1; sum=0;
for (i=1; i<=n; i++)
  {
    x=x*i;
    sum=sum+x;
  }
```

1.4　自测练习答案

1. 选择题解答：

(1) B. 线性结构和非线性结构

(2) C. 随机存取

(3) D. 时间复杂性和空间复杂性

(4) A. 不一定连续的

2. 填空题解答：

(1) ① 逻辑结构　　② 存储结构

(2) ① 线性结构　　② 非线性结构(又可分为树形结构和图形结构)

(3) ① 顺序存储　　② 链式存储　　③ 索引存储

(4) ① 执行算法所需要的时间

② 执行算法所需要的存储空间

③ 算法应易于理解，易于编程，易于调试。

3. 分析算法时间复杂度习题解答：

(1) 其中语句 y=y+1 的频度是 n−1，语句 x=x+1 的频度是$(n-1)(2n+1)=2n^2-n-1$。所以该程序段的算法时间复杂度是：$T(n)=O(2n^2-n-1)=O(n^2)$。

(2) 在该程序段中最深层的语句是 s=s+1，该语句的执行频度是依赖于循环判断条件：n>=(s+1)*(s+1)，这个条件判断句可以改成：(s+1)*(s+1)<=n，就相当于语句：$(s+1)^2<=n$(即$(s+1)<=\sqrt{n}$)，因此，该程序段的算法时间复杂度为：$O(\sqrt{n})$。

(3) 第 3 个程序段中嵌套最深层语句是：x=x+1；sum=sum+x，它们的频度为 n 次，所以其算法时间复杂度是 O(n)。

第2章 线 性 表

本章主要介绍线性表的逻辑结构和各种存储表示方法，以及定义在逻辑结构上的各种基本运算及其在存储结构上如何实现这些基本运算。要求在熟悉这些内容的基础上，能够针对具体应用问题的要求和性质，选择合适的存储结构设计出相应的有效算法，解决与线性表相关的实际问题。

2.1 知识点和学习方法

本章需要考核的知识点如下：

“识记”层次：(1) 线性表的逻辑结构特征；(2) 线性表上定义的基本运算，并能利用基本运算构造出较复杂的运算。

“领会”层次：(1) 顺序表和链表的主要优缺点；(2) 针对线性表上所需要执行的主要操作，知道选择顺序表还是链表作为其存储结构才能取得较优的时空性能。

“综合应用”层次：(1) 顺序表的含义及特点，即顺序表如何反映线性表中元素之间的逻辑关系；(2) 顺序表上的插入、删除操作及其平均时间性能分析；(3) 利用顺序表设计算法解决简单的应用问题；(4) 链表如何表示线性表中元素之间的逻辑关系；(5) 链表中头指针和头结点的使用；(6) 单链表、双链表、循环链表链接方式上的区别；(7) 单链表上实现的建表、查找、插入和删除等基本算法，并分析其时间复杂度；(8) 循环表上尾指针取代头指针的作用，以及单循环链表上的算法与单链表上相应算法的异同点；(9) 双链表的定义及其相关的算法；(10) 利用链表设计算法解决简单的应用问题。

本章内容是全书的学习重点，对它掌握的好坏直接影响后面许多章节的学习，因此本章大多数内容要求达到“综合应用层次”。对本章的学习，主要是通过对线性表的顺序存储和链式存储结构及在其基础上的基本运算算法的理解，并通过实例提高设计算法解决简单应用问题的能力，为后面章节的学习，特别是第 3 章“栈和队列”的学习打下良好的基础。

2.2 重点与难点分析

本章的重点是熟练掌握顺序表(即线性表的顺序存储，在后面的内容中不再说明)和单链表上实现的各种基本算法以及相关的时间复杂度分析，难点是能够使用本章所学到的基本知识设计有效算法，解决与线性表相关的应用问题。下面举例来加深对这些重点难点的分析和理解。

【例 2.1】 已知顺序表 L 递增有序，试写一算法删除顺序表中值重复多余的元素，使

得所得结果表中各元素值均不相同。

分析：由于顺序表是递增有序，值相同的元素必为相邻的元素，因此依次比较相邻两个元素，若值相等，则删除其中一个，否则继续向后查找，直至搜索完毕，其算法如下：

```
void DeleteDP(SeqList * L)
  {
      int i,j;
      i=1;
      while(i<=L->length-1)
        if(L->data[i]!=L->data[i+1])
            i++;  /*元素值不相等,继续向下找 */
        else
            {
              for(j=i+2; j<=L->length; j++)
                L->data[j-1]=L->data[j];  /*删除第 i+1 个元素 */
              L->length=L->length-1;  /*表长减 1 */
            }
  }
```

【例 2.2】 已知线性表的长度为 n，试写一算法将线性表逆置。

分析：可以设线性表为$(a_1,a_2,\cdots,a_n)$，逆置之后为$(a_n,a_{n-1},\cdots,a_1)$，并且表以顺序存储方式存储。实现其算法的基本思想是：先以表长的一半为循环控制次数，将表中最后一个元素同顺数第一个元素交换，将倒数第二个元素同顺数第二个元素交换，……，直至交换完为止，其算法如下：

```
void Converts(SeqList * A)
  {
    DataType x;
    int i,k;
    k=A->length/2;
    for(i=1;i<=k;i++)
      {
        x=A->data[i];
        A->data[i]=A->data[A->length-i+1];
        A->data[A->length-i+1]=x;
      }
  }
```

这个算法只需要进行数据元素的交换操作，其主要时间花在 for 循环上，因此整个算法的时间复杂度显然为 O(n)。

【例 2.3】 试写一算法，实现在线性表中找出最大值和最小值的元素及其所在位置。

分析：如果在查找最大值和最小值的元素时各扫描一遍所有元素，则至少要比较 2n 次，但也可以使用一遍扫描同时找出最大值和最小值的元素。实现本功能的算法如下：

```
void MaxMin(SeqList * L,DatdType &max,DataType
            &min,int &k,int &j)
  {
    int i;
    max=L->data[1]; min=L->data[1];
    k=j=1; /*先假设第一个元素既是最大值,也是最小值 */
    for(i=2;i<=L->length;i++)
      if(L->data[i]>max)
        {
          max=L->data[i]; k=i;
        }
      else
        if(L->data[i]<min)
          {
            min=L->data[i]; j=i;
          }
  }
```

在这个算法中,最坏情况是线性表的元素递减有序排列,这时 L->data[i]>max 条件均不成立,此时比较的次数为 n-1 次(设表长为 n),另外每次都要比较 L->data[i]<min,同样需要比较 n-1 次,因此,总的比较次数为:2(n-1)。

最好的情况是表的元素按递增有序排列,这时(L->data[i]>max)条件均成立,不会再执行 else 后的比较,所以总的比较次数为:n-1。

那么,该算法的平均比较次数为:(2(n-1)+(n-1))/2=3(n-1)/2。

因此,它的时间复杂度应为:O(n)。

【例 2.4】 试写一个算法,将一个头结点指针为 a 的单链表 A 分解成两个单链表 A 和 B,其中头结点指针分别为 a 和 b,使得 A 链表中含有原链表 A 中序号为奇数的元素,而 B 链表中含有原链表中序号为偶数的元素,并保持原来的相对顺序。

分析:根据题目要求,需要遍历整个链表 A,当遇到序号为偶数的结点时,先将其删除,并在删除时把这些结点链接到 B 表中,一直到遍历结束。其实现算法如下:

```
void(LinkList a,LinkList b)
  { /*按序号奇偶分解单链表*/
    LinkNode *p,*q,*r;
    p=a;
    b=a->next;
    r=b;
    while(p<>NULL && p->next<>NULL)
      {
        q=p->next;  /*q 指向偶数序号的结点 */
        p->next=q->next;  /*将 q 从原 A 中删除 */
        r->next=q;  /*将 q 结点连接到 B 链表的末尾 */
        r=q;  /*r 总是指向 B 链表的最后一个结点 */
```

```
        p=p->next;  /* p 指向原链表 A 中的奇数序号的结点 */
      }
    r->next=NULL;
  }
```

这个算法要从头至尾扫描整个链表，所以它的时间复杂度是 O(n)。

【例 2.5】 已知有一个单链表 L(至少有一个结点)，其头指针为 head，试写一个算法将该单链表逆置，即最后一个结点变成第一个结点，原来倒数第二个结点变成第二个结点，如此等等。

分析：这个题目与例 2.2 的顺序表逆置要求是一样的，只不过是线性表的存储结构不同，但对链式结构的逆置比顺序表的逆置要困难得多。本题的算法思想是：从头到尾扫描单链表 L，将第一个结点的 next 域置为 NULL，将第二个结点的 next 域指向第一个结点，将第三个结点的 next 域指向第二个结点，如此等等，直到最后一个结点，使用 head 指向它，这样就达到了本题的要求。实现本题功能的算法如下：

```
void Converts(LinkList head)
  { /* 单链表的逆置 */
    ListNode *p, *q, *r;
    p=head;
    q=p->next;
    p->next=NULL; /* 第一个结点的 next 域置为空值 NULL */
    while(q<>NULL) /* 当表 L 没有后继结点时为止 */
      { r=q->next;
        q->next=p; /* 后一个结点的 next 域指向前一个结点 */
        p=q;
        q=r;
      }
    head=p; /* head 指向最后一个结点 */
  }
```

【例 2.6】 已知有一个结点数据域为整型的，而且是按从大到小顺序排列的循环链表(L 为头结点指针，表非空)，试写一算法插入一个结点 S(其数据域为 X)至循环链表的适当位置，使之保持链表的有序性。

分析：要解决这个算法问题，首先要弄清楚循环链表和单链表的区别，判断单链表是否访问结束，是看结点的指针域是否等于 NULL，而对于循环链表则是看结点的指针域是否等于头指针。因此，实现本题功能的算法如下：

```
void InsertList(LinkList L,int x)
  { /* 将值为 X 的新结点插入到有序循环链表中适当的位置 */
    ListNode *s , *p, *q ;
    s=(ListNode *)malloc(sizeof(ListNode));/* 申请存储空间 */
    s->data=x;
    p=L;
```

```
    q=p->next; /*q指向开始结点*/
    while(q->data > x && q!=L)
      { /*查找插入位置 */
        p=p->next; /*p指向双亲结点*/
        q=p->next; /*q指向当前结点*/
      }
    p->next=s; /*插入s结点 */
    s->next=q;
}
```

该算法在最好情况下，即插入在第一个结点前面，循环一次也不执行；在最坏情况下，即插在最后一个结点之后，循环执行 n 次，因此，该算法的时间复杂度为 O(n)。

【例 2.7】 假设有一个循环双链表，其结点类型结构包括 3 个域：prior、data 和 next，其中 data 为数据域，next 为指针域，指向其后继结点，prior 也为指针域，其值为空(NULL)，因此，该双链表其实是一个单循环链表。试写一算法，将其修改为真正的双链表。

分析：该题目的要求，其实就是将表中的每个 prior 域填上相应的前驱结点地址。假设该链表的头指针为 head，实现本题要求的算法如下：

```
void trans(DlinkList head)
  {
    DlistNode *p, *q;
    p=head->next;/*使p指向下一个结点*/
    q=head;/*使q指向p的前一个结点*/
    while(p!=head)
      {/*依次从左向右访问每个结点，对每个结点的prior赋值*/
        p->prior=q; /*p的前驱结点为q所指向的结点*/
        q=p; /*q指向当前结点*/
        p=p->next; /*p指针指向下一个结点*/
      }
    head->prior=q;
      /*循环结束后，q指向最后一个结点，再将head的prior域指向最后一个结点*/
  }
```

2.3 自测练习

1. 选择题。在下列各题的被选答案中选出一个正确答案，并将其代号(A，B，C，D 中的一个)填写在题目后面的括号内。

(1) 如果一个顺序表中第一个元素的存储地址为 1000，每个元素占 4 个地址单元，那么，第 6 个元素的存储地址应是(　)。

A. 1020　　B. 1010　　C. 1016　　D. 1024

(2) 带头结点的单链表(以 head 为头指针)为空的判断条件是(　)。

A. head!=NULL　　B. head->next==head