江苏大学

计算机科学与通信工程学院

课程设计报告

课　　程	数据结构 B
课 题 名 称	王的传承
学 生 姓 名	沈鑫楠
学　　号	3150604028
专 业 班 级	信息安全 1501
指 导 教 师	王 新 宇

成绩评定表

	评价内容	考核指标	权重	评分
平时成绩	学习与工作态度	平时出勤率、学习与工作认真程度	10%	
	独立工作能力	工作进展情况，平时抽查 1~2 次/人	20%	
答辩成绩	任务完成情况	课题功能完善程度与合理性	10%	
	陈述与回答问题	陈述思路、表达能力与回答正确性	20%	
课程设计质量	论证与分析	方案论证与综合分析的正确、合理性	10%	
	设计与实现	设计与编码的正确、合理性；实验运行的合理性与数据的可靠性	10%	
	报告书质量	条理清晰、文理通顺、用语符合技术规范、书写格式规范化	10%	
	创新	工作中有创新意识或有独特见解。	10%	
总分				
指导教师签名				

特别说明

1. 无故缺席或请假未获批准擅自离开达到 1 次或课程设计上机时间内做与课程设计无关的事情达到 2 次，学习与工作态度评分项为 0 分；
2. 答辩结束后才能提交课程设计报告，未完成答辩者提交的课程设计报告无效；
3. 课程设计抄袭者，课程设计成绩即总分为 0 分。

目 录

1.课题简介与设计要求

1.1 课题背景

孛儿只斤·铁木真（1162 年 5 月 31 日—1227 年 8 月 25 日），蒙古帝国可汗，尊号“成吉思汗” 意为“拥有海洋四方”。世界史上 杰出的政治家、军事家。 1206 年春天建立大蒙古国，此后多次发动对 外征服战争，征服地域西达中亚、东欧的黑 海海滨。 1227 年在征伐西夏的时候去世，之后被密葬。 除了一生征战，亦创建蒙古文字、颁布文法、 采取开明的宗教政策，是一位在历史上具有 颇多争议的枭雄。 其后代传承对普通百姓而言，扑朔迷离…… 让 我 们 拨 开 迷 雾 ！

本课题要求完成一个简单的成吉思汗家谱存储软件，实现家谱的读取、增添、修改、查询、删除、保存等功能。在家谱查询软件中，把成吉思汗家谱中的每个成员的姓名、出生年份、死亡年份，以及相应的父子关系保存起来。虽然这款软件与正式的我们平常用的信息存储软件在功能上和界面上有些差别，但是基本功能都包括信息的读取、增添、修改、查询、删除、保存等。

1.2 课题基本功能

通过分析，系统应包含以下功能：

（1）.初始化系统：读取族谱数据文件（txt 文件），建立初始 族谱。族谱数据文件预先手工建立。

（2）.显示族谱：输出族谱。

（3）.添加成员：提示输入成员信息，在族谱中添加成员。

（4）.修改成员：提示输入成员姓名，修改成员信息。

（5）.删除成员：提示输入成员姓名，从族谱中删除成员。

（6）.查询：根据指定条件查询数据并输出。

（7）保存：保存当前族谱数据到文件（txt 文件）。

（8）退出：退出系统。

【说明】

1）成员基本信息：姓名，生年，卒年。

2）显示族谱：输出成员姓名，注意可读性。

3）添加、修改成员：注意长幼有序的问题

4）查询：提供按姓名查询、按“代”查询、查询孩子、查询 父亲、查询兄弟等五种方式，输出结果注意可读性。

5）铁木真族谱数据文件： Genghis Khan.txt

2.总体设计

要求系统功能完善，并且使用方便，所以系统应该有一个菜单来方便操作，并且具有必要的提示信息，供用户进行选择。系统主界面设置项分别为：显示族谱、添加成员、修改成员、删除成员、查询信息、退出系统、恢复默认。

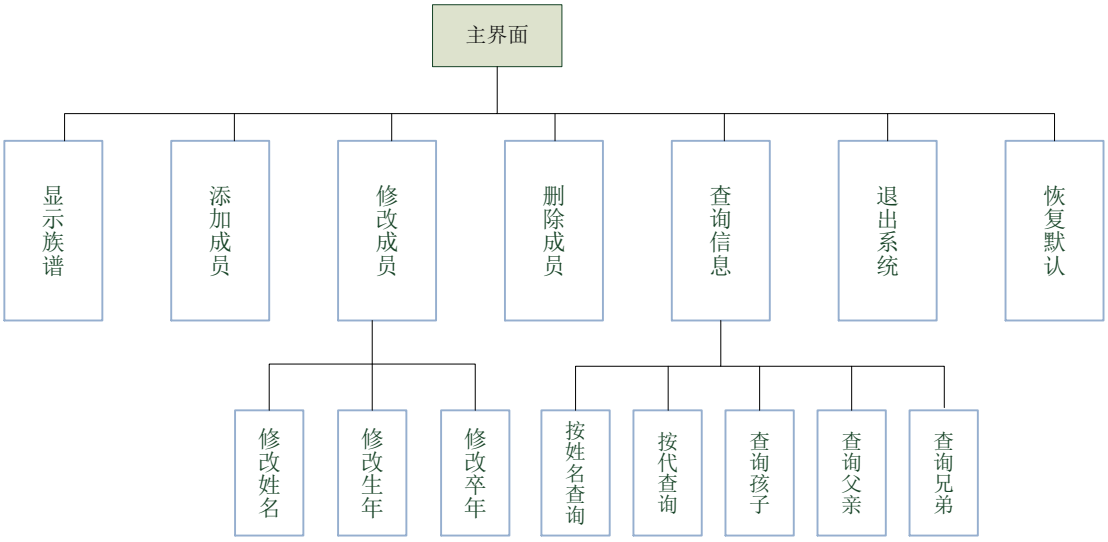

图 2-1 系统功能模块图

3.详细设计

3.1 涉及的知识点

（1）文件的读取和写入

（2）用孩子——兄弟表示法存储树

（3）树的创建

（4）树中元素的增加、删除、修改

（5）树的层序遍历

3.2 结点类型的设计

树中的每一个结点包含了族谱中一个成员的信息域以及两个指针域。其中，信息域中包含了成员的姓名（字符串）、成员的出生年份（整数）、成员的死亡年份（整数）；指针域包含了两个指针，分别是指向该结点第一个孩子结点和下一个兄弟结点。

3.3 采用的逻辑结构

成吉思汗家谱软件主要是用来存储成吉思汗家族中各成员的相关信息以及相互的父子或者是兄弟关系。本软件要求明确而合理地表示成吉思汗家族中的成员之间的关系，可以采用树这种逻辑结构。每个成员可以看作树中的一个结点。

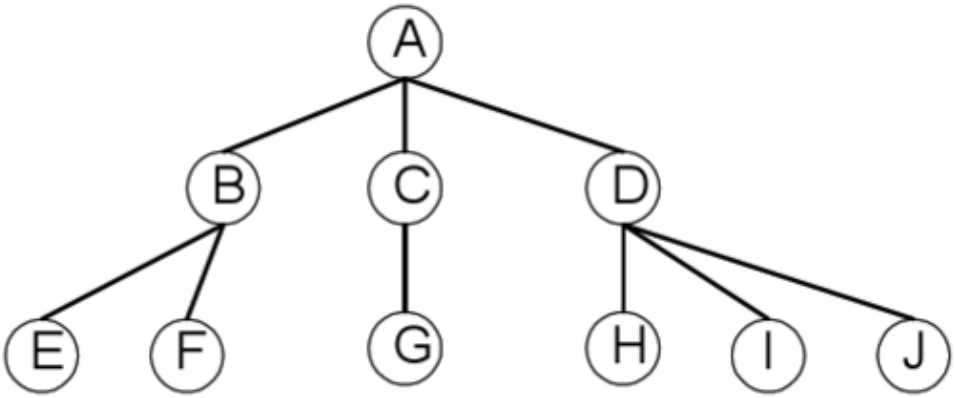

图 3-1 树的逻辑结构

树是一种一对多的非线性结构。这种结构的特点是：除了根结点与叶子结点之外，每个结点有且只有一个前驱，而每个结点有多个后继。对应于成吉思汗家谱中的关系也就是除了没有前一代的和没有后一代的人，每个人都有且仅有一个父亲，但是每个人可能会有多个孩子。因此，使用树作为本软件的逻辑结构是比较妥当的。

对于别的逻辑结构，例如线性表和图。虽然它们也能很好地存储每个人的相关信息，但是它们不能很好地表示祖谱中各个成员之间的辈分关系，不符合本软件的需求，所以用这些作为逻辑结构是不恰当的。

3.4 采用的存储结构

树的存储分为四种：双亲表示法、孩子表示法、双亲——孩子表示法、孩子——兄弟表示法。

双亲表示法指用一组连续的存储空间存储树中的每一个结点，数组中的一个元素表示为树中的一个结点。在数组元素中除包括结点本身的数据信息外，还保存该节点的双亲结点在数组

中的序号（根结点的双亲值赋予-1）。

	0	1	2	3	4	5	6	…	n
data									
parent									

图 3-2 树的双亲表示法

树的双亲表示法对于实现求双亲操作很方便，但对于求某结点的孩子结点的操作，则需要查询整个数组。另外，这种存储方式不能直接反映兄弟结点之间的关系，所以实现求兄弟的操作也比较困难。因此，这种存储方式不适用于成吉思汗家谱软件。

树的孩子表示法有两种形式。第一，用多重链表表示树，链表中的每一个结点包含一个数据域和多个指针域。数据域存储树节点的自身信息，每个指针指向该结点的孩子结点，通过各个节点反应树中各个节点之间的关系。其中指针域的设置有两种方法。其一，每个结点的指针域等于该结点的度数，它虽然在一定程度上节约了存储空间，但由于链表中各结点是不同构的，各种操作不易实现，所以家谱软件不能采用这种方式。其二，每个结点指针域的个数等于树的度数，这种存储方式下各种操作都容易实现，但由于多数结点的度都小于树的度，它为此付出的代价是存储空间的浪费，所以成吉思汗家谱软件采用这种存储方式也是不合理的。第二，用一维数组顺序存储各结点的信息，并将各结点的孩子信息组成一个单链表。孩子信息链中的每一个结点表示一个孩子结点，它有两个域组成。一个域表示该孩子结点在数组中的序号，另外一个域存储指向兄弟结点的指针。在结点数组中，每个元素包括结点的自身信息以及该结点的信息链表的头指针。这种存储结构在查找信息时不是很方便，因此家谱软件不能用这种存储结构。

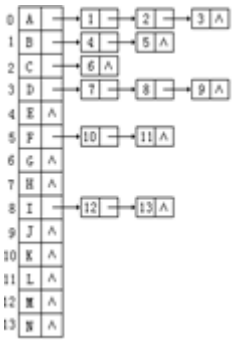

图 3-3 树的孩子表示法

树的双亲——孩子表示法是将双亲表示法与孩子表示法结合起来，分别将各结点的孩子结点组成一个单链表，同时用一维数组顺序存储树中的各结点，数组元素包括结点的自身信息、双亲结点在数组中的序号以及指向该结点的孩子链表的头指针。单链表的每一个结点表示一个孩子结点，它由两个域组成，其中一个域表示该孩子结点在数组中的序号，另外一个域存储指向兄弟结点的指针。

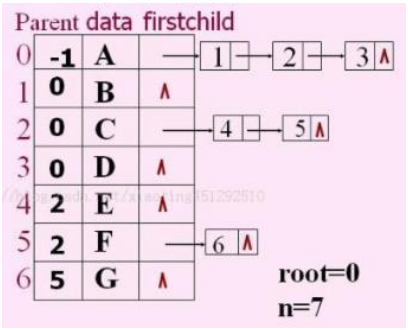

图 3-4 树的双亲——孩子表示法

树的孩子——兄弟表示法是一种链式存储结构，链表中的一个结点代表树中的一个结点。它除信息域外，另外还有两个指针域分别指向该结点的第一个孩子结点和下一个兄弟结点。

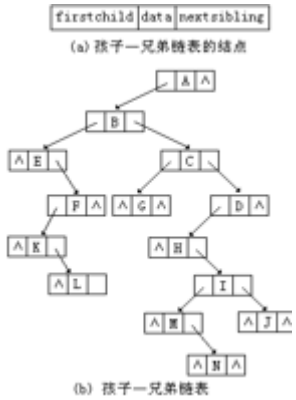

图 3-5 树的孩子——兄弟表示法

考虑到实际的功能需求，祖谱中各成员的存储可以用孩子——兄弟表示法，每个成员是一个结点。结点中包含成员的相关信息，以及指向他的第一个孩子和他的下一个兄弟的指针。有了指针所确定的对应关系，对族谱进行增加、删除、修改、查询就显得方便多了。

3.5 相关算法

插入一个成员就是要先判断根结点是否为空，如果根结点为空，直接把结点插入到根结点处，否则根据姓名找到父亲结点，再做判断。如果父亲结点没有孩子，直接将该结点插入到父亲结点的第一个孩子，否则先找出应该插入的位置，再进行插入。由于要求长幼有序的原则，所以要通过将出生年份进行比较，得出要插入的结点的出生年份在哪个位置更合适，就在这个位置进行插入。插入时，也要进行分情况插入。如果需要插入的位置是第一个孩子结点之前，那么把父亲结点的第一个孩子指针赋值为这个结点，并把这个结点的后一个兄弟指针赋值为原来的第一个孩子结点。如果需要插入的结点为最后一个孩子结点之后，那么把最后一个孩子结点的下一个兄弟指针指向该结点，并把这个结点的下一个兄弟结点指针赋值为空。其他情况就是插入在两个结点之间，首先找到这两个结点，然后把前一个结点的下一个兄弟指针指向这个结点，把这个结点的下一个兄弟指针指向后一个结点，插入即完成。

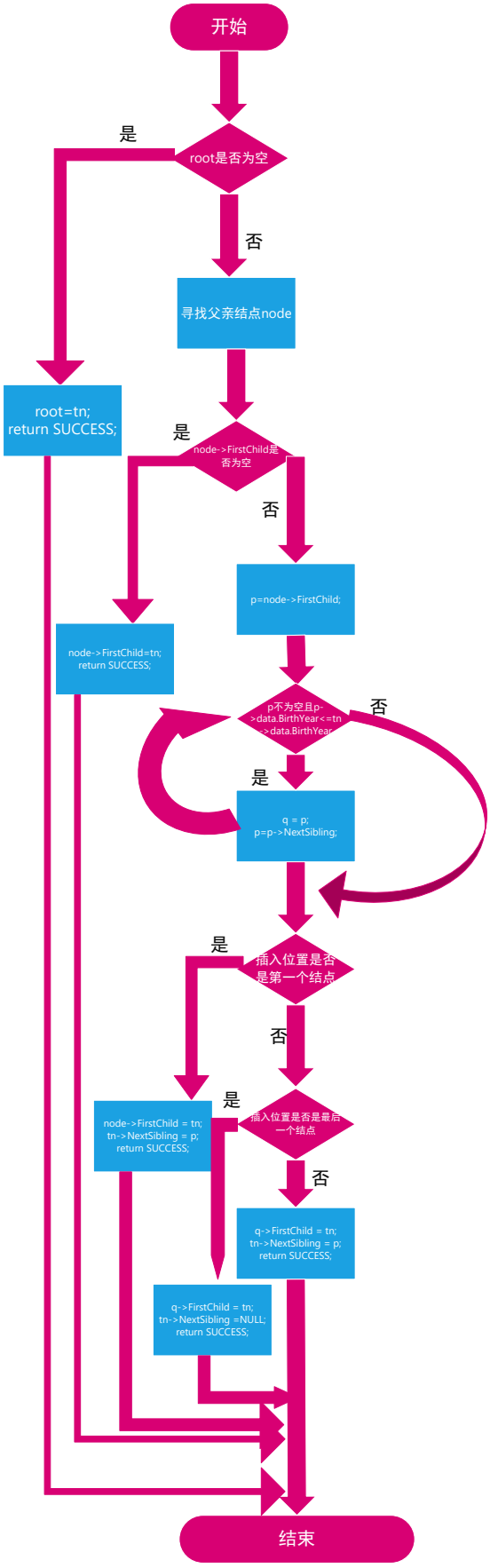
开始
root是否为空
是
否
root=tn;
return SUCCESS;
寻找父亲结点node
node->FirstChild是否为空
是
否
node->FirstChild=tn;
return SUCCESS;
p=node->FirstChild;
p不为空且p->data.BirthYear<=tn->data.BirthYear
否
是
q = p;
p=p->NextSibling;
插入位置是否是第一个结点
是
否
node->FirstChild = tn;
tn->NextSibling = p;
return SUCCESS;
插入位置是否是最后一个结点
是
否
q->FirstChild = tn;
tn->NextSibling = p;
return SUCCESS;
q->FirstChild = tn;
tn->NextSibling =NULL;
return SUCCESS;
结束

图 3-6 插入成员流程图

删除和插入类似，先找到要删除的结点，如果它有孩子先删除它的所有孩子，再删除这个结点，否则直接删除这个结点，删除的方法也是先改变指针的指向再删除。修改也是同样，先找到这个结点再做修改，也要考虑长幼有序的问题，改变指针的指向关系。

查询功能分为按姓名查找、按代查询、查询孩子、查询父亲、查询兄弟，大部分的方法是先找到它的父亲结点，然后再根据条件进行查找。按照姓名查找是通过姓名找到结点，然后输出这个结点。查询孩子是通过这个结点的第一个孩子指针查找，沿着第一个孩子的下一个兄弟指针依次输出结果。

增加、删除、修改都是在树中进行操作，如果不进行保存，下次启动族谱软件时数据不会更改。因此，在进行完操作之后要进行保存，把操作后的结果写入文件中。通过 saving 函数进行操作，写入到 Genghis Khan.txt 中。

4.编码

4.1 数据结构定义

```
struct TreeNode
{
public:
    DATA data;//数据域
    TreeNode *FirstChild;//孩子指针
    TreeNode *NextSibling;//兄弟指针
};
struct DATA //数据结构体
{
    string MemName;//姓名
    int BirthYear;//生年
    int DeathYear;//卒年
};
struct Node//队列结点
{
```

```
public:
    ElemType data;
    Node<ElemType>*next;
};
template<class ElemType>
class LinkQueue//队列类
{
protected:
    Node<ElemType>*front, *rear
};
class Tree
{
private:
    struct TreeNode *root;//根结点
};
```

4.2 成员函数的设计

```
struct DATA //数据结构体
{
    DATA()//无参构造函数
    DATA(string name, int year_birth, int year_death)//有参构造函数
    DATA(DATA &d)//拷贝构造函数
};
struct TreeNode
{
    TreeNode();//无参构造函数
```

```
    TreeNode(DATA d,TreeNode*child=NULL,TreeNode*sibling=NULL);//有参构造函数
    //辅助函数
    Status GetName(string &name)const;
};
class Tree
{
public:
    Tree();//无参构造函数
    Tree(DATA &d);//有参构造函数
    virtual ~Tree();//析构函数
    Status Init();//初始化，建立初始的树
    void Menu();//开始界面
protected:
    //辅助函数
    int    GetGeneration(TreeNode *r);//返回结点是第几代
    int GetNumber(TreeNode *r);//返回结点是一代中的第几个
    TreeNode *GetParent(TreeNode *r);//返回结点的双亲节点
    void LOrder();//层序遍历
    void NOrder(TreeNode *r);//普通遍历
    void Print(TreeNode *r );
    Status GetNode(string name, struct TreeNode *&node,struct TreeNode *r)const;//通过姓名查找结点
    Status Saving();//将结果保存到文件中
    Status Insert(struct TreeNode *tn, string ParentName);//插入新的结点
    void Delete(TreeNode *&node);//删除结点
    void Enquire_Generation(int gen);
        //各级菜单内部实现的方法
```

```
    void Choice();//主菜单
    void Menu_1();//一级菜单（显示）
    void Menu_2();//二级菜单（添加）
    void Menu_3();//三级菜单（修改）
    void Menu_4();//四级菜单（删除）
    void Menu_5();//五级菜单（查询）
    void Menu_6();//六级菜单（退出）
    void Menu_7();//七级菜单（恢复默认）
        //子菜单
    void Menu_3_1(TreeNode*&node);//三级子菜单中修改姓名
    void Menu_3_2(TreeNode*&node);//三级子菜单中修改生年
    void Menu_3_3(TreeNode*&node);//三级子菜单中修改卒年
    void Menu_5_1();//五级子菜单中按姓名查询
    void Menu_5_2();//五级子菜单中按代查询
    void Menu_5_3();//五级子菜单中查询孩子
    void Menu_5_4();//五级子菜单中查询父亲
    void Menu_5_5();//五级子菜单中查询兄弟
};
template<class ElemType>
struct Node//队列结点
{
public:
    Node();//无参构造函数
    Node(ElemType e, Node<ElemType>*link = NULL);//有参构造函数
    virtual ~Node();//析构函数
};
```

```
template<class ElemType>
class LinkQueue//队列类
{
public:
    LinkQueue();//构造函数
    virtual ~LinkQueue();//析构函数
    int GetLength()const;//求队列长度
    bool IsEmpty()const;//判断队列是否为空
    void Clear();//清空队列
    void Traverse(void(*Visit)(const ElemType&))const;//遍历队列
    Status DelQueue(ElemType &e);//出队
    Status GetHead(ElemType &e)const;//求队头指针
    Status EnQueue(const ElemType e);//入队
};
```

4.3 程序实现

全部代码实现如下：

```
/*文件：Source.cpp*/
#include "resource.h"
int main(int argc,char *argv[])
{
    system("color 64");//设置背景颜色和字体颜色
    HWND hwnd = GetForegroundWindow();//使 hwnd 代表最前端的窗口
    ShowWindow(hwnd, SW_MAXIMIZE);//最大化  hwnd  所代表的窗口
    SetWindowTextW(hwnd, L"王的传承——数据结构课程设计");//设置窗口标题
HANDLE  handle_out  =  GetStdHandle(STD_OUTPUT_HANDLE);  //获得标准输出设备句柄
      CONSOLE_CURSOR_INFO cci;              //定义光标信息结构体
```

```
    GetConsoleCursorInfo(handle_out, &cci);        //获得当前光标信息
    cci.bVisible = false;            //设置光标为不可见
    SetConsoleCursorInfo(handle_out, &cci);
    Tree t;
    Status s=t.Init();
    if (s==FAILURE)
    {
        ::MessageBoxW(NULL, L"初始化失败", L"注意", MB_ICONERROR | MB_OK);
        return -1;
    }
    t.Menu();
    system("pause");
    return 0;
}
/*文件：resource.h*/
#pragma warning(disable:4996)
#ifndef RESOURCE_HEADER
#define RESOURCE_HEADER
//定义宏
#define _CRT_SECURE_NO_DEPRECATE ""
#define _AFXDLL ""//防止 CString 出错
//头文件
#include <afx.h>//CString 使用
#include <windows.h>//Windows API 使用
#include <iostream>//cin cout 流使用
#include <fstream>//文件读入都出使用
```

```
#include <stdlib.h>
#include <conio.h>
#include <string>//string 字符串使用
#include "Status.h"
#include "Tree.h"
using namespace std;//使用命名空间
//{{NO_DEPENDENCIES}}
// Microsoft Visual C++ generated include file.
// Used by INHERITANCE_OF_KING.rc
#define IDI_ICON1                       101
// Next default values for new objects
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        102
#define _APS_NEXT_COMMAND_VALUE         40001
#define _APS_NEXT_CONTROL_VALUE         1001
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
#endif
/*文件：Status.h*/
#pragma once
#ifndef STATUS_HEADER
#define STATUS_HEADER
#include <string>
using namespace std;
```

```
enum Status{SUCCESS,FAILURE};//定义 Status 枚举类型
struct DATA //数据结构体
{
    string MemName;//姓名
    int BirthYear;//生年
    int DeathYear;//卒年
    DATA()//无参构造函数
    {
        MemName = "";
        BirthYear = 0;
        DeathYear = 0;
    }
    DATA(string name, int year_birth, int year_death)//有参构造函数
    {
        MemName = name;
        BirthYear = year_birth;
        DeathYear = year_death;
    }
    DATA(DATA &d)//拷贝构造函数
    {
        this->MemName = d.MemName;
        this->BirthYear = d.BirthYear;
        this->DeathYear = d.DeathYear;
    }
};
#endif
```

```
/*文件：TreeNode.h*/
#include "Status.h"
using namespace std;
struct TreeNode
{
public:
    DATA data;//数据域
    TreeNode *FirstChild;//孩子指针
    TreeNode *NextSibling;//兄弟指针
    TreeNode();//无参构造函数
    TreeNode(DATA d,TreeNode*child=NULL,TreeNode*sibling=NULL);//有参构造函数
    //辅助函数
    Status GetName(string &name)const;
};
TreeNode::TreeNode():data()
{
    FirstChild = NULL;
    NextSibling = NULL;
}
TreeNode::TreeNode(DATA d, TreeNode*child/* =NULL */, TreeNode*sibling/* =NULL */) :
data(d)//调用数据域结构体的构造函数
{
    FirstChild = child;
    NextSibling = sibling;
    //两个指针赋值
}
Status TreeNode::GetName(string &name)const
```

```
{
    name = data.MemName;
    return SUCCESS;
}
#endif
/*文件：Tree.h*/
#pragma once
#ifndef TREE_HEADER
#define TREE_HEADER
#include <graphics.h>//EasyX 图形库文件
#include "Status.h"
#include "TreeNode.h"
#include "LinkQueue.h"
class Tree
{
private:
    struct TreeNode *root;//根结点
public:
    Tree();//无参构造函数
    Tree(DATA &d);//有参构造函数
    virtual ~Tree();//析构函数
    Status Init();//初始化，建立初始的树
    void Menu();//开始界面
protected:
    //辅助函数
    int    GetGeneration(TreeNode *r);//返回结点是第几代
```

int GetNumber(TreeNode *r);//返回结点是一代中的第几个

TreeNode *GetParent(TreeNode *r);//返回结点的双亲节点

void LOrder();//层序遍历

void NOrder(TreeNode *r);//普通遍历

void Print(TreeNode *r);

Status GetNode(string name, struct TreeNode *&node,struct TreeNode *r)const;//通过姓名查找结点

Status Saving();//将结果保存到文件中

Status Insert(struct TreeNode *tn, string ParentName);//插入新的结点

void Delete(TreeNode *&node);//删除结点

void Enquire_Generation(int gen);

//各级菜单内部实现的方法

void Choice();//主菜单

void Menu_1();//一级菜单（显示）

void Menu_2();//二级菜单（添加）

void Menu_3();//三级菜单（修改）

void Menu_4();//四级菜单（删除）

void Menu_5();//五级菜单（查询）

void Menu_6();//六级菜单（退出）

void Menu_7();//七级菜单（恢复默认）

//子菜单

void Menu_3_1(TreeNode*&node);//三级子菜单中修改姓名

void Menu_3_2(TreeNode*&node);//三级子菜单中修改生年

void Menu_3_3(TreeNode*&node);//三级子菜单中修改卒年

void Menu_5_1();//五级子菜单中按姓名查询

void Menu_5_2();//五级子菜单中按代查询

void Menu_5_3();//五级子菜单中查询孩子

```
    void Menu_5_4();//五级子菜单中查询父亲
    void Menu_5_5();//五级子菜单中查询兄弟
};
TreeNode *Tree::GetParent(TreeNode *r)
{
    //层序遍历
    LinkQueue<TreeNode *>q;
    TreeNode *p = NULL;
    TreeNode *temp;
    if (r==NULL||r==root)//根结点没有父亲， r 为空也返回空
        return NULL;
    if (root) q.EnQueue(root);//根结点入队
    while (!q.IsEmpty())
    {
        q.DelQueue(p);//出队
        TreeNode *t = p->FirstChild;
        while (t)//遍历元素的孩子结点
        {
            if (t == r) return p;//遍历到了所要找的结点，返回
            t = t->NextSibling;//继续遍历
        }
        if (p->FirstChild) q.EnQueue(p->FirstChild);//入队孩子结点
        temp = p->FirstChild;
        while (temp&&temp->NextSibling)
        {
            q.EnQueue(temp->NextSibling);//入队兄弟结点
```

```
            temp = temp->NextSibling;
        }
    }
    return NULL;//没找到，返回空
}
int Tree::GetGeneration(TreeNode *r)
{
    if (r==root)//根结点是第一代
        return 1;
    else
    {
        TreeNode *p = GetParent(r);//父亲结点
        return GetGeneration(p)+1;//递归，代数是父亲结点加 1
    }
}
void Tree::LOrder()
{
    system("cls");
    //使用层序遍历
    LinkQueue<TreeNode *>q;
    TreeNode *p=NULL;
    TreeNode *temp;
    if (root)
        q.EnQueue(root);
    while (!q.IsEmpty())
    {
```

```
            q.DelQueue(p);
            TreeNode *t = GetParent(p);
cout << p->data.MemName << " " << p->data.BirthYear << " " << p->data.DeathYear;//输出信息
            if (p!=root) cout << " " << t->data.MemName;//父亲姓名
            cout << endl;//换行
            if (p->FirstChild)    q.EnQueue(p->FirstChild);
            temp = p->FirstChild;
            while (temp&&temp->NextSibling)
            {
                q.EnQueue(temp->NextSibling);
                temp = temp->NextSibling;
            }
        }
}
Tree::Tree(){root =NULL;}
Tree::Tree(DATA &d){root = new struct TreeNode(d);}
Tree::~Tree(){ delete root;}
Status Tree::GetNode(string name, struct TreeNode *&node, struct TreeNode *r)const
{
    //使用层序遍历
    LinkQueue<TreeNode *>q;
    TreeNode * p=NULL;
    TreeNode * temp;
    if (r)
        q.EnQueue(root);//根结点入队
    while (!q.IsEmpty())
```

```
    {
        q.DelQueue(p);//出队
        string d = "";
        p->GetName(d);
        if (d==name)//根据姓名找到结点
        {
            node = p;//赋值
            return SUCCESS;//返回
        }
        if (p->FirstChild) q.EnQueue(p->FirstChild);//孩子入队
        temp = p->FirstChild;
        while (temp&&temp->NextSibling)
        {
            q.EnQueue(temp->NextSibling);//兄弟入队
            temp = temp->NextSibling;
        }
    }
    return FAILURE;//没找到，失败
}
void Tree::NOrder(TreeNode *r)
{
    if (r) cout << r->data.MemName << endl ;//打印根结点姓名
    else return;//r 为空，返回
    if (r->FirstChild)
    {
        TreeNode *p = r->FirstChild;
```

```
        while (p) //依次打印孩子姓名
        {
            cout << "|";
            for (int i = 1; i < GetGeneration(p);i++)
                cout << "————————";//分割
            NOrder(p);//遍历孩子
            p = p->NextSibling;//遍历到下一个兄弟结点
        }
    }
}
void Tree::Print(TreeNode *r)
{
    if (!r)//根结点不存在，返回
        return;
    else
    {
        int width = 1551;
        int height = 695;
        initgraph(width, height, NOCLOSE | NOMINIMIZE );//初始化分辨率为 1551*695 的绘图环境，并禁止关闭和最小化操作
        setbkcolor(YELLOW);//设置背景色
        cleardevice();//用背景色刷新绘图环境
        HWND hWnd = GetHWnd();//获得窗口句柄
        SetWindowTextW(hWnd, L"王的传承——数据结构课程设计");//设置窗口标题
        settextcolor(GREEN);//设置字体颜色
        BeginBatchDraw();//开始批量绘图
        /*
```

```
这个函数执行后所有的绘图操作暂时不输出到屏幕上
直到 FlushBatchDraw()后才把之前的绘图输出
使用这个函数可以防止绘图时屏幕的闪烁
*/
/*
按照层序遍历方式依次打印祖谱中各个成员
同一代的在同一行，中间以一段间距隔开
*/
LinkQueue<TreeNode *>q;
TreeNode *p = NULL;
TreeNode *temp;
if (root)    q.EnQueue(root);//入队
while (!q.IsEmpty())
{
    q.DelQueue(p);//出队
    int y = 90 * GetGeneration(p) + 15;//设置纵坐标
    int x = 92 * GetNumber(p) + 30;//设置横坐标
    CString name;
    name=p->data.MemName.c_str();
    outtextxy(x, y, name);//在坐标为（x，y）的位置打印成员姓名
    if (p->FirstChild)    q.EnQueue(p->FirstChild);//孩子入队
    temp = p->FirstChild;
    while (temp&&temp->NextSibling)
    {
        q.EnQueue(temp->NextSibling);//兄弟依次入队
        temp = temp->NextSibling;
```

```
        }
    }
    FlushBatchDraw();//执行未完成的绘图任务
    /*
    按照层序遍历方式画出父子关系的连线
    首先将父亲出队
    然后寻找他所有的孩子，将他们之间进行连线
    */
    setlinecolor(MAGENTA);//设置线条颜色
    LinkQueue<TreeNode *>r;
    TreeNode *s = NULL;
    TreeNode *te;
    if (root) r.EnQueue(root);//根结点入队
    while (!r.IsEmpty())
    {
        r.DelQueue(s);//出队
        TreeNode *t = s->FirstChild;//寻找出队元素的孩子
        while (t)//对于每一个孩子结点，把它与它的父亲结点进行连线
        {
            int x_x = 92 * GetNumber(s) + 50;//父亲结点的横坐标
            int y_x = 90 * GetGeneration(s) + 29;//父亲结点的纵坐标
            int x_y = 92 * GetNumber(t) + 52;//孩子结点的横坐标
            int y_y = 90 * GetGeneration(t) + 22;//孩子结点的纵坐标
            line(x_x, y_x, x_y, y_y);//画线
            t = t->NextSibling;//遍历父亲结点的下一个孩子结点
        }
```

```
                if (s->FirstChild)     r.EnQueue(s->FirstChild);
                te = s->FirstChild;
                while (te&&te->NextSibling)
                {
                    r.EnQueue(te->NextSibling);
                    te = te->NextSibling;
                }
            }
            FlushBatchDraw();//执行未完成的绘图任务
            EndBatchDraw();//结束批量绘图
            char c = 0;
            while (c != 27)//当读入 Esc 时返回
                c = _getch();
            closegraph();//关闭绘图环境
        }
}
int Tree::GetNumber(TreeNode *r)
{
    int i = 0;//计数器
    int gen = GetGeneration(r);//第几代
    //采用层序遍历
    LinkQueue<TreeNode *>q;
    TreeNode *p = NULL;
    TreeNode *temp;
    if (root)    q.EnQueue(root);//根结点入队
    while (!q.IsEmpty())
```

```
    {
        q.DelQueue(p);//元素出队
        TreeNode *t = GetParent(p);//找父亲结点
        if (GetGeneration(p) == gen)//第几代相同
        {
            i++;//统计是一代中的第几个孩子
            if (p==r)  break;//统计完毕，跳出循环
        }
        if (p->FirstChild) q.EnQueue(p->FirstChild);//孩子入队
        temp = p->FirstChild;
        while (temp&&temp->NextSibling)
        {
            q.EnQueue(temp->NextSibling);//兄弟入队
            temp = temp->NextSibling;
        }
    }
    return i;//返回结果
}
Status Tree::Insert(struct TreeNode *tn,string ParentName)
{
    if (!root)//根结点为空
    {
        root = tn;//直接赋值
        return SUCCESS;//返回
    }
    else//根结点非空
```

```
{
    struct TreeNode *p = root;
    struct TreeNode *q = root;
    struct TreeNode *node=NULL;
    if (GetNode(ParentName,node,root)==SUCCESS)//找到父亲结点
    {
        if (node->FirstChild)//有孩子
        {
            p = node->FirstChild;
            while (p&&p->data.BirthYear<=tn->data.BirthYear)//找到合适的插入位置
            {
                q = p;
                p=p->NextSibling;
            }//插入在 q 和 p 之间
            if (p)
            {
                if (p==node->FirstChild)//插入在第一个结点
                {
                    node->FirstChild = tn;
                    tn->NextSibling = p;
                    return SUCCESS;
                }
                q->NextSibling = tn; //不是插入在第一个结点
                tn->NextSibling = p;
                return SUCCESS;
            }
```

```
                else //插入在最后一个兄弟结点的末尾
                {
                    q->NextSibling = tn;
                    tn->NextSibling = NULL;
                    return SUCCESS;
                }
                return SUCCESS;
            }
            else//没有孩子
            {
                node->FirstChild = tn;//插入到它的第一个孩子
                return SUCCESS;
            }
        }
        else  return FAILURE;
    }
}
void Tree::Delete(TreeNode *&node)
{
    if (node->FirstChild==NULL)//没有孩子
    {
        if (node==root)//根结点
        {
            delete node;//直接删除
            root = NULL;
            return;//返回
```

```
        }//不是根结点，先访问到这个结点的父亲结点
        TreeNode *p = GetParent(node);//找父亲结点
        TreeNode*q = p->FirstChild;//第一个孩子结点
        TreeNode *t = node;//临时变量
        //通过下一兄弟指针找到这个结点，进行删除
        if (q == node)//要删的是第一个结点
        {
            p->FirstChild = q->NextSibling;//改变指针指向
            delete q;//删除
            return;//返回
        }
        while (q != node)
        {
            t = q;
            q = q->NextSibling;
        }//退出循环时，q 所指结点即为要删除的结点
        t->NextSibling = q->NextSibling;//改变指针指向
        delete q;//删除
        return;//返回
    }
    else//有孩子
    {//先删除孩子，再删除这个结点
        TreeNode*temp = node->FirstChild;//第一个孩子
        TreeNode*q ;
        while (temp)
        {
```

```
        q = temp->NextSibling;//改变指针指向
        delete temp;//删除孩子
        temp = q;//指向下一个孩子
    }//退出循环时，孩子全部删除
    node->FirstChild = NULL;//孩子指针赋值为空
    //下面删除这个结点，与上面的思路相同
    if (node==root)//根结点
    {
        delete node;//直接删除
        root = NULL;
        return;//返回
    }//不是根结点
    TreeNode *p = GetParent(node);//找到父亲结点
    q = p->FirstChild;
    TreeNode *t=node;
    if (q==node)//第一个孩子就是要删除的结点
    {
        p->FirstChild = q->NextSibling;//改变指针指向
        delete q;//删除
        return;//返回
    }
    while (q!=node)//依次向后找，直到找到要删除的结点
    {
        t = q;
        q = q->NextSibling;
    }
```

```
            t->NextSibling = q->NextSibling;//改变指针指向
            delete q;//删除
        }
        return;//返回
}
Status Tree::Init()
{
        ifstream fin("G:\\VS\\C++\\subject designing\\Genghis Khan.txt", ios::in);//打开文件
        if (!fin)    return FAILURE;//打开失败，返回
        else
        {
            string name1,name2=" ";
            int y1, y2;
            fin >> name1 >> y1 >> y2;//读入根结点的姓名、生年、卒年
            DATA dr = { name1, y1, y2 };
            struct TreeNode *p = new struct TreeNode(dr);
            Insert(p,name2);//把根节点添加到树中
            while (!fin.eof())//循环读文件
            {
                fin >> name1 >> y1 >> y2 >> name2;//读入结点的姓名、生年、卒年、父亲姓名
                DATA d = { name1, y1, y2 };
                struct TreeNode *p1 = new struct TreeNode(d);//创建结点
                Insert(p1, name2);//把节点插入到树中
            }
            fin.close();//关闭文件
            return SUCCESS;
```

```
    }
}
Status Tree::Saving()
{
    ofstream fout("G:\\VS\\C++\\subject designing\\Genghis Khan.txt", ios::out);//打开文件
    if (!fout)
        return FAILURE;//打开失败，返回
    //层序遍历
    LinkQueue<TreeNode *>q;
    TreeNode *p = NULL;
    TreeNode *temp;
    if (root) q.EnQueue(root);//入队
    while (!q.IsEmpty())
    {
        q.DelQueue(p);//出队
        TreeNode *t = GetParent(p);
        fout << p->data.MemName << " " << p->data.BirthYear << " " << p->data.DeathYear;//
把结点信息输出到文件中
        if (p != root) fout << " " << t->data.MemName;//不是根结点输出父亲姓名
        if (p->FirstChild) q.EnQueue(p->FirstChild);
        temp = p->FirstChild;
        while (temp&&temp->NextSibling)
        {
            q.EnQueue(temp->NextSibling);
            temp = temp->NextSibling;
        }
```

```
            if (!q.IsEmpty())     fout << endl;//除最后一行外，其余行尾有一个回车
        }
        fout.close();//关闭文件
        return SUCCESS;
}
void Tree::Enquire_Generation(int gen)
{
        int i = 0;//标志
        system("cls");//清屏
        //字符串拼接，在弹出框中显示查找的结果
        CString str1("查询到的数据信息如下:\n");
        CString str2("姓名          生年          卒年\n");
        CString str = str1 + str2;
        CString temp1, temp2, tt, tem;
        LinkQueue<TreeNode *>q;
        TreeNode *p = NULL;
        TreeNode *temp;
        if (root) q.EnQueue(root);//根结点入队
        while (!q.IsEmpty())
        {
            q.DelQueue(p);//出队
            TreeNode *t = GetParent(p);
            if (GetGeneration(p)==gen)//查询到了那一代
            {
                //将查询到的信息保存起来，拼接到字符串中
                temp1.Format(L"%d", p->data.BirthYear);
```

```
            temp2.Format(L"%d", p->data.DeathYear);
            tt = p->data.MemName.c_str();
            tem = tt + L"          " + temp1 + L"          " + temp2;
            str = str + tem;
            i = 1;//标志位改变，暗示有查找结果
        }
        if (p->FirstChild) q.EnQueue(p->FirstChild);//孩子入队
        temp = p->FirstChild;
        while (temp&&temp->NextSibling)
        {
            q.EnQueue(temp->NextSibling);//兄弟入队
            temp = temp->NextSibling;
        }
        if (!q.IsEmpty() && GetGeneration(p) == gen)    str = str + L"\n";//字符串拼接
    }
    //根据是否有查询结果，弹出框显示不同信息
    if (i == 0)
    {
::MessageBoxW(NULL, L"未查询到!", L"注意", MB_ICONERROR | MB_OK);//没有查询结果
        return;
    }
    else
    {
::MessageBoxW(NULL, str, L"查询结果", MB_ICONINFORMATION | MB_OK);//有查询结果
        return;
    }
```

```
}
void Tree::Menu()
{
    system("cls");//清屏
    //打印系统初始欢迎界面
    cout << endl << endl << endl << endl << endl;
    cout << "\t\t\t\t\t**********************************" << endl;
    cout << "\t\t\t\t\t*                                *" << endl;
    cout << "\t\t\t\t\t*                                *" << endl;
    cout << "\t\t\t\t\t*                                *" << endl;
    cout << "\t\t\t\t\t*                                *" << endl;
    cout << "\t\t\t\t\t*                                *" << endl;
    cout << "\t\t\t\t\t*                                *" << endl;
    cout << "\t\t\t\t\t*                                *" << endl;
    cout << "\t\t\t\t\t*          欢迎进入系统          *" << endl;
    cout << "\t\t\t\t\t*                                *" << endl;
    cout << "\t\t\t\t\t*                                *" << endl;
    cout << "\t\t\t\t\t*                                *" << endl;
    cout << "\t\t\t\t\t*                                *" << endl;
    cout << "\t\t\t\t\t*                                *" << endl;
    cout << "\t\t\t\t\t*                                *" << endl;
    cout << "\t\t\t\t\t*                                *" << endl;
    cout << "\t\t\t\t\t*                                *" << endl;
    cout << "\t\t\t\t\t**********************************" << endl;
    Sleep(1000);//暂停 1 秒
    Choice();//调用系统主菜单
```

```
}
void Tree::Choice()
{
LOOP2:  system("cls");//清屏
    //打印系统主菜单界面
    cout << endl << endl << endl << endl << endl;
    cout << "\t\t\t\t\t**********************************" << endl;
    cout << "\t\t\t\t\t*                                *" << endl;
    cout << "\t\t\t\t\t*          ◎1.显示族谱           *" << endl;
    cout << "\t\t\t\t\t*                                *" << endl;
    cout << "\t\t\t\t\t*          ◎2.添加成员           *" << endl;
    cout << "\t\t\t\t\t*                                *" << endl;
    cout << "\t\t\t\t\t*          ◎3.修改成员           *" << endl;
    cout << "\t\t\t\t\t*                                *" << endl;
    cout << "\t\t\t\t\t*          ◎4.删除成员           *" << endl;
    cout << "\t\t\t\t\t*                                *" << endl;
    cout << "\t\t\t\t\t*          ◎5.查询信息           *" << endl;
    cout << "\t\t\t\t\t*                                *" << endl;
    cout << "\t\t\t\t\t*          ◎6.退出系统           *" << endl;
    cout << "\t\t\t\t\t*                                *" << endl;
    cout << "\t\t\t\t\t*          ◎7.恢复默认           *" << endl;
    cout << "\t\t\t\t\t*                                *" << endl;
    cout << "\t\t\t\t\t*                                *" << endl;
    cout << "\t\t\t\t\t**********************************" << endl;
    //打印系统时间
```

```
SYSTEMTIME sy;
GetLocalTime(&sy);//获取系统时间
cout << "\t\t\t\t\t          " << sy.wYear << "年" << sy.wMonth << "月" << sy.wDay << "日" << endl;//打印日期
cout << "\t\t\t\t\t          " << sy.wHour << ":" << sy.wMinute << ":" << sy.wSecond << endl;//打印时间
char c=0;
while (c<'1' || c>'7')//如果不是菜单中的正确选项
{
    fflush(stdin);//清空输入缓冲区
    c = _getch();//从键盘获取读入的字符
}
int selected = c - '0';//把字符转换为菜单中对应的数字
switch (selected)//根据选择的菜单项目跳转到相应的子菜单函数执行相应的功能
{
case 1:
    Menu_1();
    break;
case 2:
    Menu_2();
    break;
case 3:
    Menu_3();
    break;
case 4:
    Menu_4();
    break;
```

```
        case 5:
            Menu_5();
            break;
        case 6:
            Menu_6();
            break;
        case 7:
            Menu_7();
            break;
        default:
            break;
        }
        goto LOOP2;//主菜单函数不能退出，除非用户要求退出
}
void Tree::Menu_1()
{
        system("cls");//清屏
        if (root==NULL)//根结点不存在
        {
            ::MessageBoxW(NULL, L"家谱中已不存在任何结点", L"提示", MB_ICONWARNING | MB_OK);
            return;
        }
        Print(root); //依次打印每个结点
        return;
}
void Tree::Menu_6()
```

```
{
        int n6 = ::MessageBoxW(NULL, L"是否保存", L"提示", MB_ICONQUESTION | MB_YESNOCANCEL);//提示是否保存
        if (n6 == IDYES)//用户按下保存键
        {
            Status s = Saving();//进行保存操作
            if (s == FAILURE)//保存失败，输出失败信息
            ::MessageBoxW(NULL, L"保存失败", L"注意", MB_ICONERROR | MB_OK);
            else//保存成功，输出成功信息
        ::MessageBoxW(NULL, L"已保存", L"提示", MB_ICONINFORMATION | MB_OK);
            exit(0);//退出系统
        }
        else if (n6==IDNO) exit(1); //用户不保存，直接退出系统
        else return; //用户不想退出系统，继续返回主菜单
}
void Tree::Menu_7()
{
int nRe7 = ::MessageBoxW(NULL, L"是否恢复默认值？", L"提示", MB_ICONQUESTION | MB_YESNO);//提示用户是否恢复默认值
    if (nRe7 == IDYES)//用户选择是
    {
        Delete(root);//清空整棵树
        Status s = Init();//重新从文件中载入并构建出整棵树
        if (s == FAILURE)//载入失败，显示错误信息并退出系统
        {
        ::MessageBoxW(NULL, L"恢复默认失败", L"注意", MB_ICONERROR | MB_OK);
            exit(1);
```

```
            }
            else//载入成功，显示成功信息
            {
                ::MessageBoxW(NULL, L"已恢复默认值", L"提示", MB_ICONINFORMATION
| MB_OK);
                return;
            }
        }
        else return;//不恢复默认，直接返回主菜单
}
void Tree::Menu_2()
{
LOOP1:  system("cls");//清屏
        string name1, name2;//自己的名字，父亲的名字
        int year1, year2;//生年，卒年
        //打印输出的界面
        cout << endl << endl << endl << endl << endl;
        cout << "\t\t\t\t\t**********************************" << endl;
        cout << "\t\t\t\t\t*                                *" << endl;
        cout << "\t\t\t\t\t*                                *" << endl;
        cout << "\t\t\t\t\t*                                *" << endl;
        cout << "\t\t\t\t\t*              ┌●姓名：";
        cin >> name1;//输入要添加的人的姓名
        cout << "\t\t\t\t\t*              ┃                 *" << endl;
        cout << "\t\t\t\t\t*              ┃                 *" << endl;
        cout << "\t\t\t\t\t*              ┃ ●生年：";
        cin >> year1;//输入要添加的人的生年
```

```
cout << "\t\t\t\t\t*                ‖                          *" << endl;
cout << "\t\t\t\t\t*                ‖                          *" << endl;
cout << "\t\t\t\t\t*                ‖●卒年：";
cin >> year2; //输入要添加的人的卒年
cout << "\t\t\t\t\t*                ‖                          *" << endl;
cout << "\t\t\t\t\t*                ‖                          *" << endl;
cout << "\t\t\t\t\t*                ┗●父亲姓名：";
cin >> name2;//输入要添加的人的父亲姓名
cout << "\t\t\t\t\t*                                           *" << endl;
cout << "\t\t\t\t\t*                                           *" << endl;
cout << "\t\t\t\t\t*                                           *" << endl;
cout << "\t\t\t\t\t*************************************" << endl;
//拼接字符串，将 string 和 int 类型全部转化为 CString 类型
//把将要添加的人的信息以弹出框形式展现在屏幕上，要求用户确认是否添加
CString str1("数据信息如下:\n");
CString str2("姓名："), str3("生年："), str4("卒年："),str5("父亲姓名："),temp1,temp2,tt1,tt2;
temp1.Format(L"%d", year1);
temp2.Format(L"%d", year2);
tt1 = name1.c_str();
tt2 = name2.c_str();
CString cnn("\n");
CString str = str1 + str2 + tt1 + cnn + str3 + temp1 + cnn + str4 + temp2 + cnn + str5 + tt2;
int re = ::MessageBoxW(NULL, str, L"是否确定添加？", MB_ICONQUESTION | MB_YESNO);//弹出框，询问用户是否确认添加
if (re==IDYES)//选择是，进行添加
{
    DATA d = { name1, year1, year2 };//结构体变量
```

```
        struct TreeNode *p = new struct TreeNode(d);//新建结点
        Insert(p, name2);//插入到树中
    }
    re = ::MessageBoxW(NULL, L"是否继续添加？", L"提示", MB_ICONQUESTION | MB_YESNO);//询问用户是否继续添加
    if (re==IDYES) goto LOOP1; //选择是，跳转到本函数头继续执行
    return;
}
void Tree::Menu_3()
{
LOOP1:  system("cls");//清屏
    cout << endl << endl << endl << endl << endl;
    cout << "\t\t\t\t\t**********************************" << endl;
    cout << "\t\t\t\t\t*       ●需要修改的成员姓名：";//
    string name;
    cin >> name;//输入需要修改的人的姓名
    TreeNode*node_modify;//需要修改的结点
    Status s = GetNode(name, node_modify, root);//根据姓名寻找这个结点
    if (s==FAILURE)//没找到，显示错误
    {
        CString str("未找到!"),temp;
        temp = name.c_str();
        str =temp +str ;
        ::MessageBoxW(NULL, str, L"注意", MB_ICONERROR | MB_OK);
        int nRe = ::MessageBoxW(NULL, L"是否继续修改", L"提示", MB_ICONQUESTION | MB_YESNO);//弹出框询问是否继续修改
        if (nRe==IDYES)//选择是，从本函数头继续执行
```

```
            goto LOOP1;
        return;
    }
    else//找到
    {
        //打印子菜单，让用户选择
        system("cls");
        cout << endl << endl << endl << endl << endl;
        cout << "\t\t\t\t\t**********************************" << endl;
        cout << "\t\t\t\t\t*                                *" << endl;
        cout << "\t\t\t\t\t*                                *" << endl;
        cout << "\t\t\t\t\t*                                *" << endl;
        cout << "\t\t\t\t\t*          ◎1.修改姓名           *" << endl;
        cout << "\t\t\t\t\t*                                *" << endl;
        cout << "\t\t\t\t\t*                                *" << endl;
        cout << "\t\t\t\t\t*                                *" << endl;
        cout << "\t\t\t\t\t*          ◎2.修改生年           *" << endl;
        cout << "\t\t\t\t\t*                                *" << endl;
        cout << "\t\t\t\t\t*                                *" << endl;
        cout << "\t\t\t\t\t*                                *" << endl;
        cout << "\t\t\t\t\t*          ◎3.修改卒年           *" << endl;
        cout << "\t\t\t\t\t*                                *" << endl;
        cout << "\t\t\t\t\t*                                *" << endl;
        cout << "\t\t\t\t\t*                                *" << endl;
        cout << "\t\t\t\t\t*                                *" << endl;
        cout << "\t\t\t\t\t**********************************" << endl;
```

```
            char c = 0;
            while (c<'1'||c>'3')//当输入不正确时，继续输入
            {
                fflush(stdin);
                c = _getch();
            }
            int select = c - '0';//把输入的字符转化为菜单中对应的数字
            switch (select)//根据选择的项目跳转到子菜单函数中继续执行
            {
            case 1:
                Menu_3_1(node_modify);
                break;
            case 2:
                Menu_3_2(node_modify);
                break;
            case 3:
                Menu_3_3(node_modify);
                break;
            default:
                break;
            }
        }
        return;
}
void Tree::Menu_4()
{
```

```
LOOP:    system("cls");//清屏
        cout << endl << endl << endl << endl << endl;
        cout << "\t\t\t\t\t**********************************" << endl;
        cout << "\t\t\t\t\t*       ●需要删除的成员姓名：";
        string name;
        cin >> name;//输入需要删除的成员姓名
        TreeNode*node_delete;//待删除的结点
        Status s = GetNode(name, node_delete, root);//寻找这个结点
        if (s == FAILURE)//没找到，弹出错误信息
        {
            CString str("未找到!"), temp;
            temp = name.c_str();
            str = temp + str;
            ::MessageBoxW(NULL, str, L"注意", MB_ICONERROR | MB_OK);
            int nRe = ::MessageBoxW(NULL, L"是否继续删除？", L"提示",
MB_ICONQUESTION | MB_YESNO);//提示是否继续删除
            if (nRe==IDYES)//选择是，跳转到本函数开头继续执行
                goto LOOP;
            return;
        }
        else//找到了结点
        {
            //字符串拼接，弹出框显示即将删除的节点信息
            //让用户选择是否确认删除
            CString s1("即将删除的结点：\n"), s2("  姓名："), s3("  生年："), s4("  卒年：");
            CString tem,tt1,tt2,s;
            tem = name.c_str();
```

```
            tt1.Format(L"%d", node_delete->data.BirthYear);
            tt2.Format(L"%d", node_delete->data.DeathYear);
            s = s1 + s2 + tem + L"\n" + s3 + tt1 + L"\n" + s4 + tt2 ;
            int n=::MessageBoxW(NULL, s, L"是否确认删除？", MB_ICONQUESTION | MB_YESNO);//让用户选择是否确认删除
            if (n==IDNO)//选择否，不删除
            {
                int nRe = ::MessageBoxW(NULL, L"是否继续删除？", L"提示", MB_ICONQUESTION | MB_YESNO);//提示是否继续删除
                if (nRe == IDYES)
                    goto LOOP;
            }
            else//选择是，删除结点
            {
                Delete(node_delete);//进行删除操作
                //字符串拼接，提示用户结点已删除
                CString str("已删除!"), temp;
                temp = name.c_str();
                str = temp + str;
                ::MessageBoxW(NULL, str, L"提示", MB_ICONINFORMATION | MB_OK);
                int nR = ::MessageBoxW(NULL, L"是否继续删除？", L"提示", MB_ICONQUESTION | MB_YESNO);//让用户选择是否继续删除
                if (nR == IDYES)//用户选择是
                    goto LOOP;//转到本函数头继续执行
            }
        }
        return;
```

```
}
void Tree::Menu_5()
{
    system("cls");//清屏
    //打印菜单，让用户选择查询方式
    cout << endl << endl << endl << endl << endl;
    cout << "\t\t\t\t**********************************" << endl;
    cout << "\t\t\t\t*                                *" << endl;
    cout << "\t\t\t\t*          ◎1.按姓名查询         *" << endl;
    cout << "\t\t\t\t*                                *" << endl;
    cout << "\t\t\t\t*          ◎2.按代查询           *" << endl;
    cout << "\t\t\t\t*                                *" << endl;
    cout << "\t\t\t\t*          ◎3.查询孩子           *" << endl;
    cout << "\t\t\t\t*                                *" << endl;
    cout << "\t\t\t\t*          ◎4.查询父亲           *" << endl;
    cout << "\t\t\t\t*                                *" << endl;
    cout << "\t\t\t\t*          ◎5.查询兄弟           *" << endl;
    cout << "\t\t\t\t*                                *" << endl;
    cout << "\t\t\t\t*                                *" << endl;
    cout << "\t\t\t\t*                                *" << endl;
    cout << "\t\t\t\t*                                *" << endl;
    cout << "\t\t\t\t*                                *" << endl;
    cout << "\t\t\t\t**********************************" << endl;
    char c = 0;
    while (c<'1' || c>'5')//如果输入不合法
```

```
    {
        fflush(stdin);//清空输入缓冲区
        c = _getch();//读取键盘输入内容
    }
    int selected = c - '0';//把输入的字符转化为数字
    switch (selected)//根据选择的菜单跳转到相应的子菜单
    {
    case 1:
        Menu_5_1();
        break;
    case 2:
        Menu_5_2();
        break;
    case 3:
        Menu_5_3();
        break;
    case 4:
        Menu_5_4();
        break;
    case 5:
        Menu_5_5();
        break;
    default:
        break;
    }
}
```

```
void Tree::Menu_3_1(TreeNode*&node)
{
    system("cls");//清屏
    cout << endl << endl << endl << endl << endl;
    cout << "\t\t\t\t\t**********************************" << endl;
    cout << "\t\t\t\t\t*       ●修改后的姓名：";
    string name;
    cin >> name;//输入修改后的姓名
    /*
    字符串拼接
    弹出框对比修改前后的数据差异
    提示用户是否确认修改
    */
    CString str1("              修改之前的数据：      修改之后的数据：\n"), str2("姓名："), str3("生年："), str4("卒年："), temp1, temp2,tt1,tt2;
    temp1 = node->data.MemName.c_str();
    temp2 = name.c_str();
    tt1.Format(L"%d", node->data.BirthYear);
    tt2.Format(L"%d", node->data.DeathYear);
    str2 = str2 + temp1 + L"                                  " + temp2 + L"\n";
    str3 = str3 + tt1 + L"                                " + tt1 + L"\n";
    str4 = str4 + tt2 + L"                                " + tt2 + L"\n";
    int nRe=::MessageBoxW(NULL, str1 + str2 + str3 + str4, L"是否确定修改？", MB_ICONQUESTION | MB_YESNO);//弹出框提示用户选择是否确认修改
    if (nRe == IDYES)//如果用户确认了修改
    {
        node->data.MemName = name;//进行修改
```

```
        int re = ::MessageBoxW(NULL, L"是否继续修改?", L"提示", MB_ICONQUESTION | MB_YESNO);//弹出框询问是否继续修改
        if (re == IDYES)//选择是，继续修改
        {
            Menu_3();//调用修改函数
            return;//返回
        }
        else  return;//不继续修改，直接返回
    }
    else return; //用户未确认修改，返回
}
void Tree::Menu_3_2(TreeNode*&node)
{
    system("cls");//清屏
    cout << endl << endl << endl << endl << endl;
    cout << "\t\t\t\t\t**********************************" << endl;
    cout << "\t\t\t\t\t*       ●修改后的生年：";
    int year;
    cin >> year;//输入修改后的生年
    /*
    字符串拼接
    弹出框对比修改前后的数据
    提示用户选择是否确认修改
    */
    CString str1("          修改之前的数据：      修改之后的数据：\n"), str2("姓名："), str3("生年："), str4("卒年："), temp1, temp2, tt1, tt2;
    tt1 = node->data.MemName.c_str();
```

```
temp1.Format(L"%d", node->data.BirthYear);
temp2.Format(L"%d", year);
tt2.Format(L"%d", node->data.DeathYear);
str2 = str2 + tt1 + L"                    " + tt1 + L"\n";
str3 = str3 + temp1 + L"                    " + temp2 + L"\n";
str4 = str4 + tt2 + L"                    " + tt2 + L"\n";
int nRe = ::MessageBoxW(NULL, str1 + str2 + str3 + str4, L"是否确定修改？", MB_ICONQUESTION | MB_YESNO);//弹出框提示用户是否确认修改
if (nRe == IDYES)//用户选择是，修改数据
{
    if (node==root)
    {
        node->data.BirthYear = year;
        int Re = ::MessageBoxW(NULL, L"是否继续修改?", L"提示", MB_ICONQUESTION | MB_YESNO);//提示是否继续修改
        if (Re == IDYES)//选择是，继续修改
        {
            Menu_3();
            return;//返回
        }
        else        return;
    }
    else
    {
        TreeNode *pa=GetParent(node);//获取结点的父亲结点
        TreeNode *p = pa->FirstChild;
        TreeNode *q=NULL;
```

```
if (p == node)//如果要修改的结点是第一个孩子结点，先将父亲结点的孩子指针指向它的后一个结点
{
    pa->FirstChild = node->NextSibling;
    p = NULL;//防止进入下一个循环引发异常
}
while (p)
{

    if (p&&p->NextSibling == node)//如果所修改的结点为下一个结点
    {
        p->NextSibling = node->NextSibling;//将此结点的兄弟指针指向修改的结点的下一个兄弟，即把将要修改的结点隔离开来
        break;
    }
    p = p->NextSibling;
}
p = pa->FirstChild;//重新赋值 p
if (year<=p->data.BirthYear)//如果要插入的位置是第一个位置
{
    pa->FirstChild = node;
    node->NextSibling = p;//插入
    goto LOOP;//跳过下面的步骤，直接进行修改操作
}
while (p)
{
    q = p;//q 保存当前结点的前一个结点
```

```
                p = p->NextSibling;//p 保存当前结点
                if (p&&q->data.BirthYear <= year&&p->data.BirthYear >= year)//如果修改后的数据在两个之间，这就是应该插入的位置
                {
                    q->NextSibling = node;
                    node->NextSibling = p;//插入
                    break;//插入完毕，跳出循环
                }
            }
            if (!p&&q->data.BirthYear<=year)//如果要插入的位置为最后一个位置
            {
                q->NextSibling = node;
                node->NextSibling = NULL;//插入
            }
        LOOP:node->data.BirthYear = year;//修改数据
            int re = ::MessageBoxW(NULL, L"是否继续修改?", L"提示", MB_ICONQUESTION | MB_YESNO);//提示是否继续修改
            if (re == IDYES)//选择是，继续修改
            {
                Menu_3();
                return;//返回
            }
            else return;
        }
    }
    else return;
}
```

```
void Tree::Menu_3_3(TreeNode*&node)
{
    system("cls");//清屏
    cout << endl << endl << endl << endl << endl;
    cout << "\t\t\t\t\t**********************************" << endl;
    cout << "\t\t\t\t\t*       ●修改后的卒年：";
    int year;
    cin >> year;//输入修改后的卒年
    /*
    字符串拼接
    弹出框对比修改前后的数据差异
    提示用户选择是否确认修改
    */
    CString str1("              修改之前的数据：      修改之后的数据：\n"), str2("姓名："), str3("生年："), str4("卒年："), temp1, temp2, tt1, tt2;
    tt1 = node->data.MemName.c_str();
    temp1.Format(L"%d", node->data.DeathYear);
    temp2.Format(L"%d", year);
    tt2.Format(L"%d", node->data.BirthYear);
    str2 = str2 + tt1 + L"                                " + tt1 + L"\n";
    str3 = str3 + tt2 + L"                                 " + tt2 + L"\n";
    str4 = str4 + temp1 + L"                                    " + temp2 + L"\n";
    int nRe = ::MessageBoxW(NULL, str1 + str2 + str3 + str4, L"是否确定修改？", MB_ICONQUESTION | MB_YESNO);//弹出框让用户选择是否确认修改
    if (nRe == IDYES)//确认修改
    {
        node->data.BirthYear = year;//修改数据
```

```
        int re = ::MessageBoxW(NULL, L"是否继续修改?", L"提示", MB_ICONQUESTION | MB_YESNO);//提示用户是否继续修改
        if (re == IDYES)//继续修改，调用修改的函数
        {
            Menu_3();
            return;//返回
        }
        else        return;
    }
    else  return;
}
void Tree::Menu_5_1()
{
LOOP3:  system("cls");//清屏
    cout << endl << endl << endl << endl << endl;
    cout << "\t\t\t\t\t**********************************" << endl;
    cout << "\t\t\t\t\t*       ●需要查询的姓名：";
    string name;
    cin >> name;//输入需要查询的姓名
    TreeNode *temp = NULL;
    Status s = GetNode(name, temp, root);//根据姓名查找结点
    /*
    字符串拼接
    如果没有查询成功，弹出框显示错误
    如果查询成功，弹出框显示查询出来的结果
    */
    CString st;
```

```
    st = name.c_str();
    if (s==FAILURE)//查询失败
    {
        st = st + L"未查询到！";
        ::MessageBoxW(NULL, st, L"注意", MB_ICONERROR | MB_OK);//提示错误信息
    }
    else//查询成功
    {
        CString str1("查询到的数据信息如下:\n");
        CString str2("姓名："), str3("生年："), str4("卒年："), temp1, temp2, tt1;
        temp1.Format(L"%d", temp->data.BirthYear);
        temp2.Format(L"%d", temp->data.DeathYear);
        tt1 = name.c_str();
        CString cnn("\n");
        CString str = str1 + str2 + tt1 + cnn + str3 + temp1 + cnn + str4 + temp2 + cnn ;
        ::MessageBoxW(NULL, str, L"提示", MB_ICONINFORMATION | MB_OK);//显示查询的结果
    }
    //询问是否继续查询
    int nRet = ::MessageBoxW(NULL, L"是否继续查询?", L"提示", MB_ICONQUESTION | MB_YESNO);
    if (nRet == IDYES)//继续查询
        goto LOOP3;//跳转到函数头部继续执行
}
void Tree::Menu_5_2()
{
LOOP4:  system("cls");//清屏
```

```
    cout << endl << endl << endl << endl << endl;
    cout << "\t\t\t\t\t**********************************" << endl;
    cout << "\t\t\t\t\t*       ●需要查询的代：";
    int generation;
    cin >> generation;//输入代的号码
    Enquire_Generation(generation);//根据第几代查询成员信息
    int nRet = ::MessageBoxW(NULL, L"是否继续查询?", L"提示", MB_ICONQUESTION | MB_YESNO);//弹出框提示是否继续查询
    if (nRet==IDYES)//继续查询
        goto LOOP4;//跳转到函数头继续执行
}
void Tree::Menu_5_3()
{
LOOP3:  system("cls");//清屏
    cout << endl << endl << endl << endl << endl;
    cout << "\t\t\t\t\t**********************************" << endl;
    cout << "\t\t\t\t\t*       ●需要查询的成员姓名：";
    string name;
    cin >> name;//输入要查询的姓名
    TreeNode *temp = NULL;
    Status s = GetNode(name, temp, root);//根据姓名找到这个结点
    /*
    字符串拼接
    如果查找失败弹出失败信息
    如果查找成功弹出查找结果
    */
    CString st;
```

```
st = name.c_str();
if (s == FAILURE)
{
    st = st + L"未查询到！";
    ::MessageBoxW(NULL, st, L"注意", MB_ICONERROR | MB_OK);//出错信息
}
else
{
    TreeNode *p = temp->FirstChild;
    CString str1("查询到的孩子信息如下:\n");
    CString str2("姓名        生年        卒年\n");
    CString str = str1 + str2;
    CString temp1, temp2, tt, tem;
    st = name.c_str();
    st = st + L"没有孩子!";
    if (!p)
    ::MessageBoxW(NULL,st , L"注意", MB_ICONERROR | MB_OK);//没有孩子，出错
    else
    {
        while (p)
        {
            temp1.Format(L"%d", p->data.BirthYear);
            temp2.Format(L"%d", p->data.DeathYear);
            tt = p->data.MemName.c_str();
            tem = tt + L"        " + temp1 + L"        " + temp2;
            str = str + tem;
```

```
                p = p->NextSibling;
                if (p)
                    str = str + L"\n";
            }
            ::MessageBoxW(NULL, str, L"查询结果", MB_ICONINFORMATION | MB_OK);//弹出框显示查询结果
        }
    }
    int nRet = ::MessageBoxW(NULL, L"是否继续查询?", L"提示", MB_ICONQUESTION | MB_YESNO);//提示是否继续查询
    if (nRet == IDYES) goto LOOP3; //继续查询，从本函数开头继续执行
}
void Tree::Menu_5_4()
{
LOOP5:  system("cls");//清屏
    cout << endl << endl << endl << endl << endl;
    cout << "\t\t\t\t\t**********************************" << endl;
    cout << "\t\t\t\t\t*       ●需要查询的成员姓名：";
    string name;
    cin >> name;//查询的成员姓名
    TreeNode *temp = NULL;
    Status s = GetNode(name, temp, root);//根据姓名查找结点
    TreeNode *q=GetParent(temp);//找到这个结点的父亲
    /*
    字符串拼接
    如果查询成功，弹出框显示查询出的结果
    如果查询失败，弹出框显示出错信息
```

```
*/
if (q)
{
    CString str1("查询到的父亲信息如下:\n");
    CString str2("姓名： "), str3("生年： "), str4("卒年： "), temp1, temp2, tt1;
    temp1.Format(L"%d", q->data.BirthYear);
    temp2.Format(L"%d", q->data.DeathYear);
    tt1 = q->data.MemName.c_str();
    CString cnn("\n");
    CString str = str1 + str2 + tt1 + cnn + str3 + temp1 + cnn + str4 + temp2 + cnn;
    ::MessageBoxW(NULL, str, L"提示", MB_ICONINFORMATION | MB_OK);//弹出框显示查询结果}
else
{
    CString st;
    st = name.c_str();
    st = st + L"没有父亲!";
::MessageBoxW(NULL, st, L"注意", MB_ICONERROR | MB_OK);//没有父亲，显示错误
}
int nRet = ::MessageBoxW(NULL, L"是否继续查询?", L"提示", MB_ICONQUESTION | MB_YESNO);//提示是否继续查询
if (nRet == IDYES) goto LOOP5; //继续查询，从本函数头开始继续运行
}
void Tree::Menu_5_5()
{
LOOP1:  system("cls");//清屏
cout << endl << endl << endl << endl << endl;
```

```
cout << "\t\t\t\t\t**********************************" << endl;
cout << "\t\t\t\t\t*       ●需要查询的成员姓名：";
string name;
cin >> name;//输入需要查询的成员姓名
int i = 0;
TreeNode *temp = NULL;
Status s = GetNode(name, temp, root);//查找这个结点
/*字符串拼接
查找成功，弹出框显示查找结果
查找失败，弹出框显示出错信息
*/
CString st;
st = name.c_str();
if (s == FAILURE)
{
    st = st + L"未查询到！";
::MessageBoxW(NULL, st, L"注意", MB_ICONERROR | MB_OK);//没查找到这个结点，出错
}
else
{
    TreeNode *q = GetParent(temp);//查找父亲结点
    CString str1("查询到的兄弟信息如下:\n");
    CString str2("姓名        生年        卒年\n");
    CString str = str1 + str2;
    CString temp1, temp2, tt, tem;
    if (q)
```

```
{//查找成功
    q = q->FirstChild;
    while (q)//依次遍历它的孩子结点
    {
        if (q->data.MemName != name)//跳过所查找的结点，寻找它的兄弟节点
        {
            temp1.Format(L"%d", q->data.BirthYear);
            temp2.Format(L"%d", q->data.DeathYear);
            tt = q->data.MemName.c_str();
            tem = tt + L"        " + temp1 + L"        " + temp2;
            str = str + tem;
        }
        q = q->NextSibling;
        if (q&&q->data.MemName == name)
            q = q->NextSibling;
        if (q)
            str = str + L"\n";
    }
    ::MessageBoxW(NULL, str, L"查询结果", MB_ICONINFORMATION | MB_OK);//显示查询结果
    }
    else
        ::MessageBoxW(NULL, L"查询失败！", L"注意", MB_ICONERROR | MB_OK);//没有父亲结点，出错
}
int nRet = ::MessageBoxW(NULL, L"是否继续查询?", L"提示", MB_ICONQUESTION | MB_YESNO);//弹出框询问是否继续查询
```

```
    if (nRet == IDYES) goto LOOP1; //继续查询，跳转到本函数开头继续执行
}
#endif
/*文件：Node.h*/
#pragma once
#ifndef NODE_HEADER_FILE
#define NODE_HEADER_FILE
#include "Status.h"
template<class ElemType>
struct Node
{
public:
    ElemType data;
    Node<ElemType>*next;
    Node();//无参构造函数
    Node(ElemType e, Node<ElemType>*link = NULL);//有参构造函数
    virtual ~Node();//析构函数
};
template<class ElemType>
Node<ElemType>::Node(){next = NULL; }//赋值为空
template<class ElemType>
Node<ElemType>::Node(ElemType e, Node<ElemType>*link /* = NULL */)
{ data = e;       next = link;} //赋值
template<class ElemType>
Node<ElemType>::~Node(){}
#endif // !NODE_HEADER_FILE
```

```
/*文件：LinkQueue.h*/
#pragma once
#ifndef LINK_QUEUE_HEADER
#define LINK_QUEUE_HEADER
#include "Node.h"
#define OVER_FLOW FAILURE
#define UNDER_FLOW FAILURE
template<class ElemType>
class LinkQueue
{
protected:
    Node<ElemType>*front, *rear;//队头、队尾指针
public:
    LinkQueue();//构造函数
    virtual ~LinkQueue();//析构函数
    int GetLength()const;//求队列长度
    bool IsEmpty()const;//判断队列是否为空
    void Clear();//清空队列
    void Traverse(void(*Visit)(const ElemType&))const;//遍历队列
    Status DelQueue(ElemType &e);//出队
    Status GetHead(ElemType &e)const;//取队头指针
    Status EnQueue(const ElemType e);//入队
};
template<class ElemType>
LinkQueue<ElemType>::LinkQueue()
{rear = front = new Node<ElemType>;}//新建结点
```

```
template<class ElemType>
LinkQueue<ElemType>::~LinkQueue(){Clear();//先清空队列
delete front; }//释放队头指针
template<class ElemType>
int LinkQueue<ElemType>::GetLength()const
{
    int count = 0;//计数器
    Node<ElemType> *p = front->next;
    while (p)//从头至尾遍历
    {
        count++;//计数器自增
        p = p->next;
    }
    return count;//返回计数器的值
}
template<class ElemType>
bool LinkQueue<ElemType>::IsEmpty()const
{return front == rear; }//判断队头队尾指针是否指向同一单元
template<class ElemType>
void LinkQueue<ElemType>::Clear()
{
    Node<ElemType> *p = front->next;
    while (p)
    {
        front->next = p->next;
        delete p;//依次删除每一个
```

```
            p = front->next;
        }
        rear = front;//队尾指针归位
}
template<class ElemType>
void LinkQueue<ElemType>::Traverse(void(*Visit)(const ElemType&))const
{
        Node<ElemType> *p = front->next;
        while (p)
        {
            (*Visit)(p->data);//访问
            p = p->next;//遍历下一个
        }
}
template<class ElemType>
Status LinkQueue<ElemType>::EnQueue(const ElemType e)
{
        Node<ElemType> *p = new Node<ElemType>(e);//新建结点
        if (p)
        {
            rear->next = p;
            rear = rear->next;
            return SUCCESS;//正常入队成功
        }
        else  return OVER_FLOW;//异常
}
```

```
template<class ElemType>
Status LinkQueue<ElemType>::GetHead(ElemType &e)const
{
    if (!IsEmpty())
    {
        e = front->next->data;
        return SUCCESS;//成功取出头指针
    }
    else  return UNDER_FLOW;//异常
}
template<class ElemType>
Status LinkQueue<ElemType>::DelQueue(ElemType &e)
{
    if (!IsEmpty())//非空
    {//出队
        Node<ElemType>*p = front->next;
        e = p->data;
        front->next = p->next;
        if (rear == p)
            rear = front;
        delete p;//删除元素
        return SUCCESS;//正常出队
    }
    else  return UNDER_FLOW;//异常
}
#endif // !LINK_QUEUE_HEADER
```

5.测试

5.1 测试用例

程序运行后，会自动把文本文件 Genghis Khan.txt 中的内容读入到孩子兄弟树中，然后进入主菜单界面。

表 5-1 系统初始化功能测试用例

字段名称	描述
标识符	5100
测试项	系统初始化功能调试
输入标准	运行调试
输出标准	系统把 Genghis Khan.txt 中的内容读入到树中，读入成功后提示“欢迎进入系统”，等候 1 秒后转到主菜单界面
测试用例间的关联	5200（程序主菜单功能调试）

程序主菜单功能测试用例如表 5-2 所示

表 5-2 程序主菜单功能测试用例

字段名称	描述
标识符	5200
测试项	程序主菜单功能调试
输入标准	1.输入 1 2.输入 2 3.输入 3 4.输入 4 5.输入 5 6.输入 6 7.输入 7 8.输入其他

输出标准	1.进入显示族谱功能界面 2.进入添加成员功能界面 3.进入修改成员功能界面 4.进入删除成员功能界面 5.进入查询信息子菜单界面 6.提示是否保存，用户选择后退出系统 7.先把整个树删除，然后重新把文本文件的内容读入并构造这棵树 8.没有做任何操作，等待继续输入
测试用例间的关联	5210（显示族谱功能测试） 5220（添加成员功能测试） 5230（修改成员功能测试） 5240（删除成员功能测试） 5250（查询信息功能子菜单测试）

显示族谱功能测试用例如表 5-3 所示

表 5-3 显示族谱功能测试用例

字段名称	描述
标识符	5210
测试项	显示族谱功能测试
输入标准	转向显示族谱的功能模块
输出标准	如果族谱树为空，提示“祖谱中已不存在任何结点”，然后返回主菜单 如果族谱树为空，用树形显示族谱，等待输入 Esc 键，然后返回主菜单
测试用例间的关联	

添加成员功能测试用例如表 5-4 所示

表 5-4 添加成员功能测试用例

字段名称	描述
标识符	5220

测试项	添加成员功能测试
输入标准	根据提示输入要添加的成员姓名，然后按 Enter 键 根据提示输入要添加的成员生年，然后按 Enter 键 根据提示输入要添加的成员卒年，然后按 Enter 键 根据提示输入要添加的成员父亲姓名，然后按 Enter 键
输出标准	输出刚刚输入的成员信息，询问“是否确定添加”让用户选择 如果用户选择是，添加这个成员，询问“是否继续添加”让用户选择；如果用户选择否，询问“是否继续添加”让用户选择 如果用户选择是，继续添加成员 如果用户选择否，返回主菜单
测试用例间的关联	

修改成员功能测试用例如表 5-5 所示

表 5-5 修改成员功能测试用例

字段名称	描述
标识符	5230
测试项	修改成员功能测试
输入标准	根据提示输入要修改的成员姓名，然后按 Enter 键 根据提示选择需要修改的内容，然后根据提示输入需要修改的内容，按 Enter 键
输出标准	如果需要修改的成员不存在，提示未找到 如果需要修改的成员存在，输出菜单提示用户选择需要修改姓名、生年还是卒年，用户选择并输入修改后的内容，系统列出该成员修改前后的内容差异，并询问用户“是否确认修改” 如果用户选择是，修改这个成员并询问“是否继续修改” 如果用户选择否，询问“是否继续修改” 如果用户选择是，继续修改成员 如果用户选择否，返回主菜单

测试用例间的关联	

删除成员功能测试用例如表 5-6 所示

表 5-6 删除成员功能测试用例

字段名称	描述
标识符	5240
测试项	删除成员功能测试
输入标准	根据提示输入需要删除的成员姓名，然后按 Enter 键
输出标准	如果需要删除的成员不存在，提示未找到 如果需要删除的成员存在，删除这个成员，提示已删除 询问用户是否继续删除，如果用户选择是，继续删除 如果用户选择否，返回主菜单
测试用例间的关联	

查询功能子菜单测试用例如表 5-7 所示

表 5-7 查询功能子菜单测试用例

字段名称	描述
标识符	5250
测试项	查询功能子菜单测试
输入标准	1.输入 1 2.输入 2 3.输入 3 4.输入 4 5.输入 5
输出标准	1.进入按姓名查询的界面 2.进入按代查询的界面 3.进入查询孩子的界面 4.进入查询父亲的界面

	5.进入查询兄弟的界面
测试用例间的关联	5251（按姓名查询功能测试） 5252（按代查询功能测试） 5253（查询孩子功能测试） 5254（查询父亲功能测试） 5255（查询兄弟功能测试）

按姓名查询功能测试用例如表 5-8 所示

表 5-8 按姓名查询功能测试用例

字段名称	描述
标识符	5251
测试项	按姓名查询功能测试
输入标准	根据提示输入要查询的成员姓名，然后按 Enter 键
输出标准	如果没有查询到这个成员，提示未查询到 如果查询到了这个成员，输出查询到的信息 询问用户是否继续查询，如果用户选择是，继续查询 如果用户选择否，返回主菜单
测试用例间的关联	

按代查询功能测试用例如表 5-9 所示

表 5-9 按代查询功能测试用例

字段名称	描述
标识符	5252
测试项	按代查询功能测试
输入标准	根据提示输入要查询的代，然后按 Enter 键
输出标准	如果没有查询到这代的成员，提示未查询到

	如果查询到了这代的成员，输出查询到的信息 询问用户是否继续查询，如果用户选择是，继续查询 如果用户选择否，返回主菜单
测试用例间的关联	

查询孩子功能测试用例如表 5-10 所示

表 5-10 查询孩子功能测试用例

字段名称	描述
标识符	5253
测试项	查询孩子功能测试
输入标准	根据提示输入要查询的成员姓名，然后按 Enter 键
输出标准	如果没有查询到这个成员或他没有孩子，提示未查询到 如果查询到了这个成员的孩子，输出查询到的信息 询问用户是否继续查询，如果用户选择是，继续查询 如果用户选择否，返回主菜单
测试用例间的关联	

查询父亲功能测试用例如表 5-11 所示

表 5-11 查询父亲功能测试用例

字段名称	描述
标识符	5254
测试项	查询父亲功能测试
输入标准	根据提示输入要查询的成员姓名，然后按 Enter 键
输出标准	如果没有查询到这个成员或他没有父亲，提示未查询到 如果查询到了这个成员的父亲，输出查询到的信息 询问用户是否继续查询，如果用户选择是，继续查询 如果用户选择否，返回主菜单

测试用例间的关联	

查询兄弟功能测试用例如表 5-12 所示

表 5-12 查询兄弟功能测试用例

字段名称	描述
标识符	5255
测试项	查询兄弟功能测试
输入标准	根据提示输入要查询的成员姓名，然后按 Enter 键
输出标准	如果没有查询到这个成员或他没有兄弟，提示未查询到 如果查询到了这个成员的兄弟，输出查询到的信息 询问用户是否继续查询，如果用户选择是，继续查询 如果用户选择否，返回主菜单
测试用例间的关联	

5.2 程序运行结果

根据功能需求，程序设计了主菜单，主菜单下又有子菜单。程序执行前建立文件 Genghis Khan.txt。开始执行程序时，首先读取文件中的内容，生成树。root 为树中的根结点。生成的主菜单有 7 个选项，1 为显示族谱，2 为添加成员，3 为修改成员，4 为删除成员，5 为查询信息，6 为退出系统，7 为恢复默认，如图 5-13 所示。

图 5-13 程序主菜单

输入数字 1，进入显示族谱界面，如图 5-14 所示。

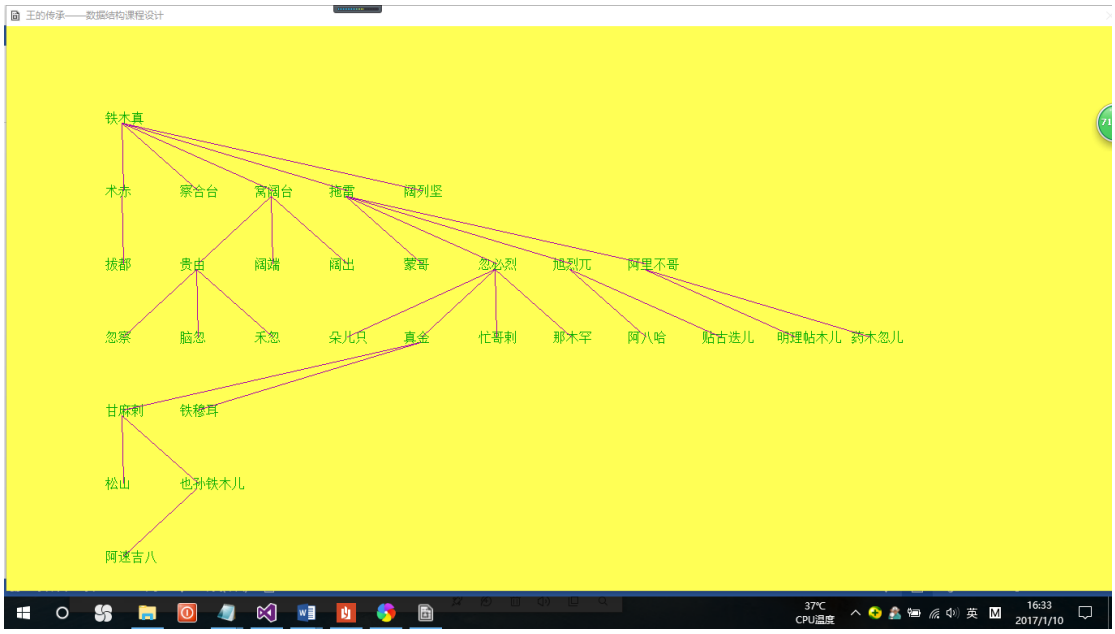

图 5-14 显示族谱的结果

输入数字 2，进入添加成员界面。添加一个成员，姓名：斡儿答 生年：1204 卒年：1280 父亲：术赤，如图 5-15 所示。

图 5-15 添加成员的界面

输入数字 3，进入修改成员界面，然后显示选择修改内容的子菜单，如图 5-16 所示。

图 5-16 选择修改内容子菜单

输入数字 1，进入修改姓名界面，如图 5-17 所示；

输入数字 2，进入修改生年界面，如图 5-18 所示；

输入数字 3，进入修改卒年界面，如图 5-19 所示。

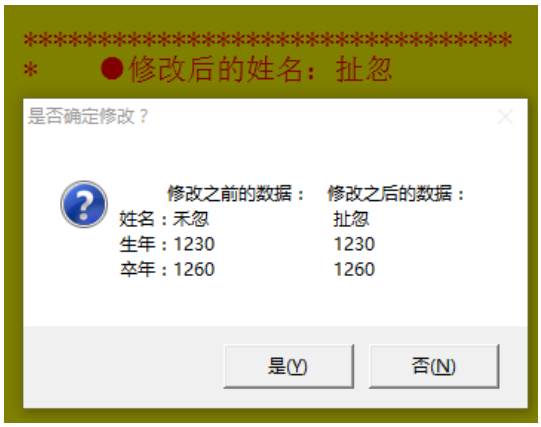

图 5-17 修改姓名的界面

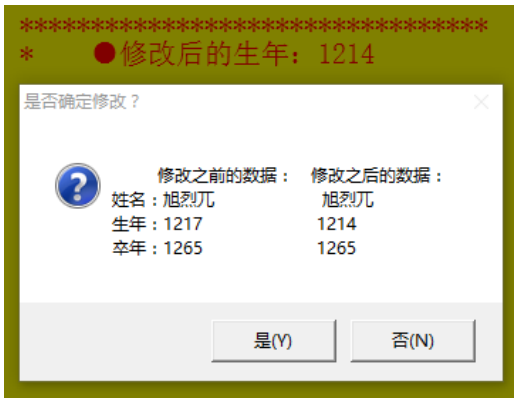

图 5-18 修改生年的界面

图 5-19 修改卒年的界面

输入数字 4，进入删除成员界面，删除一个成员，如图 5-20 所示。

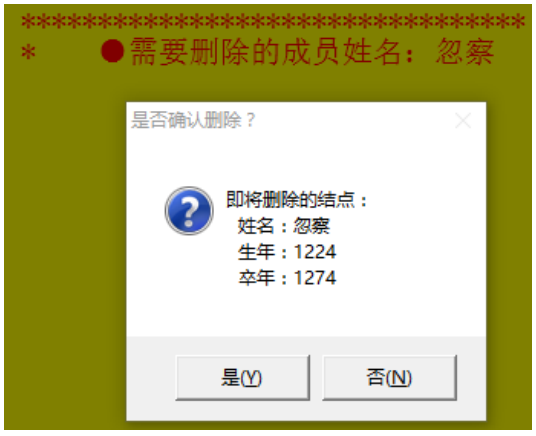

图 5-20 删除成员的界面

输入数字 5，然后进入选择查询信息的方式的子菜单，如图 5-21 所示。

图 5-21 选择查询方式子菜单

输入数字 1，进入按姓名查询界面，如图 5-22 所示；

输入数字 2，进入按代查询界面，如图 5-23 所示；

输入数字 3，进入查询孩子界面，如图 5-24 所示；

输入数字 4，进入查询父亲界面，如图 5-25 所示；

输入数字 5，进入查询兄弟界面，如图 5-26 所示。

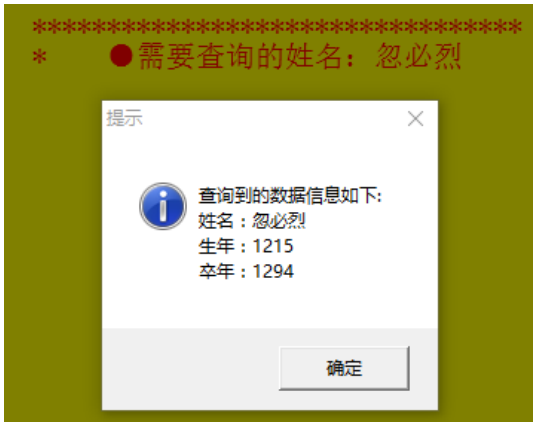

图 5-22 按姓名查询界面

图 5-23 按代查询界面

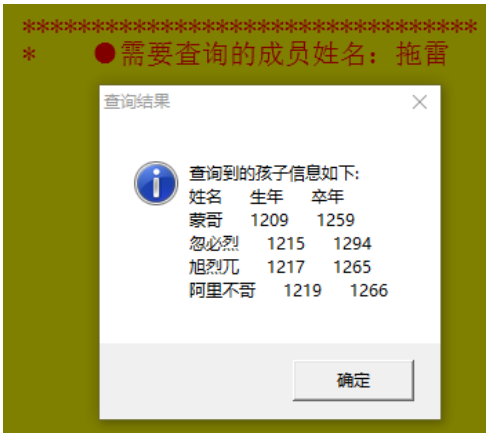

图 5-24 查询孩子界面

图 5-26 查询兄弟界面

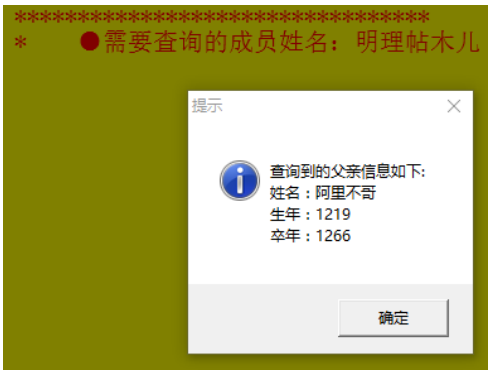

图 5-25 查询父亲界面

6.收获与体会

通过这次课程设计，我对数据结构的使用方法有了一个全新的认识，我能够较好地理解并会运用适当的数据结构来开发具有实际应用价值的软件。这次我就使用了树这种数据结构来设计成吉思汗家谱软件。此外，我的设计开发应用软件的能力也有了一个提升，这次课程设计让我能够掌握在设计时的条理性，有条理地从整体框架到内部实现方法，从一项功能到多项功能进行设计。这次课程设计也锻炼了我解决问题的能力。一开始，我并不能很好地把整棵树建立好，总是出现这样那样的问题。面对这些问题，我利用 Visual Studio 软件进行代码的调试和分析，最终这一系列的错误迎刃而解。以前，我只进行过 C++应用的设计，那只是一种注重具体算法的实现的设计，而这次的数据结构课程设计让我更加注重应用中数据的逻辑结构和存储结构的设计和实现，让我彻底改变对应用程序的认识。从数据结构的学习和这次课程设计中，我明白了数据结构和算法在应用实现时同样重要。除此以外，这次课程设计也培养了我对程序设计的兴趣，为我以后学习其他跟程序有关的课程奠定了基础。

这次课程设计也让我发现问题和解决问题的能力有所增强。我在完成全部的功能时，本来认为已经完美了，但是用一个特殊的数据进行检测却导致了意外的输出，这表明我的程序中还是存在一些问题的。针对这些问题，我利用断点和单步执行的方法定位到可能出错的位置，并对它进行修改。在设计输出族谱的功能时，一开始我只是使用空格和回车进行依次输出，发现父子关系并不能很明显地看出，然后我学习并运用连线把父子关系很好地展现出来了。这种发现问题和解决问题的能力对我今后的开发设计都是有很大帮助的。这次课程设计使我领悟到了理论和实际操作的重要性，学好数据结构理论上的知识才能更好地对一个课题进行完美的设计。然而，仅仅局限于理论知识却是没有意义的，把理论运用到实际的设计开发中来才能发挥价值。我这次就是把平时学到的数据结构理论运用到族谱系统的设计中，进行的最终的设计。

经过一个星期的设计，虽然过程不是一帆风顺的，但是苦尽甘来，经历了一系列的挫折之后我也成功地完成了设计。在这期间我曾经失落过，有时也热情高涨，但是无论如何，我最终设计出了令人满意的成果。在我看来，这次课程设计给我带来的不仅仅是知识上的收获，还有精神上的财富。我通过这次课程设计懂得了要把书上学到的理论知识运用到实际设计上去，要在设计中发现并改正自己的问题，并为以后的学习和设计作铺垫。这次设计给我带来了很多，这些收获使我以后能够更好地学习和做出设计，为我今后的学习和开发应用指导方向。