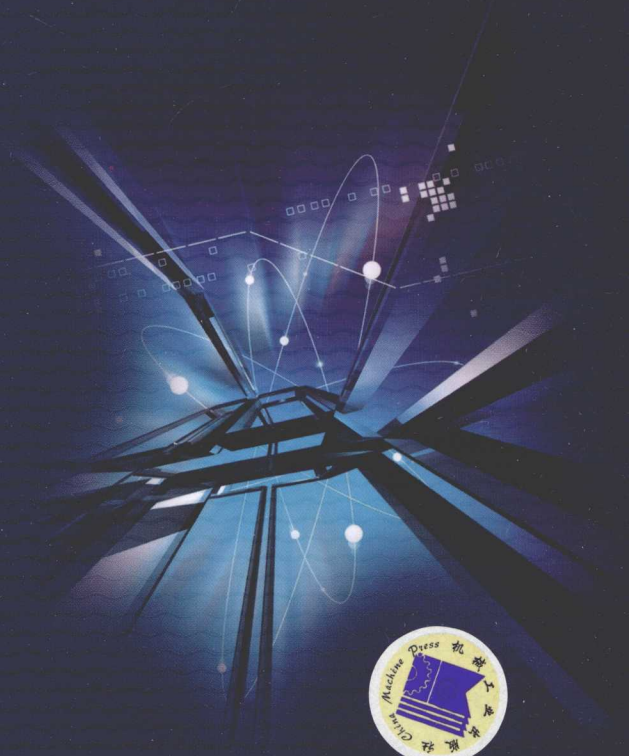
高等院校规划教材
计算机科学与技术系列

数据结构（C++版）习题解答与实验指导

马桂媛　吴小平　编著



机械工业出版社
CHINA MACHINE PRESS

高等院校规划教材·计算机科学与技术系列

数据结构（C++版）习题解答与实验指导

马桂媛　吴小平　编著

机械工业出版社

本书是《数据结构（C++版）》（ISBN 978-7-111-27794-1）的配套教学参考书。全书分为习题解答与实验指导两大部分。习题解答部分对各章的关键知识点及重要算法思想进行了梳理，并对主教材每章的习题作了较为完整的解答。实验指导部分提供了各种数据结构常见的一些实验题目，并对每个实验题目给出了提示。

本书可作为计算机及相关专业“数据结构”课程的参考用书。

图书在版编目（CIP）数据

数据结构（C++版）习题解答与实验指导/马桂媛，吴小平编著．—北京：机械工业出版社，2009.8

（高等院校规划教材·计算机科学与技术系列）

ISBN 978-7-111-27827-6

Ⅰ．数…　Ⅱ．①马…　②吴…　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．TP311.12　TP312

中国版本图书馆CIP数据核字（2009）第125312号

机械工业出版社（北京市百万庄大街22号　邮政编码100037）

责任编辑：陈　皓　常建丽

责任印制：杨　曦

北京富生印刷厂印刷

2009年8月·第1版第1次印刷

184mm×260mm·10.5印张·259千字

0001—3000册

标准书号：ISBN 978-7-111-27827-6

定价：20.00元

凡购本书，如有缺页，倒页、脱页，由本社发行部调换

销售服务热线电话：（010）68326294　68993821

购书热线电话：（010）88379639　88379641　88379643

编辑热线电话：（010）88379753　88379739

出 版 说 明

计算机技术的发展极大地促进了现代科学技术的发展，明显地加快了社会发展的进程。因此，各国都非常重视计算机教育。

近年来，随着我国信息化建设的全面推进和高等教育的蓬勃发展，高等院校的计算机教育模式也在不断改革，计算机学科的课程体系和教学内容更加科学和合理，计算机教材建设逐渐成熟。在“十五”期间，机械工业出版社组织出版了大量的计算机教材，包括“21 世纪高等院校计算机教材系列”、“21 世纪重点大学规划教材”、“高等院校计算机科学与技术‘十五’规划教材”、“21 世纪高等院校应用型规划教材”等，均取得了可喜成果，其中多个品种的教材被评为国家级、省部级的精品教材。

为了进一步满足计算机教育的需求，机械工业出版社策划开发了“高等院校规划教材”。这套教材是在总结我社以往计算机教材出版经验的基础上策划的，同时借鉴了其他出版社同类教材的优点，对我社已有的计算机教材资源进行整合，旨在大幅提高教材质量。我们邀请多所高校的计算机专家、教师及教务部门针对此次计算机教材建设进行了充分的研讨，达成了许多共识，并由此形成了“高等院校规划教材”的体系架构与编写原则，以保证本套教材与各高等院校的办学层次、学科设置和人才培养模式等相匹配，以满足其计算机教学的需要。

本套教材包括计算机科学与技术、软件工程、网络工程、信息管理与信息系统、计算机应用技术以及计算机基础教育等系列。其中，计算机科学与技术系列、软件工程系列、网络工程系列和信息管理与信息系统系列是针对高校相应专业方向的课程设置而组织编写的，体系完整，讲解透彻；计算机应用技术系列是针对计算机应用类课程而组织编写的，着重培养学生利用计算机技术解决实际问题的能力；计算机基础教育系列是为大学公共基础课层面的计算机基础教学而设计的，采用通俗易懂的方法讲解计算机的基础理论、常用技术及应用。

本套教材的内容源自致力于教学与科研一线的骨干教师与资深专家的实践经验和研究成果，融合了先进的教学理念，涵盖了计算机领域的核心理论和最新的应用技术，真正在教材体系、内容和方法上做到了创新。同时本套教材根据实际需要配有电子教案、实验指导或多媒体光盘等教学资源，实现了教材的“立体化”建设。本套教材将随着计算机技术的进步和计算机应用领域的扩展而及时改版，并及时吸纳新兴课程和特色课程的教材。我们将努力把这套教材打造成为国家级或省部级精品教材，为高等院校的计算机教育提供更好的服务。

对于本套教材的组织出版工作，希望计算机教育界的专家和老师能提出宝贵的意见和建议。衷心感谢计算机教育工作者和广大读者的支持与帮助！

机械工业出版社

前　言

“数据结构”是计算机及相关专业的核心课程，是一门理论与实践并重的课程。该门课程的主要任务是，研究现实世界中各种数据对象的逻辑结构，及其在计算机中的存储表示以及在不同存储结构上的相应算法，并掌握算法的时间分析技术和空间分析技术。

学习“数据结构”除了需要掌握基本概念、基本理论和基本算法以外，更重要的是能通过对实际问题的分析和抽象，达到为实际问题设计合适的数据结构与算法，进而编写出正确的程序等目的。完成习题与上机实验正是巩固基础知识，将理论知识和实际应用有机结合，训练学生的程序设计技能的两个至关重要的环节。

尽管在学习“数据结构”课程时实践与理论同等重要，但是，不少学生虽对课堂讲授的理论和算法都能理解，可一遇到实际问题就无从下手，更不知如何编程实现了。本书旨在对数据结构知识点和算法进行梳理，在巩固数据结构的基本概念和基本理论的同时，针对具体问题给学生提供一种解决问题的思路及程序编写范例。

本书是《数据结构（C++版）》的配套习题解答与实验指导。全书分为习题解答与实验指导两部分。习题解答部分首先简要回顾了各章节的关键知识点及重要算法的思想，帮助学生梳理各章的重点和难点，然后对教材中的典型习题逐一进行了解答，并给出了较为完整的源程序，旨在为学生提供一个参考及解决问题的思路。实验指导部分提供了各种数据结构常见的一些实验题目，并对每个实验题目给出了简要的提示，供学生上机实验时参考。

书中的所有程序均在 Visual Studio.Net 2005 中调试通过。考虑到代码复用，所有程序均采用类模板和函数模板编写。限于篇幅，大部分习题只列出了关键函数的实现部分，而省略了相关类的定义及实现。

由于作者水平有限，书中难免有不妥之处，敬请读者批评指正。

编　者

目　录

第 2 部分 实验指导

第 1 部分　习题解答

第1章　绪　　论

1.1　知识点回顾

1．基本概念

- 数据、数据元素、数据对象、数据结构、存储结构、数据类型、抽象数据类型
- 数据结构（逻辑结构）：指相互之间存在一种或多种特定关系的数据元素的集合。根据数据元素之间关系的不同，数据结构可分为线性结构（一对一）、树（一对多）、图（多对多）。
- 存储结构：数据结构在计算机中的存储表示，不仅要存储元素，还要存储元素之间的关系。根据对元素之间关系的不同存储表示，可将存储结构分为顺序、链式、索引、散列存储结构。

2．算法

（1）算法的时间效率

语句频度：算法中基本操作（最内层循环的语句）的重复执行次数（是具体值）。

时间复杂度：若算法中基本操作的语句频度是问题规模 n 的函数 f(n)，则算法的时间复杂度为 T(n)=O(f(n))，表示随着问题规模 n 的增大，算法执行时间的增长率与 f(n)的增长率相同。即，当 $n\to\infty$ 时，只取 f(n)的最高次项（略去最高次项的系数及低次项）。

常见的算法时间复杂度有：$O(1)<O(\log_2 n)<O(n)<O(n^2)<O(2^n)$。

（2）算法的空间复杂度

算法的空间复杂度即算法中使用辅助存储空间的大小。

3．函数模板与类模板

（1）函数模板与模板函数

在程序设计中，常会遇到两个函数执行的操作功能完全相同，仅参数类型不同的情况，常用的解决办法是对该函数定义多个重载函数版本。例如，下面函数交换了两整型变量的值。

【例 1-1】

```
void Swap(int &x,int &y)
{ int z=x; x=y; y=z; }
```

但要实现双精度、字符型变量值的交换，则还需定义相应的重载函数，显然效率较低。若能像函数调用传递参数那样，在对不同数据做相同运算时，将数据类型亦作为类型参数传入，则效率将显著提高。使用函数模板编写函数就能实现上述功能。编译系统在调用该函数模板

时，会根据实参类型自动产生函数模板的特定类型版本，即模板函数。例如，若将例 1-1 所示函数改写为例 1-2 形式，则定义了一个通用的 Swap 函数模板，可以实现任何类型数据的交换。

【例 1-2】

```
template <class T>   void Swap(T &x,T &y)   // 函数模板的定义
{ T z=x; x=y; y=z; }
```

定义函数模板后，如何使用该函数模板呢？与普通函数调用一样，在调用该函数时传递不同类型的实参，由编译系统根据函数调用的实参类型，自动产生相应的模板函数。如：

```
int a1=32,a2=45;
double b1=2.3,b2=5.2;
Swap(a1,a2); Swap(b1,b2);
```

第一个函数调用，编译器将 T 实例化为 int 类型，产生了 int 类型的 Swap()模板函数。第二个函数调用，编译器将 T 实例化为 double 类型，产生 double 类型的 Swap()模板函数。总之，一切内置数据类型以及用户自定义类型（但必须支持或重载了函数模板中的相关运算）均可调用 Swap 函数实现两变量值的交换。

【函数模板定义的语法规则】：

```
template <class 模板形参表>                    // 或 template <typename 模板形参表>
函数返回类型 函数名(函数形参表)
{
    函数体;
}
```

函数模板的定义与普通函数的定义类同，只是在函数头前增加了 template <class T1,class T2>的模板形参表说明，该形参表罗列出了函数定义中要用到的类型或值，函数头和函数体都没变化，只是在函数体中可使用模板形参表中声明的形参 T1、T2 等。

模板形参既可是表示类型的类型形参，也可是表示常量等的非类型形参。

（2）类模板与模板类

类模板类同函数模板，定义了一个不局限于某种数据类型的通用类的模板。例如，若要定义一个数组类 Array，该类提供求数组最大值、最小值、排序、读取数组中第 i 个元素值等操作，而数组元素的类型可能是整型、双精度型、字符、字符串、用户自定义的类型（甚至是类），即只要使用类模板，就不必为每种类型建立一个类，只需定义一个类模板就可以了。该类模板的定义方式见例 1-3。

【例 1-3】

```
template <class T> class Array{
public:
   Array(int length);
   ~Array();
   T max();      //求数组元素的最大值
private:
```

```
    T *arr; //指向 T 类型数组，具体 T 的类型，取决于实例化该类时指定的具体实参类型
    int len;
};
```

1）类模板的定义方式。类模板的定义与类的定义完全相同，只是在类定义的前面增加了类模板形参声明。

```
template <class 模板形参列表>  或 template<typename 模板形参列表>
class 类名
{
    类的数据成员、成员函数的声明（或实现）;
}
```

2）类模板中成员函数的定义方式。若类模板的成员函数要在类模板以外定义，则必须加上与类模板相同的首部 template <class 模板形参列表>，同时在限定作用域的类名后面也要加上<模板形参列表>，但不需要关键字 class 或 typename。如下所示：

```
template <class T> Array<T>::Array(int length)  //注意类名后也要加<模板形参表>
{   arr=new T[length];
    len=length;
}
template <class T> T Array<T>::max( )
{   int max=0;
    for(int i=1;i<len;i++)
      if(arr[max]>arr[i]) max=i;
    return arr[max];
}
```

3）类模板的实例化（模板类）。定义好类模板 Array 之后，要得到整型、双精度型或用户自定义类型的 Array 类，需要实例化该类模板，即在类型名后为该类模板的形参指定实参，以得到不同类型版本的模板类，这点与函数模板不同（函数模板无需显式为模板形参指定类型实参）。如：

```
Array<int> a(6);
Array<double> b(10);
```

第一条语句实例化了类模板 Array，得到一个 int 类型的 Array 模板类的对象 a，此时 Array 类中的所有类型 T 被 int 替代。第二条语句用 double 类型实例化了类模板 Array，得到一个 double 类型的 Array 类对象 b，Array 类中的所有类型 T 被 double 替代。

4）类模板的非类型参数。在类模板的形参表中，除了使用类型参数外，还可使用非类型参数。非类型参数通常是类中要用到的常数，如 Array 类中的数组长度就可用非类型参数指定，如例 1-4 所示。

【例 1-4】

```
template <class T,int N> class Array {
public:
    Array();
```

```
    ~Array();
    T max();
private:
    T arr[N];
};
template <class T,int N> T Array<T>::max( )
{   int max=0;
      for(int i=1;i<N;i++)
      if(arr[max]>arr[i]) max=i;
      return arr[max];
}
```

实例化带非类型参数的类模板时，必须为非类型参数指定一个常数。如：

```
Array<int,6> a,b;          //声明了两个具有 6 个元素的 int 模板类 Array 的对象 a，b
Array<int,10> c;           //声明了一个具有 10 个元素的 int 模板类 Array 的对象 c
Array<double,6> d;         //声明了一个具有 6 个元素的 double 模板类 Array 的对象 d
```

1.2 习题及解答

1．解释以下基本概念：数据、数据结构、存储结构、数据类型、算法、算法时间复杂度。

2．求下述程序段中有下画线的语句的语句频度和时间复杂度。

（1）
```
i=1;
while(i<=n){
    x++;
    i+=2;
}
```

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

（3）
```
count=0;
for(i=1;i<=n;i++)
  for(j=1;j<=n;j++)
    for(k=1;k<=n;k++)
        count++;
```

（4）
```
for(i=n-1;i>0;i--)
  for(j=1;j<=i;j++)
    if(a[j]>a[j+1]) a[j]↔a[j+1]
```

解答：

（1）语句频度为$\left\lfloor \frac{n+1}{2} \right\rfloor$，时间复杂度为 O(n)。

（2）语句频度为$\frac{(n-2)(n-1)}{2}$，时间复杂度为 $O(n^2)$。

（3）语句频度为 n^3，时间复杂度为 $O(n^3)$。

（4）语句频度为$\frac{n(n-1)}{2}$，时间复杂度为 $O(n^2)$。

第2章 线 性 表

2.1 知识点回顾

1. 线性表的逻辑结构

定义：具有相同类型的一组数据元素所组成的序列。

- 表中元素具有相同类型。
- 元素之间具有严格的先后次序关系。

2. 线性表的顺序存储结构（顺序表）

（1）存储特点

1）用一组地址连续的存储单元依次存放表中的元素（通常采用数组实现）。

2）用物理位置上的相邻来表示线性表中元素逻辑关系上的相邻（前驱、后继）。若数组首地址为 $loc(a_1)$，每个元素占用 L 个存储单元，则线性表中第 i 个元素的存储位置为

$$loc(a_i) = loc(a_1)+(i-1)*L。$$

3）它是一种随机存取的存储结构。

4）插入、删除操作需大量移动元素，所以不适用于经常增、删的线性表。

（2）关键操作

1）插入操作须注意以下几点：

① 需判断插入位置的合法性：$1 \leqslant i \leqslant length+1$。

② 存储空间是否足够，否则需重新分配。

③ 从最后一个元素开始到第 i 个元素（共 n-i+1 个）均逐个后移，空出插入空间。

2）删除操作须注意以下几点：

① 判断删除位置的合法性：$1 \leqslant i \leqslant length$。

② 从第 i+1 个元素起直到最后一个元素（共 n-i 个）均逐个前移。

3. 线性表的链式存储结构（单链表）

（1）单链表的类定义

单链表的类定义有两种方式，一种为嵌套类，另一种为复合类，教材中采用后者。

（2）存储特点

1）用任意存储单元存储线性表中的元素（逻辑上相邻的元素，物理位置上可能并不相邻）。

2）为表示元素之间的关系，每个元素的存储都具有两个部分，即数据域（存放元素值本身）和指针域（存放当前元素的后继元素的地址）。

3）每个元素的存储地址都在其前驱结点的指针域（next）中，故要访问某个结点需找到其前驱。类推之，访问链表中任一结点，必须从头指针出发，顺序移动指针查找（p=p->next），故单链表是非随机存取的存储结构。

4）带表头附加结点的单链表表空的条件为 head->next=NULL。

5）单链表操作中的循环结束条件通常为：p==NULL 或 p->next==NULL。

（3）重要操作

1）插入（以在第 i 个结点之前插入元素为例）：

① 要在第 i 个结点之前插入元素，则需找到第 i 个结点的前驱结点 p（即第 i-1 号结点），如图 2-1 所示。

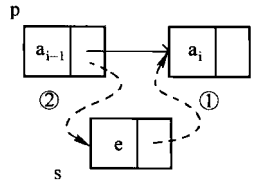


图2-1 布第i个结点之前插入

② 为插入元素生成新结点 s。

③ 插入操作：s->next=p->next;

p->next=s;

上述两条语句不能交换次序。

2）删除第 i 个结点如图 2-2 所示：

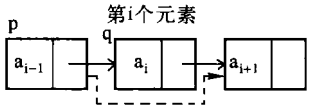


图2-2 删除第i个结点

① 找到第 i 个结点的前驱 p。

② 删除操作：q=p->next;

p->next=q->next; delete q;

3）表头插入（Prepend 方法）如图 2-3 所示：

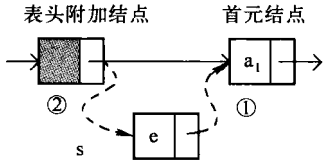


图2-3 表头插入

① 为插入元素生成新结点 s。

② 插入：s->next=head->next; head->next=s;

4）表尾插入（Append 方法）如图 2-4 所示：

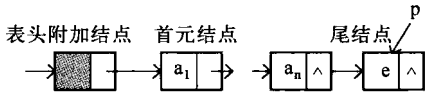


图2-4 表尾插入

① 找到表尾结点 p：

p=head;

while(p->next) p=p->next;

② 插入：p->next=s; s->next=NULL;

5）在递增有序单链表中插入元素 e，使之仍然有序：

① 当指针 p 从前往后查找时，若 p->data<e，则应插在 p 后面的某个位置，若指针 p->data>e，则元素 e 应插在结点 p 之前。若要在 p 之前插入，则必须在查找过程中记录下 p 的前驱结点的指针，所以可用 p->next 的 data 值与 e 进行比较：

p=head;

while(p->next&&p->next->data<e) p=p->next;

② 插入：s->next=p->next;

p->next=s;

注意：创建单链表的过程，实质上是从空表开始，不断向表中插入元素的过程，所以可根据需要调用上述各种插入算法实现单链表的创建。

4．单循环链表

（1）定义

单循环链表与单链表的区别仅仅在于表尾结点的指针域（next）存放头结点的地址（存在头结点时）或第一个结点的地址（不存在头结点时），使链表形成一个环。

（2）存储特点

1）从表中任意结点出发均可找到表中的其他结点。

2）表空的条件：head->next= =head。

3）算法中循环的结束条件为：p= =head 或 p->next= =head（注意与单链表的区别）。

4）根据需要还可在单循环链表中不设头指针，而是设指向表尾结点的指针，这样做可使某些操作简化（如连接两个单循环链表为一个链表等）。

5．双向循环链表

（1）定义

单链表中的每个结点只有一个指针域（next），易于确定结点后继，但无法确定其前驱（必须从表头开始查找），故在双向链表中，每个结点都设立了两个指针：prior（指向前驱）和 next（指向后继）。

（2）主要操作

插入操作：双向循环链表中每个结点有两个指针（前驱指针 prior，后继指针 next），只要找到第 i 个结点，即可访问其前驱和后继结点，并在第 i 个结点之前或之后插入结点，如图 2-5 所示。

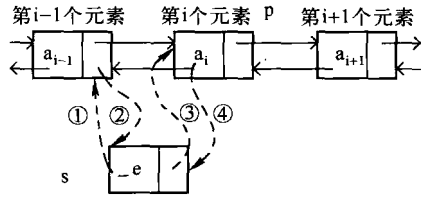


图 2-5　在双向循环链表中第 i 个结点之前插入

① 首先找到第 i 个结点 p。

② 在 p 之前插入，则 s->prior=p->prior;　p->prior->next=s;

s->next=p;　p->prior=s;

注意：第 4 条语句必须位于第 1、2 条语句之后。

③ 在 p 之后插入，则 s->next=p->next; p->next->prior=s;

s->prior=p; p->next=s;

注意：第 4 条语句必须位于第 1、2 条语句之后。

2.2　习题及解答

一、填空题（或从括号中选择）

1．线性表的顺序存储结构，采用_____实现，是_____（随机/ 非随机）存取结构。

2．在长为 n 的顺序表的第 i 个元素（1≤i≤n+1）之前插入一个元素时，需向后移动_____个元素，若每个位置插入的概率相同，则插入操作的平均移动元素次数为_____。

3．在长为 n 的顺序表中删除第 i 个元素（1≤i≤n）时，需向前移动_____个元素，若每个元素被删除的概率相同，则删除操作的平均移动元素次数为_____。

4．线性表的链式存储结构中，表中元素的存储位置是_____（连续/ 任意）的，是一种_____（随机/ 非随机）存取结构。

5．单链表的头指针为 head，带表头附加结点时，表空的条件是 _____，不带表头附加结点时，表空的条件是_____。

6．双向循环链表的头指针为 head，若带表头附加结点，则表空的条件是____或_____，若不带表头附加结点，则表空的条件是_____。

7．当线性表中的元素基本稳定且很少进行插入和删除操作，但要求以最快的速度存取线性表中的元素时，应采用_____存储结构；若线性表最常用的操作是存取任意指定序号的元素和在表尾进行插入和删除操作，则应采用_____ 存储结构；若线性表中的元素经常增删，则应采用_____存储结构。

解答：

1．数组，随机　　2．n-i+1，n/2

3．n-i，(n-1)/2　　4．任意，非随机

5．head->next= =NULL，head= =NULL

6．head->next= =head，head->prior= =head，head= =NULL

7．顺序，顺序，链式

二、算法题

1．写出教材中线性表顺序存储的类实现。

```
/*****************SeqList.h 文件，定义 SeqList 类***********/
#ifndef SEQLIST_H
#define SEQLIST_H
#include <new>
#include <stdexcept>
#define INCREAMENT 10
template <class Type> class SeqList{
private:
   Type *elem;                                    //数组首地址
     int length;                                  //表长
     int listsize;                                //表空间的大小
public:
   SeqList(int =10);                              //建立 size 大小的空表
   SeqList<Type> (const SeqList<Type> &L);        //复制构造函数
   SeqList<Type> & operator = ( const SeqList<Type> & L );//赋值运算符重载
   ~SeqList(void){delete [] elem;};               //析构函数，释放顺序表的存储空间
   int IsEmpty(void)const{return length = = 0;};  //判空
   void Clear(void){ length = 0;};                //删除顺序表的所有元素，变为空表
   int Length(void)const{return length;};         //表长
   Type &GetElem(int i)const{return elem[i-1];};  //返回顺序表第 i 个元素的值
   int Replace(int i,Type &e);                    //用 e 替代表中的第 i 个元素
   int Locate(const Type &e);                     //查找元素 e，找到则返回其位置，否则返回 0
```

```
    int InsertBefore(int i,Type &e);                    //将元素 e 插到第 i 个元素之前
    int InsertAfter(int i,Type &e);                     //将元素 e 插到第 i 个元素之后
    int Delete(int i,Type &e);                          //删除第 i 个元素，且将其值通过变量 e 返回
    Type & Prior(int i);                                //求顺序表第 i 个数据元素的直接前驱
    Type & Next(int i);                                 //求顺序表第 i 个数据元素的直接后继
};
template<class Type>SeqList<Type>::SeqList(int size){   //构造函数
        elem=new Type [size];
        if(!elem) throw bad_alloc("allocation failure in default Constructor(   )");
        length=0;
        listsize=size;
}
template<class Type>                                    //复制构造函数
SeqList<Type>::SeqList(const SeqList<Type> &L):length(L.length ),listsize(L.listsize ){
        elem=new <Type> [L.listsize ];
        if(!elem) throw bad_alloc("allocation failure in copy Constructor(   )");
        for(int i=0;i<L.length ;i++)   elem[i]=L.elem [i];
}
//赋值运算符重载
template<class Type> SeqList<Type> &SeqList<Type>::operator =(const SeqList<Type> &L) {
        if(this!=&L){
                elem=new Type [L.listsize ];
                if(!elem)   throw bad_alloc("allocation failure in   assignment operator");
                for(int i=0;i<L.length ;i++)   elem[i]=L.elem [i];
                length =L.length ;
                listsize=L.listsize;
        }
        return *this;
}
//替换表中的第 i 个元素，替换成功则返回 1，否则返回 0
template<class Type> int SeqList<Type>::Replace(int i,Type &e)   {
        if(i<1||i>length)   return 0;
        elem[i-1]=e;
        return 1;
}
//定位元素 e 在表中的位置（位序），e 不在表中，则返回 0
template<class Type> int SeqList<Type>::Locate(const Type &e)   {
        for(int i=1;i<=length;i++)
                if(elem[i-1]==e) break;
        if(i>length) return 0;
        else return i
}
//在表中的第 i 个位置（位序）之前插入元素 e，插入成功则返回 1，否则返回 0
template<class Type> int SeqList<Type>::InsertBefore(int i,Type &e){
        if(i<1||i>length+1)       return 0;                 //插入位置非法
        if(length>=listsize) {                              //若表空间不够，则需重新分配空间
```

```
            Type *newbase=new Type [listsize+INCREAMENT];
            if(!newbase)       throw bad_alloc("allocation failure in   InsertBefore( )");
            for(int i=0;i<length;i++)       newbase[i]=elem[i];
            delete[] elem;
            elem=newbase;
            listsize+=INCREAMENT;
        }
        for(int j=length;j>=i;j--)       elem[j]=elem[j-1];       //各个元素逐个后移
        elem[i-1]=e;
        length++;
        return 1;
}
template<class Type> int SeqList<Type>::InsertAfter(int i,Type &e) {
        return InsertBefore(i+1,e);              //在第 i 元素之后插入，即在第 i+1 个元素之前插入
}
template<class Type> int SeqList<Type>::Delete(int i,Type &e) {
        if(i<1||i>length)   return 0;
        e=elem[i-1];                             //从第 i 个元素起至最后一个元素，逐个前移
        for(int j=i;j<length;j++)elem[j-1]=elem[j];
        length--;
        return 1;
}
template<class Type> Type & SeqList<Type>::Prior(int i) {
        if(i<2||i>length)   throw out_of_range("subscript out of range in Prior()");
        return elem[i-2];
}
template<class Type> Type & SeqList<Type>::Next(int i) {
        if(i<1||i>length-1) throw out_of_range("subscript out of range in Next()");
        return elem[i];
}
#endif
```

2．写出教材中单链表类的定义。

```
/************ListNode.h 文件，定义 ListNode 类***********/
#ifndef LISTNODE_H
#define LISTNODE_H
template <class Type> class LinkList;                    //声明链表类
template <class Type>    class ListNode {                //结点类定义
        friend class LinkList<Type>;                     //声明链表类 LinkList 为友元类
public:
        ListNode( ): next(NULL) { };                     //构造函数
        ListNode (const Type &e, ListNode <Type>*ptr=NULL): data(e), next(ptr){}   //构造函数
        Type & Data( ) { return data;}                   //返回结点的数据值
        ListNode<Type> * Next( ) { return next;}         //返回结点的指针值
        void SetData( Type & e){ data = e; }             //设置结点的数据值
        void SetNext( ListNode<Type> * ptr) { next = ptr; }   //设置结点的指针值
```