C++实验报告

姓名 沈俞霖

学号 2151601013

实验一．银行卡管理。

一．问题描述：

实验一．银行卡管理。

问题描述：

用VC++可视化开发平台实现某银行的储蓄卡的管理程序。设：某银行共发出M张储蓄卡，每张储 蓄卡拥有唯一的卡号，每天每张储蓄卡至多支持储蓄卡持有者的N笔“存款” 或“取款” 业务。

程序中用数组card[M][N+3]中的每一行存放一张储蓄卡的有关信息，其中；

card[i][0] 存放第i张卡的卡号；

card[i][1] 存放第i张卡的余额；

card[i][2] 存放第i张卡的当日业务实际发生笔数；

card[i][3]~card[i][N+2]存放第i张卡的当日存取款金额，正值代表存款，负值代表取款。

当持卡者输入正确的卡号、存款或取款金额后，程序进行相应的处理；若输入了不正确的数据，程序会提示持卡者重新输入；若输入的卡号为负数时，银行终止当日业务。并将当天每张卡的银行业务存到文件中。

二．实验内容/程序实现

1.项目管理

使用github进行协作

(以下是git shortlog)

Crux10086 (1):

Update README.md

sylxjtu (21):

:octocat: Added .gitattributes & .gitignore files

Initialize

change .gitignore

Delete bank.pro.user

add readme

修改了接口，等待审核

统一大小写

设计UI界面（菜单栏）

尝试做了一下登录界面

完善登录界面

测试登录界面

把按值传递修改为按引用传递，减少复制开销

update giytignore

重构

update

重写readme

Merge remote-tracking branch 'origin/UI'

重构bank，以及添加列表功能

修复bug，完善交易详单查询

源码地址: <https://github.com/sylxjtu/CppBigassignment/tree/master/bank>

2.使用的第三方库

UI库：Qt５.４

内容存取：SQLite３

3.程序设计思路

使用数据库存取数据，保证数据的稳定性和可持久性

采用MVC结构，使用一个类作为UI界面和数据库通信的接口

使用SQL语句访问数据库来读取数据

**Qt5**

**MainWindow**

**NewAccount**

**DeleteAccount**

**CardList**

**Trades**

显示界面

(View)

数据交换

（Controller）

数据库

（Model）

**Bank**

**SQLite3**

**SQL**

**Qt API**

4.主要代码

主要类：bank类

#ifndef BANK\_H

#define BANK\_H

#include "sqlite3.h"

#include <QtCore>

class Bank{

QString dbname;

void databaseInit();

public:

sqlite3\* interface;

Bank(const QString& name);

~Bank();

int findAccount(const QString& ID);

int createAccount(const QString& ID, const QString& password);

int deleteAccount(const QString& ID);

int validate(const QString& ID, const QString& password);

int save(const QString& ID, double Amount);

int withdraw(const QString& ID, double Amount);

double query(const QString& ID);

};

#endif // BANK\_H

初始化

Bank::Bank(const QString &name){

dbname = name;

QByteArray ba = name.toLatin1();

int needinit = 0;

if(!fileExists(ba.data()))

needinit = 1;

sqlite3\_open(ba.data(),&interface);

if(needinit)

databaseInit();

}

析构/销毁

Bank::~Bank(){

sqlite3\_close(interface);

}

SQLite使用示例:创建账号

int Bank::createAccount(const QString &ID, const QString &password){

QString sql;

char\* err;

QByteArray ba;

if(findAccount(ID))

return 1;

sql = "INSERT INTO accounts (ID, password, money) VALUES (";

sql = sql + ID;

sql = sql + ",";

sql = sql + password;

sql = sql + ",0)";

ba = sql.toLatin1();

if(sqlite3\_exec(interface, ba.data(), InitCallback, 0, &err) != SQLITE\_OK){

std::cerr << err;

throw(err);

}

else

return 0;

}

细节处理:使用正则表达式验证用户输入

MainWindow::MainWindow(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

b = new Bank(QString("bank.db"));

QRegExp re(QString("^[0-9]+$"));

ui->cardIDLineEdit->setValidator(new QRegExpValidator(re,this));

QRegExp re2(QString("^([0-9]){6}$"));

ui->passwordLineEdit->setValidator(new QRegExpValidator(re2,this));

}

程序运行演示

|  |  |
| --- | --- |
| 程序首屏 | 菜单栏 |
| 创建账号 | 删除账号 |
| 账号列表 | 查询交易详单 |
| 错误1（密码长度不足） | 错误2（密码错误） |
| 错误3（余额不足） | 存款操作 |
| 取款操作 | 查询余额操作 |
| 用户界面设计(快速查找) | 用户界面设计(精确匹配优先) |