Project 3
族谱

班级：教务 2 班

宋渝杰 18340146

廖家源 18340105

缪贝琪 18340131

刘依澜 18340121

【题目要求】

构造一棵树来表示族谱，该族谱树要有插入（新加入的人）、删除（被剔除出族谱的人及其后代）、查询（展示查询人物的信息）、统计（展示族谱内每个人的信息）和打印（直观展示该族谱树）的功能。

【数据结构与算法】

数据结构：

我们选择用孩子兄弟表示法来构造这棵族谱树，因为考虑到一个人不应该同时存在于两份族谱中，因此妻子和丈夫并未纳入该族谱树。

先建一个 Node 结构体，用来储存每个结点即每个族谱成员的详细信息（见图 1-1）：其中成员变量包括姓名、性别、年龄、生卒时间、辈分，以及指向父亲、哥哥、弟弟和出生日期最早的孩子的指针；成员函数包括 Node 的构造函数，是为了在 main 函数中先直接构造出一棵有一定规模的基础族谱树，方便之后的操作。

```
struct Node {
    Node() {    }
    Node(string a, bool b, int c, int d, int e, int f, Node* g, Node* h, Node* i, Node* j);//构造函数
    string name;
    bool isMan;//1为男，0为女
    int age; //若死了则为死时的年龄
    int date_of_birth;
    int date_of_death; //死亡日期，-1还活着
    int generation; //辈分
    Node* father;
    Node* old_brother;
    Node* young_brother;
    Node* son;
};
```

图 1-1　Node 结构体成员

然后定义一个名为 Tree 的类（见图 1-2），其成员变量有：Node*类型的变量 root 代表根节点；成员函数有默认构造函数 Tree()和构造了根节点的构造函数 Tree(Node*　r)、插入函数、删除函数、查找函数、查询函数、统计函数以及展示函数。

```
class Tree {
public:
    Node* root;
    Tree();
    Tree(Node* r);
    void Insert();
    void Delete(string x);
    Node* Find(Node* p, string find);
    void Search(Node* p, string find);
    void Statistics(vector<Node*>& vec);
    void Display();
};
```

图 1-2　Tree 类的成员

算法：

该程序除了上文说的函数之外，还有一些辅助函数：展示主页面的 Menu 函数、用来计算族谱树中最大辈分的 max_generation 函数、以及用来直观展示树的递归函数 preorder 函数。

接下来我们会详细介绍一下整体思路以及以上函数的实现方式：

1.在 main 函数中，先利用 Tree 的构造函数构造了一个具有根节点(祖先名为“艾莉”)的族谱树；再通过 Node 的构造函数定义了一些初始的成员，并通过补全他们的关系从而构造了一棵有一定规模的基础树来作为基础族谱，然后调用菜单 Menu 函数，输出直观的菜单，之后通过选择输入 1-6 中的某个数字来进行插入、删除、查询、统计、打印、退出的操作。

2.Insert 函数：首先，输入要插入子女的上一辈的姓名，输入字符串，先通过 Find 函数来查找此人是否在该族谱树中，如果不在则重新输入；其次，输入要插入子女的姓名，输入字符串，然后通过 Find 函数来查找看是否以及存在重名的成员，若有重名则重新输入名字；然后，输入该子女的性别，1 代表男，0 代表女；输入子女的年龄，若此人已逝，则年龄代表其逝世时的年龄；输入子女的出生日期，输入子女的死亡日期，若还活着就输入-1。接着要申请内存空间给新的成员 q，把上述输入的数据都存入 q 的对应变量中，且 q 的 father 指向 p，其他指针暂时为空。接着判断 q 的父代 p 是否已经有子代，如果没有那么 q 就是其父亲 p 的大儿子，p 的 son 应该指向 q；如果父代 p 已经拥有子代，那么 q 会根据出生年龄插入到对应的子代位置中（这可以通过遍历父代 p 的子代，对比出生日期并找到对应位置插入来实现)。

3.Delete 函数：把家谱中除了祖宗之外的某个人及其后代移出家谱，输入一个名字，若此人为祖宗，则操作被拒绝；若不是，则通过 Find 函数，查看这个人是否在族谱中，若不存在，则直接返回，若存在，返回 p，之后删除 p 和他的子代，提示删除成功。

4.Find 函数：用来寻找一个人是否存在于此族谱中，存在则返回此节点，不存在则返回 NULL，利用递归遍历实现，当 p->name 等于我们要寻找的 find 时，直接返回 p，否则继续判断其兄弟及其儿子。

5.Search 函数：用 Find 函数找到我们要寻找的 find，即节点 q，输出 q 的信息：年龄，生卒，辈分，性别，父代，兄弟，孩子。其中输出兄弟的方式：先是判断 q 有无兄弟，若有，则创建一个栈 s1，把 q 的哥哥一个一个压栈，然后再一个一个输出，出栈，直到栈为空，再按顺序输出 q 的弟弟。输出孩子的方式：先判断 q 有无儿子，有则一个一个输出，没有则输出“没有孩子”。

6. Statistics 函数：按辈分排序输出族谱全部成员及信息：先定义一个 vector，为 vec，先将指向树的各个节点的指针存入 vec 中，之后通过双重循环遍历，按辈分顺序输出成员和对应信息。

7. Display 函数：输出直观的族谱树：可通过先序遍历的方式，确定每一行的成员，

之后根据辈分确定成员所在的列位置（在前面输出相应的缩进），最后在输出成员名字之前，输出一些辅助线，将成员与其父代通过辅助线连接起来，形成直观树。

异常处理：

当用户输入错误信息时（例如该输入数字时，用户输入了字符），系统调用 cin.good() 函数判断输入错误，提示“输入错误，请重新输入”，然后调用 cin.clear()、cin.sync() 函数清楚输入流的缓存，并让用户重新输入。

c#程序：

根据前两次项目的优秀 window 窗体应用程序案例，这次我们也用 c#语言编写了一个“族谱”程序。该程序使用多叉树结构，具有以上所述的所有功能，算法和代码类似，在此处不再赘述，只给出程序界面和功能截图。在程序代码压缩包中会给出程序以及 vs 的 c#源码。

图 1-3　c#程序界面

【测试数据、结果及分析】

C++部分：

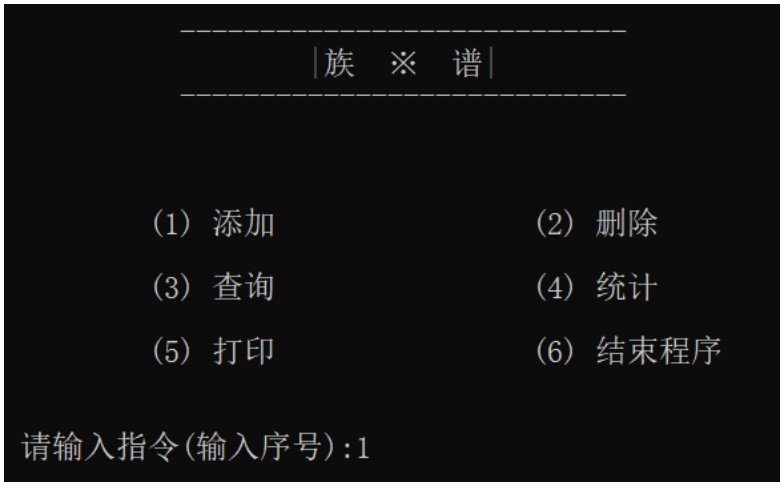

图 2-1　菜单

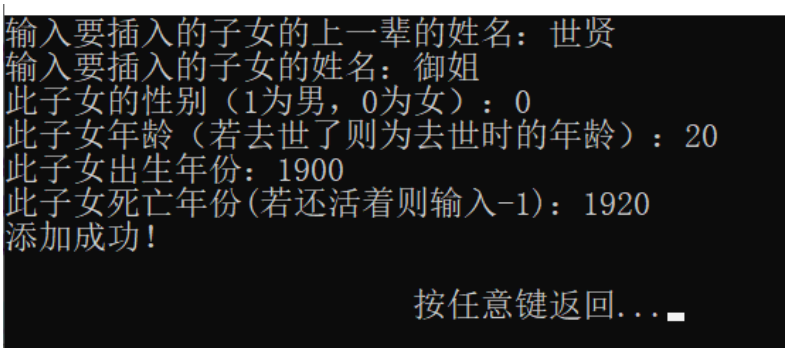

图 2-2　添加节点“御姐”和信息

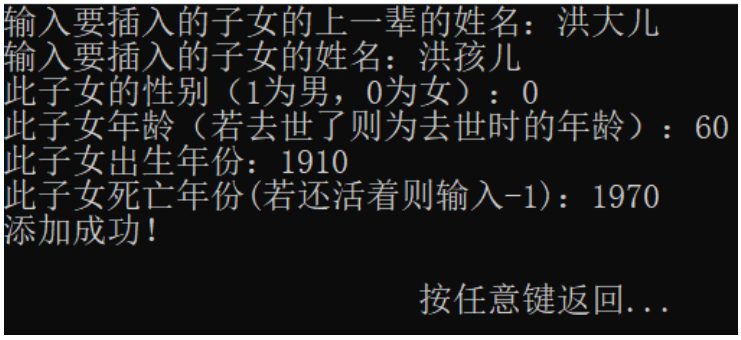

图 2-3　添加节点“洪孩儿”和信息

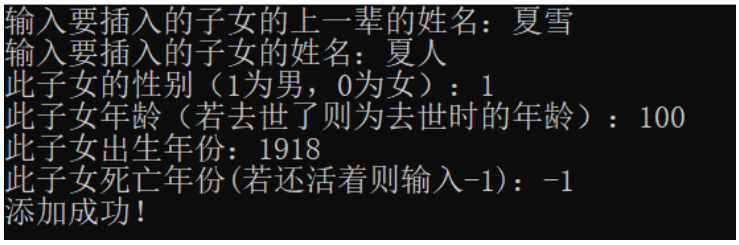

图 2-4　添加节点“夏人”和信息

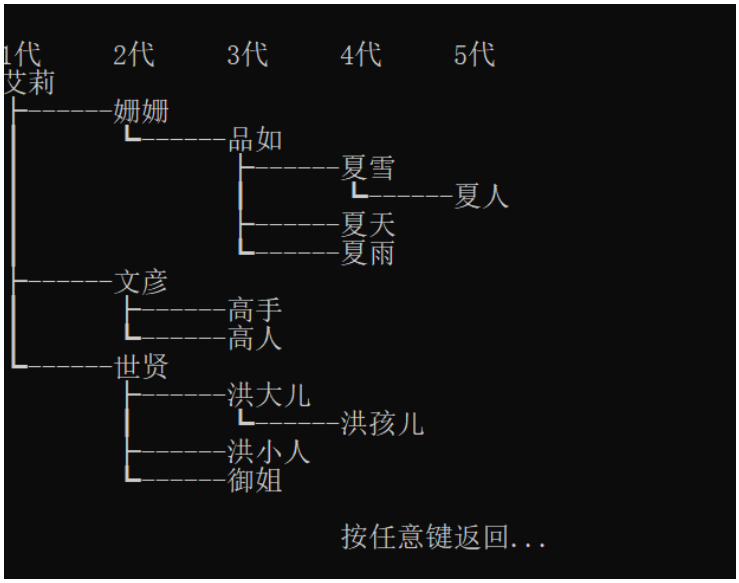

图 2-5　输入 5，查看打印结果

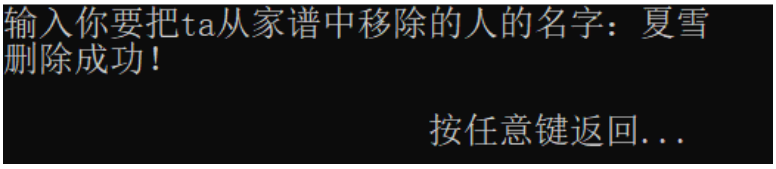

图 2-6　删除夏雪及其后代（成功）

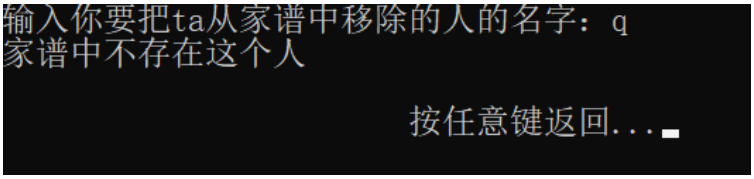

图 2-7　删除 q 及其后代（失败）

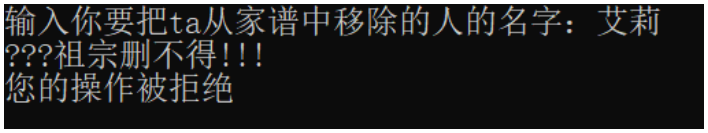

图 2-8　删除艾莉（祖宗不允许被删除，操作失败）

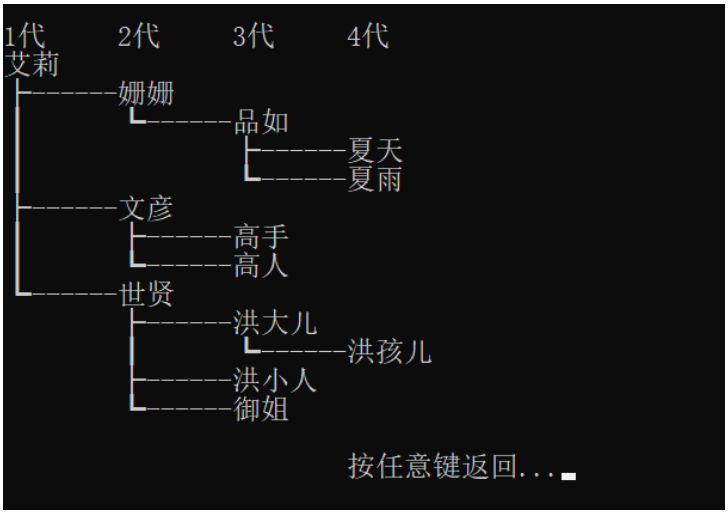

图 2-9　输入 5，查看打印结果

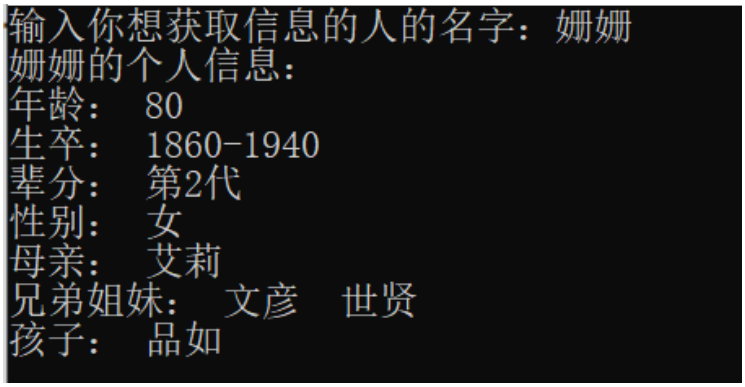

图 2-10　查询姗姗的信息

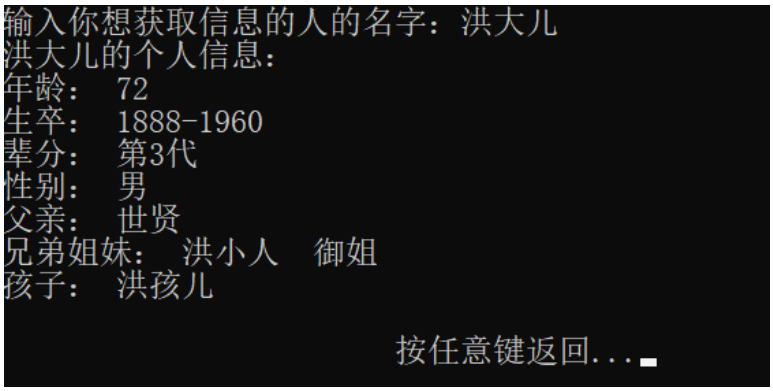

图 2-11　查询洪大儿的信息

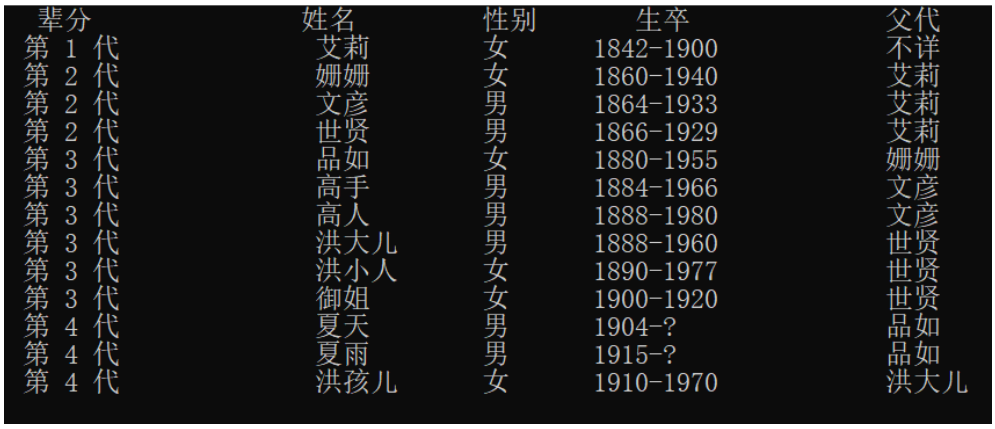

图 2-12　输入 4，查看统计结果

图 2-13　输入错误数据，需要重新输入

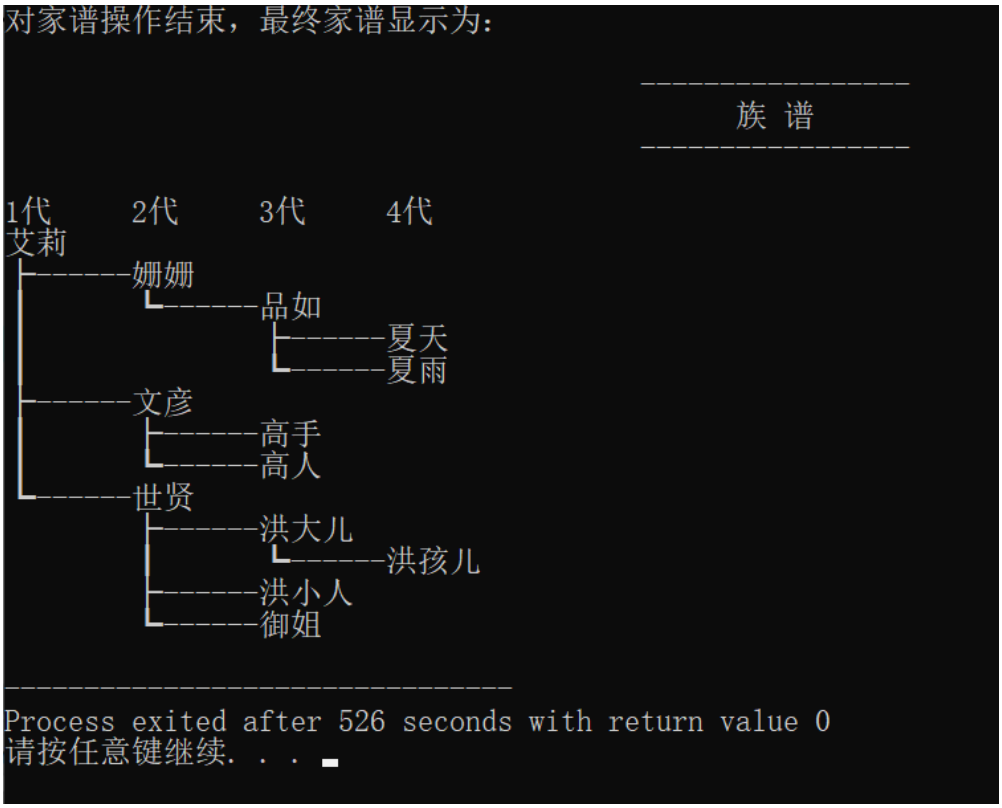

图 2-14　输入 6，退出程序，展示族谱此时的状态

C#部分：

1. 添加子代：在程序左侧的树中单击某个节点（这里是单击“第三代 1 号”），点击“添加子代”按钮，程序弹出添加窗体，提示用户输入添加信息。输入结束后点击“确定”按钮，程序提示“添加成功！”，并且在左侧树中对应位置出现新节点“第四代 1 号”.

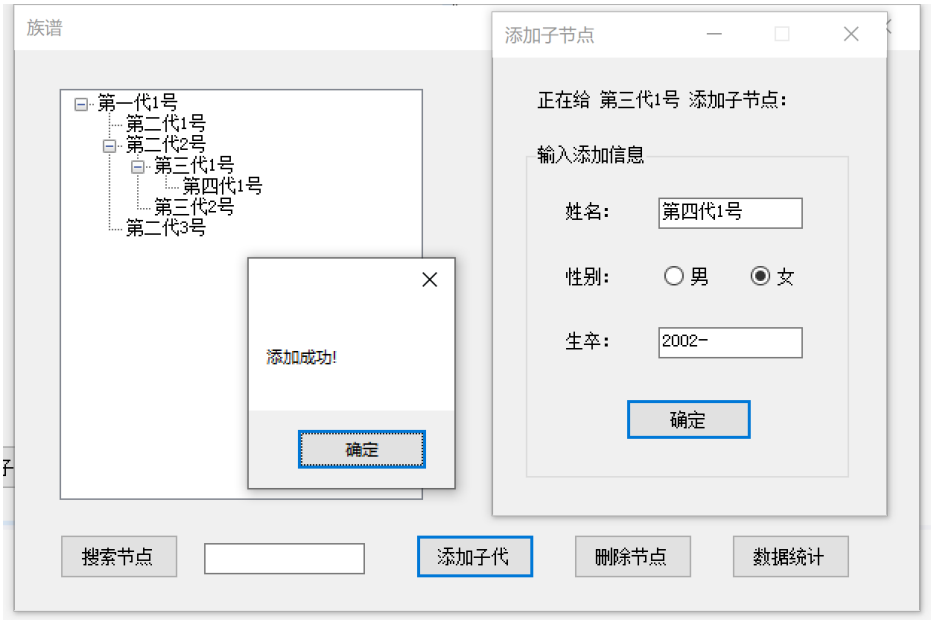

图 3-1　c#添加子代

2. 删除节点：在程序左侧的树中单击某个节点（这里是单击“第二代 2 号”），点击“删除节点”按钮，系统提示“是否确认删除？”，确定删除后程序删除该节点以及它的所有子节点，包括子节点的所有子节点，以此类推。

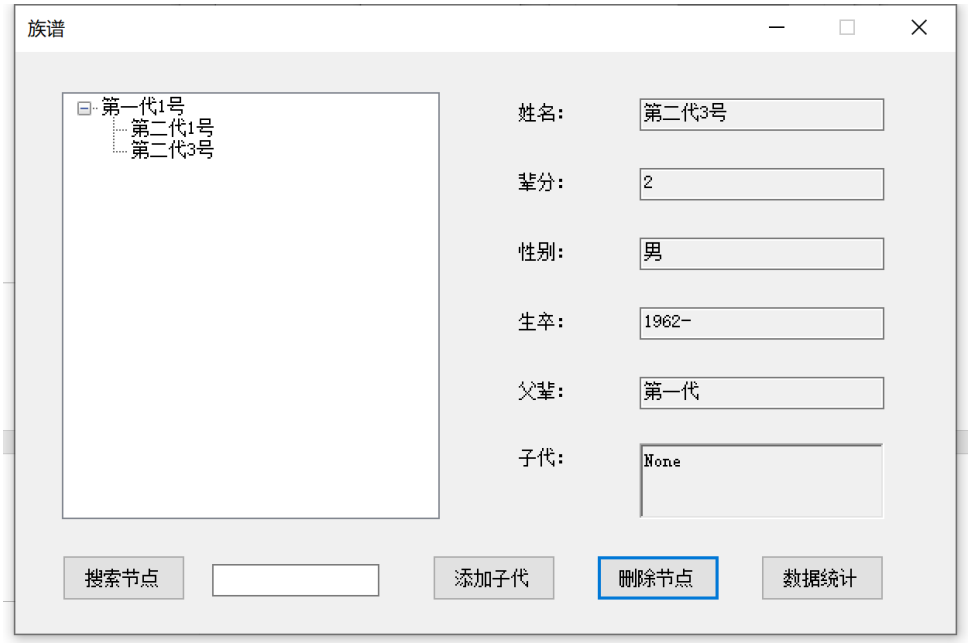

图 3-2 c#删除节点后的结果图

3. 搜索节点：在按钮“搜索节点”右侧的 textBox 输入要搜索的节点的名字（这里是第三代 2 号），点击搜索节点，在左侧的树中会突出显示该节点，右侧信息框也会显示搜索出的节点的对应信息。

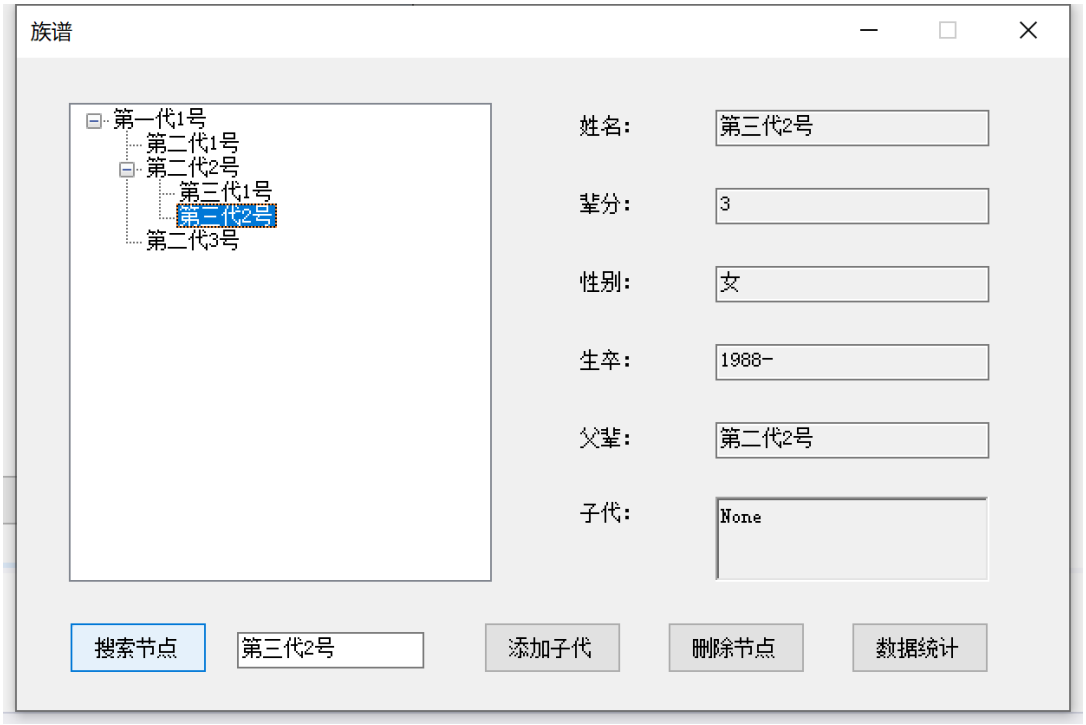

图 3-3 c#搜索节点后的结果图

4. 数据统计：点击“数据统计”按钮，系统弹出窗体，通过层次遍历，显示目前族谱所有节点的信息：

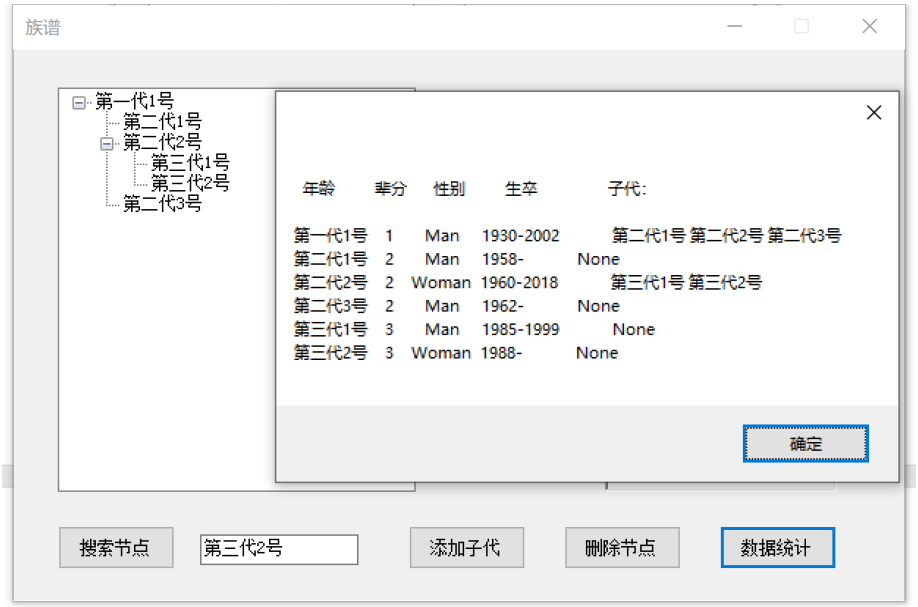

图 3-4 c#数据统计

5. 直观树：在程序的左侧 treeView 控件会实时显示直观的树，在每个操作结束后也会对处理后的树进行实时更新。

【分工、贡献%、自我评分】

分工：

宋渝杰： C#程序的设计和实现，C++代码的后期修改，25%，100 分

廖家源： C++菜单函数、输出直观树的代码编写，25%，100 分

缪贝琪： C++添加、删除函数的编写，以及实验报告的撰写，25%，100 分

刘依澜： C++查找、统计函数的编写，以及实验报告的撰写，25%，100 分

【项目总结】

比起上一次的合作，这次的小组合作有了明显的进步，大家也更加分工明确、团结共进了。但是中途还是遇到了一些困难，比如，在递归的实现上面花费了很多时间，也有很多没有考虑到的情况，一开始打出来的代码有很多 bug，导致后面更改花费了很多时间。希望项目四大家会更加优秀！

【程序清单】

C++部分：

```
#include <iostream>
#include <string>
```

```
#include <vector>
#include <stack>
#include <windows.h>
#include <iomanip>
#include <conio.h>
using namespace std;

void Menu() // 菜单栏
{
    cout << "\n";
    cout << "\t\t\t\t\t  ----------------------------" << endl;
    cout << "\t\t\t\t\t\t  |族  ※  谱|" << endl;
    cout << "\t\t\t\t\t  ----------------------------" << endl;
    cout << "\n\n\n";
    cout << "\t\t\t\t\t(1) 添加";
    cout << "\t\t(2) 删除\n\n";
    cout << "\t\t\t\t\t(3) 查询";
    cout << "\t\t(4) 统计\n\n";
    cout << "\t\t\t\t\t(5) 打印";
    cout << "\t\t(6) 结束程序\n\n";
    cout << "\n\t\t\t\t请输入指令(输入序号):";
}

struct Node { // 树节点
    Node() { }
    Node(string a, bool b, int c, int d, int e, int f, Node* g, Node* h, Node* i, Node* j) {
        name = a;
        isMan = b;
        age = c;
        date_of_birth = d;
        date_of_death = e;
        generation = f;
        father = g;
        old_brother = h;
        young_brother = i;
        son = j;
    }//构造函数
    string name;
    bool isMan;//1为男，0为女
    int age; //若死了则为死时的年龄
    int date_of_birth;
    int date_of_death; //死亡日期，-1还活着
    int generation; //辈分
    Node* father;
```

```
    Node* old_brother;
    Node* young_brother;
    Node* son;
};


class Tree {
public:
    Node* root;
    Tree();
    Tree(Node* r);
    void Insert();
    void Delete(string x);
    Node* Find(Node* p, string find);
    void Search(Node* p, string find);
    void Statistics(vector<Node*>& vec);
    void Display();
};

void max_generation(Node *p,int& max){ // 获取最大辈分
    if(p==NULL) return;
    if(p->generation > max)  max = p->generation;
    max_generation(p->son,max);
    max_generation(p->young_brother,max);
}

void preorder(Node* p, string s) { // 先序遍历输出直观树
    if (p == NULL) return;
    if (p->generation == 1) cout << left << setw(5) << p->name << endl;
    else if (p->young_brother == NULL)  cout << s << "└-------" << left << setw(5) << p->name <<
endl;
    else cout << s << "├-------" << left << setw(5) << p->name << endl;
    Node* son = p->son;
    while (son != NULL) {
        if (p->generation == 1) {
            preorder(son, s);
        }
        else {
            if (p->young_brother != NULL) {
                preorder(son, s + "│\t");
            }
            else {
                preorder(son, s + "\t");
            }
```

```
        }
        son = son->young_brother;
    }
}


Tree::Tree() {
    root = new Node();
    root->name = "艾莉";
    root->age = 58;
    root->date_of_birth = 1842;
    root->date_of_death = 1900;
    root->father = NULL;
    root->generation = 1;
    root->isMan = 0;
    root->son = NULL;
    root->old_brother = NULL;
    root->young_brother = NULL;
}

Tree::Tree(Node* r)
{
    root = new Node();
    root->name = r->name;
    root->age = r->age;
    root->date_of_birth = r->date_of_birth;
    root->date_of_death = r->date_of_death;
    root->father = r->father;
    root->generation = r->generation;
    root->isMan = r->isMan;
    root->son = r->son;
    root->old_brother = r->old_brother;
    root->young_brother = r->young_brother;
}

void Tree::Insert() // 插入函数
{
    string x, y;
    int a, b, c, d;

    cout << "输入要插入的子女的上一辈的姓名：";
    cin >> x;
    Node* p = Find(root, x); //看看这个人在不在现有的族谱里
    Node* q = new Node();
```

```
while (p == NULL)
{
    cout << "此人不存在，请重新输入：";
    cin >> x;
    p = Find(root, x);
}


cout << "输入要插入的子女的姓名：";
cin >> y;
Node* m = Find(root, y);
while (m != NULL) //有重名的情况
{
    cout << "出现重名，请重新输入：";
    cin >> y;
    m = Find(root, y);
}


cout << "此子女的性别（1为男，0为女）：";
cin >> a;
while (!cin.good()) {
    cout << "性别输入错误哦，1为男，2为女，请重新输入：" << endl;
    cin.clear();
    cin.sync();
    cin >> a;
}//判断如果a不是int类型，报错，并重新输入
while (a != 1 && a != 0)
{
    cout << "性别输入错误哦，1为男，2为女，请重新输入：";
    cin >> a;
}


cout << "此子女年龄（若去世了则为去世时的年龄）：";
cin >> b;
while (!cin.good()) {
    cout << "年龄输入错误,请重新输入：" << endl;
    cin.clear();
    cin.sync();
    cin >> b;
}//判断如果b不是int类型，报错，并重新输入
```

```
cout << "此子女出生年份：";
cin >> c;
while (!cin.good()) {
    cout << "出生年份输入错误，请重新输入：" << endl;
    cin.clear();
    cin.sync();
    cin >> c;
}//判断如果不是int类型，报错，并重新输入


cout << "此子女死亡年份(若还活着则输入-1)：";
cin >> d;
while (!cin.good()) {
    cout << "死亡年份输入错误，请重新输入：" << endl;
    cin.clear();
    cin.sync();
    cin >> c;
}//判断如果不是int类型，报错，并重新输入


q->name = y;
q->generation = p->generation + 1;
q->age = b;
q->date_of_birth = c;
q->date_of_death = d;
q->isMan = a;
q->son = NULL;
q->young_brother = NULL;
q->father = p;
if (p->son == NULL) {
    p->son = q;
    q->old_brother = NULL;
}
else {
    Node* o = p->son;     // 这里按照出生日期排序
    while (o->young_brother != NULL and o->young_brother->date_of_birth < q->date_of_birth)
        o = o->young_brother;
    if(o->date_of_birth > q->date_of_birth){
        q->old_brother = NULL;
        q->young_brother = o;
        o->old_brother = q;
        o->father->son = q;
    }
    else if(o->young_brother != NULL){
```

```
            q->young_brother = o->young_brother;
            o->young_brother->old_brother = q;
            o->young_brother = q;
            q->old_brother = o;
        }
        else{
            o->young_brother = q;
            q->old_brother = o;
        }
    }
    cout << "添加成功！" << endl;
}



void Tree::Delete(string x) //把x移出家谱
{
    if (x == "艾莉") {
        cout << "???祖宗删不得!!!" << endl;
        cout << "您的操作被拒绝" << endl;
        return;
    }
    Node* p = Find(root, x);
    if (p == NULL)
    {
        cout << "家谱中不存在这个人" << endl;
        return;
    }
    Node* q = p;
    if (p->father->son == p) {
        q = p->young_brother;
        p->father->son = q;
        q->old_brother = NULL;
        //delete(new)
        delete p;
    }
    else {
        if (p->young_brother != NULL) {
            q = p->young_brother;
            p->old_brother->young_brother = q;
            q->old_brother = p->old_brother;
        }
        else {
            p->old_brother->young_brother = NULL;
```

```
        }
        //delete(new)
        delete p;
    }
    cout << "删除成功！" << endl;
}



Node* Tree::Find(Node* p, string find) // 在族谱中找到名字为find的这个人
{
    Node* q = NULL;
    if (!p)
        return NULL;
    if (p->name == find)
        return p;
    if (p->young_brother)
    {
        q = Find(p->young_brother, find);
        if (q != NULL)
            return q;
    }
    if (p->son)
    {
        q = Find(p->son, find);
        if (q != NULL)
            return q;
    }
    return NULL;
}



void Tree::Search(Node* p, string find) // 查询函数
{
    Node* q = Find(p, find);
    if (q == NULL)
    {
        cout << "不存在这个人" << endl;
        return;
    }
    cout << q->name << "的个人信息：" << endl;
    cout << "年龄： " << q->age << endl;
    //data of birth and death
    cout << "生卒： " << q->date_of_birth << "-";
```

```
if (q->date_of_death == -1) cout << "????" << endl;
else cout << q->date_of_death << endl;
//generation
cout << "辈分： " << "第" << q->generation << "代" << endl;
//sex
cout << "性别： ";
if (q->isMan == 1)  cout << "男" << endl;
else cout << "女" << endl;
//father
if (q->father == NULL) cout << "父代不详" << endl;
else {
    if (q->father->isMan)  cout << "父亲： " << q->father->name << endl;
    else cout << "母亲： " << q->father->name << endl;
}
//brother
if (q->old_brother == NULL && q->young_brother == NULL) {
    cout << "没有兄弟姐妹" << endl;
}
else {
    cout << "兄弟姐妹： ";
    stack<string> s1;
    p = q;
    while (p->old_brother) {
        p = p->old_brother;
        s1.push(p->name);
    }
    while (!s1.empty()) {
        cout << s1.top() << "  ";
        s1.pop();
    }
    p = q;
    while (p->young_brother) {
        p = p->young_brother;
        cout << p->name << "  ";
    }
    cout << endl;
}
//son
p = q->son;
if (p == NULL) {
    cout << "没有孩子" << endl;
}
else {
    cout << "孩子： ";
```

```
        while (p) {
            cout << p->name << "  ";
            p = p->young_brother;
        }
        cout << endl;
    }
}

void Tree::Statistics(vector<Node*>& vec) // 统计函数
{
    vec.push_back(root);
    int max = 0;
    for (int i = 0; i < vec.size(); i++) {
        if (vec[i]->son != NULL) vec.push_back(vec[i]->son);
        if (vec[i]->young_brother != NULL)vec.push_back(vec[i]->young_brother);
    }
    for (int j = 0; j < vec.size(); j++) {
        if(vec[j]->generation > max) max = vec[j]->generation;
    }
    cout << "       辈分            姓名      性别       生卒           父代      " << endl;
    for (int i = 1; i <= max; i++) {
        for (int j = 0; j < vec.size(); j++) {
            if (vec[j]->generation == i) {
                Node* p = vec[j];
                cout << "      第" << right << setw(2) << p->generation << " 代          " << left
<< setw(12) << p->name;
                if (p->isMan) cout << "男    ";
                else cout << "女    ";
                cout << right << setw(5) << p->date_of_birth << "-";
                if (p->date_of_death == -1)  cout << "?     ";
                else cout << left << setw(5) << p->date_of_death << "    ";
                if (p->father == NULL) cout << "不详" << endl;
                else cout << left << setw(12) << p->father->name << endl;
            }
        }
    }
}

void Tree::Display() { // 显示直观树函数
    int max=0;
    Node* p = root;
    int n;
    cout << endl << "\t\t\t\t\t-----------------\t\t\t\t\t" << endl;
    cout << "\t\t\t\t\t      族 谱      \t\t\t\t\t" << endl;
```

```
    cout << "\t\t\t\t\t-----------------\t\t\t\t\t" << endl << endl;
    max_generation(root,max);
    int i;
    for(i=1;i<max;i++){
        cout << i << "代     " ;
    }
    cout << i <<"代" <<endl;
    preorder(root, "");
}

int main()
{
    // 初始化一个族谱
    Tree t;
    Node* shanshan = new Node("姗姗", 0, 80, 1860, 1940, 2, t.root, NULL, NULL, NULL);
    Node* wenyan = new Node("文彦", 1, 69, 1864, 1933, 2, t.root, shanshan, NULL, NULL);
    Node* shixian = new Node("世贤", 1, 63, 1866, 1929, 2, t.root, wenyan, NULL, NULL);
    Node* pinru = new Node("品如", 0, 75, 1880, 1955, 3, shanshan, NULL, NULL, NULL);
    Node* xiaxue = new Node("夏雪", 0, 100, 1900, -1, 4, pinru, NULL, NULL, NULL);
    Node* xiatian = new Node("夏天", 1, 94, 1904, -1, 4, pinru, xiaxue, NULL, NULL);
    Node* xiayu = new Node("夏雨", 1, 85, 1915, -1, 4, pinru, xiayu, NULL, NULL);
    Node* gaoshou = new Node("高手", 1, 82, 1884, 1966, 3, wenyan, NULL, NULL, NULL);
    Node* gaoren = new Node("高人", 1, 92, 1888, 1980, 3, wenyan, gaoshou, NULL, NULL);
    Node* daer = new Node("洪大儿", 1, 72, 1888, 1960, 3, shixian, NULL, NULL, NULL);
    Node* xiaoren = new Node("洪小人", 0, 87, 1890, 1977, 3, shixian, daer, NULL, NULL);
    t.root->son = shanshan;
    shanshan->young_brother = wenyan;
    wenyan->young_brother = shixian;
    daer->young_brother = xiaoren;
    gaoshou->young_brother = gaoren;
    xiaxue->young_brother = xiatian;
    xiatian->young_brother = xiayu;
    shixian->son = daer;
    wenyan->son = gaoshou;
    shanshan->son = pinru;
    pinru->son = xiaxue;

    int x;
    string a;
    while (true)
    {
        Menu();
        cin >> x;
        if (!cin.good()) {
```

```
            cout << "输入错误，请重新输入" << endl;
            cin.clear();
            cin.sync();
            Sleep(500);
            system("cls");
            continue;
        }//判断如果x不是int类型，报错，并重新输入
        system("cls");
        vector<Node*> vec;
        switch (x) {
        case 1:
            t.Insert();
            break;
        case 2:
            cout << "输入你要把ta从家谱中移除的人的名字：";
            cin >> a;
            t.Delete(a);
            break;
        case 3:
            cout << "输入你想获取信息的人的名字：";
            cin >> a;
            t.Search(t.root, a);
            break;
        case 4:
            t.Statistics(vec);
            break;
        case 5:
            t.Display();
            break;
        case 6:
            cout << "对家谱操作结束，最终家谱显示为：" << endl;
            t.Display();
            break;
        default:
            cout << "输入错误，请重新输入" << endl;
            break;
        }
        if (x == 6) break;
        cout << "\n\t\t\t按任意键返回...";
        _getch();
        system("cls");
    }
    return 0;
}
```

C#部分（分 3 个窗体文件代码）

主程序：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace project_3
{
    public struct data
    {
        static public int i = 0;
        static public string name = "";
        static public bool isMan = true;
        static public string nianfen = "";
        static public string name2 = "";
    }
    static class Program
    {
    /// <summary>
    /// 应用程序的主入口点。
    /// </summary>

         [STAThread]

         static void Main()
         {
             Application.EnableVisualStyles();
             Application.SetCompatibleTextRenderingDefault(false);
             Application.Run(new 族谱());
         }
    }
}
```

族谱主窗体：

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
```

```
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace project_3
{
    public partial class 族谱 : Form
    {
        public struct node
        {
            public string name;
            public int beifen;
            public bool isMan;
            public string nianfen;
            public string father;
            public string son;
            public bool check;
        }
        int number = 6;

        List<node> l = new List<node>();

        public 族谱()
        {
            InitializeComponent();
        }

        public void find(TreeNode n,string name)
        {
            if (n.Text == name) {
                treeView1.SelectedNode = n;
                SendKeys.SendWait("{TAB}");
            }
            else
```

```
    {
        for (int i = 0; i < n.Nodes.Count; i++)
        {
            find(n.Nodes[i], name);
        }
    }
}

public void delete(node n, List<node> l2)
{
    int i, j, index = 0;
    node n2 = n;
    n2.check = false;
    for (i = 0; i < l2.Count; i++)
    {
        if(l2[i].name == n2.name)
        {
            index = i;
        }
    }
    l[index] = n2;
    string[] str = n2.son.Split(' ');
    for (i = 0; i < str.Length; i++)
    {
        for (j = 0; j < l2.Count; j++)
        {
            if(l2[j].name == str[i])
            {
                delete(l2[j],l2);
            }
        }
    }
}
private void button1_Click(object sender, EventArgs e)// 点击“添加子节点”按钮
{
```

```
    if(treeView1.SelectedNode == null)
    {
        MessageBox.Show("请选择要添加子节点的节点!");
    }
    else
    {
        添加子节点 a = new 添加子节点();
        data.name2 = treeView1.SelectedNode.Text;
        a.ShowDialog();
    }
}

private void 族谱_Load(object sender, EventArgs e)
{
    timer1.Enabled = true;
    node n = new node();
    n.name = "第一代 1 号";
    n.beifen = 1;
    n.isMan = true;
    n.father = "None";
    n.son = "第二代 1 号 第二代 2 号 第二代 3 号";
    n.nianfen = "1930-2002";
    n.check = true;
    l.Add(n);
    n = new node();
    n.name = "第二代 1 号";
    n.beifen = 2;
    n.isMan = true;
    n.father = "第一代";
    n.son = "None";
    n.nianfen = "1958-";
    n.check = true;
    l.Add(n);
    n = new node();
    n.name = "第二代 2 号";
```

```
n.beifen = 2;
n.isMan = false;
n.father = "第一代";
n.son = "第三代 1 号 第三代 2 号";
n.nianfen = "1960-2018";
n.check = true;
l.Add(n);
n = new node();
n.name = "第二代 3 号";
n.beifen = 2;
n.isMan = true;
n.father = "第一代";
n.son = "None";
n.nianfen = "1962-";
n.check = true;
l.Add(n);
n = new node();
n.name = "第三代 1 号";
n.beifen = 3;
n.isMan = true;
n.father = "第二代 2 号";
n.son = "None";
n.nianfen = "1985-1999";
n.check = true;
l.Add(n);
n = new node();
n.name = "第三代 2 号";
n.beifen = 3;
n.isMan = false;
n.father = "第二代 2 号";
n.son = "None";
n.nianfen = "1988-";
n.check = true;
l.Add(n);
treeView1.ExpandAll();
```

```
        }


        private void treeView1_AfterSelect(object sender, TreeViewEventArgs e)
        {
            int i = Int32.Parse(treeView1.SelectedNode.Name);
            textBox1.Text = l[i].name;
            textBox2.Text = l[i].beifen.ToString();
            textBox3.Text = l[i].isMan ? "男" : "女";
            textBox4.Text = l[i].nianfen;
            textBox6.Text = l[i].father;
            richTextBox1.Text = l[i].son;
        }

        private void timer1_Tick(object sender, EventArgs e)
        {
            if(data.i == 1)
            {
                int j = treeView1.SelectedNode.Nodes.Count;
                treeView1.SelectedNode.Nodes.Add(data.name);
                treeView1.SelectedNode.Nodes[j].Name = number.ToString();
                treeView1.SelectedNode.Nodes[j].Tag                                    =
Int32.Parse(treeView1.SelectedNode.Tag.ToString()) + 1;
                if (l[Int32.Parse(treeView1.SelectedNode.Name)].son == "None")
                {
                    node n2 = new node();
                    n2.name = l[Int32.Parse(treeView1.SelectedNode.Name)].name;
                    n2.beifen = l[Int32.Parse(treeView1.SelectedNode.Name)].beifen;
                    n2.isMan = l[Int32.Parse(treeView1.SelectedNode.Name)].isMan;
                    n2.father = l[Int32.Parse(treeView1.SelectedNode.Name)].father;
                    n2.son = data.name;
                    n2.nianfen = l[Int32.Parse(treeView1.SelectedNode.Name)].nianfen;
                    n2.check = true;
                    l[Int32.Parse(treeView1.SelectedNode.Name)] = n2;
                }
```

```
        else
        {
            node n2 = new node();
            n2.name = l[Int32.Parse(treeView1.SelectedNode.Name)].name;
            n2.beifen = l[Int32.Parse(treeView1.SelectedNode.Name)].beifen;
            n2.isMan = l[Int32.Parse(treeView1.SelectedNode.Name)].isMan;
            n2.father = l[Int32.Parse(treeView1.SelectedNode.Name)].father;
            n2.son = l[Int32.Parse(treeView1.SelectedNode.Name)].son + " " + data.name;
            n2.check = true;
            n2.nianfen = l[Int32.Parse(treeView1.SelectedNode.Name)].nianfen;
            l[Int32.Parse(treeView1.SelectedNode.Name)] = n2;
        }
        treeView1.SelectedNode.Expand();
        number++;
        node n = new node();
        n.name = data.name;
        n.beifen = Int32.Parse(treeView1.SelectedNode.Tag.ToString()) + 1;
        n.isMan = data.isMan;
        n.father = treeView1.SelectedNode.Text;
        n.son = "None";
        n.nianfen = data.nianfen;
        n.check = true;
        l.Add(n);
        int i = Int32.Parse(treeView1.SelectedNode.Name);
        textBox1.Text = l[i].name;
        textBox2.Text = l[i].beifen.ToString();
        textBox3.Text = l[i].isMan ? "男" : "女";
        textBox4.Text = l[i].nianfen;
        textBox6.Text = l[i].father;
        richTextBox1.Text = l[i].son;
        data.i = 0;
    }
}

private void button2_Click(object sender, EventArgs e)// 点击“删除该节点”按钮
```

```
{
    if (treeView1.SelectedNode == null)
    {
        MessageBox.Show("请选择要删除的节点！");
    }
    else if(treeView1.SelectedNode == treeView1.Nodes[0])
    {
        MessageBox.Show("不能删除根节点！");
    }
    else
    {
        DialogResult dr = MessageBox.Show("是否删除该节点？此过程无法撤销！！！\n注：该节点的所有子节点也会被删除", "警告:", MessageBoxButtons.OKCancel, MessageBoxIcon.Information);
        if (dr == DialogResult.OK)   //如果单击“是”按钮
        {
            int index = 0, index2 = 0;
            for(int i = 0; i < l.Count; i++)
            {
                if (l[i].son.Contains(treeView1.SelectedNode.Text))
                {
                    index = i;
                }
            }
            node nn = new node();
            nn.name = l[index].name;
            nn.beifen = l[index].beifen;
            nn.isMan = l[index].isMan;
            nn.father = l[index].father;
            nn.check = true;
            string[] str = l[index].son.Split(' ');
            for(int i = 0; i < str.Length; i++)
            {
                if (str[i] == treeView1.SelectedNode.Text) index2 = i;
            }
            nn.son = "";
```

```
            for(int i = 0; i < str.Length; i++)
            {
                if(i != index2)
                {
                    if (i == 0) nn.son += str[i];
                    else if (nn.son != "")
                    {
                        nn.son = nn.son + " " + str[i];
                    }
                    else
                    {
                        nn.son = nn.son + str[i];
                    }
                }
            }
            if (nn.son == "") nn.son = "None";
            l[index] = nn;
            for (int i = 0; i < l.Count; i++)
            {
                if (l[i].name.Contains(treeView1.SelectedNode.Text))
                {
                    index = i;
                }
            }
            delete(l[index], l);
            treeView1.SelectedNode.Remove();
        }
    }
}

private void button3_Click(object sender, EventArgs e)// 点击“搜索节点”按钮
{
    find(treeView1.Nodes[0], textBox5.Text);
}
```

```
        private void button4_Click(object sender, EventArgs e)// 点击“数据统计”按钮
        {
            string str = "  年龄          辈分       性别          生卒                    子代：\n\n";
            for(int i = 0; i < l.Count; i++)
            {
                if(l[i].check == true)
                {
                    str = str + l[i].name + "    " + l[i].beifen.ToString() + "    " + (l[i].isMan ?
"   Man    " : "Woman  ") + "" + l[i].nianfen + "            " + l[i].son + "\n";
                }
            }
            MessageBox.Show(str);
        }
    }
}
```

添加节点窗体：

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace project_3
{
    public partial class 添加子节点 : Form
    {
        public 添加子节点()
        {
            InitializeComponent();
        }

        private void 添加子节点_Load(object sender, EventArgs e)
        {
            label1.Text = "正在给 " + data.name2 + " 添加子节点：";
            radioButton1.Checked = true;
        }
```

```
        private void button1_Click(object sender, EventArgs e)
        {
            if(textBox1.Text != "")
            {
                data.i = 1;
                data.name = textBox1.Text;
                if (radioButton1.Checked == true) data.isMan = true;
                else data.isMan = false;
                data.nianfen = (textBox2.Text.Length == 0 ? "None" : textBox2.Text);
                MessageBox.Show("添加成功!");
                this.Close();
            }
            else
            {
                MessageBox.Show("名字不能为空!");
            }
        }
    }
}
```