Project 5
通讯录

班级：教务二班

宋渝杰 18340146

廖家源 18340105

刘依澜 18340121

缪贝琪 18340131

【题目要求】

制作一个个人通讯录的管理软件

要求：

1.个人信息包含分类、姓名、性别、电话、邮箱、住址、邮编、QQ、微信、生日等；

2.具有增、删、改、查、显示等功能；

3.具有友好的界面和较强的容错能力。

【数据结构与算法】

C++部分

数据结构：

考虑到我们最近在学习 AVL 树，于是我们小组打算使用 AVL 树来进行对通信录的各项处理，最大的优点是在搜索操作中，可以把时间复杂度从 O（n）降低为 O（logn）。

算法：

一.实现 AVL 树：

1.我们这里定义了五棵树，分别是以 NAME,PHONE,QQ,WECHAT,EMAIL 这五个信息的字典序排列生成的平衡二叉树，这样便可以用这五个信息来查询这个人在通讯录中的全部信息。

2.定义一个结构体，存放树的节点的信息，并初始化。

3.**计算高度：**定义 int height(Node* node,Type tp)函数，用来计算 node 的高度，若 node 为空则返回 0，若 node 不为空，则令 rightheight = height(node->right[tp],tp) ， leftheight = height(node->left[tp],tp)；不断往下直到左子树及右子树为空，最后返回左子树和右子树之间更高的那个高度。

4.**计算平衡因子：**setBalance(Node* node,Type tp)函数，用来计算 node 的平衡因子，平衡因子等于 node 右子树的高度减去左子树的高度。

5.**右旋转：**定义 Node* turnRight(Node* node,Type tp)，实现右旋转。

如图 1-1 所示，插入点 C 后，A 的平衡因子绝对值大于 1，使得二叉树失衡，那么就要进行平衡调整，右旋后把 A 变成 B 的右节点，若 B 有右子树，则其右子树变为 A 的左子树。具体实现方法为：node 指向平衡因子绝对值大于 1 的节点 A，tmp 指向 A 的左节点 B，若 A 有父节点，则将 A 的父节点的左节点（或右节点，取决于 node 的位置）置为 tmp，即 node 的左子节点，然后再将 tmp 的父节

点置为 node 的父节点，而 node 的父节点变为 tmp，node 的左节点变为 tmp 的右节点（即 A 的左子树变为 B 的右子树），若 node 的左节点不为空，把 node 的左节点的父节点变为 node，再把 tmp 的右节点置为 node，便可以实现如图的右旋变化，然后调用 setBalance 函数重新计算 node 跟 tmp 的平衡因子，最后返回 tmp。

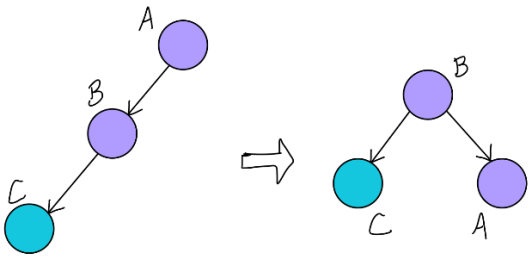


图 1-1 右旋转

6. **左旋转：**定义 Node* turnLeft (Node* node,Type tp)，实现左旋转。

如图 1-2 所示，插入点 C 后，A 的平衡因子绝对值大于 1，使得二叉树失衡，那么就要进行平衡调整，左旋使 A 变为 B 的左节点，若 B 有左子树，则 B 的左子树变为 A 的右子树。具体实现方法为：node 指向平衡因子绝对值大于 1 的节点 A，tmp 指向 A 的右节点 B，若 node 有父节点，则把 node 的父节点的右节点（或左节点，取决于 node 的位置）置为 tmp，tmp 的父节点变为 node 的父节点，node 的父节点变为 tmp，node 的右节点变为 tmp 的左节点（即 A 的右子树为 B 的左子树），tmp 的左节点变为 node，若 node 右节点不为空，则将 node 右节点的父节点置为 node，这样便实现了如图的左旋操作，然后重新计算 node 跟 tmp 的平衡因子，最后返回 tmp。

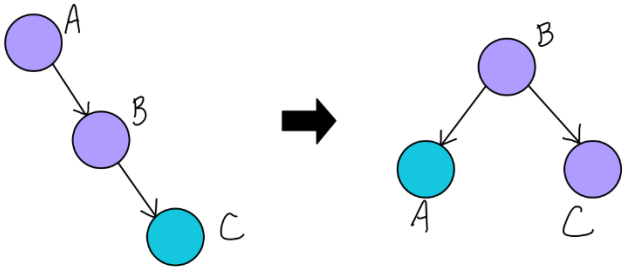


图 1-2 左旋转

7. **先右旋转再左旋转：**定义 Node* turnRightLeft(Node* node,Type tp)函数，实现右旋转后左旋转。

如图 1-3 所示，先将 node 的右节点左旋转，使 node 右节点变为 turnRight(node->right[tp],tp)，再将 node 左旋转，返回 turnLeft(node,tp)。

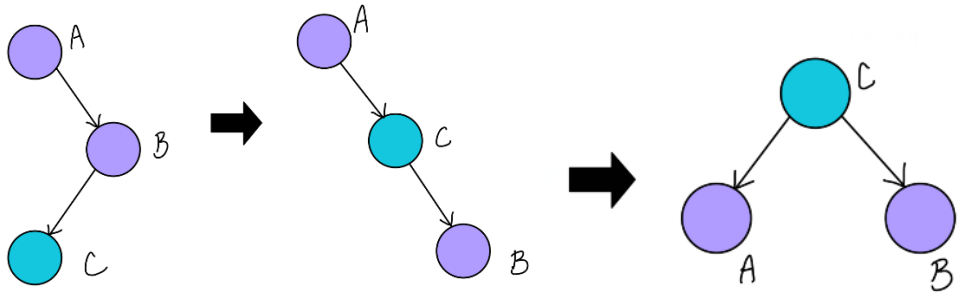


图 1-3 先右旋转后左旋转

8. **先左旋转再右旋转：** 定义 Node* turnLeftRight(Node* node,Type tp)函数，实现左旋转后右旋转。

如图 1-4 所示，先将 node 的左节点左旋转，使 node 左节点变为 turnLeft(node->left[tp],tp)，再将 node 右旋转，返回 turnRight(node,tp)。

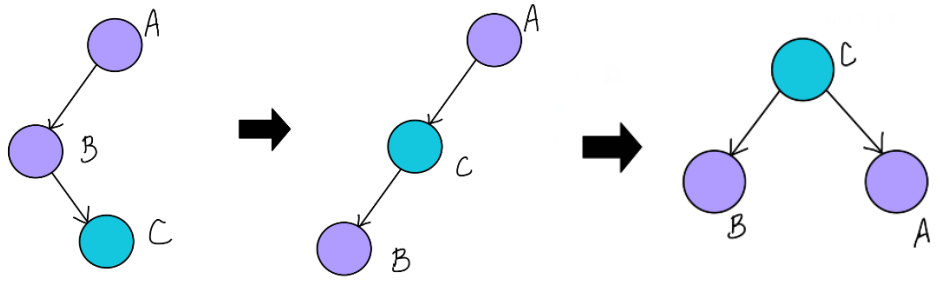


图 1-4 先左旋转后右旋转

9. **平衡调整：** 定义 void rebalance(Node* node,Type tp)函数，先计算 node 的平衡因子：

（1）node 的平衡因子为-2 时，此时 node 的左子树比右子树高 2 层：

①若 node 左子节点的左子树比右子树高，则进行一次右旋；

②若左子节点的右子树比左子树高，则先进行左旋，再进行右旋。

（2）node 的平衡因子为 2 时，此时 node 的右子树比左子树高 2 层：

①若子节点的右子树比左子树高，则进行一次左旋；

②若子节点的右子树比左子树低，则先进行右旋，再进行左旋。

进行完旋转操作后，若 node 父节点不为空，则依次往上对父节点进行平衡调整，即 rebalance(node->parent[tp],tp)，若 node 父节点为空，则直接将 Root[tp]等于 node。

二. AVL 树的各种操作

1. **插入：** void insert(Node* node,Type tp)，若 Root[tp]为空，则直接把 Root[tp]变为 node；若 Root[tp]不为空，定义 Node*tmp，令 tmp 指向 Root[tp]，（1）若 tmp 的信息字典序大于 node 的信息，①若 tmp 的左子节点不

为空，则 tmp 变为 tmp 的左子节点②若 tmp 的左子节点为空，则在 tmp 的左子节点插入 node，并让 node 的父节点变为 tmp，如此便完成插入，然后调用 rebalance(tmp,tp)对其进行平衡调整，并返回。(2) 同理，若 tmp 的信息字典序小于 node 的信息，①若 tmp 的右子节点不为空，则 tmp 变为 tmp 的右子节点②若 tmp 的右子节点为空，则在 tmp 的右子节点插入 node，并让 node 的父节点变为 tmp，如此便完成插入，然后调用 rebalance(tmp,tp)对其进行平衡调整，并返回。然后一直循环这个过程直到插入成功。

2.**删除：** void remove(Node*node,Type tp)，

(1) 若 node 左右子树都不为空，定义 Node*tmp，令 tmp 指向 node 的右子节点，令 flag=false（用来判断替换 node 的对象是否就是 node 的右子树），若 tmp 的左子节点不为空，则 flag=true，则令 tmp 等于 tmp 的左节点，一直循环直到 tmp 的左节点为空。

①flag 为 true 时，替换 node 的对象不是 node 的右子树而是 node 左子树的尽头，即现在的 tmp。此时令 tmp 的父节点的左节点置为空，tmp 的父节点变为 node 的父节点，tmp 的左节点置为 node 的左节点，tmp 的右节点置为 node 的右节点；再设置父节点的子节点指向，若 node 原本是其父节点的左子树，则将 tmp 的父节点的左节点置为 tmp，若 node 原本为其父节点的右子树，则令 tmp 父节点的右节点等于 tmp；然后设置子节点的父节点指向，若 node 左子树不为空，则令 node 的左子节点的父节点等于 tmp，同理若 node 右子树不为空，则令 node 的右子节点的父节点等于 tmp。

②flag 为 false 时，替换 node 的对象就是 node 的右子树，则令 tmp 的父节点的右子节点等于空，tmp 的左子节点置为 node 的左子节点，若 node 左子树不为空，则将 node 的左子节点的父节点置为 tmp；再设置父节点关系，令 tmp 的父节点等于 node 的父节点，若 node 父节点不为空，则令 node 的父节点的左节点（或右节点，根据 node 的位置而定）等于 tmp。

这样便完成了删除 node 节点，最后再调用 rebalance(tmp,tp)重新对这棵二叉树进行平衡调整。若 node 刚好为根，则令 Root[tp]等于 tmp。

(2) 若 node 左子树不为空，右子树为空，则左子树直接往上移替代 node。将 node 的左节点的父节点置为 node 的父节点，若 node 父节点不为空，则令 node 的父节点的左节点（或右节点，根据 node 的位置而定）等于 tmp，若 node 刚好为根，令 Root[tp]=node->left[tp]，最后再对其进行平衡调整，rebalance(node->left[tp],tp)。

(3) 若 node 左子树为空，右子树不为空，则右子树直接往上移替代 node。将 node 的右节点的父节点置为 node 的父节点，若 node 父节点不为空，则令 node 的父节点的左节点（或右节点，根据 node 的位置而定）等于 tmp，若 node 刚好为根，令 Root[tp]=node->right[tp]，最后再对其进行平衡调整，

rebalance(node->right[tp],tp)。

（4）若 node 左右子树都为空，则直接删除。若 node 为根节点，则直接令 Root[tp]等于空；若 node 不为根节点，则令 node 的父节点的左节点（或右节点，根据 node 的位置而定）等于 tmp，若 node 的父节点不为空，则对其进行平衡调整，rebalance(node->parent[tp],tp)。

3. **查找：**Node* search(string info,Type tp)，查找 info 是否已经存在于二叉树中，令 node=Root[tp]。当 node 不为空且 node 的信息不等于 info 时，若 node 的信息字典序小于 info，令 node 等于 node 的右子树；若 node 的信息字典序大于 info 的字典序，令 node 等于 node 的左子树，就这样一直循环，如果查到最后都没有 node=NULL，即 info 在本来的 tree 中不存在，最后返回 Node。

4. **判断是否插入成功：**bool Insert(string info[INFONUM])函数，用 search(info[i],Type(i))判断 info 里的前五个信息是否有与原本就在通讯录里存在的信息重复的，有则不允许添加（因为我们还要用这前五个信息来查询信息，所以不允许有重复），返回 false；没有重复则调用 insert 函数添加节点，返回 true。

5. **判断是否删除成功：**bool Remove(Node*node)函数，若 node 为空，则直接返回 false；若 node 不为空，因为开始定义了五个 AVL 树，所以要把这五棵树涉及到 node 的部分都删除，即 remove(node,Type(i))，最后 delete node，返回 true，删除成功。

6. **先序遍历输出：**void preorder(Node*node,Type tp)，先输出根节点信息，再遍历左子树，然后遍历右子树，并输出各节点信息。

三、通讯录应用中的操作

如图 2-1，定义了以下变量方便操作：有 string 类型的数组常量，以记录有哪些数据（其中前五个为每个人都不一样的个人数据，后五个为可以重复的用来分类的共同数据）；Node*型的数组，分别代表以名字、电话、QQ、微信和邮箱为 Type 的 AVL 树；int 类型的 Count 用来记录通讯录中的人数；string 类型的为 vector 的数组 Class[INFONUM-TREENUM]用来进行后五个数据的分类查找。

```
const string Str[INFONUM] = {"名字","电话号码","QQ","微信","邮箱","地址","生日","邮编","性别","类别"};
Node* Root[TREENUM];
int Count;  // 通讯录中记录的人数
vector<string> Class[INFONUM-TREENUM]; // 用于分类查找
```

图 2-1 使用的变量

如图 2-2，在这个通讯录的应用中，我们主要使用了以上函数，并实现了以下功能：（1）查看通讯录的全部信息；（2）通讯录统计查询；（3）查询联系人；

（4）添加联系人；（5）修改联系人；（6）删除联系人。

```
void Menu();         // 输出主菜单
void Initial();      // 程序启动的初始化操作
void ShowALL();      // 显示所有人信息
void FindMember();   //查找显示对象
void AddMember();    //添加对象
void ResMember();    //修改对象
void DelMember();    //删除对象
void Statistics(); // 统计通讯录信息
```

图 2-2 实现通讯录的主要函数

如图 2-3，为了代码的方便实现，我们构造了以上辅助函数来实现主要函数的功能，在报告中对此不作过多介绍。

```
char GetMember(Node*&node); //将多个函数的共同部分独立成函数，减少代码量
void ShowSpecialTraversal(Node* node,int& cou,const int& ind,const string& info); // 筛选显示
void ShowTraversal(Node* node,int& cou); //// 遍历树将信息输出显示
void WriteTraversal(Node* node,ofstream&out); // 遍历树将信息写入文件
void FileRead();     // 读取文件信息
void FileWrite();    // 修改并关闭文件
void FreeSpace(Node* node);// 释放空间
bool Find(Node* node,int ind,const string& info);
```

图 2-3 辅助函数

下面会具体介绍各个主要函数的实现：

1. **菜单函数：**在 Menu 函数中，我们主要是构建了一个稍微美观的画面来给使用者带来方便明晰的指引。

2. **初始化函数：**首先初始化 Count 为 0 以及各棵树的指针都指向 NULL，然后通过 FileRead 函数从文件中读取数据存到变量中，如图 2-4 所示。

```
void Initial(){
    Count=0;
    for(int i=0;i<TREENUM;i++) Root[i]=NULL;
    FileRead();
}
```

图 2-4 通讯录初始化

3. **显示所有信息：**首先判断通讯录是否为空，若为空则输出“通讯录为空”；否则按照数组 Str 所指定的数据来表格式输出信息。其中输出信息由辅助函数 ShowTraversal 来实现，它通过对以用 Type[0]来构造的树进行先序遍历输出数据来实现信息的输出。

```
void ShowALL(){
    cout<<"\n\t通讯录信息:\n\n";
    if(Root[NAME]==NULL or Count==0){
        cout<<"\t\t\t通讯录为空！"<<endl;
    }
    else{
        int cou = 1;
        cout<<"       序号   姓名     电话         QQ          微信          邮箱          住址     生日       邮编  性别  类别\n\n";
        ShowTraversal(Root[0],cou);
    }
    cout<<"\n\n\n\t\t\t\t\t按任意键返回...";
    _getch();
}
```

图 2-5 显示通讯录的所有信息

4. **显示查找对象：**如图 2-6，通过辅助函数 GetMember 来实现对查询方式（共有姓名、电话、QQ、微信、邮箱五种方式）的选择以及信息的输入，然后在此函数中实现此信息对应的人的所有信息的输出。

```
void FindMember(){
    FM1:
    cout<<"\n--联系人查询\n\n";
    Node* obj = NULL;
    char x = GetMember(obj);
    if(x=='q') return;
    if(x!='n'){
        system("cls");
        if(x=='r'){
            goto FM1;
        }
        else{
            cout<<"\n\n";
            cout<<"----联系人信息:\n\n";
            cout<<"\n          姓名      电话          QQ";
            cout<<"\t";
            cout<<left<<setw(7)<<obj->information[0];
            cout<<left<<setw(13)<<obj->information[1];
            cout<<left<<setw(11)<<obj->information[2];
            cout<<left<<setw(12)<<obj->information[3];
            cout<<left<<setw(18)<<obj->information[4];
            cout<<left<<setw(8)<<obj->information[5];
            cout<<left<<setw(12)<<obj->information[6];
            cout<<left<<setw(8)<<obj->information[7];
            cout<<left<<setw(5)<<obj->information[8];
            cout<<left<<setw(6)<<obj->information[9];
            cout<<endl;
        }
    }
```

图 2-6 显示查找对象的信息

5. **添加对象：**如图 2-7，先按照所需信息输入要添加的联系人的信息（输入 q 取消添加），接着通过辅助函数 Insert 来判断是否输入成功（即判断前五个数据是否有已经存在的，若有则说明输入有误，故输入失败）。而若输入成功，那么就要通过 find 函数查看后五个数据类型是否已经存在，如果存在则直接该类型数据+1，否则要在对应 Type[j]的 Class[j]中新加入对应数据，如地点的增加等。

```
void AddMember(){
    cout<<"\n--添加联系人（输入\"q\"取消添加)\n\n";
    string info[INFONUM];
    string tmp;
    for(int i=0;i<INFONUM;i++){
        cout<<"----请输入联系人的"<<Str[i]<<":";
        cin>>tmp;
        if(tmp != "q") info[i]=tmp;
        else return;
    }
    bool success = Insert(info);
    if(success){
        for(int j=0;j<INFONUM-TREENUM;j++){
            // 查找类型是否已经存在
            if(find(Class[j].begin(), Class[j].end(), info[j]) == Class[j].end()){
                Class[j].push_back(info[j+TREENUM]);
            }
        }
        cout<<"--添加成功！\n";
        Count++; // 数量++
    }
    else{
        cout<<"--信息重复，添加失败！\n";
    }
    cout<<"\n\n\n\t\t\t\t\t按任意键返回...";
    _getch();
}
```

图 2-7 添加联系人

6. **删除对象：**删除联系人与查找联系人的函数类似，因为是要先查找到该联系人才能删除，而我们在查找和删除间又加了一个“是否确认删除”的提醒，防止误删。

7. **修改对象：**修改联系人信息和查找删除函数也类似，先查找到该联系人才会进行下一步的修改。值得注意的是，如图 2-8 所示，我们并没有选择对数据的直接修改，而是通过先删除后插入来实现修改这一操作。

```
if(x=='q' or x=='Q'){
    if(changed){  // 修改方式是先 删除 再 插入 ，不能直接修改，因为涉及树要重新构造
        string tmp[INFONUM] = obj->information;
        Remove(obj);
        Insert(tmp);
    }
    return;
}
else{
    cout<<x<<endl;
    cout<<"----请输入联系人的"<<Str[index]<<":";
    string info;
    cin>>info;
    if(info!=obj->information[index]){
        obj->information[index] = info;
        changed = true;
    }
    cout<<"----修改成功！"<<endl;
    Sleep(500);
    system("cls");
    goto RM2;
}
```

图 2-8 修改联系人信息的修改部分

8. **统计通讯录信息：**首先如图 2-9，先显示各信息类型的数量，比如联系人中有多少种不同的地址。

```
cout<<"\n--通讯录统计查询\n\n";
cout<<"----联系人数量: "<<Count<<endl<<endl;
cout<<"----各信息类型数量:\n\n";
for(int i=0;i<INFONUM-TREENUM;i++){
    cout<<"\t("<<i+1<<")"<<Str[i+TREENUM]<<" : "<<Class[i].size()<<endl;
}
```

图 2-9 显示个信息类型的数目

如图 2-10，通过输入编号显示对应的数据类型的所有数据，如选择地址的编号则输出：花都区、越秀区等等……接着，我们通过输入 string 类型的数据从而可以查询符合某个数据的联系人有哪些并将其信息显示出来。

```
int ind = x-'1';
cout<<"\n--通讯录统计查询\n\n";
cout<<"----按"<<Str[ind+TREENUM]<<"类型查询统计:\n\n";
for(int i=0;i<Class[ind].size();i++){
    if(i%8==0) cout<<"     ";
    cout<<left<<setw(10)<<Class[ind][i]<<"  ";
    if((i+1)%8==0) cout<<endl;
}
string info;
cout<<"\n\n----请输入查询信息（q返回）:";
while(1){
    cin>>info;
    if(info=="q" or info=="Q"){
        system("cls");
        goto ST1;
    }
    if(find(Class[ind].begin(), Class[ind].end(), info) == Class[ind].end()){
        cout<<"----输入错误，重新输入:";
    }else break;
}
```

图 2-10 显示所要查询的数据类型详细信息及具有该信息的联系人

C#部分

数据结构：

这次我们打算继续同时使用 c#语言来实现该通讯录程序，因为实现 c#程序能够很好的满足题目要求的第三点：具有友好的界面和较强的容错能力：c#的窗体程序不仅比 c++控制台程序美观，而且 c#的控件对于错误的输入也有较易实现的异常处理功能。

同时，我们打算加入数据库技术，让这次的项目程序成为一个数据处理的程序，也能更加有意义一点。

数据库采用 Microsoft Access 2019，数据库表设计如图 3-1 所示：

字段名称	数据类型
序号	数字
分类	短文本
姓名	短文本
性别	短文本
电话	短文本
邮箱	短文本
住址	短文本
邮编	短文本
QQ	短文本
微信	短文本
生日	短文本

图 3-1 数据库表设计

同时，设计了 python 代码，可随机生成 2000 条通讯录信息，并生成对应的 csv 格式的数据文件，如图 3-2 所示：

```python
# -*- coding: UTF-8 -*-
import pandas as pd
import numpy as np
import random

dic = ["家人", "朋友" ,"小学同学", "初中同学" ,"高中同学", "大学同学" ,"姐妹", "前男友","路人"]
dic2 = ["男","女"]
dic3 = ["越秀区","番禺区","萝岗区","从化区","荔湾区","花都区","白云区","黄浦区","南沙区","增城区","海珠区","天河区"]
dic4 = ["510000","511400","510000","510900","510100","510800","510400","510700","511400","511300","510200","510600"]

#设置信息:
length = 2000

#生成csv
data = []
for i in range(length):
    name = ''.join(random.sample(['z','y','x','w','v','u','t','s','r','q','p','o','n','m','l','k','j','i','h','g','f','e','d','c','b','a'], 5))
    fen = random.randint(0,100)
    if fen < 2:
        fen = 7
    elif fen < 5:
        fen = 6
    elif fen < 12:
        fen = 0
    elif fen < 20:
        fen = 1
    elif fen < 50:
        fen  = random.randint(2,5)
    else:
        fen = 8
    xin = random.randint(0,1)
    if fen == 6:
        xin = 1
    elif fen == 7:
        xin = 0
    di = random.randint(0,len(dic3)-1)
    zhuzhi = dic3[di]
    youbian = dic4[di]
    dianhua = random.randint(13100000000,15100000000)
    qq = random.randint(10000000,1000000000)
    youxiang = str(qq) + "@qq.com"
    shengri = str(random.randint(1998,2018)) + "/" + str(random.randint(1,12)) + "/" + str(random.randint(1,28))
    weixin = "B" + str(qq)
    row = [i+1,dic[fen],name,dic2[xin],dianhua,youxiang,zhuzhi,youbian,qq,weixin,shengri]
    data.append(row)
df = pd.DataFrame(data)
print(df)
df.to_csv("data.csv",encoding='utf-8-sig',index = None,header = ["序号","分类","姓名","性别","电话","邮箱","住址","邮编","QQ","微信","生日"])
```

图 3-2 python 生成随机通讯录数据代码

运行上述 python 代码之后，生成 data.csv 文件，如图 3-3 所示：

序号	分类	姓名	性别	电话	邮箱	住址	邮编	QQ	微信	生日
1	前男友	lohmv	男	13530444396	425430015@qq.com	白云区	510400	425430015	B425430015	9/6/1999
2	小学同学	nxskm	男	13799486672	709035594@qq.com	萝岗区	510000	709035594	B709035594	12/3/2018
3	小学同学	brngh	男	14896060638	323860538@qq.com	番禺区	511400	323860538	B323860538	28/7/2008
4	路人	hbjds	男	14665247257	712721671@qq.com	番禺区	511400	712721671	B712721671	11/2/2003
5	朋友	vbpte	男	13593252689	914607232@qq.com	黄浦区	510700	914607232	B914607232	13/10/2012
6	路人	jinws	男	14228826937	488459500@qq.com	萝岗区	510000	488459500	B488459500	22/8/2010
7	路人	lnzbd	女	13877426382	341141335@qq.com	海珠区	510200	341141335	B341141335	28/5/2011
8	小学同学	uinog	女	13882170176	169618235@qq.com	天河区	510600	169618235	B169618235	7/12/2010
9	路人	qjlow	女	14693084252	103370001@qq.com	天河区	510600	103370001	B103370001	21/11/2005
10	路人	trljd	男	14765076200	641891074@qq.com	白云区	510400	641891074	B641891074	22/12/1998
11	大学同学	udxjz	女	13807312544	644372868@qq.com	黄浦区	510700	644372868	B644372868	7/6/2006
12	路人	vsdzj	女	13343939744	330855313@qq.com	萝岗区	510000	330855313	B330855313	17/7/2003
13	高中同学	rqdja	女	14523565884	276009364@qq.com	从化区	510900	276009364	B276009364	7/5/2006
14	小学同学	fuzls	女	15043792612	722565676@qq.com	白云区	510400	722565676	B722565676	20/3/2016
15	前男友	dofmr	男	14014568553	18635640@qq.com	从化区	510900	18635640	B18635640	24/4/2012
16	初中同学	yczmn	男	14891427897	852433061@qq.com	海珠区	510200	852433061	B852433061	28/8/2000
17	高中同学	ixyew	男	14544259804	225117336@qq.com	南沙区	511400	225117336	B225117336	7/12/2011

data

图 3-3 生成的随机数据的部分图

至此，我们有了程序以及相应的数据文件，如图 3-4 所示：

图 3-4 csv 数据文件、Access 数据库文件、python 生成数据文件、主程序

算法：

下面是功能以及相应实现方法的介绍：

1. 显示：程序进入之后，会按照数据库内的数据显示在表格控件内（控件为 DataGridview），分别显示通讯录用户的各项信息。

该控件具有排序功能，点击某一列的列表头，可将表格以该列字典序进行排列，点击一次为升序，再点击一次变为降序，之后每次点击可在升序降序之间切换。

关键代码：图 4-1 的代码将数据库文件的数据转化成 DataTable（可理解为 string 类型的二维数组），并显示在上述控件内。

```
OleDbConnection conn;
OleDbDataAdapter add;
DataTable dt;
conn = new OleDbConnection(@"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" + @"Database.mdb");
add = new OleDbDataAdapter("select * from 通讯录", conn);
dt = new DataTable();
add.Fill(dt);
dataGridView1.DataSource = dt;
```

图 4-1 数据显示关键代码

2. 添加用户：菜单栏点击“数据处理”，进入后选择“增加成员”分栏，在对应位置的框框内输入相应数据，最后点击“添加用户”按钮，系统提示“添加成功!”，即实现添加功能。

关键代码：

```
int r = dt.Rows.Count;
dt.Rows.Add();//添加行
dt.Rows[r][0] = textBox1.Text;
dt.Rows[r][1] = comboBox1.Text;
dt.Rows[r][2] = textBox2.Text;
dt.Rows[r][3] = comboBox2.Text;
dt.Rows[r][4] = textBox3.Text;
dt.Rows[r][5] = textBox4.Text;
dt.Rows[r][6] = textBox5.Text;
dt.Rows[r][7] = textBox6.Text;
dt.Rows[r][8] = textBox7.Text;
dt.Rows[r][9] = textBox8.Text;
dt.Rows[r][10] = dateTimePicker1.Value.Year.ToString()
    + "/" + dateTimePicker1.Value.Month.ToString()
    + "/" + dateTimePicker1.Value.Day.ToString();
OleDbCommandBuilder cmdb = new OleDbCommandBuilder(add);
add.Update(dt);//更新进数据库
MessageBox.Show("添加成功！");
```

图 4-2 添加用户关键代码

3. 查询/修改/删除用户：程序将这三个功能整合在“查询/修改/删除成员”分栏，同样在菜单栏点击“数据处理”，然后选择“查询/修改/删除成员”分栏，在查询姓名中输入想要查询的用户的姓名，输入完毕后点击回车键，系统搜索对应用户（若不存在该用户则提示“查无此人”），并在对应位置显示该用户的相关信息。

修改信息：显示出该用户的相关信息后，可直接在对应位置的框框内修改信息，修改完成后点击“修改信息”按钮，即可修改对应用户的信息。

删除用户：上述查询功能查询到相应用户之后，点击“删除用户”按钮，系统提示“是否确定删除”，确定删除后即可将对应用户删除。

关键代码：

查询：

```
r = -1;
for (int i = 0; i < dt.Rows.Count; i++)
{
    if (dt.Rows[i][2].ToString() == textBox16.Text) // 判断名字是否相同
    {
        r = i;//记录该用户在datatable的行位置
        break;
    }
}
```

图 4-3 查询用户关键代码

修改：

```
try
{
    dt.Rows[r][1] = comboBox4.Text;//修改各项信息
    dt.Rows[r][3] = comboBox3.Text;
    dt.Rows[r][4] = textBox14.Text;
    dt.Rows[r][5] = textBox13.Text;
    dt.Rows[r][6] = textBox12.Text;
    dt.Rows[r][7] = textBox11.Text;
    dt.Rows[r][8] = textBox10.Text;
    dt.Rows[r][9] = textBox9.Text;
    dt.Rows[r][10] = dateTimePicker2.Value.Year.ToString() + "/"
        + dateTimePicker2.Value.Month.ToString() + "/"
        + dateTimePicker2.Value.Day.ToString();
    OleDbCommandBuilder cmdb = new OleDbCommandBuilder(add);
    add.Update(dt);//保存到数据库中
    MessageBox.Show("修改成功！");
}
```

图 4-4 修改信息关键代码

删除：

```
DialogResult dr = MessageBox.Show("即将删除用户【" + textBox16.Text + "】。" +
    "\n确定删除？", "提示:", MessageBoxButtons.OKCancel, MessageBoxIcon.Information);
if (dr == DialogResult.OK)   //如果单击“是”按钮
{
    try
    {
        dataGridView1.DataSource = dt;
        dataGridView1.Rows.RemoveAt(r);//删除行
        dt = (DataTable)dataGridView1.DataSource;
        OleDbCommandBuilder cmdb = new OleDbCommandBuilder(add);
        add.Update(dt);//保存进数据库
        MessageBox.Show("删除成功！");
```

图 4-5 删除用户关键代码

（额外功能）数据统计：该程序额外添加了统计通讯录用户信息数据，并绘制成图标形式的功能，在菜单栏点击“数据处理”，然后选择“数据统计”分栏即可进入对应模块。

关键代码：

```
if (dt.Rows[i][3].ToString() == "男") num2[0]++;
else num2[1]++;
```

图 4-6 统计关键代码

```
for (int i = 0; i < 2; i++)
{
    chart2.Series[0].Points[i].YValues[0] = num2[i];
}
```

图 4-7 图表显示关键代码

（额外功能）导入新的通信录：通过 python 文件可生成随机的通讯录（即 data.csv 文件），程序菜单中点击“更新数据”，系统提示“是否将 csv 文件数据导入数据库中”（会删除原有数据，导入新的数据，即用新的通讯录代替旧的通讯录），选择“确定”后，程序进行原有数据的删除和新的通讯录的导入，约 5-10 秒提示“更新成功！”，之后程序将会显示新的通信录数据。

关键代码：

```
DialogResult dr = MessageBox.Show("即将把csv文件数据更新入Access数据库。" +
    "\n确定更新？", "提示:", MessageBoxButtons.OKCancel, MessageBoxIcon.Information);
if (dr == DialogResult.OK)   //如果单击“是”按钮
{
    try
    {
        conn = new OleDbConnection(@"Provider=Microsoft.Jet.OLEDB.4.0;Data Source="
            + @"Database.mdb");
        add = new OleDbDataAdapter("select * from 通讯录", conn);
        dt = new DataTable();
        add.Fill(dt);
        for (int i = 0; i < dt.Rows.Count; i++)
        {
            dataGridView1.Rows.Remove(dataGridView1.Rows[0]);
        }
        dt = (DataTable)dataGridView1.DataSource;
        OleDbCommandBuilder cmdb = new OleDbCommandBuilder(add);
        add.Update(dt);//删除所有数据
        dt2 = new DataTable();
        dt2 = OpenCSV("data.csv");
        cmdb = new OleDbCommandBuilder(add);
        cmdb.QuotePrefix = "[";
        cmdb.QuoteSuffix = "]";
        add.Update(dt2);//导入新的数据
        MessageBox.Show("更新成功！");
```

图 4-8 导入新通讯录关键代码

【测试数据、结果及分析】

C++部分：

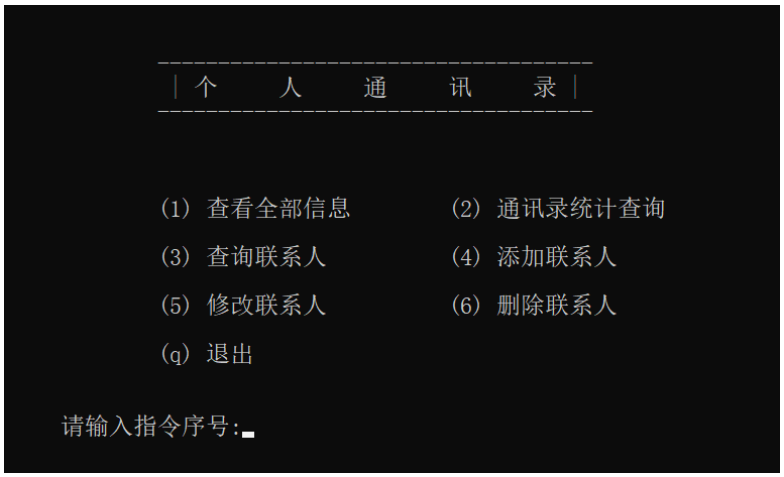


图 5-1 主界面

图 5-2 查看通讯录全部信息（截图仅为一部分）

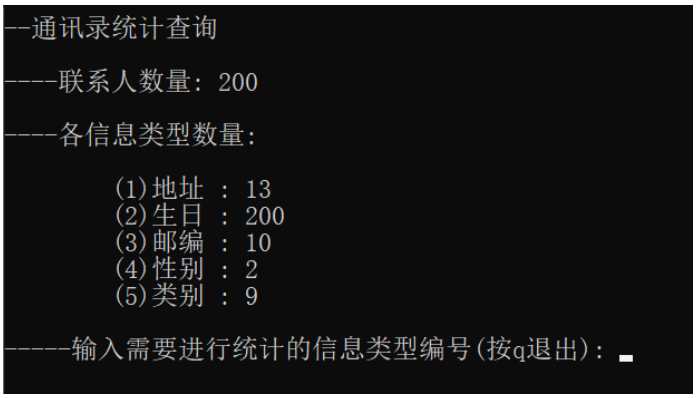


图 5-3 通讯录统计查询，显示 5 个类型信息的数量

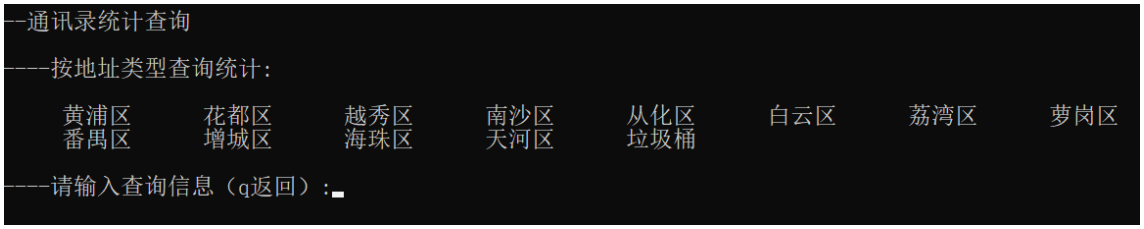


图 5-4 输入 1 后显示按地址查询

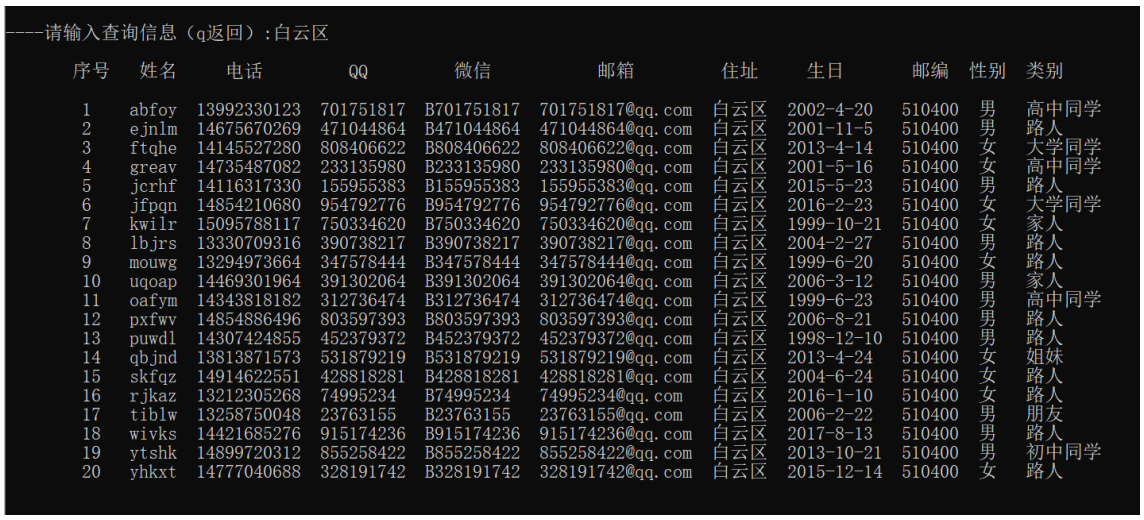


图 5-5 输入白云区后显示所有住址为白云区的联系人信息

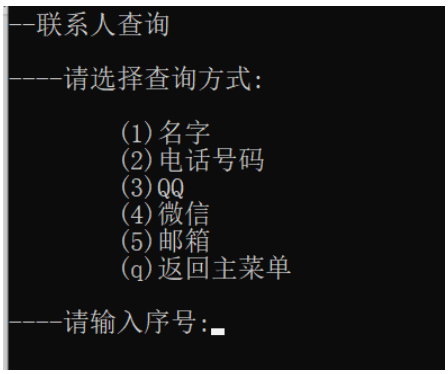


图 5-6 查询联系人，显示查询方式

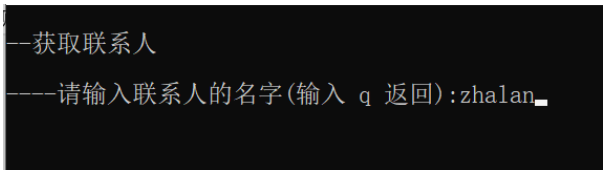


图 5-7 输入 1，按姓名查找联系人

图 5-8 成功输入联系人姓名，显示相关信息

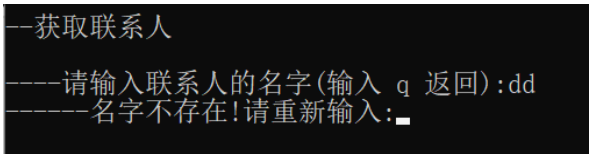


图 5-9 输入信息不存在，查询失败

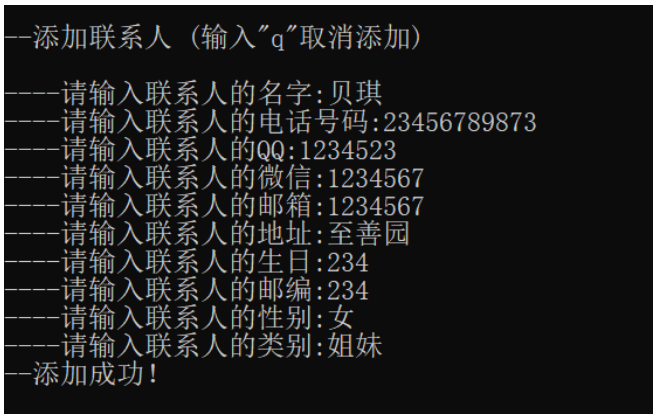


图 5-10 添加联系人，成功

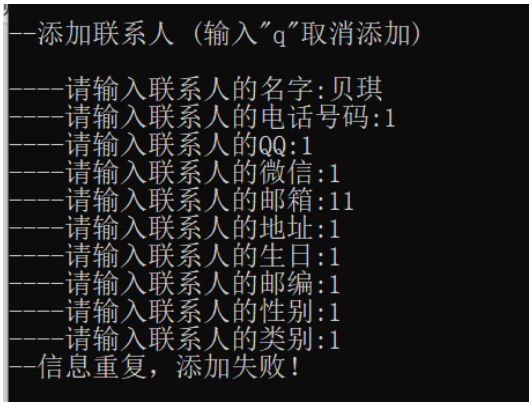


图 5-11 当输入信息的前五个与已有信息重复时，添加失败

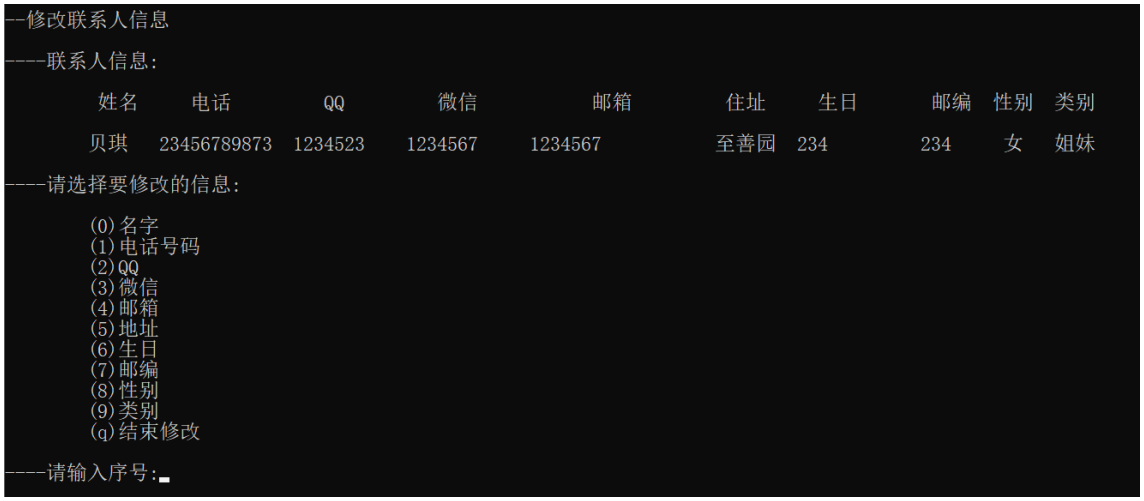


图 5-12 修改联系人的界面

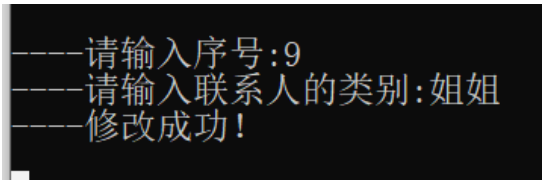


图 5-13 修改类别信息

```
--修改联系人信息
----联系人信息:
        姓名      电话          QQ          微信          邮箱          住址      生日        邮编  性别  类别
        贝琪    23456789873   1234523     1234567     1234567          至善园   234         234     女    姐姐
----请选择要修改的信息:

        (0)名字
        (1)电话号码
        (2)QQ
        (3)微信
        (4)邮箱
        (5)地址
        (6)生日
        (7)邮编
        (8)性别
        (9)类别
        (q)结束修改
----请输入序号:
```

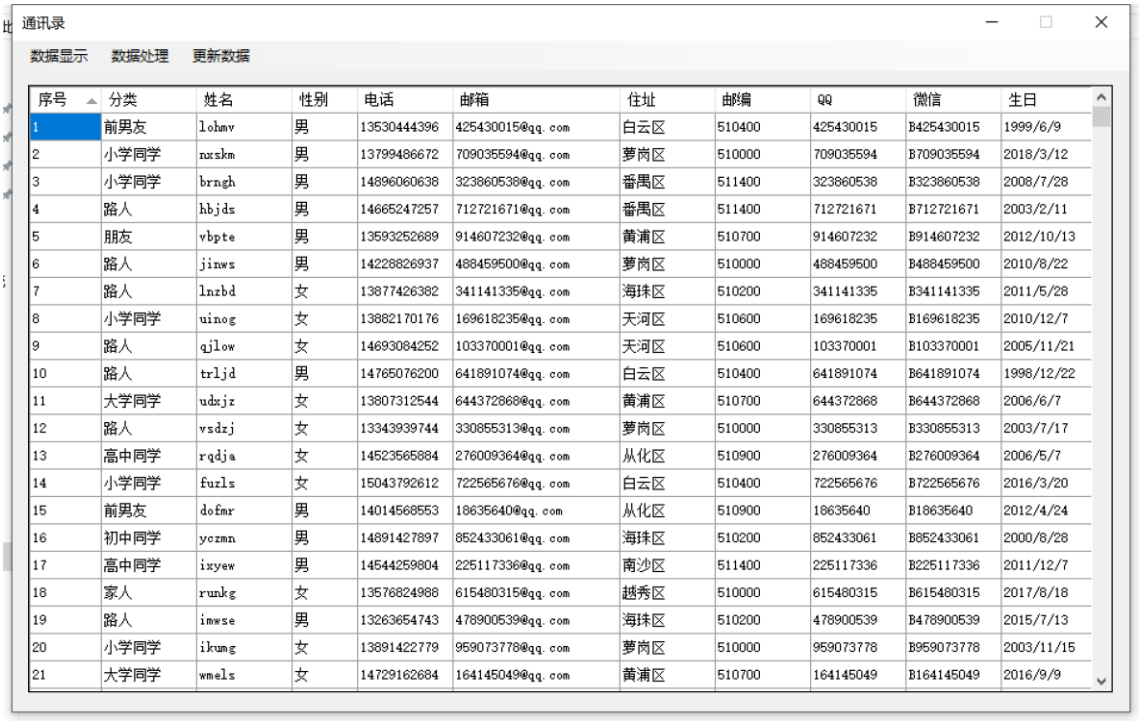
图 5-14 修改 0.5s 后页面刷新，信息修改成功

```
--删除联系人
----联系人信息:
        姓名      电话          QQ          微信          邮箱          住址      生日        邮编  性别  类别
       zhalan 12345678987  1726381862 1212121212  123456789@qq.com  垃圾桶  2000-5-18   510000  女    姐妹
  确定删除？（Y/N）

----删除成功!
```

图 5-15 删除联系人，通过姓名查找到后输入 Y 确认删除

C#部分：



序号	分类	姓名	性别	电话	邮箱	住址	邮编	QQ	微信	生日
1	前男友	lohmv	男	13530444396	425430015@qq.com	白云区	510400	425430015	B425430015	1999/6/9
2	小学同学	nxskm	男	13799486672	709035594@qq.com	萝岗区	510000	709035594	B709035594	2018/3/12
3	小学同学	brngh	男	14896060638	323860538@qq.com	番禺区	511400	323860538	B323860538	2008/7/28
4	路人	hbjds	男	14665247257	712721671@qq.com	番禺区	511400	712721671	B712721671	2003/2/11
5	朋友	vbpte	男	13593252689	914607232@qq.com	黄浦区	510700	914607232	B914607232	2012/10/13
6	路人	jinws	男	14228826937	488459500@qq.com	萝岗区	510000	488459500	B488459500	2010/8/22
7	路人	lnzbd	女	13877426382	341141335@qq.com	海珠区	510200	341141335	B341141335	2011/5/28
8	小学同学	uinog	女	13882170176	169618235@qq.com	天河区	510600	169618235	B169618235	2010/12/7
9	路人	qjlow	女	14693084252	103370001@qq.com	天河区	510600	103370001	B103370001	2005/11/21
10	路人	trljd	男	14765076200	641891074@qq.com	白云区	510400	641891074	B641891074	1998/12/22
11	大学同学	udxjz	女	13807312544	644372868@qq.com	黄浦区	510700	644372868	B644372868	2006/6/7
12	路人	vsdzj	女	13343939744	330855313@qq.com	萝岗区	510000	330855313	B330855313	2003/7/17
13	高中同学	rqdja	女	14523565884	276009364@qq.com	从化区	510900	276009364	B276009364	2006/5/7
14	小学同学	fuzls	女	15043792612	722565676@qq.com	白云区	510400	722565676	B722565676	2016/3/20
15	前男友	dofmr	男	14014568553	18635640@qq.com	从化区	510900	18635640	B18635640	2012/4/24
16	初中同学	yczmn	男	14891427897	852433061@qq.com	海珠区	510200	852433061	B852433061	2000/8/28
17	高中同学	ixyew	男	14544259804	225117336@qq.com	南沙区	511400	225117336	B225117336	2011/12/7
18	家人	runkg	女	13576824988	615480315@qq.com	越秀区	510000	615480315	B615480315	2017/8/18
19	路人	imwse	男	13263654743	478900539@qq.com	海珠区	510200	478900539	B478900539	2015/7/13
20	小学同学	ikumg	女	13891422779	959073778@qq.com	萝岗区	510000	959073778	B959073778	2003/11/15
21	大学同学	wmels	女	14729162684	164145049@qq.com	黄浦区	510700	164145049	B164145049	2016/9/9

图 6-1 数据显示

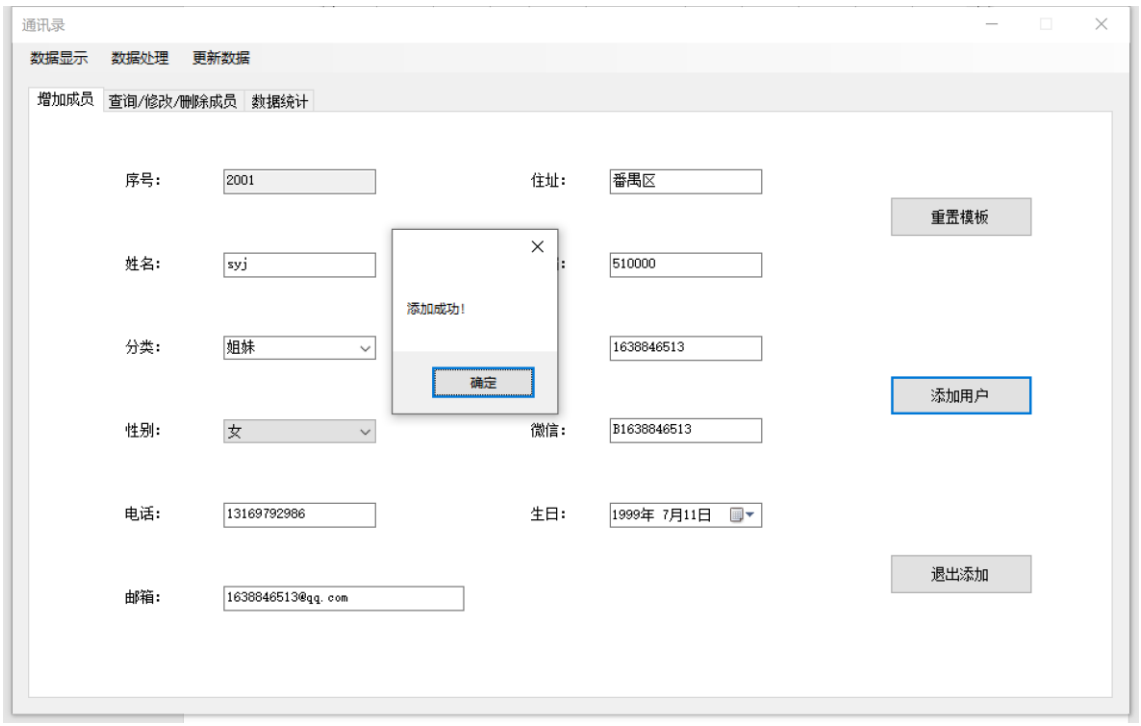


图 6-2 添加用户

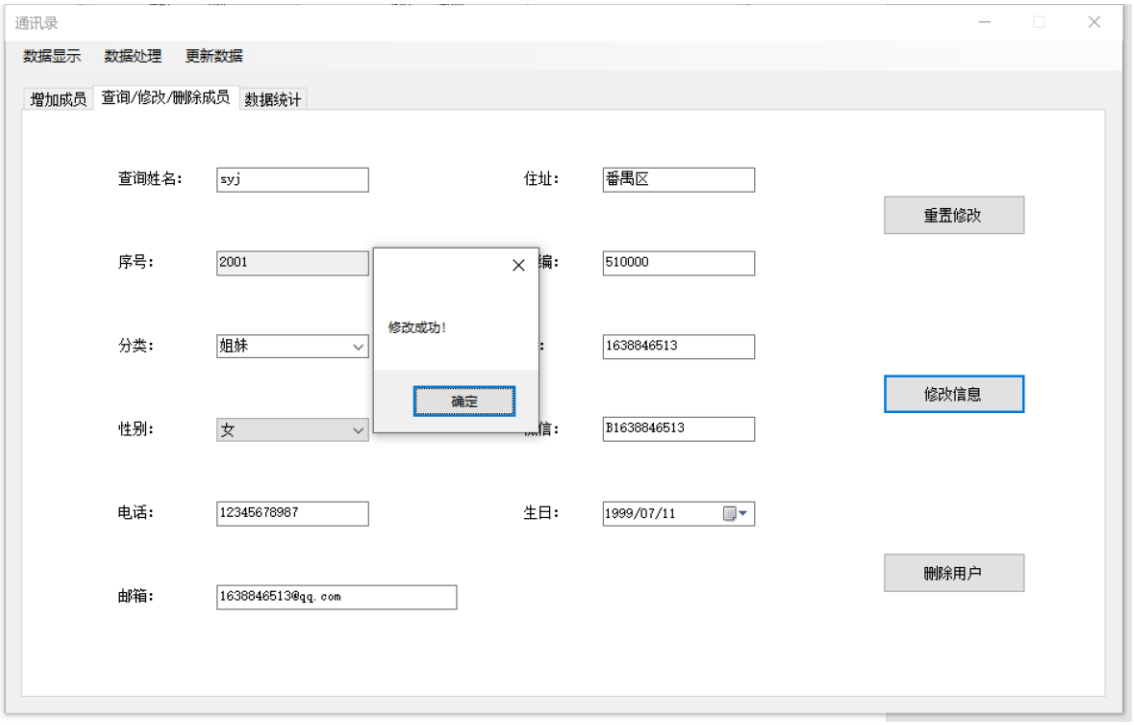


图 6-3 修改信息

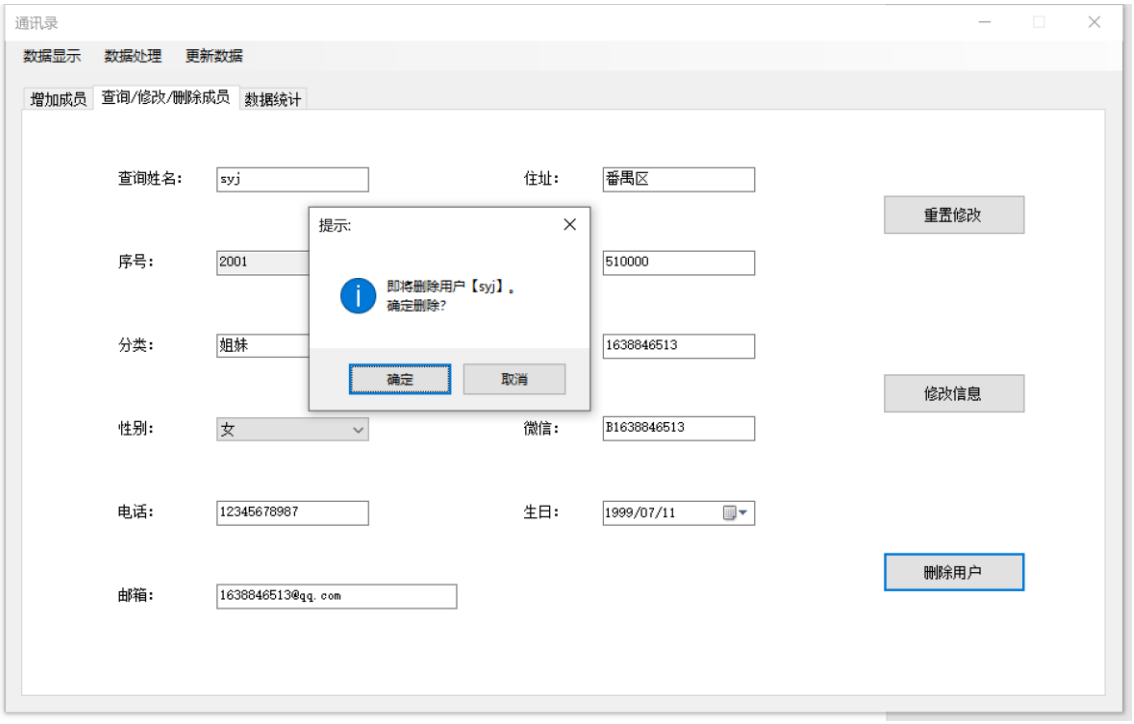


图 6-4 删除用户

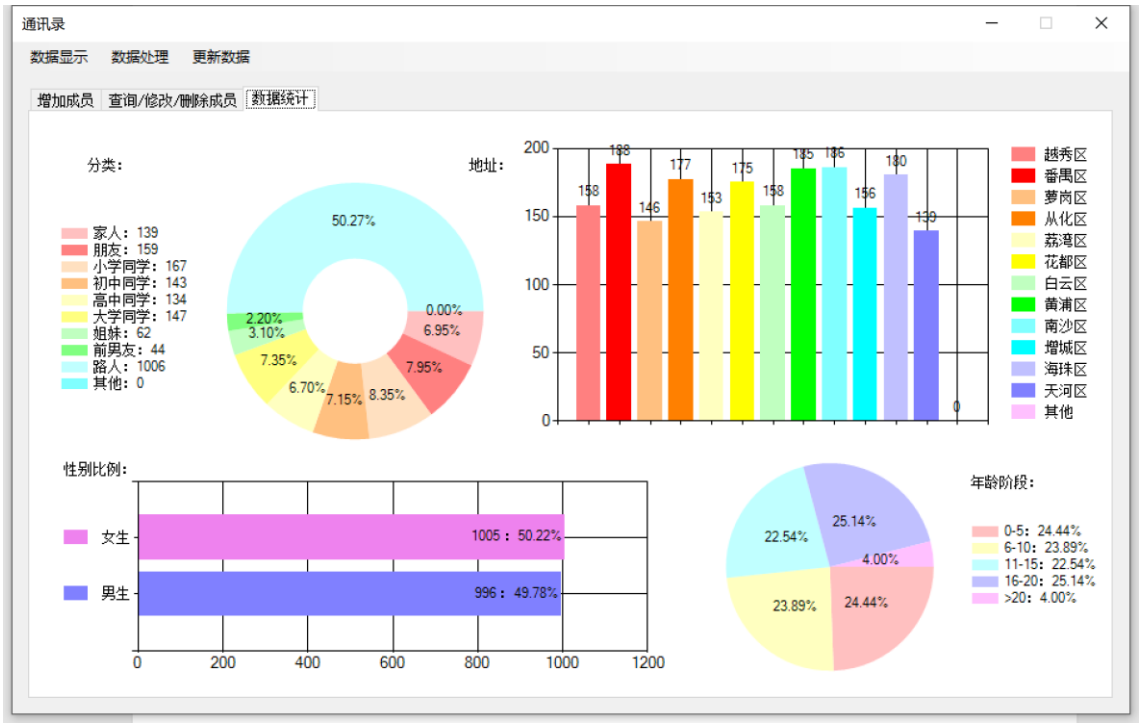


图 6-5 数据统计

【分工、贡献%、自我评分】

分工：

宋渝杰： C#程序的设计和实现，C#实验报告的编写，30%，99 分

廖家源： C++程序的设计和实现，30%，99 分

刘依澜： C++实验报告中通讯录应用和测试数据部分的编写，20%，99 分

缪贝琪： C++实验报告中 AVL 树算法部分的编写，20%，99 分

【项目总结】

C++：这次的项目五第一眼看上去只需要一个 vector<node>即可实现以上所有操作，但是考虑到最近学习的 AVL 树，我们兴趣满满地计划用 AVL 树来实现这个项目。虽然过程十分艰辛，AVL 树的平衡操作花费了大量的时间和精力，但好在最后还是成功实现了该项目，并把搜索的时间复杂度降到了 O(logn)。

虽然后来发现 stl 库中存在 map 这一容器，底层就为红黑树结构，搜索时间复杂度也为 O(logn)，下次可以考虑直接使用 map 来实现，可以减少一些精力的消耗。

C#：在这次的项目五中，虽然看上去只是要求实现一个普通的增、删、改、查、显示的功能的程序，但是我们结合了数据库技术和文件处理技术，实现一个小型的数据处理程序，也算是提高了一下程序的意义吧。

总结：通过 C++部分我们学习并熟悉了 AVL 树，从而提高了程序的效率，而在 C#中结合了数据库技术和文件处理技术，实现了一个小型的数据处理程序。而且在模拟仿真的过程中，我们特意表现出了其先查询后修改或删除的真实特征，也加入了其他操作如统计功能。但由于时间比较紧迫，我们这次项目还是存在一些小瑕疵，以后会努力改进。

【程序清单】

C++部分：

AVLTree.hpp:

```
#ifndef MYAVL
#define MYAVL
#include<iostream>
#include<string>
#define TREENUM 5
```

```
#define INFONUM 10
using namespace std;

enum Type
{NAME,PHONE,QQ,WECHAT,EMAIL,ADDRESS,BIRTHDAY,POSTCODE,SEX,TYPE};
// 查询方式  前 5 个
// 注意 枚举类型 中作为 tree 索引的项放在前面

struct Node{
    string information[INFONUM];
    int balance;  // 平衡因子取值 -2 -1 0 1 2 ( 左子树高度 - 右子树高度)
    Node* left[TREENUM];
    Node* right[TREENUM];
    Node* parent[TREENUM];
    // 构造函数需要把所有的指针置为 NULL
    Node(){
        for(int i=0;i<INFONUM;i++) information[i]="undefined";
        for(int i=0;i<TREENUM;i++){
            left[i] = NULL;
            right[i] = NULL;
            parent[i] = NULL;
        }
        balance = 0;
    }
    Node(string in[INFONUM]){
        for(int i=0;i<INFONUM;i++) information[i] = in[i];
        for(int i=0;i<TREENUM;i++){
            left[i] = NULL;
            right[i] = NULL;
            parent[i] = NULL;
        }
        balance = 0;
    }
};

extern Node* Root[TREENUM];
```

```
bool Insert(string info[INFONUM]); // 必须给全 node 的所有信息
bool Remove(Node*node); // 使用一种信息查询、删除全部

// 查找内容
Node* search(string info,Type tp);
//插入
void insert(Node* node,Type tp);
//删除
void remove(Node*node,Type tp);
// 前序遍历
void preorder(Node* node,Type tp);

//旋转
Node* turnRight(Node* node,Type tp);
Node* turnLeft(Node* node,Type tp);
Node* turnRightLeft(Node* node,Type tp);
Node* turnLeftRight(Node* node,Type tp);
//节点高度
int height(Node* node,Type tp);
// 使树恢复平衡
void setBalance(Node* node,Type tp);
void rebalance(Node* node,Type tp);

//////////////////////////////////////////////////

// 返回值表示是否插入成功，当信息已经存在时不允许插入
bool Insert(string info[INFONUM]){
    bool isExist = false;
    for(int i=0;i<TREENUM;i++){
        if(search(info[i],Type(i))!=NULL){
            isExist = true;
            break;
        }
    }
    if(isExist) return false; // 信息已经存在，不允许添加
    Node* tmp = new Node(info);
```

```
    for(int i=0;i<TREENUM;i++)
        insert(tmp,Type(i));
    return true;
}

// 返回值表示是否移除成功（tree 中是否有这个信息)
bool Remove(Node*node){
    if(node==NULL) return false;
    for(int i=0;i<TREENUM;i++) remove(node,Type(i));
    delete node;
    return true;
}

Node* search(string info,Type tp){
    Node* node = Root[tp];
    while(node && node->information[tp]!=info){
        if(node->information[tp]<info){
            node = node->right[tp];
        }else{
            node = node->left[tp];
        }
    }
    return node;
}

// 在内部不申请空间,统一管理申请
// 配合 search 使用，必须确保树中还没有这些信息
// 能够进入这个函数就说明 tree 里面不存在这些信息
void insert(Node* node,Type tp){
    if(Root[tp]==NULL){
        Root[tp] = node;
        return ;
    }else{
        Node*tmp = Root[tp];
        while(1){
            if(tmp->information[tp]>node->information[tp]){
```

```
                    if(tmp->left[tp]!=NULL) tmp=tmp->left[tp];
                    else{
                        tmp->left[tp] = node;
                        node->parent[tp] = tmp;
                        rebalance(tmp,tp);
                        return;
                    }
                }else{
                    if(tmp->right[tp]!=NULL) tmp=tmp->right[tp];
                    else{
                        tmp->right[tp] = node;
                        node->parent[tp] = tmp;
                        rebalance(tmp,tp);
                        return;
                    }
                }
            }
        }
}

// 要求 node 必须存在 tree 中
void remove(Node*node,Type tp){
    if(node->left[tp] and node->right[tp]){
        // 选择右边最小值
         Node*tmp =   node->right[tp]; // tmp 是替换 node 位置的结点
         bool flag = false; // 判断替换 node 的对象是否就是 node 的右子树
         while(tmp->left[tp]!=NULL){
            tmp = tmp->left[tp];
            flag = true;
         }
         if(flag){
            // 此时 tmp 不是 node 的右子节点
            tmp->parent[tp]->left[tp] = NULL;
            tmp->parent[tp] = node->parent[tp];
            tmp->left[tp] = node->left[tp];
            tmp->right[tp] = node->right[tp];
```

```
            //设置父节点的子节点指向
            if(tmp->parent[tp]->left[tp]==node) tmp->parent[tp]->left[tp] = tmp;
            else tmp->parent[tp]->right[tp] = tmp;
            //设置子节点的父节点指向
            if(node->left[tp]) node->left[tp]->parent[tp] = tmp;
            if(node->right[tp]) node->right[tp]->parent[tp] = tmp;

        }else{
            // 此时 tmp 是 node 的右子节点
            tmp->parent[tp]->right[tp] = NULL;
            //设置左子节点关系
            tmp->left[tp] = node->left[tp];
            if(node->left[tp]) node->left[tp]->parent[tp] = tmp;
            //设置父节点关系
            tmp->parent[tp] = node->parent[tp];
            if(node->parent[tp]){
                if(node->parent[tp]->left[tp]==node) node->parent[tp]->left[tp] = tmp;
                else node->parent[tp]->right[tp] = tmp;
            }
        }
        rebalance(tmp,tp);
        if(Root[tp]==node) Root[tp] = tmp;
    }else if(node->left[tp] and !node->right[tp]){
        //设置父子指针指向
        node->left[tp]->parent[tp] = node->parent[tp];
        if(node->parent[tp]!=NULL){
            if(node->parent[tp]->left[tp] == node) node->parent[tp]->left[tp] = node->left[tp];
            else node->parent[tp]->right[tp] = node->left[tp];
        }
        if(Root[tp]==node) Root[tp] = node->left[tp];
        rebalance(node->left[tp],tp);
    }else if(!node->left[tp] and node->right[tp]){
        node->right[tp]->parent[tp] = node->parent[tp];
        if(node->parent[tp]!=NULL){
            if(node->parent[tp]->left[tp] == node) node->parent[tp]->left[tp] = node->right[tp];
            else node->parent[tp]->right[tp] = node->right[tp];
```

```
            }
            if(Root[tp]==node) Root[tp] = node->right[tp];
            rebalance(node->right[tp],tp);
        }else{
            if(node==Root[tp]){
                Root[tp] = NULL;
            }else{
                if(node->parent[tp]->right[tp] == node) node->parent[tp]->right[tp] = NULL;
                else if(node->parent[tp]->left[tp] == node) node->parent[tp]->left[tp] = NULL;

                if(node->parent[tp]) rebalance(node->parent[tp],tp);
            }
        }
}

void preorder(Node*node,Type tp){
    if(node){
        cout<<node->information[tp]<<" ";
        if(node->left[tp]) preorder(node->left[tp],tp);
        if(node->right[tp]) preorder(node->right[tp],tp);
    }
}


Node* turnRight(Node* node,Type tp){
    Node* tmp = node->left[tp];
    if(node->parent[tp]!=0){    //   将 node 的父节点的子节点置为 node 的左子节点
        if(node->parent[tp]->right[tp]==node){
            node->parent[tp]->right[tp]=tmp;
        }else{
            node->parent[tp]->left[tp]=tmp;
        }
    }
    tmp->parent[tp]=node->parent[tp];
    node->parent[tp]=tmp;
    node->left[tp]=tmp->right[tp];
```

```
    if(node->left[tp]!=0)
        node->left[tp]->parent[tp]=node;
    tmp->right[tp]=node;
    setBalance(node,tp);
    setBalance(tmp,tp);
    return tmp;
}

Node* turnLeft(Node* node,Type tp){
    Node* tmp=node->right[tp];
    if(node->parent[tp]!=0){
        if(node->parent[tp]->right[tp]==node){
            node->parent[tp]->right[tp]=tmp;
        }else{
            node->parent[tp]->left[tp]=tmp;
        }
    }
    tmp->parent[tp]=node->parent[tp];
    node->parent[tp]=tmp;
    node->right[tp]=tmp->left[tp];
    tmp->left[tp]=node;
    if(node->right[tp]!=0)
        node->right[tp]->parent[tp]=node;
    setBalance(node,tp);
    setBalance(tmp,tp);
    return tmp;
}

Node* turnRightLeft(Node* node,Type tp){
    node->right[tp] = turnRight(node->right[tp],tp);
    return turnLeft(node,tp);
}

Node* turnLeftRight(Node* node,Type tp){
    node->left[tp] = turnLeft(node->left[tp],tp);
    return turnRight(node,tp);
```

```
}

int height(Node* node,Type tp){
    if(node==0){
        return 0;
    }
    int rightheight=height(node->right[tp],tp);
    int leftheight=height(node->left[tp],tp);
    return rightheight > leftheight ? (rightheight+1) : (leftheight+1);
}
void setBalance(Node* node,Type tp){
    if(node)
        node->balance=height(node->right[tp],tp)-height(node->left[tp],tp);
}

void rebalance(Node* node,Type tp){
    setBalance(node,tp);
    if(node->balance== -2){                    //如果左子树比右子树高 2 层，需要通过旋转来重新平衡
        if(node->left[tp]->balance<=0){        //如果左子节点的左子树比右子树高，则进行一次右旋；如果左子节点的右子树比左子树高，则先进行左旋，再进行右旋。
            node=turnRight(node,tp);
        }else{
            node=turnLeftRight(node,tp);
        }
    }else if(node->balance==2){                //如果右子树比左子树高 2 层，需要通过旋转来重新平衡
        if(node->right[tp]->balance>=0){       //如果右子节点的右子树比左子树高，则进行一次左旋；如果右子节点的右子树比左子树低，则先进行右旋，再进行左旋。
            node=turnLeft(node,tp);
        }else{
            node=turnRightLeft(node,tp);
        }
    }
    if(node->parent[tp]){
        rebalance(node->parent[tp],tp);
```

```
    }else{
        Root[tp]=node; //  后面可以将 Root[tp]设为全局变量
    }
}
#endif
```

V3.cpp:

```
#include<iostream>
#include<vector>
#include<windows.h>
#include<conio.h>
#include<fstream>
#include<string>
#include<time.h>
#include<algorithm>
#include<iomanip>
#include"AVLTree.hpp"

using namespace std;

const string Str[INFONUM] = {"名字","电话号码","QQ","微信","邮箱","地址","生日","邮编","性别","类别"};
Node* Root[TREENUM];
int Count;   // 通讯录中记录的人数
vector<string> Class[INFONUM-TREENUM]; // 用于分类查找

void Menu();         // 输出主菜单
void Initial();  // 程序启动的初始化操作
void ShowALL();   // 显示所有人信息
void FindMember();      //查找显示对象
void AddMember();       //添加对象
void ResMember(); //修改对象
void DelMember(); //删除对象
void Statistics(); //  统计通讯录信息
```

```
// 辅助函数，用于实现特定功能
char GetMember(Node*&node);  //将多个函数的共同部分独立成函数，减少代码量
void ShowSpecialTraversal(Node* node,int& cou,const int& ind,const string& info); // 筛选显示
void ShowTraversal(Node* node,int& cou); //// 遍历树将信息输出显示
void WriteTraversal(Node* node,ofstream&out); // 遍历树将信息写入文件
void FileRead();    // 读取文件信息
void FileWrite();    // 修改并关闭文件
void FreeSpace(Node* node);// 释放空间
bool Find(Node* node,int ind,const string& info);

int main(){
    Initial();
    while(1){
        Menu();
        char x;
        do{
            x=_getch();
        } while((x<'1' or x>'6') and x!='q' and x!='Q');
        if(x=='q' or x=='Q'){
            FileWrite();
            FreeSpace(Root[0]);
            return 0;
        }
        system("cls");
        switch(x){
            case '1':
                ShowALL();
                break;
            case '2':
                Statistics();
                break;
            case '3':
                FindMember();
                break;
            case '4':
```

```
                AddMember();
                break;
            case '5':
                ResMember();
                break;
            case '6':
                DelMember();
                break;
        }
        system("cls");
    }
}

void Menu(){
    cout<<"\n\n";
    cout<<"\t\t\t\t\t----------------------------------"<<endl;
    cout<<"\t\t\t\t\t | 个      人      通      讯      录 |"<<endl;
    cout<<"\t\t\t\t\t----------------------------------"<<endl;
    cout<<"\n\n\n";
    cout<<"\t\t\t\t\t(1)  查看全部信息";
    cout<<"\t(2)  通讯录统计查询\n\n";
    cout<<"\t\t\t\t\t(3)  查询联系人";
    cout<<"\t\t(4)  添加联系人\n\n";
    cout<<"\t\t\t\t\t(5)  修改联系人";
    cout<<"\t\t(6)  删除联系人\n\n";
    cout<<"\t\t\t\t\t(q)  退出\n\n";
    cout<<"\n\t\t\t\t 请输入指令序号:";
}

void Initial(){
    Count=0;
    for(int i=0;i<TREENUM;i++) Root[i]=NULL;
    FileRead();
}

//////////////////////////////////////////////////////
```

```
void ShowALL(){
    cout<<"\n\t 通讯录信息:\n\n";
    if(Root[NAME]==NULL or Count==0){
        cout<<"\t\t\t 通讯录为空！"<<endl;
    }
    else{
        int cou = 1;
        cout<<"          序号    姓名      电话            QQ            微信                邮
箱            住址      生日          邮编   性别   类别\n\n";
        ShowTraversal(Root[0],cou);
    }
    cout<<"\n\n\n\t\t\t\t\t 按任意键返回...";
    _getch();
}

void Statistics(){
    ST1:
    cout<<"\n--通讯录统计查询\n\n";
    cout<<"----联系人数量: "<<Count<<endl<<endl;
    cout<<"----各信息类型数量:\n\n";
    for(int i=0;i<INFONUM-TREENUM;i++){
        cout<<"\t("<<i+1<<")"<<Str[i+TREENUM]<<" : "<<Class[i].size()<<endl;
    }
    cout<<"\n-----输入需要进行统计的信息类型编号(按 q 退出): " ;
    char x;
    do{
        x=_getch();
    } while((x<'1' or x>'5') and x!='q' and x!='Q');
    if(x=='q' or x=='Q'){
        return ;
    }
    else{
        system("cls");
        int ind = x-'1';
        cout<<"\n--通讯录统计查询\n\n";
```

```
        cout<<"----按"<<Str[ind+TREENUM]<<"类型查询统计:\n\n";
        for(int i=0;i<Class[ind].size();i++){
            if(i%8==0) cout<<"          ";
            cout<<left<<setw(10)<<Class[ind][i]<<"    ";
            if((i+1)%8==0) cout<<endl;
        }
        string info;
        cout<<"\n\n----请输入查询信息（q 返回）:";
        while(1){
            cin>>info;
            if(info=="q" or info=="Q"){
                system("cls");
                goto ST1;
            }
            if(find(Class[ind].begin(), Class[ind].end(), info) == Class[ind].end()){
                cout<<"----输入错误，重新输入:";
            }else break;
        }
        bool flag = Find(Root[0],ind+TREENUM,info);
        if(flag){
            cout<<"\n            序号      姓名          电话              QQ              微信
邮箱              住址        生日          邮编    性别    类别\n\n";
            int cou=1;
            ShowSpecialTraversal(Root[NAME],cou,ind+TREENUM,info);
        }
        else{
            cout<<"---信息已被删除或修改"<<endl; // 两下子也接受不了吧
        }
    }
    cout<<"\n\n\n\t\t\t\t\t 按任意键返回...";
    _getch();
    system("cls");
    goto ST1;
}

void FindMember(){
```

```
    FM1:
    cout<<"\n--联系人查询\n\n";
    Node* obj = NULL;
    char x = GetMember(obj);
    if(x=='q') return;
    if(x!='n'){
        system("cls");
        if(x=='r'){
            goto FM1;
        }
        else{
            cout<<"\n\n";
            cout<<"----联系人信息:\n\n";
            cout<<"\n        姓名    电话        QQ        微信        邮
箱        住址    生日      邮编  性别  类别\n\n";
            cout<<"\t";
            cout<<left<<setw(7)<<obj->information[0];
            cout<<left<<setw(13)<<obj->information[1];
            cout<<left<<setw(11)<<obj->information[2];
            cout<<left<<setw(12)<<obj->information[3];
            cout<<left<<setw(18)<<obj->information[4];
            cout<<left<<setw(8)<<obj->information[5];
            cout<<left<<setw(12)<<obj->information[6];
            cout<<left<<setw(8)<<obj->information[7];
            cout<<left<<setw(5)<<obj->information[8];
            cout<<left<<setw(6)<<obj->information[9];
            cout<<endl;
        }
    }
    cout<<"\n\n\n\t\t\t\t\t 按任意键返回...";
    _getch();
}

void AddMember(){
    cout<<"\n--添加联系人  (输入\"q\"取消添加)\n\n";
    string info[INFONUM];
```

```
    string tmp;
    for(int i=0;i<INFONUM;i++){
        cout<<"----请输入联系人的"<<Str[i]<<":";
        cin>>tmp;
        if(tmp != "q") info[i]=tmp;
        else return;
    }
    bool success = Insert(info);
    if(success){
        for(int j=0;j<INFONUM-TREENUM;j++){
            // 查找类型是否已经存在
            if(find(Class[j].begin(), Class[j].end(), info[j]) == Class[j].end()){
                Class[j].push_back(info[j+TREENUM]);
            }
        }
        cout<<"--添加成功！\n";
        Count++; // 数量++
    }
    else{
        cout<<"--信息重复，添加失败！\n";
    }
    cout<<"\n\n\n\t\t\t\t\t 按任意键返回...";
    _getch();
}

void ResMember(){
    RM1:
    cout<<"\n--修改联系人信息\n\n";
    Node* obj = NULL;
    char x = GetMember(obj);
    if(x=='q') return;
    if(x!='n'){
        system("cls");
        if(x=='r'){
            goto RM1;
        }
```

```
else{
    bool changed = false; // 标记是否已经有修改
    RM2:
    cout<<"\n--修改联系人信息\n\n";
    cout<<"----联系人信息:\n";
    cout<<"\n 姓名 电话 QQ 微信 邮箱 住址 生日 邮编 性别 类别\n\n";
    cout<<"\t";
    cout<<left<<setw(7)<<obj->information[0];
    cout<<left<<setw(13)<<obj->information[1];
    cout<<left<<setw(11)<<obj->information[2];
    cout<<left<<setw(12)<<obj->information[3];
    cout<<left<<setw(18)<<obj->information[4];
    cout<<left<<setw(8)<<obj->information[5];
    cout<<left<<setw(12)<<obj->information[6];
    cout<<left<<setw(8)<<obj->information[7];
    cout<<left<<setw(5)<<obj->information[8];
    cout<<left<<setw(6)<<obj->information[9];
    cout<<endl;
    cout<<"\n----请选择要修改的信息:\n\n" ;
    for(int i=0;i<INFONUM;i++)
        cout<<"\t("<<i<<")"<<Str[i]<<endl;
    cout<<"\t(q)结束修改\n";
    cout<<"\n----请输入序号:" ;
    do{
        x=_getch();
    }while((x<'0' or x>'9') and x!='q' and x!='Q');
    int index = x-'0';
    if(x=='q' or x=='Q'){
        if(changed){  // 修改方式是先 删除 再 插入 ,不能直接修改，因为涉及树要重新构造
            string tmp[INFONUM] = obj->information;
            Remove(obj);
            Insert(tmp);
        }
        return;
```

```
                }
                else{
                    cout<<x<<endl;
                    cout<<"----请输入联系人的"<<Str[index]<<":";
                    string info;
                    cin>>info;
                    if(info!=obj->information[index]){
                        obj->information[index] = info;
                        changed = true;
                    }
                    cout<<"----修改成功！"<<endl;
                    Sleep(500);
                    system("cls");
                    goto RM2;
                }
            }
        }
        cout<<"\n\n\n\t\t\t\t\t 按任意键返回...";
        _getch();
}

void DelMember(){
    DM1:
    cout<<"\n--删除联系人\n\n";
    Node* obj = NULL;
    char x = GetMember(obj);
    if(x=='q') return;
    if(x!='n'){
        system("cls");
        if(x=='r'){
            goto DM1;
        }
        else{
            cout<<"\n--删除联系人\n\n";
            cout<<"----联系人信息:\n";
            cout<<"\n        姓名    电话        QQ        微信        邮
```

```
箱              住址        生日          邮编    性别    类别\n\n";
                cout<<"\t";
                cout<<left<<setw(7)<<obj->information[0];
                cout<<left<<setw(13)<<obj->information[1];
                cout<<left<<setw(11)<<obj->information[2];
                cout<<left<<setw(12)<<obj->information[3];
                cout<<left<<setw(18)<<obj->information[4];
                cout<<left<<setw(8)<<obj->information[5];
                cout<<left<<setw(12)<<obj->information[6];
                cout<<left<<setw(8)<<obj->information[7];
                cout<<left<<setw(5)<<obj->information[8];
                cout<<left<<setw(6)<<obj->information[9];
                cout<<endl;
                cout<<"\n    确定删除？（Y/N）"<<endl;
                char y;
                do{
                    y=_getch();
                }while(y!='Y' and y!='N');
                if(y == 'Y'){
                    Remove(obj);
                    Count--;
                    cout<<"\n\n----删除成功！"<<endl;
                }
                else{
                    cout<<"\n\n----取消删除~"<<endl;
                }
            }
        }
        cout<<"\n\n\n\t\t\t\t\t 按任意键返回...";
        _getch();
}

//////////////////////////////////////////////////////////

/*
    返回值
```

```
        q: 返回主菜单
        n: 表示通讯录为空
        r: 回到函数的起始部位（刷新界面
        其他（1-5)：作为索引
*/
char GetMember(Node*&node){
    if(Root[NAME]==NULL or Count==0){
        cout<<"\t\t\t 通讯录为空！"<<endl;
        return 'n'; //  表示 tree 空
    }else{
        cout<<"----请选择查询方式:\n\n";
        for(int i=0;i<TREENUM;i++)
            cout<<"\t("<<i+1<<")"<<Str[i]<<endl;
        cout<<"\t(q)返回主菜单\n\n";
        cout<<"----请输入序号:";
        char x;
        do{
            x=_getch();
        }while((x<'1' or x>'5') and x!='q' and x!='Q');
        if(x=='q' or x=='Q') return 'q';
        else{
            cout<<x<<endl;
            system("cls");
            cout<<"\n--获取联系人\n\n";
            int index = x-'1';
            string info;
            cout<<"----请输入联系人的"<<Str[index]<<"(输入  q  返回):";
            GM1:
            cin>>info;
            if(info=="q" or info=="Q"){
                return 'r'; //  表示重新回到函数开始位置
            }else{
                node = search(info,Type(index));
                if(node==NULL){
                    cout<<"------"<<Str[index]<<"不存在!请重新输入:";
                    goto GM1;
```

```
                }else{
                    return x;// x 可能作为后面程序的索引
                }
            }
        }
    }
}

void ShowTraversal(Node* node,int& cou){
    if(node!=NULL){
        cout<<"\t"<<left<<setw(5)<<cou;
        cout<<left<<setw(7)<<node->information[0];
        cout<<left<<setw(13)<<node->information[1];
        cout<<left<<setw(11)<<node->information[2];
        cout<<left<<setw(12)<<node->information[3];
        cout<<left<<setw(18)<<node->information[4];
        cout<<left<<setw(8)<<node->information[5];
        cout<<left<<setw(12)<<node->information[6];
        cout<<left<<setw(8)<<node->information[7];
        cout<<left<<setw(5)<<node->information[8];
        cout<<left<<setw(6)<<node->information[9];
        cout<<endl;
        if(node->left[NAME]) ShowTraversal(node->left[NAME],++cou);
        if(node->right[NAME]) ShowTraversal(node->right[NAME],++cou);
    }
}

void ShowSpecialTraversal(Node* node,int& cou,const int& ind,const string& info){
    if(node!=NULL){
        if(node->information[ind]==info){
            cout<<"\t"<<left<<setw(5)<<cou;
            cout<<left<<setw(7)<<node->information[0];
            cout<<left<<setw(13)<<node->information[1];
            cout<<left<<setw(11)<<node->information[2];
            cout<<left<<setw(12)<<node->information[3];
            cout<<left<<setw(18)<<node->information[4];
```

```
            cout<<left<<setw(8)<<node->information[5];
            cout<<left<<setw(12)<<node->information[6];
            cout<<left<<setw(8)<<node->information[7];
            cout<<left<<setw(5)<<node->information[8];
            cout<<left<<setw(6)<<node->information[9];
            cout<<endl;
        }else{
            cou--;
        }
        if(node->left[NAME]) ShowSpecialTraversal(node->left[NAME],++cou,ind,info);
        if(node->right[NAME]) ShowSpecialTraversal(node->right[NAME],++cou,ind,info);
    }
}

void FileRead(){
    ifstream read("./information.txt",ios::in);
    int i=0;
    if(read){
        string info[INFONUM];
        while(read>>info[0]){
            for(int j=1;j<INFONUM;j++){
                read>>info[j];
                if(j>=TREENUM){
                    // 查找类型是否已经存在
                    if(find(Class[j-TREENUM].begin(), Class[j-TREENUM].end(), info[j]) ==
Class[j-TREENUM].end()){
                        Class[j-TREENUM].push_back(info[j]);
                    }
                }
            }
            bool success = Insert(info);
            i++  ;
            if(success) Count++;
        }
        read.close();
    }
```

```
}

void FileWrite(){
	ofstream write("./information.txt",ios::out);
	if(write.is_open()){
		WriteTraversal(Root[0],write);
		write.close();
	}
}

void WriteTraversal(Node* node,ofstream &write){
	if(write and node){
		for(int i=0;i<INFONUM;i++){
			write<<node->information[i]<<(i==INFONUM-1?"\n":"	");
		}
		WriteTraversal(node->left[0],write);
		WriteTraversal(node->right[0],write);
	}
}

void FreeSpace(Node* node){
	if(node){
		FreeSpace(node->left[0]);
		FreeSpace(node->right[0]);
		delete node;
	}
}

bool Find(Node* node,int ind,const string& info){
	if(node){
		if(node->information[ind]==info) return true;
		else return Find(node->left[0],ind,info) or Find(node->right[0],ind,info);
	}
	return false;
}
```