Tongji University

Data Structure Course Curriculum Design Document

Task6: Family tree management system

Author:
嘉杰 李
Jiajie Li

Supervisor:
颖 张
Prof. Ying Zhang

A companion use documentation for a program submitted in fulfillment of the requirements for the Data Structure curriculum design

in the

School of Software Engineering, Tongji

December 27, 2019

“Thanks to my solid academic training, today I can write hundreds of words on virtually any topic without possessing a shred of information, which is how I got a good job in journalism.”

Dave Barry

Contents

Chapter 1

Analysis

1.1 Background Analysis

Genealogy is a special book genre in the form of a genealogy, which records a family's hereditary reproduction and important missions. Genealogy is a unique cultural heritage in China. It is one of the three major documents of the Chinese nation (national history, topography, genealogy). It belongs to precious humanities. , Have their unique and irreplaceable features. This project conducts a simple simulation of family tree management to achieve the functions of viewing personal information of ancestors and offspring, inserting family members, and deleting family members.

1.2 Functional Analysis

The essence of this project is to complete the creation, search, insertion, modification, and deletion of family tree member information. You can first define the family member data structure, and then use each function as a member function to complete the operation on the data. Function to verify the function of each function and get the results.

Chapter 2

Design

2.1 Data structure design

As described in the functional analysis above, the system requires the functions of establishment, search, insertion, modification, and deletion in the family tree system, and the relationship between children and parents is recursive, so a recursive data structure such as a tree is used.

2.2 Member function design

Vector Class:

```
template <class T>
class Vector {
public:
    Vector(int size=0):theSize(size),theCapacity(size+SPACE_CAPACITY){ data = new
        T[theCapacity];} // 构造函数
    ~Vector(void) { delete[] data;} // 析构函数
    Vector& operator=(Vector& other){
        // 判断是否为自身赋值
        if (this == &other)
            return *this;
        else {
            delete[]data;
            theSize = other.theSize;
            theCapacity = other.theCapacity;
            this->data = new T[theCapacity];

            for (int i = 0; i < theSize; ++i) {
                data[i] = other.data[i];
            }
            return *this;
        }
    } // 重载赋值函数
    Vector(Vector& other) :theSize(0),theCapacity(0),data(NULL){ *this=other; }//
        复制构造函数
        // 重新分配空间大小函数
        void reServe(int newCapacity) {
            if (newCapacity <= theCapacity)
                return;

            T *temp = data;
            data = new T[newCapacity];
            theCapacity=newCapacity;
            //std::cout << newCapacity << std::endl;
            for (int i = 0; i < theSize; ++i)
                data[i] = temp[i];
            delete[] temp;
        }
        // push_back函数
        void push_back(T val) {
            if (theSize == theCapacity)
                reServe(2 * theCapacity + 1);
            data[theSize++] = val;
        }
        bool pop_back(){
            if (!theSize) {
                std::cerr << "Fail to pop back, the vector is empty!\n";
                return false;
            }
            theSize--;
            return true;
        }
        bool Delete(int ind){
            if (ind<0||ind>=theSize){
                std::cerr << "Fail to delete, the ind is out of range!\n";
                return false;
            }
            for (int i=ind;i<theSize-1;++i) data[i]=data[i+1];
            return pop_back();
        }
        T& operator[] (int n) {return data[n];}
        T *data;
        int size() const {return theSize;}
    private:
        const int SPACE_CAPACITY = 16;
        int theCapacity;
        int theSize;
    };
```

Tree Class:

```
    template <class T> class __treenode {
private:
    Vector<__treenode<T>* > spring;
    __treenode<T>* father;
    T value;
public:
    __treenode(){
        register T temp;
        value = temp;
        father = nullptr;
    }
    __treenode(T theVal,__treenode<T> * theFather= nullptr):value(theVal){
        father = theFather;
    }
    ~__treenode(){
        father=nullptr;
    }

    T& getValue() {return value;}
    const T& getValue() const {return value;}
    void setValue(T val) {value = val;}

    __treenode<T>* getFather() const {return father; }
    void setFather(__treenode<T>* theFather) {father = theFather; }

    Vector<__treenode<T>* > &getSpring() {return spring;}
    const Vector<__treenode<T>* > &getSpring() const {return spring;}
    void setSpring(Vector<__treenode<T>* > theSpring) {spring = theSpring;}
};

template <class T>
class Tree {
private:
    __treenode<T>* root;
public:
    Tree() {root=nullptr;}
    Tree(T val) {root= new __treenode<T>(val);}
    ~Tree();
    bool isRoot(T val){return find(val)==root;}
    void ergodic();
    void ergodic(__treenode<T>* fa);
    void build(T val, Vector<__treenode<T>* > v);
    void clear();
    void clear(T val);
    bool clear(__treenode<T>* fa);
    void outSpring(T val);
    bool add(T target, T val);
    //void Delete(T val);
    bool change(T target, T val);
    __treenode<T>* find(T val);
    __treenode<T>* find(__treenode<T>* cur, T val);
```

```
};

template <class T>
void Tree<T>::build(T val, Vector<__treenode<T>* > v) {
    Vector<__treenode<T>* > &tempv = find(val)->getSpring();
    while (tempv.size()) tempv.Delete(0);
    while (v.size())
        if (find(v[0]->getValue())!= nullptr) {
            cerr << "This name '"<< v[0]->getValue() <<"' already exists! try again plz
                " << endl;
            v.Delete(0);
        }
        else {
            tempv.push_back(v[0]);
            v.Delete(0);
        }
}


template <class T>
bool Tree<T>::add(T target, T val){
    if (find(val)!= nullptr) {
        cerr << "The name '"<< val <<"' already exists! try again plz" << endl;
        return false;
    }
    if (find(target) == nullptr) {
        cerr << "The target '"<< target <<"' doesn't exist! try again plz" << endl;
        return false;
    }
    __treenode<T>* thefather = find(target);
    __treenode<T>* theSpring = new __treenode<T>(val, thefather);
    thefather->getSpring().push_back(theSpring);
    return true;
}

template <class T>
bool Tree<T>::change(T target, T val){
    if (find(val)!= nullptr) {
        cerr << "The name '"<< val <<"' already exists! try again plz" << endl;
        return false;
    }
    if (find(target) == nullptr) {
        cerr << "The target '"<< target <<"' doesn't exist! try again plz" << endl;
        return false;
    }
    __treenode<T>* father = find(target);
    father->setValue(val);
    return true;
```

```
}

template <class T>
__treenode<T>* Tree<T>::find(T val){
    return find(root, val);
}


template <class T>
__treenode<T>* Tree<T>::find(__treenode<T>* cur, T val){
    if (cur->getValue()==val) return cur;
    int len=cur->getSpring().size();
    for (int i=0;i<len;++i) {
        __treenode<T>* temp = find(cur->getSpring()[i], val);
        if (temp) return temp;
    }
    return nullptr;
}



template <class T>
void Tree<T>::ergodic() {
    ergodic(root);
    cout << endl;
}

template <class T>
void Tree<T>::ergodic(__treenode<T>* fa) {
    int len=fa->getSpring().size();
    cout << fa->getValue() << " ";
    for (int i=0;i<len;++i) ergodic(fa->getSpring()[i]);
}

template <class T>
void Tree<T>::outSpring(T val){
    Vector<__treenode<T>* > &v = find(val)->getSpring();
    int len=v.size();
    for (int i=0;i<len;++i) cout << v[i]->getValue() << " ";
    cout << endl;
}



template <class T>
Tree<T>::~Tree(){
    clear();
}
```

```
template <class T>
void Tree<T>::clear(){
   clear(root);
}

template <class T>
void Tree<T>::clear(T val){
   clear(find(val));
}

template <class T>
bool Tree<T>::clear(__treenode<T>* fa){
   if (fa==root) {
      cerr << "The name '"<< fa->getValue() <<"' is protected as the ancestor" <<
          endl;
      return false;
   }
   int len=fa->getSpring().size();
   Vector<__treenode<T>* > v = fa->getSpring();
   for (int i=0;i<len;++i) clear(v[i]);
   if (fa->getFather()!= nullptr)
      for (int i=0;i<fa->getFather()->getSpring().size();++i)
         if (fa->getFather()->getSpring()[i] == fa) {
            fa->getFather()->getSpring().Delete(i);
            break;
         }
   delete fa;
   return true;
}
```

2.3 System design

The system first initializes the screen, prints the system description on the screen, and the user enters the name of the ancestor to build a family.

Chapter 3

Implementation

3.1 Perfect family tree Function

3.1.1 Perfect family tree core code

```
cout << "Please input the name to establish family: ";
          cin >> name;
          if (geneTree.find(name)== nullptr) {
             cerr << "The name '" << name << "' doesn't exist! try again plz" <<
                 endl;
             break;
          }
          __treenode<string>* father = geneTree.find(name);
          cout << "Please input the number of " << name << "'s offsprings: ";
          cin >> n;
          Vector<__treenode<string> *> temp;
          cout << "Please input the names of " << name << "'s offsprings in order
              : ";
          for (int i=0; i<n; ++i) {
             cin >> name_t;
             __treenode<string> *son = new __treenode<string>(name_t, father);
             temp.push_back(son);
          }
          geneTree.build(name, temp);
          cout << name << "'s first generation offsprings are: ";
          geneTree.outSpring(name);
          cout << endl;
```

3.1.2 Perfect family tree flowchart

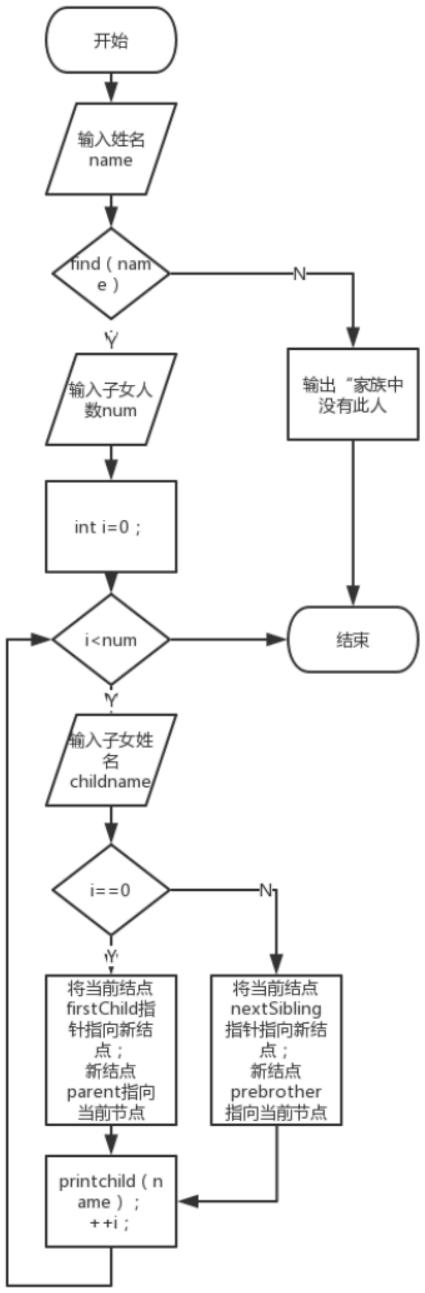

开始
输入姓名 name
find (name)
N
Y
输入子女人数num
输出 "家族中没有此人
int i=0 ;
i<num
结束
Y
输入子女姓名 childname
i==0
N
Y
将当前结点firstChild指针指向新结点 ; 新结点parent指向当前节点
将当前结点nextSibling指针指向新结点 ; 新结点prebrother指向当前节点
printchild (name) ; ++i ;

Figure 3.1: Perfect family tree flowchart

3.2 Add family members Function

3.2.1 Add family members code

```
cout << "Please input the name to add offspring: ";
          cin >> name;
          if (geneTree.find(name)== nullptr) {
                cerr << "The name '" << name << "' doesn't exist! try again plz" <<
                    endl;
                break;
            }
            cout << "Please input the names of " << name << "'s offspring: ";
            cin >> name_t;
            geneTree.add(name, name_t);
            cout << name << "'s first generation offsprings are: ";
            geneTree.outSpring(name);
            cout << endl;
            break;
```

3.2.2 Add family members flowchart

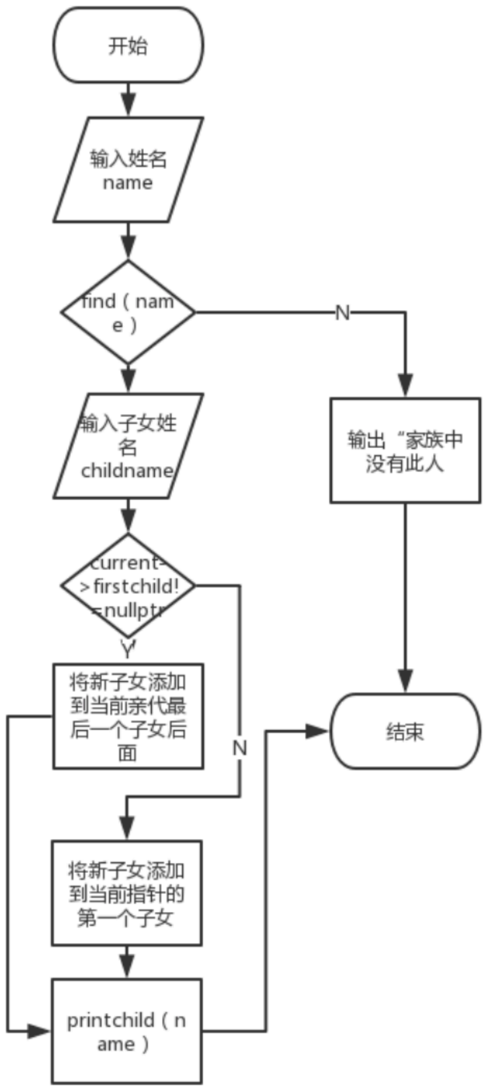


Figure 3.2: Add family members flowchart

3.3 Dissolution of local families Function

3.3.1 Dissolution of local families core code

```
cout << "Please input the name to disband the family: ";
          cin >> name;
          if (geneTree.find(name)== nullptr) {
             cerr << "The name '" << name << "' doesn't exist! try again plz" <<
                 endl;
             break;
          }
          if (!geneTree.isRoot(name)) {
             cout << name << "'s family is going to be disbanded. " << endl;
             cout << name << "'s first generation offsprings are: ";
             geneTree.outSpring(name);
          }
          geneTree.clear(name);
          cout << endl;
```

3.3.2 Dissolution of local families flowchart

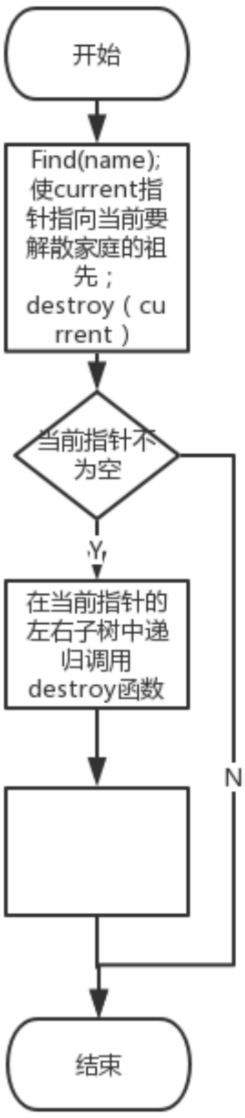

开始
Find(name);
使current指针指向当前要解散家庭的祖先；
destroy (current)
当前指针不为空
Y
在当前指针的左右子树中递归调用destroy函数
N
结束

Figure 3.3: Dissolution of local families flowchart

3.4 Change of family member name Function

3.4.1 Change of family member name core code

```
    cout << "Please input the current name to change: ";
            cin >> name;
            if (geneTree.find(name)== nullptr) {
               cerr << "The name '" << name << "' doesn't exist! try again plz" <<
                   endl;
               break;
            }
            cout << "Please input the new names of " << name << "' : ";
            cin >> name_t;
            geneTree.change(name, name_t);
            cout << name << "'s has been changed to " << name_t << endl <<endl;
            break;
```

3.4.2 Change of family member name flowchart

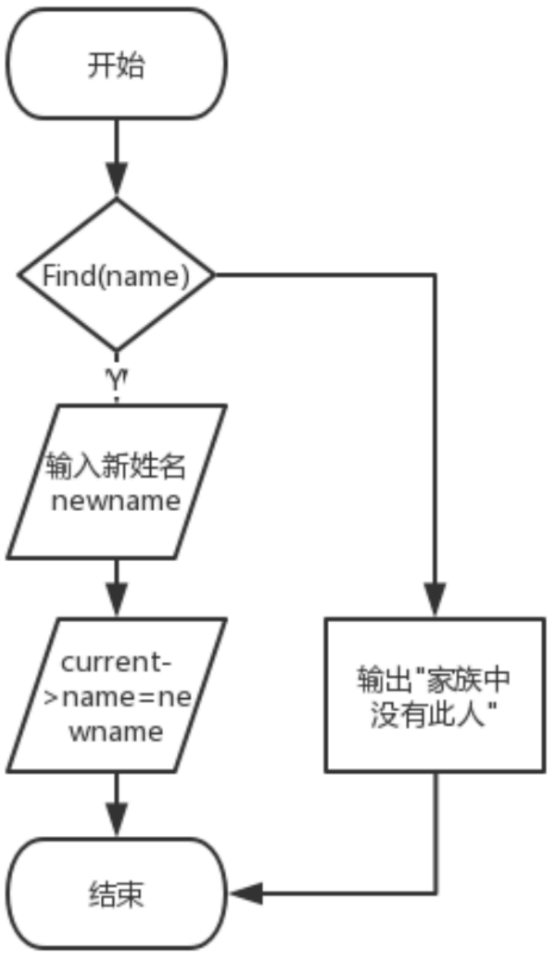


Figure 3.4: Change of family member name flowchart

Chapter 4

Test

4.1 Function Tests

4.1.1 Establish family tree

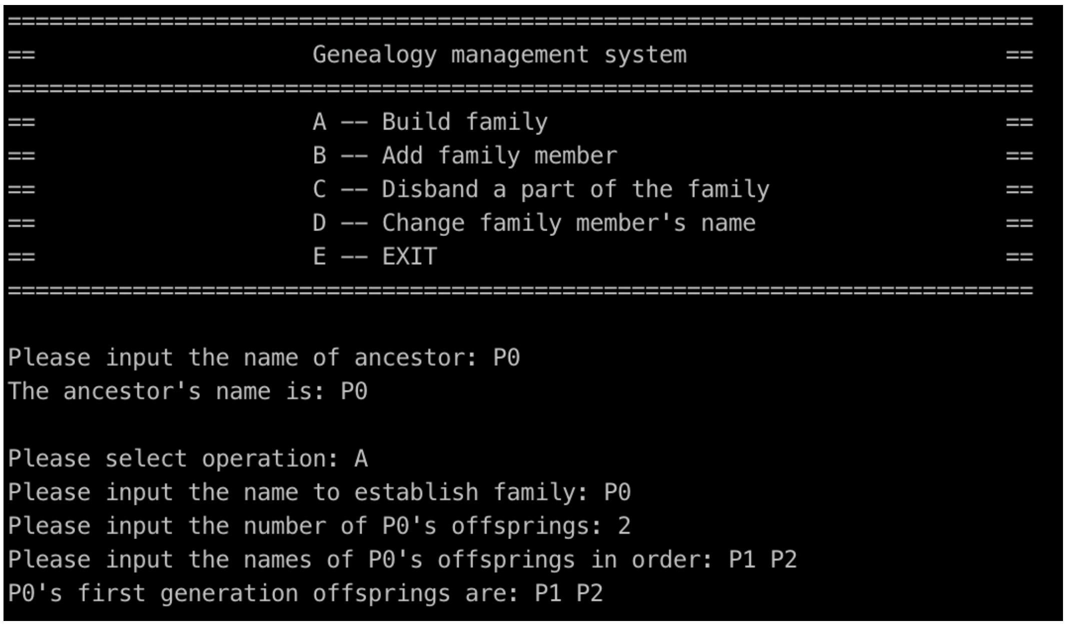
Test Case:

P0’sons are P1,P2

Expected Result:

P0’sons are P1,P2

Program Result:



```
===========================================================================
==                    Genealogy management system                        ==
===========================================================================
==                    A -- Build family                                  ==
==                    B -- Add family member                             ==
==                    C -- Disband a part of the family                  ==
==                    D -- Change family member's name                   ==
==                    E -- EXIT                                          ==
===========================================================================

Please input the name of ancestor: P0
The ancestor's name is: P0

Please select operation: A
Please input the name to establish family: P0
Please input the number of P0's offsprings: 2
Please input the names of P0's offsprings in order: P1 P2
P0's first generation offsprings are: P1 P2
```

Figure 4.1: Screenshots

4.1.2 Add family member

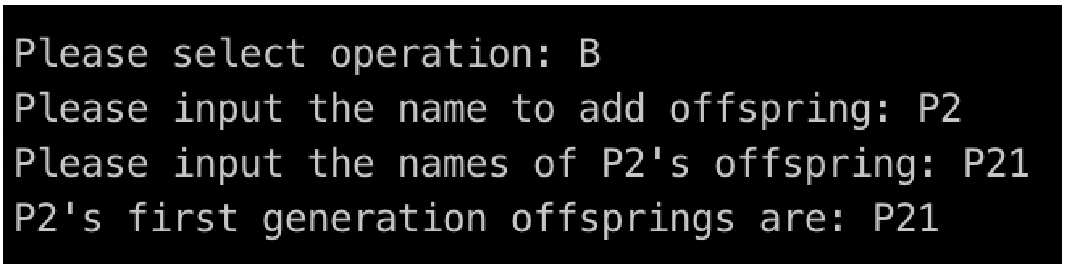
Test Case:

add P21 to P2’s offsprings

Expected Result:

P2’s first generation offsprings are: P21

Program Result:



```
Please select operation: B
Please input the name to add offspring: P2
Please input the names of P2's offspring: P21
P2's first generation offsprings are: P21
```

Figure 4.2: Screenshots

4.1.3 Dissolution of local families

Test Case:

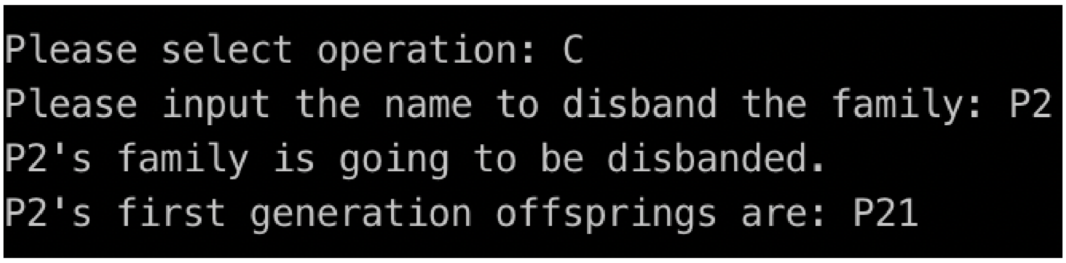
disband P2's family

Expected Result:

P2's family is going to be disbanded.

P2's first generation offsprings are: P21

Program Result:



```
Please select operation: C
Please input the name to disband the family: P2
P2's family is going to be disbanded.
P2's first generation offsprings are: P21
```

Figure 4.3: Screenshots

4.1.4 Change of family member name

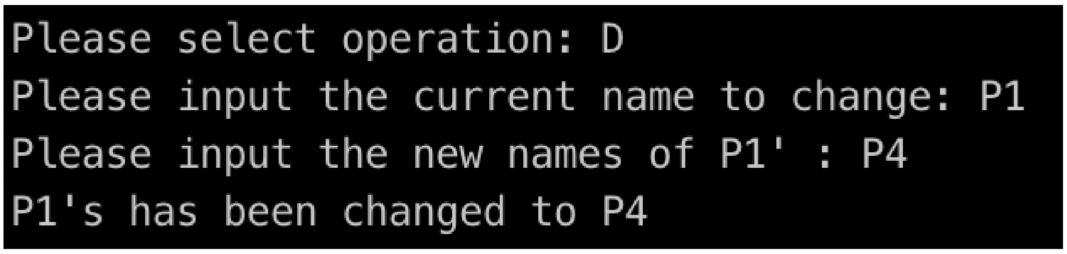
Test Case:

change P1's name to P4

Expected Result:

P1's has been changed to P4

Program Result:



```
Please select operation: D
Please input the current name to change: P1
Please input the new names of P1' : P4
P1's has been changed to P4
```

Figure 4.4: Screenshots

4.2 Boundary Tests

4.2.1 Delete ancestor

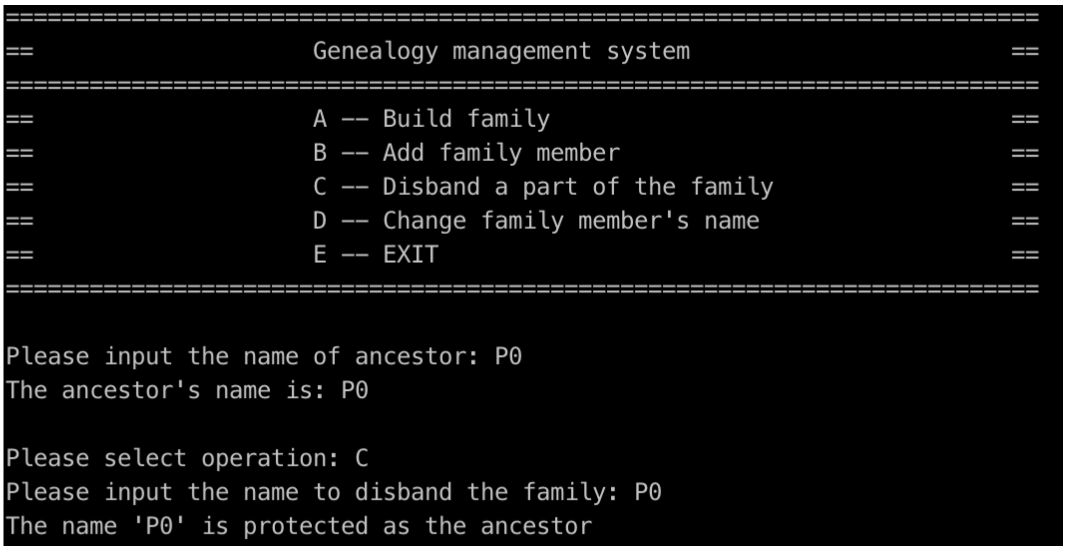
Test Case:

delete ancestoe P0

Expected Result:

The name ’P0’ is protected as the ancestor

Program Result:



```
==========================================================================
==                    Genealogy management system                       ==
==========================================================================
==                    A -- Build family                                 ==
==                    B -- Add family member                            ==
==                    C -- Disband a part of the family                 ==
==                    D -- Change family member's name                  ==
==                    E -- EXIT                                         ==
==========================================================================

Please input the name of ancestor: P0
The ancestor's name is: P0

Please select operation: C
Please input the name to disband the family: P0
The name 'P0' is protected as the ancestor
```

Figure 4.5: Screenshots

4.3 Exception

4.3.1 No such name

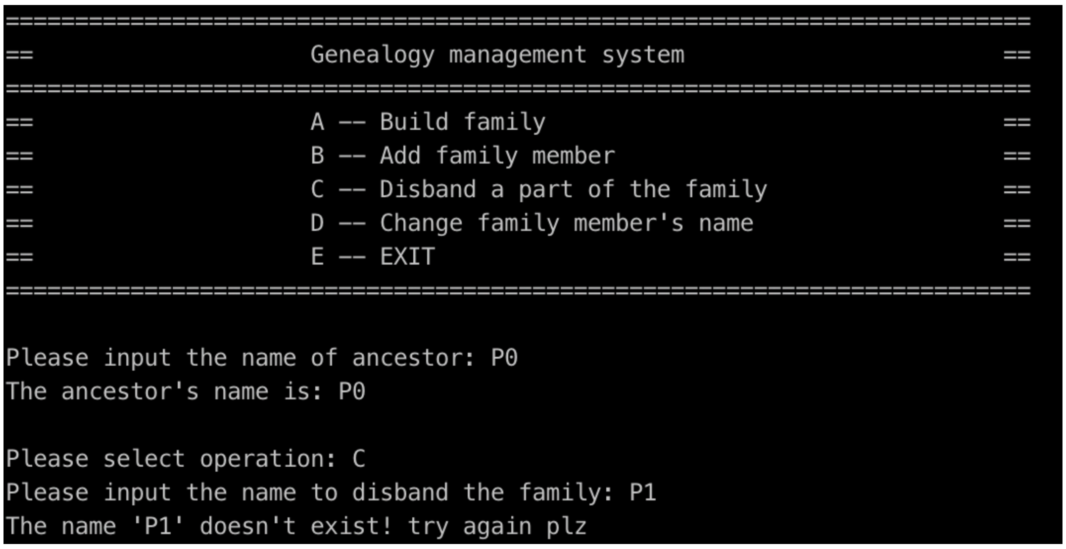
Test Case:

can’t find name P0 in system

Expected Result:

The program gives a prompt message, the program runs normally without crashing.

Program Result:



```
===========================================================================
==                    Genealogy management system                        ==
===========================================================================
==                    A -- Build family                                  ==
==                    B -- Add family member                             ==
==                    C -- Disband a part of the family                  ==
==                    D -- Change family member's name                   ==
==                    E -- EXIT                                          ==
===========================================================================

Please input the name of ancestor: P0
The ancestor's name is: P0

Please select operation: C
Please input the name to disband the family: P1
The name 'P1' doesn't exist! try again plz
```

Figure 4.6: Screenshots

Appendix A

Frequently Asked Questions

A.1 Why this document looks so sparse?

This document is built using LaTeX and is arranged in book format. There may be blank pages before and after each chapter.

A.2 Why is there header files in this project?

I wrote basic stl header files, such as vector.h, etc. Except for the header files I wrote, the only header files used in all projects are iostream and string.h