语言课程设计实验报告

姓　名	王晨阳
学　号	57119101
专　业	网络空间安全
起始日期	2021 年 7 月
结束日期	2021 年 7 月
总评（教师填写）	

设　计　文　档

设计任务

银行储蓄管理系统

[要求] 该系统需创建和管理以下信息：1、储户信息：帐号、姓名、密码、地址、储种(定期1年、3年、5年)、利息（1年到期利率1.98%；3年到期利率2.25%：5年到期利率 3 .5%）、是否已挂失、挂失日期、营业员工号、存取款数据集；2、存取款信息：日期、金额、业务类型(存或取)、营业员工号。

系统功能要求如下：

1．创建和管理储户：创建储户对象(开户)；

2．登陆账户：根据储户的帐号、密码登陆账户，有则登录。1）在里面可修改储户信息、账户挂失等；2）办理存取款业务，将存取款对象数据加入到储户对象的数据成员中；

3. 基本查询功能；

4．数据文件读写：文件中包含所有储户信息、每个储户的存取款业务等数据；

5．基本信息显示：1）显示所有储户信息；2）显示特定储户的存取款业务；

6. 可选功能提升：统计今后 3 天每天到期的储金数，以便备款；2）本金和利息计算等。

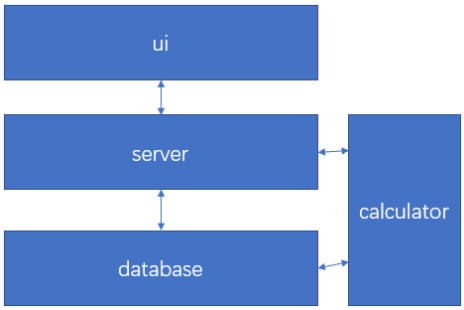
扩展功能

1. 用户列表、流水列表导出到 html 表格

设计实现的方法

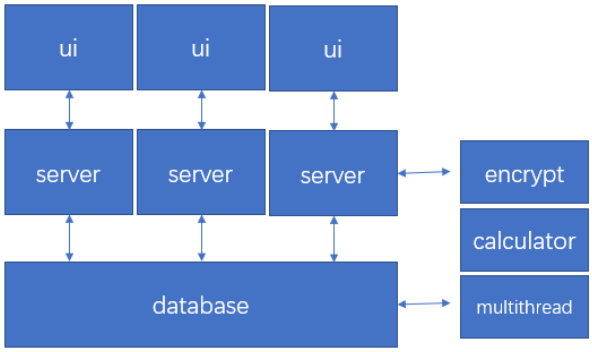
1. 软件结构

目前的软件结构模型如下：



最下层 database 实现数据存取、密码验证；中间层 server 实现合法性验证、用户登录等，沟通数据库与用户界面；最上层为用户界面，所有用户界面等级相同，互相切换; calculator 为辅助函数库，同时与数据库及服务层交互。

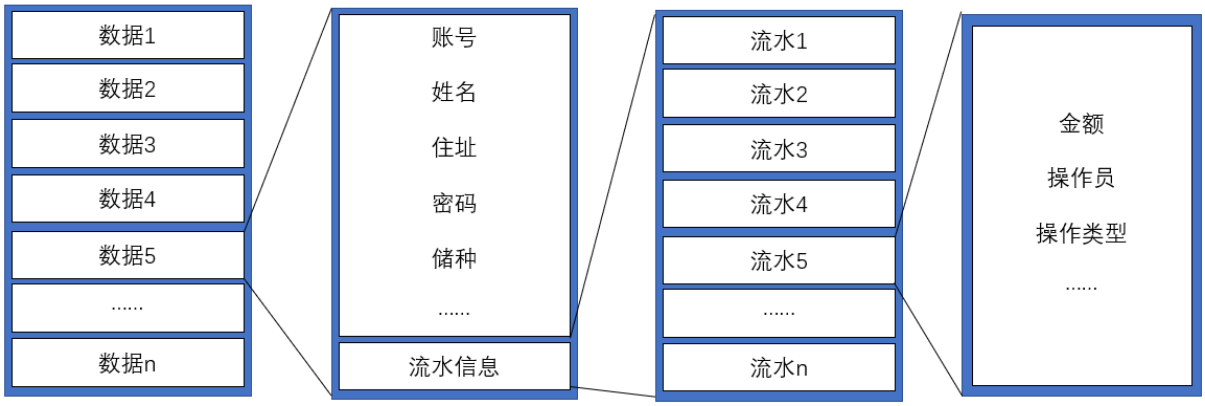
对于上面的模型，中间层 server 看起来可以和最底层合并，但这样的设计是为了更方便扩展为最终版本：



这是一个**多用户、易扩展、层次化**的模型。

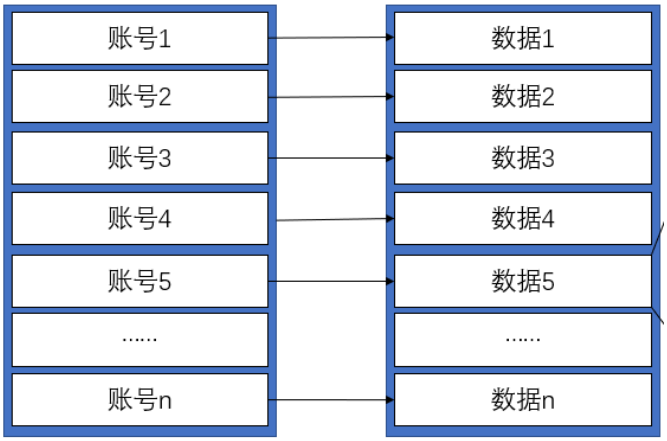
2. 数据库实现

数据库采用 4 层嵌套结构，存储结构如下：



我们使用 vector 实现可变容、可按索引查找的数据存储。

为了实现更加高效的账号查找，我们使用 map 进行了映射，如下图所示



这将账号搜索的时间复杂度从 O(n) 降低到了 O(1)。

为了确保数据安全，防止软件运行中因不可抗拒因素强制关闭，我们规定数据库在内存中进行任何读写后都立即将数据写入磁盘文件。这是以效率换稳定，增强软件鲁棒性。

3. 利息计算

由于存在非同时存储、不确定取款金额、滚利计算等几大难点，我们创造性地使用了链表式流水表单，如下图所示

每当有存款时，链表向后增加一条记录；

每当有取款时，链表检查最后一项，如果小于要取的金额，则扣除取款金额；否则删除最后一项，继续比较前面的项；

每当要查询时，只需从表头扫描到表尾，计算出每一个存款记录的利息。

这样的结构使得分开计算每一笔存款成为可能。

4. 其余功能实现

由于需求较为简单，其余功能全部简单模拟即可。

5. 功能特色

- 创建账户时，账号自动生成，作为账户唯一标识符
- 密码存取实现严格管理，仅数据库有读取密码的权限，同时隔离数据库与用户操作界面
- 一键以标准格式导出数据，方便数据查看、携带及二次处理
- 性能优化，账户数目较多时，各项操作效率优于平凡算法
- 数据保护，始终同步内存与外存，软件鲁棒性大大提高

关键代码	**软件所有代码均使用 git 进行管理，并同步至*我的 GitHub 仓库（点击查看）*** **以下为数据库模块、计算模块的关键代码以及详细注释：**

说明

bankdb.h

```
/*****************************************************************************
 * Copyright (C) 2021 Chenyang https://wcy-dt.github.io                      *
 *                                                                           *
 * @file     bankdb.h                                                        *
 * @brief    Bankdb simulates a database for the project. The speed of       *
 *           this database is slow for that we must ensure the safety        *
 *           of every deal.                                                  *
 * @author   Chenyang                                                        *
 * @date     2021 - 07                                                       *
 *****************************************************************************/

#ifndef BANKDB_H
#define BANKDB_H

#include <map>
#include <string>
#include <vector>

using namespace std;

class bankdb
{
public:
    bankdb();

    /**
     * @defgroup file operations
     */
    void GetFile();
    void WriteFile();

    /**
     * @defgroup account operations
     */
    void AddAccount(string, string, string, string, int, double, string);
    void EditAccount(string, string, string, int, double);
    void EditPasswd(string, string);
    void SetLost(string, time_t);
    bool ExistAccount(string);
    bool CheckAccount(string, string);

    /**
     * @defgroup get information by account number
     */
    string GetName(string);
```

```
    /**
     * @defgroup get information by the index in vAccount
     */
    string GetName(int);

    /**
     * @defgroup get number
     */
    int GetNumberOfAccounts();
    int GetNumberOfFlows(string);

    /**
     * @defgroup get information of the flow by account number
     *           and index of the flow
     */
    time_t GetTime(string, int);
    double GetMoney(string, int);
    int GetOperationType(string, int);
    string GetOperator(string, int);

    /**
     * @defgroup deposit and withdraw money
     */
    void DepositMoney(string, time_t, double, string);
    bool WithdrawMoney(string, time_t, double, string);

private:
    /**
     * @brief This structure records the flow for every account.
     */
    struct flowInfo
    {
        time_t tTime;       /// when the deal happens
        double dMoney;      /// how much money
        int iOperationType; /// the type of the operation,
                            /// -# 0:deposit
                            /// -# 1:withdraw
        string strOperator; /// the operator of the deal
    };

    /**
     * @brief This structure records the information of every accounts.
     */
    struct accountInfo
    {
```

```
        string strNumber;       /// account number, generated automatically
        string strName;         /// name of the account owner
        string strPasswd;       /// password of the account
        string strAddress;      /// address of the account owner
        int iType;              /// type of the account
                                /// -# 0:1 year
                                /// -# 1:3 year
                                /// -# 2:5 year
        double dInterest;       /// interest, set according to the type
        bool bLost;             /// lost or not
        time_t tLostTime;       /// if lost, this variable has some value
        string strOperator;     /// the operator who open the account
        vector<flowInfo> vFlow; /// flow of the account
    };

    vector<accountInfo> vAccount; /// use a vector to store all the account information
    map<string, int> mAccount; /// map account number to the index of the account in vAccount
    map<string, double> mMoney; /// map account number to how much money the account has
};

#endif /// BANKDB_H
```

bankdb.cpp

```
/*************************************************************************
 * Copyright (C) 2021 Chenyang https://wcy-dt.github.io                  *
 *                                                                       *
 * @file     bankdb.cpp                                                  *
 * @brief    Bankdb simulates a database for the project. The speed of   *
 *           this database is slow for that we must ensure the safety    *
 *           of every deal.                                              *
 * @author   Chenyang                                                    *
 * @date     2021 - 07                                                   *
 *************************************************************************/

#include "bankdb.h"
#include <QDebug>
#include <QDialog>
#include <QMessageBox>
#include <fstream>
#include <map>
#include <time.h>

using namespace std;
```

```
/**
 * @brief initialization and read from the file
 */
bankdb::bankdb()
{
    GetFile();
}

/**
 * @brief read from the file which stores the data
 */
void bankdb::GetFile()
{
    vAccount.clear();
    mAccount.clear();
    mMoney.clear();

    int iNumOfAccounts; /// number of accounts
    int iNumOfFlow;     /// number of flows

    ifstream iFile("bankdb.db", ios::binary);
    if (!iFile) /// the file not exist
    {
        iNumOfAccounts = 0; /// zero account
        WriteFile();
        return;
    }

    iFile >> iNumOfAccounts;

    for (int i = 0; i < iNumOfAccounts; i++)
    {
        accountInfo tmpAccount; /// get a record, and pushback to vector
        string strTmpString;    /// for the convenience of infile

        /**
         * @note here i use a lot of getline(), for that spaces may exist in the strings
         *       getline(iFile, strTmpString) is for jump over ENTER
         */
        iFile >> tmpAccount.strNumber;
        getline(iFile, strTmpString);
        getline(iFile, tmpAccount.strName);
        getline(iFile, tmpAccount.strPasswd);
        getline(iFile, tmpAccount.strAddress);
        iFile >> tmpAccount.iType >> tmpAccount.dInterest >> tmpAccount.bLost >> tmpAccount.t
LostTime;
```

```
        getline(iFile, strTmpString);
        getline(iFile, tmpAccount.strOperator);
        iFile >> iNumOfFlow;

        mMoney[tmpAccount.strNumber] = 0; /// initialize the money in the account

        for (int j = 0; j < iNumOfFlow; j++)
        {
            flowInfo tmpFlow; /// get a flow, and pushback to vector

            iFile >> tmpFlow.tTime >> tmpFlow.dMoney >> tmpFlow.iOperationType;
            getline(iFile, strTmpString);
            getline(iFile, tmpFlow.strOperator);

            if (tmpFlow.iOperationType == 0) /// deposit
                mMoney[tmpAccount.strNumber] += tmpFlow.dMoney;
            else /// withdraw
                mMoney[tmpAccount.strNumber] -= tmpFlow.dMoney;
            tmpAccount.vFlow.push_back(tmpFlow);
        }

        mAccount[tmpAccount.strNumber] = vAccount.size(); /// map account number to the index
                                                          /// @see bankdb.h
        vAccount.push_back(tmpAccount);
    }
}

/**
 * @brief write data to the file
 */
void bankdb::WriteFile()
{
    int iNumOfAccounts = vAccount.size();

    /**
     * @note use trunc for that it is not much slower than find where to insert or edit
     */
    ofstream oFile("bankdb.db", ios::trunc | ios::binary);

    oFile << iNumOfAccounts << "\n";

    for (int i = 0; i < iNumOfAccounts; i++)
    {
        accountInfo tmpAccount = vAccount[i];

        int iNumOfFlow = tmpAccount.vFlow.size();
```

```
        oFile << tmpAccount.strNumber << "\n"
              << tmpAccount.strName << "\n"
              << tmpAccount.strPasswd << "\n"
              << tmpAccount.strAddress << "\n"
              << tmpAccount.iType << "\n"
              << tmpAccount.dInterest << "\n"
              << tmpAccount.bLost << "\n"
              << tmpAccount.tLostTime << "\n"
              << tmpAccount.strOperator << "\n"
              << iNumOfFlow << "\n";

        for (int j = 0; j < iNumOfFlow; j++)
        {
            flowInfo tmpFlow = tmpAccount.vFlow[j];

            oFile << tmpFlow.tTime << "\n"
                  << tmpFlow.dMoney << "\n"
                  << tmpFlow.iOperationType << "\n"
                  << tmpFlow.strOperator << "\n";
        }
    }
}

/**
 * @brief add an account
 *
 * @param [in] strNum number of the account
 * @param [in] strNam name of the account owner
 * @param [in] strPas password of the account
 * @param [in] strAdd address of the account owner
 * @param [in] iType type of the account
 * @param [in] dInt interest of the account
 * @param [in] strOpe operator of the deal
 *
 * requirments of these params @see bankdb.h
 */
void bankdb::AddAccount(string strNum, string strNam, string strPas, string strAdd, int iTyp,
 double dInt, string strOpe)
{
    accountInfo tmpAccountInfo;
    tmpAccountInfo.strNumber = strNum;
    tmpAccountInfo.strName = strNam;
    tmpAccountInfo.strPasswd = strPas;
    tmpAccountInfo.strAddress = strAdd;
    tmpAccountInfo.iType = iTyp;
```

```
    tmpAccountInfo.dInterest = dInt;
    tmpAccountInfo.bLost = false;
    tmpAccountInfo.tLostTime = time(nullptr);
    tmpAccountInfo.strOperator = strOpe;
    vAccount.push_back(tmpAccountInfo);
    vAccount[vAccount.size() - 1].vFlow.clear();

    /**
     * @note do not delete Getfile()!
     *       i do not think it is necessary, but without it the program has bugs
     *       it is to refresh the memory
     */
    WriteFile();
    GetFile();
}

/**
 * @brief edit an account
 *
 * @param [in] strNum number of the account [unchangable]
 * @param [in] strNam name of the account owner
 * @param [in] strAdd address of the account owner
 * @param [in] iType type of the account
 * @param [in] dInt interest of the account
 *
 * requirments of these params @see bankdb.h
 */
void bankdb::EditAccount(string strNum, string strNam, string strAdd, int iTyp, double dInt)
{
    vAccount[mAccount[strNum]].strName = strNam;
    vAccount[mAccount[strNum]].strAddress = strAdd;
    vAccount[mAccount[strNum]].iType = iTyp;
    vAccount[mAccount[strNum]].dInterest = dInt;
    WriteFile();
    GetFile();
}

/**
 * @brief edit password
 *
 * @param [in] strNum number of the account [unchangable]
 * @param [in] strPas password of the account
 *
 * requirments of these params @see bankdb.h
 */
void bankdb::EditPasswd(string strNum, string strPas)
```

```
{
    vAccount[mAccount[strNum]].strPasswd = strPas;
    WriteFile();
}

/**
 * @brief marked as lost
 *
 * @param [in] strNum number of the account [unchangable]
 * @param [in] tTim time of the operation
 *
 * requirments of these params @see bankdb.h
 */
void bankdb::SetLost(string strNum, time_t tTim)
{
    vAccount[mAccount[strNum]].bLost = true;
    vAccount[mAccount[strNum]].tLostTime = tTim;
    WriteFile();
}

/**
 * @brief judge whether the account exists
 *
 * @param [in] strNum number of the account
 *
 * @retval true  finded
 *         false not finded
 */
bool bankdb::ExistAccount(string strNumber)
{
    /**
     * @note a common way to find sth. in a map
     */
    map<string, int>::iterator iter;
    iter = mAccount.find(strNumber);
    return (iter != mAccount.end());
}

/**
 * @brief judge whether the account number matches the password
 *
 * @param [in] strNumber number of the account
 * @param [in] strPasswd password of the account
 *
 * @retval true  match
 *         false not match
```

```
 */
bool bankdb::CheckAccount(string strNumber, string strPasswd)
{
    if (!ExistAccount(strNumber)) /// the account does not exist
        return false;

    return (vAccount[mAccount[strNumber]].strPasswd == strPasswd);
}

/**
 * @brief get name by the account number
 *
 * @param [in] strNum number of the account
 *
 * @return the name @see bandb.h
 */
string bankdb::GetName(string strNumber)
{
    return vAccount[mAccount[strNumber]].strName;
}

/**
 * @brief get name by the index
 *
 * @param [in] iNum index in vAccount
 *
 * @return the name @see bandb.h
 */
string bankdb::GetName(int iNum)
{
    return vAccount[iNum].strName;
}

/**
 * @brief get number of accounts
 *
 * @return the number @see bandb.h
 */
int bankdb::GetNumberOfAccounts()
{
    return vAccount.size();
}

/**
 * @brief get number of flows by the account number
 *
```

```
 * @param [in] strNumber number of the accountt
 *
 * @return the number @see bandb.h
 */
int bankdb::GetNumberOfFlows(string strNumber)
{
    return vAccount[mAccount[strNumber]].vFlow.size();
}

/**
 * @brief get time by the number of the account and index of the flow
 *
 * @param [in] strNumber number of the accountt
 * @param [in] iFlow index in vFlow
 *
 * @return the time @see bandb.h
 */
time_t bankdb::GetTime(string strNumber, int iFlow)
{
    return vAccount[mAccount[strNumber]].vFlow[iFlow].tTime;
}

/**
 * @brief deposit money
 *
 * @param [in] strNumber number of the accountt
 * @param [in] tTime time of the operation
 * @param [in] dMon money to deposit
 * @param [in] strOpe operator
 *
 * format of params @see bandb.h
 */
void bankdb::DepositMoney(string strNumber, time_t tTim, double dMon, string strOpe)
{
    flowInfo tmpFlowInfo;
    tmpFlowInfo.tTime = tTim;
    tmpFlowInfo.dMoney = dMon;
    tmpFlowInfo.iOperationType = 0;
    tmpFlowInfo.strOperator = strOpe;
    vAccount[mAccount[strNumber]].vFlow.push_back(tmpFlowInfo);
    mMoney[strNumber] += dMon;
    WriteFile();
    GetFile();
}

/**
```

```
 * @brief withdraw money
 *
 * @param [in] strNumber number of the accountt
 * @param [in] tTime time of the operation
 * @param [in] dMon money to withdraw
 * @param [in] strOpe operator
 *
 * @retval true  succeed
 *         false money wanted > money left
 *
 * format of params @see bandb.h
 */
bool bankdb::WithdrawMoney(string strNumber, time_t tTim, double dMon, string strOpe)
{
    if (dMon > mMoney[strNumber])
    {
        return false;
    }
    flowInfo tmpFlowInfo;
    tmpFlowInfo.tTime = tTim;
    tmpFlowInfo.dMoney = dMon;
    tmpFlowInfo.iOperationType = 1;
    tmpFlowInfo.strOperator = strOpe;
    vAccount[mAccount[strNumber]].vFlow.push_back(tmpFlowInfo);
    mMoney[strNumber] -= dMon;
    WriteFile();
    GetFile();
    return true;
}
```

bankserver.h

```
/*********************************************************************
 * Copyright (C) 2021 Chenyang https://wcy-dt.github.io               *
 *                                                                    *
 * @file     bankserver.h                                            *
 * @brief    Bankserver is a layer upon bankdb and beneath ui layer. *
 *           it is designed for muti-users. however,                  *
 *           i do not have enough time complete it.                   *
 * @author   Chenyang                                                *
 * @date     2021 - 07                                               *
 *********************************************************************/

#ifndef BANKSERVER_H
#define BANKSERVER_H
```

```cpp
#include "bankdb.h"
#include <string>

using namespace std;

class bankServer
{
public:
    bankServer();

    /**
     * @defgroup account operations
     */
    bool AccountExist(string);
    bool AccountAdd(string, string, string, string, int, double, string);
    bool AccountEdit(string, string, int, double);
    bool PasswdEdit(string, string);
    bool Login(string, string);
    void ReportLost(time_t);

    /**
     * @defgroup deposit and withdraw money
     */
    void Deposit(double, string);
    bool Withdraw(double, string);

    /**
     * @defgroup get information of temporary account
     */
    string GetName();

    /**
     * @defgroup get information by the index in vAccount
     */

string GetName(int);

    /**
     * @defgroup get number
     */
    int GetNumberOfAccounts();
    int GetNumberOfFlows();

    time_t GetTime(int);
```

```
    /**
     * @defgroup get information of the flow by index of the flow
     */
    double GetTotal();
    double GetPrepare(time_t);

    /**
     * @defgroup export the list
     */
    void ExportAccountFile(string);
    void ExportFlowFile(string);

private:
    bankdb db; /// as for we only have one server now, we can define a database here

    string strCurrentNumber; // the account login now
};

#endif // BANKSERVER_H
```

bankserver.cpp

```
/*****************************************************************************
 * Copyright (C) 2021 Chenyang https://wcy-dt.github.io                      *
 *                                                                           *
 * @file     bankserver.cpp                                                  *
 * @brief    Bankserver is a layer upon bankdb and beneath ui layer.         *
 *           it is designed for muti-users. however,                         *
 *           i do not have enough time complete it.                          *
 * @author   Chenyang                                                        *
 * @date     2021 - 07                                                       *
 *****************************************************************************/

#include "bankserver.h"
#include "bankdb.h"
#include "economiccalc.h"
#include <fstream>
#include <QDebug>

/**
 * @brief initialization
 */
bankServer::bankServer()
{
}
```

```
/**
 * @brief judge whether the account exists
 *
 * @param [in] strNum number of the account
 *
 * @retval true  finded
 *         false not finded
 */
bool bankServer::AccountExist(string strNumber)
{
    return (db.ExistAccount(strNumber));
}

/**
 * @brief add an account
 *
 * @param [in] strNum number of the account
 * @param [in] strNam name of the account owner
 * @param [in] strPas password of the account
 * @param [in] strAdd address of the account owner
 * @param [in] iType type of the account
 * @param [in] dInt interest of the account
 * @param [in] strOpe operator of the deal
 *
 * @retval true  input legal
 *         false illegal
 *
 * requirments of these params @see bankdb.h
 */
bool bankServer::AccountAdd(string strNum, string strNam, string strPas, string strAdd, int i
Typ, double dInt, string strOpe)
{
    if (strNam != "" &&
        strPas != "" &&
        strAdd != "" &&
        strOpe != "")
    {
        db.AddAccount(strNum, strNam, strPas, strAdd, iTyp, dInt, strOpe);
        return true;
    }
    return false;
}

/**
 * @brief edit an account
```

```
 *
 * @param [in] strNam name of the account owner
 * @param [in] strAdd address of the account owner
 * @param [in] iType type of the account
 * @param [in] dInt interest of the account
 *
 * @retval true  account exist
 *         false not exist
 *
 * requirments of these params @see bankdb.h
 */
bool bankServer::AccountEdit(string strNam, string strAdd, int iTyp, double dInt)
{
    if (strNam != "" && strAdd != "")
    {
        db.EditAccount(strCurrentNumber, strNam, strAdd, iTyp, dInt);
        return true;
    }
    return false;
}

/**
 * @brief edit password
 *
 * @param [in] strPas password of the account
 *
 * @retval true  account exist
 *         false not exist
 *
 * requirments of these params @see bankdb.h
 */
bool bankServer::PasswdEdit(string strOldPasswd, string strPasswd)
{
    if (!db.CheckAccount(strCurrentNumber, strOldPasswd))
    {
        return false;
    }
    db.EditPasswd(strCurrentNumber, strPasswd);
    return true;
}

/**
 * @brief judge whether the account number matches the password
 *
 * @param [in] strNumber number of the account
 * @param [in] strPasswd password of the account
```

```
 *
 * @retval true  match
 *         false not match
 */
bool bankServer::Login(string strNumber, string strPasswd)
{
    if (!db.CheckAccount(strNumber, strPasswd))
        return false;

    strCurrentNumber = strNumber; /// set the account which has login
    return true;
}

/**
 * @brief marked as lost
 *
 * @param [in] tTim time of the operation
 *
 * requirments of these params @see bankdb.h
 */
void bankServer::ReportLost(time_t tTime)
{
    db.SetLost(strCurrentNumber, tTime);
}

/**
 * @brief deposit money
 *
 * @param [in] dMon money to deposit
 * @param [in] strOpe operator
 *
 * format of params @see bandb.h
 */
void bankServer::Deposit(double dMoney, string strOpe)
{
    time_t tTime = time(NULL);
    db.DepositMoney(strCurrentNumber, tTime, dMoney, strOpe);
}

/**
 * @brief withdraw money
 *
 * @param [in] dMon money to withdraw
 * @param [in] strOpe operator
 *
 * @retval true  succeed
```

```
 *          false money wanted > money left
 *
 * format of params @see bandb.h
 */
bool bankServer::Withdraw(double dMoney, string strOpe)
{
    time_t tTime = time(NULL);
    return db.WithdrawMoney(strCurrentNumber, tTime, dMoney, strOpe);
}

/**
 * @brief get name
 *
 * @return the name @see bandb.h
 */
string bankServer::GetName()
{
    return db.GetName(strCurrentNumber);
}

/**
 * @brief get name by the index
 *
 * @param [in] iNum index in vAccount
 *
 * @return the name @see bandb.h
 */
string bankServer::GetName(int iNum)
{
    return db.GetName(iNum);
}

/**
 * @brief get number of accounts
 *
 * @return the number @see bandb.h
 */
int bankServer::GetNumberOfAccounts()
{
    return db.GetNumberOfAccounts();
}

/**
 * @brief get number of flows by the account number
 *
 * @return the number @see bandb.h
```

```
 */
int bankServer::GetNumberOfFlows()
{
    return db.GetNumberOfFlows(strCurrentNumber);
}

/**
 * @brief get time by the index of the flow
 *
 * @param [in] iFlow index in vFlow
 *
 * @return the time @see bandb.h
 */
time_t bankServer::GetTime(int iFlow)
{
    return db.GetTime(strCurrentNumber, iFlow);
}

/**
 * @brief get total money the account owns
 *
 * @return the money @see bandb.h
 */
double bankServer::GetTotal()
{
    economiccalc Ecocalc;
    Ecocalc.SetType(GetType()); /// boot up a calculate service

    for (int i = 0; i < GetNumberOfFlows(); i++)
    {
        if (GetOperationType(i) == 0) /// deposit
            Ecocalc.Add(GetMoney(i), GetTime(i));
        else /// withdraw
            Ecocalc.Sub(GetMoney(i), GetTime(i));
    }

    return Ecocalc.Query(time(nullptr));
}

/**
 * @brief calculate how much money needed for some day
 *
 * @param [in] iDay date to be calculated
 *
 * @return the money @see bandb.h
 */
```

```
double bankServer::GetPrepare(time_t iDay)
{
    double dAns = 0; /// store the ans

    time_t tTime1 = time(nullptr);
    time_t tTime2 = iDay;

    if (tTime1 >= tTime2) /// illegal time
        return 0;

    for (int k = 0; k < GetNumberOfAccounts(); k++) /// go over all the accounts
    {
        strCurrentNumber = GetNumber(k);

        economiccalc Ecocalc; /// boot up a calculate service

        Ecocalc.SetType(GetType());

        for (int i = 0; i < GetNumberOfFlows(); i++)
        {
            if (GetOperationType(i) == 0) /// deposit
                Ecocalc.Add(GetMoney(i), GetTime(i));
            else /// withdraw
                Ecocalc.Sub(GetMoney(i), GetTime(i));
        }

        dAns += (Ecocalc.QueryPrepare(tTime2));
    }

    return dAns;
}

/**
 * @brief export the account list as html format
 *
 * @param [out] strFileName outfile
 */
void bankServer::ExportAccountFile(string strFileName)
{
    ofstream oFile(strFileName, ios::trunc);

    oFile<<"<html>\n<body>\n<table border='1'>\n<tr>\n<th>账号</th>\n<th>姓名</th>\n<th>地址
</th>\n<th>储种</th>\n"
         <<"<th>利息</th>\n<th>是否挂失</th>\n<th>操作员</th>\n</tr>\n";

    int iNumberOfAccounts = GetNumberOfAccounts();
```

```
for (int i = 0; i < iNumberOfAccounts; i++)
{
    int iTmpType = GetType(i);
    bool bLost = GetLost(i);

    string strTypeString; /// type to output
    string strInterestString; /// interest to output
    switch (iTmpType)
    {
        /// transform the format of type
    case 0:
        strTypeString = "一年期";
        strInterestString = "1.98";
        break;
    case 1:
        strTypeString = "三年期";
        strInterestString = "2.25";
        break;
    case 2:
        strTypeString = "五年期";
        strInterestString = "3.5";
        break;
    }

    string strLostString;
    if (bLost)
    {
        /// transform the format of date
        time_t tTime = GetLostTime(i);

        char szTime[100] = {'\0'};

        tm *pTm = new tm;
        localtime_s(pTm, &tTime);

        pTm->tm_year += 1900;
        pTm->tm_mon += 1;

        sprintf_s(szTime, "%04d 年%02d 月%02d 日 %02d:%02d:%02d",
                  pTm->tm_year,
                  pTm->tm_mon,
                  pTm->tm_mday,
                  pTm->tm_hour,
                  pTm->tm_min,
                  pTm->tm_sec);
```

```
            strLostString = szTime;

            delete pTm;
            pTm = NULL;
        }
        else
        {
            strLostString = "否";
        }

        oFile<<"<tr>\n";
        oFile<<"<td>"<<GetNumber(i)<<"</td>\n";
        oFile<<"<td>"<<GetName(i)<<"</td>\n";
        oFile<<"<td>"<<GetAddress(i)<<"</td>\n";
        oFile<<"<td>"<<strTypeString<<"</td>\n";
        oFile<<"<td>"<<strInterestString<<"</td>\n";
        oFile<<"<td>"<<strLostString<<"</td>\n";
        oFile<<"<td>"<<GetOperator(i)<<"</td>\n";
        oFile<<"</tr>\n";
    }


    oFile<<"</table>\n</body>\n</html>\n";
}

/**
 * @brief export the account list as html format
 *
 * @param [out] strFileName outfile
 */
void bankServer::ExportFlowFile(string strFileName)
{
    ofstream oFile(strFileName, ios::trunc);

    oFile<<"<html>\n<body>\n<table border='1'>\n<tr>\n<th>操作时间</th>\n<th>操作类型
</th>\n<th>金额</th>\n<th>操作员</th>\n</tr>\n";

    int iNumberOfFlows = GetNumberOfFlows();

    for (int i = 0; i < iNumberOfFlows; i++)
    {
        int iTmpType = GetOperationType(i);
        time_t tTime = GetTime(i);

        string strTypeString; /// type to output
```

```
        switch (iTmpType)
        {
            /// transform the format of type
        case 0:
            strTypeString = "存款";
            break;
        case 1:
            strTypeString = "取款";
            break;
        }

        /// transform the format of date
        string strTime;
        char szTime[100] = {'\0'};

        tm *pTm = new tm;
        localtime_s(pTm, &tTime);
        //pTm = localtime(&time_t_time);
        pTm->tm_year += 1900;
        pTm->tm_mon += 1;

        sprintf_s(szTime, "%04d 年%02d 月%02d 日 %02d:%02d:%02d",
                  pTm->tm_year,
                  pTm->tm_mon,
                  pTm->tm_mday,
                  pTm->tm_hour,
                  pTm->tm_min,
                  pTm->tm_sec);

        strTime = szTime;
        //qDebug()<<"here";

        delete pTm;
        pTm = NULL;

        oFile<<"<tr>\n";
        oFile<<"<td>"<<strTime<<"</td>\n";
        oFile<<"<td>"<<strTypeString<<"</td>\n";
        oFile<<"<td>"<<GetMoney(i)<<"</td>\n";
        oFile<<"<td>"<<GetFlowOperator(i)<<"</td>\n";
        oFile<<"</tr>\n";
    }

    oFile<<"</table>\n</body>\n</html>\n";
}
```

economiccalc.h

```
/*************************************************************************
 * Copyright (C) 2021 Chenyang https://wcy-dt.github.io                  *
 *                                                                       *
 * @file     economicclac.cpp                                           *
 * @brief    Economic calc is used as auxiliary for database and server.*
 *           it can calculate the interest.                              *
 * @author   Chenyang                                                   *
 * @date     2021 - 07                                                  *
 *************************************************************************/

#ifndef ECONOMICCALC_H
#define ECONOMICCALC_H

#include "bankdb.h"
#include <time.h>
#include <vector>

class economiccalc
{
public:
    economiccalc();
    void SetType(int);
    void Add(double, time_t);
    void Sub(double, time_t);
    double Query(time_t);
    double QueryPrepare(time_t);

private:
    int iType; /// type

    /**
     * @note the struct of Flow is like a chain.
     *       when we deposit, the chain adds,
     *       when we withdraw, the chain minus.
     */
    struct Flow
    {
        double dMoney; /// money change
                       /// -# +:deposit
                       /// -# -:withdraw
        time_t tTime;  /// time of operation
    };
```

```
    double dMoney; /// money till some point of time
    vector<Flow> vFlow;
};

#endif // ECONOMICCALC_H
```

economiccalc.cpp

```
/*************************************************************************
 * Copyright (C) 2021 Chenyang https://wcy-dt.github.io                  *
 *                                                                       *
 * @file     economicclac.cpp                                           *
 * @brief    Economic calc is used as auxiliary for database and server.*
 *           it can calculate the interest.                              *
 * @author   Chenyang                                                   *
 * @date     2021 - 07                                                  *
 *************************************************************************/

#include "economiccalc.h"
#include <QDebug>

/**
 * @brief initialization
 */
economiccalc::economiccalc()
{
    dMoney = 0;
    vFlow.clear();
}

/**
 * @brief set type
 *
 * @param [in] iSetType type
 *
 * @see bankdb.h
 */
void economiccalc::SetType(int iSetType)
{
    iType = iSetType;
}

/**
 * @brief deposit
 *
```

```
 * @param [in] dMoney money to deposit
 * @param [in] tTime time
 */
void economiccalc::Add(double dMoney, time_t tTime)
{
    Flow tmpFlow;
    tmpFlow.tTime = tTime;
    tmpFlow.dMoney = dMoney;
    vFlow.push_back(tmpFlow);
}

/**
 * @brief withdraw
 *
 * @param [in] dMoney money to withdraw
 * @param [in] tTime time
 */
void economiccalc::Sub(double dMoney, time_t tTime)
{
    double dMoneyNeeded = dMoney;
    int iTmpPos = vFlow.size() - 1; /// pointer to the flow
    while (dMoneyNeeded > 0 && iTmpPos >= 0)
    {
        if (tTime < vFlow[iTmpPos].tTime) /// time no match
        /**
         * @note actually this will not happen.
         *       i write this just to debug.
         */
            iTmpPos--;
        else if (vFlow[iTmpPos].dMoney > dMoneyNeeded) /// enough money
        {
            vFlow[iTmpPos].dMoney -= dMoneyNeeded;
            dMoneyNeeded = 0;
            iTmpPos--;
        }
        else /// money not enough
        {
            dMoneyNeeded -= vFlow[iTmpPos].dMoney;
            vFlow[iTmpPos].dMoney = 0;
            iTmpPos--;
        }
    }
}

/**
 * @brief calculate the interest
```

```
 *
 * @param [in] tTime time
 *
 * @return money in the account till the time
 */
double economiccalc::Query(time_t tTime)
{
    double dAns = 0; /// ans for the query
    int iTmpTime; /// relate to type
    double dInterest; /// store the interest
    switch (iType)
    {
    case 0: /// 1 year
        iTmpTime = 31536000;
        dInterest = 1.98;
        break;
    case 1: /// 3 year
        iTmpTime = 94608000;
        dInterest = 2.25;
        break;
    case 2: /// 5 year
        iTmpTime = 157680000;
        dInterest = 3.5;
        break;
    }

    for (int i = 0; i < (int)vFlow.size(); i++)
    {
        /**
         * @note it can calculate yield compound interest
         */
        int iYieldInterestYear = ((int)difftime(tTime, vFlow[i].tTime)) / iTmpTime;
        double dYieldInterest = vFlow[i].dMoney;
        for (int j = 1; j <= iYieldInterestYear; j++)
            dYieldInterest = dYieldInterest * (1 + (dInterest / 100));
        dAns += dYieldInterest;
    }
    return dAns;
}

/**
 * @brief calculate the money to prepare
 *
 * @param [in] tTime2 time
 *
 * @return money to prepare till the time
```

```
 */
double economiccalc::QueryPrepare(time_t tTime2)
{
    time_t tTime1 = time(nullptr);
    double dAns = 0; /// ans for the query
    int iTmpTime; /// relate to type
    double dInterest; /// store the interest
    switch (iType)
    {
    case 0: /// 1 year
        iTmpTime = 31536000;
        dInterest = 1.98;
        break;
    case 1: /// 3 year
        iTmpTime = 94608000;
        dInterest = 2.25;
        break;
    case 2: /// 5 year
        iTmpTime = 157680000;
        dInterest = 3.5;
        break;
    }

    for (int i = 0; i < (int)vFlow.size(); i++)
    {
        int iDifftime1 = (int)difftime(tTime1, vFlow[i].tTime);
        int iDifftime2 = (int)difftime(tTime2, vFlow[i].tTime);
        if (((iDifftime1 / iTmpTime) + 1) * iTmpTime <= iDifftime2)
        {
            /**
             * @note the judgement above means the money matures
             */
            int iYieldInterestYear = iDifftime2 / iTmpTime;
            double dYieldInterest = vFlow[i].dMoney;
            for (int j = 1; j <= iYieldInterestYear; j++)
                dYieldInterest = dYieldInterest * (1 + (dInterest / 100));
            dAns += dYieldInterest;
        }
    }
    return dAns;
}
```

测试报告

1. 测试结论

软件实现需求大纲所有功能，无或有少量 bug，达到发布标准。

2. 风险

软件未对密码和信息进行加密处理，存在较高泄露风险。

3. 测试时间

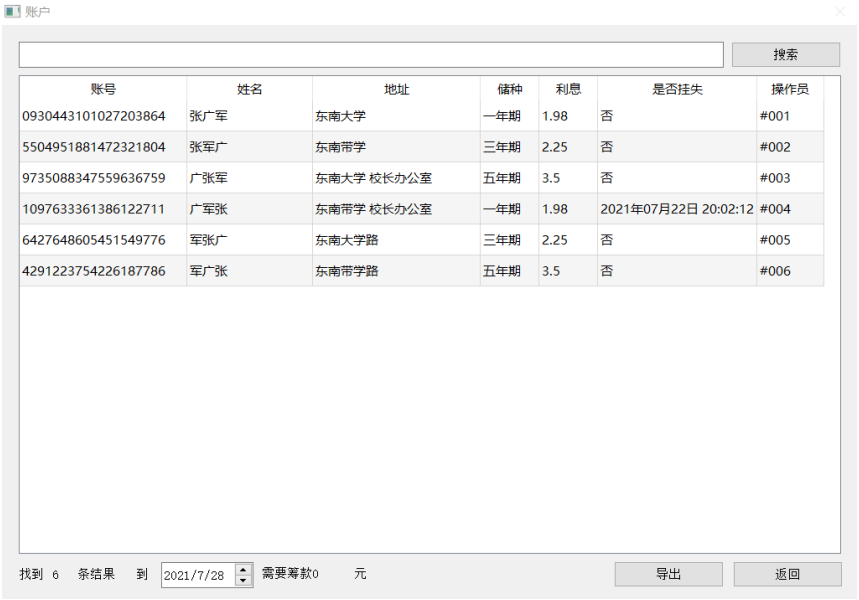
2021 年 7 月 27 日

4. 测试环境

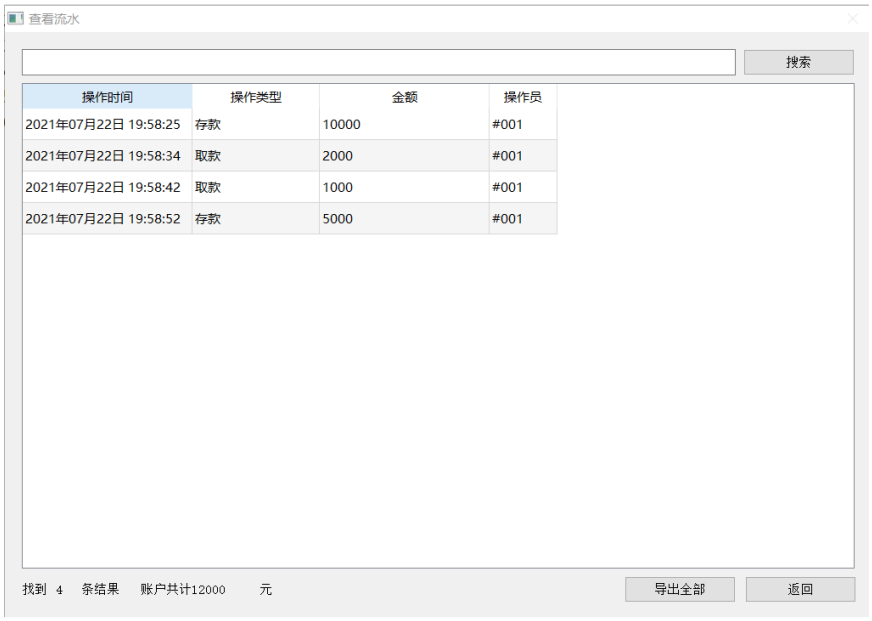
Windows10-21H1

5. 测试流程

账户查看 & 搜索 & 资金准备



流水查看 & 利息计算

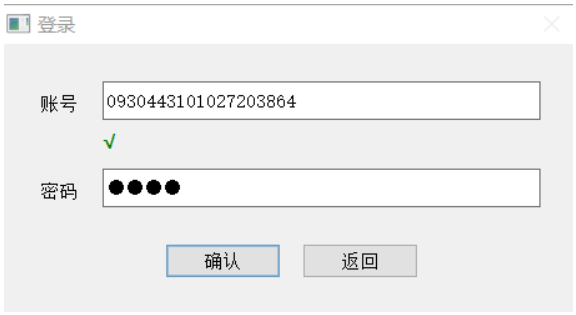


测试数据存储在 bankdb.db 文件中，随软件一起打包。
初始账户和密码如下：

账户	密码
0930443101027203864	1111
5504951881472321804	2222
9735088347559636759	3333
1097633361386122711	4444
6427648605451549776	5555
4291223754226187786	6666

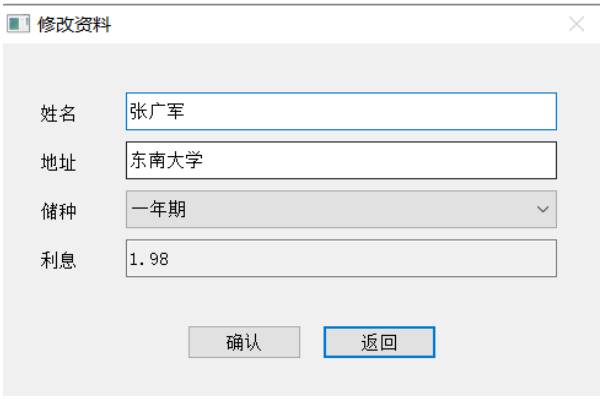
创建账户

登陆账户



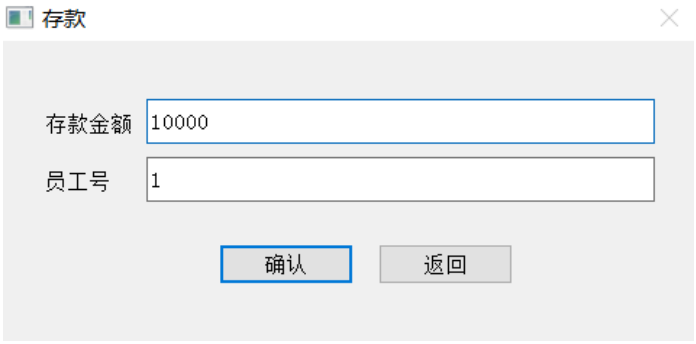
操作选择

修改资料



修改密码

存取款



6. 测试总结

该版本完成所有需求，未发现 bug，但仍应改进数据安全性。

暑期学校学习小结

转专业后补的这门课，一个暑校要学五门课。于是找来找去找了四天空闲时间，前前后后总共写了 3000 多行（时间有限，部分冗余代码没来得及删除）。

类型 ▲	文件 ▲	行数 ▼	注释 ▲	空行 ▲	平均行数 ▲	最大行数 ▲	最小行数 ▲
.cpp	14	2,682	769	306	191	616	66
.h	13	689	206	159	53	128	38
总计	27	3,371	975	465	124	616	38

当然，这 3000 多行没有一行是复制粘贴来的。

平时后贡献别人的项目就会发现，前期规划非常重要，规划不好，后面迟早变成屎山。因此，这个大作业开始前我也进行了详细的规划，画了一个巨大无比的饼，最后发现时间不允许我实现这个大饼。

qt 很早之前学的了，这次也趁机复习了一下。

最后吐槽一下规定的代码格式。前缀表示变量类型的命名方法不是为 C++ 发明的，也根本不适用于 C++ 这样高度灵活且复杂的语言。建议今后开这门课时，能接轨 Google 目前的命名规范。