****

数据结构课程设计任务书

**（2020/2021学年第二学期第17周）**

**指导教师：胡洁 郭奕亿**

**学生1：计算机类(4) 2020333503081 陈伟剑**

**学生2：计算机类(7) 2020329621237 程玉**

数据结构与算法课程设计报告

**一、选题**

1.通讯录背景

通讯录作为通讯录地址的书本，当今的通讯录可以涵盖多项内容现今的移动通讯录是一种利用互联网或移动互联网实现通讯录信息同步更新和备份的应用/服务。你可以在个人电脑、掌上电脑、移动电话等任何联网设备上录入你的联系人的手机\电话号码、Email、QQ、MSN、通信地址等通讯录信息。

2.选题意义

在这个信息化时代， 日益繁多的人际交往使得我们很难搞清楚与每个人之间的联系方式， 所以通讯录能够便捷的给我们带来所需要的相关信息。 通过通讯录系统记录下对方的联系方式和基本信息， 既减轻了大脑的负担，又不在受各种条件的约束 , 做到便捷随身管理相关联系信息，摆脱了传统名片的累赘。

3.可行性分析

技术可行性:当通讯录信息存储到一定量时,查找起来时间会很长,但如果将记录的储存位置与它的关键词之间建立一个确定的关系，在查找时根据对应关系计算出给定关键字对应的值，就可以得到记录的存储位置，再以记录的关键字值为自变量，根据哈希函数计算出对应的哈希地址，并在此存储该记录的内容，运用这种方法时会产生一些同义词，再根据线性探测再散列法解决此冲突。

**二、需求分析**

1.功能划分

通讯录系统可分为3个部分：

1. 通讯录管理：导入联系人文件、添加联系人、删除联系人、清空联系人
2. 通讯录查询：显示联系人、查找联系人
3. 其他：保存通讯录

2．功能描述

数据前提：每条记录为 中文名(至少一个汉字)+电话号码(11位，同一个名字只对应一个电话号码)

（1）导入联系人文件：读取同文件下的联系人文本文件并存入哈希表

（2）添加联系人：向通讯录中添加新人，信息包括（姓名、电话），在程序中设计联系人的结构体

（3）删除联系人：按照姓名进行删除指定人。

（4）显示联系人：显示所有联系人信息。

（5）查找联系人：按照姓名查找联系人。

（6）清空联系人:清空整个通讯录。

（7）保存通讯录：将信息存入文件。

3.性能需求

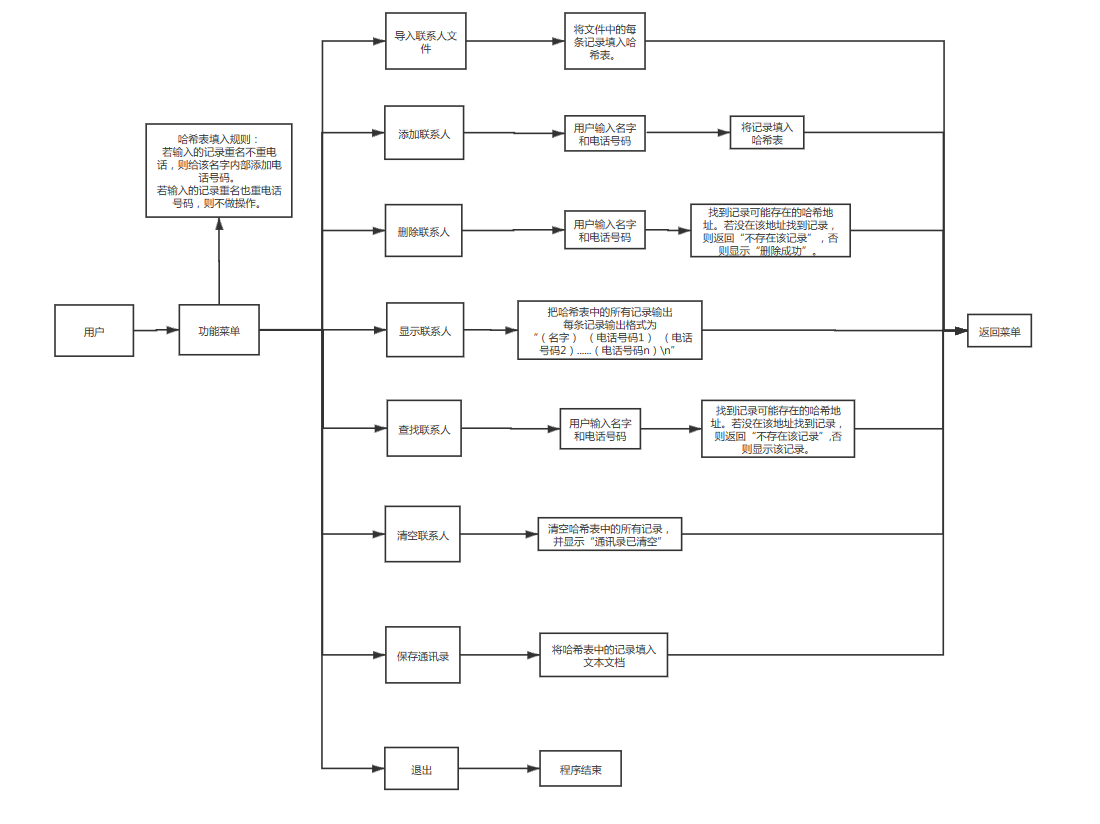
为了实现通讯录快捷性的特点，查询功能必须在几秒内实现。

4.程序的错误处理

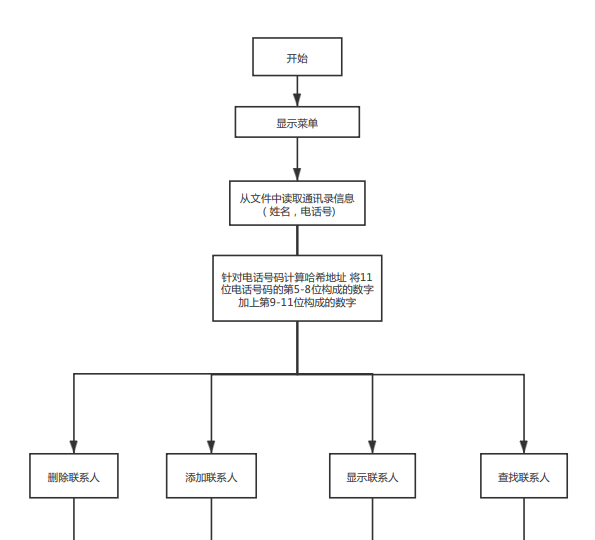
如果输入非法编号，直接结束程序

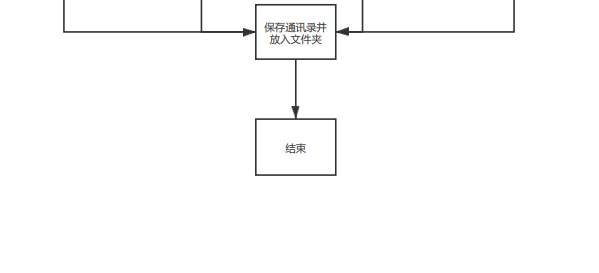
**三、基本框架**

1. 系统结构图



1. 业务流程图





1. 数据结构的设计

（1）单个联系人的存储结构

系统的数据类型有：整型 int，字符型 char，结构体 struct；

（2）数据结构

数据结构：通讯录好友  
含义说明：通讯录系统的主体数据结构，定义一个好友的有关信息。  
组成：好友姓名，好友电话号码。

typedef struct info

{

char number[13]; //号码//默认一个联系人只对应一个联系方式

char name[29]; //姓名

}information;

typedef struct Person

{

info\* member; //联系人记录

bool\* ifempty; //判断对应位置是否被占用。1为空，0为非空

int size; //当前存储的联系人数

int listsize; //哈希表的最大存储容量

int hash\_num; //除留余数法的模数

}Per;

（3）数据项

数据项：好友姓名  
含义说明：好友的姓名  
别名：姓名  
类型：字符型  
长度：29

数据项：好友联系方式  
含义说明：手机号  
别名：联系方式  
类型：字符型  
长度：11  
取值范围：00000000000~99999999999

（4）功能函数

Ⅰ.主要函数

void Test(); //执行程序

int load\_in(Per\* S); //导入联系人文件

void insert\_member(Per\* S);//添加联系人

void delete\_member(Per\* S);//删除联系人

void display(Per\* S);//显示联系人

void search(Per\* S);//查找联系人

void ClearList(Per\* S);//清空联系人

void save(Per\* S);//保存通讯录

int hash\_change(char \*t,Per\* S);//计算哈希地址

Ⅱ.辅助函数

inline int InitList(Per\* S); //初始化哈希表

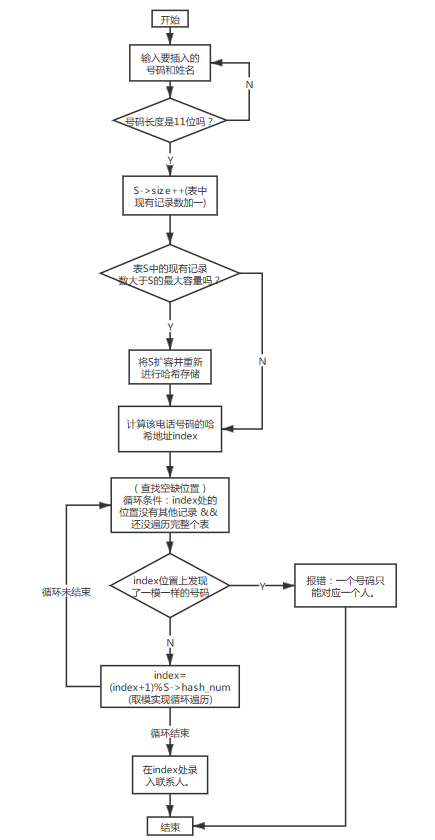
inline void againmalloc(Per\* S); //扩大容量

inline void declinemalloc(Per\* S); //缩减容量

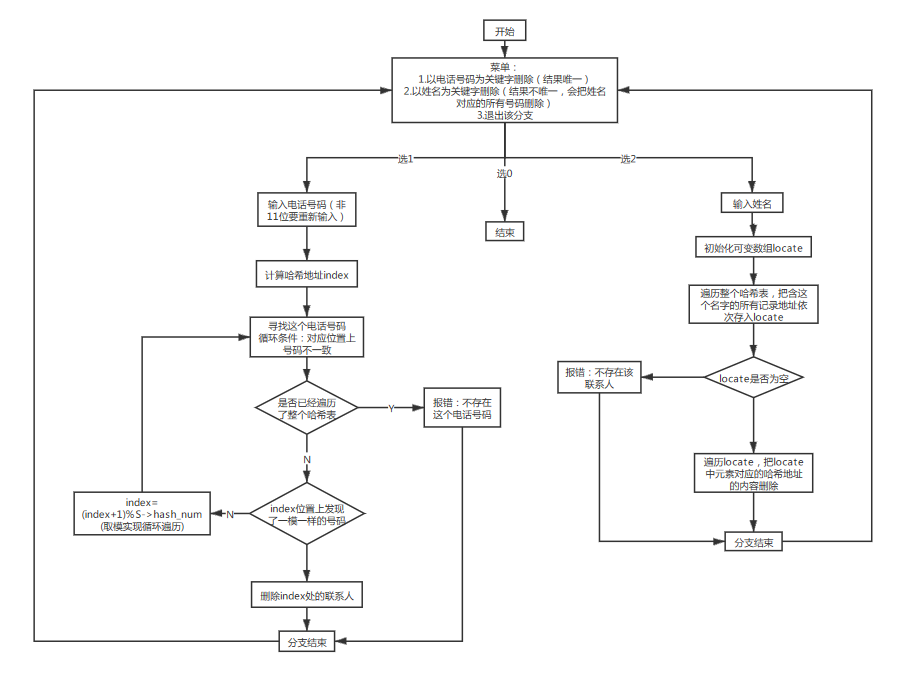
int load\_in\_saved(Per\* S); //功能与load\_in()相同，区别在于该函数读取 //的文件为“connect\_save.txt”

（5）功能函数具体流程图

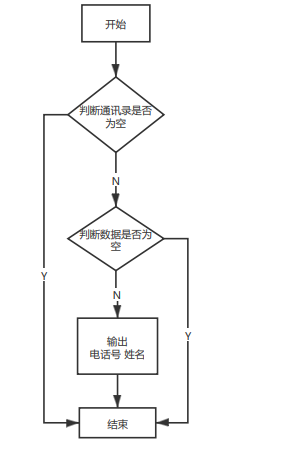
1.添加联系人



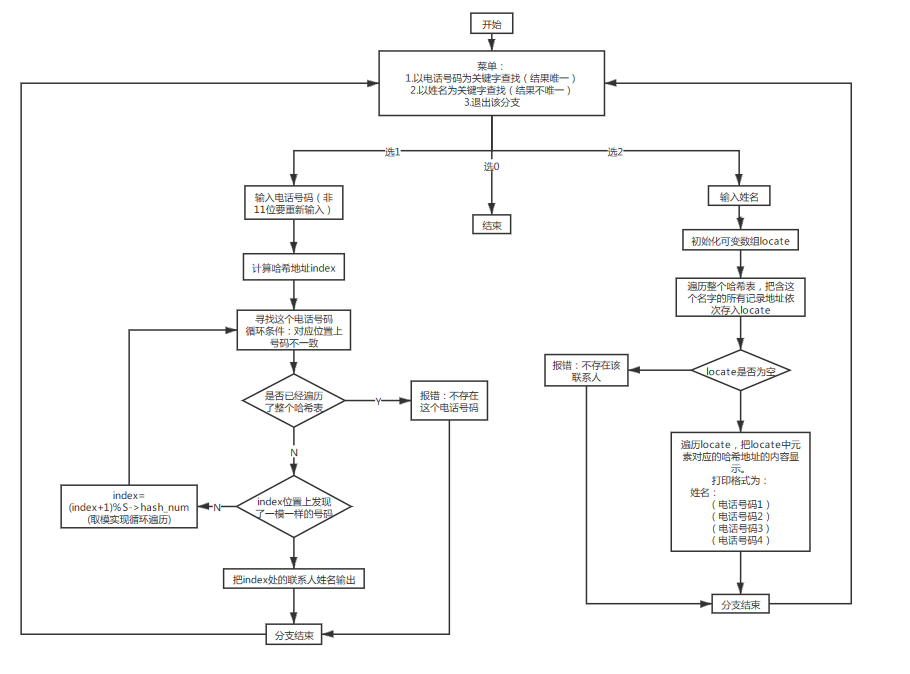
2.删除联系人



3.显示联系人

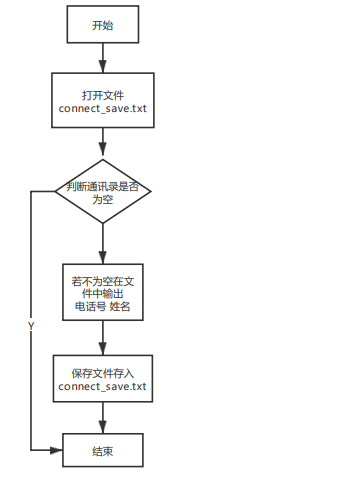


4.查找联系人

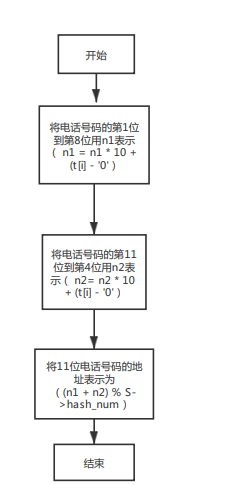


5.清空联系人 （注：直接调用Initlist（）函数初始化）

6.保存通讯录



7.计算哈希地址

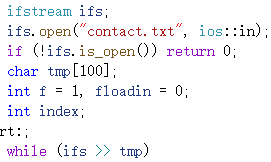


**四、调试分析**

①int load\_in(Per\* S); //导入联系人文件

前提：假设文件里有n行记录，哈希表初始最大容量为2。

第一部分（读取）：文件内部的每一行均为一条记录，形式为“（号码）+（空格）+（姓名）\n”。读取方法为一块一块读（下图的方法会略过空格和换行符）。



每条记录都是先读入号码再读入姓名。读入号码时先判断号码的合法性（是否是11位，是否全为数字），再选择要不要存储下一次读入的姓名。若不合法，该行的记录存储将被跳过。

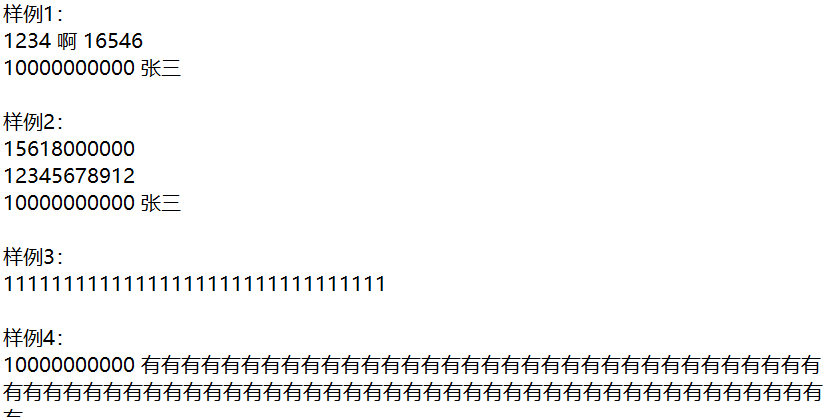
第二部分（哈希存储）：采用除留余数法计算哈希地址，若发现计算出的哈希地址被占用，就对哈希地址+1并取模（模数为哈希表的最大容量）。此外，发现容量不够时采用再散列法扩容（容量扩为2倍后从头开始哈希存储），一共要进行 次扩容，对应会重复进行次哈希存储。第 **i** 次重新开始哈希存储的时间复杂度最坏情况为O()(刚好存完)。最好情况为O()(扩容前只剩一条记录未存入)。

整个过程的时间复杂度为O(n()。空间复杂度为O(n)。

问题分析：

Ⅰ.该算法的整体效率在数据量超过十万级时会显著降低，因为对每一条记录都可能会重复进行10次以上的存储操作。

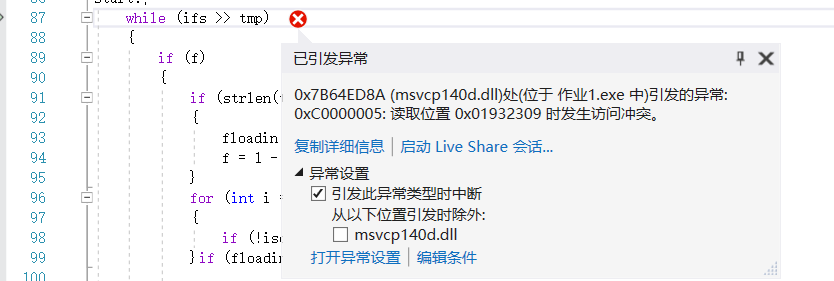
Ⅱ.该算法读文件的方法比较取巧，泛用性较差。调试时试想出了以下非法输入数据。



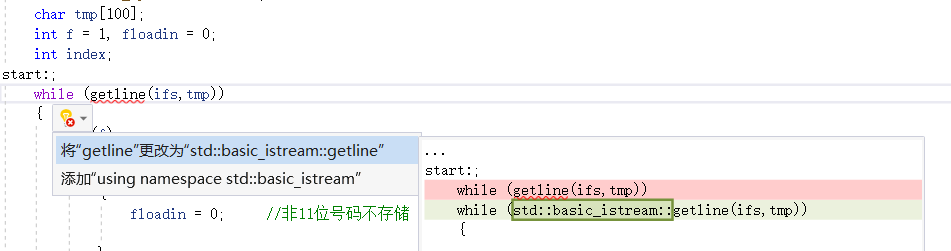
针对样例1和样例2，函数会直接跳过非法行的存储。

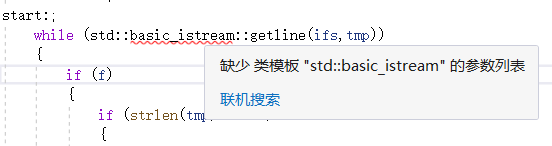
针对样例3，函数会把判定号码过长而跳过。

针对样例4，由于每条记录的姓名都有最大容量，这种样例越界后会报错。



Ⅲ.原本考虑用getline(ifs,tmp)的方式读取完整的一行，出现了以下尚未解决的问题。





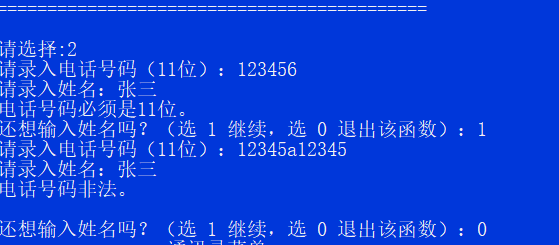
Ⅳ.使用InitList（）初始化时没有考虑好容量初始化的问题。由于忘加MAXSIZE导致频繁的文件读入失败(空指针)。

②void insert\_member(Per\* S);//添加联系人

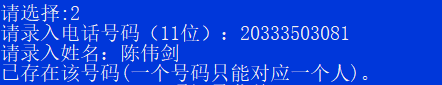
具体存储方法与load\_in中的方法一样。由于遇见超出容量的情况会比读文件时的情况低很多，故考虑将不扩容时的复杂度作为平均情况。时间复杂度为O(n)，空间复杂度为O(1)。

问题分析：

Ⅰ.可能出现非法输入（号码不是11位或者号码不全为数字）。后期对代码修改之后会判断输入的合法性，若为非法，就会给出重新输入的提示。



Ⅱ.输入电话号码或可能会重复，修改代码后会先进行哈希查找，若发现相同号码会弹出提示。（同姓名不同号码可以存储，因为一个号码只能对应一个联系人，而一个联系人不一定对应唯一的号码）。



③void delete\_member(Per\* S);//删除联系人

注：该函数与insert\_member函数思路上几乎相同，故只做该函数的分析。

第一部分（以电话号码为关键字删除）。从号码对应的哈希地址为起点，循环遍历哈希表直至找到该电话号码。假如遍历了一圈也找不到或者遇见空地址就弹出不存在该联系人的提示。时间复杂度为O(n),空间复杂度为O(1)。

第二部分（以姓名为关键字删除）。从头到尾遍历哈希表，把含有该姓名的记录地址存储并在最后一起输出。时间复杂度为0(n),空间复杂度为O(1)。

问题分析：

Ⅰ.该函数的初期形式比较简陋，只以号码为关键字进行删除。调试了一会才发现是需求分析没有分析完整。后来修改了需求后就加入了以姓名为关键字进行删除的功能并整合成了一个菜单。

Ⅱ.若进行频繁的删除联系人，会突然出现很多冗余的空间。于是考虑在记录总数低于一定值（最大容量的一半）时将最大容量缩减为3/4，不缩为1/2是为了防止出现删除一条记录，空间减半后，又添加一条记录导致容量又翻倍。因为空间变化会使整个哈希表重新排列，数据量大时会比较费时，所以要尽量避免以上情况（也就有了缩减量少于一半空间的想法）。

④void display(Per\* S);//显示联系人

按哈希表顺序将非空的地址处的记录依次输出。

时间复杂度为0(n),空间复杂度为O(1)。

⑤void ClearList(Per\* S);//清空联系人

直接调用函数将哈希表初始化。

时间复杂度为0(1),空间复杂度为O(1)。

⑥int hash\_change(char\* t,Per\* S);//计算哈希地址

将11为电话号码的第5-8位构成的数字加上第9-11位构成的数字，取模（哈希表的最大容量）得到初始哈希地址。

====辅助函数====

①inline int InitList(Per\* S); //初始化哈希表

注：againmalloc() ，declinemalloc()的原理与此函数十分相近，故不再多做分析。

给member,ifempty内容申请动态内存。哈希表的最大容量赋值为MAXSIZE，当前含有记录数赋值为0。

时间复杂度为0(1),空间复杂度为O(1)。

问题分析：

Ⅰ.使用InitList（）初始化时没有考虑好容量初始化的问题。由于忘加MAXSIZE导致频繁的文件读入失败(空指针)。



inline void againmalloc(Per\* S); //扩大容量

inline void declinemalloc(Per\* S); //缩减容量

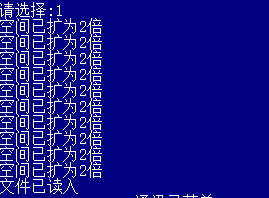
**五、用户使用说明**

本程序为命令行界面，用户将会见到一个选项菜单，只需要输入相应选项的标号即可执行相应程序。其他各种输入将会被系统跳过，重新显示菜单并提示重新输入。菜单如下



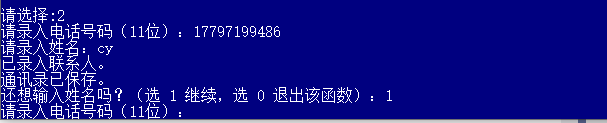
**六、测试结果**

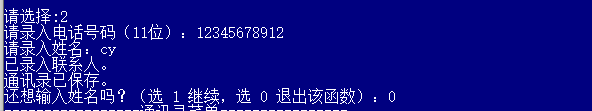
**1.导入联系人文件**



**2.添加联系人**

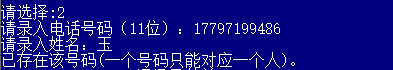
（1）全部合法数据



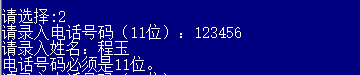


（2）整体非法数据

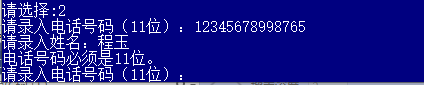
（a）输入相同电话号码不同姓名



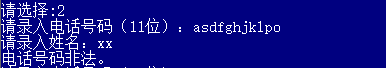
（b）输入不足11位的电话号码

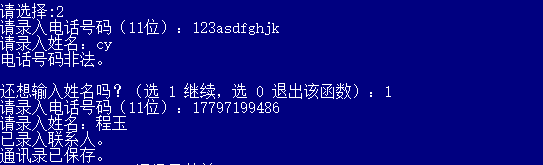


（c）输入超过11位的电话号码

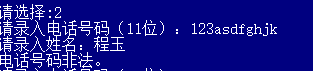


（d）输入其他字符



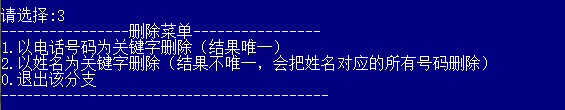


（3）局部非法数据



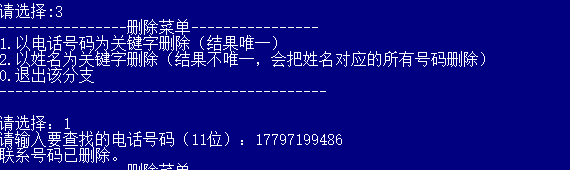
**3.删除联系人**

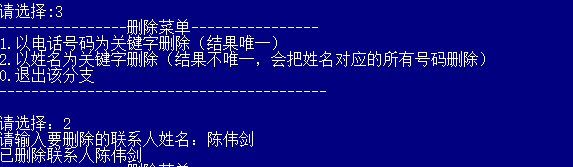
删除菜单；



（a）合法数据

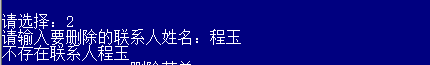
①号码及联系人存在



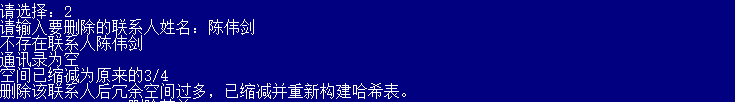


②号码及联系人不存在





③当通讯录为空时



（b）整体非法数据



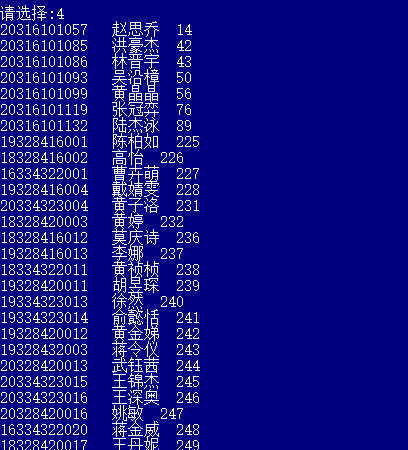
同时会提示操作者是否还想再试一次。

（c）局部非法数据

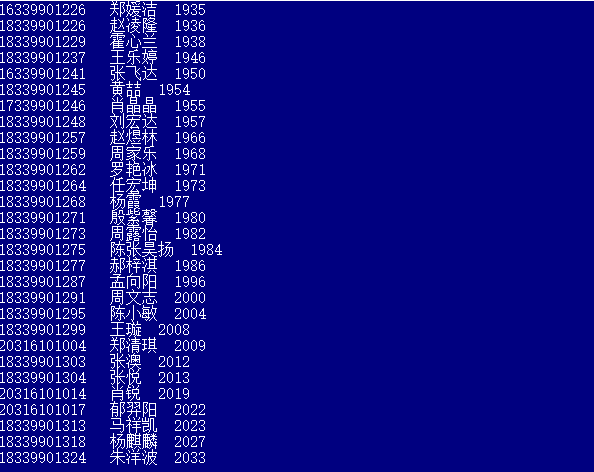


同时会提示操作者是否还想再试一次。

**4.显示联系人**

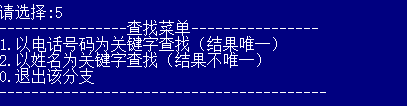
****

**……**

****

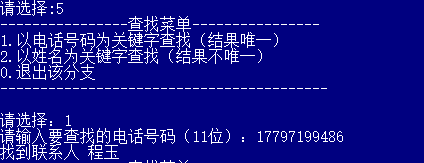
此处只显示部分，整体文件可在文件connect\_save.txt查看

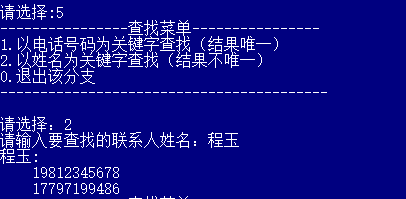
**5.查找联系人**

****

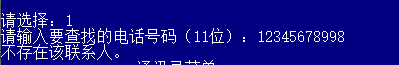
（a）合法数据

①号码及联系人存在





②号码及联系人不存在





（b）整体非法数据



（c）局部非法数据



**6.清空联系人**



**7.保存联系人**

（1）通讯录为空



（2）合法输入



通讯录保存至connect\_save.txt中

**7、附录**

7.1 函数代码

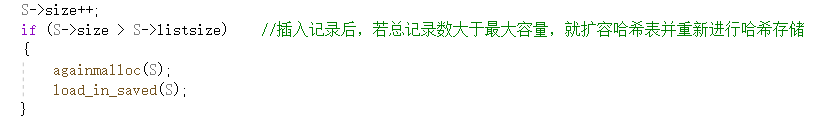
7.1.1 主要函数

①void insert\_member(Per\* S);

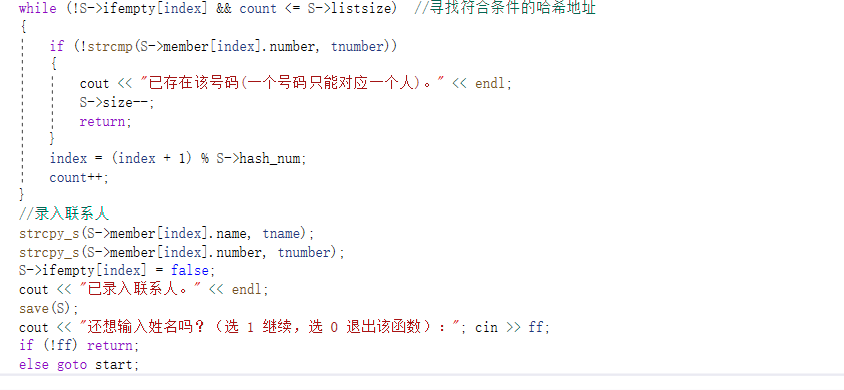
初始化部分



容量处理部分



哈希存储部分



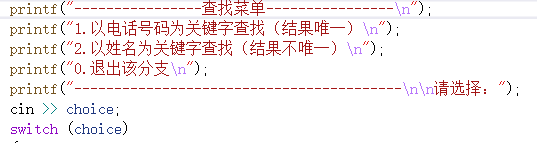
②void search(Per\* S);

注：delete\_member()函数的具体代码与此函数大体相同，故不再贴出。

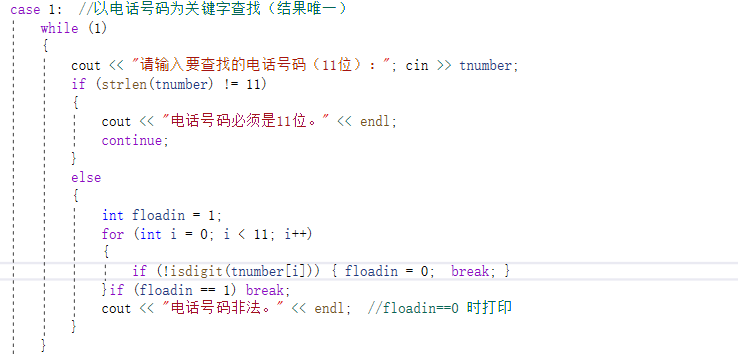
初始化部分

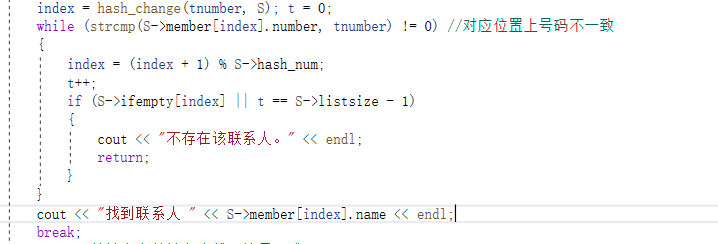


菜单部分

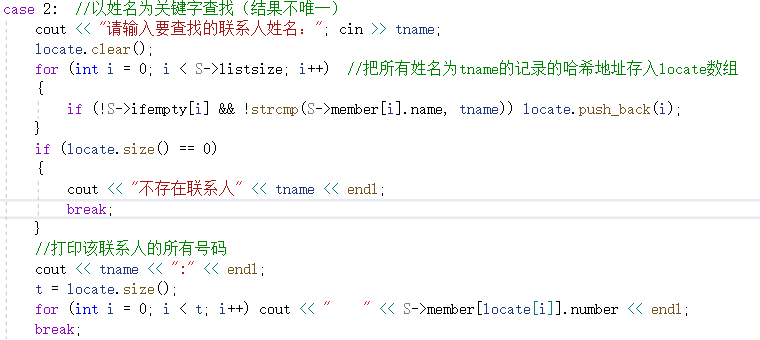


选项1：以电话号码为关键字查找



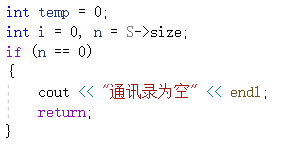


选项2：以姓名为关键字查找

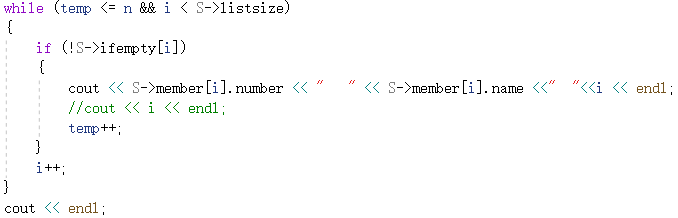


③void display(Per\* S);

初始化部分

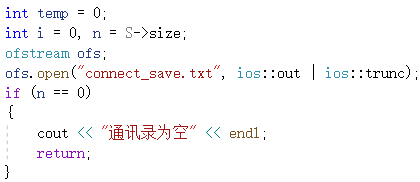


打印部分

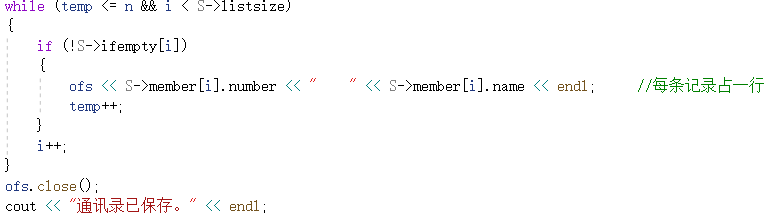


④void save(Per\* S);

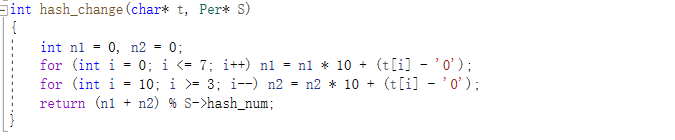
初始化部分



保存部分



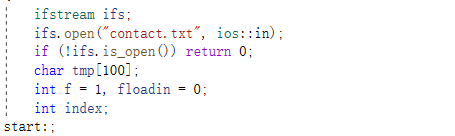
⑤int hash\_change(char\* t,Per\* S);



⑥int load\_in(Per\* S);

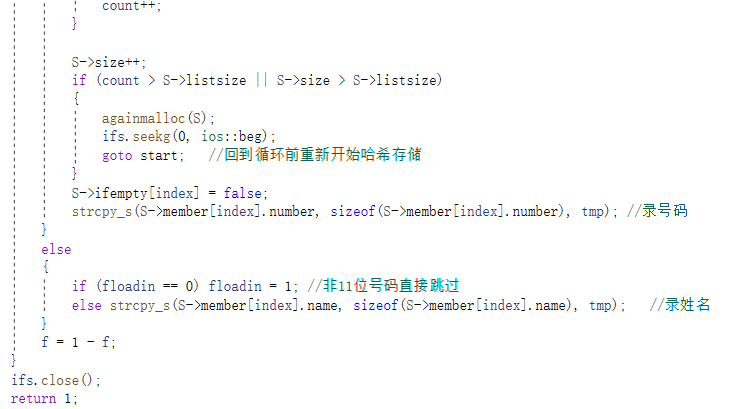
注：load\_in\_saved()具体代码与此函数一样，故不再贴出。

初始化部分



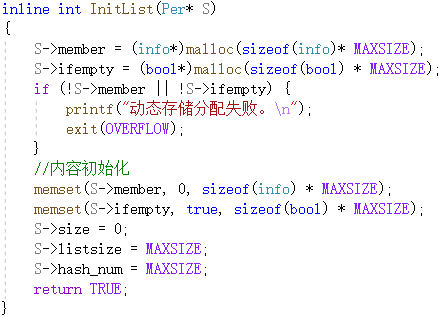
读入部分





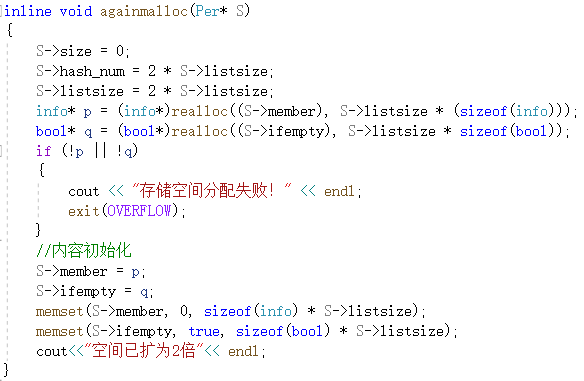
7.1.2辅助函数

①inline int InitList(Per\* S);



②inline void againmalloc(Per\* S);

注：declinemalloc()函数的具体代码与此函数相近，故不再贴出。



7.2小组分工

|  |  |  |
| --- | --- | --- |
|  | 陈伟剑 | 程玉 |
| 需求分析 | 30% | 70% |
| 框架设计 | 75% | 25% |
| 代码编写 | 90% | 10% |
| 程序调试 | 70% | 30% |
| 课设报告 | 30% | 70% |

八、心得体会

陈伟剑：作为第一次完成的通讯录课程设计，通讯录所应该拥有的功能：文件导入、插入记录、删除记录、查找记录、保存记录等，都已实现。不过我认为成品只能算得上是完整通讯录程序的一个大体框架。原因有一下几点：

①没有做好队内分工。由于需求分析主要由队友完成，在队友做需求分析的过程中我独自开始实现整个通讯录框架。实践证明，没有完备的需求分析的情况下，完成一个程序的过程中很容易把握不住方向。比如：在写完一个功能函数后，会突然发现函数内部还可以添加功能或者优化。假如先完成需求分析再去写代码，想必要考虑的东西会少很多。

②技术知识方面仍有待加强。本次通讯录内部每个记录的大小都是确定好的，假如绝大部分记录的姓名只占2-3个单位的空间，那么很容易造成空间浪费。其实一开始就想到了用string模板实现空间的节省，但是出现了乱码、空间越界以及其他仍未弄清楚的代码问题，因此只能退一步用char来存储记录。另外，实现动态空间分配也出现了各式各样的问题，好在这一方面在翻阅技术书籍+上网查资料+咨询老师后都已解决，自己对C/C++的动态内存分配的理解与应用又深入了一层。最后，限于时间与其他课程的原因，C++很多很好的特性没能实际应用在此次课程设计中。

③调试技能还不够强。在遇到BUG时，很多时候没有用好断点、查询内存空间等VS自带的调试功能，导致花了太多的时间去DEBUG。事实上，如何查看内存空间我也是在此次课程设计中才首次用上。

④仍存在相当一部分样例会无法在此程序中通过。主要是因为文件读入函数里有一部分过程比较取巧，整个函数大体是建立在文件内部格式没有问题的大前提下实现的，假如文件发生损坏（如号码多几位或少几位、姓名与号码连在一起等等各式各样的错误格式），本程序的文件读入函数会比较难以处理。不过好在后期追加上了一些方法以实现错误格式的判断。事实上，在完成和完善文件读入函数上所花的时间是最多的。这次经历也让我深刻感受到了一个人完成一个相对智能的文件读入函数是有多么困难。

最后，经历此次课程设计后，我的信息检索能力有所提高，这对将来遇到未知问题后快速寻找解决方法并快速学习想必会有不少好处。另外，以后的小组任务我应该投入更多时间完成小组分工，这是高效完成一个项目的必要条件。希望未来面对更大规模的项目时，我能更从容地完成自己部分的工作、更高效地解决遇见的每一个问题。

程玉：在这次的课程设计中，我掌握的不仅是有关哈希表的知识，更是对数据结构整体的认识，我们小组课程设计的内容是，设计散列表实现通讯录的查找系统，但我们不局限于此，设计了一个完整的通讯录系统，结合生活，完善通讯录的整体功能。有添加、删除、清空联系人等等，当然，最重要的还是查找联系人，我最初想的哈希函数的构造是折叠法，由于通讯录的电话号码比较长，关键字位数较多，因此可以采用折叠法，得到的哈希地址比较均匀。但是在种种实验和调试下，我队友通过将11位电话号码的第5到8位构成的数字加上第9到11位构成的数字来计算电话号码的哈希地址，这个方法的确是很好的一种储存方式，成功解决了通讯录的存储问题，文件中几千条的电话号码可以用Excel表随机生成，姓名是同学的真实姓名，这样我们的通讯录信息就有了来源。  
 在这次课程设计里，我十分感谢我的队友，他认真严谨的学习态度值得我学习，报告中的流程图也是他带我入门，我的每一个问题，他也会耐心的讲解，并且和我讨论调试中的问题，在课设的过程中，我们仔细检查，认真核对，寻找问题，解决问题，完成课设。课程设计，就是把学到的知识灵活的运用到生活中，解决更实际的问题，在学习过程中，不仅巩固了课本里的知识，还学到了很多课外的知识，真是让我受益匪浅，感慨颇多，理论和实践相结合十分重要。课程设计是一个把需求分析，程序编写，程序调试，撰写报告结合为一体的过程，这个过程环环相扣，锻炼了我们的思维和毅力。  
 在今后的学习生活中，我会努力巩固课本中的知识，因为数据结构是一种工具，在以后的工作中也必不可少。再次感谢我的队友，通过向他学习和与他积极配合才有了这份满意的成果，这次课程设计让我满载而归！