一、 问题描述

（正文使用小四号宋体，1.5 倍行距；每个图应该配有说明；双面打印并装订）

家谱管理系统是查询家谱信息必不可少的一部分，利用家谱管理系统可以清楚地查询成员的详细信息。成员的信息包括：姓名（假设姓名不重复）、出生年月日、目前状况（健在、身故）。请设计合理的数据结构存储该家谱的信息，要求能够实现以下功能：

该系统应具有以下功能（即可以重复操作）：

家谱信息预先设置好，程序启动，从文件中读入家谱信息，自动建立家谱数据结构，使用菜单实现以下功能，每次操作后应该输出当前结果：

（1）插入：给某个人添加一个孩子

（2）删除：删除某个人，如果某人有后代，不允许删除

（3）堂兄弟：根据所在世代的字，可以输出该世代所有人的信息

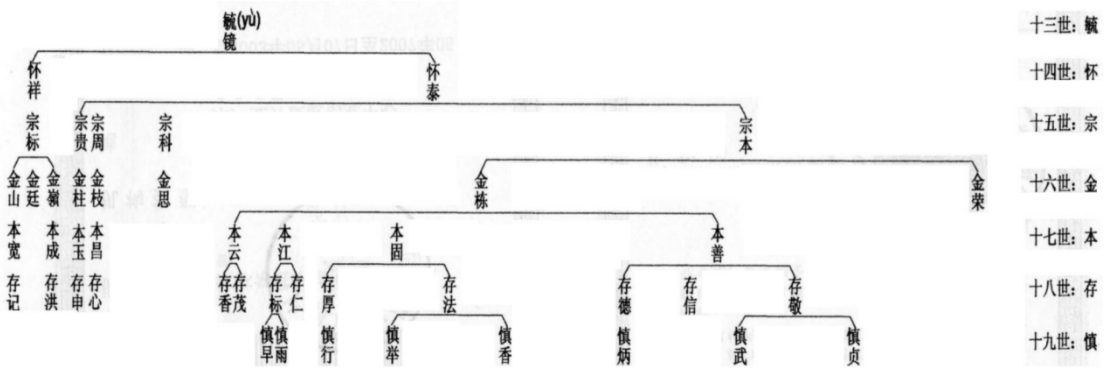
（4）父子：可以根据姓名查询，并输出该人的父亲和孩子信息

（5）显示：以凹入表的形式显示家谱

（6）退出时，家谱信息保存到文件中

测试数据如下：

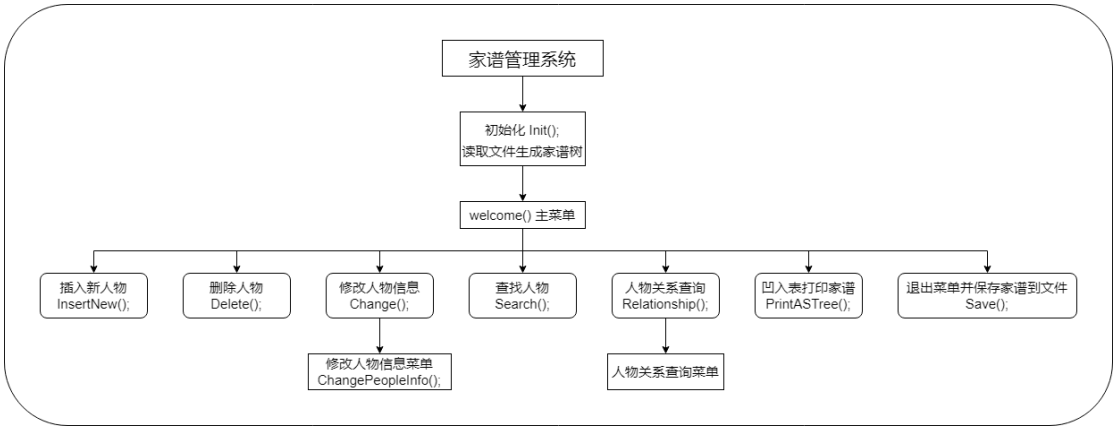
仇氏十三世祖毓镜公系下世代系列表



二、 系统功能说明

2.1 系统功能

（使用功能结构图说明系统功能）

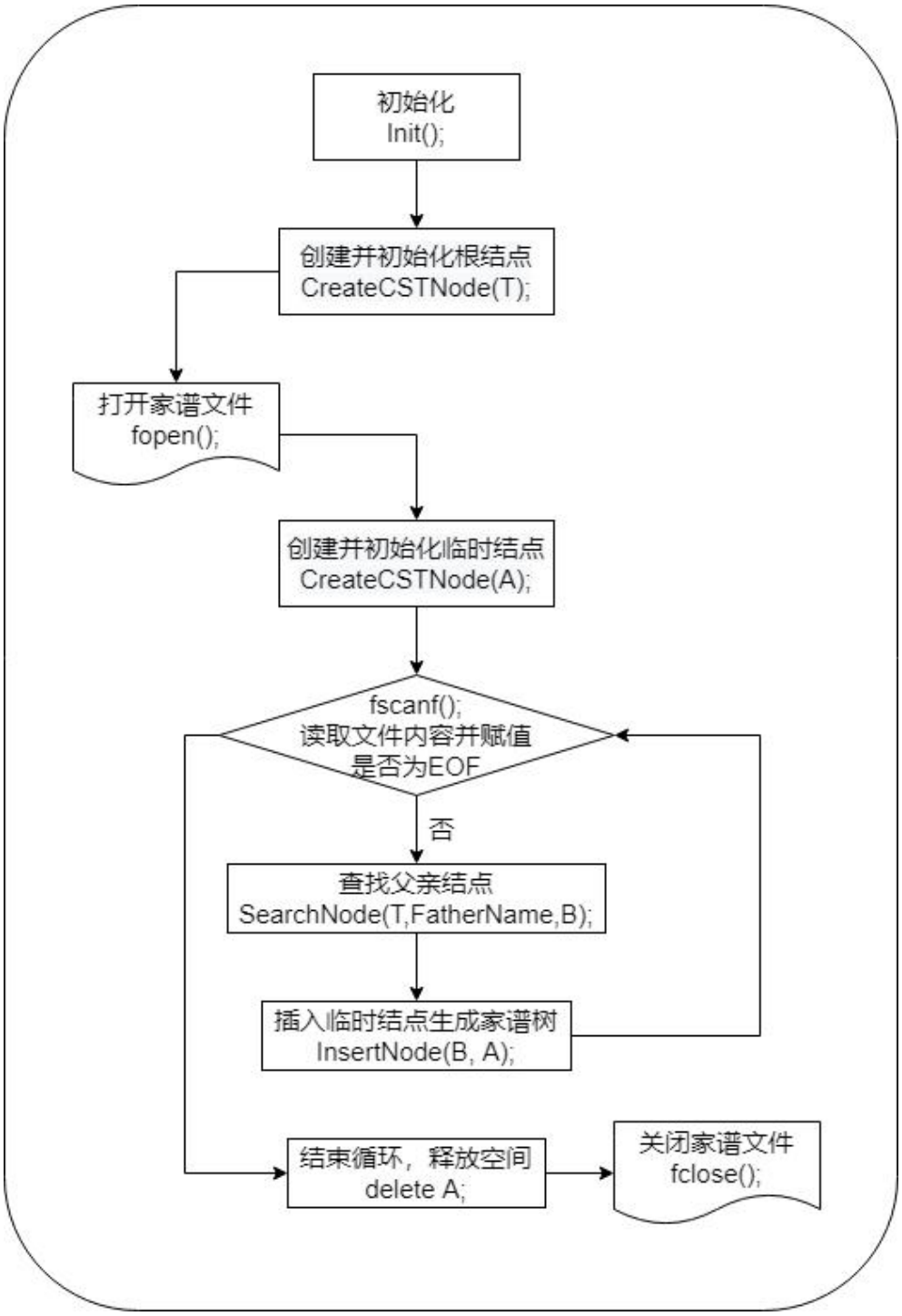


2.2 功能详细说明

（使用流程图等方式说明每一个功能的过程）

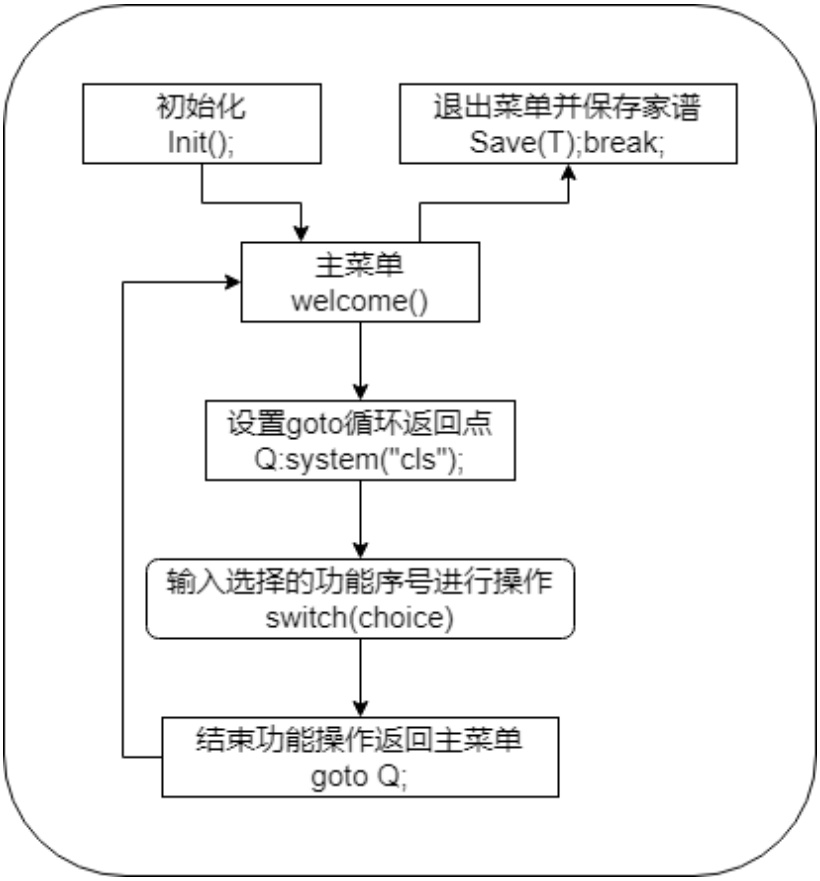
2.2.1 初始化 Init();

家谱信息预先设置好，成员的信息包括：姓名（假设姓名不重复）、性别（男为 1、女为 0）、伴侣姓名、出生年月日、目前状况（健在为 1、身故为 0）、父亲姓名。打开程序首先初始化，读取文件内容，从文件中读入家谱信息，经过依次判断信息内容无误，在程序里自动生成一个孩子兄弟树。



2.2.2 主菜单 welcom();

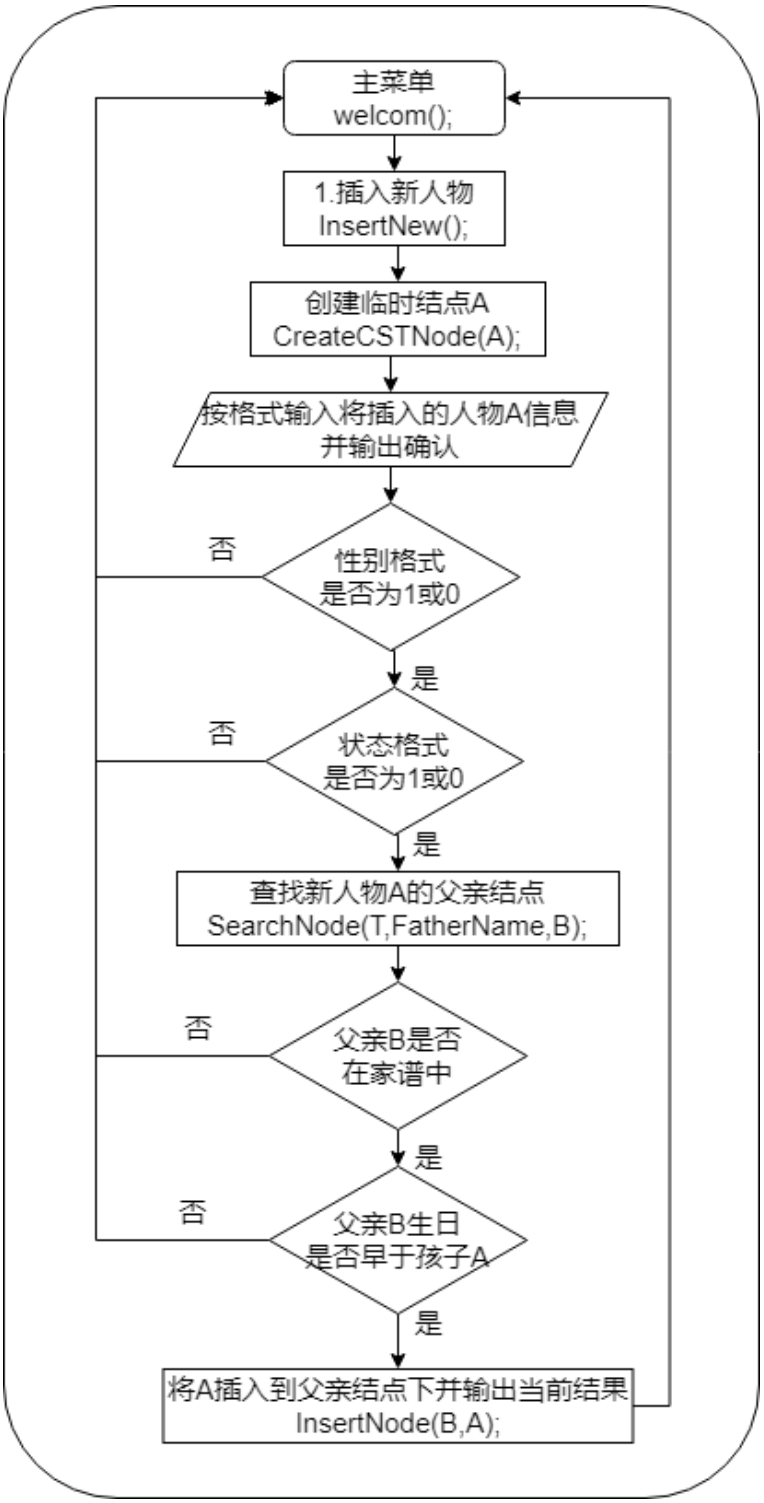
在主菜单中，可以选择需要的操作功能模块，每个模块执行完后又回到菜单界面，直到用户选择退出菜单。如 2.1 系统功能中的结构图所示。



2.2.3 插入新人物 InsertNew();

功能 1 插入，即给某个人添加一个孩子：

选择该功能后，系统会输出一个插入新人物时的输入例子，按照格式输入即将插入新人物的具体信息包括父亲姓名，输入后系统将输出当前输入的内容并判断信息格式是否规范合理，不规范则结束当前操作返回主菜单，规范则将新人物插入其父亲结点下并输出当前结果，结束后返回主菜单。

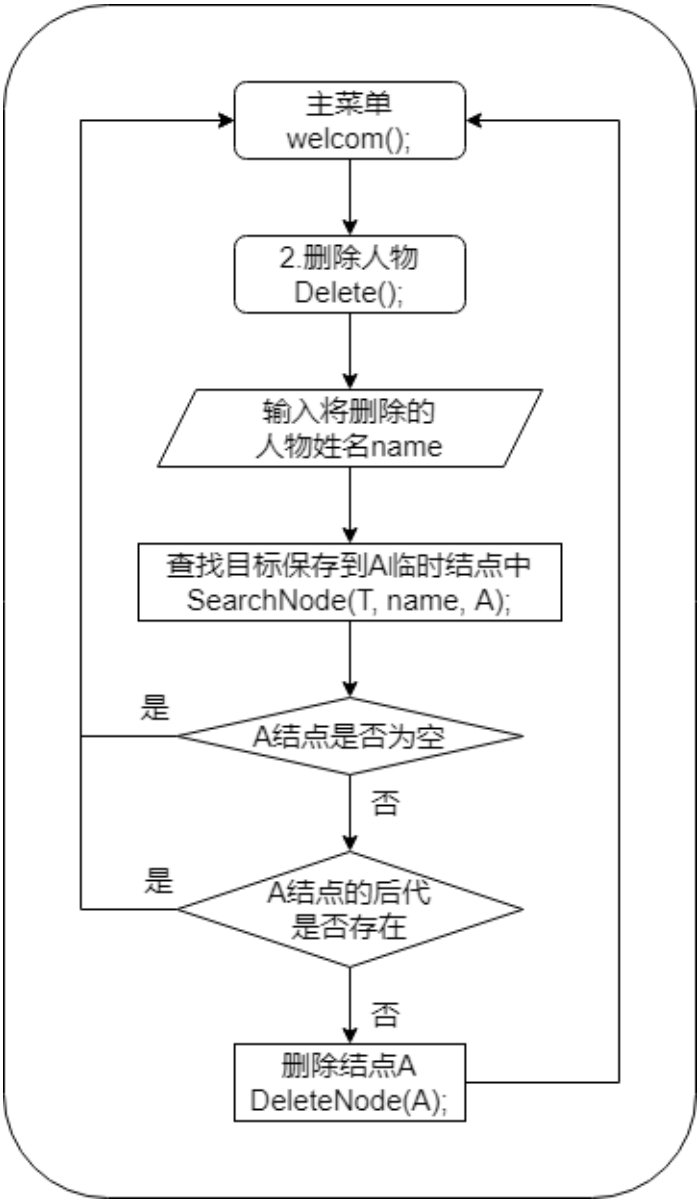


2.2.4 删除人物 Delete();

功能 2 删除：

选择该功能后，需要输入将要删除的人物姓名，系统将判断该人物是否存在于家谱中，若不存在则返回主菜单，若存在则判断该人物是否有后代，如果该人

物有后代，则不允许删除，若无后代，则删除该人物。

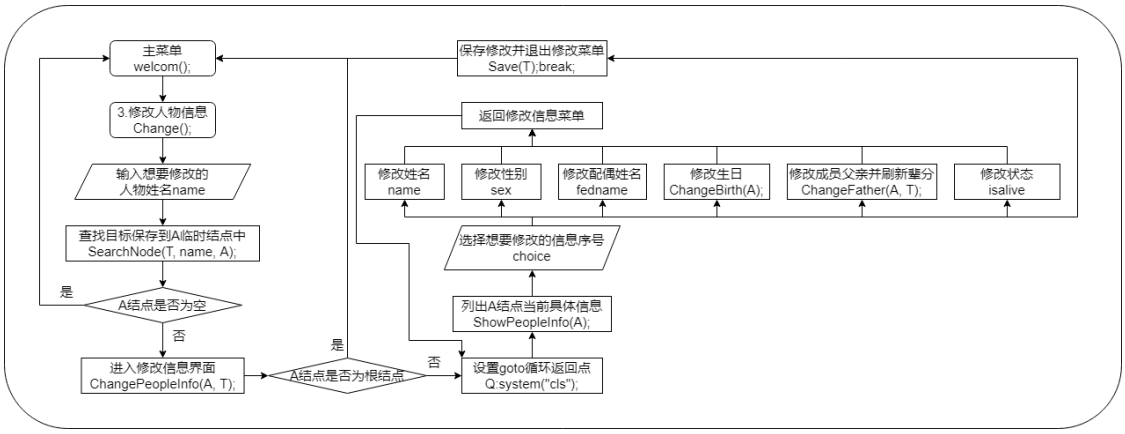


2.2.5 修改人物信息 Change();

功能 3 修改：

选择该功能后，需要输入将要修改信息的人物姓名，系统将判断该人物是否存在于家谱中，若不存在则返回主菜单，若存在则进入修改人物信息界面。此后系统将判断该人物结点是否为森林根结点，若是则返回主菜单，若不是则进入修改人物信息菜单。系统将列出该人物当前的具体信息，此后选择想要修改的某项信息的序号，按照系统指令进行修改，修改后将返回修改人物信息菜单并列出该人物当前具体信息，即修改后的信息。在菜单中选择 break 的序号则将保存修改

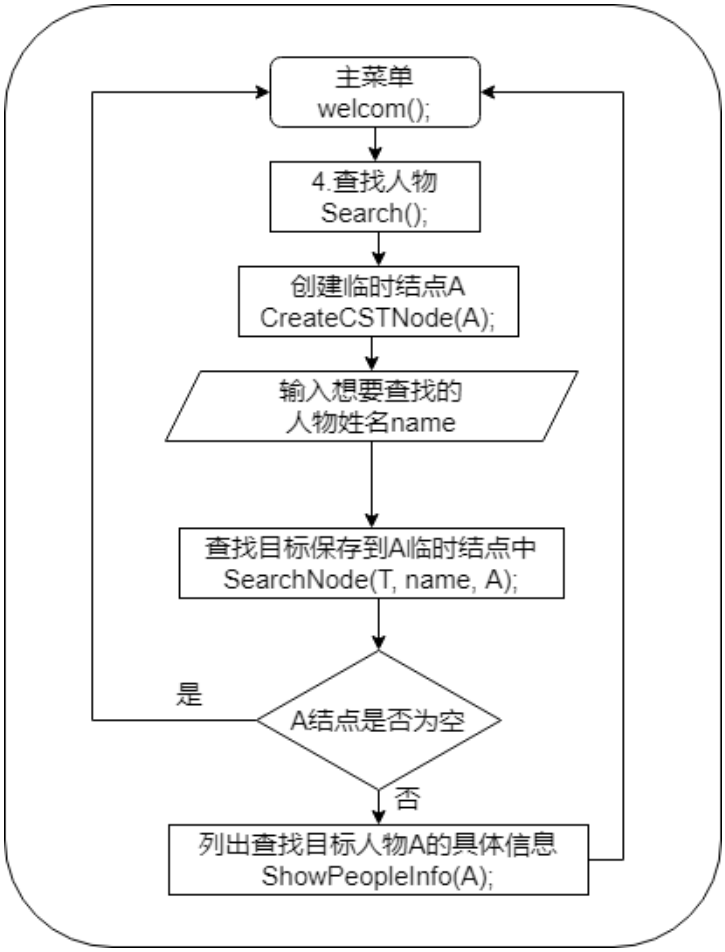
后的家谱到文件中并退出修改菜单。



2.2.6 查找人物 Search();

功能 4 查找：

选择该功能后，需要输入将要修改信息的人物姓名，系统将判断该人物是否存在于家谱中，若不存在则返回主菜单，若存在则列出该人物的具体信息，即姓名、性别、配偶姓名、辈分、世数、生日年月日、目前状况、父亲姓名。

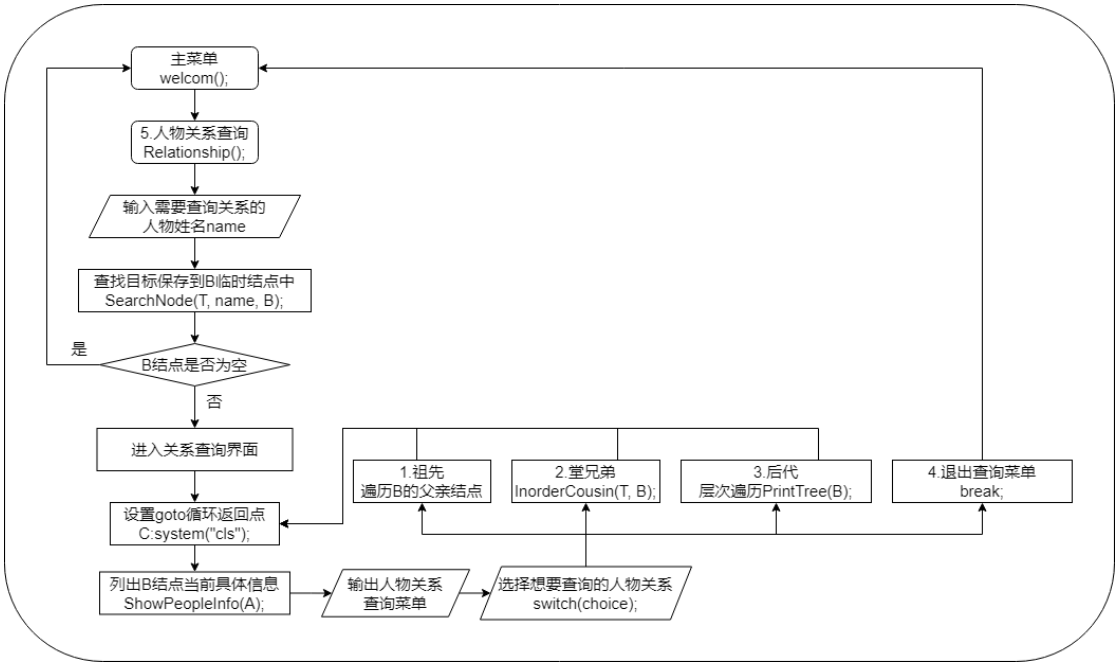


2.2.7 人物关系查询 Relationship();

功能 5 通过人物关系查询：

选择该功能后，需要输入需要查询关系人物的人物姓名，系统将判断该人物是否存在于家谱中，若不存在则返回主菜单，若存在保存该人物在家谱树中的结点，进入关系查询界面。进入界面后系统将列出该人物的具体信息，即姓名、性别、配偶姓名、辈分、世数、生日年月日、目前状况、父亲姓名，随后输出人物关系查询菜单。

在人物关系查询菜单中，可选择的查询关系有：其祖先、其堂兄弟、及其后代。查询其祖先：系统可以遍历该人物结点的父亲结点，并输出该人的父亲及其祖宗的信息。查询其堂兄弟：系统采用一个递归函数从根结点开始遍历，输出该人该世代所有人（即其堂兄弟）的信息。查询其后代：系统将采用层次遍历的方法，输出该人的孩子及其后代的信息。若在菜单中选择 break 的序号则将退出人物关系查询菜单，返回到主菜单。

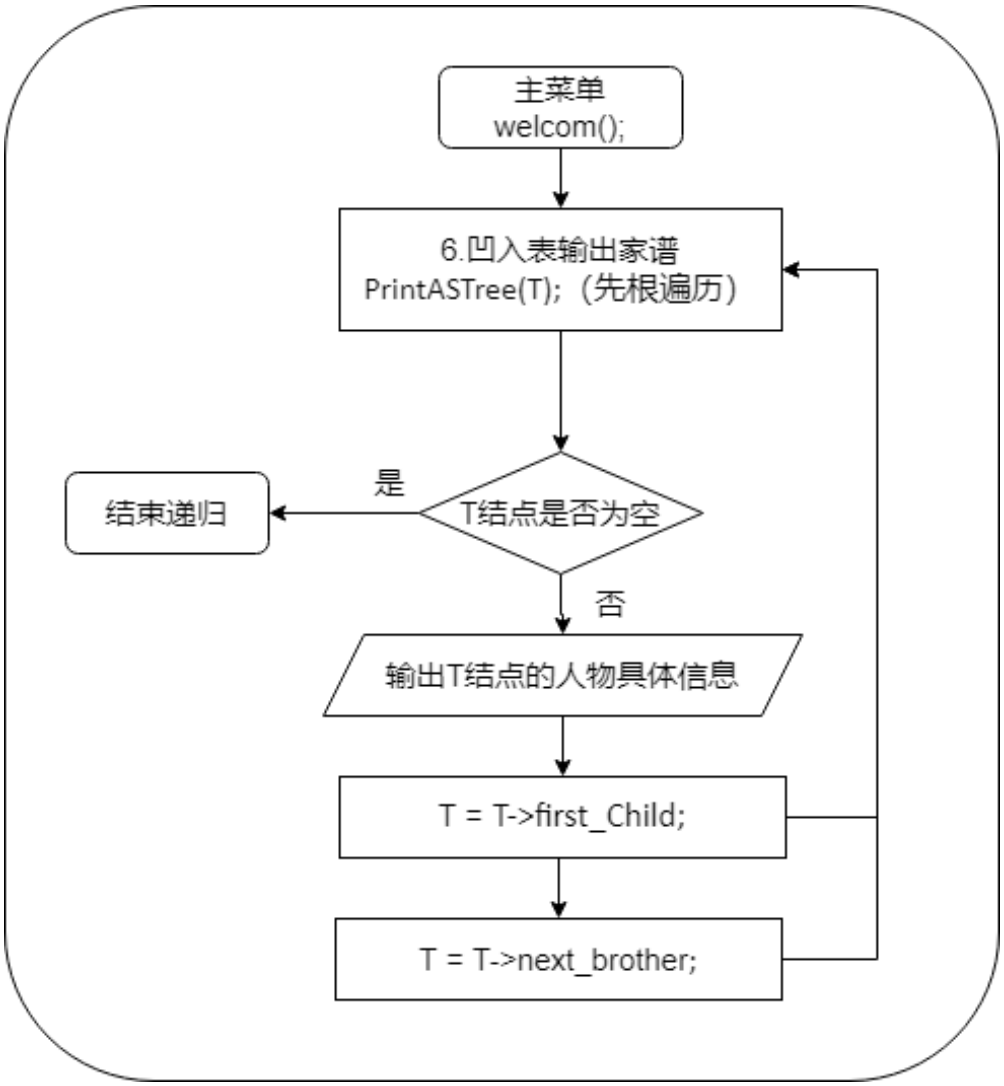


2.2.8 凹入表方式打印树状家谱 PrintASTree(T);

功能 6 显示：

选择该功能，系统将递归地从根结点开始采用先根遍历方法（即二叉树存储

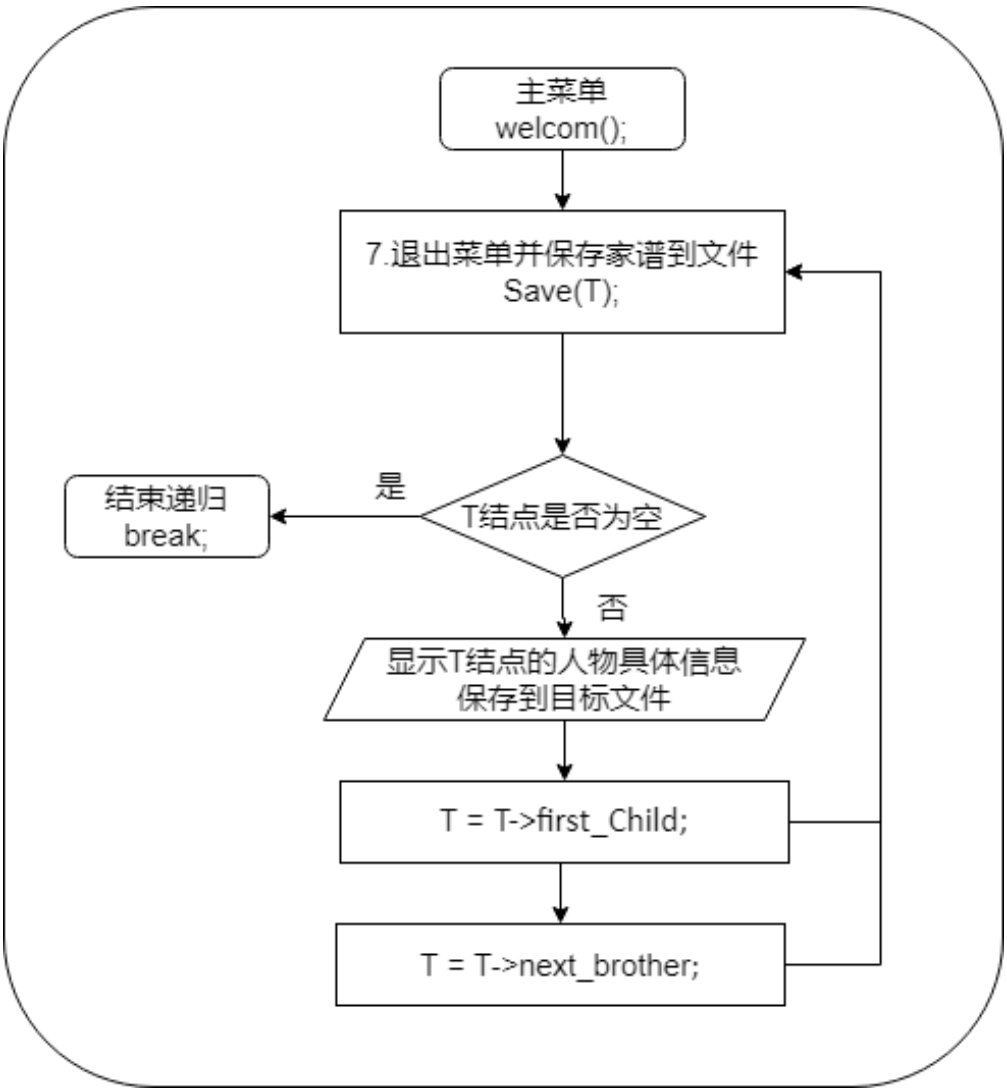
的先序遍历），以凹入表的方式打印显示树状家谱。



2.2.9 退出菜单并保存家谱 Save(T);

功能 7 退出保存：

选择该功能，系统将递归地从根结点开始采用先根遍历方法（即二叉树存储的先序遍历），以凹入表的方式将树状家谱保存到目标文件中，并退出系统。



三、 数据与结构说明

3.1 数据结构

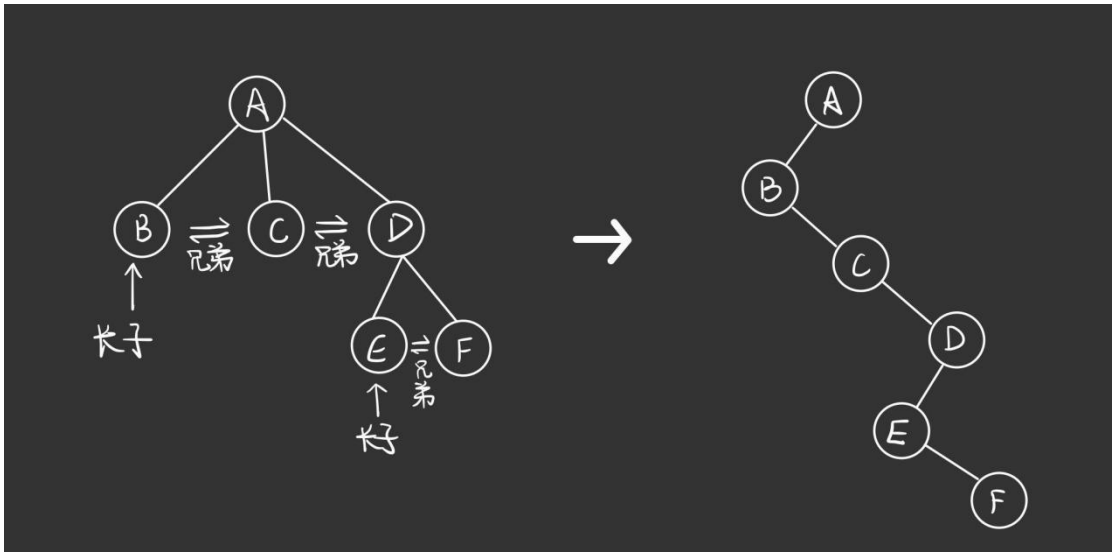
（说明系统中的定义的主要数据结构及其作用）

通过分析系统功能要求，考虑使用树作为整个家谱管理系统的数据结构，同时由于每个父亲的孩子数量不确定，因此采用链表作为数据结构保存每个父结点的第一个孩子的指针、下一个兄弟指针以及父亲指针，即使用孩子兄弟二叉树多重链表存储结构。同时因为家谱管理系统的查询人物关系功能模块需要层次遍历家谱，因此还需使用到队列链表。

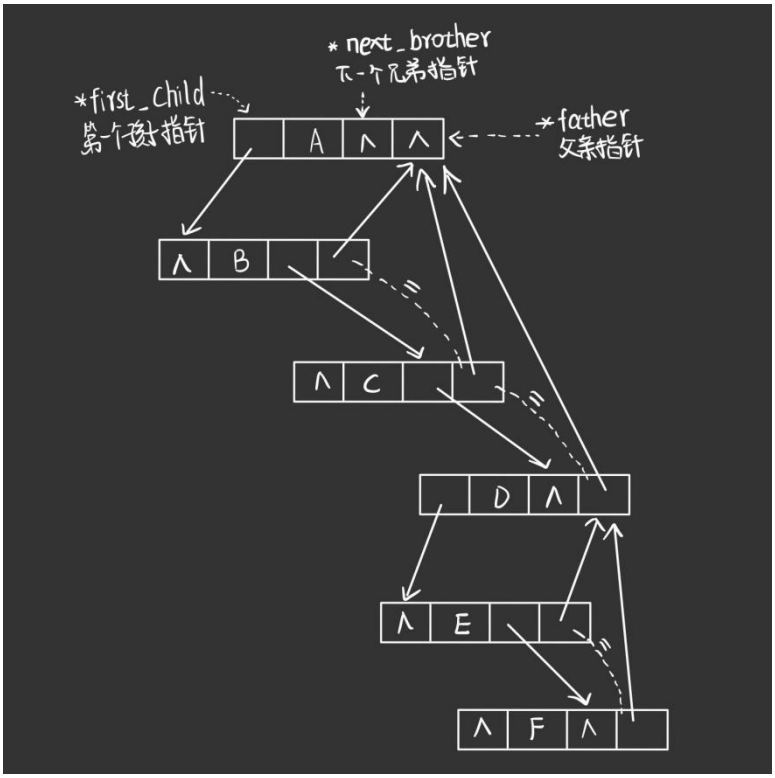
3.2 初始化逻辑结构

（画图说明初始化数据构成的逻辑结构）

家族谱与数据结构中树的概念相结合，每一个结点就是族谱中的个人，于是就需要知道每个人最基本的特性，于是就可以抽象化变成树中的属性。家谱包含许多家庭，可以看作一片森林，而由孩子兄弟表示法可知森林有唯一对应的二叉树。



由于每个父亲的孩子数量不确定，因此采用链表作为数据结构保存每个父结点的第一个孩子的指针、下一个兄弟指针以及父亲指针，即使用孩子兄弟二叉树多重链表存储结构。



初始化时，选用全局变量 T 作为整个森林的根结点，在保留根结点的基础上再读取文件，同时使用全局变量 A、B 作为临时储存结点逐渐创建出家谱树。

3.3 模块函数

（根据情况，说明程序中的类、模块、函数的作用）

3.3.1 数据类型（结构体）定义

①结构体-日期-年月日（Data）

储存成员信息的生日日期，包含年（int）、月（int）、日（int）。

```
typedef struct DATA
{
    int year;
    int month;
    int day;
}DATA;
```

②结构体-成员信息数据域（PeopleInfo）

作为成员树结点的信息数据域，包含辈分数（int）、性别（int）、目前状况（int）、年龄（int）、姓名（char）、配偶姓名（char）、出生日期（DATA）。

```
typedef struct PeopleInfo
{
    int seniority;  //辈分数 第几世
    int sex;  //性别 1 为男性,0 为女性
    int isalive;  //目前状况 1 为健在,0 为身故
    int age;  //年龄
    char name[100];  //姓名
    char fedname[100];  //配偶姓名
    DATA birth_data;  //结构 Data 定义的出生日期
}PeopleInfo;
```

③结构体-孩子兄弟树结点（CSTNode, * CSTree, * CSForest）

作为成员树结点，包含该结点的数据域（PeopleInfo）、第一个孩子指针（CSTNode *）、下一个兄弟指针（CSTNode *）、父亲指针（CSTNode *）。

```
typedef struct CSTNode
{
    PeopleInfo people_data;  //该结点的数据域
    struct CSTNode* first_Child, * next_brother, * father;
    //第一个孩子指针，下一个兄弟指针，父亲指针
}CSTNode, * CSTree, * CSForest;  //树结点，树指针，森林指针
```

④结构体-队列链表结点（LNode, * LinkList）

作为成员队列链表的结点，包含队列链表的结点数据域（CSTree）、指向下一个队列链表结点的指针域（LNode *）。

```
typedef struct LNode
{
    CSTree tree_data;  //链表结点数据域
    struct LNode* next;  //指向下一个的指针域
   }LNode, * LinkList;  //链表结点，链表指针
```

⑤结构体-枚举类型返回值（Status）

作为枚举类型结构体，包含返回值 TRUE、FALSE、OK、ERROR、SUCCESS、EMPTY。

```
typedef enum status
{
    TRUE,  //1
    FALSE,  //0
    OK,  //1
    ERROR,  //0
    SUCCESS,  //0
    //OVERFLOW,  //-2
    EMPTY  //空或 0 返回 1，非空返回 0
   }Status;
```

3.3.2 基本树操作模块

①初始化结点数据函数，避免出现野指针（void InitCSTNode(CSTree& A);）

```
//初始化结点函数,避免出现野指针，参数为树结点指针
void InitCSTNode(CSTree& A)
{
    if (A == NULL)
        return;
    A->father = NULL;
    A->first_Child = NULL;
    A->next_brother = NULL;
    A->people_data.seniority = 0;
    A->people_data.sex = 0;
    A->people_data.isalive = 0;
    A->people_data.age = 0;
    A->people_data.birth_data.year = 0;
    A->people_data.birth_data.month = 0;
    A->people_data.birth_data.day = 0;
```

```
        strcpy(A->people_data.name, "无");
        strcpy(A->people_data.fedname, "无");
        return;
    }
```

②创建结点函数（void CreateCSTNode(CSTree& A);）

```
//创建结点函数，参数为树结点指针
void CreateCSTNode(CSTree& A)
{
    A = new CSTNode;  //分配内存空间
    //(*A)=(CSTree)malloc(sizeof(CSTNode));
    if (A == NULL)
        return;
    InitCSTNode(A);  //初始化结点
}
```

③销毁树函数（void Destroy(CSTree A);）

```
//销毁树函数(同二叉树一样，递归)，参数为树结点指针
void Destroy(CSTree A)
{
    if (A == NULL)
        return;
    Destroy(A->first_Child);
    Destroy(A->next_brother);
    delete A;
}
```

④删除结点函数（void DeleteNode(CSTree A);）

```
//删除结点函数，参数为树结点指针
void DeleteNode(CSTree A)
{
    CSTree prebro;

    //如果 A 是父亲结点的第一个孩子，则让 A 的兄弟成为父亲结点的第一个孩子
    if (A->father->first_Child == A)
    {
        A->father->first_Child = A->next_brother;
    }
    else  //A 不是父亲结点的第一个孩子，让 A 的前一个兄弟指向 A 的下一个兄弟
    {
        //把 A 的前一个兄弟赋值给 prebro
        for (prebro = A->father->first_Child; prebro->next_brother != A;)
        {
            prebro = prebro->next_brother;
```

```
            }
            prebro->next_brother = A->next_brother;
        }
        A->next_brother = NULL;  //释放 A 指向下一个兄弟的指针域
        A->father = NULL;  //释放 A 指向父亲的指针域
        system("pause");
        Destroy(A);  //将 A 的后代全部移除，释放 A 的空间
    }
```

⑤插入结点函数（void InsertNode(CSTree &father, CSTree child);）

```
//插入结点函数，参数为父亲结点指针和孩子结点指针
void InsertNode(CSTree& father, CSTree child)
{
    child->people_data.seniority = father->people_data.seniority + 1;  //孩子的代数比父亲大一
    child->father = father;  //配置插入孩子的父亲指针
    int birth = (child->people_data.birth_data.year) * 10000 + (child->people_data.birth_data.month) * 100 + (child->people_data.birth_data.day);
    CSTree bro1 = father->first_Child;

    //如果 father 没有 first_Child，直接插入 child
    if (bro1 == NULL)
    {
        father->first_Child = child;
        return;
    }

    int bro1birth = (bro1->people_data.birth_data.year) * 10000 + (bro1->people_data.birth_data.month) * 100 + (bro1->people_data.birth_data.day);
    //如果插入孩子的生日早于父亲的第一个孩子，那么插入孩子成为新 first_Child
    if (bro1birth >= birth)
    {
        child->next_brother = bro1;
        father->first_Child = child;
        return;
    }

    //如果插入孩子生日晚于父亲第一个孩子，那么遍历兄弟链表找到插入位置
    CSTree bro2 = father->first_Child;
    int bro2birth = (bro2->people_data.birth_data.year) * 10000 + (bro2->people_data.birth_data.month) * 100 + (bro2->people_data.birth_data.day);
    bro1 = bro2->next_brother;
    //找到 bro2 生日早于 child 生日早于 bro1 生日的位置
    for (; (bro1 != NULL) && (birth > bro1birth);)
```

```
    {
        bro2 = bro2->next_brother;
        bro1 = bro2->next_brother;
    }
    child->next_brother = bro1;
    bro2->next_brother = child;

    return;
}
```

⑥查找结点函数（void SearchNode(CSForest T, char name[100], CSTree &B);）

```
//先序遍历查找函数，T 为树的根结点，name 为需要找的人，B 储存找到的结点位置
void SearchNode(CSForest T, char name[100], CSTree& B)
{
    if (T == NULL)
        return;
    if (strcmp(T->people_data.name, name) == 0)  //如果 T 的 name 与查找目标相同
    {
        B = T;
        return;
    }
    SearchNode(T->first_Child, name, B);
    if (B != NULL)
        return;
    SearchNode(T->next_brother, name, B);
}
```

⑦展示成员信息数据域函数（void ShowPeopleInfo(CSTree A);）

```
//展示成员信息数据域函数
void ShowPeopleInfo(CSTree A)
{
    if (A == NULL)
        return;
    if (A->people_data.seniority == 0)  //A 的辈分为 0，是根结点
    {
        cout << "该结点为森林根结点，禁止操作！\n";
        system("pause");
        return;
    }
    cout << "姓名：" << A->people_data.name << "  ";
    if (A)
        cout << "性别：男  ";
    else
        cout << "性别：女  ";
```

```
        cout << "配偶姓名：" << A->people_data.fedname << "  ";
        cout << "辈分：" << A->people_data.seniority << "  ";
        cout << "第" << (A->people_data.seniority) + 12 << "世  ";
        cout << "生日：" << A->people_data.birth_data.year << "." <<
A->people_data.birth_data.month << "." << A->people_data.birth_data.day << "  ";
        if (A->people_data.isalive == 1)
            cout << "目前状况：健在  ";
        else
            cout << "目前状况：身故  ";

        if (A->people_data.seniority != 1)
            cout << "父亲姓名：" << A->father->people_data.name << "  ";
        else
            cout << "为家族祖先" << endl;

        cout << endl;

    }
```

⑧修改成员信息数据域函数（void ChangePeopleInfo(CSTree &B,CSForest T);）

```
//修改成员信息数据域函数，B 为欲修改的结点，T 为根结点
void ChangePeopleInfo(CSTree& B, CSForest T)
{
    if (B == NULL)
        return;
    if (B->people_data.seniority == 0)  //B 的辈分为 0，是根结点
    {
        cout << "该结点为森林根结点，禁止操作！\n";
        system("pause");
        return;
    }
    int choice;
W:system("cls");
    ShowPeopleInfo(B);  //列出 B 结点当前具体信息
    cout << endl;
    cout << "请选择想要修改的信息序号：" << endl << endl;
    cout << "************************" << endl << endl;
    cout << "    1.name" << endl;
    cout << "    2.sex" << endl;
    cout << "    3.fedname" << endl;
    cout << "    4.birth" << endl;
    cout << "    5.father" << endl;
    cout << "    6.isalive" << endl;
    cout << "    7.break" << endl << endl;
```

```
cout << "************************" << endl << endl;
cout << "请输入序号：";
//fflush(stdin); //cin.clear;
cin >> choice;

switch (choice)
{
case 1:  //修改 name
     cout << "请输入新的姓名：";
     char name[100];
     cin >> name;
     strcpy(B->people_data.name, name);
     system("pause");
     goto W;
case 2:  //修改 sex
     cout << "你的性别为：（男性请输入 1，女性请输入 2）";
     int sex;
     cin >> sex;
     if ((sex != 2) && (sex != 1))
     {
          cout << "输入有误" << endl;
          system("pause");
          goto W;
     }
     B->people_data.sex = sex;
     system("pause");
     goto W;
case 3:  //修改 fedname
     cout << "请输入其配偶的新姓名：";
     char fedname[100];
     cin >> fedname;
     strcpy(B->people_data.fedname, fedname);
     system("pause");
     goto W;
case 4:  //修改 birth
     ChangeBirth(B);  //修改成员生日函数
     system("pause");
     goto W;
case 5:  //修改 father
     ChangeFather(B, T);  //修改成员父亲函数
     system("pause");
     goto W;
case 6:  //修改 isalive
     if (B->people_data.isalive)
```

```
            B->people_data.isalive = 0;
        else
            B->people_data.isalive = 1;
        system("pause");
        goto W;
    case 7:  //结束修改成员信息模块
        Save(T);
        break;
    default:
        goto W;
    }

    return;

}
```

⑨修改成员生日函数（void ChangeBirth(CSTree &A);）

```
//修改成员生日函数
void ChangeBirth(CSTree& A)
{
    cout << "生日修改为：（输入格式：代表年份的四位整数 代表月份的1-12整数 代表日期的0-31整数）";
    int year, month, day;
    cin >> year >> month >> day;
    if ((year < 1) || (year > 10000) || (month < 1) || (month > 12) || (day < 1) || (day > 31))
    {
        cout << "输入有误" << endl;
        system("pause");
        return;
    }
    int Abirth = year * 10000 + month * 100 + day;
    int fatherbirth = (A->father->people_data.birth_data.year) * 10000 + (A->father->people_data.birth_data.month) * 100 + (A->father->people_data.birth_data.day);
    //如果A的父亲存在且A的父亲的生日晚于A
    if ((A->father->people_data.seniority != 0) && (fatherbirth >= Abirth))
    {
        cout << "生日不能比父亲早" << endl;
        return;
    }
    A->people_data.birth_data.year = year;
    A->people_data.birth_data.month = month;
    A->people_data.birth_data.day = day;
```

```
        CSTree B;
        CSTree C;
        C = A->father;
        //修改生日后需要重新排序兄弟结点，可先把该结点分离出来
        if (A->father->first_Child == A)
        {
            A->father->first_Child = A->next_brother;
            A->next_brother = NULL;
            A->father = NULL;
        }
        else
        {
            for (B = A->father->first_Child; B->next_brother != A; B = B->next_brother);
            B->next_brother = A->next_brother;
            A->next_brother = NULL;
            A->father = NULL;
        }
        //再调用含自动排序功能的 InsertNode 函数插入回二叉树内
        InsertNode(C, A);
    }
```

⑩修改成员父亲函数（void ChangeFather(CSTree &A, CSForest T);）

```
    //修改成员父亲函数
    void ChangeFather(CSTree& A, CSForest T)
    {
        cout << "父亲修改为：" << endl;
        int Abirth = (A->people_data.birth_data.year) * 10000 +
(A->people_data.birth_data.month) * 100 + (A->people_data.birth_data.day);
        char fathername[100];
        cin >> fathername;

        //调用 SearchNode 函数找到新的父亲结点
        CSTree newfather = NULL;
        SearchNode(T, fathername, newfather);
        ShowPeopleInfo(newfather);  //显示新父亲结点的详细信息

        //查看新父亲结点是否存在
        if (newfather == NULL)
        {
            cout << "该成员不存在" << endl;
            system("pause");
            return;
        }
```

```
        system("pause");

        //查看新父亲结点生日是否早于孩子
        int fatherbirth = (newfather->people_data.birth_data.year) * 10000 +
(newfather->people_data.birth_data.month) * 100 +
(newfather->people_data.birth_data.day);
        if ((newfather->people_data.seniority != 0) && (fatherbirth >= Abirth))
        {
            cout << "生日不能早于父亲" << endl;
            return;
        }

        //将欲修改的 A 结点分离出来插入到新父亲结点下
        CSTree B;
        CSTree C;
        C = A->father;
        if (A->father->first_Child == A)
        {
            A->father->first_Child = A->next_brother;
            A->next_brother = NULL;
            A->father = NULL;
        }
        else
        {
            B = A->father->first_Child;
            for (; B->next_brother != A; B = B->next_brother);
            B->next_brother = A->next_brother;
            A->next_brother = NULL;
            A->father = NULL;
        }

        InsertNode(newfather, A);  //将 A 结点插入新父亲结点下
        RefreshSeniority(T);  //刷新辈分
        }
```

⑪辈分刷新函数（void RefreshSeniority(CSForest &T);）

```
    //辈分刷新函数，利用先序遍历思想递归
    void RefreshSeniority(CSForest& T)
    {
        if (T == NULL)
            return;
        if (T->father == NULL)
            T->people_data.seniority = 0;
        else
```

```
        {
            T->people_data.seniority = T->father->people_data.seniority + 1;
        }
        RefreshSeniority(T->first_Child);
        RefreshSeniority(T->next_brother);
        return;
    }
```

⑫输出所有堂兄堂弟函数（void InorderCousin(CSTree T, CSTree A);）

及输出同世代的兄弟函数（void InorderBro(CSTree T, CSTree A);）

```
//输出所有堂兄堂弟
void InorderCousin(CSTree T, CSTree A)
{
    if (T == NULL)
        return;
    //从根结点开始遍历，如果找到与 A 结点父亲的父亲相同，即与 A 互为堂兄弟，输出
    if (T->people_data.seniority >= 3)
        if ((T->father->father == A->father->father) && (T != A))
            cout << "[" << T->people_data.seniority << "][第" <<
T->people_data.seniority + 12 << "世][" << T->people_data.birth_data.year << "." <<
T->people_data.birth_data.month << "." << T->people_data.birth_data.day << "]" <<
T->people_data.name << endl;
    InorderCousin(T->first_Child, A);
    InorderCousin(T->next_brother, A);
}

//输出所有同世代的兄弟
void InorderBro(CSTree T, CSTree A)
{
    if (T == NULL)
        return;
    //从根结点开始遍历，找到与 A 结点辈数相同的兄弟输出
    if (T->people_data.seniority >= 2)
        if ((T->people_data.seniority == A->people_data.seniority) && (T != A))
            cout << "[" << T->people_data.seniority << "][第" <<
T->people_data.seniority + 12 << "世][" << T->people_data.birth_data.year << "." <<
T->people_data.birth_data.month << "." << T->people_data.birth_data.day << "]" <<
T->people_data.name << endl;
    InorderBro(T->first_Child, A);
    InorderBro(T->next_brother, A);
}
```

⑬凹入表输出家谱函数（void PrintASTree(CSTree T);）

```
//凹入表输出家谱，先根遍历（即二叉树存储的先序遍历）
```

```
void PrintASTree(CSTree T)
{
    int cnt;
    if (T)
    {
        //输出空格
        for (cnt = 1; cnt < T->people_data.seniority; cnt++)
            cout << "    ";
        //输出字符
        cout <<"["<< T->people_data.seniority <<"][第"<< T->people_data.seniority
+ 12 <<"世]["<< T->people_data.birth_data.year <<"."<< T->people_data.birth_data.month
<<"."<< T->people_data.birth_data.day <<"]"<< T->people_data.name << endl;
        cout << endl;
        PrintASTree(T->first_Child);
        PrintASTree(T->next_brother);
    }
}
```

⑭层次遍历输出函数（Status PrintTree(CSTree t);）

```
//层次遍历输出函数
Status PrintTree(CSTree t)
{
    LinkList L;
    if (t == NULL)
    {
        cout << "森林不存在" << endl;
        return OK;
    }
    InitQueue(L);  //初始化队列
    Traverse(t->first_Child, L);  //利用队列输出
    DestroyQueue(L);  //销毁队列
    return OK;
}
```

⑮队列相关函数

(1)初始化队列（Status InitQueue(LinkList& L);）

```
//初始化队列
Status InitQueue(LinkList& L)
{
    L = new LNode;  //分配结点空间
    if (L == NULL)  //分配失败
    {
        return ERROR;
    }
```

```
    L->next = NULL;
    return OK;
}
```

(2)新建结点（LNode* CreateLNode(CSTNode* p);）

```
//新建结点
LNode* CreateLNode(CSTNode* p)
{
    LNode* q;
    q = new LNode;  //分配结点空间
    if (q != NULL)  //分配成功
    {
        q->tree_data = p;
        q->next = NULL;
    }
    return q;
}
```

(3)元素 q 入队列（Status Enqueue(LNode* p, CSTNode* q);）

```
//元素 q 入队列
Status Enqueue(LNode* p, CSTNode* q)
{
    if (p == NULL)
        return ERROR;
    while (p->next != NULL)  //调至队列最后
        p = p->next;
    p->next = CreateLNode(q);  //生成 q 的队列结点让 q 进入队列
    return OK;
}
```

(4)出队列，并保存 q 返回值（Status Dequeue(LNode* p, CSTNode* q); ）

```
//出队列，并保存 q 返回值
Status Dequeue(LNode* p, CSTNode* q)
{
    LNode* aq;
    if (p == NULL || p->next == NULL)  //出队列的位置不合理
        return ERROR;
    aq = p->next;  //修改被出队结点 aq 的指针域
    p->next = aq->next;  //让 p 的下一个指针域指向 p->next->next
    q = aq->tree_data;  //用 q 保存出队 aq 的数据域
    delete aq;  //释放结点 aq
    return OK;
}
```

(5)队列判空（Status IfEmpty(LinkList L);）

```
//队列判空
Status IfEmpty(LinkList L)
{
    if (L == NULL)  //队列不存在
        return ERROR;
    if (L->next == NULL)  //队列为空
        return TRUE;
    return FALSE;  //队列非空
}
```

(6)销毁队列（void DestroyQueue(LinkList L);）

```
//销毁队列
void DestroyQueue(LinkList L)
{
    LinkList p;
    if (L != NULL)
    {
        p = L;
        L = L->next;
        delete p;  //逐一释放
        DestroyQueue(L);  //递归销毁
    }
}
```

(7)用队列遍历输出 CSTree（Status Traverse(CSTree T, LinkList L);）

```
//用队列遍历输出 CSTree
Status Traverse(CSTree T, LinkList L)
{
    CSTree P = T;
    CSTree K;
    int i = 0;
    int j = 1;
    while (P)  //p 指向森林里的每棵树
    {
        cout << "第" << j << "个家庭" << endl;
        K = P;  //利用 K 来遍历以 P 为根结点的子树中的结点
        Enqueue(L, K);  //根结点入队
        while (IfEmpty(L) == FALSE)  //只要队列不为空就依次出队直到队空
        {
            Dequeue(L, K);
            if (i != K->people_data.seniority)  //将 K 的辈数赋给 i
            {
                cout << endl;
```

```
                    i = K->people_data.seniority;
                }
                cout << "[" << K->people_data.seniority << "][第" <<
K->people_data.seniority + 12 << "世][" << K->people_data.birth_data.year << "." <<
K->people_data.birth_data.month << "." << K->people_data.birth_data.day << "]" <<
K->people_data.name << endl;

                if (K->first_Child)  //如果该结点不是森林中的叶节点则进入下一层即孩子
层
                {
                    K = K->first_Child;
                    Enqueue(L, K);
                    while (K->next_brother)  //入队同一层的兄弟结点
                    {
                        K = K->next_brother;
                        Enqueue(L, K);
                    }
                }

            }
            P = P->next_brother;  //递归
            cout << endl;
            j++;
        }
        return OK;
    }
```

3.3.3 主程序代码实现模块

①初始化函数，从文件中读取信息自动生成孩子兄弟二叉树（void Init();）

```
//初始化函数，从文件中读取信息自动生成孩子兄弟二叉树
void Init()
{
    CreateCSTNode(T);  //创建根结点并初始化，此时 T 的 name 为“无”

    FILE* r;
    r = fopen("C:\\Users\\youye\\Desktop\\Family.txt", "r");  //打开文件夹
    if (r == NULL)
    {
        cout << "打开文件失败！" << endl;
    }

    A = NULL;
```

```
        B = NULL;
        char FatherName[100], tmp1[2], tmp2[2];

        CreateCSTNode(A);  //创建 A 临时结点

        cout << "读取文件中..." << endl;

        //读取文件直到循环结束
        while (fscanf(r, "%s %d %s %d%s%d%s%d %d %s", &A->people_data.name,
&A->people_data.sex, &A->people_data.fedname, &A->people_data.birth_data.year, tmp1,
&A->people_data.birth_data.month, tmp2, &A->people_data.birth_data.day,
&A->people_data.isalive, FatherName) != EOF)
        {
            cout << A->people_data.name << " " << A->people_data.sex << "    " <<
A->people_data.fedname << " " << A->people_data.birth_data.year << tmp1 <<
A->people_data.birth_data.month << tmp2 << A->people_data.birth_data.day << "  " <<
A->people_data.isalive << " " << FatherName;
            cout << endl << "Search..." << endl;
            SearchNode(T, FatherName, B);  //查找 FatherName 的结点储存到 B 临时父亲结点
中
            //如果父亲结点存在就将 A 孩子结点插入其下，如果父亲结点为无就作为根结点的子
节点存在

            if (B == NULL)
            {
                cout << "此人的父亲不在家谱里！" << endl;
                delete A;
            }
            int Abirth = (A->people_data.birth_data.year) * 10000 +
(A->people_data.birth_data.month) * 100 + (A->people_data.birth_data.day);
            int Bbirth = (B->people_data.birth_data.year) * 10000 +
(B->people_data.birth_data.month) * 100 + (B->people_data.birth_data.day);
            //如果 B 作为临时父亲结点存在且不是根结点且生日晚于 A
            if ((B->people_data.seniority != 0) && (Bbirth >= Abirth))
            {
                cout << "孩子生日不能比父亲早" << endl;
                delete A;
            }
            else
            {
                cout << "Insert..." << endl;
                InsertNode(B, A);  //存在父亲结点，将 A 插入父亲结点下
            }
```

```
            B = NULL;
            A = NULL;
            CreateCSTNode(A);  //继续循环
        }
        delete A;
        A = NULL;
        fclose(r);
        }
```

②主菜单函数（void welcome();）

```
    //主菜单函数
    void welcome()
    {
        int choice;
        system("pause");
    Q:system("cls");  //清屏
        cout << "                    主菜单" << endl;
        cout << "----------------------------------------" << endl;
        cout << "****************************************" << endl;
        cout << endl;
        cout << "1.插入新人物" << endl << endl;
        cout << "2.删除人物" << endl << endl;
        cout << "3.修改人物信息" << endl << endl;
        cout << "4.查找人物" << endl << endl;
        cout << "5.人物关系查询" << endl << endl;
        cout << "6.凹入表方式打印树状家谱" << endl << endl;
        cout << "7.退出菜单并保存家谱" << endl << endl;
        cout << "****************************************" << endl;
        cout << "----------------------------------------" << endl << endl;
        cout << "请输入序号：";
        fflush(stdin);
        cin >> choice;

        switch (choice)
        {
        case 1:  //插入新人物
            InsertNew();
            system("pause");
            goto Q;  //从 A 处继续循环
        case 2:  //删除人物
            Delete();
            system("pause");
            goto Q;
        case 3:  //修改人物信息
```

```
            Change();
            system("pause");
            goto Q;
        case 4:  //查找人物
            Search();
            system("pause");
            goto Q;
        case 5:  //人物关系查询
            Relationship();
            system("pause");
            goto Q;
        case 6:  //凹入表方式打印树状家谱
            PrintASTree(T);
            system("pause");
            goto Q;
        case 7:  //退出菜单并保存家谱
            Save(T);
            cout << "已保存到 c:\\Users\\youye\\Desktop\\lastFamily.txt" << endl;
            break;
        default:
            goto Q;
        }
        return;
    }
```

③插入新人物函数（void InsertNew();）

```
//插入新人物函数
void InsertNew()
{
    char FatherName[100];
    A = NULL;  //使全局变量 A 和 B 都为空
    B = NULL;
    CreateCSTNode(A);  //创建临时结点 A
    cout << "请输入：姓名    性别（男为 1/女为 2）    配偶姓名  出生日期  状态（健在为 1/身故为 0）    父亲姓名" << endl;
    cout << "例如：张三    1    无    2002 01 01    1    张大" << endl;
    cin >> A->people_data.name >> A->people_data.sex >> A->people_data.fedname >> A->people_data.birth_data.year >> A->people_data.birth_data.month >> A->people_data.birth_data.day >> A->people_data.isalive >> FatherName;
    cout << A->people_data.name << " " << A->people_data.sex << "    " << A->people_data.fedname << " " << A->people_data.birth_data.year << "." << A->people_data.birth_data.month << "." << A->people_data.birth_data.day << "    " << A->people_data.isalive << "    " << FatherName;
```

```
        if ((A->people_data.sex != 1) && (A->people_data.sex != 0))
        {
            cout << "性别输入有误！" << endl;
            delete A;
            A = NULL;
            return;
        }
        if ((A->people_data.isalive != 1) && (A->people_data.isalive != 0))
        {
            cout << "状态输入有误！" << endl;
            delete A;
            A = NULL;
            return;
        }

        SearchNode(T, FatherName, B);  //查找 A 的父亲结点并赋值到 B 结点中
        if (B == NULL)
        {
            cout << "此人的父亲不在家谱里！" << endl;
            delete A;
            A = NULL;
            return;
        }
        int Abirth = (A->people_data.birth_data.year) * 10000 +
(A->people_data.birth_data.month) * 100 + (A->people_data.birth_data.day);
        int Bbirth = (B->people_data.birth_data.year) * 10000 +
(B->people_data.birth_data.month) * 100 + (B->people_data.birth_data.day);
        //如果 B 作为临时父亲结点存在且不是根结点且生日晚于 A
        if ((B->people_data.seniority != 0) && (Bbirth >= Abirth))
        {
            cout << "孩子生日不能比父亲早" << endl;
            delete A;
            A = NULL;
            B = NULL;
            return;
        }

        InsertNode(B, A);  //在父亲结点下插入 A
        A = NULL;  //令全局变量 A 恢复为空
        return;
    }
```

④删除人物函数（void Delete();）

```
    //删除人物函数
```

```
void Delete()
{
    A = NULL;  //全局变量 A 为空
    char name[100];
    cout << "请输入要删除的人物姓名：";
    cin >> name;
    SearchNode(T, name, A);  //根据名字查找目标并存到 A 结点中
    if (A == NULL)
    {
        cout << "不存在姓名为" << name << "的人";
        return;
    }
    if (A->first_Child != NULL)
    {
        cout << "此人有后代，不允许删除！";
        return;
    }
    DeleteNode(A);
    A = NULL;
}
```

⑤修改人物信息函数（void Change();）

```
//修改人物信息
void Change()
{
    A = NULL;
    char name[100];
    cout << "请输入要修改信息的人物姓名：";
    cin >> name;
    SearchNode(T, name, A);  //根据名字查找目标并存在 A 结点中
    if (A == NULL)
    {
        cout << "不存在名字为" << name << "的人" << endl;
        return;
    }
    ChangePeopleInfo(A, T);  //进入修改信息界面
    A = NULL;  //令全局变量 A 为空
}
```

⑥查找人物函数（void Search();）

```
//查找人物
void Search()
{
    A = NULL;
```

```
    char name[100];
    cout << "请输入要查找的人物姓名：";
    cin >> name;
    SearchNode(T, name, A);  //根据名字查找目标并存在 A 结点中
    if (A == NULL)
    {
        cout << "不存在名字为" << name << "的人" << endl;
        return;
    }
    ShowPeopleInfo(A);  //列出目标信息
    A = NULL;  //令全局变量 A 为空
}
```

⑦人物关系查询函数（void Relationship();）

```
//人物关系查询
void Relationship()
{
    A = NULL;
    B = NULL;  //令全局变量都为 NULL
    char name[100];
    cout << "请输入需要查询关系的人物姓名：";
    cin >> name;
    SearchNode(T, name, B);  //查询目标并将其保存到 B 结点中
    if (B == NULL)
    {
        cout << "不存在名字为" << name << "的人" << endl;
        return;
    }

    int choice;
C:system("cls");  //清屏
    ShowPeopleInfo(B);  //列出目标人物的具体信息
    cout << endl;

    cout << "          人物关系查询菜单" << endl << endl;
    cout << "****************************************" << endl << endl;
    cout << "1.祖先" << endl;
    cout << "2.堂兄弟" << endl;
    cout << "3.同世代的兄弟" << endl;
    cout << "4.后代" << endl;
    cout << "5.break" << endl << endl;
    cout << "****************************************" << endl << endl;
    cout << "请输入序号：";
    cin >> choice;
```

```
        switch (choice)
        {
        case 1:  //查询祖先
            A = B->father;
            //只要 A 不等于根结点，遍历 A 的父亲
            for (; A->people_data.seniority != 0; A = A->father)
            {
                cout << "[" << A->people_data.seniority << "][第" <<
A->people_data.seniority + 12 << "世][" << A->people_data.birth_data.year << "." <<
A->people_data.birth_data.month << "." << A->people_data.birth_data.day << "]" <<
A->people_data.name << endl;
            }
            system("pause");
            goto C;  //返回查询菜单继续循环
        case 2:  //查询堂兄弟
            InorderCousin(T, B);
            system("pause");
            goto C;
        case 3:  //查询同世代的兄弟
            InorderBro(T, B);
            system("pause");
            goto C;
        case 4:  //查询后代关系
            PrintTree(B);  //层次遍历
            system("pause");
            goto C;
        case 5:  //退出查询菜单
            break;
        default:
            goto C;
        }
        B = NULL;
        A = NULL;
        return;
            }
```

⑧退出菜单保存修改后的文件,凹入表保存家谱（void Save(CSTree T);）

```
//退出菜单保存修改后的文件,凹入表保存家谱
void Save(CSTree T)
{
    int cnt;
    if (T)
    {
```

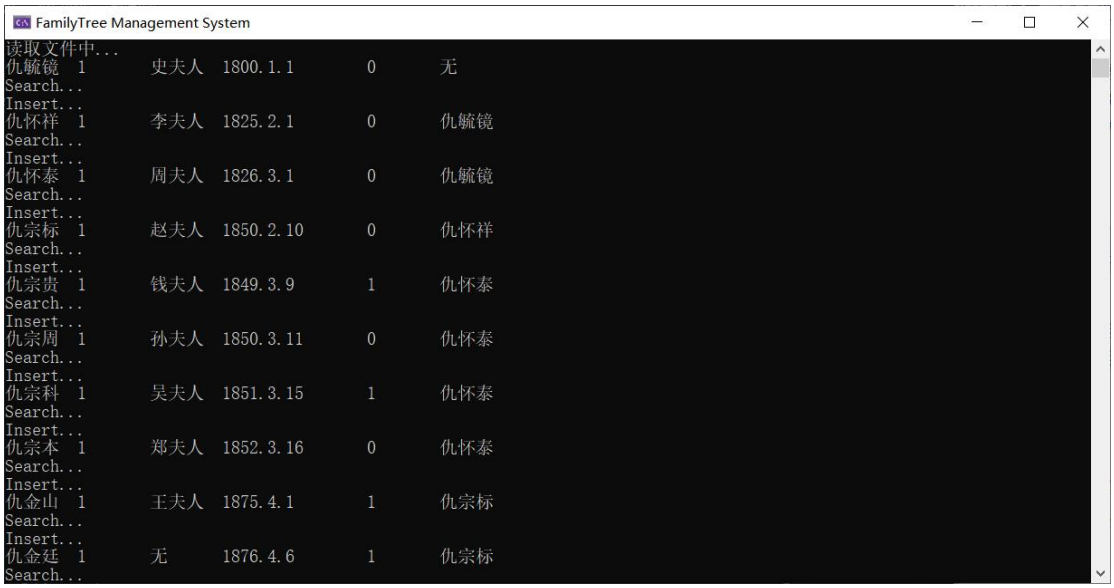
```
            //输出空格
            for (cnt = 1; cnt < T->people_data.seniority; cnt++)
                in << "    ";
            //输出字符
            in << "[" << T->people_data.seniority << "][第" << T->people_data.seniority
+ 12 << "世][" << T->people_data.birth_data.year << "." << T->people_data.birth_data.month
<< "." << T->people_data.birth_data.day << "]" << T->people_data.name << endl;
            in << endl;
            Save(T->first_Child);
            Save(T->next_brother);
        }
    }
```

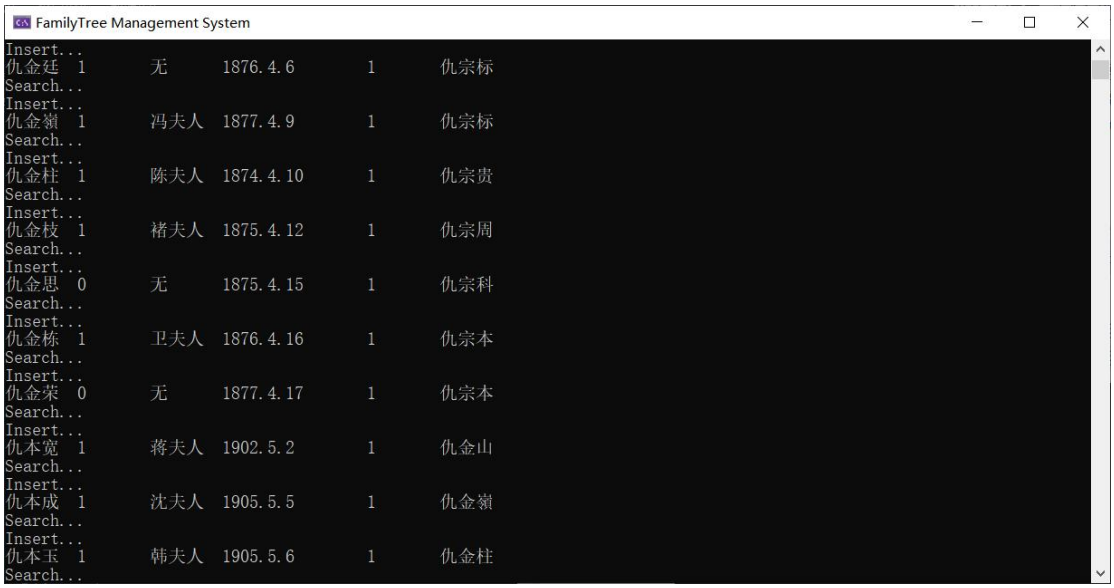
四、 运行截图

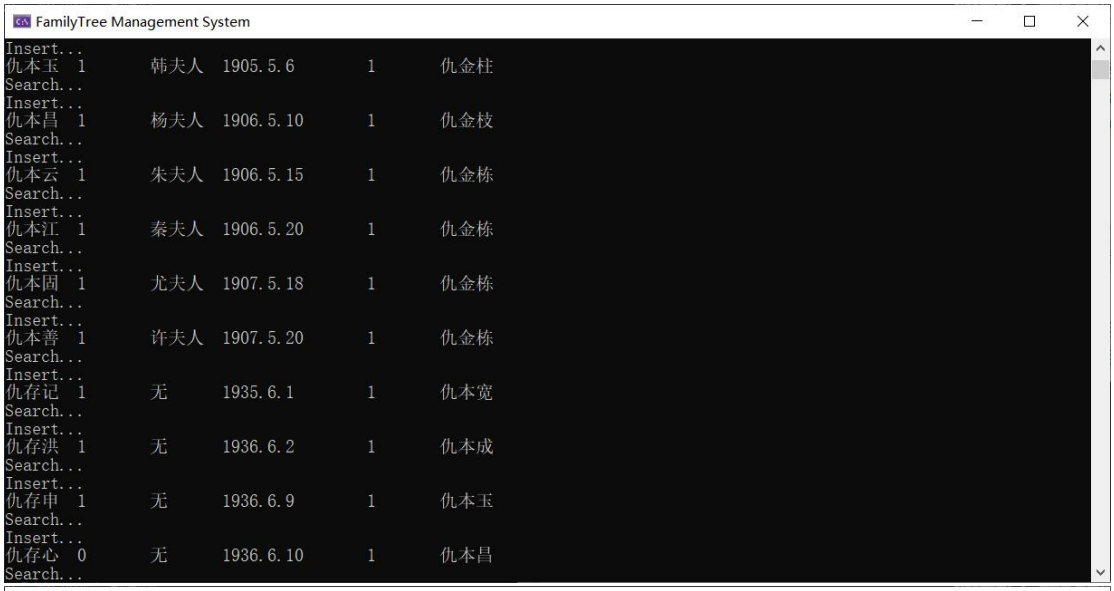
（通过功能截图和文字的方式，说明系统的使用方法）

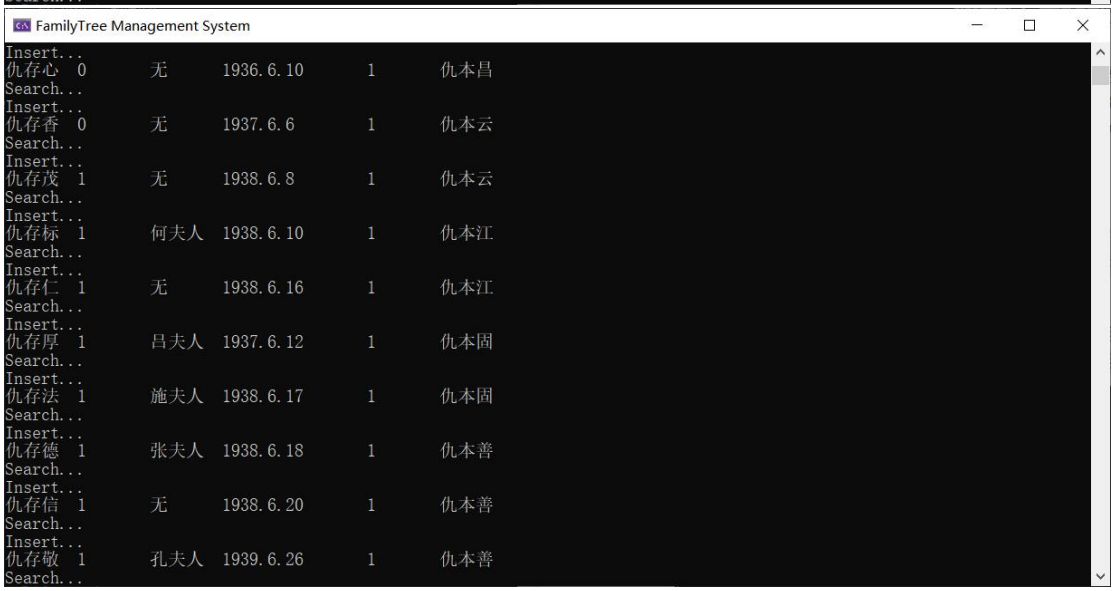
4.1 打开程序，程序自动读取 FamilyTree.txt 文件，自动生成兄弟孩子树：

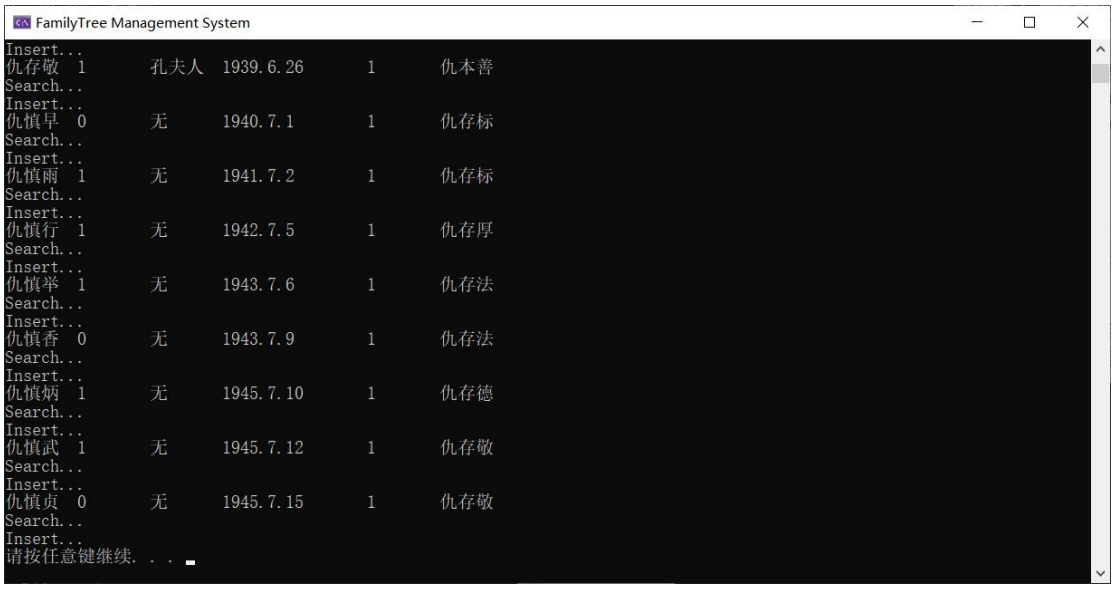
控制台将显示：“读取文件中...”，逐行读取的过程中，“Search...”表明在家谱树中查找读取的人物及其父亲结点，并判断其内容是否合理合规，“Insert...”表明查找结束在将该读取内容插入新人物结点。



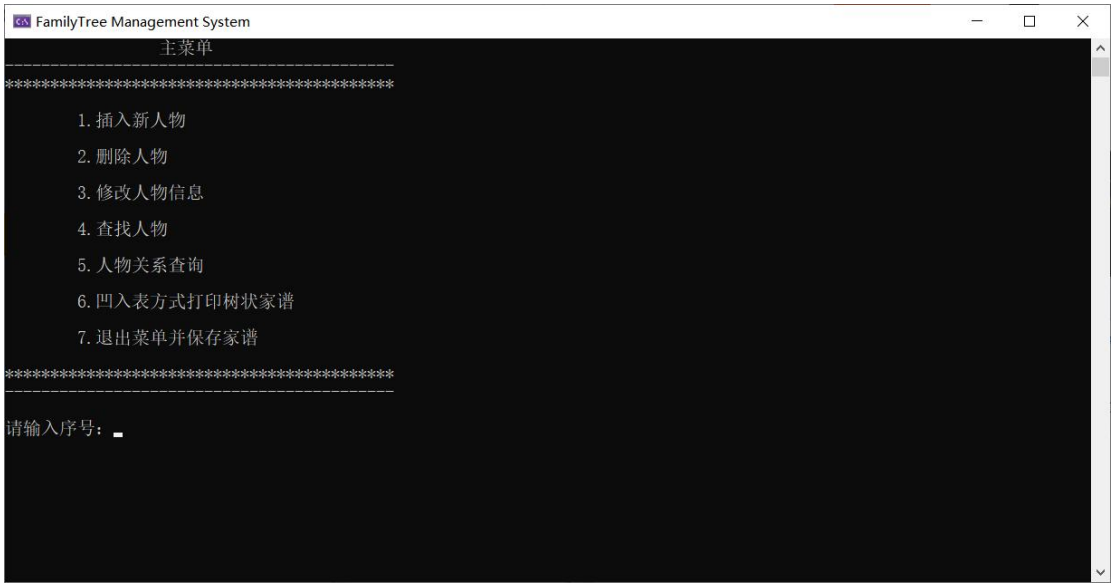

FamilyTree Management System
Insert...
仇金廷 1 无 1876.4.6 1 仇宗标
Search...
Insert...
仇金嶺 1 冯夫人 1877.4.9 1 仇宗标
Search...
Insert...
仇金柱 1 陈夫人 1874.4.10 1 仇宗贵
Search...
Insert...
仇金枝 1 褚夫人 1875.4.12 1 仇宗周
Search...
Insert...
仇金思 0 无 1875.4.15 1 仇宗科
Search...
Insert...
仇金栋 1 卫夫人 1876.4.16 1 仇宗本
Search...
Insert...
仇金荣 0 无 1877.4.17 1 仇宗本
Search...
Insert...
仇本宽 1 蒋夫人 1902.5.2 1 仇金山
Search...
Insert...
仇本成 1 沈夫人 1905.5.5 1 仇金嶺
Search...
Insert...
仇本玉 1 韩夫人 1905.5.6 1 仇金柱
Search...


FamilyTree Management System
Insert...
仇本玉 1 韩夫人 1905.5.6 1 仇金柱
Search...
Insert...
仇本昌 1 杨夫人 1906.5.10 1 仇金枝
Search...
Insert...
仇本云 1 朱夫人 1906.5.15 1 仇金栋
Search...
Insert...
仇本江 1 秦夫人 1906.5.20 1 仇金栋
Search...
Insert...
仇本固 1 尤夫人 1907.5.18 1 仇金栋
Search...
Insert...
仇本善 1 许夫人 1907.5.20 1 仇金栋
Search...
Insert...
仇存记 1 无 1935.6.1 1 仇本宽
Search...
Insert...
仇存洪 1 无 1936.6.2 1 仇本成
Search...
Insert...
仇存申 1 无 1936.6.9 1 仇本玉
Search...
Insert...
仇存心 0 无 1936.6.10 1 仇本昌
Search...


FamilyTree Management System
Insert...
仇存心 0 无 1936.6.10 1 仇本昌
Search...
Insert...
仇存香 0 无 1937.6.6 1 仇本云
Search...
Insert...
仇存茂 1 无 1938.6.8 1 仇本云
Search...
Insert...
仇存标 1 何夫人 1938.6.10 1 仇本江
Search...
Insert...
仇存仁 1 无 1938.6.16 1 仇本江
Search...
Insert...
仇存厚 1 吕夫人 1937.6.12 1 仇本固
Search...
Insert...
仇存法 1 施夫人 1938.6.17 1 仇本固
Search...
Insert...
仇存德 1 张夫人 1938.6.18 1 仇本善
Search...
Insert...
仇存信 1 无 1938.6.20 1 仇本善
Search...
Insert...
仇存敬 1 孔夫人 1939.6.26 1 仇本善
Search...



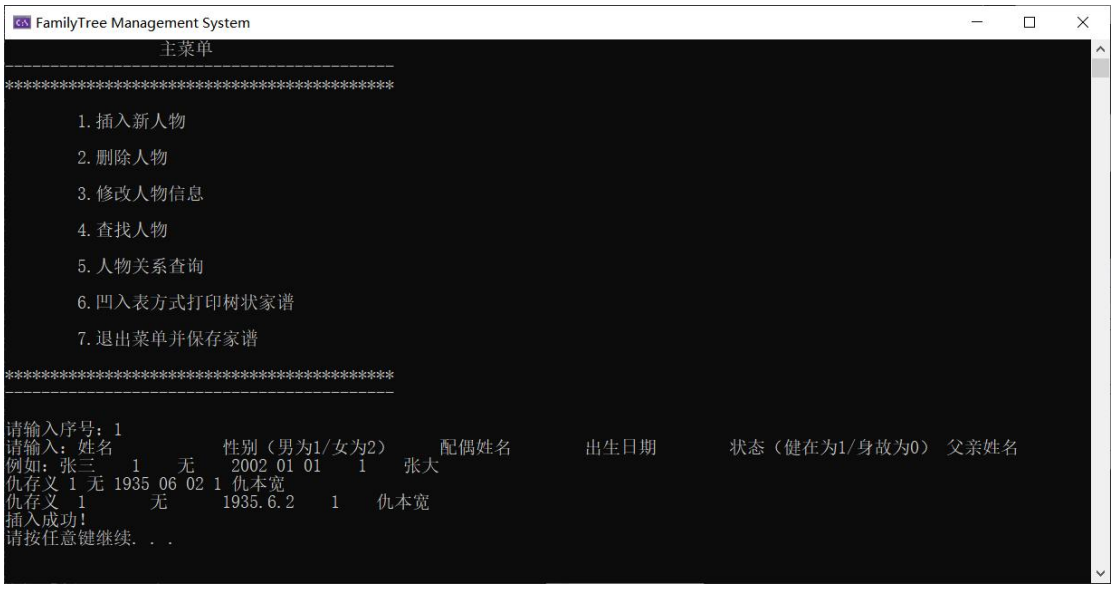
4.2 按任意键继续后，进入主菜单，可以按需输入对应功能序号：



4.3 插入新人物：

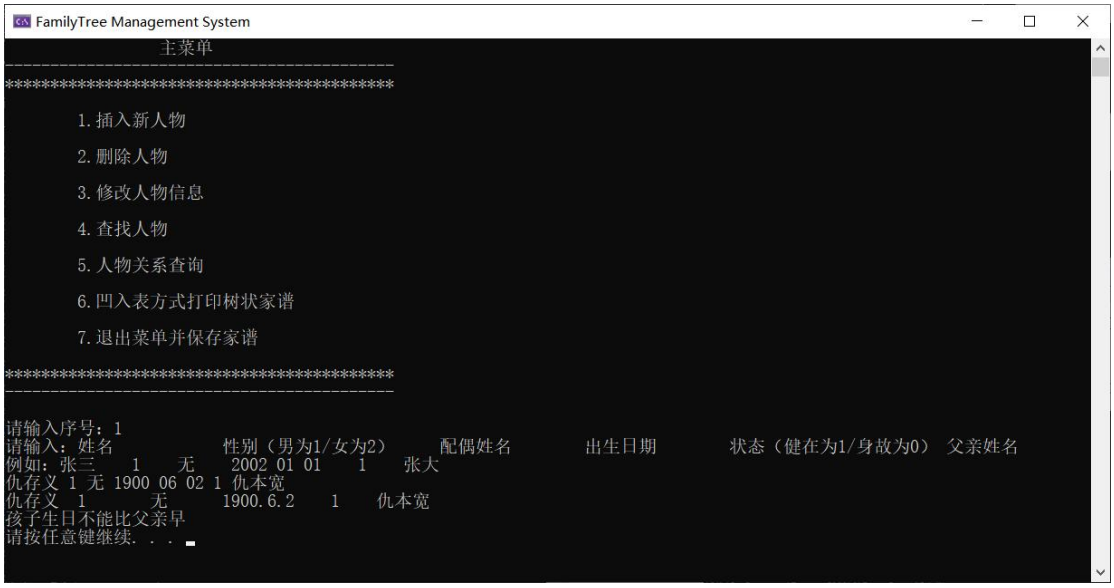
输入序号 1，可以插入新人物，需要输入正确的格式和值才能够插入成功。

正确输入：

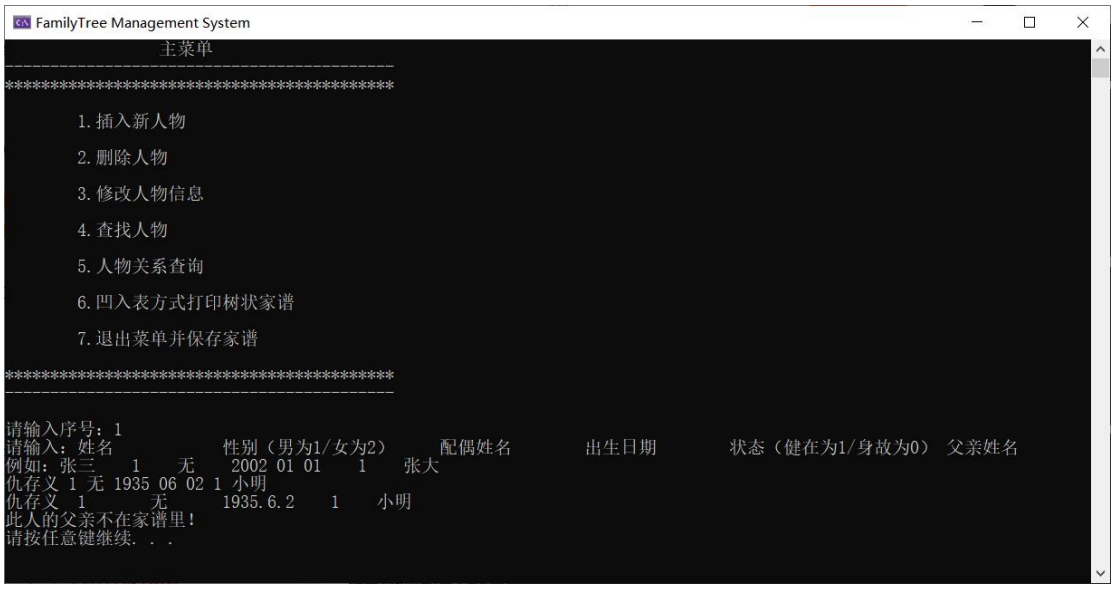


错误输入：

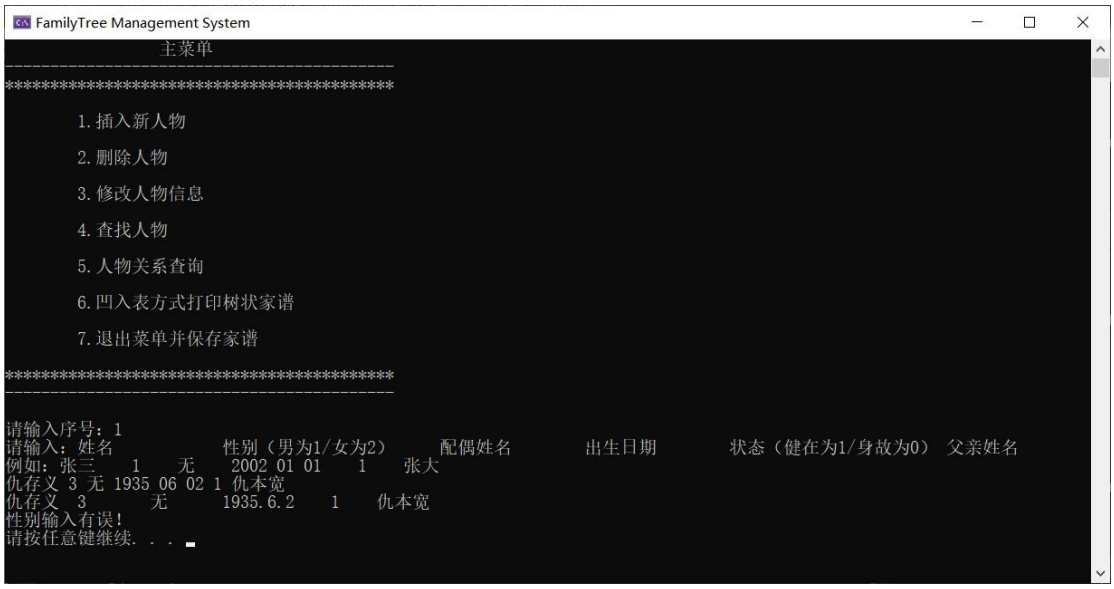
①生日输入有误-早于父亲



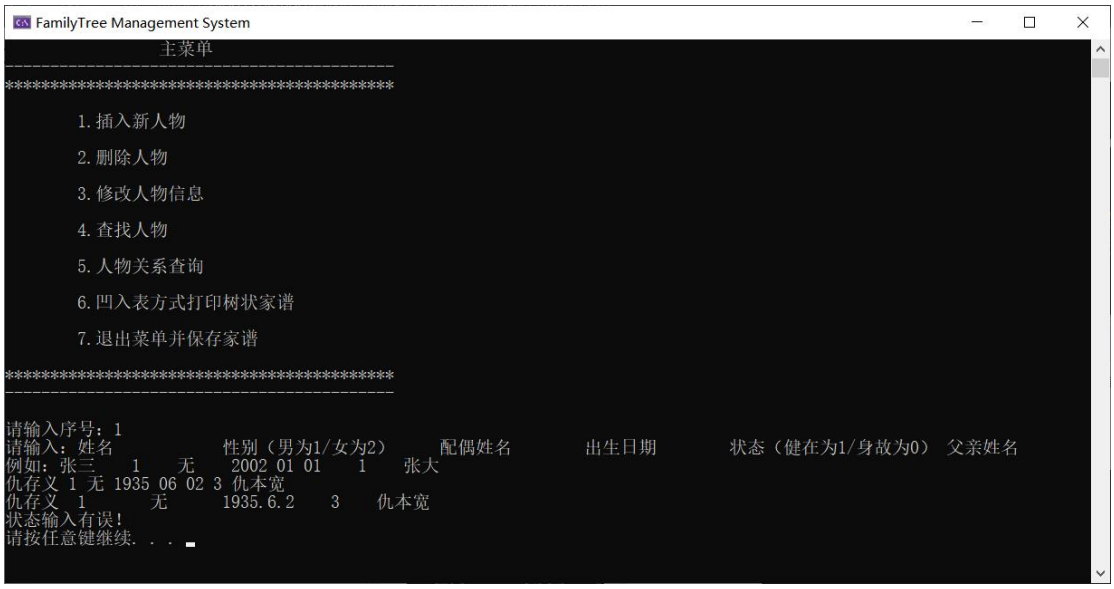
②父亲姓名输入有误-父亲不在家谱中



③性别输入有误-非男为 1，女为 2



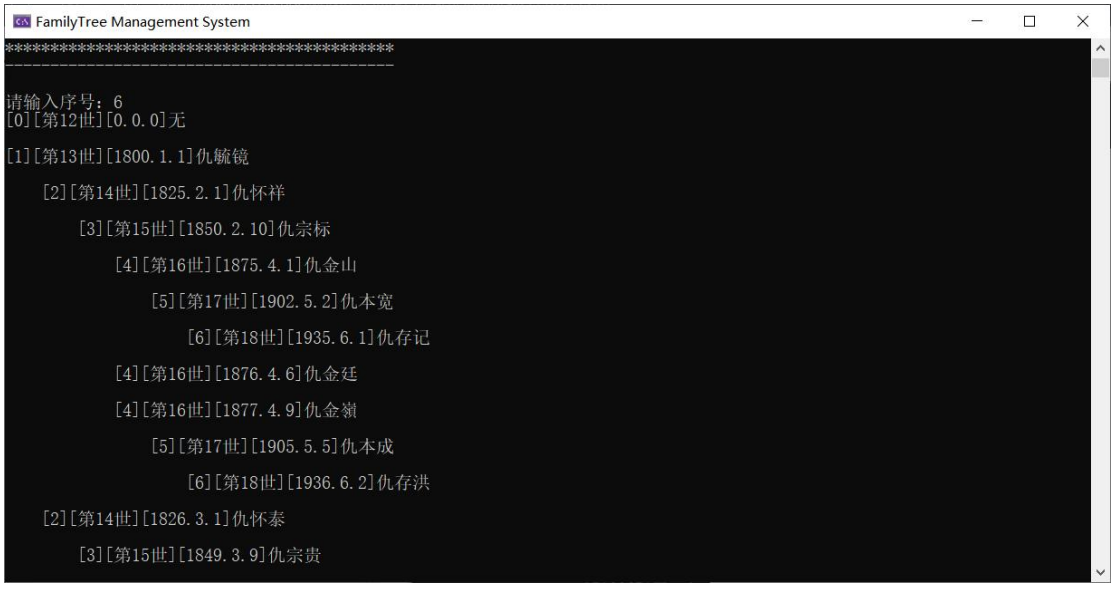
④状态输入有误-非健在为 1，身故为 2



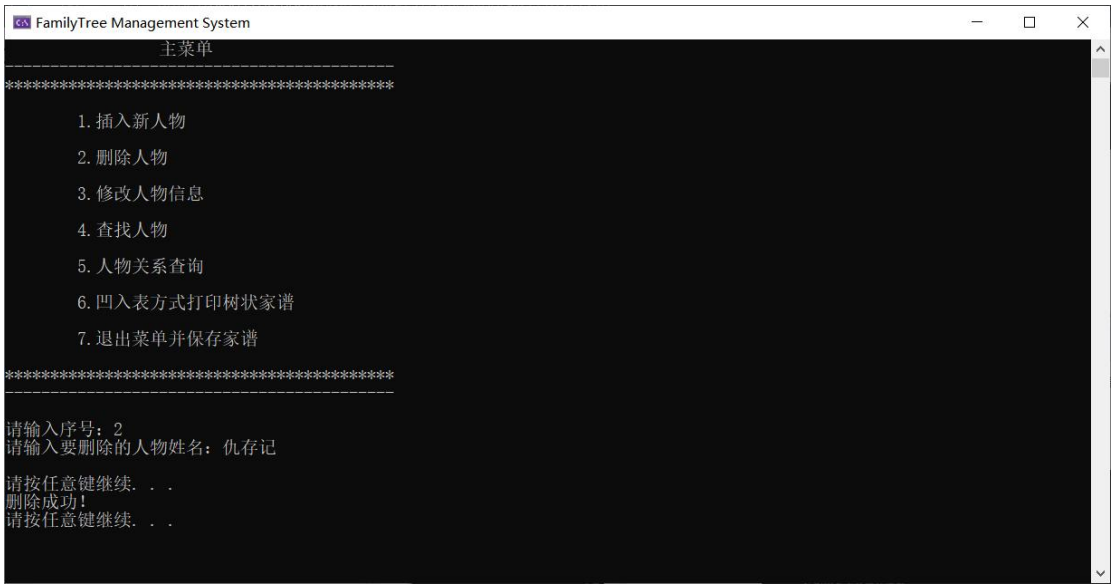
4.4 删除人物：

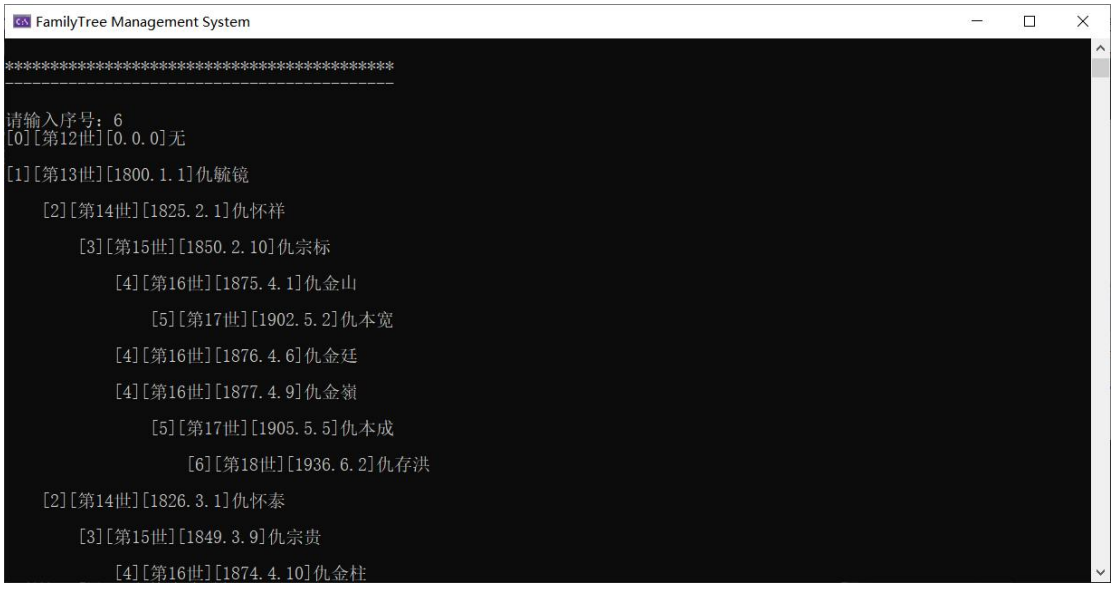
输入序号 2，可以删除家谱中某个查询到的人物，如果该人物有后代则不允许删除。

删除前：

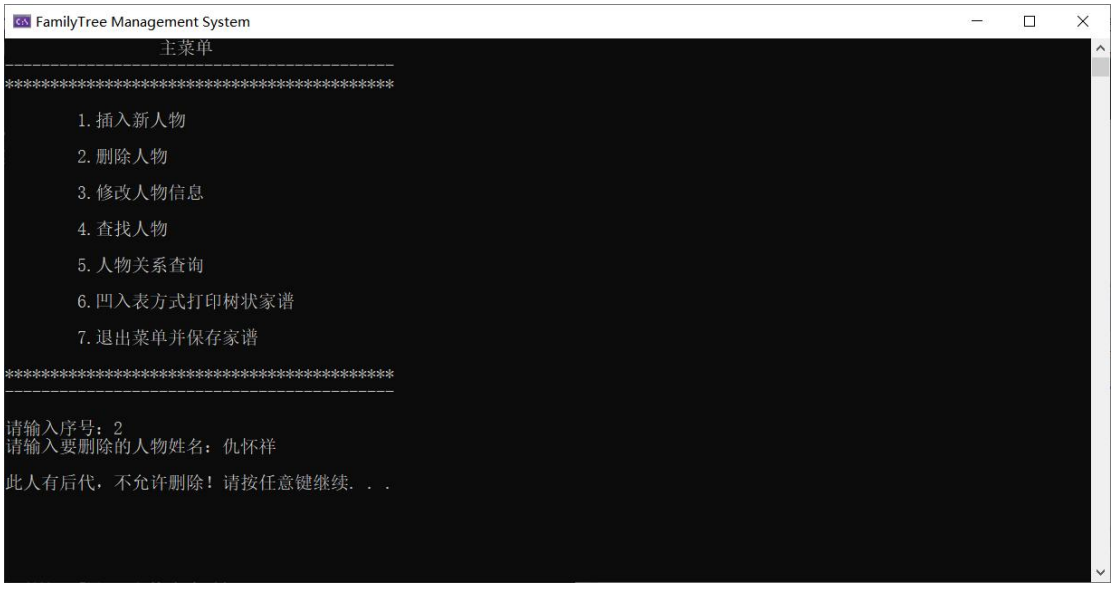


删除无后代的仇存记后：


FamilyTree Management System
主菜单
1.插入新人物
2.删除人物
3.修改人物信息
4.查找人物
5.人物关系查询
6.凹入表方式打印树状家谱
7.退出菜单并保存家谱
请输入序号：2
请输入要删除的人物姓名：仇存记
请按任意键继续. . .
删除成功!
请按任意键继续. . .


FamilyTree Management System
请输入序号：6
[0][第12世][0.0.0]无
[1][第13世][1800.1.1]仇毓镜
[2][第14世][1825.2.1]仇怀祥
[3][第15世][1850.2.10]仇宗标
[4][第16世][1875.4.1]仇金山
[5][第17世][1902.5.2]仇本宽
[4][第16世][1876.4.6]仇金廷
[4][第16世][1877.4.9]仇金嶺
[5][第17世][1905.5.5]仇本成
[6][第18世][1936.6.2]仇存洪
[2][第14世][1826.3.1]仇怀泰
[3][第15世][1849.3.9]仇宗贵
[4][第16世][1874.4.10]仇金柱

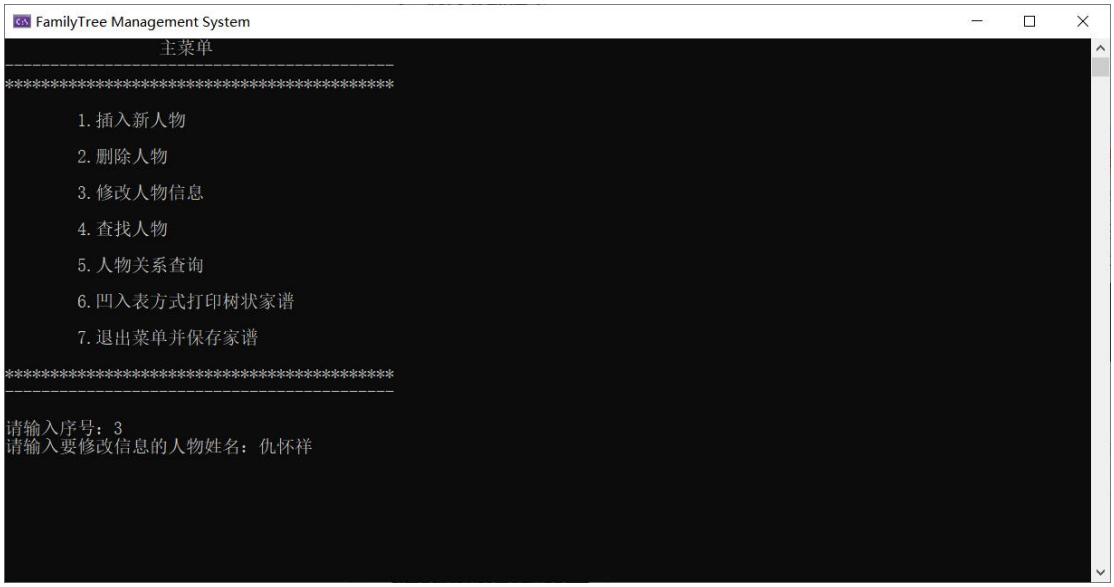
删除有后代的仇怀祥：

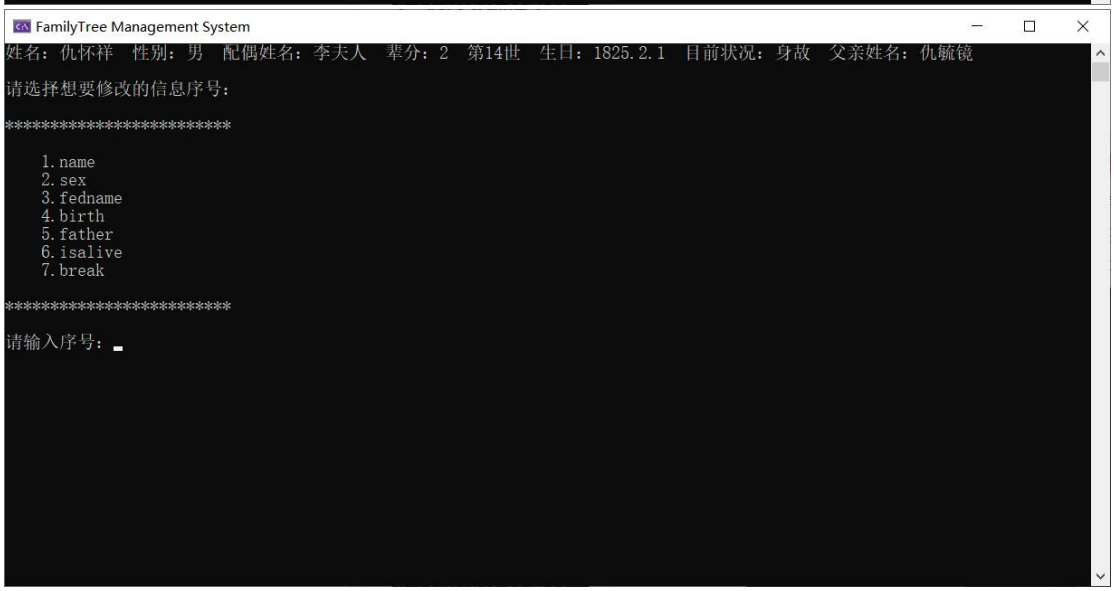

FamilyTree Management System
主菜单
1.插入新人物
2.删除人物
3.修改人物信息
4.查找人物
5.人物关系查询
6.凹入表方式打印树状家谱
7.退出菜单并保存家谱
请输入序号：2
请输入要删除的人物姓名：仇怀祥
此人有后代，不允许删除！请按任意键继续. . .

4.5 修改人物信息：

输入序号 3，可以修改家谱中某个人物的具体信息，进入修改信息的菜单界面，可以修改六个性质的信息，输入 7 则返回主菜单。

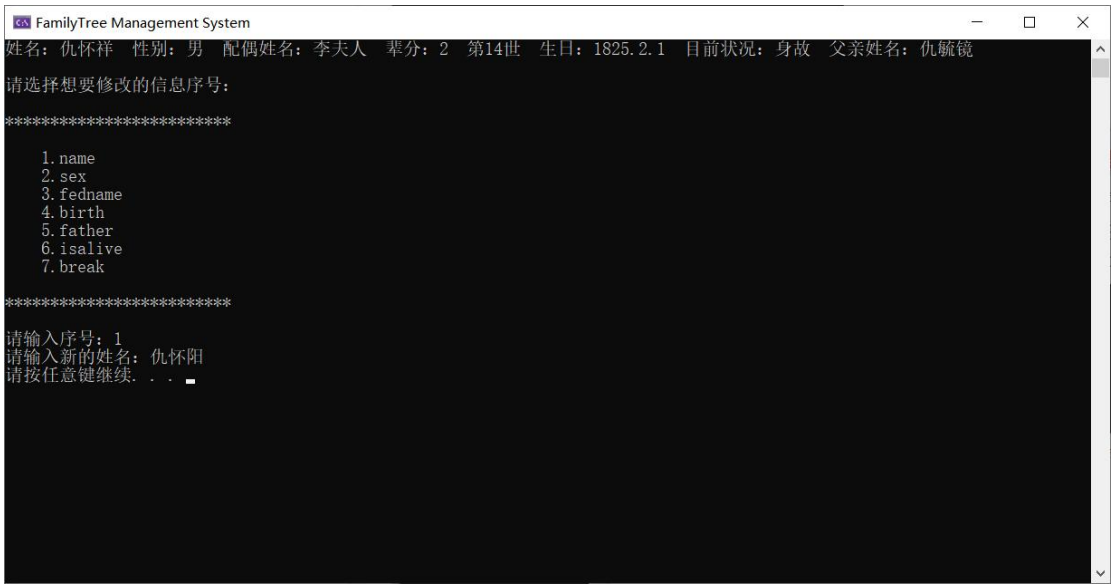
修改仇怀祥的信息：

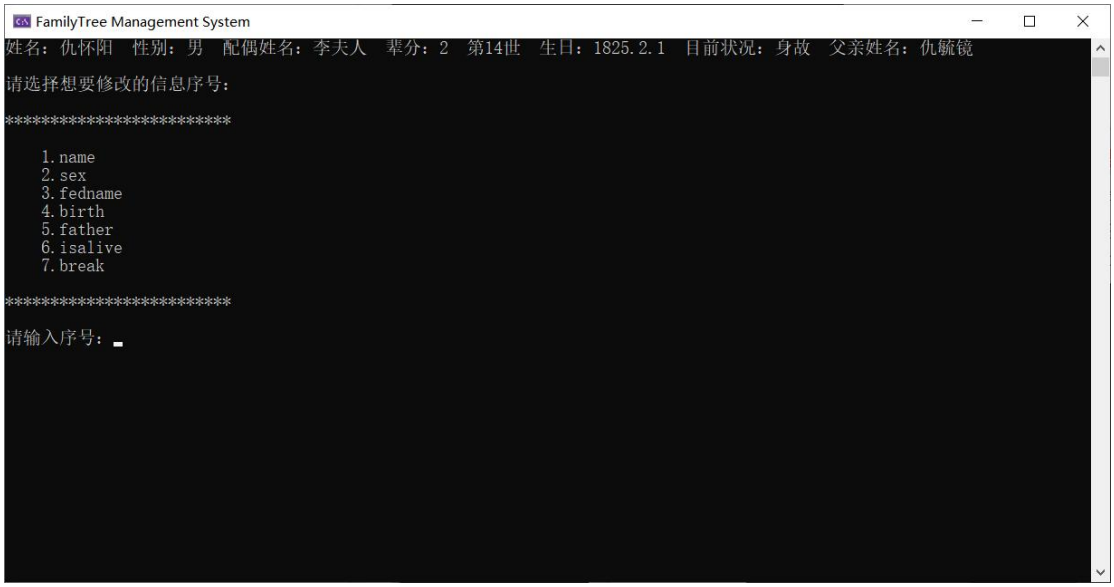




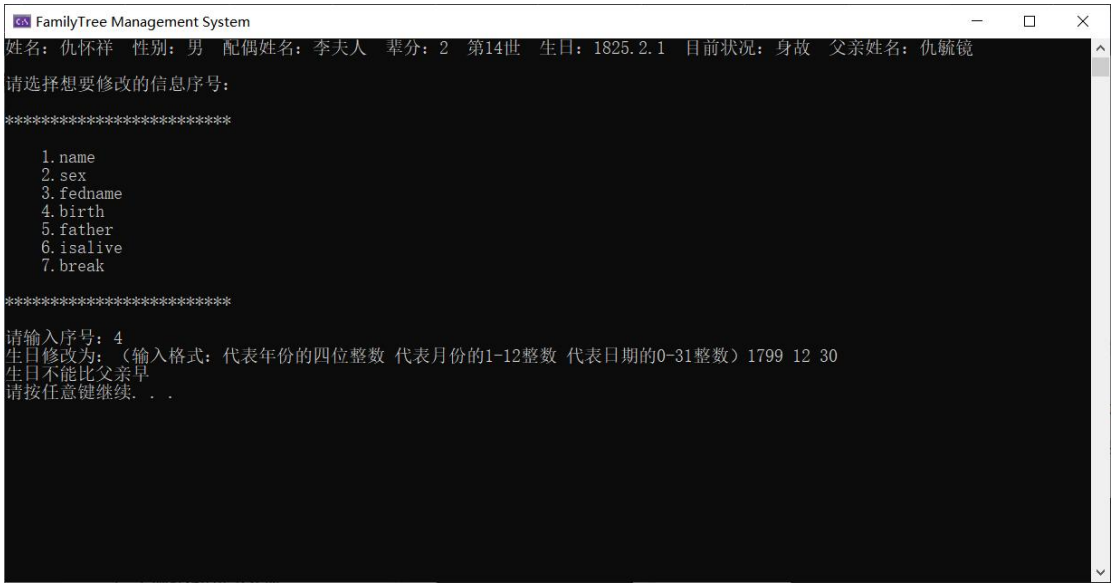
进入新界面后可以看到其详细信息，可选择欲修改的信息序号。

修改姓名：

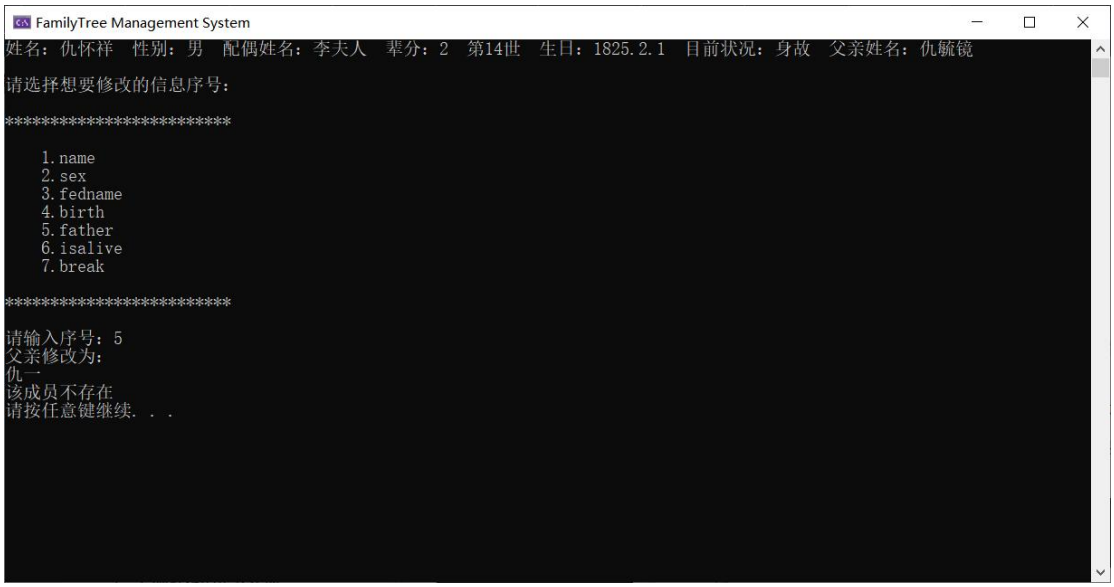

FamilyTree Management System
姓名：仇怀祥 性别：男 配偶姓名：李夫人 辈分：2 第14世 生日：1825.2.1 目前状况：身故 父亲姓名：仇毓镜
请选择想要修改的信息序号：
****
1.name
2.sex
3.fedname
4.birth
5.father
6.isalive
7.break
****
请输入序号：1
请输入新的姓名：仇怀阳
请按任意键继续. . .

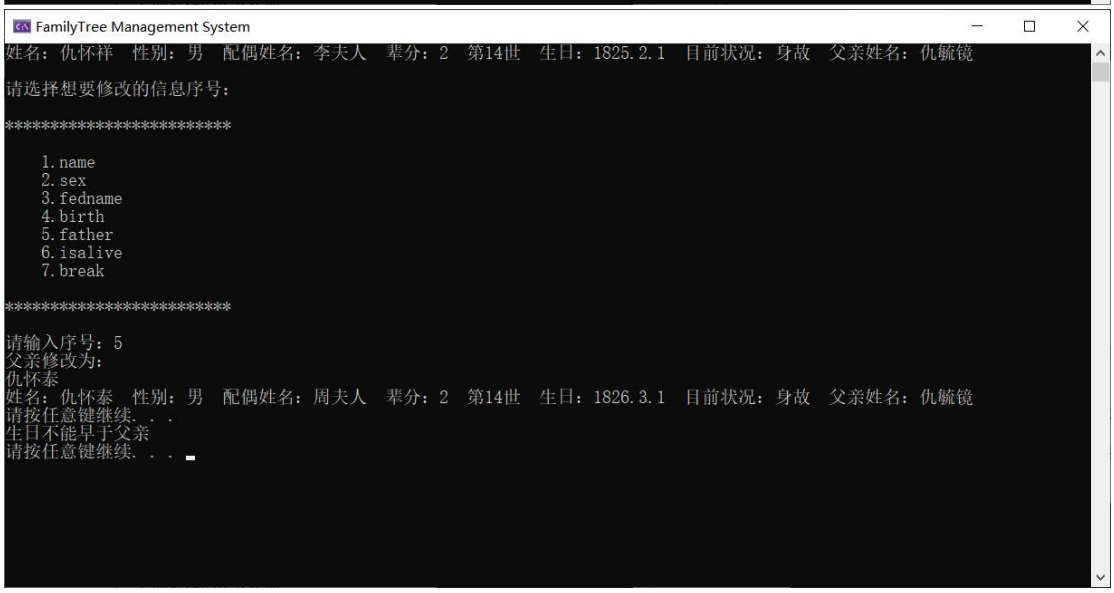

FamilyTree Management System
姓名：仇怀阳 性别：男 配偶姓名：李夫人 辈分：2 第14世 生日：1825.2.1 目前状况：身故 父亲姓名：仇毓镜
请选择想要修改的信息序号：
****
1.name
2.sex
3.fedname
4.birth
5.father
6.isalive
7.break
****
请输入序号：

修改生日：


FamilyTree Management System
姓名：仇怀祥 性别：男 配偶姓名：李夫人 辈分：2 第14世 生日：1825.2.1 目前状况：身故 父亲姓名：仇毓镜
请选择想要修改的信息序号：
****
1.name
2.sex
3.fedname
4.birth
5.father
6.isalive
7.break
****
请输入序号：4
生日修改为：（输入格式：代表年份的四位整数 代表月份的1-12整数 代表日期的0-31整数）1799 12 30
生日不能比父亲早
请按任意键继续. . .

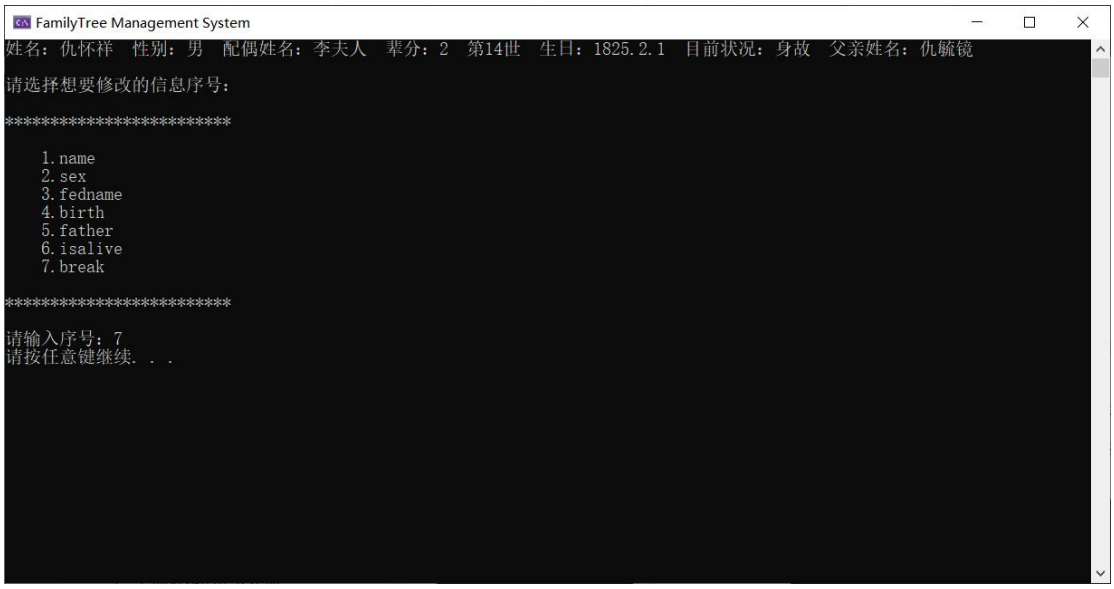
修改父亲：


FamilyTree Management System
姓名：仇怀祥　性别：男　配偶姓名：李夫人　辈分：2　第14世　生日：1825.2.1　目前状况：身故　父亲姓名：仇毓镜
请选择想要修改的信息序号：
****
1.name
2.sex
3.fedname
4.birth
5.father
6.isalive
7.break
****
请输入序号：5
父亲修改为：
仇一
该成员不存在
请按任意键继续. . .


FamilyTree Management System
姓名：仇怀祥　性别：男　配偶姓名：李夫人　辈分：2　第14世　生日：1825.2.1　目前状况：身故　父亲姓名：仇毓镜
请选择想要修改的信息序号：
****
1.name
2.sex
3.fedname
4.birth
5.father
6.isalive
7.break
****
请输入序号：5
父亲修改为：
仇怀泰
姓名：仇怀泰　性别：男　配偶姓名：周夫人　辈分：2　第14世　生日：1826.3.1　目前状况：身故　父亲姓名：仇毓镜
请按任意键继续. . .
生日不能早于父亲
请按任意键继续. . .

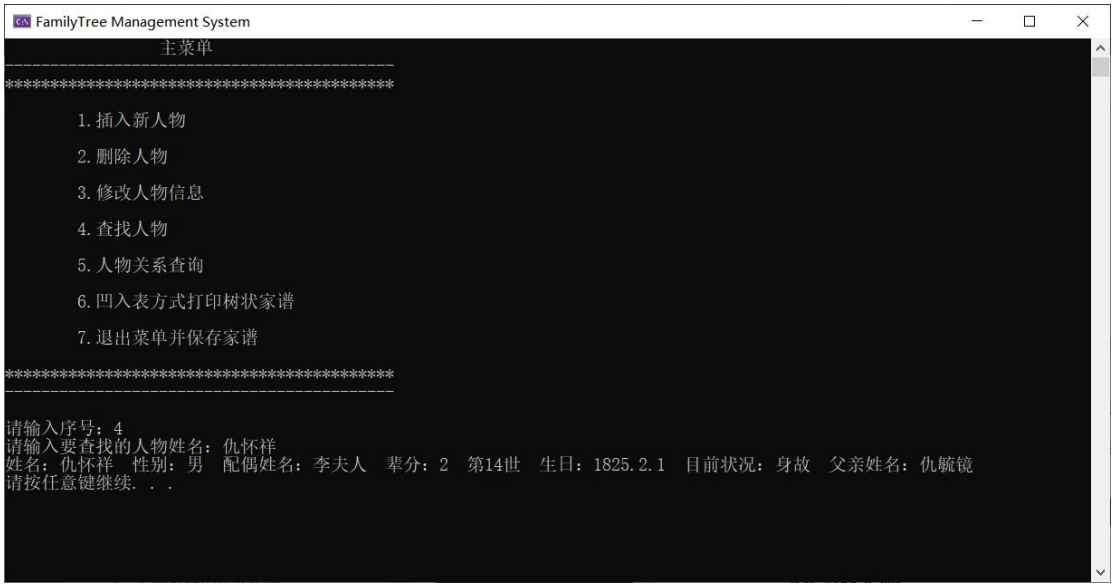
……

退出该界面：



4.6 查找人物：

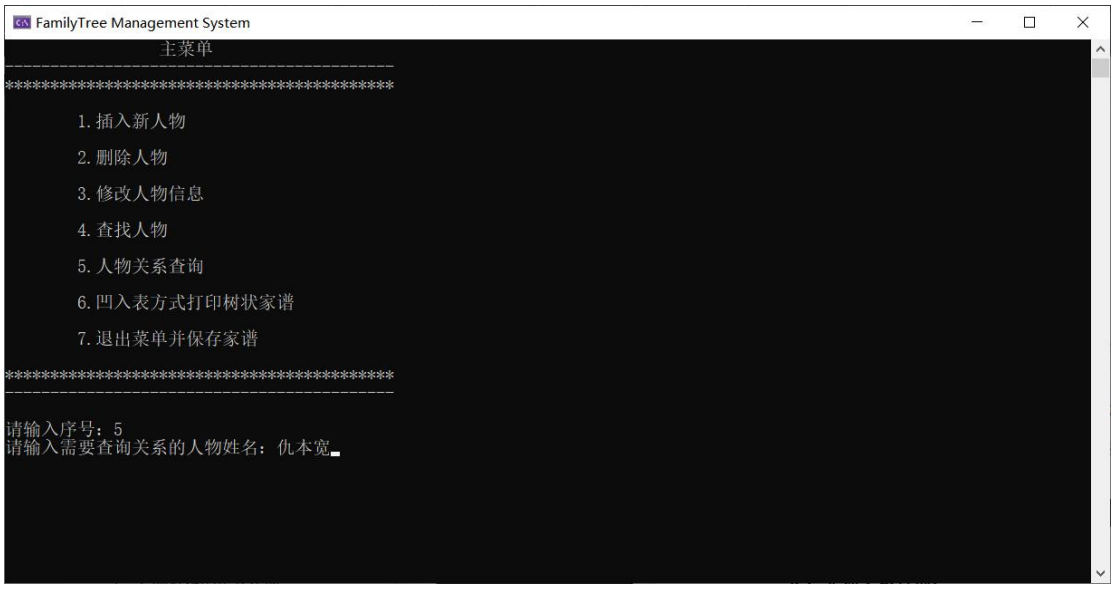
输入序号 4，可以查找家谱中某人的具体信息。



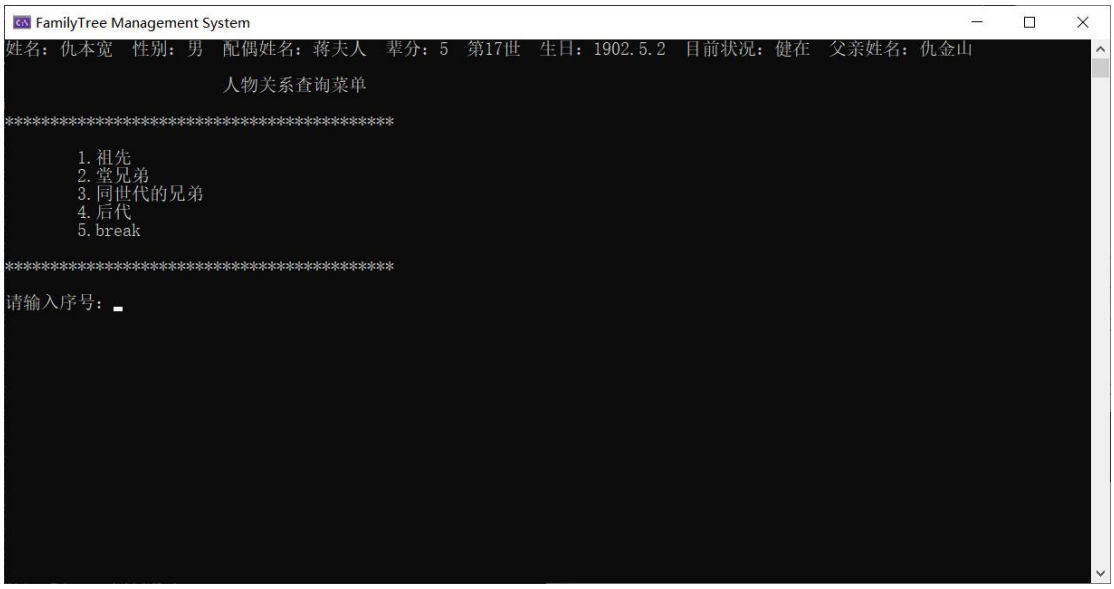
4.7 人物关系查询：

输入序号 5，可以进入人物关系查询的菜单界面，查找某人的父亲及祖先、同世代堂兄弟、孩子及后代的基本信息。

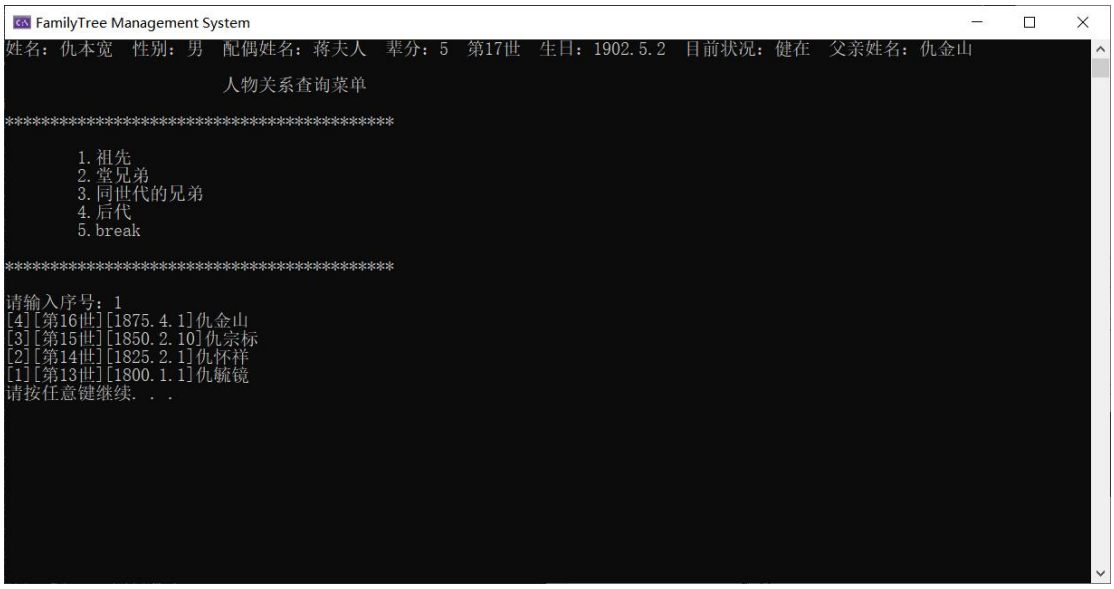
查询仇本宽的人物关系：



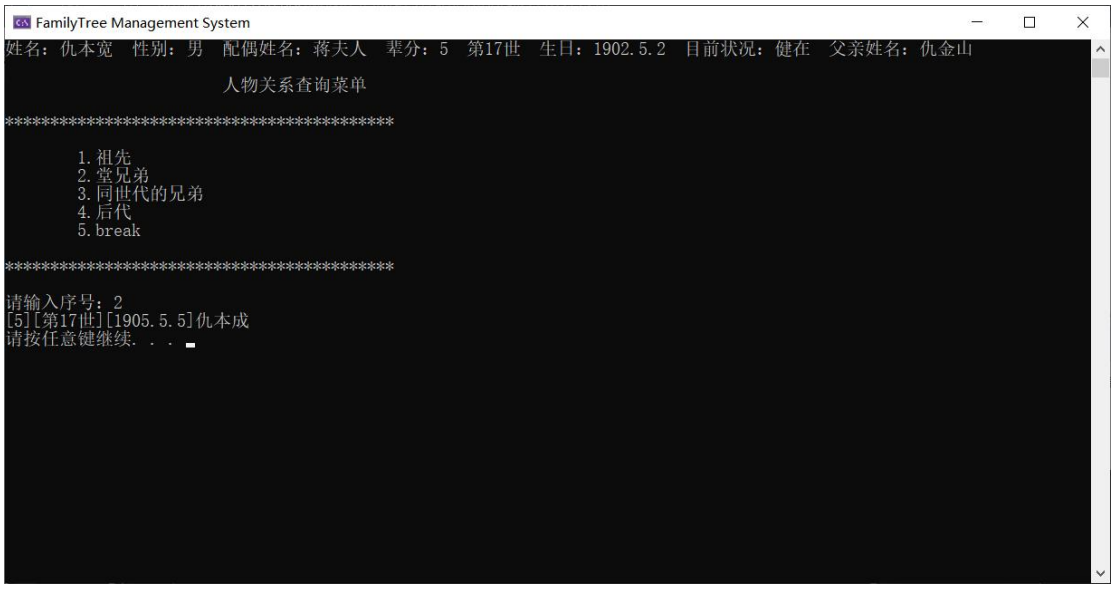
进入查询菜单：



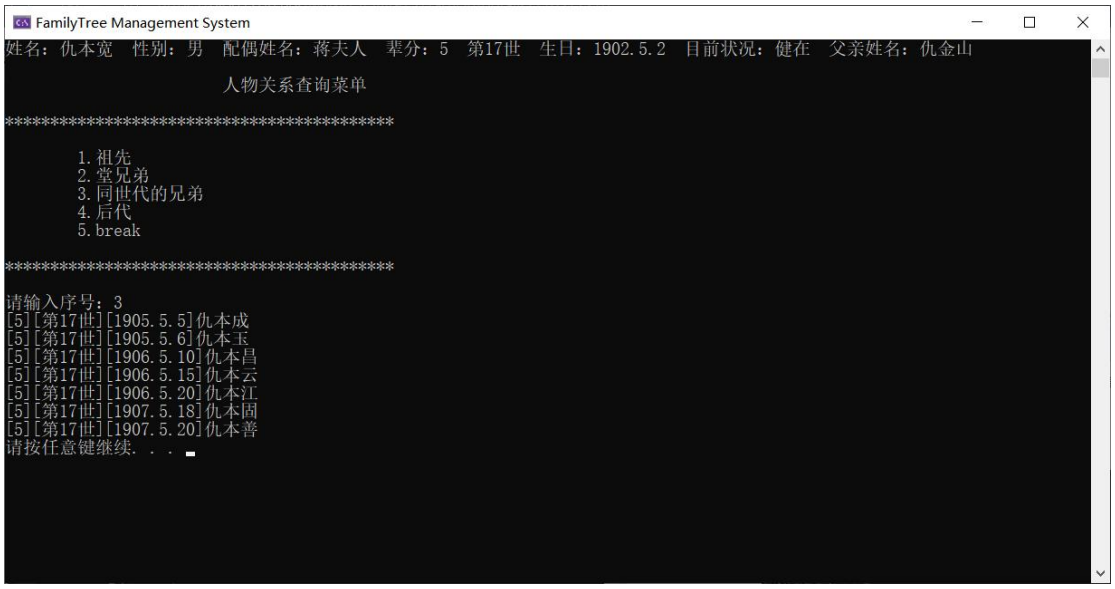
查询其父亲及其祖先：



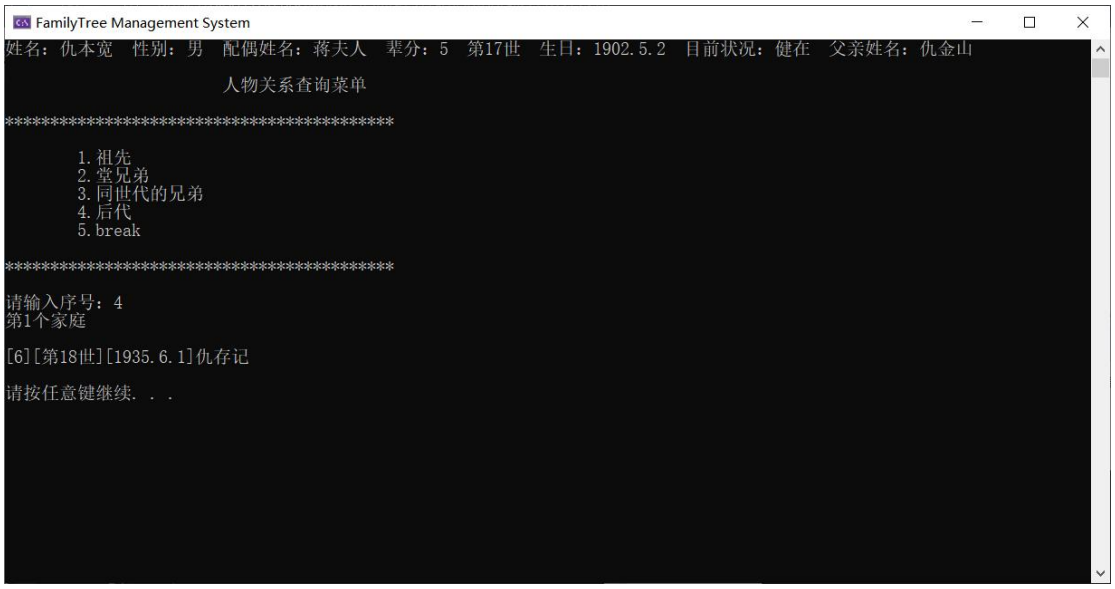
查询堂兄弟：



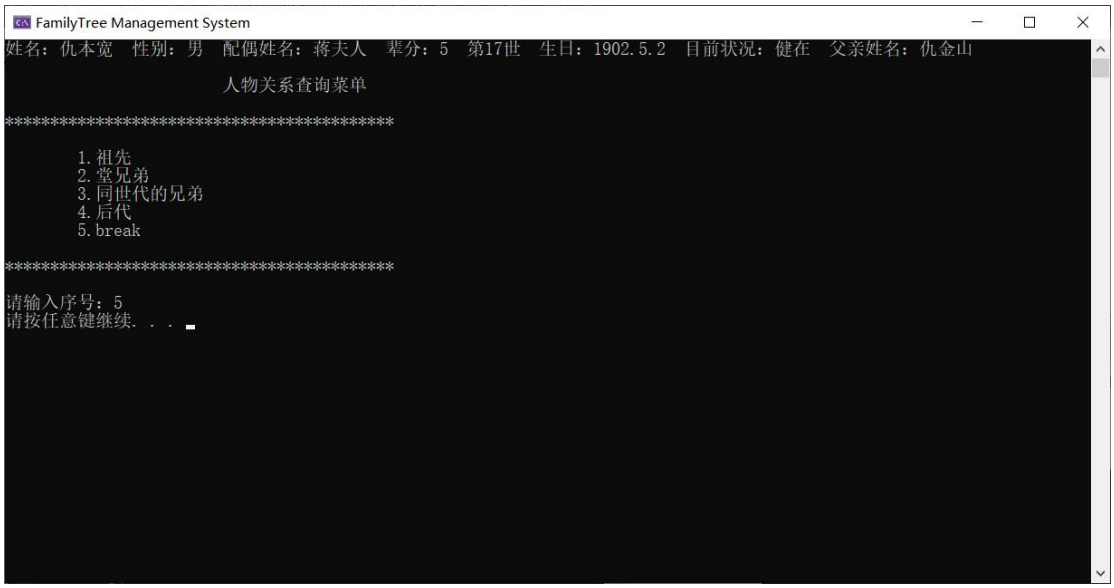
查询同世代的兄弟：

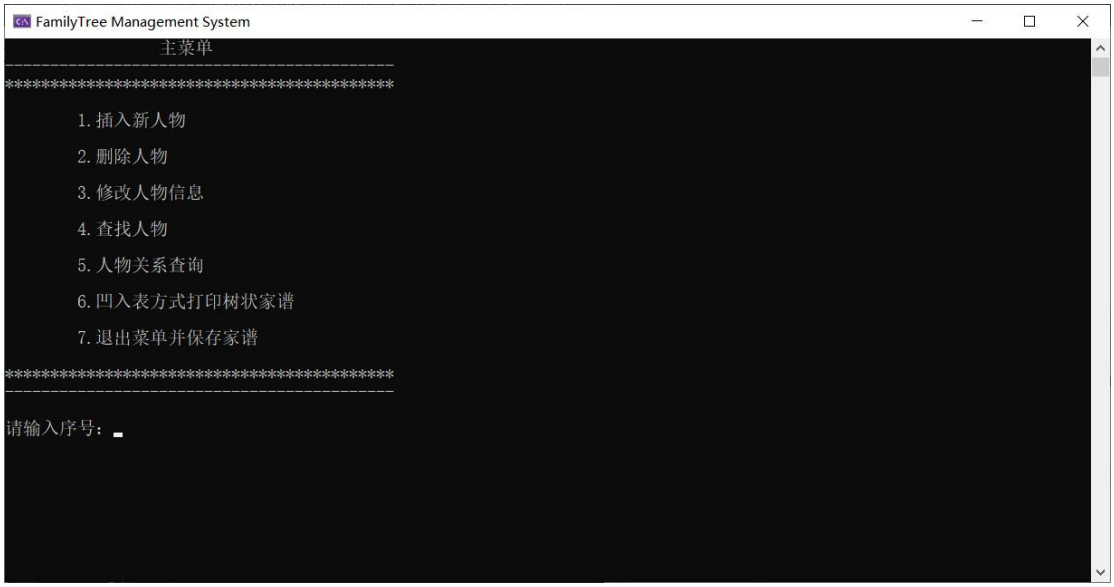


查询孩子及其后代：



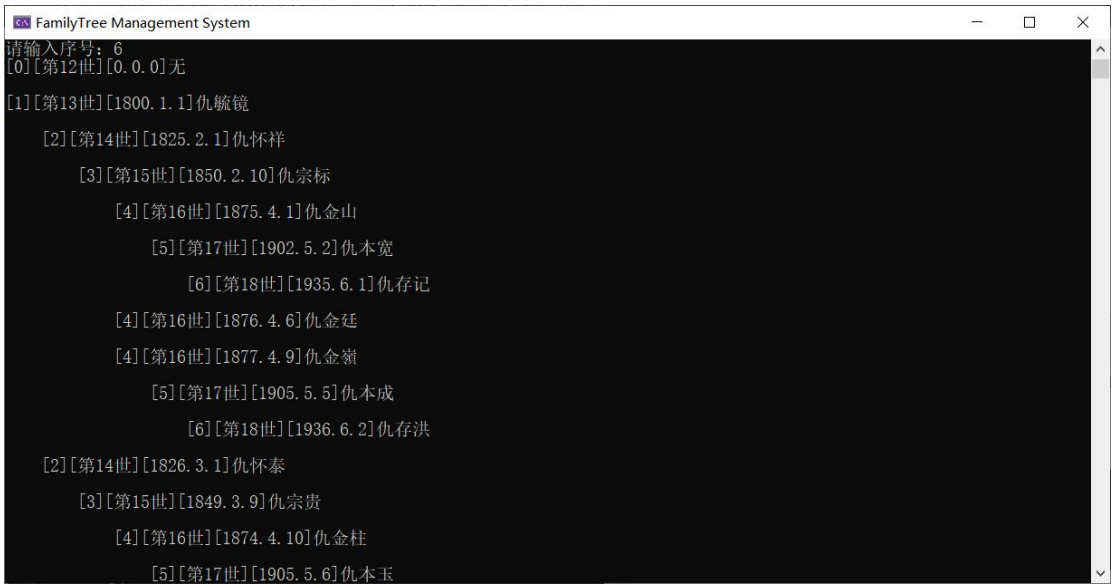
退出查询菜单界面：

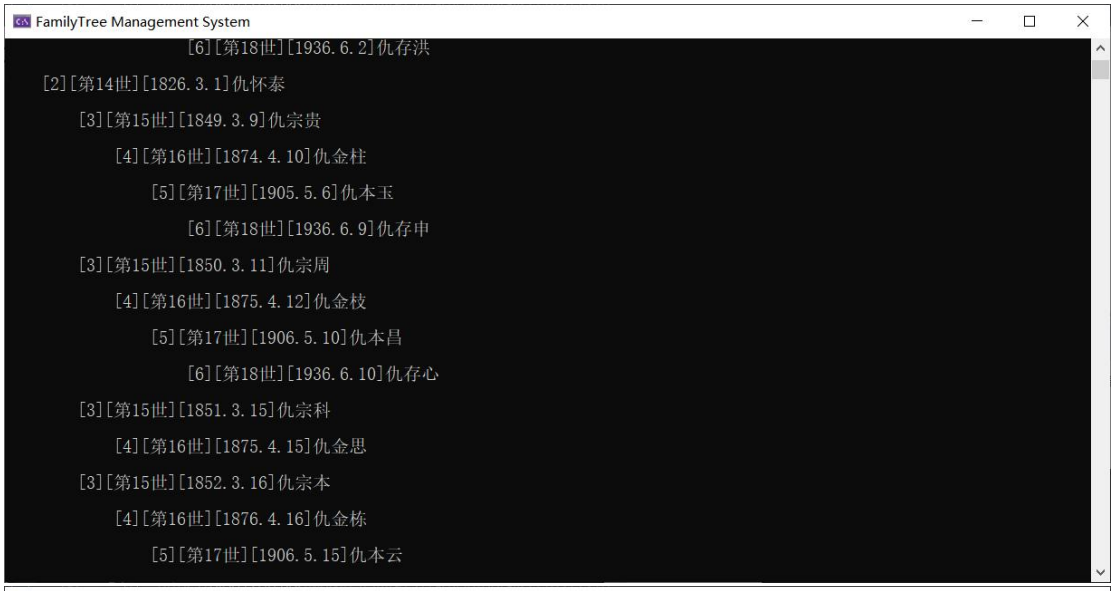


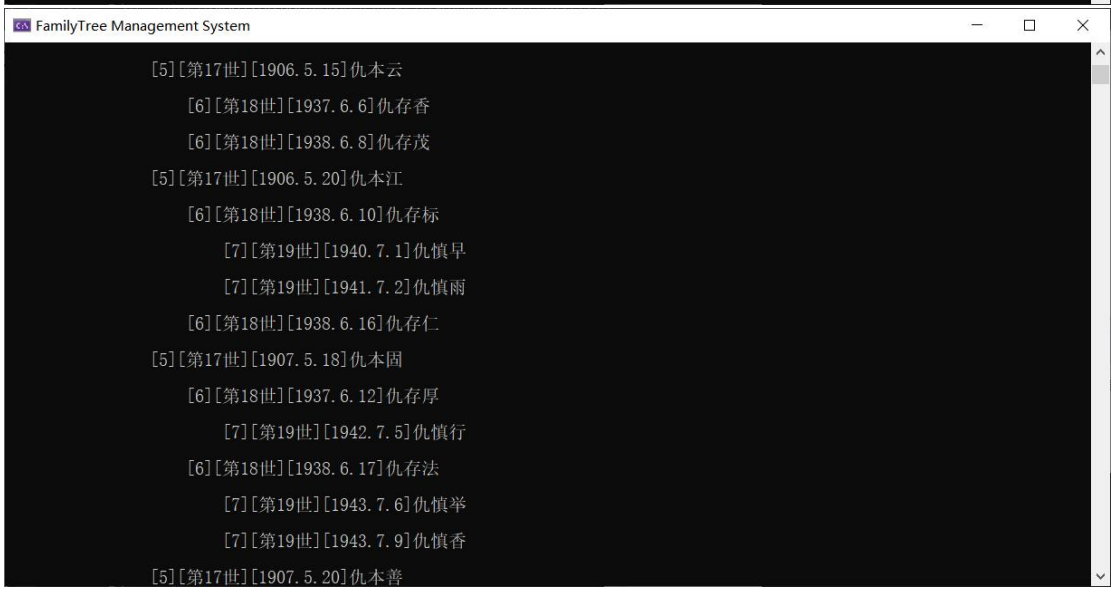


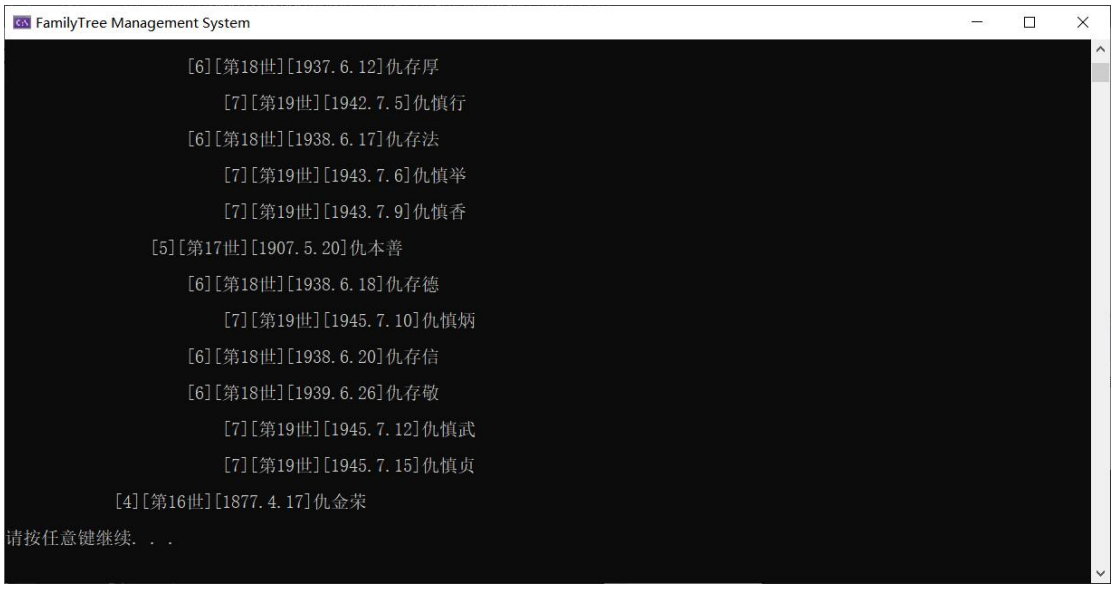
4.8 凹入表方式打印树状家谱：

输入序号 6，可以看到以凹入表形式显示的家谱。


FamilyTree Management System
请输入序号：6
[0][第12世][0.0.0]无
[1][第13世][1800.1.1]仇毓镜
[2][第14世][1825.2.1]仇怀祥
[3][第15世][1850.2.10]仇宗标
[4][第16世][1875.4.1]仇金山
[5][第17世][1902.5.2]仇本宽
[6][第18世][1935.6.1]仇存记
[4][第16世][1876.4.6]仇金廷
[4][第16世][1877.4.9]仇金嶺
[5][第17世][1905.5.5]仇本成
[6][第18世][1936.6.2]仇存洪
[2][第14世][1826.3.1]仇怀泰
[3][第15世][1849.3.9]仇宗贵
[4][第16世][1874.4.10]仇金柱
[5][第17世][1905.5.6]仇本玉

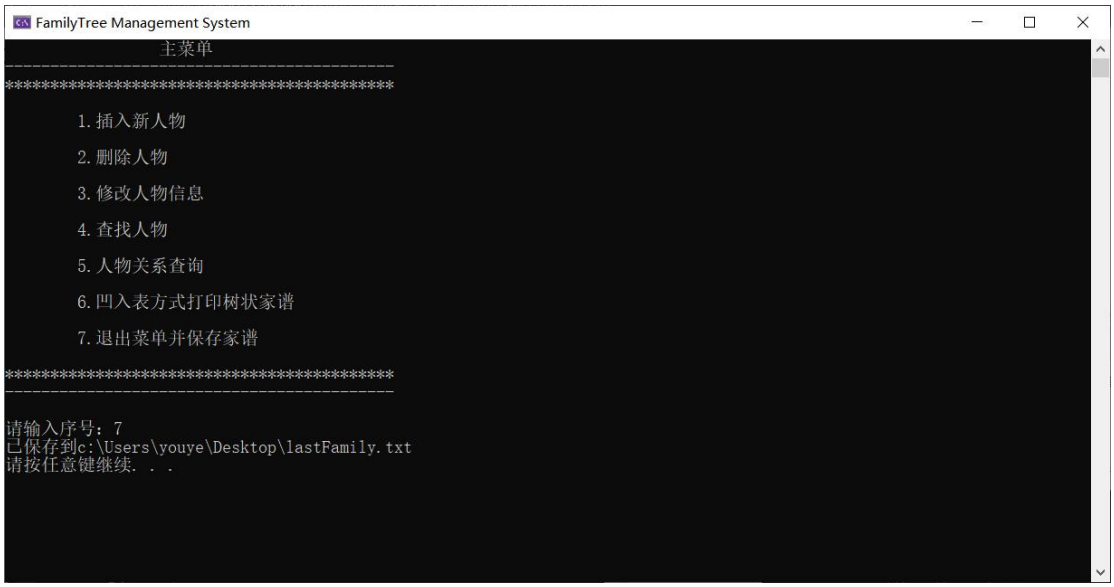

FamilyTree Management System
[6][第18世][1936.6.2]仇存洪
[2][第14世][1826.3.1]仇怀泰
[3][第15世][1849.3.9]仇宗贵
[4][第16世][1874.4.10]仇金柱
[5][第17世][1905.5.6]仇本玉
[6][第18世][1936.6.9]仇存申
[3][第15世][1850.3.11]仇宗周
[4][第16世][1875.4.12]仇金枝
[5][第17世][1906.5.10]仇本昌
[6][第18世][1936.6.10]仇存心
[3][第15世][1851.3.15]仇宗科
[4][第16世][1875.4.15]仇金思
[3][第15世][1852.3.16]仇宗本
[4][第16世][1876.4.16]仇金栋
[5][第17世][1906.5.15]仇本云


FamilyTree Management System
[5][第17世][1906.5.15]仇本云
[6][第18世][1937.6.6]仇存香
[6][第18世][1938.6.8]仇存茂
[5][第17世][1906.5.20]仇本江
[6][第18世][1938.6.10]仇存标
[7][第19世][1940.7.1]仇慎早
[7][第19世][1941.7.2]仇慎雨
[6][第18世][1938.6.16]仇存仁
[5][第17世][1907.5.18]仇本固
[6][第18世][1937.6.12]仇存厚
[7][第19世][1942.7.5]仇慎行
[6][第18世][1938.6.17]仇存法
[7][第19世][1943.7.6]仇慎举
[7][第19世][1943.7.9]仇慎香
[5][第17世][1907.5.20]仇本善



4.9 退出菜单并保存家谱：

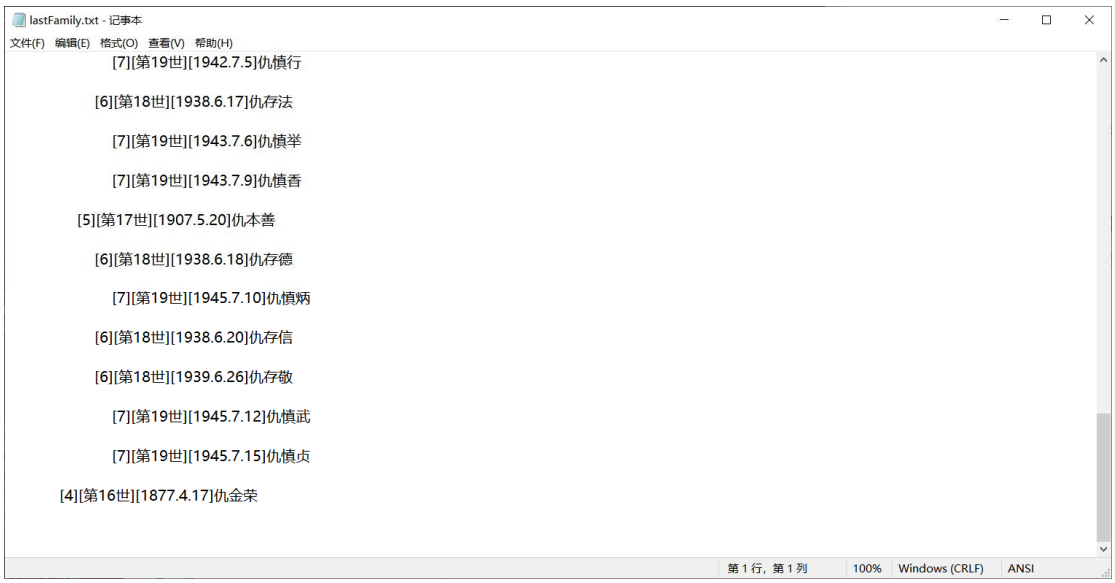
输入序号 7，即自动退出系统菜单程序的同时，会将以上修改完毕的新家谱保存到目标文件 lastFamily.txt 中。




lastFamily.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
[0][第12世][0.0.0]无
[1][第13世][1800.1.1]仇毓镜
[2][第14世][1825.2.1]仇怀祥
[3][第15世][1850.2.10]仇宗标
[4][第16世][1875.4.1]仇金山
[5][第17世][1902.5.2]仇本宽
[6][第18世][1935.6.1]仇存记
[4][第16世][1876.4.6]仇金廷
[4][第16世][1877.4.9]仇金嶺
[5][第17世][1905.5.5]仇本成
[6][第18世][1936.6.2]仇存洪
[2][第14世][1826.3.1]仇怀泰
[3][第15世][1849.3.9]仇宗贵
第 1 行，第 1 列
100%
Windows (CRLF)
ANSI


lastFamily.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
[3][第15世][1849.3.9]仇宗贵
[4][第16世][1874.4.10]仇金柱
[5][第17世][1905.5.6]仇本玉
[6][第18世][1936.6.9]仇存申
[3][第15世][1850.3.11]仇宗周
[4][第16世][1875.4.12]仇金枝
[5][第17世][1906.5.10]仇本昌
[6][第18世][1936.6.10]仇存心
[3][第15世][1851.3.15]仇宗科
[4][第16世][1875.4.15]仇金思
[3][第15世][1852.3.16]仇宗本
[4][第16世][1876.4.16]仇金栋
[5][第17世][1906.5.15]仇本云
第 1 行，第 1 列
100%
Windows (CRLF)
ANSI


lastFamily.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
[5][第17世][1906.5.15]仇本云
[6][第18世][1937.6.6]仇存香
[6][第18世][1938.6.8]仇存茂
[5][第17世][1906.5.20]仇本江
[6][第18世][1938.6.10]仇存标
[7][第19世][1940.7.1]仇慎早
[7][第19世][1941.7.2]仇慎雨
[6][第18世][1938.6.16]仇存仁
[5][第17世][1907.5.18]仇本固
[6][第18世][1937.6.12]仇存厚
[7][第19世][1942.7.5]仇慎行
[6][第18世][1938.6.17]仇存法
[7][第19世][1943.7.6]仇慎举
第 1 行，第 1 列
100%
Windows (CRLF)
ANSI



五、 遇到问题及心得体会

5.1 遇到问题

（说明遇到的问题及解决方法）

①最初选用的是多叉树，但是设计进行到一半发现效率并不高，又重新分析并通过复习和查阅资料，最后选用了孩子兄弟树结构来完成设计。

②在设计读文件内容的同时创建二叉树时，因对于 C++设计语言的不熟练，查阅了一些资料最后才决定选用 C 语言的 fscanf 来读取和赋值。

③因整个程序较大，故数据结构也较多，细节也更需要注意，尤其是指针的运用，因此在设计编写代码的过程中很多地方都标上了注释并且一边编写一边进行运行调试，以达到高效仔细的目的。

④关于家谱系统根据某世代的字输出该世代所有人信息这一功能，暂且使用的是先根遍历及：

```
if (T->people_data.seniority >= 2)
        if ((T->people_data.seniority == A->people_data.seniority) && (T != A))
```

这样的两重逻辑判断条件来代替完成。（另：查询堂兄弟的设计逻辑是父亲的父亲相同的兄弟。）

5.2 心得体会

（说明课设的心得体会）

通过这次课程设计，我深刻体会到：①编程过程应该采用先建框架、逐步求精的方式。应该进行迭代的调试，而不是全部一次集成。分析问题和功能要求，初步设计好框架后再开始分别完善。②在做较大程序过程中应学会边写程序边运行，每完成一个功能就对其调试，这样整个过程会更加高效，同时也能大大避免调试 BUG 的时候不明白该如何改善的麻烦，降低了完善难度。③编写设计代码过程中，同时也要具有良好的编程习惯，结构明确、注释易懂，这样更有利于代码的阅读和维护，同时增加精细度减少 BUG 的产生，也让编写者和使用程序者对操作过程更加明确。④在编写实现目标功能的代码时，应多想想算法是否还有更高效的改进方案，可以在实现功能的基础上尝试减少代码量和冗余度。

六、 带注释的代码

（不要求每一条语句都注释，按功能块，例如循环、判断等进行注释，代码使用小五号字，单倍行距，以节省纸张）

```
//家谱管理系统

#include<iostream>
#include<cstdio>
#include<stdio.h>
#include<cstring>
#include<cstdlib>
#include<fstream>
#include<malloc.h>
#pragma warning(disable:4996)
using namespace std;


//结构体-日期-年月日
typedef struct DATA
{
    int year;
    int month;
    int day;
}DATA;

//结构体-成员信息数据域
typedef struct PeopleInfo
{
    int seniority;  //辈分数 第几世
    int sex;  //性别 1 为男性,0 为女性
```

```
    int isalive;  //目前状况 1为健在,0为身故
    int age;  //年龄
    char name[100];  //姓名
    char fedname[100];  //配偶姓名
    DATA birth_data;  //结构Data定义的出生日期
}PeopleInfo;

//结构体-孩子兄弟树结点
typedef struct CSTNode
{
    PeopleInfo people_data;  //该结点的数据域
    struct CSTNode* first_Child, * next_brother, * father;
    //第一个孩子指针，下一个兄弟指针，父亲指针
}CSTNode, * CSTree, * CSForest;  //树结点，树指针，森林指针

//结构体-队列链表结点
typedef struct LNode
{
    CSTree tree_data;  //链表结点数据域
    struct LNode* next;  //指向下一个的指针域
}LNode, * LinkList;  //链表结点，链表指针

//结构体-枚举类型返回值
typedef enum status
{
    TRUE,  //1
    FALSE,  //0
    OK,  //1
    ERROR,  //0
    SUCCESS,  //0
    //OVERFLOW,  //-2
    EMPTY  //空或0返回1，非空返回0
}Status;


//全局变量
CSForest T;  //根结点指针
CSTree A, B;  //记录search函数返回值的指针
fstream file("c:\\Users\\youye\\Desktop\\lastFamily.txt", ios::out);  //清空txt文件
fstream in("c:\\Users\\youye\\Desktop\\lastFamily.txt");  //打开txt文件


//基本树操作模块函数声明
void InitCSTNode(CSTree& A);  //初始化结点数据函数,避免出现野指针
```

```
void CreateCSTNode(CSTree& A);  //创建结点函数
void Destroy(CSTree A);  //销毁树函数
void DeleteNode(CSTree A);  //删除结点函数
void InsertNode(CSTree& father, CSTree child);  //插入结点函数
void SearchNode(CSForest T, char name[100], CSTree& B);  //查找函数
void ShowPeopleInfo(CSTree A);  //展示成员信息数据域函数
void ChangePeopleInfo(CSTree& B, CSForest T);  //修改成员信息数据域函数
void ChangeBirth(CSTree& A);  //修改成员生日函数
void ChangeFather(CSTree& A, CSForest T);  //修改成员父亲函数
void RefreshSeniority(CSForest& T);  //辈分刷新函数
void InorderCousin(CSTree T, CSTree A);  //输出所有堂兄弟函数
void InorderBro(CSTree T, CSTree A);  //输出所有同世代的兄弟
void PrintASTree(CSTree T);  //凹入表输出家谱函数

Status PrintTree(CSTree t);  //层次遍历输出函数
Status InitQueue(LinkList& L);  //初始化队列
LNode* CreateLNode(CSTNode* p);  //新建结点
Status Enqueue(LNode* p, CSTNode* q);  //元素 q 入队列
Status Dequeue(LNode* p, CSTNode* q);  //出队列，并保存 q 返回值
Status IfEmpty(LinkList L);  //队列判空
void DestroyQueue(LinkList L);  //销毁队列
Status Traverse(CSTree T, LinkList L);  //用队列遍历输出 CSTree


//主程序代码实现模块函数声明
void Init();  //初始化函数，从文件中读取信息自动生成孩子兄弟二叉树
void welcome();  //主菜单函数
void InsertNew();  //插入新人物函数
void Delete();  //删除人物函数
void Change();  //修改人物信息函数
void Search();  //查找人物函数
void Relationship();  //人物关系查询函数
void Save(CSTree T);  //退出菜单保存修改后的文件,凹入表保存家谱



int main()
{
    system("title FamilyTree Management System");  //标题
    Init();  //初始化,读取文件创建树
    welcome();  //主菜单
    system("pause");
}
```

```
//基本树操作模块函数声明

//初始化结点函数,避免出现野指针，参数为树结点指针
void InitCSTNode(CSTree& A)
{
     if (A == NULL)
          return;
     A->father = NULL;
     A->first_Child = NULL;
     A->next_brother = NULL;
     A->people_data.seniority = 0;
     A->people_data.sex = 0;
     A->people_data.isalive = 0;
     A->people_data.age = 0;
     A->people_data.birth_data.year = 0;
     A->people_data.birth_data.month = 0;
     A->people_data.birth_data.day = 0;
     strcpy(A->people_data.name, "无");
     strcpy(A->people_data.fedname, "无");
     return;
}

//创建结点函数，参数为树结点指针
void CreateCSTNode(CSTree& A)
{
     A = new CSTNode;  //分配内存空间
     //(*A)=(CSTree)malloc(sizeof(CSTNode));
     if (A == NULL)
          return;
     InitCSTNode(A);  //初始化结点
}

//销毁树函数(同二叉树一样，递归)，参数为树结点指针
void Destroy(CSTree A)
{
     if (A == NULL)
          return;
     Destroy(A->first_Child);
     Destroy(A->next_brother);
     delete A;
```

```
}

//删除结点函数，参数为树结点指针
void DeleteNode(CSTree A)
{
    CSTree prebro;

    //如果 A 是父亲结点的第一个孩子，则让 A 的兄弟成为父亲结点的第一个孩子
    if (A->father->first_Child == A)
    {
        A->father->first_Child = A->next_brother;
    }
    else  //A 不是父亲结点的第一个孩子，让 A 的前一个兄弟指向 A 的下一个兄弟
    {
        //把 A 的前一个兄弟赋值给 prebro
        for (prebro = A->father->first_Child; prebro->next_brother != A;)
        {
            prebro = prebro->next_brother;
        }
        prebro->next_brother = A->next_brother;
    }
    A->next_brother = NULL;  //释放 A 指向下一个兄弟的指针域
    A->father = NULL;  //释放 A 指向父亲的指针域
    system("pause");
    Destroy(A);  //将 A 的后代全部移除，释放 A 的空间
}

//插入结点函数，参数为父亲结点指针和孩子结点指针
void InsertNode(CSTree& father, CSTree child)
{
    child->people_data.seniority = father->people_data.seniority + 1;  //孩子的代数比父亲大
一
    child->father = father;  //配置插入孩子的父亲指针
    int birth = (child->people_data.birth_data.year) * 10000 +
(child->people_data.birth_data.month) * 100 + (child->people_data.birth_data.day);
    CSTree bro1 = father->first_Child;

    //如果 father 没有 first_Child，直接插入 child
    if (bro1 == NULL)
    {
        father->first_Child = child;
        return;
    }
```

```
    int bro1birth = (bro1->people_data.birth_data.year) * 10000 +
(bro1->people_data.birth_data.month) * 100 + (bro1->people_data.birth_data.day);
    //如果插入孩子的生日早于父亲的第一个孩子，那么插入孩子成为新 first_Child
    if (bro1birth >= birth)
    {
        child->next_brother = bro1;
        father->first_Child = child;
        return;
    }

    //如果插入孩子生日晚于父亲第一个孩子，那么遍历兄弟链表找到插入位置
    CSTree bro2 = father->first_Child;
    int bro2birth = (bro2->people_data.birth_data.year) * 10000 +
(bro2->people_data.birth_data.month) * 100 + (bro2->people_data.birth_data.day);
    bro1 = bro2->next_brother;
    //找到 bro2 生日早于 child 生日早于 bro1 生日的位置
    for (; (bro1 != NULL) && (birth > bro1birth);)
    {
        bro2 = bro2->next_brother;
        bro1 = bro2->next_brother;
    }
    child->next_brother = bro1;
    bro2->next_brother = child;

    return;
}

//先序遍历查找函数，T 为树的根结点，name 为需要找的人，B 储存找到的结点位置
void SearchNode(CSForest T, char name[100], CSTree& B)
{
    if (T == NULL)
        return;
    if (strcmp(T->people_data.name, name) == 0)  //如果 T 的 name 与查找目标相同
    {
        B = T;
        return;
    }
    SearchNode(T->first_Child, name, B);
    if (B != NULL)
        return;
    SearchNode(T->next_brother, name, B);
}

//展示成员信息数据域函数
```

```
void ShowPeopleInfo(CSTree A)
{
    if (A == NULL)
        return;
    if (A->people_data.seniority == 0)  //A 的辈分为 0，是根结点
    {
        cout << "该结点为森林根结点，禁止操作！\n";
        system("pause");
        return;
    }
    cout << "姓名：" << A->people_data.name << "  ";
    if (A)
        cout << "性别：男  ";
    else
        cout << "性别：女  ";
    cout << "配偶姓名：" << A->people_data.fedname << "  ";
    cout << "辈分：" << A->people_data.seniority << "  ";
    cout << "第" << (A->people_data.seniority) + 12 << "世  ";
    cout <<"生日：" <<A->people_data.birth_data.year <<"." <<A->people_data.birth_data.month
<< "." << A->people_data.birth_data.day << "  ";
    if (A->people_data.isalive == 1)
        cout << "目前状况：健在  ";
    else
        cout << "目前状况：身故  ";

    if (A->people_data.seniority != 1)
        cout << "父亲姓名：" << A->father->people_data.name << "  ";
    else
        cout << "为家族祖先" << endl;

    cout << endl;

}

//修改成员信息数据域函数，B 为欲修改的结点，T 为根结点
void ChangePeopleInfo(CSTree& B, CSForest T)
{
    if (B == NULL)
        return;
    if (B->people_data.seniority == 0)  //B 的辈分为 0，是根结点
    {
        cout << "该结点为森林根结点，禁止操作！\n";
        system("pause");
        return;
```

```
    }
    int choice;
W:system("cls");
    ShowPeopleInfo(B);  //列出B结点当前具体信息
    cout << endl;
    cout << "请选择想要修改的信息序号：" << endl << endl;
    cout << "************************" << endl << endl;
    cout << "    1.name" << endl;
    cout << "    2.sex" << endl;
    cout << "    3.fedname" << endl;
    cout << "    4.birth" << endl;
    cout << "    5.father" << endl;
    cout << "    6.isalive" << endl;
    cout << "    7.break" << endl << endl;
    cout << "************************" << endl << endl;
    cout << "请输入序号：";
    //fflush(stdin); //cin.clear;
    cin >> choice;

    switch (choice)
    {
    case 1:  //修改name
        cout << "请输入新的姓名：";
        char name[100];
        cin >> name;
        strcpy(B->people_data.name, name);
        system("pause");
        goto W;
    case 2:  //修改sex
        cout << "你的性别为：（男性请输入1，女性请输入2）";
        int sex;
        cin >> sex;
        if ((sex != 2) && (sex != 1))
        {
            cout << "输入有误" << endl;
            system("pause");
            goto W;
        }
        B->people_data.sex = sex;
        system("pause");
        goto W;
    case 3:  //修改fedname
        cout << "请输入其配偶的新姓名：";
        char fedname[100];
```

```
            cin >> fedname;
            strcpy(B->people_data.fedname, fedname);
            system("pause");
            goto W;
        case 4:  //修改 birth
            ChangeBirth(B);  //修改成员生日函数
            system("pause");
            goto W;
        case 5:  //修改 father
            ChangeFather(B, T);  //修改成员父亲函数
            system("pause");
            goto W;
        case 6:  //修改 isalive
            if (B->people_data.isalive)
                B->people_data.isalive = 0;
            else
                B->people_data.isalive = 1;
            system("pause");
            goto W;
        case 7:  //结束修改成员信息模块
            Save(T);
            break;
        default:
            goto W;
        }

        return;

}

//修改成员生日函数
void ChangeBirth(CSTree& A)
{
        cout << "生日修改为：（输入格式：代表年份的四位整数 代表月份的 1-12 整数 代表日期的 0-31
整数）";
        int year, month, day;
        cin >> year >> month >> day;
        if ((year < 1) || (year > 10000) || (month < 1) || (month > 12) || (day < 1) || (day > 31))
        {
            cout << "输入有误" << endl;
            system("pause");
            return;
        }
        int Abirth = year * 10000 + month * 100 + day;
```

```
    int fatherbirth = (A->father->people_data.birth_data.year) * 10000 +
(A->father->people_data.birth_data.month) * 100 + (A->father->people_data.birth_data.day);
    //如果 A 的父亲存在且 A 的父亲的生日晚于 A
    if ((A->father->people_data.seniority != 0) && (fatherbirth >= Abirth))
    {
        cout << "生日不能比父亲早" << endl;
        return;
    }
    A->people_data.birth_data.year = year;
    A->people_data.birth_data.month = month;
    A->people_data.birth_data.day = day;

    CSTree B;
    CSTree C;
    C = A->father;
    //修改生日后需要重新排序兄弟结点，可先把该结点分离出来
    if (A->father->first_Child == A)
    {
        A->father->first_Child = A->next_brother;
        A->next_brother = NULL;
        A->father = NULL;
    }
    else
    {
        for (B = A->father->first_Child; B->next_brother != A; B = B->next_brother);
        B->next_brother = A->next_brother;
        A->next_brother = NULL;
        A->father = NULL;
    }
    //再调用含自动排序功能的 InsertNode 函数插入回二叉树内
    InsertNode(C, A);
}

//修改成员父亲函数
void ChangeFather(CSTree& A, CSForest T)
{
    cout << "父亲修改为：" << endl;
    int Abirth = (A->people_data.birth_data.year) * 10000 + (A->people_data.birth_data.month)
* 100 + (A->people_data.birth_data.day);
    char fathername[100];
    cin >> fathername;

    //调用 SearchNode 函数找到新的父亲结点
    CSTree newfather = NULL;
```

```
    SearchNode(T, fathername, newfather);
    ShowPeopleInfo(newfather);  //显示新父亲结点的详细信息

    //查看新父亲结点是否存在
    if (newfather == NULL)
    {
        cout << "该成员不存在" << endl;
        system("pause");
        return;
    }
    system("pause");

    //查看新父亲结点生日是否早于孩子
    int fatherbirth = (newfather->people_data.birth_data.year) * 10000 +
(newfather->people_data.birth_data.month) * 100 + (newfather->people_data.birth_data.day);
    if ((newfather->people_data.seniority != 0) && (fatherbirth >= Abirth))
    {
        cout << "生日不能早于父亲" << endl;
        return;
    }

    //将欲修改的 A 结点分离出来插入到新父亲结点下
    CSTree B;
    CSTree C;
    C = A->father;
    if (A->father->first_Child == A)
    {
        A->father->first_Child = A->next_brother;
        A->next_brother = NULL;
        A->father = NULL;
    }
    else
    {
        B = A->father->first_Child;
        for (; B->next_brother != A; B = B->next_brother);
        B->next_brother = A->next_brother;
        A->next_brother = NULL;
        A->father = NULL;
    }

    InsertNode(newfather, A);  //将 A 结点插入新父亲结点下
    RefreshSeniority(T);  //刷新辈分
}
```

```
//辈分刷新函数，利用先序遍历思想递归
void RefreshSeniority(CSForest& T)
{
    if (T == NULL)
        return;
    if (T->father == NULL)
        T->people_data.seniority = 0;
    else
    {
        T->people_data.seniority = T->father->people_data.seniority + 1;
    }
    RefreshSeniority(T->first_Child);
    RefreshSeniority(T->next_brother);
    return;
}

//输出所有堂兄堂弟
void InorderCousin(CSTree T, CSTree A)
{
    if (T == NULL)
        return;
    //从根结点开始遍历，如果找到与 A 结点父亲的父亲相同，即与 A 互为堂兄弟，输出
    if (T->people_data.seniority >= 3)
        if ((T->father->father == A->father->father) && (T != A))
            cout << "[" << T->people_data.seniority << "][第" << T->people_data.seniority +
12 << "世][" << T->people_data.birth_data.year << "." << T->people_data.birth_data.month << "."
<< T->people_data.birth_data.day << "]" << T->people_data.name << endl;
    InorderCousin(T->first_Child, A);
    InorderCousin(T->next_brother, A);
}

//输出所有同世代的兄弟
void InorderBro(CSTree T, CSTree A)
{
    if (T == NULL)
        return;
    //从根结点开始遍历，找到与 A 结点辈数相同的兄弟输出
    if (T->people_data.seniority >= 2)
        if ((T->people_data.seniority == A->people_data.seniority) && (T != A))
            cout << "[" << T->people_data.seniority << "][第" << T->people_data.seniority +
12 << "世][" << T->people_data.birth_data.year << "." << T->people_data.birth_data.month << "."
<< T->people_data.birth_data.day << "]" << T->people_data.name << endl;
    InorderBro(T->first_Child, A);
    InorderBro(T->next_brother, A);
```

```
}


//凹入表输出家谱，先根遍历（即二叉树存储的先序遍历）
void PrintASTree(CSTree T)
{
    int cnt;
    if (T)
    {
        //输出空格
        for (cnt = 1; cnt < T->people_data.seniority; cnt++)
            cout << "    ";
        //输出字符
        cout << "[" << T->people_data.seniority << "][第" << T->people_data.seniority + 12 <<
"世][" << T->people_data.birth_data.year << "." << T->people_data.birth_data.month << "." <<
T->people_data.birth_data.day << "]" << T->people_data.name << endl;
        cout << endl;
        PrintASTree(T->first_Child);
        PrintASTree(T->next_brother);
    }
   }



//关于队列

//层次遍历输出函数
Status PrintTree(CSTree t)
{
    LinkList L;
    if (t == NULL)
    {
        cout << "森林不存在" << endl;
        return OK;
    }
    InitQueue(L);  //初始化队列
    Traverse(t->first_Child, L);  //利用队列输出
    DestroyQueue(L);  //销毁队列
    return OK;
}


//队列相关操作
```

```
//初始化队列
Status InitQueue(LinkList& L)
{
    L = new LNode;  //分配结点空间
    if (L == NULL)  //分配失败
    {
        return ERROR;
    }
    L->next = NULL;
    return OK;
}

//新建结点
LNode* CreateLNode(CSTNode* p)
{
    LNode* q;
    q = new LNode;  //分配结点空间
    if (q != NULL)  //分配成功
    {
        q->tree_data = p;
        q->next = NULL;
    }
    return q;
}

//元素 q 入队列
Status Enqueue(LNode* p, CSTNode* q)
{
    if (p == NULL)
        return ERROR;
    while (p->next != NULL)  //调至队列最后
        p = p->next;
    p->next = CreateLNode(q);  //生成 q 的队列结点让 q 进入队列
    return OK;
}

//出队列，并保存 q 返回值
Status Dequeue(LNode* p, CSTNode* q)
{
    LNode* aq;
    if (p == NULL || p->next == NULL)  //出队列的位置不合理
        return ERROR;
    aq = p->next;  //修改被出队结点 aq 的指针域
```

```
    p->next = aq->next;  //让 p 的下一个指针域指向 p->next->next
    q = aq->tree_data;  //用 q 保存出队 aq 的数据域
    delete aq;  //释放结点 aq
    return OK;
}

//队列判空
Status IfEmpty(LinkList L)
{
    if (L == NULL)  //队列不存在
        return ERROR;
    if (L->next == NULL)  //队列为空
        return TRUE;
    return FALSE;  //队列非空
}

//销毁队列
void DestroyQueue(LinkList L)
{
    LinkList p;
    if (L != NULL)
    {
        p = L;
        L = L->next;
        delete p;  //逐一释放
        DestroyQueue(L);  //递归销毁
    }
}

//用队列遍历输出 CSTree
Status Traverse(CSTree T, LinkList L)
{
    CSTree P = T;
    CSTree K;
    int i = 0;
    int j = 1;
    while (P)  //p 指向森林里的每棵树
    {
        cout << "第" << j << "个家庭" << endl;
        K = P;  //利用 K 来遍历以 P 为根结点的子树中的结点
        Enqueue(L, K);  //根结点入队
        while (IfEmpty(L) == FALSE)  //只要队列不为空就依次出队直到队空
        {
            Dequeue(L, K);
```

```
            if (i != K->people_data.seniority)  //将 K 的辈数赋给 i
            {
                cout << endl;
                i = K->people_data.seniority;
            }
            cout << "[" << K->people_data.seniority << "][第" << K->people_data.seniority +
12 << "世][" << K->people_data.birth_data.year << "." << K->people_data.birth_data.month << "."
<< K->people_data.birth_data.day << "]" << K->people_data.name << endl;

            if (K->first_Child)  //如果该结点不是森林中的叶节点则进入下一层即孩子层
            {
                K = K->first_Child;
                Enqueue(L, K);
                while (K->next_brother)  //入队同一层的兄弟结点
                {
                    K = K->next_brother;
                    Enqueue(L, K);
                }
            }

        }
        P = P->next_brother;  //递归
        cout << endl;
        j++;
    }
    return OK;
}
```

```
//主程序代码实现模块函数

//初始化函数，从文件中读取信息自动生成孩子兄弟二叉树
void Init()
{
    CreateCSTNode(T);  //创建根结点并初始化，此时 T 的 name 为“无”

    FILE* r;
    r = fopen("C:\\Users\\youye\\Desktop\\Family.txt", "r");  //打开文件夹
    if (r == NULL)
    {
        cout << "打开文件失败！" << endl;
```

```
    }

    A = NULL;
    B = NULL;
    char FatherName[100], tmp1[2], tmp2[2];

    CreateCSTNode(A);  //创建 A 临时结点

    cout << "读取文件中..." << endl;

    //读取文件直到循环结束
    while (fscanf(r, "%s %d %s %d%s%d%s%d %d %s", &A->people_data.name, &A->people_data.sex,
&A->people_data.fedname, &A->people_data.birth_data.year, tmp1,
&A->people_data.birth_data.month, tmp2, &A->people_data.birth_data.day,
&A->people_data.isalive, FatherName) != EOF)
    {
        cout << A->people_data.name << " " << A->people_data.sex << "     " <<
A->people_data.fedname << " " << A->people_data.birth_data.year << tmp1 <<
A->people_data.birth_data.month << tmp2 << A->people_data.birth_data.day << "  " <<
A->people_data.isalive << " " << FatherName;
        cout << endl << "Search..." << endl;
        SearchNode(T, FatherName, B);  //查找 FatherName 的结点储存到 B 临时父亲结点中
        //如果父亲结点存在就将 A 孩子结点插入其下，如果父亲结点为无就作为根结点的子节点存在

        if (B == NULL)
        {
            cout << "此人的父亲不在家谱里！" << endl;
            delete A;
        }
        int Abirth = (A->people_data.birth_data.year) * 10000 +
(A->people_data.birth_data.month) * 100 + (A->people_data.birth_data.day);
        int Bbirth = (B->people_data.birth_data.year) * 10000 +
(B->people_data.birth_data.month) * 100 + (B->people_data.birth_data.day);
        //如果 B 作为临时父亲结点存在且不是根结点且生日晚于 A
        if ((B->people_data.seniority != 0) && (Bbirth >= Abirth))
        {
            cout << "孩子生日不能比父亲早" << endl;
            delete A;
        }
        else
        {
            cout << "Insert..." << endl;
            InsertNode(B, A);  //存在父亲结点，将 A 插入父亲结点下
        }
```

```
            B = NULL;
            A = NULL;
            CreateCSTNode(A);  //继续循环
        }
        delete A;
        A = NULL;
        fclose(r);
       }



//主菜单函数
void welcome()
{
     int choice;
     system("pause");
Q:system("cls");  //清屏
     cout << "                  主菜单" << endl;
     cout << "----------------------------------------" << endl;
     cout << "****************************************" << endl;
     cout << endl;
     cout << "1.插入新人物" << endl << endl;
     cout << "2.删除人物" << endl << endl;
     cout << "3.修改人物信息" << endl << endl;
     cout << "4.查找人物" << endl << endl;
     cout << "5.人物关系查询" << endl << endl;
     cout << "6.凹入表方式打印树状家谱" << endl << endl;
     cout << "7.退出菜单并保存家谱" << endl << endl;
     cout << "****************************************" << endl;
     cout << "----------------------------------------" << endl << endl;
     cout << "请输入序号：";
     fflush(stdin);
     cin >> choice;

     switch (choice)
     {
     case 1:  //插入新人物
          InsertNew();
          system("pause");
          goto Q;  //从 A 处继续循环
     case 2:  //删除人物
          Delete();
          system("pause");
```

```
            goto Q;
        case 3:  //修改人物信息
            Change();
            system("pause");
            goto Q;
        case 4:  //查找人物
            Search();
            system("pause");
            goto Q;
        case 5:  //人物关系查询
            Relationship();
            system("pause");
            goto Q;
        case 6:  //凹入表方式打印树状家谱
            PrintASTree(T);
            system("pause");
            goto Q;
        case 7:  //退出菜单并保存家谱
            Save(T);
            cout << "已保存到 c:\\Users\\youye\\Desktop\\lastFamily.txt" << endl;
            break;
        default:
            goto Q;
        }
        return;
}


//插入新人物函数
void InsertNew()
{
        char FatherName[100];
        A = NULL;  //使全局变量 A 和 B 都为空
        B = NULL;
        CreateCSTNode(A);  //创建临时结点 A
        cout << "请输入：姓名        性别（男为 1/女为 2）    配偶姓名  出生日期  状态(健在为 1/身故为 0）    父亲姓名" << endl;
        cout << "例如：张三    1    无    2002 01 01    1    张大" << endl;
        cin >> A->people_data.name >> A->people_data.sex >> A->people_data.fedname >> A->people_data.birth_data.year >> A->people_data.birth_data.month >> A->people_data.birth_data.day >> A->people_data.isalive >> FatherName;
        cout << A->people_data.name << " " << A->people_data.sex << "     " << A->people_data.fedname << " " << A->people_data.birth_data.year << "." << A->people_data.birth_data.month << "." << A->people_data.birth_data.day << "    " <<
```

```
A->people_data.isalive << "    " << FatherName << endl;

    if ((A->people_data.sex != 1) && (A->people_data.sex != 0))
    {
        cout << "性别输入有误！" << endl;
        delete A;
        A = NULL;
        return;
    }
    if ((A->people_data.isalive != 1) && (A->people_data.isalive != 0))
    {
        cout << "状态输入有误！" << endl;
        delete A;
        A = NULL;
        return;
    }

    SearchNode(T, FatherName, B);  //查找 A 的父亲结点并赋值到 B 结点中
    if (B == NULL)
    {
        cout << "此人的父亲不在家谱里！" << endl;
        delete A;
        A = NULL;
        return;
    }
    int Abirth = (A->people_data.birth_data.year) * 10000 + (A->people_data.birth_data.month)
* 100 + (A->people_data.birth_data.day);
    int Bbirth = (B->people_data.birth_data.year) * 10000 + (B->people_data.birth_data.month)
* 100 + (B->people_data.birth_data.day);
    //如果 B 作为临时父亲结点存在且不是根结点且生日晚于 A
    if ((B->people_data.seniority != 0) && (Bbirth >= Abirth))
    {
        cout << "孩子生日不能比父亲早" << endl;
        delete A;
        A = NULL;
        B = NULL;
        return;
    }

    InsertNode(B, A);  //在父亲结点下插入 A
    cout << "插入成功！" << endl;
    A = NULL;  //令全局变量 A 恢复为空
    return;
}
```

```
//删除人物函数
void Delete()
{
    A = NULL;  //全局变量 A 为空
    char name[100];
    cout << "请输入要删除的人物姓名：";
    cin >> name;
    cout << endl;
    SearchNode(T, name, A);  //根据名字查找目标并存到 A 结点中
    if (A == NULL)
    {
        cout << "不存在姓名为" << name << "的人";
        return;
    }
    if (A->first_Child != NULL)
    {
        cout << "此人有后代，不允许删除！";
        return;
    }
    DeleteNode(A);
    cout << "删除成功！" << endl;
    A = NULL;
}


//修改人物信息
void Change()
{
    A = NULL;
    char name[100];
    cout << "请输入要修改信息的人物姓名：";
    cin >> name;
    SearchNode(T, name, A);  //根据名字查找目标并存在 A 结点中
    if (A == NULL)
    {
        cout << "不存在名字为" << name << "的人" << endl;
        return;
    }
    ChangePeopleInfo(A, T);  //进入修改信息界面
    A = NULL;  //令全局变量 A 为空
}
```

```
//查找人物
void Search()
{
    A = NULL;
    char name[100];
    cout << "请输入要查找的人物姓名：";
    cin >> name;
    SearchNode(T, name, A);  //根据名字查找目标并存在 A 结点中
    if (A == NULL)
    {
        cout << "不存在名字为" << name << "的人" << endl;
        return;
    }
    ShowPeopleInfo(A);  //列出目标信息
    A = NULL;  //令全局变量 A 为空
}


//人物关系查询
void Relationship()
{
    A = NULL;
    B = NULL;  //令全局变量都为 NULL
    char name[100];
    cout << "请输入需要查询关系的人物姓名：";
    cin >> name;
    SearchNode(T, name, B);  //查询目标并将其保存到 B 结点中
    if (B == NULL)
    {
        cout << "不存在名字为" << name << "的人" << endl;
        return;
    }

    int choice;
C:system("cls");  //清屏
    ShowPeopleInfo(B);  //列出目标人物的具体信息
    cout << endl;

    cout << "          人物关系查询菜单" << endl << endl;
    cout << "****************************************" << endl << endl;
    cout << "1.祖先" << endl;
    cout << "2.堂兄弟" << endl;
    cout << "3.同世代的兄弟" << endl;
```

```
    cout << "4.后代" << endl;
    cout << "5.break" << endl << endl;
    cout << "*****************************************" << endl << endl;
    cout << "请输入序号：";
    cin >> choice;

    switch (choice)
    {
    case 1:  //查询祖先
        A = B->father;
        //只要 A 不等于根结点，遍历 A 的父亲
        for (; A->people_data.seniority != 0; A = A->father)
        {
            cout << "[" << A->people_data.seniority << "][第" << A->people_data.seniority +
12 << "世][" << A->people_data.birth_data.year << "." << A->people_data.birth_data.month << "."
<< A->people_data.birth_data.day << "]" << A->people_data.name << endl;
        }
        system("pause");
        goto C;  //返回查询菜单继续循环
    case 2:  //查询堂兄弟
        InorderCousin(T, B);
        system("pause");
        goto C;
    case 3:  //查询同世代的兄弟
        InorderBro(T, B);
        system("pause");
        goto C;
    case 4:  //查询后代关系
        PrintTree(B);  //层次遍历
        system("pause");
        goto C;
    case 5:  //退出查询菜单
        break;
    default:
        goto C;
    }
    B = NULL;
    A = NULL;
    return;
}


//退出菜单保存修改后的文件,凹入表保存家谱
void Save(CSTree T)
```

```
{
    int cnt;
    if (T)
    {
        //输出空格
        for (cnt = 1; cnt < T->people_data.seniority; cnt++)
            in << "    ";
        //输出字符
        in << "[" << T->people_data.seniority << "][第" << T->people_data.seniority + 12 <<
"世][" << T->people_data.birth_data.year << "." << T->people_data.birth_data.month << "." <<
T->people_data.birth_data.day << "]" << T->people_data.name << endl;
        in << endl;
        Save(T->first_Child);
        Save(T->next_brother);
    }
   }
```