****

项目说明文档

**数据结构课程设计**

**——家谱管理系统**

**培养单位：软件学院**

**本 科 生：蓝 笙 聆**

**学 号：1951096**

**指导老师：张 颖**

二○二○年十二月

目录

[第1章 分析 1](#_Toc58809014)

[1.1 背景分析 1](#_Toc58809015)

[1.2 功能分析 1](#_Toc58809016)

[第2章 设计 2](#_Toc58809017)

[2.1 数据结构设计 2](#_Toc58809018)

[2.2 类结构设计 2](#_Toc58809019)

[2.3 成员与操作设计 2](#_Toc58809020)

[2.4 系统设计 3](#_Toc58809021)

[第3章 实现 4](#_Toc58809022)

[3.1 插入功能的实现 4](#_Toc58809023)

[3.1.1 插入功能流程图 4](#_Toc58809024)

[3.1.2 插入功能核心代码 4](#_Toc58809025)

[3.1.3 插入功能截屏示例 6](#_Toc58809026)

[3.2 删除功能的实现 7](#_Toc58809027)

[3.2.1 删除功能流程图 7](#_Toc58809028)

[3.2.2 删除代码实现 7](#_Toc58809029)

[3.2.3 删除功能截屏示例 8](#_Toc58809030)

[3.3 查找功能的实现 9](#_Toc58809031)

[3.3.1 查找功能流程图 9](#_Toc58809032)

[3.3.2 查找代码实现 9](#_Toc58809033)

[3.3.2 查找功能截屏示例 11](#_Toc58809034)

[3.4 修改功能的实现 11](#_Toc58809035)

[3.4.1 修改功能流程图 11](#_Toc58809036)

[3.4.2 修改代码实现 11](#_Toc58809037)

[3.4.3 修改功能截屏示例 12](#_Toc58809038)

[3.5 总体系统的实现 13](#_Toc58809039)

[3.5.1 总体系统流程图 13](#_Toc58809040)

[3.5.2 总体系统代码实现 13](#_Toc58809041)

[3.5.3 总体功能截屏示例 15](#_Toc58809042)

[第4章 测试 16](#_Toc58809043)

[4.1 功能测试 16](#_Toc58809044)

[4.1.1 插入功能测试 16](#_Toc58809045)

[4.1.2 删除功能测试 16](#_Toc58809046)

[4.1.3 修改功能测试 16](#_Toc58809047)

[4.1.4 搜索功能测试 17](#_Toc58809048)

[4.2 出错测试 17](#_Toc58809049)

[4.3.1 删除树根节点 17](#_Toc58809050)

[4.3.2 操作码错误 17](#_Toc58809051)

[4.3.3 插入位置不存在 18](#_Toc58809052)

[4.3.4 插入元素已存在 18](#_Toc58809053)

[4.3.5 删除元素不存在 19](#_Toc58809054)

[4.3.6 查找元素不存在 19](#_Toc58809055)

[4.3.7 修改元素不存在 19](#_Toc58809056)

[4.3.8 修改后元素已存在 19](#_Toc58809057)

# 第1章 分析

## 1.1 背景分析

家谱是一种以表谱形式，记载一个以血缘关系为主体的家族世袭繁衍和重要任务事迹的特殊图书体裁。家谱是中国特有的文化遗产，是中华民族的三大文献（国史，地志，族谱）之一，属于珍贵的人文资料，对于历史学，民俗学，人口学，社会学和经济学的深入研究，均有其不可替代的独特功能。本项目兑对家谱管理进行简单的模拟，以实现查看祖先和子孙个人信息，插入家族成员，删除家族成员的功能。

随着计算机科学技术的不断成熟，使用计算机对家谱系统进行管理，具有手工管理所无法比拟的优势。这些优点能够极大地提高效率，也是家族走向信息化、科学化、国际化的重要条件。因此，开发一套家谱管理系统具有十分重要的意义。

## 1.2 功能分析

作为一个最简易的家谱管理系统，首先应该有的功能就是输入家族信息并且可以予以显示。其次，家谱管理系统还应该具有插入、删除、修改功能，以保证家族可以随时更改自己的家铺信息。最后，家谱管理系统软件还应该确保软件可以正常关闭。

综上所述，一个家谱管理系统至少应该具有输入、输出、插入、删除、修改、退出的功能。

# 第2章 设计

## 2.1 数据结构设计

如上功能分析所述，该系统要求大量的增加、删除、修改操作，且为树状结构，因此考虑使用树数据结构。同时，为了实现简易，使用vector将树的节点包装，使得程序简洁。

## 2.2 类结构设计

经典的树一般包括两个抽象数据类型（ADT）——树结点类（TreeNode）与树类（Tree），而两个类之间的耦合关系可以采用嵌套、继承等多种关系。为方便处理，本系统采用模板struct结构体描述树结点类（TreeNode <T>），这样使得树类（Tree <T>）可以访问链表结点。

## 2.3 成员与操作设计

**树节点结构体（TreeNode）**

template <class T>

struct TreeNode {

    T \_val;                       //节点的值

    TreeNode\* \_prev;              //上一个节点

    vector<TreeNode\*> \_children;  //所有的孩子节点

    TreeNode();

    TreeNode(T x);

    ~TreeNode();

};

**树类（Node）**

template <class T>

class Tree {

   private:

    TreeNode<T>\* \_root;

   public:

    TreeNode<T>\* GetRoot() const { return this->\_root; }  //返回根节点

    TreeNode<T>\* Search(const T val);                  //搜索是否存在val

    void InsChild(const T node, const T val);          //插入一个孩子

    void InsChild(const T node, const vector<T> val);  //插入多个孩子

    void Changename(const T prev, const T cur);  //对存在的孩子修改他的名字

    void Del(const T val);  //对存在的孩子进行删除

    void PrintChildren(ostream& os, const T p);  //输出这个孩子

    TreeNode<T>\* Bfs(queue<TreeNode<T>\*> tree,

                     const T val);  //使用广度优先搜索进行搜索

    Tree();

    Tree(T x);

};

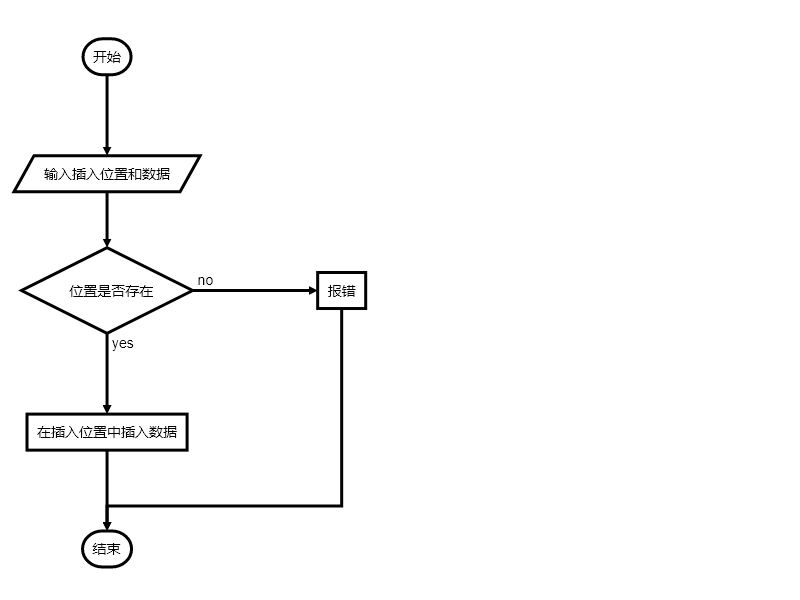
## 2.4 系统设计

系统首先调用opening ()函数实现对屏幕的初始化，创建树family，然后进入loop ()循环，根据用户所输入的操作码（operatorCode）执行树family对应的成员函数。

# 第3章 实现

## 3.1 插入功能的实现

### 3.1.1 插入功能流程图



### 3.1.2 插入功能核心代码

template <class T>

void Tree<T>::InsChild(const T par, const vector<T> children) {

    TreeNode<T>\* node = this->Search(par);

    if (node == nullptr) throw "Error: Data not found. ";

    for (auto&& i : children) {

        auto child = new TreeNode<T>(i);

        child->\_prev = node;

        node->\_children.push\_back(child);

    }

}

template <class T>

void Tree<T>::Del(const T val) {

    TreeNode<T>\* node = this->Search(val);

    if (node == nullptr) throw "Error: Data not found. ";

    TreeNode<T>\* parent = node->\_prev;

    for (int i = 0; i < parent->\_children.size(); i++) {

        if (parent->\_children[i]->\_val == val) {

            auto v = parent->\_children;

            parent->\_children.clear();

            for (auto&& i : v)

                if (i->\_val != val) parent->\_children.push\_back(i);

            delete v[i];

        }

    }

}

void addChildren(Tree<string>& family) {

    int num = 0;

    string temps = "";

    string p = "";

    string child = "";

    vector<string> children = {};

    cout << "Please enter the name of the person who will establish a "

            "family:";

    cin >> p;

    if (family.Search(p) == nullptr)

        throw "Error: Data not found. Please try again. ";

    cout << "Please enter the number of children of " << p << ".";

    cin >> temps;

    if (isNum(temps))

        num = str2num(temps);

    else

        throw "Error: Invalid input detected. Please try again. ";

    cout << "Please enter the name of children of " << p << " in turn.";

    for (int i = 0; i < num; i++) {

        cin >> child;

        if (family.Search(child) != nullptr)

            throw "Error: Already in the tree. Please try again. ";

        children.push\_back(child);

    }

    family.InsChild(p, children);

    family.PrintChildren(cout, p);

    return;

}

void addAChild(Tree<string>& family) {

    string child = "";

    string p = "";

    cout << "Please enter the name of the person you want to add the "

            "child to:";

    cin >> p;

    if (family.Search(p) == nullptr)

        throw "Error: Data not found. Please try again. ";

    cout << "Please enter the name of the child " << p << " will add:";

    cin >> child;

    if (family.Search(child) != nullptr)

        throw "Error: Already in the tree. Please try again. ";

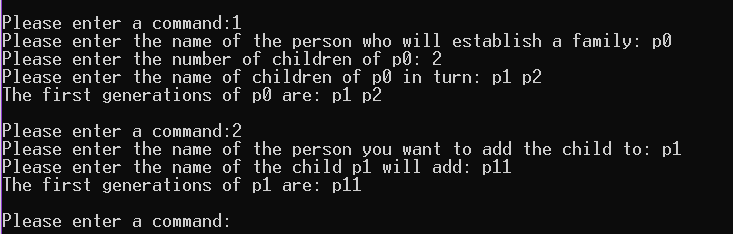
    family.InsChild(p, child);

    family.PrintChildren(cout, p);

    return;

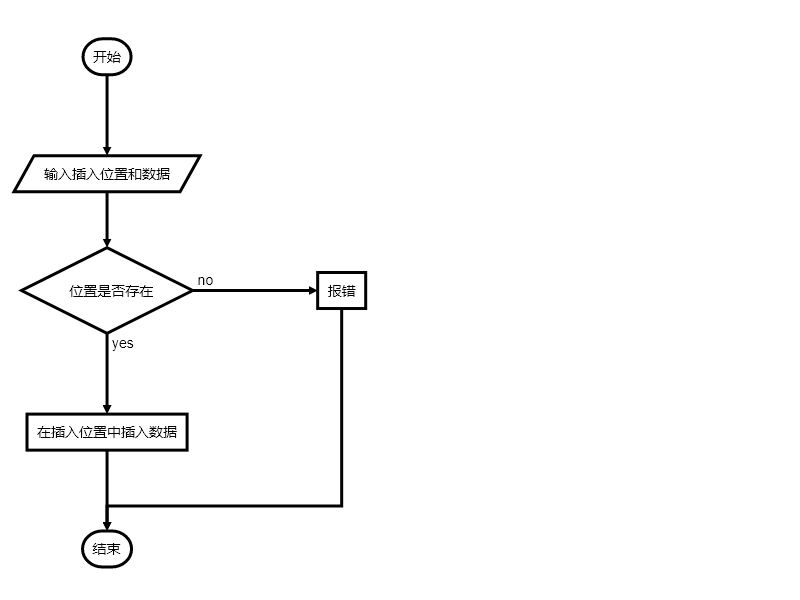
}

### 3.1.3 插入功能截屏示例



## 3.2 删除功能的实现

### 3.2.1 删除功能流程图



### 3.2.2 删除代码实现

template <class T>

void Tree<T>::Del(const T val) {

    TreeNode<T>\* node = this->Search(val);

    if (node == nullptr) throw "Error: Data not found. ";

    TreeNode<T>\* parent = node->\_prev;

    for (int i = 0; i < parent->\_children.size(); i++) {

        if (parent->\_children[i]->\_val == val) {

            auto v = parent->\_children;

            parent->\_children.clear();

            for (auto&& i : v)

                if (i->\_val != val) parent->\_children.push\_back(i);

            delete v[i];

        }

    }

}

void delSubTree(Tree<string>& family) {

    string p = "";

    cout << "Please enter the name of the person whose family you want "

            "to dissolve";

    cin >> p;

    auto pp = family.Search(p);

    if (pp == nullptr) throw "Error: Data not found. Please try again. ";

    if (pp == family.GetRoot())

        throw "Error: You are not allowed to delete the root node. Please try again.";

    cout << "The person whose family will be dissolved is " << p << ".\n";

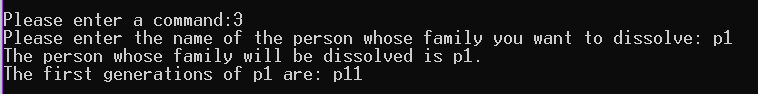
    family.PrintChildren(cout, p);

    family.Del(p);

    return;

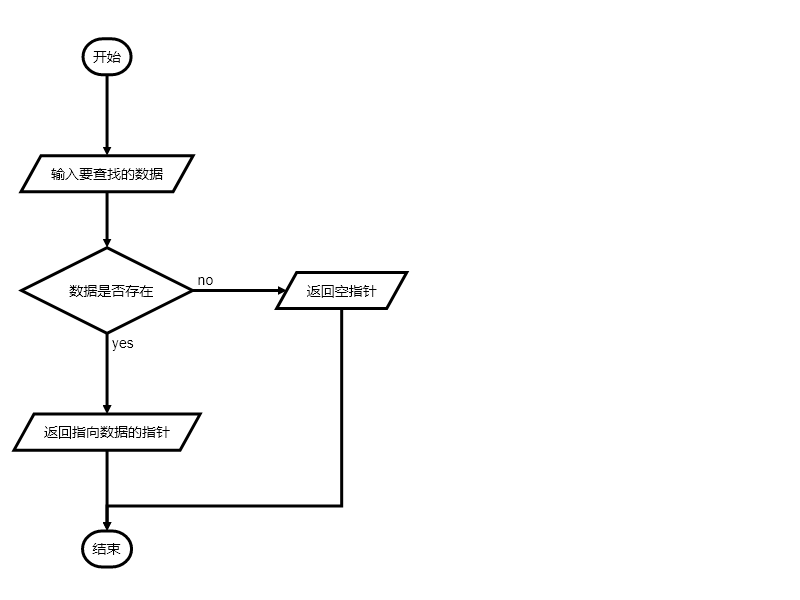
}

### 3.2.3 删除功能截屏示例



## 3.3 查找功能的实现

### 3.3.1 查找功能流程图



### 3.3.2 查找代码实现

template <class T>

TreeNode<T>\* Tree<T>::Search(T val) {

    if (this->\_root->\_val == val) return \_root;

    queue<TreeNode<T>\*> children;

    for (auto&& j : this->\_root->\_children) children.push(j);

    if (!children.empty()) return Bfs(children, val);

    return nullptr;

}

template <class T>

TreeNode<T>\* Tree<T>::Bfs(queue<TreeNode<T>\*> tree, T val) {

    queue<TreeNode<T>\*> children;

    while (!tree.empty()) {

        TreeNode<T>\* i = tree.front();

        tree.pop();

        if (i->\_val == val) return i;

        for (auto&& j : i->\_children) children.push(j);

    }

    if (!children.empty()) return Bfs(children, val);

    return nullptr;

}

void findMember(Tree<string>& family) {

    string p = "";

    cout << "Please enter the name of the person you want to search: ";

    cin >> p;

    auto pp = family.Search(p);

    if (pp == nullptr)

        cout << "Can not find the person. ";

    else {

        if (pp->\_prev != nullptr)

            cout << p << "'s parent is " << pp->\_prev->\_val;

        else

            cout << p << " is the root parent.";

        cout << endl;

        if (pp->\_children.size() != 0) {

            cout << "The first generations of " << p << " are: ";

            for (auto&& i : pp->\_children) cout << i->\_val << ' ';

            cout << endl;

        } else

            cout << p << " do not have any children.";

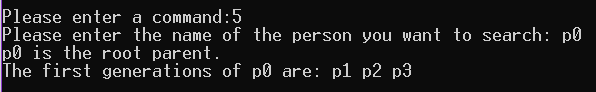
    }

    return;

    cout << endl;

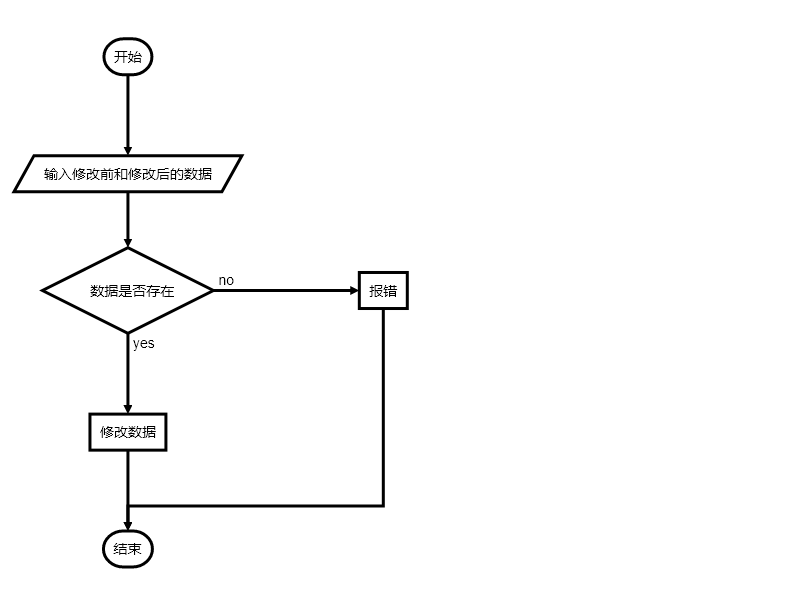
}

### 3.3.2 查找功能截屏示例



## 3.4 修改功能的实现

### 3.4.1 修改功能流程图



### 3.4.2 修改代码实现

template <class T>

void Tree<T>::ChangeName(const T prev, const T cur) {

    auto i = this->Search(prev);

    if (i == nullptr) throw "Error: Data not found. ";

    i->\_val = cur;

    return;

}

void changeName(Tree<string>& family) {

    string child = "";

    string p = "";

    cout << "Please enter the current name of the person whose name "

            "you want to change:";

    cin >> p;

    if (family.Search(p) == nullptr)

        throw "Error: Data not found. Please try again. ";

    cout << "Please enter the name you want to change:";

    cin >> child;

    if (family.Search(child) != nullptr)

        throw "Error: Already in the tree. Please try again. ";

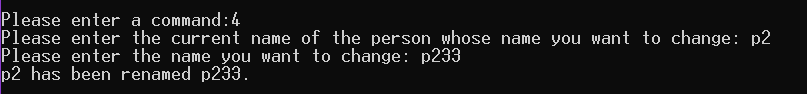
    family.ChangeName(p, child);

    cout << p << " has been renamed " << child << ".\n";

    return;

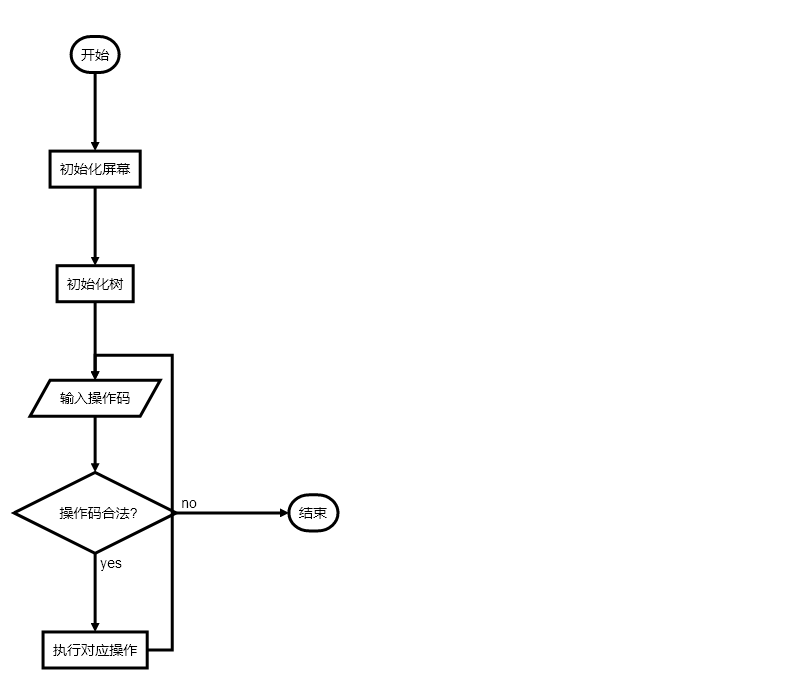
}

### 3.4.3 修改功能截屏示例



## 3.5 总体系统的实现

### 3.5.1 总体系统流程图



### 3.5.2 总体系统代码实现

string opening() {

    cout << "\*\*                Genealogy management system                \*\*"

         << endl

         << "==============================================================="

         << endl

         << "\*\*           Please select the action to perform:            \*\*"

         << endl

         << "\*\*             1 --- Perfect the genealogy                   \*\*"

         << endl

         << "\*\*             2 --- Add family members                      \*\*"

         << endl

         << "\*\*             3 --- Disband a part of the family            \*\*"

         << endl

         << "\*\*             4 --- Change the names of family members      \*\*"

         << endl

         << "\*\*             5 --- Find a family member                    \*\*"

         << endl

         << "\*\*             6 --- Exit                                    \*\*"

         << endl;

    cout << "Start with a family tree." << endl

         << "Please enter the name of the ancestor:";

    string temp = "";

    cin >> temp;

    return temp;

}

bool loop(Tree<string>& family) {

    cout << "\nPlease enter a command:";

    char temp = '\0';

    cin >> temp;

    try {

        switch (temp) {

            case '1':

                // add children

                addChildren(family);

                return true;

            case '2':

                // add a child;

                addAChild(family);

                return true;

            case '3':

                // delete sub tree;

                delSubTree(family);

                return true;

            case '4':

                // change name;

                changeName(family);

                return true;

            case '5':

                // find member;

                findMember(family);

                return true;

            default:

                return false;

        }

    } catch (const char\* e) {

        std::cerr << e << '\n';

        return true;

    }

}

int main() {

    string p = opening();

    Tree<string> tree(p);

    cout << "The ancestors of this family tree is " << p << ".\n" << endl;

    while (loop(tree))

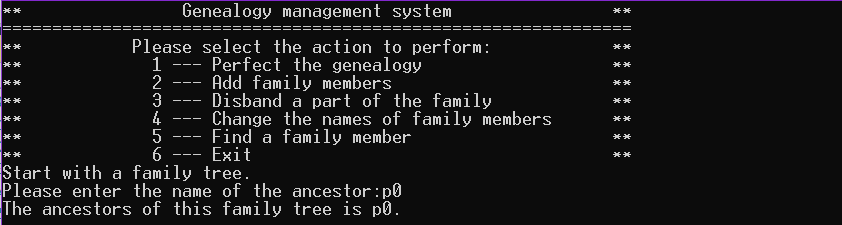
        ;

    std::cout << "Thanks for using it. See you next time! " << std::endl;

    return 0;

}

### 3.5.3 总体功能截屏示例



# 第4章 测试

## 4.1 功能测试

### 4.1.1 插入功能测试

**测试用例**：

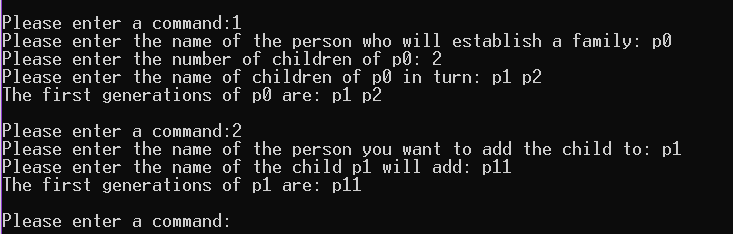
1 2 p1 p2

2 p1 p11

**预期结果**：

P0有两个孩子p1，p2；p1有一个孩子p11

**实验结果**



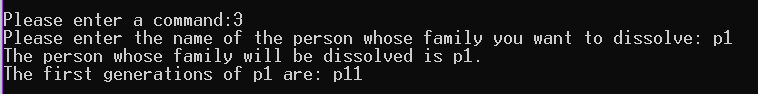
### 4.1.2 删除功能测试

**测试用例：**删除P1

**预期结果：**

删除p1和他的孩子p11

**实验结果：**



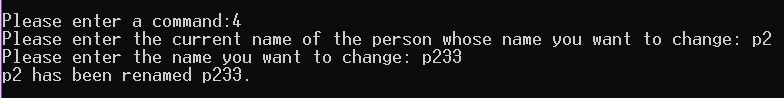
### 4.1.3 修改功能测试

**测试用例：**将p2更名为p233。

**预期结果：**

p2更名为p233。

**实验结果：**



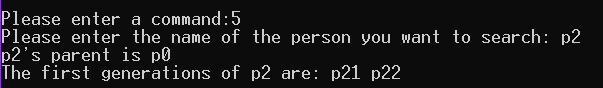
### 4.1.4 搜索功能测试

**测试用例：**搜索p2

**预期结果：**

p2具有孩子p21，p22，其父母为p0

**实验结果：**



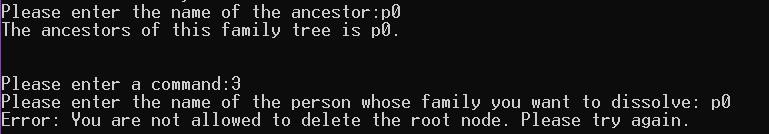
## 4.2 出错测试

### 4.2.1 删除树根节点

**测试用例：**删除树根节点p0

**预期结果：**程序不予删除根节点，抛出异常，程序正常运行不崩溃。

**实验结果：**

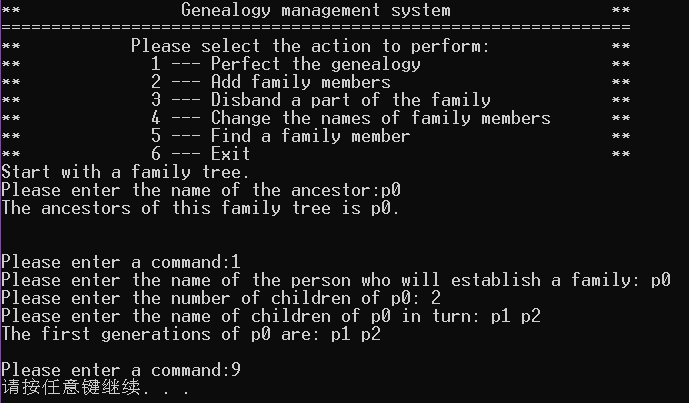


### 4.2.2 操作码错误

**测试用例：**输入操作码错误

**预期结果：**程序认为输入退出指令，自动退出。

**实验结果：**

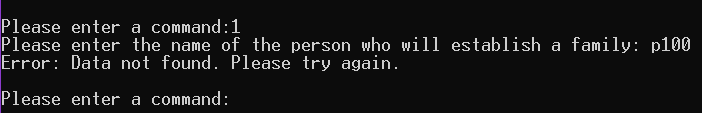


### 4.3.3 插入位置不存在

**测试用例：**向树中不存在的P100插入孩子

**预期结果：**程序抛出异常，程序正常运行不崩溃。

**实验结果：**

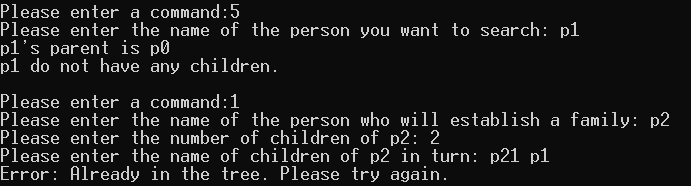


### 4.3.4 插入元素已存在

**测试用例：**向已有p1的树中尝试插入p1

**预期结果：**程序抛出异常，程序正常运行不崩溃。

**实验结果：**

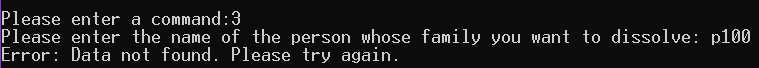


### 4.3.5 删除元素不存在

**测试用例：**删除树中不存在的P100

**预期结果：**程序抛出异常，程序正常运行不崩溃。

**实验结果：**

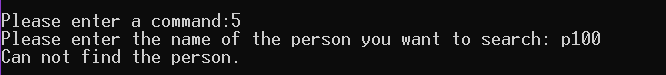


### 4.3.6 查找元素不存在

**测试用例：**查找树中不存在的P100

**预期结果：**程序抛出异常，程序正常运行不崩溃。

**实验结果：**

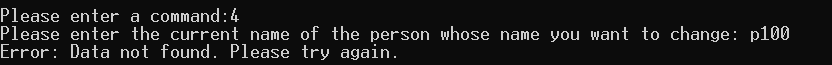


### 4.3.7 修改元素不存在

**测试用例：**修改树中不存在的P100

**预期结果：**程序抛出异常，程序正常运行不崩溃。

**实验结果：**



### 4.3.8 修改后元素已存在

**测试用例：**向已有p1的树中尝试将p2更名为p1

**预期结果：**程序抛出异常，程序正常运行不崩溃。

**实验结果：**

