实验 2-2　线性表的动态链式存储实现

一、实验目标

通过本实验，你将能够在以下几个方面得到锻炼：

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、动态内存分配和递归函数，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链式存储结构的 C 语言描述；
6. 掌握线性表的动态链式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

用动态链式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 Lab2-2。
2. 从 Moodle 网络课堂下载实验文件 Lab2-2.rar，包含五个代码文件：ElemType.h、ElemType.cpp、DynaLnkList.h、DynaLnkList.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
DynaLnkList.h	动态链表 ADT 的定义

DynaLnkList.cpp	**动态链表 ADT 的实现（待完成）**
Lab.cpp	**测试动态链表的主程序（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. DynaLnkList.h 文件是动态链表的数据结构储存定义和基本操作的声明。DynaLnkList.cpp 是动态链表基本操作的实现。**本次实验的主要内容就是编写动态链表所有操作函数的实现代码，也就是完成 DynaLnkList.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。

DynaLnkList.h

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

#include "ElemType.h"
```

```
/*-----------------------------------------------------------
// 链表结构的定义
-----------------------------------------------------------*/
typedef struct Node
{
   ElemType data;          // 元素数据
   struct Node *next;      // 链表中结点元素的指针
} LNode, *LinkList;

/*-----------------------------------------------------------
// 链表的基本操作
-----------------------------------------------------------*/

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

#endif /* DYNALNKLIST_H */
```

DynaLnkList.cpp

```
/***
*DynaLnkList.cpp - 动态链表，即顺序表的动态链式存储实现
*
*
*题目：实验2-2 线性表的动态链式存储实现
*
*班级：
*
*姓名：
*
*学号：
*
****/
```

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

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

/*----------------------------------------------------------
操作目的：销毁链表
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：
      LinkList *L 待销毁的线性表
返回值：
      无
----------------------------------------------------------*/
void DestroyList(LinkList *L)
{
}

/*----------------------------------------------------------
操作目的：判断链表是否为空
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：
      LinkList L  待判断的线性表
返回值：
      bool        是否为空
----------------------------------------------------------*/
bool ListEmpty(LinkList L)
{
}
```

```
/*----------------------------------------------------------
操作目的：得到链表的长度
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：
      LinkList L  线性表L
返回值：
      int         数据元素的个数
----------------------------------------------------------*/
int ListLength(LinkList L)
{
}

/*----------------------------------------------------------
操作目的：得到链表的第i个元素
初始条件：线性表L已存在，<=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：
      LinkList 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(LinkList L, int i, ElemType *e)
{
}

/*----------------------------------------------------------
操作目的：得到链表指定元素的位置
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。
函数参数：
      LinkList 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(LinkList L, ElemType e, int (*fp)(ElemType,
ElemType))
{
}
```

```
/*------------------------------------------------------------
操作目的：得到链表指定元素的前驱
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：
      LinkList 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(LinkList L, ElemType cur_e, ElemType *pre_e)
{
}

/*------------------------------------------------------------
操作目的：得到链表指定元素的后继
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：
      LinkList 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(LinkList L, ElemType cur_e, ElemType *nxt_e)
{
}

/*------------------------------------------------------------
操作目的：遍历链表
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：
      LinkList 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(LinkList L, void (*fp)(ElemType))
{
}
```

```
/*-------------------------------------------------------------
操作目的：清空链表
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：
      LinkList L  线性表L
返回值：
      无
-------------------------------------------------------------*/
void ClearList(LinkList L)
{
}

/*-------------------------------------------------------------
操作目的：在链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，<=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
函数参数：
      LinkList 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(LinkList L, int i, ElemType e)
{
}

/*-------------------------------------------------------------
操作目的：删除链表的第i个结点
初始条件：线性表L已存在且非空，<=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
函数参数：
      LinkList 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(LinkList 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 "DynaLnkList.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. typedef 定义链表数据类型，为什么需要声明两个新类型：LNode 和 LinkList？只定义一个 LinkList 类型，而不定义 LNode 类型可以吗？为什么？
2. 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？
3. LinkList 中的元素类型为 ElemType，有什么好处？
4. 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 LinkList ADT，在不改动 DynaLnkList.h 和 DynaLnkList.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？
5. LinkList L，这个函数形参的 L 是普通变量吗？
6. LinkList *L，这个函数形参的 L 是什么？
7. 本实验中 PriorElem 函数和 NextElem 函数的声明和定义是有问题的，请说明问题出在什么地方？为什么实验 2-1 中的 PriorElem 函数和 NextElem 函数不存在这个问题？
8. PriorElem 函数和 NextElem 函数都可以通过调用 LocateElem 函数来实现，请评价这种实现方法的优劣。