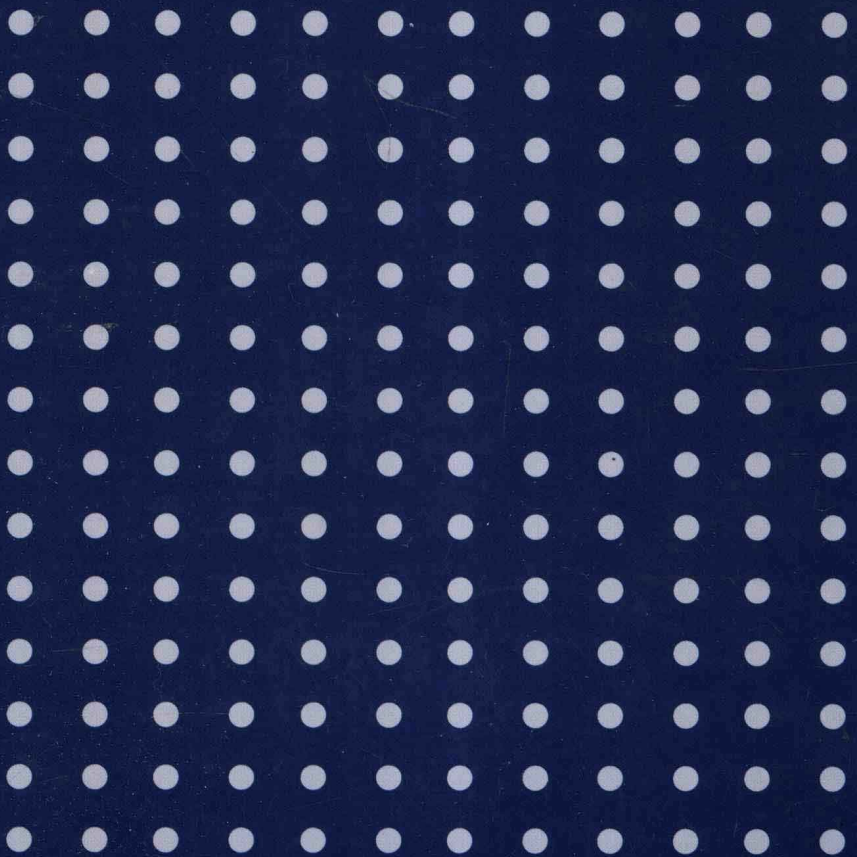
重点大学计算机专业系列教材

面向对象的C++数据结构算法实现与解析

高一凡 著

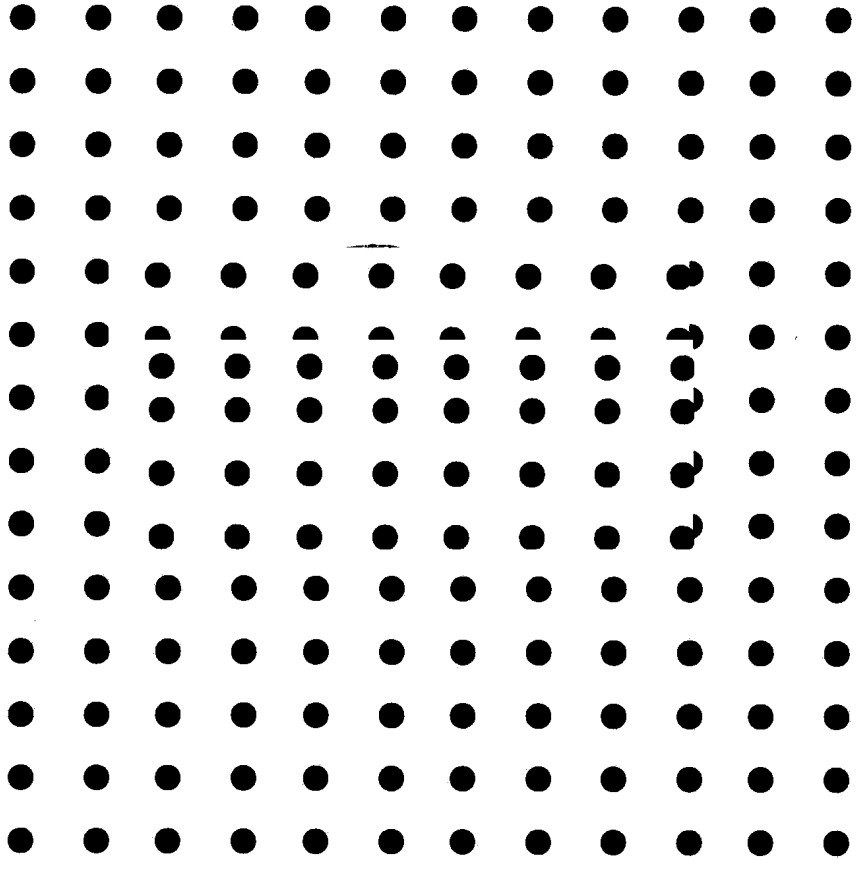


清华大学出版社

重点大学计算机专业系列教材

面向对象的C++数据结构算法实现与解析

高一凡 著



清华大学出版社
北京

内 容 简 介

本书是采用面向对象的 C++语言数据结构教材的学习辅导书，主要内容包括采用 C++语言的类、模板、虚函数、友元、友类编写的各种主要数据存储结构的算法、基本操作成员函数、调用这些成员函数的主程序和程序运行结果以及各主要数据存储结构的图示。本书还介绍了 STL 模板的应用。

本书结合存储结构和算法，配合大量的图示，对于一些较难理解的算法，还配有文字说明。本书所有程序均在计算机上运行通过，这些程序可通过清华大学出版社网站（www.tup.com.cn）下载。

本书适用于高等学校学生和自学者，同时也是很好的考研参考书。

图书在版编目（CIP）数据

面向对象的 C++数据结构算法实现与解析 / 高一凡著. —北京：清华大学出版社，2011.9
（重点大学计算机专业系列教材）
ISBN 978-7-302-24788-3

Ⅰ. ①面… Ⅱ. ①高… Ⅲ. ①C 语言－程序设计－高等学校－教材 ②数据结构－高等学校－教材 Ⅳ. ①TP312 ②TP311.12

中国版本图书馆 CIP 数据核字（2011）第 032107 号

责任编辑：郑寅堃　王冰飞
责任校对：李建庄
责任印制：何　芊
出版发行：清华大学出版社　　**地　　址**：北京清华大学学研大厦 A 座
http://www.tup.com.cn　　**邮　　编**：100084
社　总　机：010-62770175　　**邮　　购**：010-62786544
投稿与读者服务：010-62795954，jsjjc@tup.tsinghua.edu.cn
质　量　反　馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：北京市清华园胶印厂
经　　销：全国新华书店
开　　本：185×260　**印　张**：25.25　**字　数**：630 千字
版　　次：2011 年 9 月第 1 版　　**印　　次**：2011 年 9 月第 1 次印刷
印　　数：1～3000
定　　价：39.00 元

产品编号：035753-01

INTRODUCTION
出版说明

随着国家信息化步伐的加快和高等教育规模的扩大，社会对计算机专业人才的需求不仅体现在数量的增加上，而且体现在质量要求的提高上，培养具有研究和实践能力的高层次的计算机专业人才已成为许多重点大学计算机专业教育的主要目标。目前，我国共有16个国家重点学科、20个博士点一级学科、28个博士点二级学科集中在教育部部属重点大学，这些高校在计算机教学和科研方面具有一定优势，并且大多以国际著名大学计算机教育为参照系，具有系统完善的教学课程体系、教学实验体系、教学质量保证体系和人才培养评估体系等综合体系，形成了培养一流人才的教学和科研环境。

重点大学计算机学科的教学与科研氛围是培养一流计算机人才的基础，其中专业教材的使用和建设则是这种氛围的重要组成部分，一批具有学科方向特色优势的计算机专业教材作为各重点大学的重点建设项目成果得到肯定。为了展示和发扬各重点大学在计算机专业教育上的优势，特别是专业教材建设上的优势，同时配合各重点大学的计算机学科建设和专业课程教学需要，在教育部相关教学指导委员会专家的建议和各重点大学的大力支持下，清华大学出版社规划并出版本系列教材。本系列教材的建设旨在“汇聚学科精英、引领学科建设、培育专业英才”，同时以教材示范各重点大学的优秀教学理念、教学方法、教学手段和教学内容等。

本系列教材在规划过程中体现了如下一些基本组织原则和特点。

1．面向学科发展的前沿，适应当前社会对计算机专业高级人才的培养需求。教材内容以基本理论为基础，反映基本理论和原理的综合应用，重视实践和应用环节。

2．反映教学需要，促进教学发展。教材要能适应多样化的教学需要，正确把握教学内容和课程体系的改革方向。在选择教材内容和编写体系时注意体现素质教育、创新能力与实践能力的培养，为学生知识、能力、素质协调发展创造条件。

3．实施精品战略，突出重点，保证质量。规划教材建设的重点依然是专业基础课和专业主干课；特别注意选择并安排了一部分原来基础比较好

的优秀教材或讲义修订再版，逐步形成精品教材；提倡并鼓励编写体现重点大学计算机专业教学内容和课程体系改革成果的教材。

4．主张一纲多本，合理配套。专业基础课和专业主干课教材要配套，同一门课程可以有多本具有不同内容特点的教材。处理好教材统一性与多样化的关系；基本教材与辅助教材以及教学参考书的关系；文字教材与软件教材的关系，实现教材系列资源配套。

5．依靠专家，择优落实。在制订教材规划时要依靠各课程专家在调查研究本课程教材建设现状的基础上提出规划选题。在落实主编人选时，要引入竞争机制，通过申报、评审确定主编。书稿完成后要认真实行审稿程序，确保出书质量。

繁荣教材出版事业，提高教材质量的关键是教师。建立一支高水平的以老带新的教材编写队伍才能保证教材的编写质量，希望有志于教材建设的教师能够加入到我们的编写队伍中来。

教材编委会

FOREWORD
前言

随着计算机技术的发展，面向对象的 C++的应用越来越广泛。因此，越来越多的“数据结构”教材采用 C++语言描述算法。这是一个很好的趋势，因为数据结构所涉及的内容主要应用在较大型的程序中，而较大型的程序一般都要采用有“类”的面向对象的算法语言实现。但作者在讲授“数据结构”课程时，发现部分学生在学习用 C++描述的算法时还有一些困难。究其原因，有以下几点：

首先，对 C++语言的运用还不够熟练。一是受学习 C++的学时所限，还达不到完全掌握的程度；二是在初学 C++的时候，教师主要（也只能）注重于 C++的语法、结构等方面的知识，所举例子的规模都很小，不容易看出 C++的优点——面向对象的方法用在小程序中反而显得繁琐，只有在大规模的情况下，才彰显 C++高效的优点。因此，学生对于 C++的特点、优势还不能完全了解。

其次，限于篇幅与重点，数据结构教材中的算法一般都是片段式的，不是完整的可执行程序。这对于精通 C++的读者来说没问题，但对于 C++的初学者就有困难了，他们往往无法把算法和实际程序结合到一起。即使理解了算法，还是无法实际应用。数据结构课程并不是一门纯理论的课程，它是十分强调理论与实践的结合的。

笔者根据以上问题，在参考文献［5］的基础上，将程序修改为用面向对象的 C++语言编写了本书，并且尽量将模板、虚函数、友类、继承等面向对象的概念应用到程序中。笔者相信，本书不仅会使读者轻松学好数据结构，还会使读者在掌握 C++方面有长足的进步。

本书在参考文献[5]的基础上还增添了一些新的算法，包括 N 皇后算法、马踏棋盘算法、背包算法、Boyer-Moore 模式匹配算法、红黑树算法、伸展树算法、AVL 树、B 树、键树和哈希表的删除算法。本书还增加了“动态存储管理”一章。作者始终认为，“动态存储管理”是数据结构中很重要的内容，但由于学时所限，往往被删减掉。本书还增加了使用 C++的 STL（标准模板库）的程序。STL 是公共数据结构，在掌握了数据结构的原理后，使用 STL 会收到事半功倍的效果。

本书延续了作者一贯的风格，采用图文并茂的方式解释算法。作者认为结构问题用图来说明是最好的。

本书所有程序都在 Microsoft Visual C++ 6.0 下调试通过。这些程序都可通过清华大学出版社网站（www.tup.tsinghua.edu.cn 或 www.tup.com.cn）下载。

本书提供可执行程序的目的有三个：第一是为了让读者对于 C++用于数据结构有个整体概念；第二是为了便于读者理解算法——读者可以在算法的适当地方增加输出语句，随时查看各变量的变化情况；第三是提供给读者一个编程平台，大量短小的诸如求长度、求是否为空的基本操作可以在读者编写自己的程序时采用，从而提高编程效率。

在作者编写本书时，作者的同事李鹏以他精湛的 C++功底，给了作者巨大的帮助。在此，作者向他表示衷心的感激。

尽管作者尽了最大努力，但限于水平，书中疏漏之处在所难免，希望读者及同仁不吝赐教。读者可通过 gyfan@chd.edu.cn 与作者取得联系。

作者
2011 年 6 月于长安大学

CONTENTS

目录

CHAPTER 1

线 性 表

第1章

文件 C.h 中用宏命令#include 包含了本书中所有程序要用到的 C++的头文件。后面的所有程序，只要用到 C++的头文件，就只需将文件 C.h 包含即可。

```
//C.h 几乎各程序都需要用到的文件包含宏命令和使用名空间
#ifndef _C_H_
#define _C_H_
#include <iostream>//cout、cin、std 等
#include <fstream>//fin 等
#include <iomanip>//setw()等
#include <cmath>//数学函数头文件
#include <string>//字符串
#include <vector>//STL 中的向量
#include <list>//STL 中的链表
#include <stack>//STL 中的栈
#include <queue>//STL 中的队列和优先队列
#include <bitset>//STL 中的位集合
#include <algorithm>//STL 中的算法
#include <ctime>//time()
#include <cstdarg>//提供宏 va_start、va_arg 和 va_end，用于存取变长参数表
#include <assert.h>//assert 宏
using namespace std;//使用名空间
#endif
```

线性表是一种抽象的数据类型，本章将介绍几种具体的线性表存储结构（即物理结构）：顺序、链式和静态链式（这几种存储结构也应用于后续章节中）。无论线性表采用哪种物理结构，它们的逻辑结构都是一样的，都有以下的性质：除第一个元素外，线性表中每个元素都有一个前驱元素；除最后一个元素外，线性表中每个元素都有一个后继元素。文件 AList.h 定义了线性表的抽象类。

```
//AList.h 线性表抽象类的定义
#ifndef _ALIST_H_
#define _ALIST_H_
```

```
template<typename T>class AList//带模板的线性表抽象类
{
public:
  void ClearList();//重置线性表为空表
  bool ListEmpty()const;//若线性表为空表，则返回 true；否则返回 false
  int LocateElem(T e, bool(*eq)(T, T))const;
  //返回第一个与 e 满足关系 eq()的数据元素的位序。若这样的数据元素不存在，则返回值为 0
  bool PriorElem(T e, bool(*eq)(T, T), T &pre_e)const;
  //若 e 与表的某数据元素满足定义的 eq()相等关系，且该数据元素不是表中第一个，
  //则用 pre_e 返回它的前驱；否则操作失败，pre_e 无定义
  bool NextElem(T e, bool(*eq)(T, T), T &next_e)const;
  //若 e 与表的某数据元素满足定义的 eq()相等关系，且该数据元素不是表中最后一个，
  //则用 next_e 返回它的后继；否则操作失败，next_e 无定义
  bool ListDelete(int i, T &e);
  //删除线性表第 i 个数据元素（1≤i≤ListLength()），并用 e 返回其值
  void ListTraverse(void(*visit)(T*))const;
  //依次对每个数据元素调用函数 visit()
  virtual bool GetElem(int i, T &e)const=0;//纯虚函数
  //用 e 返回线性表第 i 个数据元素的值（1≤i≤ListLength()）
  virtual bool ListInsert(int i, T e)=0;
  //在线性表第 i 个位置（1≤i≤ListLength()+1）之前插入新的数据元素 e
  virtual int ListLength()const=0;//返回线性表的长度，常成员函数，不改变对象的值
};
#endif
```

1.1 顺序存储结构

顺序存储结构容易实现随机查找线性表的第 i 个数据元素的操作，但在实现插入或删除操作时要移动大量数据元素。所以，它适用于数据相对稳定的线性表，如职工工资表、学生学籍表等。

```
//SqList.h 顺序表的类（SqList 类）
#ifndef _SQLIST_H_
#define _SQLIST_H_
#include "AList.h"//线性表抽象类的定义
template<typename T>class SqList:public AList<T>
{//带模板并继承 AList 的顺序表类
  friend void MergeList(const SqList<T>&, const SqList<T>&, SqList<T>&);
  //声明普通函数 MergeList()为 SqList 类的友元
private://3 个私有数据成员（见图 1-1）
  T *elem;//线性表存储空间的基址
  int length;//线性表的当前表长
  int listsize;//线性表当前的存储容量
public://12 个成员函数
  SqList(int k=1)
  {//构造函数，动态生成具有 k 个初始存储空间的空线性表（见图 1-2）
    elem=new T[k];
```

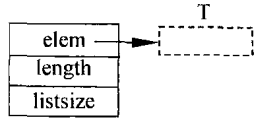


图 1-1 SqList 类的 3 个数据成员

```
  assert(elem!=NULL);//存储分配失败，退出
  length=0;//空表长度为 0
  listsize=k;//初始存储容量
}
~SqList()
{//析构函数（见图 1-3）
  delete[]elem;//释放 elem 所指的存储空间数组
}
void ClearList()
{//重置线性表为空表（见图 1-4）
  length=0;
}
bool ListEmpty()const//常成员函数，不会改变对象的值
{//若线性表为空表，则返回 true；否则返回 false
  return length==0;
}
int ListLength()const
{//返回线性表的长度
  return length;
}
bool GetElem(int i, T &e)const
{//用 e 返回线性表第 i 个数据元素的值
 //（1≤i≤ListLength()）
  if(i<1 || i>length)//i 不在表的范围之内
    return false;
  e=*(elem+i-1);//将第 i 个元素的值赋给 e
  return true;
}
int LocateElem(T e, bool(*eq)(T, T))const
{//返回第一个与 e 满足关系 eq()的数据元素的位序。若这样的数据元素不存在，则返回值为 0
  int i=1;//i 的初值指示第一个元素
  while(i<=length && !eq(*(elem+i-1), e))
  //i 没超出表的范围且 i 所指的数据元素与 e 不满足关系
    i++;//计数加 1，继续向后找
  if(i<=length)//找到满足关系的数据元素
    return i;//返回其位序
  else//没找到满足关系的数据元素
    return 0;
}
bool PriorElem(T e, bool(*eq)(T, T), T &pre_e)const
{//若 e 与表的某数据元素满足定义的 eq()相等关系，且该数据元素不是表中第一个，
 //则用 pre_e 返回它的前驱；否则操作失败，pre_e 无定义
  int i=LocateElem(e, eq);//将第一个与 e 满足 eq()相等关系的数据元素的位序赋给 i
  if(i<=1)//没找到，或是第一个元素
    return false;//操作失败
  else//找到值为 e 的元素，其位序为 i
  {
    pre_e=*(elem+i-2);//将 i 指示的前一个元素的值赋给 pre_e
```

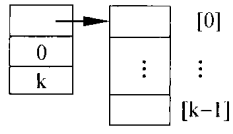


图 1-2　构造函数生成的线性表

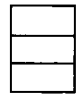
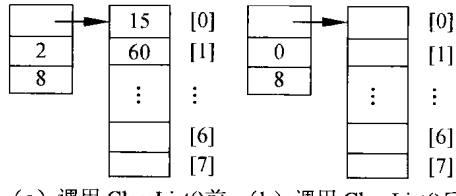


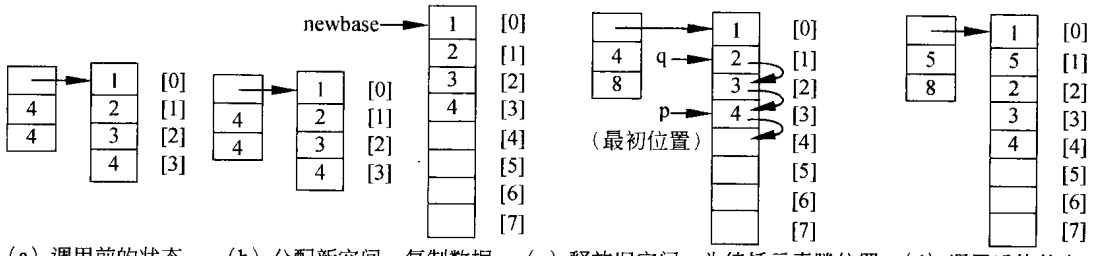
图 1-3　析构函数释放存储空间



（a）调用 ClearList()前　（b）调用 ClearList()后

图 1-4　ClearList()重置顺序表为空表

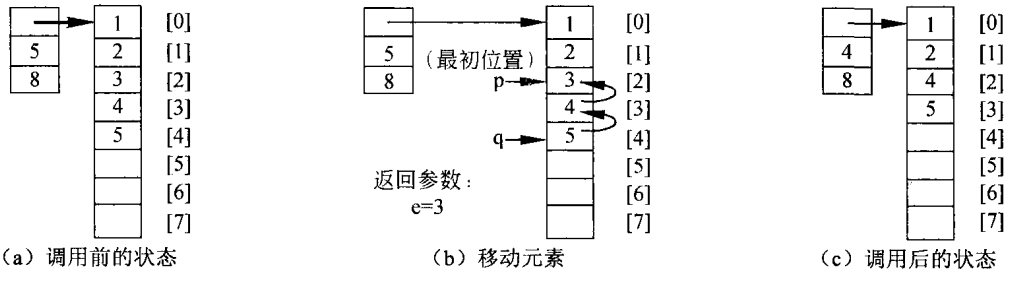
```
    return true;//操作成功
  }
}
bool NextElem(T e, bool(*eq)(T, T), T &next_e)const
{//若e与表的某数据元素满足定义的eq()相等关系，且该数据元素不是表中最后一个，
 //则用next_e返回它的后继；否则操作失败，next_e无定义
  int i=LocateElem(e, eq);//将第一个与e满足eq()相等关系的数据元素的位序赋给i
  if(i==0 || i==length)//没找到，或是最后一个元素
    return false;//操作失败
  else//找到值为e的元素，其位序为i
  {
    next_e=*(elem+i);//将i指示的下一个元素的值赋给next_e
    return true;//操作成功
  }
}
bool ListInsert(int i, T e)
{//在线性表第i个位置（1≤i≤ListLength()+1）之前插入新的数据元素e（见图1-5）
  T *newbase, *q, *p;
  if(i<1 || i>length+1)//i值不合法
    return false;
  if(length==listsize)//当前存储空间已满
  {
    newbase=new T[listsize*2];//新空间为原先的2倍
    assert(newbase!=NULL);//存储分配失败，退出
    for(int j=0; j<length; j++)
      *(newbase+j)=*(elem+j);//将原表空间中的数据复制到新的表空间
    delete[]elem;//释放原表空间
    elem=newbase;//新基址赋给elem
    listsize*=2;//更新存储容量
  }
  q=elem+i-1;//q为插入位置
  for(p=elem+length-1; p>=q;--p)//插入位置及之后的元素后移（由表尾元素开始移）
    *(p+1)=*p;
  *q=e;//插入e
  ++length;//表长增1
  return true;
}
```



（a）调用前的状态　（b）分配新空间，复制数据　（c）释放旧空间，为待插元素腾位置　（d）调用后的状态

图1-5　ListInsert()插入元素示例（i=2，e=5）

```
bool ListDelete(int i, T &e)
{//删除线性表第 i 个数据元素（1≤i≤ListLength()），并用 e 返回其值（见图 1-6）
  T *p, *q;
  if(i<1 || i>length)//i 值不合法
    return false;
  p=elem+i-1;//p 为被删除元素的位置
  e=*p;//被删除元素的值赋给 e
  q=elem+length-1;//q 为表尾元素的位置
  for(++p; p<=q; ++p)//被删除元素之后的元素前移（由被删除元素的后继元素开始移）
    *(p-1)=*p;
  length--;//表长减 1
  return true;
}
```



（a）调用前的状态　　（b）移动元素　　（c）调用后的状态

图 1-6　ListDelete()删除元素示例（i=3）

```
  void ListTraverse(void(*visit)(T*))const;//在类内声明成员函数
};
template<typename T>
void SqList<T>::ListTraverse(void(*visit)(T*))const//在类外定义成员函数
{//依次对每个数据元素调用函数 visit()
  for(int i=0; i<length; i++)//从表的第一个元素到最后一个元素
    visit(elem+i);//对每个数据元素调用 visit()
  cout<<endl;
}
#endif
```

SqList 类中的 ListTraverse()是本书唯一一个在类内声明类外定义的成员函数，其目的是给出格式。由于本书中的大部分函数都较简单，为了节约篇幅选择在类内定义函数。

在图 1-1 中，用一个始于 elem 变量框之中的箭头表示 elem 是指针类型。用该箭头止于 T 类型的虚框表明 elem 是 T 类型的指针。

文件 SqList.h 中定义的 SqList 类顺序表，其存储容量是可变的。首先确定一个初始存储容量（默认为 1）。当表长等于存储容量，又要插入元素时，先将存储容量加倍，再插入元素。这种增加存储容量的方法其时间复杂度是最优的（证明见参考文献[4]），其空间复杂度也是可接受的。

在带模板 T 的 SqList 类中，遍历线性表的函数 ListTraverse()的作用是依次对线性表中所有数据元素做某种操作。为了能够适应模板的各种实际类型的任何操作又不需要修改 SqList 类，要将操作函数 visit()指针设为 ListTraverse()的形参。其相应实参在 SqList 类外

根据具体情况定义。在函数指针形参的声明中指定了 visit()的函数类型，也就是函数返回值的类型（void），还指定了 visit()函数形参的个数（一个）和类型（T*）。在 SqList 类外，凡是满足该条件的函数均可作为 ListTraverse()的实参函数。文件 Func1-1.cpp 中的函数 print()和形参 visit()函数的类型一致，故可作为 ListTraverse()的实参函数。它的作用是输出数据元素。

函数 LocateElem()也有函数指针形参 eq()，称为“比较器”。为了能够适应模板的各种实际类型的任何比较，eq()的实参函数也要在类外实现。文件 Func1-1.cpp 中的函数 equal()和形参 eq()函数的类型一致，可作为 LocateElem()的实参函数。它的作用是判断两数据元素是否相等。

在 Main1-1.cpp 中，若选择第 5 行，把第 6 行作为注释，则 T 为整型，主程序第 4 行定义线性表对象 L 的数据类型就是整型；选择第 6 行，把第 5 行作为注释，则 T 为双精度型，主程序第 4 行定义线性表对象 L 的数据类型就是双精度型。它们分别使用 Func1-1.cpp 中各自类型重载的 equal()函数。

```
//Func1-1.cpp 3个常用的函数
bool equal(int c1, int c2)//PriorElem()等可调用
{//判断是否相等的函数，用于T为int类型
  return c1==c2;
}
bool equal(double c1, double c2)//PriorElem()等可调用
{//判断是否相等的函数，用于T为double类型
  return abs(c1-c2)<1.0e-6;
}
inline void print(T* c)//ListTraverse()可调用，内联函数
{//输出c所指元素，用于T为基本类型
  cout<<*c<<" ";
}

//Main1-1.cpp 验证顺序表SqList<T>类的成员函数
#include "C.h"//文件包含宏命令
#include "SqList.h"//SqList<T>类
//以下两行可根据需要选其一（且只能选其一），而无须改变SqList.h
typedef int T;//定义以下的数据类型T为整型，print()需要。第5行
//typedef double T;//定义以下的数据类型T为双精度型，print()需要。第6行
#include "Func1-1.cpp"//3个常用的函数
void main()
{
  bool i;
  int j, k;
  T e, e0;
  SqList<T> L;//顺序表类的对象，主程序第4行
  for(j=1; j<=5; j++)
    L.ListInsert(1, j);//在L的表头插入j
  cout<<"在L的表头依次插入1~5后，L=";
  L.ListTraverse(print);//依次对表L中的元素调用print()函数（输出元素的值）
```

```
  cout<<"L是否空? "<<boolalpha<<L.ListEmpty()<<", 表长="<<L.ListLength()<<endl;
  L.GetElem(4, e);//用e返回L的第4个数据元素的值
  cout<<"第4个元素的值为"<<e<<endl;
  for(j=L.ListLength(); j<=L.ListLength()+1; j++)
  {
    k=L.LocateElem(j, equal);//查找表L中与j相等的元素，并将其位序赋给k
    if(k)//k不为0，表明有符合条件的元素
      cout<<"值为"<<j<<"的元素是表L的第"<<k<<"个元素, ";
    else//k为0，没有符合条件的元素
      cout<<"没有值为"<<j<<"的元素\n";
  }
  for(j=1; j<=2; j++)//测试头两个数据
  {
    L.GetElem(j, e);//将表L中的第j个元素的值赋给e
    i=L.PriorElem(e, equal, e0);//求e的前驱，如成功，将值赋给e0
    if(i)//操作成功
      cout<<"元素"<<e<<"的前驱为"<<e0<<endl;
    else//操作失败
      cout<<"元素"<<e<<"无前驱, ";
  }
  for(j=L.ListLength()-1; j<=L.ListLength(); j++)//测试最后两个数据
  {
    L.GetElem(j, e);//将表L中的第j个元素的值赋给e
    i=L.NextElem(e, equal, e0);//求e的后继，如成功，将值赋给e0
    if(i)//操作成功
      cout<<"元素"<<e<<"的后继为"<<e0;
    else//操作失败
      cout<<", 元素"<<e<<"无后继"<<endl;
  }
  k=L.ListLength();//k为表长
  for(j=k+1; j>=k; j--)
  {
    i=L.ListDelete(j, e);//删除第j个数据，如成功，将值赋给e
    if(i)//删除成功
      cout<<"删除第"<<j<<"个元素成功，其值为"<<e;
    else//表中不存在第j个数据
      cout<<"删除第"<<j<<"个元素失败。";
  }
  L.ClearList();//清空表L
  cout<<endl<<"清空L后，L是否空? "<<boolalpha<<L.ListEmpty();
  cout<<", 表长="<<L.ListLength()<<endl;
}
```

程序运行结果：

```
在L的表头依次插入1~5后，L=5 4 3 2 1
L是否空? false，表长=5
第4个元素的值为2
值为5的元素是表L的第1个元素，没有值为6的元素
```