实验 6-1　二叉树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. 理解并熟练掌握二叉树动态链式存储结构的 C 语言描述；
6. 掌握二叉树的动态链式存储结构的算法实现；

二、ADT规格说明

二叉树的 ADT 规格说明请参考教材 121 页。

三、实验要求

用动态链式存储的方式实现二叉树的 ADT 规格说明。

四、实验步骤

1. 建立本次实验的文件夹 Lab6-1。
2. 从Moodle网络课堂下载实验文件Lab6-1.rar，包括五个代码文件：ElemType.h、ElemType.cpp、DynaLnkBiTree.h、DynaLnkBiTree.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
DynaLnkBiTree.h	动态二叉链树 ADT 的定义

DynaLnkBiTree.cpp	**动态二叉链树 ADT 的实现**
Lab.cpp	**测试动态二叉树的主程序**

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，这样二叉树中就可以存放 ElemType 这种类型的数据了。本实验为了方便演示，ElemType 表示的是字符（char）类型数据。这两个文件的代码如下。

ElemType.h

```
#ifndef ELEMTYPE_H
#define ELEMTYPE_H

typedef char 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("%c\n", e);
}
```

4. DynaLnkBiTree.h 文件是动态二叉链树的数据结构储存定义和基本操作的声明。DynaLnkBiTree.cpp 是动态二叉链树基本操作的实现。**本次实验的主要内容就是编写动态二叉链树所有操作函数的实现代码，也就是完成 DynaLnkBiTree.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。

DynaLnkBiTree.h

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

```
#include "ElemType.h"

/*-------------------------------------------------------------
// 二叉树结构的定义
-------------------------------------------------------------*/

typedef struct BiNode
{
   ElemType data;          // 结点数据元素
   struct BiNode *l;       // 结点的左孩子指针
   struct BiNode *r;       // 结点的右孩子指针
} BinTNode, *BinTree;

enum LR {LEFT, RIGHT};


/*-------------------------------------------------------------
// 二叉树的基本操作
-------------------------------------------------------------*/

bool InitBinTree(BinTree *T);
void DestroyBinTree(BinTree *T);
bool CreateBinTree(BinTree *T);
void ClearBinTree(BinTree *T);
bool BinTreeEmpty(BinTree T);
int  BinTreeDepth(BinTree T);
BinTNode *Root(BinTree T);
bool NodeExist(BinTree T, BinTNode* n);
ElemType Value(BinTree T, BinTNode* n);
void Assign(BinTree T, BinTNode* n, ElemType e);
BinTNode* Parent(BinTree T, BinTNode* n);
BinTNode* LeftChild(BinTree T, BinTNode* n);
BinTNode* RightChild(BinTree T, BinTNode* n);
BinTNode* LeftSibling(BinTree T, BinTNode* n);
BinTNode* RightSibling(BinTree T, BinTNode* n);
void InsertNode(BinTree T, BinTNode* p, LR d, BinTNode* n);
void DeleteNode(BinTree T, BinTNode* p, LR d);
void PreOrderTraverse(BinTree T, void (*fp)(ElemType));
void InOrderTraverse(BinTree T, void (*fp)(ElemType));
void PostOrderTraverse(BinTree T, void (*fp)(ElemType));
void LevelOrderTraverse(BinTree T, void (*fp)(ElemType));

#endif /* DYNALNKBITREE_H */
```

DynaLnkBiTree.cpp

```
/***
*DynaLnkBiTree.cpp - 动态链式二叉树，即二叉树的动态链式存储实现
*
*
*题目：实验6-1 二叉树的动态链式存储实现
*
*班级：
*
*姓名：
*
*学号：
*
****/

#include <stdlib.h>
#include <stdio.h>
#include <malloc.h>
#include <memory.h>
#include <assert.h>
#include "DynaLnkBiTree.h"

/*----------------------------------------------------------
操作目的：初始化二叉树
初始条件：无
操作结果：构造空二叉树
函数参数：
      BinTree *T  待初始化的二叉树
返回值：
      bool        操作是否成功
----------------------------------------------------------*/
bool InitBinTree(BinTree *T)
{
}

/*----------------------------------------------------------
操作目的：销毁二叉树
初始条件：二叉树T已存在
操作结果：销毁二叉树
函数参数：
      BinTree *T  待销毁的二叉树
返回值：
      无
----------------------------------------------------------*/
void DestroyBinTree(BinTree *T)
{
```

```
}
/*----------------------------------------------------------
操作目的：创建二叉树
初始条件：二叉树T已存在
操作结果：销毁二叉树
函数参数：
      BinTree *T  二叉树T
返回值：
      bool        操作是否成功
参考提示：
         请按照教材页算法.4的方式来创建二叉树。
----------------------------------------------------------*/
bool CreateBinTree(BinTree *T)
{
}

/*----------------------------------------------------------
操作目的：清空二叉树
初始条件：二叉树T已存在
操作结果：清空二叉树
函数参数：
      BinTree *T  待清空的二叉树
返回值：
      无
----------------------------------------------------------*/
void ClearBinTree(BinTree *T)
{
}

/*----------------------------------------------------------
操作目的：二叉树判空
初始条件：二叉树T已存在
操作结果：若T为空，则返回true；否则返回false
函数参数：
      BinTree T   二叉树T
返回值：
      bool        二叉树是否为空
----------------------------------------------------------*/
bool BinTreeEmpty(BinTree T)
{
}

/*----------------------------------------------------------
操作目的：二叉树求深度
初始条件：二叉树T已存在
```

```
操作结果：返回二叉树T的深度
函数参数：
      BinTree T   二叉树T
返回值：
      int         二叉树的深度
-----------------------------------------------------------*/
int BinTreeDepth(BinTree T)
{
}

/*-----------------------------------------------------------
操作目的：得到二叉树的根结点
初始条件：二叉树T已存在
操作结果：返回二叉树T的根结点
函数参数：
      BinTree T   二叉树T
返回值：
      BinTNode*   二叉树的根结点指针
-----------------------------------------------------------*/
BinTNode *Root(BinTree T)
{
}

/*-----------------------------------------------------------
操作目的：判断结点n是否为树T的合法结点
初始条件：二叉树T已存在
操作结果：n为T的结点返回true，否则返回false
函数参数：
      BinTree T   二叉树T
      BinTNode* n 二叉树的结点n
返回值：
      bool        结点n是否存在与二叉树T中
-----------------------------------------------------------*/
bool NodeExist(BinTree T, BinTNode* n)
{
}

/*-----------------------------------------------------------
操作目的：得到二叉树指定结点的值
初始条件：二叉树T已存在，n是二叉树T中的结点
操作结果：返回结点n的值
函数参数：
      BinTree T   二叉树T
      BinTNode* n 二叉树结点n
返回值：
```

```
      ElemType    结点n的值
-------------------------------------------------------------*/
ElemType Value(BinTree T, BinTNode* n)
{
}

/*-------------------------------------------------------------
操作目的：对二叉树的指定结点赋值
初始条件：二叉树T已存在，n是二叉树T中的结点
操作结果：操作成功返回true，操作失败返回false
函数参数：
      BinTree T   二叉树T
      BinTNode* n 二叉树结点n
      ElemType e  结点值
返回值：
      无
-------------------------------------------------------------*/
void Assign(BinTree T, BinTNode* n, ElemType e)
{
}

/*-------------------------------------------------------------
操作目的：得到二叉树指定结点的父结点
初始条件：二叉树T已存在，n是二叉树T中的结点
操作结果：如果二叉树结点n有父结点则返回父结点指针，否则返回NULL
函数参数：
      BinTree T   二叉树T
      BinTNode* n 二叉树结点n
返回值：
      BinTNode*   父结点指针
-------------------------------------------------------------*/
BinTNode* Parent(BinTree T, BinTNode* n)
{
}

/*-------------------------------------------------------------
操作目的：得到二叉树指定结点的左孩子
初始条件：二叉树T已存在，n是二叉树T中的结点
操作结果：如果二叉树结点n有左孩子则返回左孩子结点指针，否则返回NULL
函数参数：
      BinTree T   二叉树T
      BinTNode* n 二叉树结点n
返回值：
      BinTNode*   左孩子结点指针
-------------------------------------------------------------*/
```

```
BinTNode* LeftChild(BinTree T, BinTNode* n)
{
}

/*-------------------------------------------------------------
操作目的：得到二叉树指定结点的右孩子
初始条件：二叉树T已存在，n是二叉树T中的结点
操作结果：如果二叉树结点n有右孩子则返回右孩子结点指针，否则返回NULL
函数参数：
      BinTree T   二叉树T
      BinTNode* n 二叉树结点n
返回值：
      BinTNode*   右孩子结点指针
-------------------------------------------------------------*/
BinTNode* RightChild(BinTree T, BinTNode* n)
{
}

/*-------------------------------------------------------------
操作目的：得到二叉树指定结点的左兄弟
初始条件：二叉树T已存在，n是二叉树T中的结点
操作结果：如果二叉树结点n有左兄弟则返回左兄弟结点指针，否则返回NULL
函数参数：
      BinTree T   二叉树T
      BinTNode* n 二叉树结点n
返回值：
      BinTNode*   左兄弟结点指针
-------------------------------------------------------------*/
BinTNode* LeftSibling(BinTree T, BinTNode* n)
{
}

/*-------------------------------------------------------------
操作目的：得到二叉树指定结点的右兄弟
初始条件：二叉树T已存在，n是二叉树T中的结点
操作结果：如果二叉树结点n有右兄弟则返回右兄弟结点指针，否则返回NULL
函数参数：
      BinTree T   二叉树T
      BinTNode* n 二叉树结点n
返回值：
      BinTNode*   右兄弟结点指针
-------------------------------------------------------------*/
BinTNode* RightSibling(BinTree T, BinTNode* n)
{
}
```

```
/*------------------------------------------------------------
操作目的：在二叉树中插入结点
初始条件：二叉树T已存在，p是二叉树T中的结点，n为待插入的结点
操作结果：在二叉树的p结点之前插入结点n
函数参数：
      BinTree T   二叉树T
      BinTNode* p 二叉树结点p
      LR d        二叉树结点p成为新结点n的左孩子或者右孩子
      BinTNode* p 待插入结点n
返回值：
      无
------------------------------------------------------------*/
void InsertNode(BinTree T, BinTNode* p, LR d, BinTNode* n)
{
}

/*------------------------------------------------------------
操作目的：在二叉树中删除结点
初始条件：二叉树T已存在，p是二叉树T中的结点
操作结果：删除二叉树的p结点
函数参数：
      BinTree T   二叉树T
      BinTNode* p 二叉树结点p
      LR d        结点p的左孩子或者右孩子来取代p
返回值：
      无
------------------------------------------------------------*/
void DeleteNode(BinTree T, BinTNode* p, LR d)
{
}

/*------------------------------------------------------------
操作目的：先序遍历二叉树
初始条件：二叉树T已存在
操作结果：先序的方式，对二叉树的每个结点调用(*fp)一次且仅一次
函数参数：
      BinTree T   二叉树T
      *fp         访问函数指针
返回值：
      无
------------------------------------------------------------*/
void PreOrderTraverse(BinTree T, void (*fp)(ElemType))
{
}
```

```
/*-----------------------------------------------------------
操作目的：中序遍历二叉树
初始条件：二叉树T已存在
操作结果：中序的方式，对二叉树的每个结点调用(*fp)一次且仅一次
函数参数：
      BinTree T   二叉树T
      *fp         访问函数指针
返回值：
      无
-----------------------------------------------------------*/
void InOrderTraverse(BinTree T, void (*fp)(ElemType))
{
}

/*-----------------------------------------------------------
操作目的：后序遍历二叉树
初始条件：二叉树T已存在
操作结果：后序的方式，对二叉树的每个结点调用(*fp)一次且仅一次
函数参数：
      BinTree T   二叉树T
      *fp         访问函数指针
返回值：
      无
-----------------------------------------------------------*/
void PostOrderTraverse(BinTree T, void (*fp)(ElemType))
{
}

/*-----------------------------------------------------------
操作目的：层序遍历二叉树
初始条件：二叉树T已存在
操作结果：层序的方式，对二叉树的每个结点调用(*fp)一次且仅一次
函数参数：
      BinTree T   二叉树T
      *fp         访问函数指针
返回值：
      无
-----------------------------------------------------------*/
void LevelOrderTraverse(BinTree T, void (*fp)(ElemType))
{
}
```

5. Lab.cpp 文件包含程序入口函数 main 用来声明和定义二叉树对象，并使用二叉树 ADT 提供的操作。在这里主要写测试代码，用来保证二叉树 ADT 实现的正确性。初始代

码如下。

Lab.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include "DynaLnkBiTree.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. BinTree 数据结构的定义中可以不定义 BinTNode 数据类型吗？为什么？
2. 二叉树 BinTree 为空的条件是什么？
3. BinTree 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 BinTree ADT，在不改动 DynaLnkBiTree.h 和 DynaLnkBiTree.cpp 代码的前提下，能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？