实验 2-1　线性表的动态顺序存储实现

一、实验目标

通过本实验，你将能够：

1. 用 C 语言熟练编写程序和调试程序；
2. 掌握和运用 C 语言的高级语法特性，包括：结构体、指针、函数指针、typedef、动态内存分配，等。
3. 掌握从分析问题到编写伪代码、编写程序、调试程序和测试程序的思路；
4. 能够根据给定的 ADT 规格说明熟练的编写代码；
5. 理解并熟练掌握线性表动态顺序存储结构的 C 语言描述；
6. 掌握线性表的动态顺序存储结构的算法实现。

二、ADT 规格说明

线性表的 ADT 规格说明请参考教材 19 页。

三、实验要求

用动态顺序存储的方式实现线性表的 ADT 规格说明。

四、实验步骤

1. 建立本次实验的文件夹 Lab2-1。
2. 从 Moodle 网络课堂下载实验文件 Lab2-1.rar，包含五个代码文件：ElemType.h、ElemType.cpp、DynaSeqList.h、DynaSeqList.cpp 和 Lab.cpp。各代码文件的作用如下表所示：

文件	作用
ElemType.h	定义 ElemType 数据类型
ElemType.cpp	实现 ElemType 数据类型的 Compare 和 visit 两个操作
DynaSeqList.h	动态顺序表 ADT 的定义

DynaSeqList.cpp	**动态顺序表 ADT 的实现（待完成）**
Lab.cpp	**测试动态顺序表的主程序（待完成）**

3. ElemType.h 文件定义了 ElemType 类型，ElemType.cpp 文件针对 ElemType 类型实现了 Compare 和 visit 函数，这样线性表中就可以存放 ElemType 这种类型的数据了。这两个文件的代码如下。

ElemType.h

```
#ifndef ELEMTYPE_H
#define ELEMTYPE_H

typedef int ElemType;

int  compare(ElemType x, ElemType y);
void visit(ElemType e);

#endif /* ELEMTYPE_H */
```

ElemType.cpp

```
#include <stdio.h>
#include "ElemType.h"

int compare(ElemType x, ElemType y)
{
   return(x-y);
}

void visit(ElemType e)
{
   printf("%d\n", e);
}
```

4. DynaSeqList.h 文件是动态顺序表的数据结构储存定义和基本操作的声明。DynaSeqList.cpp 是动态顺序表基本操作的实现。**本次实验的主要内容就是编写动态顺序表所有操作函数的实现代码，也就是完成 DynaSeqList.cpp 代码文件的编写。**注意每个操作函数前面都有函数头注释，包括：操作目的、初始条件、操作结果、函数参数和返回值。这些信息对于保证函数代码的逻辑正确性以及对函数进行测试是非常有用的。这两个文件的代码如下。

DynaSeqList.h

```
#if !defined(DYNASEQLIST_H)
#define DYNASEQLIST_H

#include "ElemType.h"
```

```
/*-----------------------------------------------------------
// 顺序表结构的定义
-----------------------------------------------------------*/
typedef struct List
{
   ElemType *elem;          // 存储空间的基址
   int length;              // 顺序表中结点元素的个数
   int listsize;            // 顺序表的存储空间大小
} SqList;

/*-----------------------------------------------------------
// 顺序表的基本操作
-----------------------------------------------------------*/

bool InitList(SqList *L);
void DestroyList(SqList *L);
bool ListEmpty(SqList L);
int  ListLength(SqList L);
bool GetElem(SqList L, int i, ElemType *e);
int  LocateElem(SqList L, ElemType e, int (*fp)(ElemType,
ElemType));
bool PriorElem(SqList L, ElemType cur_e, ElemType *pre_e);
bool NextElem(SqList L, ElemType cur_e, ElemType *nxt_e);
void ListTraverse(SqList L, void (*fp)(ElemType));
void ClearList(SqList *L);
bool ListInsert(SqList *L, int i, ElemType e);
bool ListDelete(SqList *L, int i, ElemType *e);

#endif /* DYNASEQLIST_H */
```

DynaSeqList.cpp

```
/***
*DynaSeqList.cpp - 动态顺序表，即顺序表的动态数组实现
*
*
*题目：实验 2-1 线性表的动态顺序存储实现
*
*班级：
*
*姓名：
*
*学号：
*
****/
```

```c
#include <stdlib.h>
#include <malloc.h>
#include <memory.h>
#include <assert.h>
#include "DynaSeqList.h"

const int LIST_INIT_SIZE = 100; // 表初始分配的最大长度
const int LISTINCREMENT  = 10;  // 分配内存的增量

/*----------------------------------------------------------
操作目的：初始化顺序表
初始条件：无
操作结果：构造一个空的线性表
函数参数：
      SqList *L   待初始化的线性表
返回值：
      bool        操作是否成功
----------------------------------------------------------*/
bool InitList(SqList *L)
{
}

/*----------------------------------------------------------
操作目的：销毁顺序表
初始条件：线性表 L 已存在
操作结果：销毁线性表 L
函数参数：
      SqList *L   待销毁的线性表
返回值：
      无
----------------------------------------------------------*/
void DestroyList(SqList *L)
{
}

/*----------------------------------------------------------
操作目的：判断顺序表是否为空
初始条件：线性表 L 已存在
操作结果：若 L 为空表，则返回 true，否则返回 false
函数参数：
      SqList L 待判断的线性表
返回值：
      bool     是否为空
----------------------------------------------------------*/
bool ListEmpty(SqList L)
```

```
{
}

/*----------------------------------------------------------
操作目的：得到顺序表的长度
初始条件：线性表 L 已存在
操作结果：返回 L 中数据元素的个数
函数参数：
      SqList L 线性表 L
返回值：
      int      数据元素的个数
----------------------------------------------------------*/
int ListLength(SqList L)
{
}

/*----------------------------------------------------------
操作目的：得到顺序表的第 i 个元素
初始条件：线性表 L 已存在，1<=i<=ListLength(L)
操作结果：用 e 返回 L 中第 i 个数据元素的值
函数参数：
      SqList L    线性表 L
      int i       数据元素的位置
      ElemType *e 第 i 个数据元素的值
返回值：
      bool        操作是否成功
----------------------------------------------------------*/
bool GetElem(SqList L, int i, ElemType *e)
{
}

/*----------------------------------------------------------
操作目的：得到顺序表指定元素的位置
初始条件：线性表 L 已存在
操作结果：返回 L 中第一个与 e 满足关系 compare()的数据元素的位序。
          若这样的元素不存在则返回 0。
函数参数：
      SqList L    线性表 L
      ElemType e  数据元素 e
      int (*fp)() 用于比较相等的函数指针
返回值：
      int         与 e 满足关系 compare()的数据元素的位序
----------------------------------------------------------*/
int LocateElem(SqList L, ElemType e, int (*fp)(ElemType,
ElemType))
```

```
{
}

/*----------------------------------------------------------
操作目的：得到顺序表指定元素的前驱
初始条件：线性表 L 已存在
操作结果：若 cur_e 是 L 的数据元素，且不是第一个，则用 pre_e 返回
          它的前驱，否则操作失败，pre_e 无定义
函数参数：
      SqList L          线性表 L
      ElemType cur_e    数据元素 cur_e
      ElemType *pre_e   前驱数据元素
返回值：
      bool              操作是否成功
----------------------------------------------------------*/
bool PriorElem(SqList L, ElemType cur_e, ElemType *pre_e)
{
}

/*----------------------------------------------------------
操作目的：得到顺序表指定元素的后继
初始条件：线性表 L 已存在
操作结果：若 cur_e 是 L 的数据元素，且不是最后一个，则用 nxt_e 返
          回它的后继，否则操作失败，nxt_e 无定义
函数参数：
      SqList L          线性表 L
      ElemType cur_e    数据元素 cur_e
      ElemType *nxt_e   后继数据元素
返回值：
      bool              操作是否成功
----------------------------------------------------------*/
bool NextElem(SqList L, ElemType cur_e, ElemType *nxt_e)
{
}

/*----------------------------------------------------------
操作目的：遍历顺序表
初始条件：线性表 L 已存在
操作结果：依次对 L 的每个元素调用函数 fp
函数参数：
      SqList L        线性表 L
      void (*fp)()    访问每个数据元素的函数指针
返回值：
      无
----------------------------------------------------------*/
```

```
void ListTraverse(SqList L, void (*fp)(ElemType))
{
}

/*------------------------------------------------------------
操作目的：清空顺序表
初始条件：线性表 L 已存在
操作结果：将 L 置为空表
函数参数：
      SqList *L   线性表 L
返回值：
      无
------------------------------------------------------------*/
void ClearList(SqList *L)
{
}

/*------------------------------------------------------------
操作目的：在顺序表的指定位置插入结点，插入位置 i 表示在第 i 个
          元素之前插入
初始条件：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
函数参数：
      SqList *L   线性表 L
      int i       插入位置
      ElemType e  待插入的数据元素
返回值：
      bool        操作是否成功
------------------------------------------------------------*/
bool ListInsert(SqList *L, int i, ElemType e)
{
}

/*------------------------------------------------------------
操作目的：删除顺序表的第 i 个结点
初始条件：线性表 L 已存在且非空，1<=i<=ListLength(L)
操作结果：删除 L 的第 i 个数据元素，并用 e 返回其值，L 的长度减 1
函数参数：
      SqList *L   线性表 L
      int i       删除位置
      ElemType *e 被删除的数据元素值
返回值：
      bool        操作是否成功
------------------------------------------------------------*/
bool ListDelete(SqList *L, int i, ElemType *e)
```

```
{
}
```

5. Lab.cpp 文件包含程序入口函数 main 用来声明和定义线性表对象，并使用线性表 ADT 提供的操作。在这里主要写测试代码，用来保证线性表 ADT 实现的正确性。初始代码如下。

Lab.cpp

```
#include <stdio.h>
#include <stdlib.h>

#include "DynaSeqList.h"

int main()
{
   // TODO: place your test code here

   return 0;
}
```

6. 使用 VC++ 6.0 建立一个空的控制台工程，把上面的五个代码文件加入到工程中，进行代码的编写和调试。

五、思考问题

1. SqList 数据结构的定义中可以没有 length 字段吗？为什么？
2. SqList 数据结构的定义中 listsize 字段的作用是什么？
3. SqList 中的元素类型为 ElemType，有什么好处？
4. LocateElem 函数的第三个入口参数是一个函数指针，为什么需要这个参数？如何没有这个参数可以实现元素的查找和定位吗？
5. SqList 数据结构的 12 个操作中，哪些操作需要传递 SqList 对象的指针，哪些操作不需要传递 SqList 对象的指针。这里面有什么规律？为什么某些操作需要传递 SqList 对象的指针？
6. 自己定义一个 student 类型（包括：name，age，gender，三个字段），再定义一个 book 类型（包括：name, author, publisher, datetime，四个字段），请问能用本实验完成的 SqList ADT，在不改动 DynaSeqList.h 和 DynaSeqList.cpp 代码的前提下，能对这两种不同类型的数据进行增、删、改、查操作吗？请写这样的测试程序？如果可以实现，请问这其中的原理何在？