实验 3-2　队列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. 理解并熟练掌握队列动态链式存储结构的 C 语言描述；
6. 掌握队列的动态链式存储结构的算法实现。

二、ADT 规格说明

队列的 ADT 规格说明请参考教材 59 页。

三、实验要求

用动态链式存储的方式实现队列的 ADT 规格说明。

四、实验步骤

1. 建立本次实验的文件夹 Lab3-2。
2. 从 Moodle 网络课堂下载实验文件 Lab3-2.rar，包括五个代码文件：ElemType.h、ElemType.cpp、DynaLnkQueue.h、DynaLnkQueue.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
DynaLnkQueue.h	动态队列 ADT 的定义

DynaLnkQueue.cpp	**动态队列 ADT 的实现（待完成）**
Lab.cpp	**测试动态队列的主程序（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，这样队列中就可以存放 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. DynaLnkQueue.h 文件是动态队列的数据结构储存定义和基本操作的声明。DynaLnkQueue.cpp 是动态队列基本操作的实现。**本次实验的主要内容就是编写动态队列所有操作函数的实现代码，也就是完成 DynaLnkQueue.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。

DynaLnkQueue.h

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

#include "ElemType.h"
```

```
/*-----------------------------------------------------------
// 链式队列结构的定义
-----------------------------------------------------------*/

typedef struct Node
{
   ElemType data;          // 元素数据
   struct Node *next;      // 链式队列中结点元素的指针
} QNode, *QueuePtr;

typedef struct
{
   QueuePtr front;         // 队列头指针
   QueuePtr rear;          // 队列尾指针
} LinkQueue;

/*-----------------------------------------------------------
// 链式队列的基本操作
-----------------------------------------------------------*/

bool InitQueue(LinkQueue *Q);
void DestroyQueue(LinkQueue *Q);
bool QueueEmpty(LinkQueue Q);
int  QueueLength(LinkQueue Q);
bool GetHead(LinkQueue Q, ElemType *e);
void QueueTraverse(LinkQueue Q, void (*fp)(ElemType));
void ClearQueue(LinkQueue *Q);
bool EnQueue(LinkQueue *Q, ElemType e);
bool DeQueue(LinkQueue *Q, ElemType *e);

#endif /* DYNALNKQUEUE_H */
```

DynaLnkQueue.cpp

```
/***
*DynaLnkQueue.cpp - 动态链式队列，即队列的动态链式存储实现
*
*
*题目：实验3-2 队列的动态链式存储实现
*
*班级：
*
*姓名：
*
*学号：
```

```
*
****/

#include <stdlib.h>
#include <malloc.h>
#include <memory.h>
#include <assert.h>
#include "DynaLnkQueue.h"

/*----------------------------------------------------------
操作目的：初始化队列
初始条件：无
操作结果：构造一个空的队列
函数参数：
      LinkQueue *Q   待初始化的队列
返回值：
      bool           操作是否成功
----------------------------------------------------------*/
bool InitQueue(LinkQueue *Q)
{
}

/*----------------------------------------------------------
操作目的：销毁队列
初始条件：队列Q已存在
操作结果：销毁队列Q
函数参数：
      LinkQueue *Q   待销毁的队列
返回值：
      无
----------------------------------------------------------*/
void DestroyQueue(LinkQueue *Q)
{
}

/*----------------------------------------------------------
操作目的：判断队列是否为空
初始条件：队列Q已存在
操作结果：若Q为空队列，则返回true，否则返回false
函数参数：
      LinkQueue Q    待判断的队列
返回值：
      bool           是否为空
----------------------------------------------------------*/
bool QueueEmpty(LinkQueue Q)
```

```
{
}

/*-------------------------------------------------------------
操作目的：得到队列的长度
初始条件：队列Q已存在
操作结果：返回Q中数据元素的个数
函数参数：
      LinkQueue Q    队列Q
返回值：
      int            数据元素的个数
-------------------------------------------------------------*/
int QueueLength(LinkQueue Q)
{
}

/*-------------------------------------------------------------
操作目的：得到队列首元素
 
函数参数：
      LinkQueue Q    队列Q
      ElemType *e    队列首元素的值
返回值：
      bool           操作是否成功
-------------------------------------------------------------*/
bool GetHead(LinkQueue Q, ElemType *e)
{
}

/*-------------------------------------------------------------
操作目的：遍历队列
初始条件：队列Q已存在
操作结果：依次对Q的每个元素调用函数fp
函数参数：
      LinkQueue Q    队列Q
      void (*fp)()   访问每个数据元素的函数指针
返回值：
      无
-------------------------------------------------------------*/
void QueueTraverse(LinkQueue Q, void (*fp)(ElemType))
{
}

/*-------------------------------------------------------------
```

```
操作目的：清空队列
初始条件：队列Q已存在
操作结果：将队列清空
函数参数：
      LinkQueue *Q   队列Q
返回值：
      无
-------------------------------------------------------------*/
void ClearQueue(LinkQueue *Q)
{
}

/*-------------------------------------------------------------
操作目的：在队列末尾插入元素e
初始条件：队列Q已存在
操作结果：插入元素e作为队列新的尾结点
函数参数：
      LinkQueue *Q   队列Q
      ElemType e     待插入的数据元素
返回值：
      bool           操作是否成功
-------------------------------------------------------------*/
bool EnQueue(LinkQueue *Q, ElemType e)
{
}

/*-------------------------------------------------------------
操作目的：删除链式队列的头结点
初始条件：队列Q已存在
操作结果：删除链式队列的头结点
函数参数：
      LinkQueue *Q   队列Q
      ElemType *e    待插入的数据元素
返回值：
      bool           操作是否成功
-------------------------------------------------------------*/
bool DeQueue(LinkQueue *Q, ElemType *e)
{
}
```

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

Lab.cpp

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

#include "DynaLnkQueue.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. 定义 LinkQueue 数据结构时可以不定义 QNode 类型吗？为什么？
2. LinkQueue 数据结构的定义中可以去掉 rear 指针吗？为什么？
3. 如何判断 LinkQueue 对象为空队列？
4. LinkQueue 中的元素类型为 ElemType，有什么好处？
5. 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 LinkQueue 对这两种不同类型的数据进行相应操作吗？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？
6. LinkQueue 数据结构的 9 个操作中，哪些操作需要传递 LinkQueue 对象的指针，哪些操作不需要传递 LinkQueue 对象的指针。这里面有什么规律？为什么某些操作需要传递 LinkQueue 对象的指针？