

高等学校计算机规划教材

# 数据结构 项目实训教程

◆ 赵君喆 戴文华 主 编  
◆ 卢社阶 闻 彬 副主编



中国工信出版集团



电子工业出版社  
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY  
<http://www.phei.com.cn>

高等学校计算机规划教材

# 数据结构项目实训教程

赵君喆 戴文华 主 编

卢社阶 闻 彬 副主编



电子工业出版社

Publishing House of Electronics Industry

北京 • BEIJING

## 内 容 简 介

本书可作为《数据结构（C语言版）》一书的配套实训教程。全书由11章和1个附录组成，其中第0章给出所有项目的总体实训规范，第1~10章描述各种数据结构的实训项目，各章节大致由结构特点总结、项目实训具体要求、核心代码提示和实训拓展四个部分组成，附录提供了标准化代码风格参考规范。本书配套素材中包含了所有实训项目的完整参考代码。

本书内容丰富、实践性强，不仅可作为高等学校数据结构课程的配套教材，还可作为广大工程技术人员和自学者的辅助学习资料。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。

版权所有，侵权必究。

## 图书在版编目（CIP）数据

数据结构项目实训教程 / 赵君喆，戴文华主编. —北京：电子工业出版社，2017.8

ISBN 978-7-121-31939-6

I. ①数… II. ①赵… ②戴… III. ①数据结构—高等学校—教材 IV. ①TP311.12

中国版本图书馆 CIP 数据核字（2017）第 139820 号

策划编辑：戴晨辰

责任编辑：裴 杰

印 刷：三河市华成印务有限公司

装 订：三河市华成印务有限公司

出版发行：电子工业出版社

北京市海淀区万寿路 173 信箱 邮编 100036

开 本：787×1 092 1/16 印张：19.75 字数：550 千字

版 次：2017 年 8 月第 1 版

印 次：2017 年 8 月第 1 次印刷

定 价：42.50 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 [zltts@phei.com.cn](mailto:zltts@phei.com.cn)，盗版侵权举报请发邮件至 [dbqq@phei.com.cn](mailto:dbqq@phei.com.cn)。

本书咨询联系方式：[dcc@phei.com.cn](mailto:dcc@phei.com.cn)。

# 前 言

计算机程序由数据结构和算法构成。在计算机科学理论方面，数据结构揭示了信息的逻辑结构、存储结构和相应操作；在编程实践方面，数据结构是程序设计的技术基础。因此，数据结构既是一门理论性学科，又具有很强的实践性。数据结构的实践性体现在工程开发的多个阶段。在实际软件工程施工过程中能否正确并灵活地运用数据结构，是衡量数据结构实践教学质量的一个重要标准。

传统的数据结构实践教材往往立足于离散知识点，并针对各种数据结构和算法设计小规模验证实验。这样的实践方式由于缺乏系统性，而显得较为枯燥，难以激发学习者对数据结构实践应用的兴趣，也难以使学习者体会数据结构在软件工程项目开发中所扮演的角色。

本书打破传统数据结构实践教材的格局，以工程项目思想贯穿始终，将所有数据结构及相关知识点封装成一个个完整的工程项目，并明确项目成果要求和标准化项目开发过程要求。本书的学习者将站在项目开发者的角度进行数据结构的应用和实践，同时遵循标准化软件开发流程，以实训的方式体验工程项目的实现过程。本书所构造的所有项目均具有较高的实用性，项目成果可作为实用工具应用于学习者今后的学习和工作中。

全书由 11 章和 1 个附录组成，各部分的主要内容如下：第 0 章给出所有项目的总体实训规范（即项目开发流程的公共要求）；第 1~7 章对各种基本数据结构（顺序表、栈、队列、串、广义表、树、图）的特性进行了总结，并针对各种数据结构设计了相应的项目实训和拓展；第 8 章设计了多种动态存储结构的项目实训和拓展；第 9 章设计了各种查找表和算法的项目实训和拓展；第 10 章设计了各种排序算法的项目实训和拓展；附录提供了标准化代码风格参考规范。本书所有实训项目的完整参考代码和其他相关配套资源，读者可登录华信教育资源网（[www.hxedu.com.cn](http://www.hxedu.com.cn)）注册后免费下载或联系编者（E-mail: [daiwenh@163.com](mailto:daiwenh@163.com)）索取。

本书的编写人员有赵君喆、戴文华、卢社阶、闻彬等，由于编者水平有限，加之时间仓促，书中难免存在不当之处，恳请广大读者批评指正。

编 者

# 目 录

|                          |    |
|--------------------------|----|
| 第0章 项目总体实训规范 .....       | 1  |
| 第1章 抽象数据类型项目实训 .....     | 4  |
| 第2章 线性表项目实训 .....        | 7  |
| 2.1 顺序表 .....            | 8  |
| 2.2 单链表 .....            | 13 |
| 2.3 循环链表 .....           | 18 |
| 2.4 双向循环链表 .....         | 20 |
| 2.5 静态链表 .....           | 22 |
| 2.6 线性表应用项目（多项式运算） ..... | 26 |
| 2.7 线性表项目实训拓展 .....      | 31 |
| 第3章 栈和队列项目实训 .....       | 32 |
| 3.1 栈 .....              | 32 |
| 3.1.1 顺序栈 .....          | 32 |
| 3.1.2 链栈 .....           | 37 |
| 3.2 队列 .....             | 41 |
| 3.2.1 顺序队列 .....         | 42 |
| 3.2.2 链队列 .....          | 46 |
| 3.2.3 循环队列 .....         | 50 |
| 3.3 栈和队列应用项目 .....       | 53 |
| 3.4 栈和队列项目实训拓展 .....     | 71 |
| 第4章 串项目实训 .....          | 73 |
| 4.1 串的定长存储 .....         | 74 |

|              |                         |            |
|--------------|-------------------------|------------|
| 4.2          | 串的堆分配存储 .....           | 80         |
| 4.3          | 串的块链存储 .....            | 83         |
| 4.4          | 串项目实训拓展 .....           | 94         |
| <b>第 5 章</b> | <b>数组和广义表项目实训 .....</b> | <b>96</b>  |
| 5.1          | 数组的顺序存储 .....           | 96         |
| 5.2          | 三元组稀疏矩阵 .....           | 99         |
| 5.3          | 行逻辑链接稀疏矩阵 .....         | 104        |
| 5.4          | 广义表头尾链式存储 .....         | 110        |
| 5.5          | 数组与广义表项目实训拓展 .....      | 116        |
| <b>第 6 章</b> | <b>树和二叉树项目实训 .....</b>  | <b>117</b> |
| 6.1          | 树 .....                 | 117        |
| 6.1.1        | 树的双亲表示法 .....           | 118        |
| 6.1.2        | 树的孩子兄弟表示法 .....         | 127        |
| 6.2          | 二叉树项目实训 .....           | 138        |
| 6.2.1        | 二叉树的顺序存储 .....          | 139        |
| 6.2.2        | 二叉树的链式存储 .....          | 147        |
| 6.2.3        | 线索二叉树 .....             | 156        |
| 6.3          | 树和二叉树应用项目 .....         | 161        |
| 6.4          | 树和二叉树项目实训拓展 .....       | 167        |
| <b>第 7 章</b> | <b>图结构项目实训 .....</b>    | <b>168</b> |
| 7.1          | 图的邻接矩阵表示 .....          | 169        |
| 7.2          | 图的邻接表表示 .....           | 183        |
| 7.3          | 图的十字链表表示 .....          | 193        |
| 7.4          | 图的邻接多重表表示 .....         | 202        |
| 7.5          | 图的高级算法项目 .....          | 214        |
| 7.6          | 图项目实训拓展 .....           | 228        |
| <b>第 8 章</b> | <b>动态存储管理项目实训 .....</b> | <b>230</b> |
| 8.1          | 边界标识法 .....             | 231        |

|               |                        |            |
|---------------|------------------------|------------|
| 8.2           | 伙伴系统 .....             | 235        |
| 8.3           | 动态内存管理项目实训拓展 .....     | 239        |
| <b>第 9 章</b>  | <b>查找表项目实训 .....</b>   | <b>240</b> |
| 9.1           | 静态查找表 .....            | 241        |
| 9.1.1         | 顺序查找表 .....            | 241        |
| 9.1.2         | 有序查找表 .....            | 243        |
| 9.1.3         | 静态查找树表 .....           | 246        |
| 9.2           | 动态查找表 .....            | 250        |
| 9.2.1         | 二叉排序树 .....            | 250        |
| 9.2.2         | 平衡二叉树 .....            | 255        |
| 9.2.3         | B-树 .....              | 260        |
| 9.2.4         | 双链键树 .....             | 265        |
| 9.2.5         | Trie 树 .....           | 270        |
| 9.3           | 哈希表 .....              | 274        |
| 9.4           | 查找项目实训拓展 .....         | 279        |
| <b>第 10 章</b> | <b>排序项目实训 .....</b>    | <b>280</b> |
| 10.1          | 常见排序算法 .....           | 280        |
| 10.2          | 链式基数排序 .....           | 286        |
| 10.3          | 排序项目实训拓展 .....         | 289        |
| <b>附录</b>     | <b>标准化代码规范参考 .....</b> | <b>291</b> |

# 第0章 项目总体实训规范

数据结构是一门实践性非常强的课程，不但要深入理解各种结构的原理，还要能在解决实际问题的过程中灵活运用。因此，本书从项目实践入手，在商业项目的标准化运作过程中提炼核心开发流程，结合软件工程学的知识针对各种数据结构设计专门的实训项目。

本书所有实训项目的总体性实训规范如下。

## 一、实训要求

- 1) 程序必须严格按照标准代码规范进行编写；
- 2) 依据软件标准开发流程完成各种文档的编写；
- 3) 按照教师要求，独立完成项目。

## 二、实训环境

操作系统 Windows XP 以上，C/C++编译环境 VC 6.0 以上。

## 三、实训安排

### 1. 准备阶段

实训前准备以下空白文档：

- 1) 项目计划及进度控制表；
- 2) 需求规格说明表；
- 3) 功能模块结构说明表；
- 4) 测试计划表；
- 5) 缺陷记录表。

### 2. 计划、设计阶段

- 1) 估计项目规模及开发时间，针对项目细节做出详细时间安排，并跟踪完成情况；
- 2) 对项目进行需求分析，完成需求规格说明表；
- 3) 设计各程序模块，完成功能模块结构说明表。

### 3. 编码调试阶段

- 1) 遵循标准代码规范书写规整的代码；
- 2) 根据设计文档编码实现程序；
- 3) 编译调试程序，使程序能够顺利运行。

### 4. 项目测试阶段

- 1) 完成项目测试计划表，根据项目功能填写测试用例，构造测试数据；

- 2) 依据测试计划表对程序进行测试;
- 3) 修复程序缺陷, 填写缺陷记录表。

5. 文档提交阶段

整理并汇总所有文档以及可顺利编译运行的源程序, 提交任课教师。

四、文档规范

1. 项目计划及进度控制表

针对项目开发流程中的每一个阶段, 制订准确的时间约束, 在项目进行中跟踪每一个开发环节, 每天更新各环节的完成进度以及任务延迟情况, 如表 0.1 所示。

表 0.1 项目计划及进度控制表

| 开发阶段 | 起始时间 | 终止时间 | 延迟时间 | 完成量/ (%) | 备注 |
|------|------|------|------|----------|----|
| 计划制订 |      |      |      |          |    |
| 需求分析 |      |      |      |          |    |
| 设计   |      |      |      |          |    |
| 编码   |      |      |      |          |    |
| 测试   |      |      |      |          |    |
| 整理打包 |      |      |      |          |    |
| 总计   |      |      |      |          |    |

2. 需求规格说明表

对项目的功能设计, 应站在数据结构使用者的角度去进行分析, 在需求分析的过程中要着重考虑结构操作的实用性、健壮性和可扩展性。对于一些常用的结构, 甚至可以将其设计成一个具有实用意义的库以供今后的开发使用, 如表 0.2 所示。

表 0.2 需求规格说明表 (以顺序表结构为例)

| 功能需求    | 详细说明                               |
|---------|------------------------------------|
| 用户操作菜单  | 用户可以参照菜单输入操作编号来选择对结构的特定操作          |
| 创建顺序表   | 用户可选择以文件的方式输入结构初始数据, 也可采用实时的方式输入数据 |
| 按位置查询元素 | 如果输入的位置超过表长, 则提示用户输入错误, 不会对表越界访问   |
| 删除元素    | 如果表中无任何元素, 则直接提示空表                 |
| ...     | ...                                |

3. 功能模块结构说明表

程序的功能模块的说明是项目的重要资料, 这些资料是程序的使用者以及程序的维护者的主要参考依据。因此, 针对程序中每一个重要的函数模块都应该给出详细的接口信息以及内部运作逻辑, 如表 0.3 所示。

表 0.3 功能模块结构说明表（以顺序表结构为例）

| 函数名         | 参数说明                                                    | 返回值说明                | 操作行为说明                                                     |
|-------------|---------------------------------------------------------|----------------------|------------------------------------------------------------|
| InitList    | 参数：传入顺序表结构引用                                            | 无返回值                 | 为顺序表申请默认长度的内存空间，并将表置空                                      |
| LoopCommand | 参数：传入顺序表结构引用                                            | 返回逻辑值，表示是否结束循环命令输入状态 | 使用户循环输入操作代号，依据代号调用不同操作函数，其中一个代号代表退出程序                      |
| GetElem     | 参数 1：传入顺序表结构变量<br>参数 2：传入查询元素位置<br>参数 3：外部变量引用，传出查到的元素值 | 返回状态码                | 判断参数 2 是否在表长度之内，如果不在，则返回错误值（负值）；将参数 3 赋值为查询到的元素值，并返回正确值（0） |
| ...         | ...                                                     | ...                  | ...                                                        |

4. 测试计划表

项目编码和调试结束之后，应该执行严格的测试流程来尽可能发现程序的缺陷。而测试计划设计的好坏决定着测试的效果，因此应尽可能从多种角度设计测试项目和测试步骤。而一份完善的数据结构测试计划表，可作为将来结构修改或维护时的质量标准，如表 0.4 所示。

表 0.4 测试计划表（以顺序表结构为例）

| 测试项目      | 测试步骤                              | 期望结果             | 测试目的                     |
|-----------|-----------------------------------|------------------|--------------------------|
| 顺序表插入元素测试 | 1. 插入元素，输入合理插入位置<br>2. 列出表信息，观察结果 | 元素正确插入到特定位置      | 正面测试，测试插入功能的正常使用情况       |
| 顺序表插入元素测试 | 1. 插入元素，输入不合理的插入位置<br>2. 观察程序输出状态 | 程序提示输入位置不合理而不会崩溃 | 负面测试，测试插入位置错误情况下的程序健壮性   |
| 顺序表插入元素测试 | 1. 销毁顺序表<br>2. 在合理的位置插入元素，观察结果    | 程序提示表已销毁，不能插入元素  | 负面测试，测试在表销毁情况下插入元素的程序健壮性 |
| ...       | ...                               | ...              | ...                      |

5. 缺陷记录表

对项目的测试应该严格依据之前制订的测试计划表来进行，在测试的过程中每发现一个程序缺陷都应该详细记录。在修复缺陷之后，还要进行回归测试，并跟踪每一个缺陷的修复状态，整理成缺陷记录表，如表 0.5 所示。

表 0.5 缺陷记录表（以顺序表结构为例）

| 序号  | 程序缺陷说明                     | 修复情况 | 备注               |
|-----|----------------------------|------|------------------|
| 1   | 空表删除元素导致程序崩溃               | 已修复  | 严重漏洞，必须修复        |
| 2   | 表中元素数量达到表容量后，插入元素不会自动扩充表容量 | 待修复  | 可采用 2 倍递增的方式扩充空间 |
| ... | ...                        | ...  | ...              |

# 第 1 章 抽象数据类型项目实训

对于一些基本数据类型无法描述的信息结构，可以将其逻辑特性抽象出来，形成一组数据以及对这些数据的一组逻辑操作，这称为**抽象数据类型**。

逻辑特征抽象层次越高，抽象数据类型的复用性程度就越高，因此，有必要掌握如何定义一个抽象数据类型，以及如何用代码来实现该数据类型。

## 一、本章实训目的

- 用 C 或 C++ 语言实现一个抽象数据类型；
- 实现一个用户操作界面来验证该数据类型；
- 运行程序并对其进行测试。

## 二、实训项目要求

编写程序实现以下抽象数据类型，并实现一个主函数调用三元组的所有函数操作。

ADT Triplet {

数据对象： $D = \{e_1, e_2, e_3 \mid e_1, e_2, e_3 \in \text{Elemset}\}$

数据关系： $R_1 = \{ \langle e_1, e_2 \rangle, \langle e_2, e_3 \rangle \}$

基本操作：

InitTriplet(&T, v1, v2, v3)

操作结果：构造三元组 T，元素  $e_1$ 、 $e_2$ 、 $e_3$  分别赋值为  $v_1$ 、 $v_2$ 、 $v_3$ 。

DestroyTriplet(&T)

操作结果：三元组 T 被销毁。

Get(T, i, &e)

初始条件：三元组 T 已存在， $1 \leq i \leq 3$ 。

操作结果：用 e 返回 T 的第 i 元值。

Put(&T, i, e)

初始条件：三元组 T 已存在， $1 \leq i \leq 3$ 。

操作结果：改变 T 的第 i 元值为 e。

IsAscending(T)

初始条件：三元组 T 已存在。

操作结果：若 T 的 3 个元素按升序排列，则返回 1，否则返回 0。

IsDescending(T)

初始条件：三元组 T 已存在。

操作结果：若 T 的 3 个元素按降序排列，则返回 1，否则返回 0。

Max(T, &e)

初始条件：三元组 T 已存在。

操作结果：用 e 返回 T 的 3 个元素最大值。

Min(T, &e)

初始条件：三元组 T 已存在。

操作结果：用 e 返回 T 的 3 个元素最小值。

} ADT Triplet

### 三、重要代码提示

使用连续的空间来存储三元组中的元素，如果是动态三元组的内存，则只定义三元组类型 Triplet 为其元素类型的指针即可。

```
// Triplet类型是ElemType类型的指针，用于存放ElemType类型的地址
typedef ElemType *Triplet; // 由InitTriplet分配3个元素存储空间
```

三元组初始化函数 InitTriplet 传入已定义的 Triplet 变量的引用 T（因此，函数要将新申请的空间指针赋予三元组变量，所以参数必须传入三元组变量的引用），同时传入三元组的 3 个初值 v1、v2、v3。在函数中先为 T 申请 3 个 ElemType 类型的空间，如果申请失败，则直接退出程序，否则将 v1、v2、v3 分别赋值给 T[0]、T[1]、T[2]，参考代码如下。

```
Status InitTriplet(Triplet &T, ElemType v1, ElemType v2, ElemType v3)
{
    // 操作结果：构造三元组T，依次置T的3个元素的初值为v1、v2和v3
    if (!(T = (ElemType *) malloc(3 * sizeof(ElemType))))
    {
        exit(OVERFLOW);
    }
    T[0] = v1, T[1] = v2, T[2] = v3;
    return OK;
}
```

因为三元组 T 是连续分配的空间，所以销毁三元组的函数 DestroyTriplet 只需要直接释放 T 所指向的空间，随机将 T 置为空即可。

```
Status DestroyTriplet(Triplet &T)
{
    // 操作结果：三元组T被销毁
    free(T);
    T = NULL;
    return OK;
}
```

获取三元组元素值的函数 Get 通过参数 i 传入元素顺序，通过引用 e 传出元素值。首先，判断 i 的合法性，因为是三元组，所以 i 值只能为 1~3。因为下标编号是从 0 开始的，所以 T[i-1] 是三元组第 i 个元素的值，最后将 T[i-1] 的值赋予 e。

```
Status Get(Triplet T, int i, ElemType &e)
{
    // 初始条件：三元组T已存在，1≤i≤3
    // 操作结果：用e返回T的第i个元素的值
    if (i < 1 || i > 3)
    {
        return ERROR;
    }
    e = T[i - 1];
}
```

```

    return OK;
}

```

修改三元组元素值的函数 Put 和函数 Get 类似，只是将要修改的值通过参数 e 传递进来，并将其赋值给 T[i-1]，在此之前同样要对 i 的合法性进行判断，见如下代码。

```

Status Put(Triplet T, int i, ElemType e)
{
    // 初始条件：三元组T已存在， $1 \leq i \leq 3$ 
    // 操作结果：改变T的第i元素的值为e
    if (i < 1 || i > 3)
    {
        return ERROR;
    }
    T[i - 1] = e;
    return OK;
}

```

判断三元组是否升序排列的函数 IsAscending 实现起来很简单，只需要比较相邻的两个元素是不是后者比前者大，如果后者比前者大则返回 1，否则返回 0。

```

Status IsAscending(Triplet T)
{
    // 初始条件：三元组T已存在
    // 操作结果：如果T的3个元素按升序排列，则返回1，否则返回0
    return (T[0] <= T[1] && T[1] <= T[2]);
}

```

判断降序排列的函数 IsDescending 和 IsAscending 类似，代码略过。从三元组中找出最大元素的函数 Max 先比较 T[0]和 T[1]，将大的赋值给引用参数 e，再比较 e 和 T[2]，将大的赋值给 e 即可。找寻最小数的函数 Min 和函数 Max 类似，具体代码这里不再赘述。

```

Status Max(Triplet T, ElemType &e)
{
    // 初始条件：三元组T已存在
    // 操作结果：用e返回指向T的最大元素的值
    e = T[0] >= T[1] ? T[0] : T[1];
    e = e >= T[2] ? e : T[2];
    return OK;
}

```

## 第2章 线性表项目实训

线性表是具有相同属性的数据元素的一个有限序列，是最简单最常用的一种数据结构，其基本特点是结构中的元素之间满足线性关系。

### 一、线性表的基本操作

- 创建表；
- 求表长度；
- 查找元素；
- 插入元素；
- 删除元素；
- 遍历元素。

### 二、本章实训目的

- 1) 用 C 或 C++ 语言实现本章所学的各种线性表结构；
- 2) 编写线性表的基本操作函数（求长度、查找、插入、删除、遍历等）；
- 3) 实现一个对线性表进行各种操作的用户界面（见图 2.1）；
- 4) 运行程序并对其进行测试。

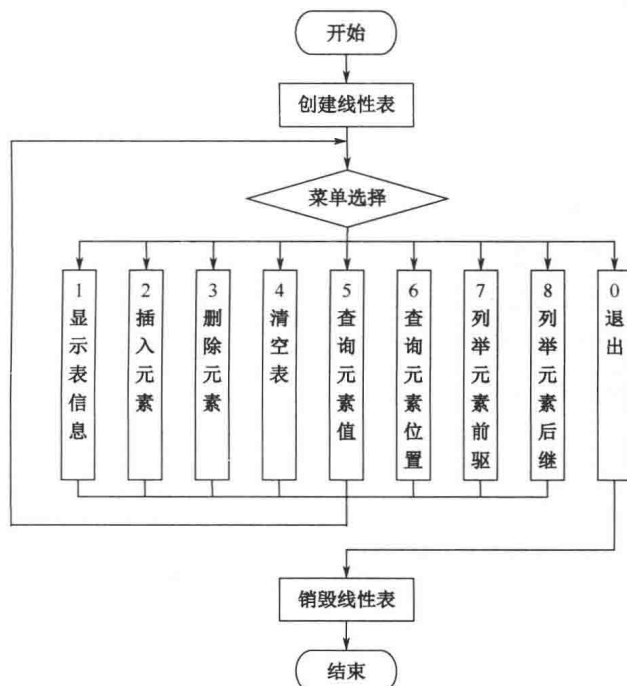


图 2.1 线性表操作程序流程图

三、线性表的实现形式

- 顺序存储形式（顺序表）；
- 链式存储形式（单链表、双向链表、循环链表）；
- 静态存储形式（静态链表）。

2.1 顺序表

一、顺序表结构特点

- 1) 采用一组地址连续的存储单元来存储数据；
- 2) 表的存储容量长度不易改变；
- 3) 方便随机查找；
- 4) 适用于频繁访问元素的应用；
- 5) 不适用于频繁增删元素的应用。



图 2.2 顺序表存储结构示意图

二、实训项目要求

开发一个顺序表的操作程序，要求程序至少具备以下顺序表的操作接口。

- InitList（顺序表初始化函数）；
- DestroyList（顺序表销毁函数）；
- ClearList（顺序表清空函数）；
- ListEmpty（判断是否为空表的函数）；
- ListLength（计算表长的函数）；
- GetElem（依据位置查询元素值的函数）；
- LocateElem（判断元素位置的函数）；
- PriorElem（查询前驱函数）；
- NextElem（查询后继函数）；
- ListInsert（元素插入函数）；
- ListDelete（元素删除函数）；
- ListTraverse（元素遍历函数）。

要求程序具有任用户选择操作的菜单，并支持以下菜单项。

- 列举表元素；
- 插入元素；
- 删除元素；
- 清空表；
- 查询元素值；
- 查询元素位置；
- 列举前驱元素；

- 列举后继元素；
- 退出程序。

### 三、重要代码提示

在构造结构之前应该确定顺序表的元素数据类型、表初始存储容量以及每次扩大顺序表的容量时所分配的增量，参考以下代码。

```
typedef int ElemType;    //用户指定数据类型
#define LIST_INIT_SIZE 10 // 线性表存储空间的初始分配量
#define LIST_INCREMENT 2  // 线性表存储空间的分配增量
```

一个完整的顺序表结构必须拥有的属性包括存储空间的起始地址、顺序表的数据长度以及存储容量；因此，顺序表结构定义中包含了 3 个成员变量以对应 3 个必要的顺序表属性。参考如下代码，定义一个顺序表结构 SqList，其中存储空间起始指针变量 elem 必须定义为顺序表元素的数据类型 ElemType 的指针，而表容量 listsize 则以 sizeof (ElemType) 为单位。

```
struct SqList
{
    ElemType *elem;    // 存储空间基址
    int      length;    // 当前长度
    int      listsize;  // 当前分配的存储容量
}
```

下面给出了构造空线性表的参考函数 InitList，此处采用 SqList 变量的引用作为函数参数传递到函数中进行处理，因此必须要有明确定义的 SqList 变量才能调用该函数。需要注意的是，顺序表的空间是通过动态内存分配申请的，因此在销毁顺序表的时候必须调用 free 函数释放内存。

```
void InitList(SqList &L) // 创建顺序表
{
    // 初始条件：顺序表L已定义
    // 操作结果：构造一个空的顺序表L
    L.elem = (ElemType*) malloc(LIST_INIT_SIZE * sizeof(ElemType));
    if (!L.elem)
    {
        exit(OVERFLOW);    // 若存储分配失败，则退出程序
    }
    L.length = 0;    // 空表长度为0
    L.listsize = LIST_INIT_SIZE; // 初始存储容量
}
```

对于顺序表而言，依据元素的位置查找元素值实现起来非常方便，下面的代码实现了随机查找顺序表元素的函数 GetElem。利用 ElemType 型变量 e 的引用传递出查询到的元素值，因此调用函数前要先定义变量 e，而对于传入元素编号的变量 i 一定要在函数中判断其值是否在顺序表长度以内，以免造成对顺序表的越界访问。

```
Status GetElem(SqList L, int i, ElemType &e)
{
    // 初始条件：顺序线性表L已存在,  $1 \leq i \leq \text{ListLength}(L)$ 
    // 操作结果：用e返回L中第i个数据元素的值
    if (i < 1 || i > L.length)
    {
        return ERROR;
    }
}
```

```

        e = *(L.elem + i - 1);
        return OK;
    }

```

根据元素值满足的条件来查询元素编号的函数 `LocateElem` 可以设计得比较灵活, 因为元素值和条件比较结果是一个二元逻辑值 (真和假), 所以可以约定一个回调函数形式来比较元素值和参考值, 并约定回调函数的 `Status` 类型返回值来判定元素值和参考值之间的比较结果。所以用户可以自由地设计回调函数, 并将函数指针通过 `compare` 参数传递到 `LocateElem` 函数中。

下面给出了参考代码, 依次将顺序表的元素和参数 `e` 调用 `compare` 所指向的函数进行比较处理, 如果结果为 1 则代表查询成功, 否则查询失败, 查询到的元素编号通过函数返回。

```

int LocateElem(SqList L, ElemType e, Status(*compare)(ElemType,
ElemType))
{
    // 初始条件: 顺序线性表L已存在, compare()是数据元素判定函数(满足为1, 否
    //           则为0)
    // 操作结果: 返回L中第1个与e满足关系compare()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
    ElemType *p;
    int i = 1; // i的初值为第1个元素的位序
    p = L.elem; // p的初值为第1个元素的存储位置
    while (i <= L.length && !compare(*p++, e))
    {
        ++i;
    }
    if (i <= L.length)
    {
        return i;
    }
    else
    {
        return 0;
    }
}

```

顺序表的元素插入函数 `ListInsert` 的关键操作在于插入元素之前要先判定插入位置 `i` 的合法性, 以及顺序表是否还有足够的空间以供元素插入。参考如下代码, 若线性表的存储空间已满, 则调用 `realloc` 函数增加 `LIST_INCREMENT` 个元素的存储空间, 同时将顺序表的容量值增加 `LIST_INCREMENT`。由于顺序表的元素在内存中是连续存储的, 所以插入元素之前, 要从第 `i` 个元素开始, 将后续所有元素后移一个单位。注意, 元素插入成功之后, 一定要将顺序表的长度值增加 1。

```

Status ListInsert(SqList &L, int i, ElemType e)
{
    // 初始条件: 顺序线性表L已存在, 1≤i≤ListLength(L)+1。
    // 操作结果: 在L中第i个位置之前插入新的数据元素e, L的长度加1
    ElemType *newbase, *q, *p;
    if (i < 1 || i > L.length + 1) // i值不合法
    {
        return ERROR;
    }
}

```