**1、设计目的与内容(需求分析)。**

我按照一般软件的使用模式，入口界面需要提供新建通讯录、打开已有项目、合并已有项目、退出程序的功能。其中新建和打开都应再进入一个选择界面，新建则直接进入项目，打开则输入文件名打开该文件。

选择功能应有排序、清除全部数据、添加联系人、保存、查找编辑、输出全部、退出项目的功能,并在不能使用该功能时不显示该功能并不提供选项，其中保存应询问路径。退出应该在有更改时询问是否保存。

查找编辑应该单独提供分选项，下属：更改姓名、添加电话号码、删除该名下的全部电话号码、删除此联系人、删除指定电话号码、退出查找的功能，并在不能使用该功能时不显示该功能并不提供选项。

1. **算法的基本思想**

(1)设计思想

1)数据结构设计

通讯录类：

{

是否进行过更改的记录

类关联的文件名

通讯录内容（采用双向链表C++ STL中的list）

通讯录内容的节点：

{

联系人名字

联系人电话列表：（采用双向链表C++ STL中的list）

联系人电话列表的节点：

单个电话号码

}

}

2)算法设计

每一层选择都使用了分支结构 + 循环结构，即提供诸多功能选项或更深一层的选择列表，每次选择一项功能完成或失败后循环，直到选择退出时才退出本层循环。

(2)设计表示

1)函数调用关系图及其说明

格式为：引用者->被引用者，其中一个框的上方只能被箭头指，下方只能引出箭头，左右方任意。

上图省略了对void Address\_book::Continue()const的引用，该函数较简单，且引用次数很多，不加入图中看得更清楚。

1. 函数接口说明

Interface.h

void InterfaceCentered(const int offset, const std::string &content)//参数1：居中输出的偏移量，左负右正。参数2：输出的字符串

void InterfaceLeft(const int offset, const std::string &content)//参数1：左对齐的偏移量。

参数2：输出的字符串。

void Interface//交互界面

(

const std::string &title,//交互界面标题

const unsigned int & ch\_num,//选项数量

const std::string \*choice//选项字符串数组

)

Address\_book.h

Address\_book(const std::string & fname)//文件名

std::list<Info>::iterator Find\_Info(const std::string &)//查找时输入的关键字，用于查找某个联系人姓名

void Out\_line(std::list<Info>::const\_iterator)const;//想输出的个人信息在个人信息链表中对应的迭代器

void Out(std::ofstream &)const//当前打开的文件流

bool operator< (const Info & i)const//另一个Info类对象，用来与当前对象比较大小。

(3)详细设计

详细设计（包括数据结构的类型定义，每个操作的算法描述）

以下是通讯录类和个人信息类（通讯录内容节点）两个类的定义。几乎囊括了本程序的所有数据结构和算法。

class Info;//提前声明类Info

class Address\_book//使用的通讯录类

{

protected:

bool m\_changed;//上一次保存或读取后是否进行过更改

std::string m\_fname;//对应文件名

std::list<Info> m\_Ab;//通讯录内容

public:

Address\_book() : m\_changed(false) {};

Address\_book(const std::string & fname) :m\_changed(false), m\_fname(fname)

{

Read();

}

std::list<Info>::iterator Find\_Info(const std::string &);//遍历查找名字

void Out\_line(std::list<Info>::const\_iterator)const;//输出一人的数据

void Out()const;//输出全部数据

void Out(std::ofstream &)const;//向文件输出全部数据

int Read();//从文件中读取全部数据

int Exit();//询问是否保存并退出

int Find();//查找编辑界面

int Save();//保存

void Add();//添加联系人

void Continue()const;//功能结束时等待界面

int SpliceAb();//合并两个文件

int OpenAb();//打开一个项目

int SelectFuction();//选择功能界面

int Run();//运行的主界面

};

class Info

{

protected:

std::string m\_name;//联系人名字

std::list<std::string> m\_phnums;//联系人电话列表

public:

friend std::list<Info>::iterator Address\_book::Find\_Info(const std::string &);

friend void Address\_book::Out\_line(std::list<Info>::const\_iterator)const;

friend void Address\_book::Out()const;

friend void Address\_book::Out(std::ofstream &)const;

friend int Address\_book::Read();

friend int Address\_book::Find();

friend void Address\_book::Add();

bool operator< (const Info & i)const//重载<，用于调用std::list::sort()排序时的比较

{

return this->m\_name < i.m\_name;

}

};

类Info也可以声明为结构体，我认为这样会破坏类的封装性，所以改想声明Adress\_book为其友元类，最后为了最大程度地维持类Info的封装性选择仅在其中声明使用它的protected成员变量的函数为友元函数。

bool m\_changed：用来记录是否进行过可能更改项目的操作。在每一次读取或是保存后都置false，进行可能发生更改项目的最小单位操作后置真。用来判断当前是否提供保存选项和退出时是否询问保存（若为假则不询问直接退出）

std::string m\_fname：该对象关联的文件名，采用string类保存。它的空与否决定了当前项目是新建项目还是打开的项目，并决定了在保存时是否询问路径。同时string与list同属STL体系下的类，其实string的容器也可以指定为list，不过我使用的是默认的容器。总之string也支持和list相似的操作。

std::list<Info> m\_Ab：C++ STL的list类双向链表。用来存储通讯录中的信息，其中的内容也正是需保存至文件的内容。同时其成员函数强大地支持对该双向链表的操作。

std::string m\_name：联系人名字。

std::list<std::string> m\_phnums：联系人电话列表，也用list实现。

class Address\_book的成员函数：

int Run();//运行的主界面

相当于程序的入口函数，可以进行第一层操作界面的输出和选择，每项功能执行完毕（无论是否成功）都循环回选择界面，直到选择退出时返回0值退出该功能。

选项有

"n:新建通讯录",

"o:打开已有项目",

"s:合并已有项目",

"e:退出程序"

其中前三个选项都有对应的函数，退出则直接返回0.

以及选项输入错误时给出错误提示并返回选择界面。

每次完成选择和功能后都清一次屏。

int SelectFuction();//选择功能界面

算法基本同上述Run()函数。

提供了不同的选项

"r:排序（按联系人名字编码）",

"c:清除全部数据",

"a:添加联系人",

"s:保存",

"f:查找编辑",

"o:输出全部",

"e:退出项目"

基本上是对一个项目的基本操作。

不同之处在于会检测此时m\_Ab的长度和m\_changed

在某些选项不能使用或使用无意义的时候禁用它们。

具体操作为将某些选项的文字输出替换成’\0’并在输入这些选项时将输入字符替换成’\0’，使选项变成错误输入。

并且在此层选择或更深一层选择中开始考虑若发生更改则将m\_changed置真。

其中没有调用自定义函数的选项的说明：

case'r'://排序（按联系人名字编码）

m\_Ab.sort();

cout << "排序成功！" << endl;

m\_changed = true;

break;

其中，std::list::sort()是一个将两个list模板类参数（即list双向链表的节点类型）类定义的的对象直接使用’<’进行比较来进行升序排列，其时间复杂度为O(nlog2n)。这里节点的类型Info并不能直接使用’<’比较大小，我对其进行了重载，令其直接比较联系人的姓名字符串大小。最后排序的结果是按编码排序，如果是Unicode码的话，字母按字母顺序排序，汉字按笔画排序。

case'c'://清除全部数据

m\_Ab.clear();

cout << "清除成功！" << endl;

m\_changed = true;

break;

调用了std::list::clear()清空m\_Ab中的全部元素。

int OpenAb();//打开一个项目

提示输入文件名。

若无法读取文件则返回0值。

若能够读取文件则返回1。

int SpliceAb();//合并两个文件

根据提示打开两个文件，将其中一个读入通讯录类对象并以追加方式写在另一个文件的后面。这个过程中合并入的文件是没有读入内存的，节约了时间空间。

void Continue()const;//功能结束时等待界面

就是一个起到输出"请按任意键继续. . ."并等待作用的函数，之所以写成函数是因为

被反复多次调用且可能有修改的空间。

void Add();//添加联系人



由于要使用类Info的protected成员变量，在Info中将其声明为友元函数。

其中调用了list::push\_back()，本来是栈支持的操作，在这里作用只是将指定元素添加到list的最后。

int Save();//保存

若打开文件错误返回0，保存（向文件输出）成功返回1。

int Find();//查找编辑界面

输入关键字查找某联系人，然后进入对单人信息进行操作的选择界面。选择过程和SelectFuction()是相同的设计。

由于其中某些选项涉及到调用Info类的protected成员变量，所以在Info类中声明为友元函数。

选项为：

"n:更改姓名",

"a:添加电话号码",

"c:删除该名下的全部电话号码",

"d:删除此联系人",

"p:删除指定电话号码",

"e:退出查找编辑"

int Exit();//询问是否保存并退出

也是和Run()函数相同的选择循环结构。选项为：

"y:是",

"n:否"

int Read();//从文件中读取全部数据

由于其中涉及到调用Info类的protected成员变量，所以在Info类中声明为友元函数。



void Out(std::ofstream &)const;//向文件输出全部数据

由于使用了Info类的protected成员变量，所以在Info类中将其声明为友元函数。

void Address\_book::Out(std::ofstream & outfile)const

{//向文件输出全部数据

using namespace std;

for (

list<Info>::const\_iterator Info\_iter = m\_Ab.begin();

Info\_iter != m\_Ab.end();

Info\_iter++

)

{

outfile << setw(8) << Info\_iter->m\_name;

for (

list<string>::const\_iterator str\_iter = Info\_iter->m\_phnums.begin();

str\_iter != Info\_iter->m\_phnums.end();

str\_iter++

)

{

outfile << setw(16) << \*str\_iter;

}

outfile << "\t\\" << endl;//使用'\\'作为一人数据的结束标志。

}

}

最终输出的格式是每行一个个人信息，行尾有一个反斜杠。

void Out()const;//输出全部数据

由于使用了Info类的protected成员变量，所以在Info类中将其声明为友元函数。

与上面的函数功能基本相同，只不过是对标准输出流（屏幕）输出，行尾不加\

void Out\_line(std::list<Info>::const\_iterator)const;//输出一人的数据

由于使用了Info类的protected成员变量，所以在Info类中将其声明为友元函数。

向屏幕输出一行数据 。

std::list<Info>::iterator Find\_Info(const std::string &);//遍历查找名字

std::list<Info>::iterator Address\_book::Find\_Info(const std::string & keyword)

{//遍历查找名字

std::list<Info>::iterator iter = m\_Ab.begin();

for (; iter != m\_Ab.end(); iter++)

if (iter->m\_name.find(keyword) < iter->m\_name.length())//如果名字与参数部分相同

return iter;

return iter;

}

由于使用了Info类的protected成员变量，所以在Info类中将其声明为友元函数。

模糊查找联系人姓名，利用了string::find()函数，最终返回查找到的名字向而对应的迭代器。

class Info 的成员函数：

bool operator< (const Info & i)const//重载<，用于调用std::list::sort()排序时的比较

{

return this->m\_name < i.m\_name;

}

这个函数是用来重载使用Info类的<，在class Address\_book中int Address\_book::Find()函数选择排序操作时调用了std::list::sort()，这个函数在实现时使用了<，我重载它为两个个人信息中的名字直接比较，一般使用Unicode码，可以起到英文按字母顺序排序，汉字按笔画排序的效果。

Interface.h中的函数

void Interface//交互界面

(

const std::string &title,//交互界面标题

const unsigned int & ch\_num,//选项数量

const std::string \*choice//选项字符串数组

)

此函数的功能为输出一个交互界面。其中调用了void InterfaceCentered(const int offset, const std::string &content)和void InterfaceLeft(const int offset, const std::string &content)。

void InterfaceCentered(const int offset, const std::string &content)//居中输出，指定偏移量。

此函数功能为在交互界面中居中输出一个字符串，其中第一个参数为居中偏移量，左负右正。

实现原理：调用std::string::length()计算字符串长度，和要输出的界面的宽度一同计算字符串的位宽，实现不改变本行宽度的情况下居中输出一个字符串，并可以任意指定其偏移量。

void InterfaceLeft(const int offset, const std::string &content)//左对齐输出，指定偏移量。

此功能为左对齐输出一个字符串，其中第一个参数为左对齐偏移量，一般非负。

实现原理：调用std::string::length()计算字符串长度，和要输出的界面的宽度一同计算字符串的位宽，实现不改变本行宽度的情况下左对齐输出一个字符串，并可以任意指定其偏移量。

1. **系统调试出现的问题**

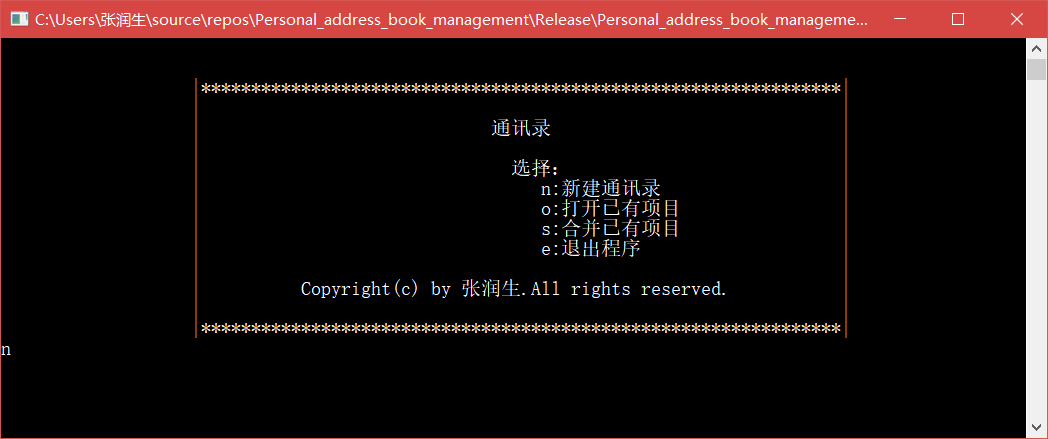
曾经出过无法正常声明友元函数的错误，后发现应该先定义class Address\_book再定义class Info。不过这样又出现了新的错误，经排查应该提前声明class Info

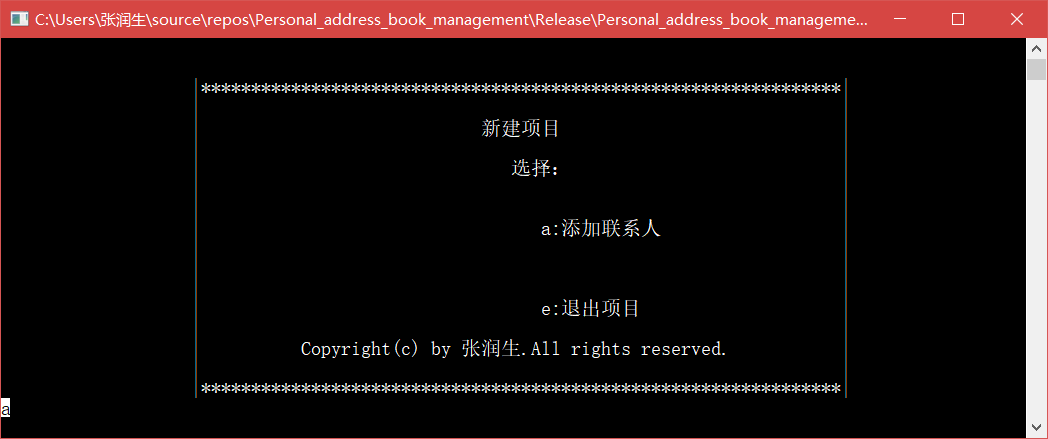
在编译链接阶段出现过无法指出所在行数的错误，出在xutility文件，排查了好久后来发现是因为对函数有重定义造成的类型不匹配。删掉了重复的定义。

输入选项时总是需要多输入一个回车键，是因为在忽略输入流的字符时以’\n’作结，事实上也相当于必须强制输入’\n’（因为忽略字符数上限设置得足够大）。这样只要在程序开头和每次清空缓冲区后向输入缓冲区中写回一个’\n’就可以统一操作，每次只需输入一个回车。

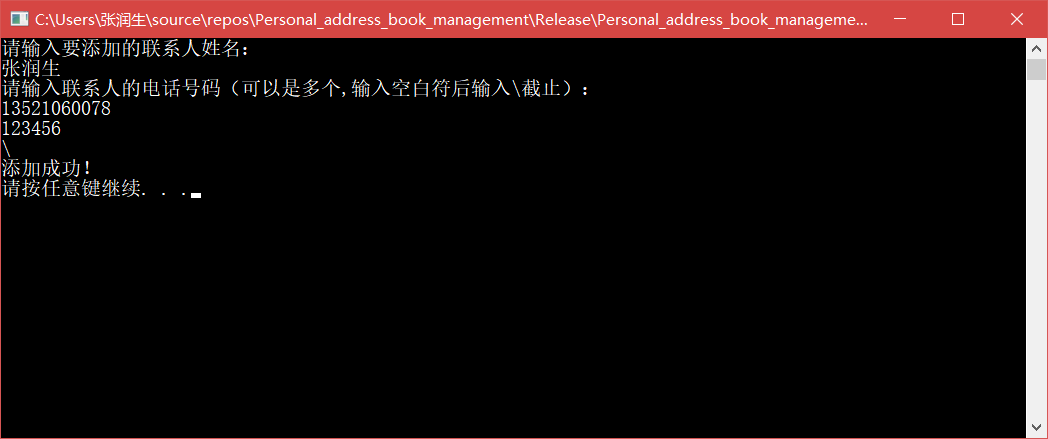
曾经有一个错误是每次保存一次在文件的末尾都会多一个’\\’，经反复排查发现不是保存的问题，而是在读取时由于EOF前面还有一个’\n’，在读取完最后一个字符串后并没有遇到EOF，还会再次进行一次循环，并且把一个空的节点添加到m\_Ab后面，虽然在读入时和输出全部显示时都没有问题，但一旦再次写入文件，多的那个节点也会循环一次，并写入一个’\\’，解决方法就是在读取时加了限制条件if (inf.m\_name.length() != 0)//如果姓名不为空

1. **测试数据及测试结果分析**



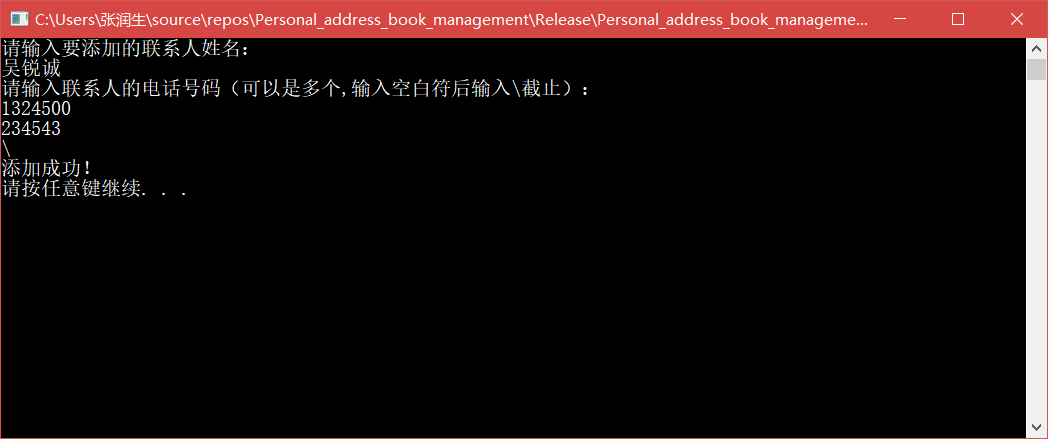


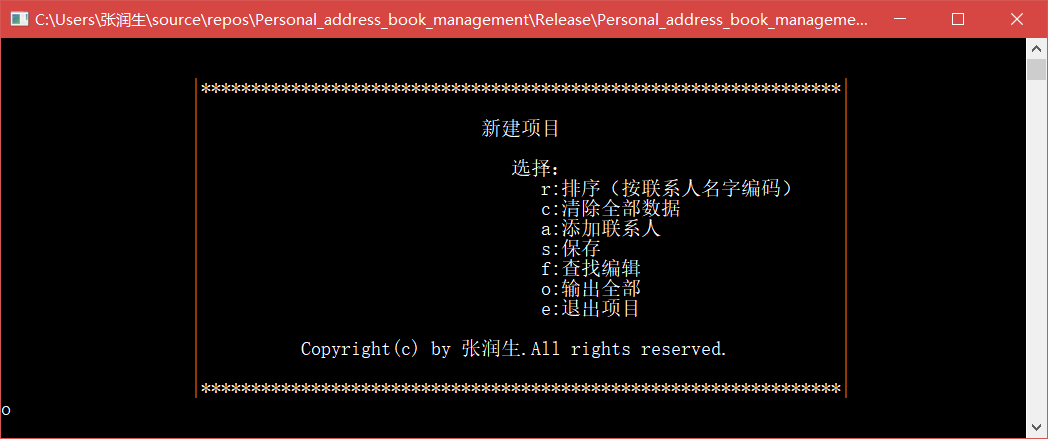
由于此时还是空项目，支持的操作只有添加和退出



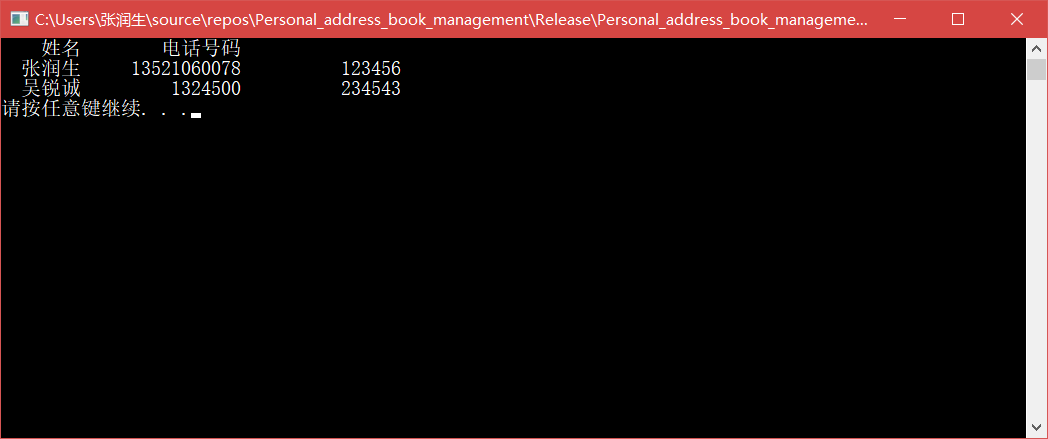


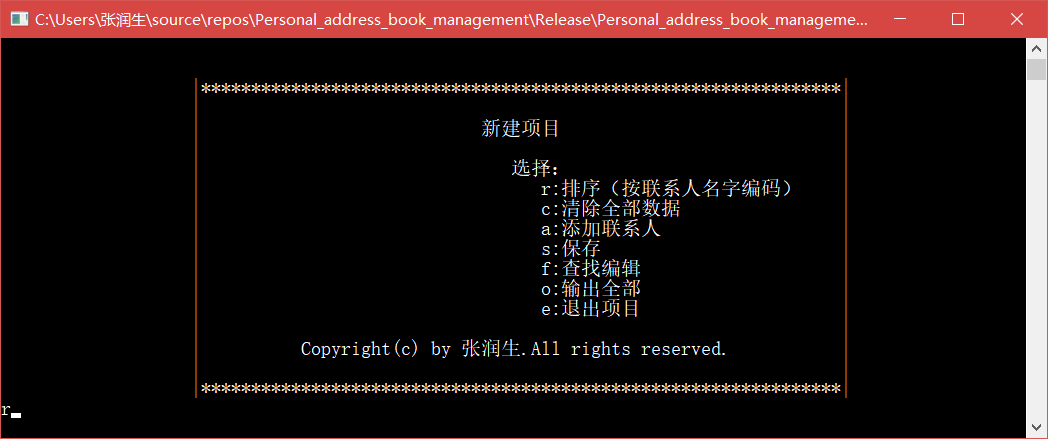
现在有一个元素了，不过仍不支持排序

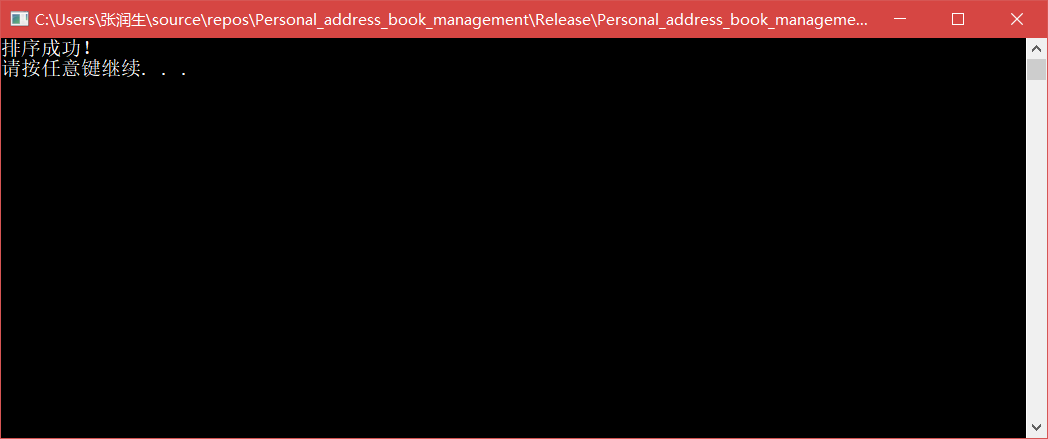


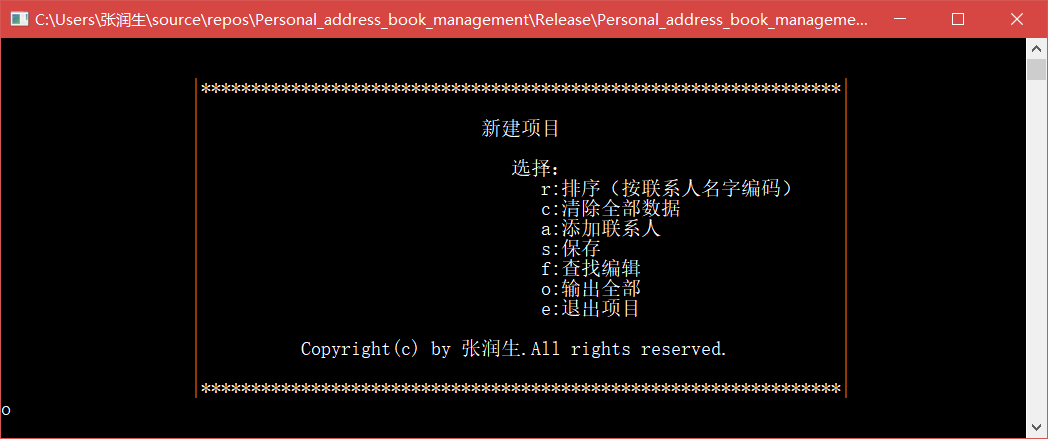


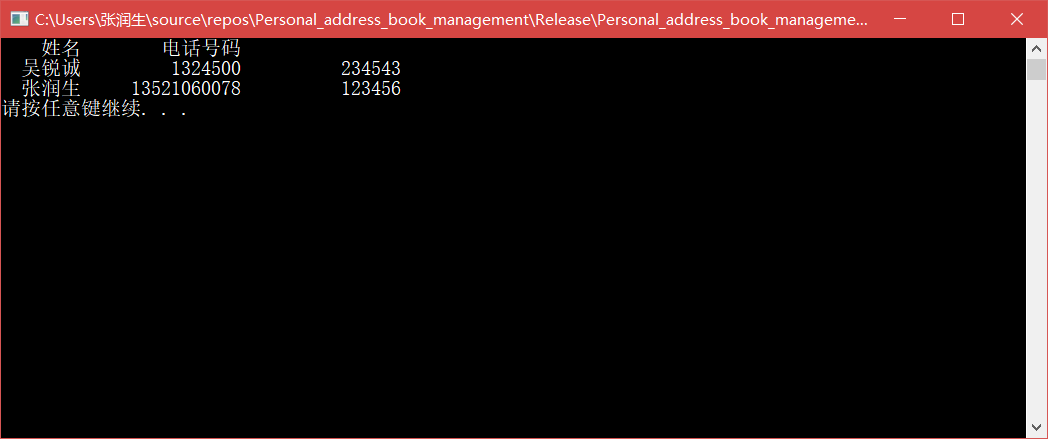
现在有两个元素了，支持排序

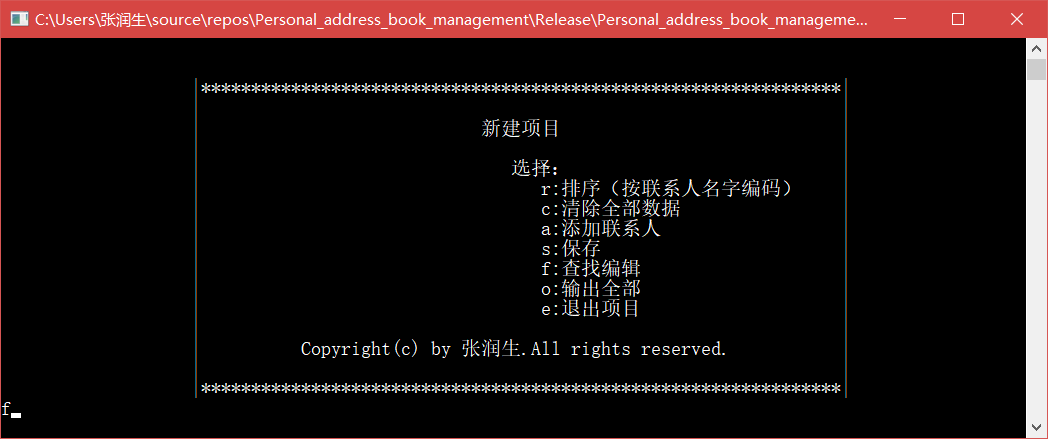


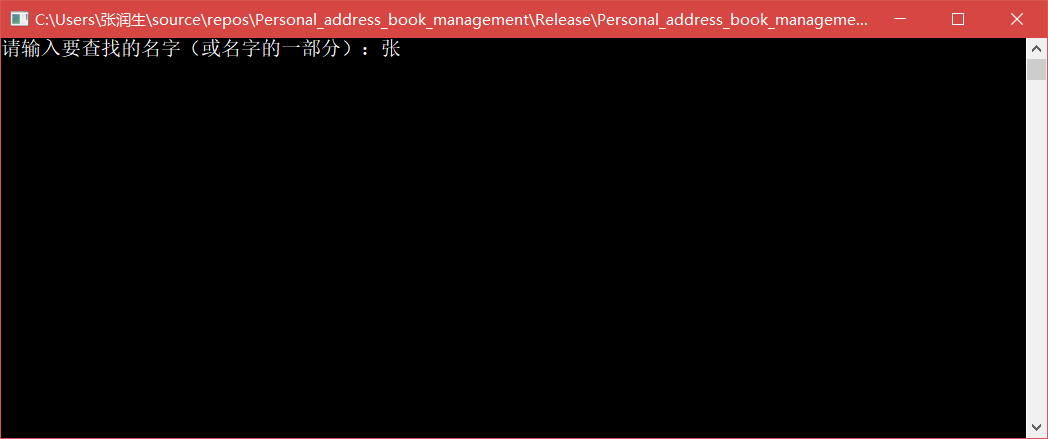


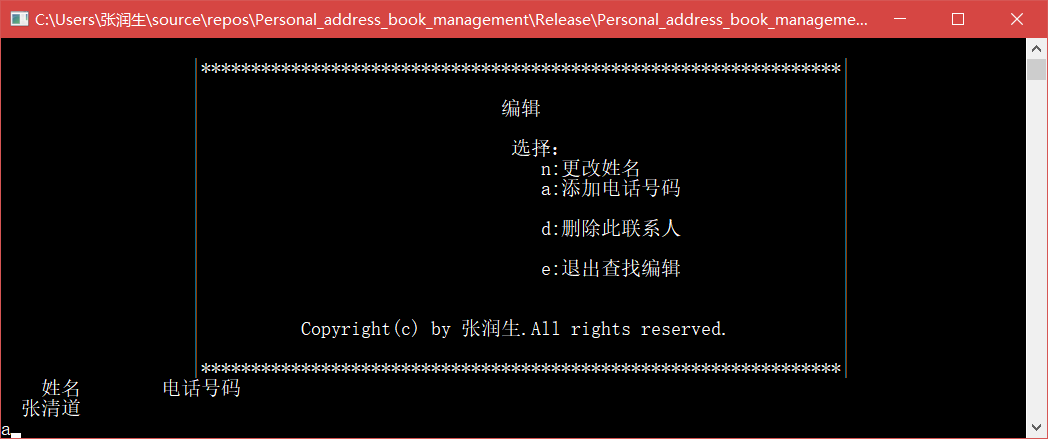
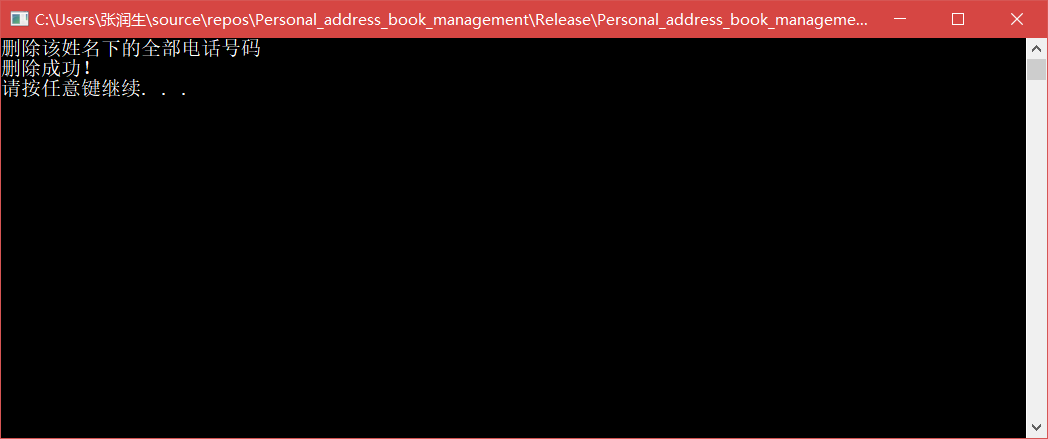
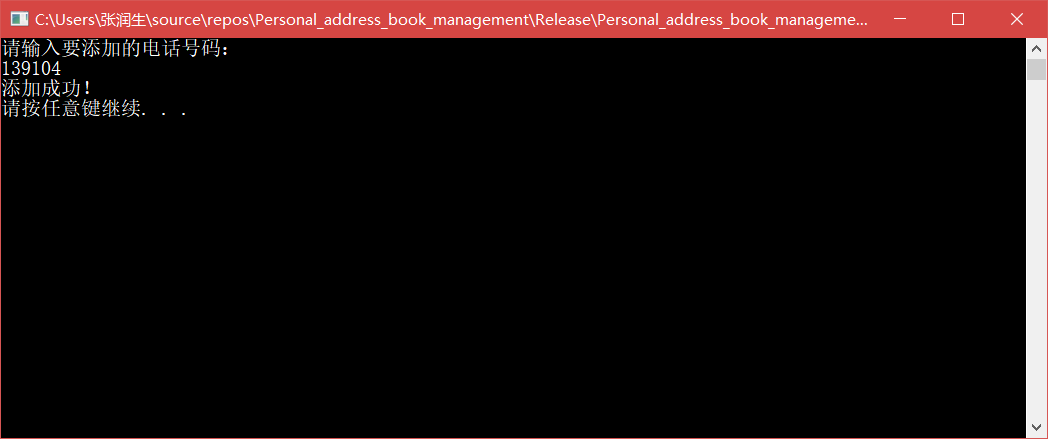
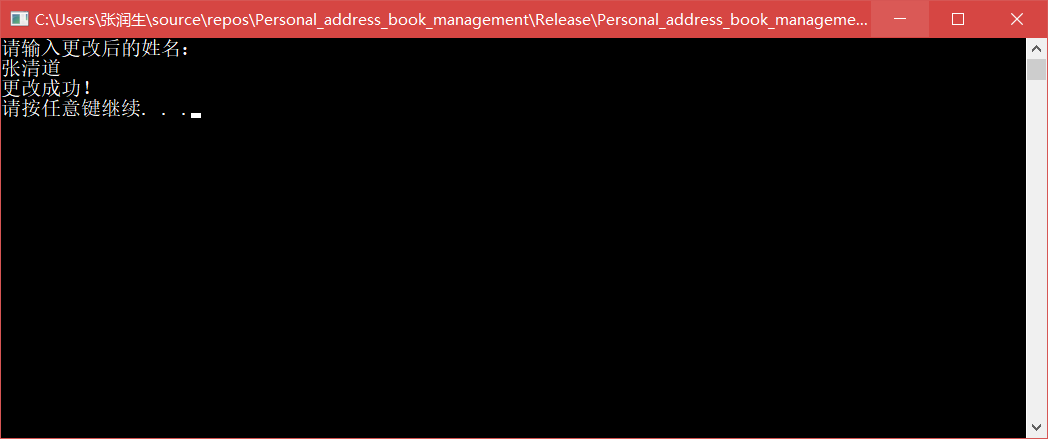




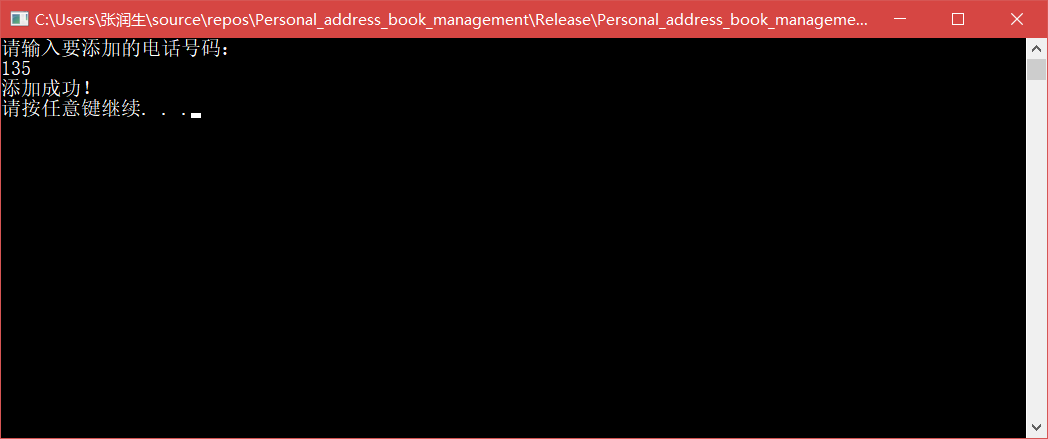




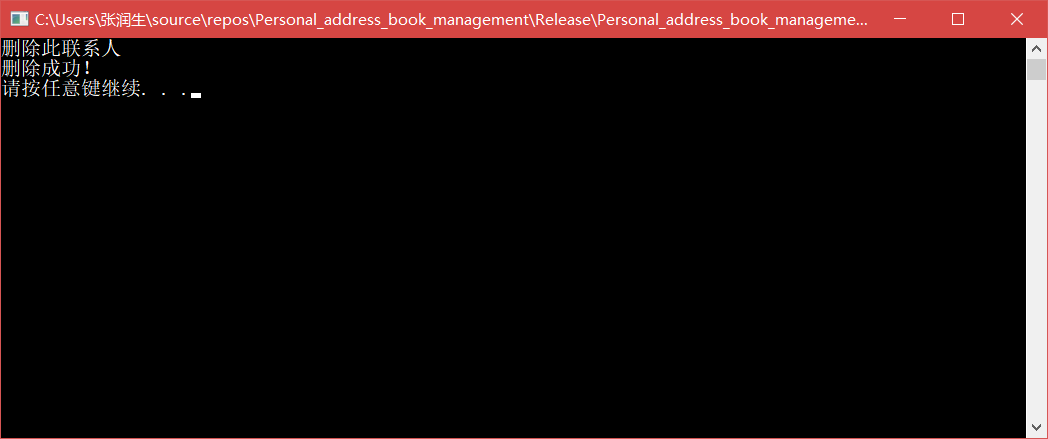
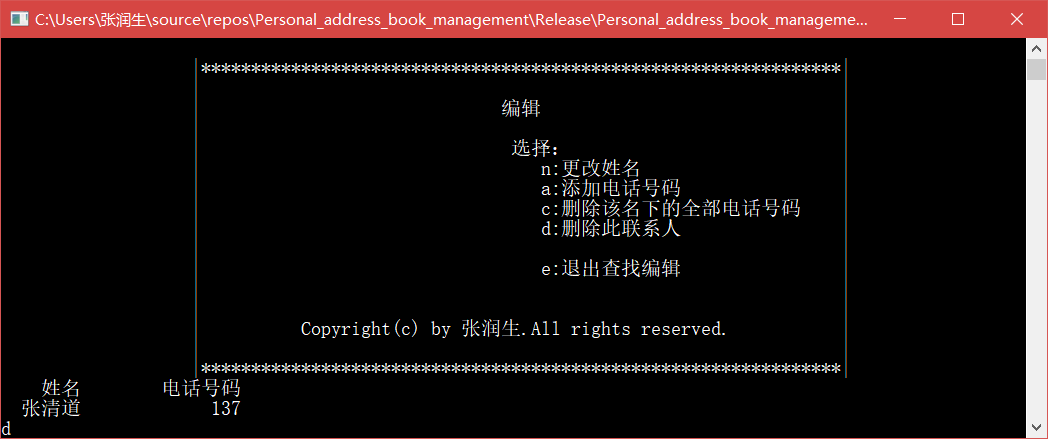
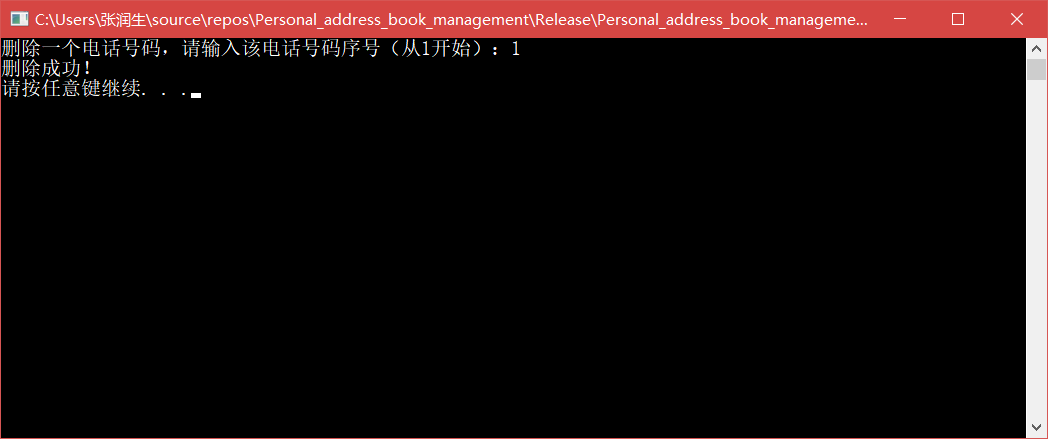
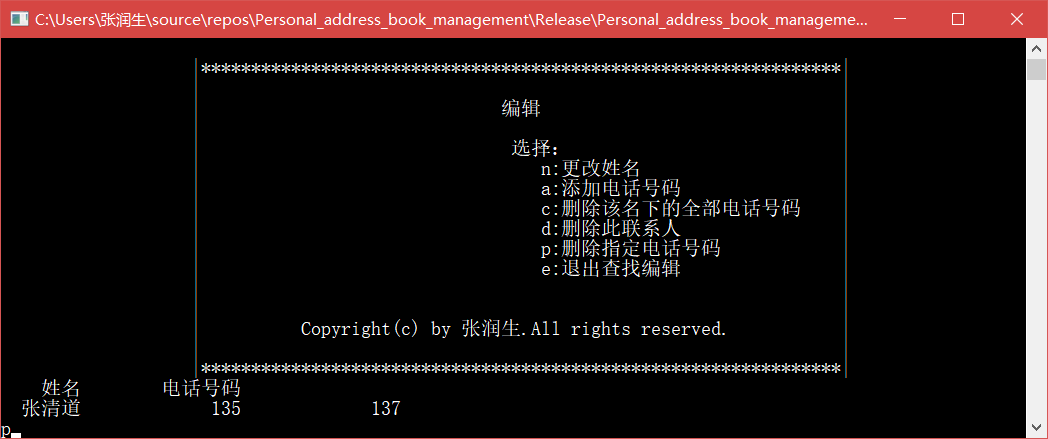
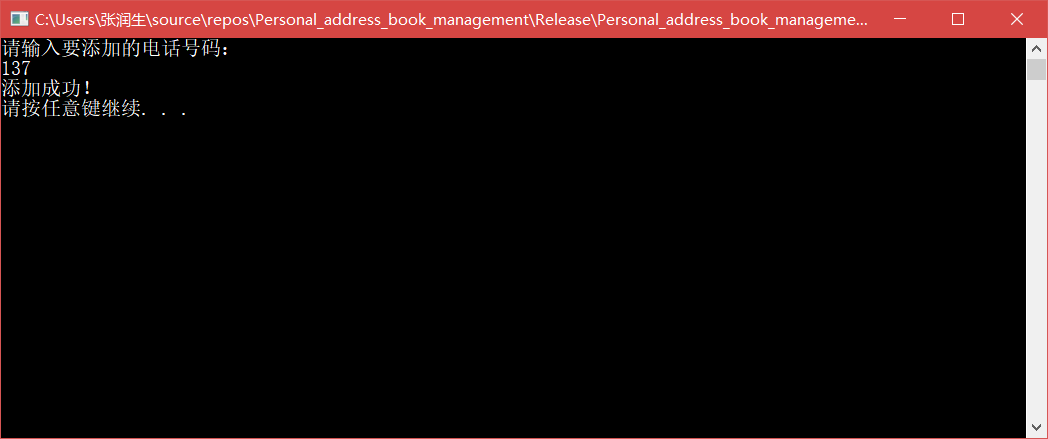


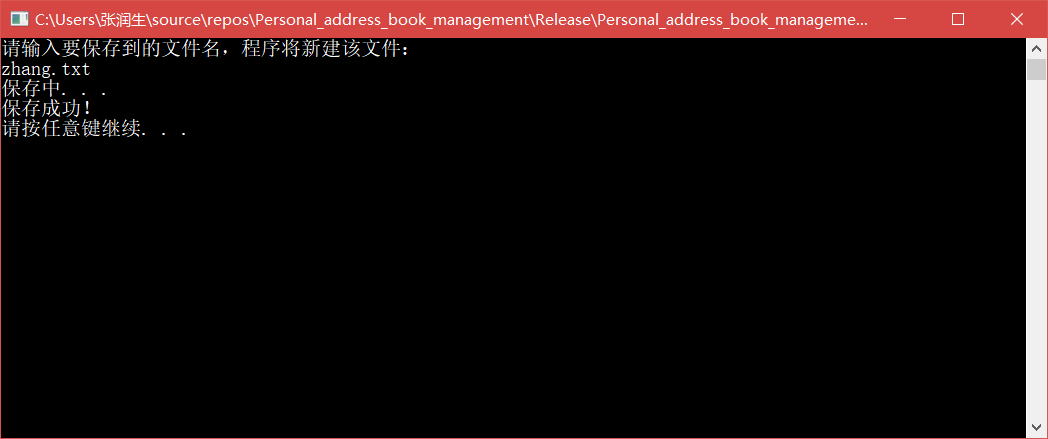


此时由于电话列表为空，也削除了相应的操作

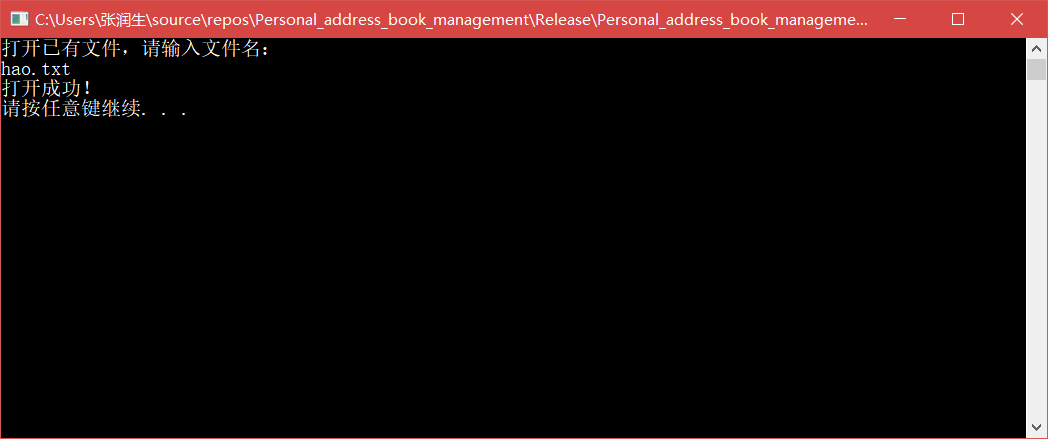
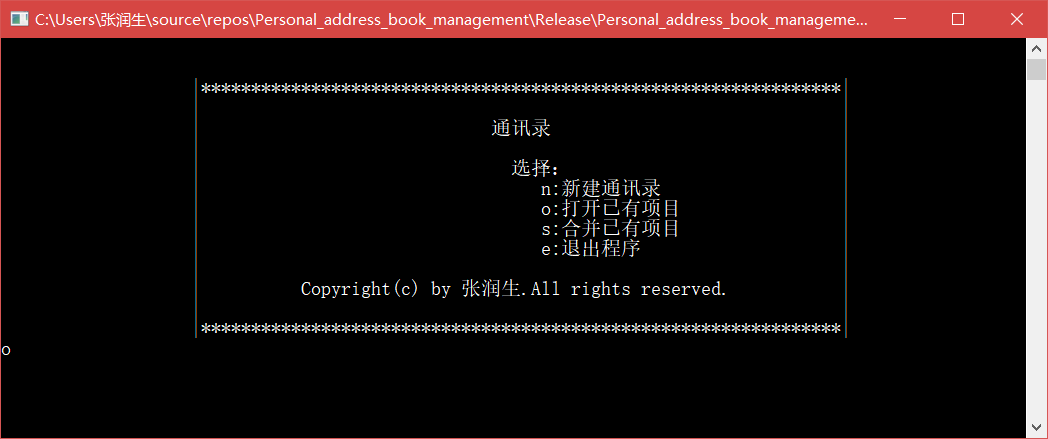


电话列表有一个元素，此时不支持按位删除（可用删除全部替代）

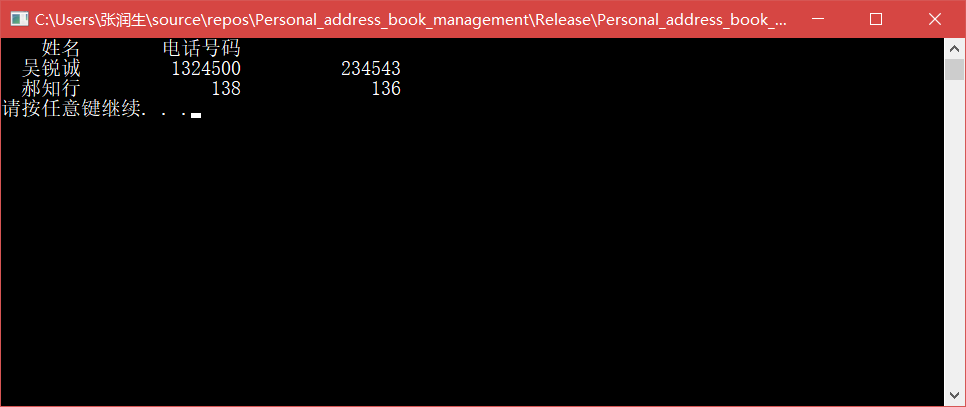
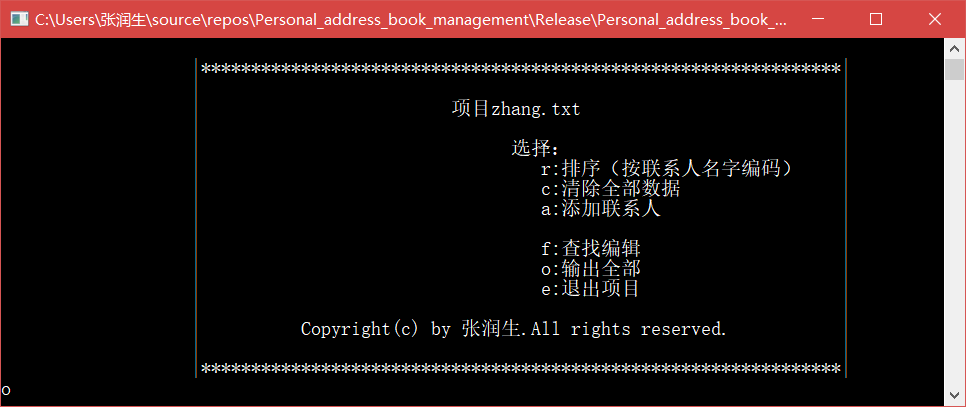
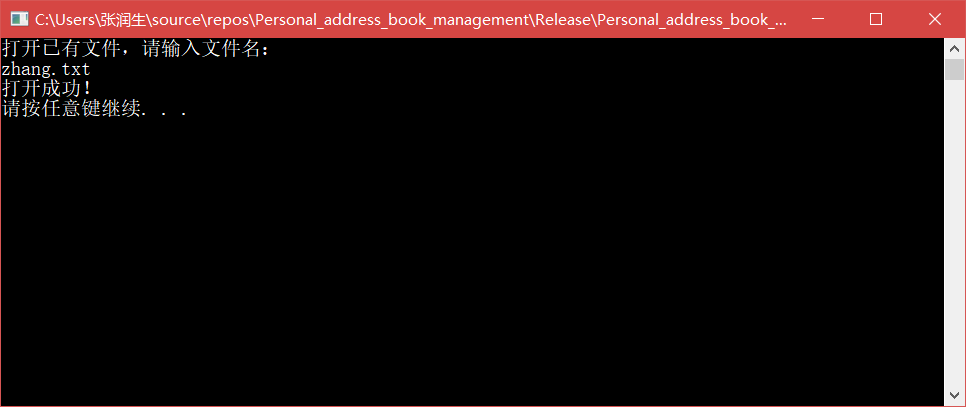
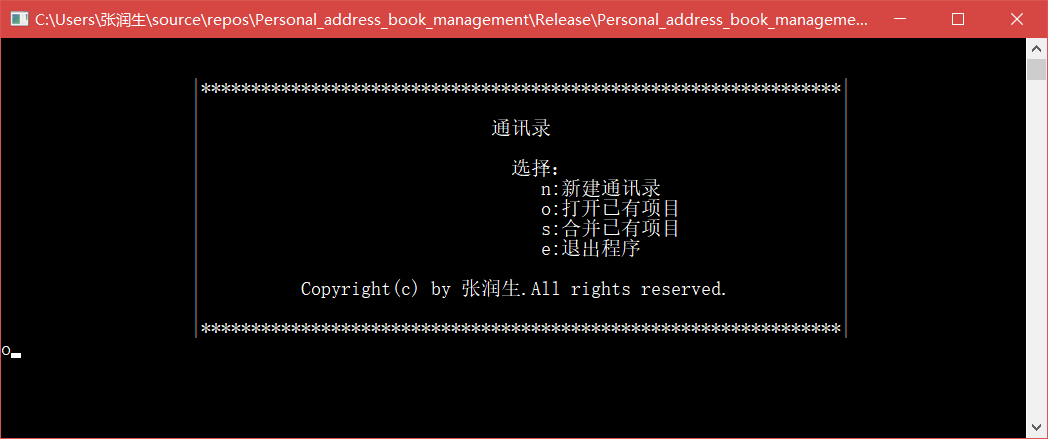
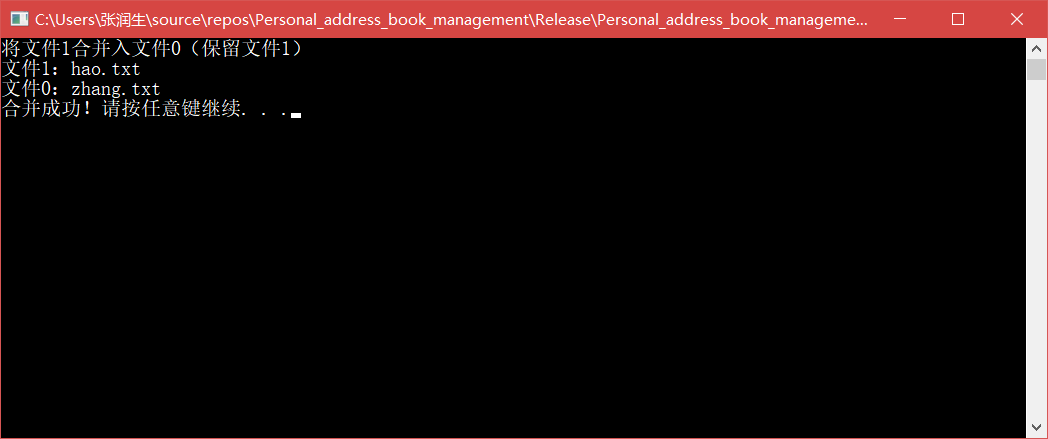
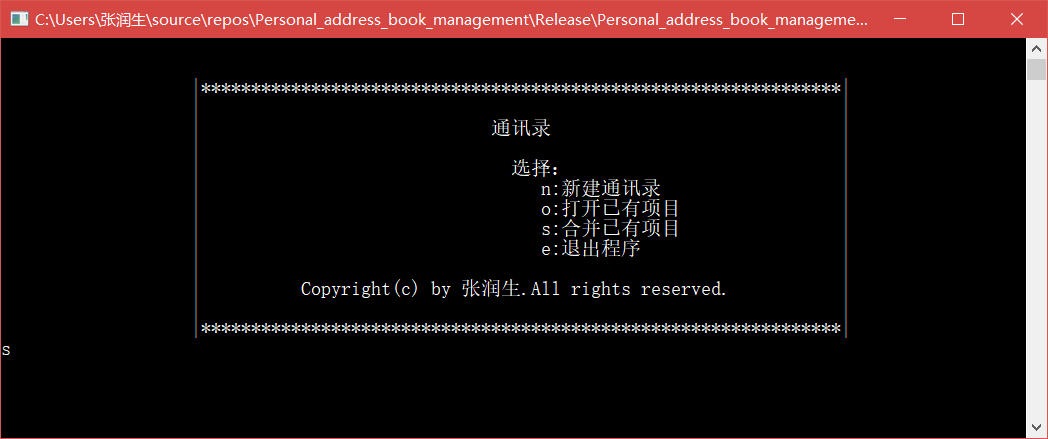
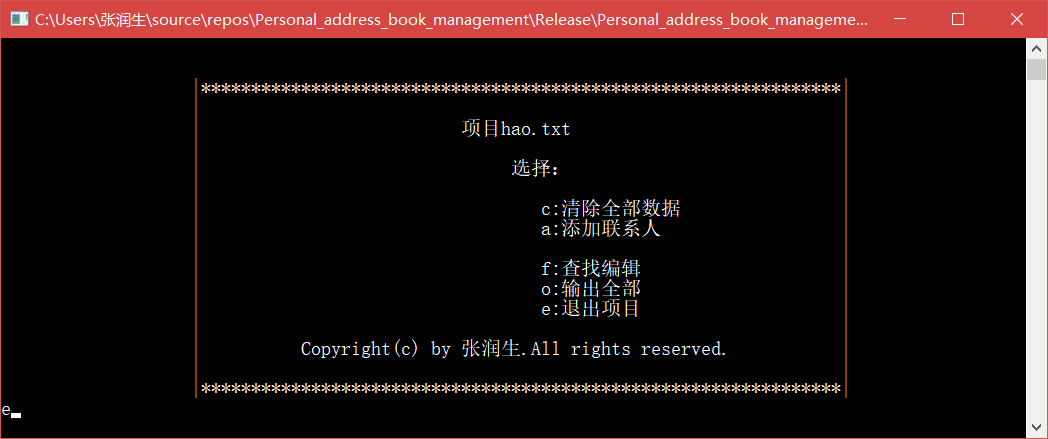
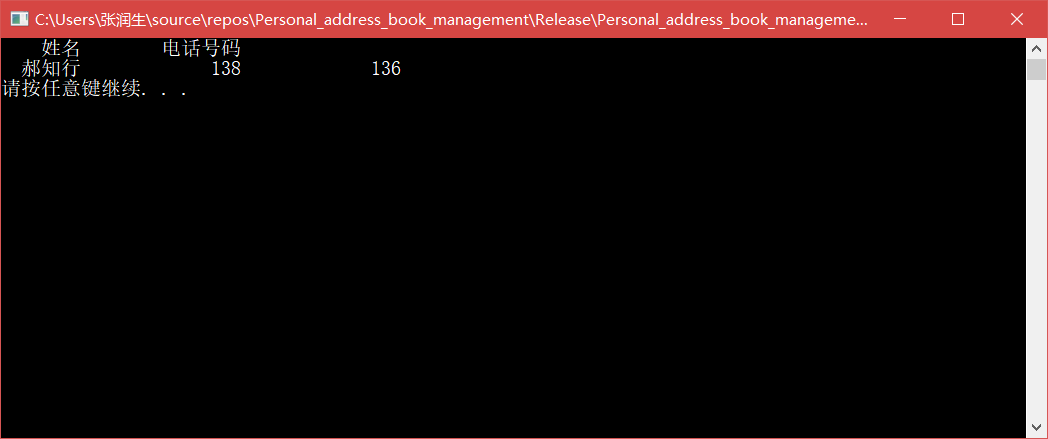
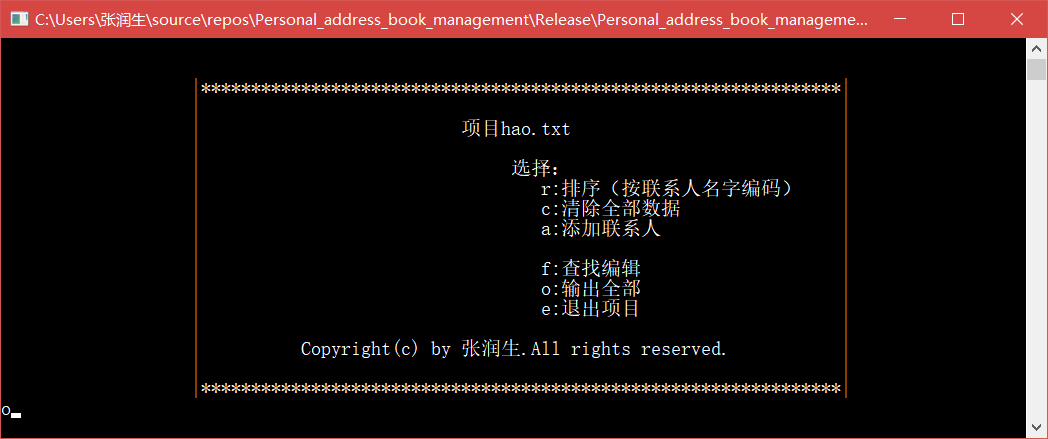




保存后还未更改，所以不提供保存选项

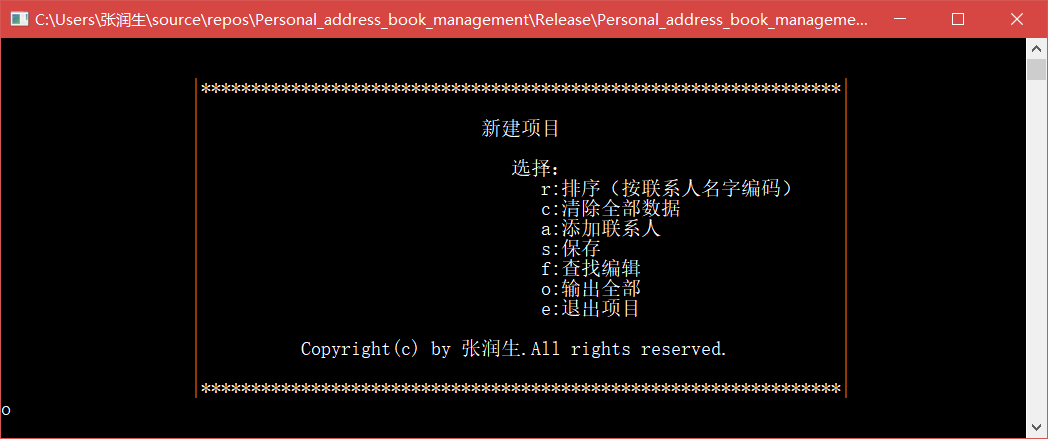
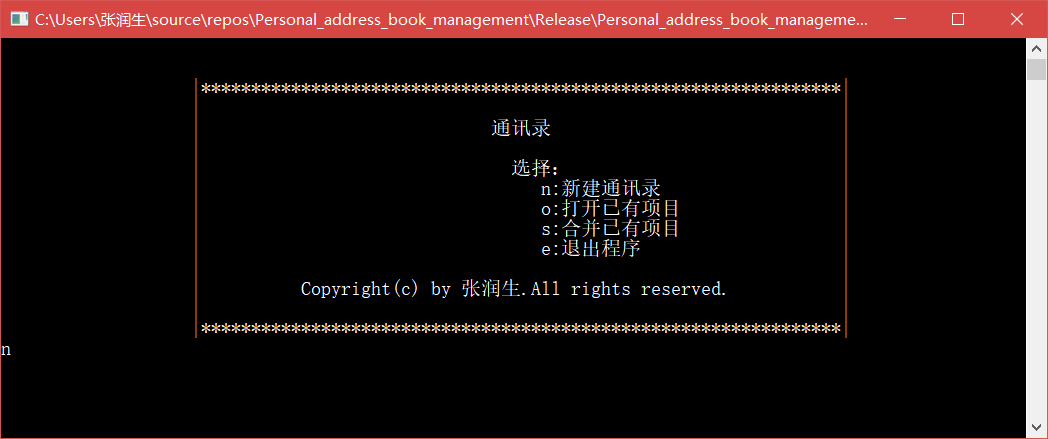


之前写好的文件



1. **用户手册**

三个界面：



1. **心得体会**

首先我感到STL确实非常好用，内置了很多成员函数，时间复杂度和空间复杂度都很优秀，更是有着出色的稳定性。list作为一个双向链表模板可以自定义节点类型，又有着链表的特性，不用指定长度，也不用担心溢出。

不过一般因为要保证代码的标准性，最好不能继承STL，易破坏STL的标准性。最好采用将STL作为类成员。但这样想要在STL基础上扩展功能时则需要将用到的函数全部重写一遍。

这个课设我在一开始在做时对文件操作这方面非常迷茫，不光是如何操作，我甚至不知道文件操作的需求（也就是何时提供文件操作？），还以为每做一次更改就要操作一次文件，并且使用过程中也要读取文件，绕了不少弯路。后来仔细一想，平时使用的软件都是编辑——由用户指定保存——退出时询问是否保存的模式。这种模式非常清晰地体现了何时应进行文件操作，并且文件操作频度低，还考虑到了我之前没考虑过的问题：如果想要放弃更改怎么办？

这些也引出另一个问题，就是实际上很多时候给程序员的需求并不是特别明确，好多地方需要我们自己「脑补」，有很大自己决定的空间。我觉得如果有迷茫的话，不如想想自己平时是如何使用软件的，有哪些自己觉得便利的地方和不便的地方，来补充不完整的需求。

在程序调试时我觉得应该充分发挥IDE的优势。我们毕竟不是用记事本写代码，有些IDE是很强大的，例如可以监视局部变量，这样完全可以代替「printf调试大法」，不用对代码做任何改动。另外逐语句调试和逐过程调试更是可以轻易定位在编译连接成功后运行阶段出现的问题。

C++作为一门面向对象编程的语言，和C语言在思想上有很大的不同。每一个对象就像是一个功能健全的个体，既包含了数据结构又包含了算法，容易对应生活中或问题中实际存在的事物。数据结构和算法就好像物质和运动的关系，在C++中得以得到统一起来，实际上更符合实际问题中对问题中不同部分的划分，这就决定了C++编程中，模块与模块之间的耦合度更低，在编程中不至于「牵一发而动全身」，一段好的代码更是能用到不同的项目中。