**计算机科学与技术 专业课程设计任务书**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **学生姓名** |  | **专业班级** | |  | **学号** | |  |
| **题 目** | 银行存取款管理系统 | | | | | | |
| **课题性质** | 工程设计 | | **课题来源** | | | 自拟 | |
| **指导教师** |  | | **同组姓名** | | | 无 | |
| **主要内容** | 开发一个简单的银行存取款管理系统。  **【数据结构】**  主要包括每个客户的账号、姓名、性别、年龄、电话号码、密码、存款记录金额及日期、取款金额及日期、存款总金额。  **【系统实现功能】**  **1．管理员功能：**（1）能浏览存取款记录；（2）能根据客户姓名、操作日期等信息查询、存款记录；（3）能按客户姓、操作日期等进行排序；（4）能添加客户；（5）可以修改、删除存取款信息等。  **2．客户功能：**（1）客户能够根据自己的账号进行存款功能；（2）客户能够根据自己的账号进行取款功能等。  **3. 功能扩展：**按照自己对该系统的需求理解进行扩展。例如（1）模糊查询；（2）综合查询；（3）统计功能等，并可以对文件中的信息进行读取与显示； | | | | | | |
| **任务要求** | 一、提交材料应包括：（1）系统源代码 （2）课程报告  二、整个设计过程具体要求：（1）**需求分析** 要求学生对案例系统进行分析，设计出需要完成的功能，完善各个模块的调用关系；（2）**设计过程** 要求学生进一步明确各模块调用关系，进一步完善模块函数细节（函数名、参数、返回值等）；（3）**实现过程** 要求学生养成良好的编码习惯、完成各个模块并进行测试，最终完成系统整体测试；（4）**总结阶段** 按照要求完成系统设计和实现报告，并进行总结、答辩。 | | | | | | |
| **参考文献** | [1]甘勇,李晔,卢冰.C语言程序设计（第二版）,北京: 中国铁道出版社,2015.  [2]谭浩强.C程序设计（第五版），北京：清华大学出版社,2017.  [3]严蔚敏,李冬梅,吴伟民.数据结构（C语言版|第2版）,北京：人民邮电出版社,2015.  [4]程杰.大话数据结构,北京：清华大学出版社,2011.  [5]明日科技.C语言经典编程282例,北京：清华大学出版社,2012. | | | | | | |
| **审查意见** | **指导教师签字：**  **教研室主任签字： 2022年2月17日** | | | | | | |

说明：本表由指导教师填写，由教研室主任审核后下达给选题学生，装订在设计（论文）首页

**目录**

[1 需求分析 1](#_Toc28931)

[1.1 系统概述 1](#_Toc26485)

[1.2 系统中的角色 1](#_Toc29213)

[1.3 功能性需求 1](#_Toc20121)

[2 概要设计 3](#_Toc7876)

[2.1 数据结构设计 3](#_Toc27211)

[2.2 模板划分及相关函数定义 5](#_Toc3580)

[2.3 功能结构图 3](#_Toc29641)

[3 运行环境 1](#_Toc17223)2

[4 开发工具和编程语言 1](#_Toc16359)2

[5 详细设计 1](#_Toc9376)2

[5.1 双向链表快速排序 1](#_Toc18134)2

[5.2 KMP算法 1](#_Toc3713)4

[6 调试分析 1](#_Toc9600)6

[6.1 存取款记录单链表模块（record.h） 1](#_Toc18134)6

[6.2 银行卡操作模块（bankcard.h） 1](#_Toc18134)6

[6.3 客户信息单链表模块（idcard.h） 1](#_Toc18134)6

[6.4 查询匹配模块（search.h） 1](#_Toc18134)6

[6.5 排序模块（mysort.h） 1](#_Toc18134)6

[6.6 文件处理模块（myfile.h） 1](#_Toc18134)6

[6.7 页面模块（view.h） 1](#_Toc18134)7

[7 测试结果 1](#_Toc30198)7

[参考文献 2](#_Toc27534)0

[心得体会 2](#_Toc27534)1

**1 需求分析**

* 1. **系统概述**

通过对现有银行系统的调查分析，将其主要功能剖离组合，形成这个初步的银行存取款系统。该系统分为管理员系统和客户系统两部分，共计七个模块，四十六个函数，涵盖了增添客户、办理新卡、记录浏览、存取款、信息查询、转账、注销银行卡、存读文件等基本功能。

**1.2系统中的角色**

表1-1 系统人员分类

|  |  |  |
| --- | --- | --- |
| 角色 | 类别 | 描述 |
| 管理员 | 工作人员 | 系统工作人员，对系统具有相当大的权限，可以进行增添客户、办理新卡或注销、浏览或修改客户信息等功能操作。 |
| 客户 | 服务对象 | 被服务对象，对系统的权限较小，只能对自己名下的银行卡进行基本的存取款、转账、余额查询、修改密码等操作。 |

**1.3功能性需求**

**1.3.1 用户类型**

（1）管理员：系统工作人员，负责对银行客户信息的维护，具有系统赋予的巨大权限，属于系统所属单位的内部人员。

（2）客 户：系统的主要受众，享受系统带来的便利服务，系统赋予的功能较小，只有在拥有银行卡和密码的情况下，才能进行基本操作。

**1.3.2 增加客户**

管理系统基本功能之一，需要完成客户基本信息的填写。客户创建成功后，可以通过注册时所使用的身份证号进行银行卡的申请。每个身份信息只能注册一次，不可重复注册。

**1.3.2 申请新卡**

完成信息注册的客户可通过注册时填写的身份证号进行银行新卡的申请，每个客户最多只能申请四张银行卡号（包括已注销的银行卡号）。

银行卡申请成功后，客户将会获得一个专属的银行卡号，银行卡号与客户注册的信息相绑定。银行卡号组成为：BIN码 + 地方码 + 私人码 + 检验码，共十九位。银行暂定BIN码（62 + 型号）为622911、622913、622915，分别为普通卡、金卡、白金卡。私人码随机生成，检验码通过对前十八位用公式计算得来。

**1.3.4 信息查询**

（1）存取款记录：管理员可以根据本行银行卡号查询到银行卡的存取款记录，记录包含存取款时间、存取款金额、存取款后余额等。即使银行卡号被注销，也可已根据已注销银行卡号找到存取款记录。

（2）银行卡信息：管理员可以根据客户的身份证号，查询到该身份信息名下的所有本行银行卡（包括已注销卡号）。输出界面包括银行卡号、当前状态、开卡时间、及最后存取款操作时间。

（3）身份信息：管理员可以根据本行银行卡号查询卡主身份信息，其中包括卡主姓名、性别、国籍、联系电话、地址、状态等基本信息。

（4）姓名日期查询：管理员可以根据客户的姓名、操作时间（年月日）查询所有该名字下的所有银行卡特定时间的存取款记录。如果年输入为0，则输出所有存取款记录，如果年不为0，月为0，则输出当年所有存取款记录，并以此类推。

（5）模糊查询：管理员可以根据客户姓名片段（关键字）查询并输出所有相关客户信息，其中包括姓名、身份证号、银行卡号等基本信息。该功能采用KMP算法实现，以达到快速精准查找的效果。

（6）信息修改：管理员可以根据客户提供的身份证号对客户的基本信息（姓名、国籍、手机号、地址、通过银行卡号修改密码）进行修改操作，同时更新到该客户名下的所有本行银行卡信息处。在修改时，可以弹出该客户原信息，并输入新信息。

**1.3.5 信息排序**

（1）姓名排序：系统可以根据客户注册时填写的姓名对所有银行卡完成排序输出，并输出内容包括姓名、身份证号、银行卡号等基本信息。该功能采用单链表冒泡排序实现，可对数据结构顺序进行相应修改。

（2）操作时间排序：系统可以根据银行卡最后一次的存取款信息进行排序，时间为最近到最远，并输出排序结果，其中包含姓名、银行卡号、最后存取款信息等基本内容。该排序采用双向链表的快速排序实现，由于存储结构的设计，该功能不能对现有数据结构产生改变。

**1.3.6 银行卡注销**

管理员可以根据客户提供的银行卡号进行银行卡注销操作，若是已经注销，则弹出“已注销”。若是未注销，则进入注销页面，确认是否注销，“1”表示确认注销，“0”及其它非“1”字符均表示取消操作。

注销后，信息将同步更新到银行卡号，客户将无法在客户端通过银行卡号及密码进行登录操作，并弹出“该卡已注销”字样。但该卡依旧可以在管理员系统中的“信息查询”功能中使用。

**1.3.7 存取款和转账**

（1）存取款：客户在客户端系统中通过未注销的银行卡账号密码进行存取款操作，存取款信息可以同步到管理员系统，所取金额不能大于此卡所剩余额。

（2）转 账：客户进入系统后可以选择转账功能，并输入转入卡号，确认后可完成转账操作。转账信息会同步到双方银行卡存取款记录处，以当查证。

（3）修改密码：客户登录之后，可以修改银行卡密码，但要提供原密码，并且新密码需要两次输入确认，若是两次密码不同，则密码修改失败。密码修改成功之后，客户会被强制退出，此时需要用新密码重新登录。

**1.3.8 登陆界面**

（1）管理员系统：管理员通过内部密码可以进入管理员系统，并获取系统赋予的相当大的权限，可进行客户信息的增添、维护、查询和修改等操作。

（2）客户系统：客户可以通过在该行申请的银行卡号及所设置的密码进入客户系统，并进行基本的存钱、取钱和转账等基本操作。

**2 概要设计**

**2.1 数据结构设计**

本系统的存储设计为结构体单链表+结构体数组+结构体单链表的混搭使用，其中还有排序时所使用的双向链表的使用。

**2.1.1 客户基本信息的存储定义**

typedef struct IdCard {

//客户总数量

int count;

//客户姓名

char name[10];

//客户性别

char sex[6];

//客户国籍

char nationality[6];

//客户身份证号

char idCard[20];

//客户移动电话

char mobileTelephone[12];

//客户联系地址

char contactAddress[29];

//一个身份证最多办理四张银行卡

BankCard bankCards[4];

//身份信息标识符

//0-4表示累计的银行卡数量,5表示身份注销

int isFlag;

//指向下一个客户

struct INode\* next;

//存储最后一个结点的地址

struct INode\* tail;

}INode, \* IdcardList;

**2.2.2 客户银行卡存储定义**

typedef struct BankCards {

//银行卡号

char bankCard[20];

//银行卡密码

char passward[8];

//当前余额

double nowBalance;

//存取款信息

RecordList L;

//标识符

//当为0时未办理，当为1时，已办理，当为2时已注销

int isFlag;

}BankCardNode, \* BankCard;

**2.1.3 银行卡存取款存储定义**

typedef struct Record {

//存取款记录金额、存取款后当前余额

double amountOfMoney;

double balance;

//存取款操作年月日、时分秒

//采用整型，用于根据操作时间的排序比较

int year;

int month;

int day;

int yearMonthDay;

int hourMinSec;

//存取款完整日期，方便输出

char operationTime[30];

//标识符,负责判断客户是否具有访问权限

//当为1时，客户可以访问并浏览自己的存取款信息

//当为0时，客户无权访问

int isFlag;

//记录存取款结点个数

int count;

//指向下一条存取款记录的储存地址

struct Record\* next;

//存储最后一个结点的地址，在进行增加存取款记录时不必从头查询

struct Record\* tail;

}RNode, \* RecordList;

**2.1.4 双向链表结点**

typedef struct sort {

//0表示未当过枢轴，1表示当过枢轴，-1表示链表首尾空结点

int isFlag;

//姓名

char name[20];

//身份证号

char Idcard[20];

//银行卡号

char bankCard[20];

//最后操作时间

int yearMonthDay;

int hourMinSec;

char operationTime[30];

//指向上一结点

struct sort\* previous;

//指向下一结点

struct sort\* next;

}SNode, \* MySortList;

**2.2 模块划分及相关函数定义**

**2.2.1 存取款记录单链表模块（record.h）**

算法设计：该模块通过单链表的结构进行存储，头结点不仅存储下一结点的地址，而且存储尾结点的地址，从而提高增加新结点的速率。在进行相关操作时也可以大幅提高尾结点的查询速度。

/\*初始化存取款记录单链表，构造一个空的单链表。并使头结点中的next指针置空，tail指针指向头结点，每增加一个结点记录，都改变tail指针指向尾结点的地址，可加快结点增加效率\*/

RecordList InitRecordList(RecordList L);

/\*增加存取款记录单链表的结点，该节点中存储着该条存取款记录的余额，以及此时银行卡的余额、存取款时间等，并改变头结点的tail指针指向该节点，更新头结点存储的银行卡余额以及最后存取款时间\*/

double addRNode(RecordList L, int isFlag);

/\*预留函数，修改存取款记录单链表结点的数据，仅有单条存取款信息被修改，账户总额不变。不符合银行系统，未实现，但已预留函数，可快速实现\*/

void modifyRNode(RecordList L, double amountOfMoney, double balance, int yearMonthDay, int hourMinSec, char operationTime[]);

/\*通过改变存取款记录单链表特定结点的标记符isFlag的值，使得输出存取款函数无法通过，从而删除单条存取款信息，仅在逻辑上删除单链表信息，在物理上仍旧存在，但此时只有管理员有权限查询\*/

void deleteRNode(RecordList L);

/\*通过改变存取款记录单链表头结点的标记符isFlag的值，使得输出存取款函数无法通过，从而删除单条存取款信息，仅在逻辑上删除单链表信息，在物理上仍旧存在，但此时只有管理员有权限查询\*/

void deleteRecordList(RecordList L);

/\*输出存取款单链表各个结点存储的记录，实现银行卡账户存取款的输出\*/

int printRecordList(RecordList L);

/\*获取当前时间，对存取款新结点的时间变量进行赋值\*/

void getTime(RecordList L);

**2.2.2 银行卡操作模块（bankcard.h）**

算法设计：在存储结构中，银行卡采用固定大小的结构体数组，以确定每个身份信息最多申请四张银行卡号。在进行匹配时，也可以提高速率。

/\*申请银行卡，根据增添客户信息时输入的身份证号申请本行银行卡号，每个身份证号只能申请四张本行身份证号\*/

void RequestBankCard(IdcardList L);

/\*生成银行卡号，根据银行的BIN码、地方码、随机生成的私人码、检验码组成该客户独有的银行卡号\*/

void GenerateBankCard(IdcardList L, int class, int belong);

/\*通过改变银行卡的标识符isFlag，注销银行卡号，仅在逻辑上注销。客户无法再通过银行卡号和密码登录，但管理员仍旧可以根据银行卡查询所有信息\*/

void CancelBankCard(BankCard bc);

/\*通过改变银行卡标识符，恢复银行卡号\*/

void RecoverBankCard(BankCard bc);

/\*根据输入的身份证信息，输出该身份信息名下的所有银行卡号，及各银行卡的开户时间、状态、最后存取款时间等信息\*/

void PrintBankcard(IdcardList L);

/\*通过改变银行卡的标识符isFlag，注销银行卡号，仅在逻辑上注销。客户无法再通过银行卡号和密码登录，但管理员仍旧可以根据银行卡查询所有信息\*/

void Logout(IdcardList L);

/\*转账功能，客户进入系统后可以进行转账操作，目前仅支持本行银行卡的行内转账，需要提供收款银行卡号，转账金额不能超过该银行卡的余额\*/s

double Transfer(IdcardList IdL, RecordList ReL);

**2.2.3 客户信息单链表模块（idcard.h）**

算法设计：该模块通过单链表的结构进行存储，头结点不仅存储下一结点的地址，而且存储尾结点的地址，从而提高增加新结点的速率。

/\*初始化身份信息链表，在程序启动阶段进行，有利于文件信息的读取或客户信息的增添\*/

IdcardList InitIdCard(IdcardList L);

/\*添加身份信息，管理员在该函数中需要进行新客户基本信息的输入。新客户的省份证号不能与之前已经办理该业务的身份证号重复\*/

void AddIdCard(IdcardList L);

/\*删除单个客户身份信息结点，仅在逻辑上删除单链表信息，在物理上仍旧存在，但此时只有管理员有权限查询\*/

void deleteINode(INode\* N);

/\*删除全部客户信息记录，通过改变标识符isFlag进行操作。仅在逻辑上删除单链表信息，在物理上仍旧存在，但此时只有管理员有权限查询\*/

void deleteIdcardList(IdcardList L);

/\*根据银行卡号输出卡主身份信息，可以根据输入的银行卡号匹配该银行卡号拥有者的基本信息\*/

void PrintIdCard(IdcardList L);

/\*信息修改，管理员通过客户提供的身份证号进入该客户的信息修改页面，其中包括姓名、地址、国籍、手机号、银行卡密码等，修改银行卡密码还需要输入要修改银行卡密码的银行卡号\*/

void ChangeInfo(IdcardList L);

/\*修改姓名，可以改变客户的姓名\*/

void ChangeName(IdcardList L);

/\*修改地址，可以改变客户的地址\*/

void ChangeAddress(IdcardList L);

/\*修改国籍，可以改变客户的国籍\*/

void ChangeNationality(IdcardList L);

/\*修改手机号，可以改变客户的手机号\*/

void ChangenMobileTelephone(IdcardList L);

/\*修改银行卡密码,客户可以自行修改密码，需要提供旧密码。也可以将银行卡号和旧密码提供给管理员，由管理员进行操作。客户修改密码后原登录将会失效，需要使用新密码重新登录\*/

int ChangePassward(BankCard myAccount);

**2.2.4 查询匹配模块（search.h）**

算法设计：通过传递身份链表头结点的地址，无论传递身份证号还是银行卡号都可以进行匹配，匹配为依次顺序匹配。

/\*找到存储身份证信息地址，通过输入的银行卡号查找存储该卡号所属主人的身份信息的单链表结点地址\*/

IdcardList findIdCard(IdcardList L);

/\*客户登录界面账号密码匹配，对客户输入的银行卡号及密码进行匹配，如果成功则进入系统，失败则返回主界面\*/

BankCard findMyAccount(IdcardList L, char account[], char passward[]);

/\*通过输入的银行卡号找到银行卡存储地址，从而获取该银行卡存取款单链表的头结点地址，输出数据，从而实现查找存取款记录的功能\*/

BankCard findInforAccount(IdcardList L, char account[]);

/\*根据银行卡号找到户主，输入该行银行卡号可以匹配到该卡拥有者的基本身份信息\*/

IdcardList findOwner(IdcardList L, char account[]);

/\*根据姓名日期查找存取款记录，如果输入的时间数字为0或其它字符，则说明该单位不限，若是年不限，则月日自动不限，以此类推\*/

void SearchByNameDate(IdcardList L);

/\*模糊查询，可以通过输入户主名字片段查询户主信息，采用KMP算法实现。KMP算法具体实现为下述两个函数\*/

void FuzzySearch(IdcardList L);

/\*KMP匹配字符片段，如果查找成功返回模板字符串在目标字符串位置，如果查找失败返回-1。第一个参数为目标字符串，第二个为模板符串\*/

int KMP(char dstStr[], char modelStr[]);

/\*KMP算法调用函数，否则获取模板的next数组，是KMP算法的核心部分\*/

void getNextArr(char modelStr[], int\* next);

**2.2.5 排序模块（mysort.h）**

算法设计：排序中用到的全为地址，可以降低空间溢出现象的发生，两个功能的排序分别使用了冒泡排序和快速排序。

/\*冒泡排序，按客户姓名进行排序，存储结构为单链表\*/

void SortByName(IdcardList L);

/\*按客户的最后操作时间进行排序，存储结构为为双向链表，实现方法为纯指针的操作，只通过改变结点的指向实现\*/

void SortByTime(IdcardList L);

/\*快速排序，双向链表的快速排序方法\*/

void QSort(MySortList L, MySortList low, MySortList high);

/\*快速排序调用函数，获取每轮快速排序获得的中间值地址\*/

MySortList Partition(MySortList L, MySortList low, MySortList high);

**2.2.6 文件处理模块（myfile.h）**

算法设计：按照结构体主体数据的格式进行数据的存储和读取，其中数据又分为三部分，分别为客户身份基本信息、银行卡基本信息、银行卡存取款记录，将数据拆分存储。

/\*文件读取函数，可以将保存到文件中的数据进行读取，并且可以在程序中进行相应的增删改查排序等，共有客户基本信息、银行卡基本信息、银行卡存取款记录信息三个文件\*/

void ReadFile(IdcardList L);

/\*信息存储函数，可以将程序中的主要数据信息保存到文件中，使得下次运行程序能够进行读取。该函数共使用三个文件，分别存储客户基本信息、银行卡基本信息、银行卡存取款记录信息\*/

void SaveFile(IdcardList L);

**2.2.7 页面模块（view.h）**

算法设计：该模块主要为使用页面的加载，采用while循环和switch的相互搭配，使得各个页面能够实现切换。

/\*主菜单页面，启动系统后的主页面，可选择管理员系统和客户系统，选择后登录进入\*/

void MainView();

/\*管理员页面，其中有管理员可进行的添加客户、办理新卡、信息查询、信息排序、注销银行卡等五大功能模块\*/

void ManageView(IdcardList L);

/\*在主页面选择管理员系统后，需要输入管理员密码才能进入，该函数负责匹配输入的管理员密码是否正确\*/

void ManagePass(IdcardList L);

/\*客户页面，选择客户系统后将自动跳转到该页面并进行登录操作\*/

void CustomerView(IdcardList L, BankCard myAccount);

/\*客户登录函数，客户在此页面可根据银行卡号及密码进行登录操作\*/

void CustomerPass(IdcardList L);

/\*信息查询页面，管理员在该页面可进行存取款记录、银行卡信息、身份信息、姓名日期查记录、模糊查询、信息修改等操作\*/

void InforSearchView(IdcardList L);

/\*输出特定客户的存取款信息，根据输入的银行卡号输出该卡的存取款信息，其中包括存取款金额、余额、操作时间等\*/

void PrintRecord(IdcardList L);

/\*信息排序界面，管理员可在本页面选择姓名排序或最后存取款操作时间排序等排序操作\*/

void InforSort(IdcardList L);

**2.3 功能结构图**

功能结构图如图2-1所示：

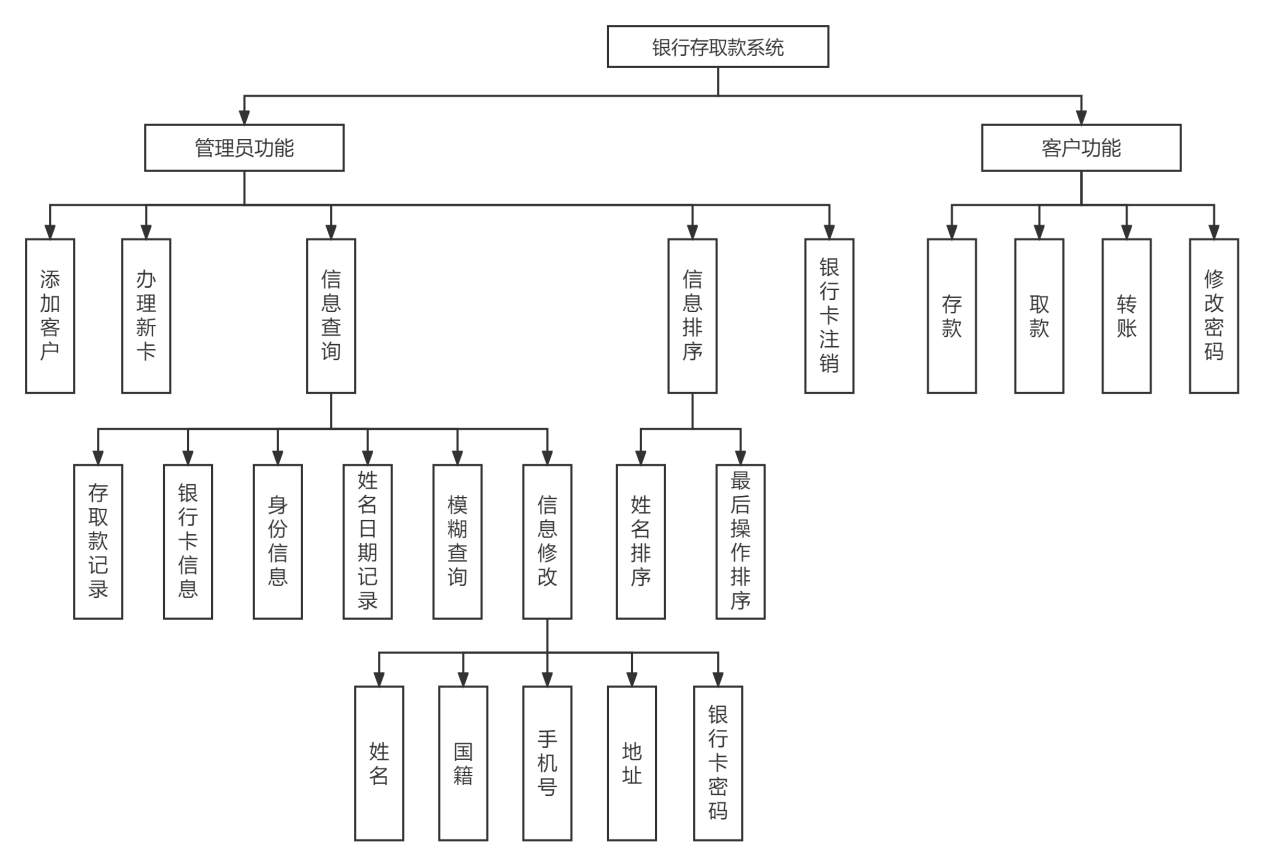
****

图2-1 系统结构图

**3 运行环境**

硬件环境： PC机内存：16G 硬盘：512G

软件环境： 操作系统：windows10

**4 开发工具和编程语言**

开发环境：Microsoft Visual studio 2022

编程语言：C语言

**5 详细设计**

**5.1 双向链表快速排序**

双向链表快速排序，需要首先定义四个相关类型的地址变量，负责通过改变结点内部指针的指向实现排序的功能。在进行快速排序过程中，要注意判断快速排序的结束标志，我采用设置声明变量的方式，每一个结点充当“哨兵”之后都会对其声明变量进行数值上的改变，在进行快速排序时可以充当范围的结束标志，从而使得快速排序能够结束。同时，为了能够使双向链表能够正常排序，采用的双向链表为首位空结点的形式。

void QSort(MySortList L, MySortList low, MySortList high) {

if (low == high) {

return;

}

MySortList pivotloc = Partition(L, low, high);

if (pivotloc->previous->isFlag != -1) {

if (pivotloc->previous->isFlag == 1) {

low = pivotloc->previous;

}

else {

low = pivotloc;

while (1) {

low = low->previous;

if (low->previous->isFlag != 0) {

break;

}

}

QSort(L, low, pivotloc->previous);

}

}

if (pivotloc->next->isFlag != -1) {

if (pivotloc->previous->isFlag == 1) {

high = pivotloc->previous;

}

else {

high = pivotloc;

while (1) {

high = high->next;

//碰到枢轴或首尾空结点

if (high->next->isFlag != 0) {

break;

}

}

QSort(L, pivotloc->next, high);

}

}

}

//快速排序调用函数

MySortList Partition(MySortList L, MySortList low, MySortList high) {

MySortList isFlag = low;

isFlag->isFlag = 1;

MySortList SortOne;

MySortList SortTwo;

MySortList SortThree;

MySortList SortFour;

int i = 0;

while (low != high) {

/\* printf("%s\n%s\n", low->operationTime, high->operationTime);

system("pause");\*/

//比较是相邻两个结点，即最后一次比较

if (low->next == high) {

if (strcmp(low->operationTime, high->operationTime) < 0) {

SortOne = low->previous;

SortTwo = high->next;

SortOne->next = high;

SortTwo->previous = low;

high->previous = SortOne;

high->next = low;

low->previous = high;

low->next = SortTwo;

low = SortOne->next;

high = SortTwo->previous;

}

break;

}

if (strcmp(low->operationTime, high->operationTime) < 0) {

SortOne = low->previous;

SortTwo = low->next;

SortThree = high->previous;

SortFour = high->next;

SortOne->next = high;

SortTwo->previous = high;

SortThree->next = low;

SortFour->previous = low;

low->previous = SortThree;

low->next = SortFour;

high->previous = SortOne;

high->next = SortTwo;

low = SortOne->next;

high = SortThree->next;

i++;

}

if (i % 2 == 0) {

high = high->previous;

}

else {

low = low->next;

}

}

return isFlag;

}

**5.2 KMP算法**

KMP算法在匹配字符串中相对传统方法具有明显的高效性，其核心算法为next数组的获取，而next数组的获取与模板字符串有关，根据其所有位置上出现最大前缀和最大后缀的+1下标位进行存储，获取一个在字符匹配过程中能够快速跳转的数组，从而极大的提高了匹配的效率。

int KMP(char dstStr[], char modelStr[])

{//KMP匹配字符片段，如果查找成功返回位置，如果查找失败返回-1

int i = 0;

int j = 0;

int dstlen = strlen(dstStr);

int modellen = strlen(modelStr);

int next[255] = { 0 };

getNextArr(modelStr, next);

while ((i < dstlen) && (j < modellen)) {

if (j == -1 || dstStr[i] == modelStr[j]) {

i++;

j++;

}

else {

j = next[j];

}

}

if (j >= modellen) {

return i - modellen; // 匹配成功 返回子串位置

}

else {

return -1;

}

}

//KMP算法调用函数

void getNextArr(char modelStr[], int\* next)

{

// 初始化next数组第一位为-1

int i = 0;

int j = -1;

next[0] = -1;

//int mLen = sizeof(modelStr) / sizeof(modelStr[0]);

int mLen = strlen(modelStr);

while (i < mLen - 1) {

// 求最大前缀=最大后缀的过程

if (j == -1 || modelStr[i] == modelStr[j]) {

i++;

j++;

next[i] = j;

}

else {

// 当没有匹配上 则进行回溯 回溯的位置为next数组的指向的下一个匹配项

j = next[j];

}

}

}

**6 调试分析**

**6.1 存取款记录单链表模块（record.h）**

问题：在进行新结点的增加时，需要一直从头所引导尾结点，效率低下。可以通过记录尾结点的地址达到快速找到尾结点的操作。

改进设想：通过在单链表的头结点增加节点地址变量存储尾结点的地址。

**6.2 银行卡操作模块（bankcard.h）**

问题：注销银行卡，如果直接删除，会导致数据的永久丢失，但是另开辟空间存储又会浪费时间和空间。

改进设想：设置结点变量，通过改变结点变量的数值决定其状态，不仅可以快速操作，还不会造成数据的丢失。

**6.3 客户信息单链表模块（idcard.h）**

问题：在进行新结点的增加时，需要一直从头所引导尾结点，效率低下。可以通过记录尾结点的地址达到快速找到尾结点的操作。

改进设想：通过在单链表的头结点增加节点地址变量存储尾结点的地址。

**6.4 查询匹配模块（search.h）**

问题：在模糊查询功能中，如果通过字符串的一个一个字符的匹配，会造成时间上的极大浪费。KMP算法使用过程出现错误，使得查询结果有误差。

改进设想：字符串匹配可以采用KMP算法。通过改变字符串的大小取值方式解决next数组获取不准确的问题。

**6.5 排序模块（mysort.h）**

问题：由于存储结构（单链表+线性表+单链表）的设置，银行卡信息存在线性表中，使得程序无法对银行卡进行排序。

改进设想：开辟新空间存储银行卡基本信息，采取双向链表的存储结构进行快速排序。

**6.6 文件处理模块（myfile.h）**

问题：在文件的保存和读取过程中会出现信息读取超出存储的部分，使得程序运行出错。可以在开始读取结点数据个数，以达到稳定读取的效果。

改进设想：增加变量，存储节点个数，控制读取操作。

**6.7 页面模块（view.h）**

问题：页面切换时，有的页面在出错后不能重新加载，直接回到了上一个页面，使得操作繁琐。

改进设想：页面通体使用while循环和switch语句，使得页面在非自退出情况下不会强制退出。

**7 测试结果**

**参考文献**

[1]甘勇,李晔,卢冰.C语言程序设计（第二版）,北京: 中国铁道出版社,2015.

[2]谭浩强.C程序设计（第五版），北京：清华大学出版社,2017.

[3]严蔚敏,李冬梅,吴伟民.数据结构（C语言版|第2版）,北京：人民邮电出版社,2015.

[4]程杰.大话数据结构,北京：清华大学出版社,2011.

[5]明日科技.C语言经典编程282例,北京：清华大学出版社,2012.

**心得体会**

本次课程设计，我认识到了在编写代码之初，对系统整体勾画的重要性，这可以帮助我们走许多弯路。基本函数的书写规范以及相互调用的合理性尤为重要。并且在此次实践中，我也尝试了许多第一次，感受到了分模块编写代码的便利性。也在功能的不断增添中修改各种随之而来的bug使得自身的编程能力得到了提高。

也是在此次课设中，我意识到了算法对于程序的重要性，算法可以说是程序的灵魂，我也会在课下之余学习提高对算法的学习，使自己的逻辑思维能力得到提高，从而编写出效率更高效的代码。