学号 专业 信息安全 姓名

实验日期 **2024/05/27** 教师签字 成绩

实验报告

【实验名称】 面向对象程序设计理论课程大作业

【题目】

1. 实现课本 8.6 综合实例——对个人银行账户管理程序的改进
2. 实现一个矩阵类，其声明如下：

#include "stdafx.h"

#include <iostream>

using namespace std;

class Matrix;

// 矩阵类

class MatrixIndex;

// 子矩阵类

// 矩阵类

class Matrix{

private:

double \*data;

// 核心数据,用一维数组存储二维矩阵

int nRow;

// 矩阵行数

int nColumn;

// 矩阵列数

public:

// 构造一个x行y列的矩阵,元素值全零

Matrix(int x, int y);

// 构造一个x行y列的矩阵,当type为"zero"时元素值全零,

// 为"one"时元素值全一,为"rand"时元素值全随机

Matrix(int x, int y, const char\* type);

// 复制构造函数

Matrix(const Matrix&m1);

// 用临时数组构造矩阵,例如当输入为{1,2,3}时构造一个

// 1\*3的矩阵

Matrix(std::initializer\_list<int> vector);

// 用子矩阵构造一个矩阵,子矩阵仅保存某矩阵的引用,此

// 处将子矩阵引用的数据进行深拷贝以构造一个全新矩阵,

// 这使得子矩阵可以作为右值同矩阵一样使用

Matrix(const MatrixIndex& index);

// 销毁数组data

~Matrix();

// 用矩阵构造一个子矩阵,子矩阵仅保存某矩阵的引用,例

// 如m1={1,2}且m2={3,4},则m(m1,m2)构造以m的第1,2行和

// 第3,4列构造的子矩阵,子矩阵仅保存源矩阵的引用

MatrixIndex operator()(const Matrix&m1, const Matrix&m2);

// 重载等号运算符,实现m=m1

Matrix& operator=(const Matrix &m1);

// 重载等号运算符,将m的所有元素更改为a

Matrix& operator=(double a);

// 重载等号运算符,将m更改为行向量vector,此时m中的元

// 素数目需要和vector中的元素数目相同Matrix& operator=(std::initializer\_list<double> vector);

// 重载加号运算符,实现m1+m2,矩阵大小需满足运算规则

friend Matrix operator+(const Matrix &m1, const Matrix &m2);

// 重载减号运算符,实现m1-m2,矩阵大小需满足运算规则

friend Matrix operator-(const Matrix &m1, const Matrix &m2);

// 重载乘号运算符,实现m1\*m2,矩阵大小需满足运算规则

friend Matrix operator\*(const Matrix &m1, const Matrix &m2);

// 重载<<运算符,实现cout<<m1打印矩阵内容

friend ostream& operator<<(ostream &out, const Matrix &m1);

friend class MatrixIndex;

};

// 子矩阵类,其中每个元素为指向某矩阵元素的指针

class MatrixIndex{

private:

const Matrix& source;

// 子矩阵对应的源矩阵

const Matrix& rows;

// 子矩阵对应源矩阵的行

const Matrix& columns;

// 子矩阵对应源矩阵的列

public:

// 用矩阵构造一个子矩阵,子矩阵仅保存矩阵source的

// 第rows行和第columns列的引用

MatrixIndex(const Matrix&source, const Matrix&rows, const Matrix&

columns) :source(source), rows(rows), columns(columns){}

// 重载等号运算符,实现m(...)=m1,此时左右元素数目

// 需相同,这使得子矩阵可以作为左值同矩阵一样使用

MatrixIndex& operator=(const Matrix &m1);

// 重载等号运算符,将m(...)的所有元素更改为a

MatrixIndex& operator=(double a);

// 重载等号运算符,将m(...)更改为行向量vector,此

// 时左右元素数目需相同

MatrixIndex& operator=(std::initializer\_list<double> vector);

friend class Matrix;

};

// 主函数

int \_tmain(int argc, \_TCHAR\* argv[])

{

// 生成元素为10,9,8,7,6,5的1\*6矩阵m

Matrix m({ 10, 9, 8, 7, 6, 5 });

// 生成元素为4,3,2,1的1\*4矩阵p

Matrix p({ 4, 3, 2, 1 });

// 生成元素全为零的3\*4矩阵k

Matrix k(3, 4);

// 将矩阵k的第1行第0,1,2列的数据改为// 矩阵m的第0行第0,1,2列元素与矩阵m

// 的第0行第3,4,5列元素之差

k({ 1 }, { 0, 1, 2 }) = m({ 0 }, { 0, 1, 2 }) - m({ 0 }, { 3, 4, 5 });

// 将矩阵k的第2行第x列的数据改为矩阵

// m的第0行第0,1,2列元素与矩阵m的第

// 0行第3,4,5列元素之和,其中x为矩阵

// p的第0行第1,2,3列的所有元素

k({ 2 }, p({ 0 }, { 1, 2, 3 })) = m({ 0 }, { 0, 1, 2 }) + m({ 0 }, { 3, 4, 5 });

cout << k;

system("pause");

return 0;

}

【实验原理】

题目一：

将题目分为以下六个文件进行编写

**date.h:**

在该类中声明了Date 类，该类封装了一个日期，并提供了各种方法和操作符来操作和比较日期。同时重载了加减运算以及输入输出运算符，方便对时间进行加减与输入输出操控。

**accumulator.h:**

在该文件中，定义了一个名为 Accumulator 的类，用于按日累加某个数值。Accumulator 类的主要作用是跟踪某个数值在不同日期间的累加和变更情况，并提供相关操作和查询方法。

类成员变量：

Date lastDate: 保存上次变更数值的日期。

double value: 保存数值的当前值。

double sum: 保存数值按日累加之和。

类成员函数：

构造函数：

作用：初始化一个 Accumulator 对象。

参数：date：累加开始的日期。value：累加的初始值。

操作：设置 lastDate 为 date，设置 value 为初始值，sum 初始化为 0。

getSum函数：

作用：获取到指定日期 date 为止的累加结果。

参数：date - 要查询累加结果的日期。

返回值：当前累加值加上从 lastDate 到 date 这段时间内的累加值。

实现：累加和是 sum 加上 value 在从 lastDate 到 date 期间的累加部分。

change函数：

作用：在指定日期 date 将累加器中的数值变更为新的值 value。

参数：date：变更数值的日期。value：新的数值。

操作：更新 sum 为当前日期的累加结果。更新 lastDate 为新的日期 date。更新 value 为新的数值。

reset函数：

作用：重新初始化累加器。

参数：date：新的起始日期。value：新的数值。

操作：将 lastDate 设置为新的日期 date。将 value 设置为新的数值。将 sum 归零。

**account.h:**

该头文件负责实现定义一个银行账户管理系统的基本功能，包括各种账户类型、账目记录、累加器、异常处理等相关类和函数。以下是各个类的原理。

**AccountRecord 类**

* **成员变量**：
  + Date date：记录日期。
  + const Account\* account：记录所属账户。
* double amount：记录金额。
  + double balance：记录余额。
  + std::string desc：记录描述。
* **成员函数**：
  + AccountRecord(const Date& date, const Account\* account, double amount, double balance, const std::string& desc)：构造函数，初始化账目记录。
  + void show() const：输出账目记录信息。

**Account 类**

* **成员变量**：
  + std::string id：账号。
  + double balance：余额。
  + static double total：所有账户的总金额。
  + static RecordMap recordMap：账目记录。
* **成员函数**：
  + void record(const Date& date, double amount, const std::string& desc)：记录一笔账。
  + void error(const std::string& msg) const：报告错误信息。
  + virtual void deposit(const Date& date, double amount, const std::string& desc) = 0：存入现金（纯虚函数）。
  + virtual void withdraw(const Date& date, double amount, const std::string& desc) = 0：取出现金（纯虚函数）。
  + virtual void settle(const Date& date) = 0：结算（纯虚函数）。
  + virtual void show(std::ostream& out) const：显示账户信息。
  + static void query(const Date& begin, const Date& end)：查询指定时间内的账目记录。

**SavingsAccount 类**

* **成员变量**：
  + Accumulator acc：辅助计算利息的累加器。
  + double rate：存款的年利率。
* **成员函数**：
  + SavingsAccount(const Date &date, const std::string &id, double rate)：构造函数。
  + void deposit(const Date &date, double amount, const std::string &desc)：存入现金。
  + void withdraw(const Date& date, double amount, const std::string& desc)：取出现金。
  + void settle(const Date &date)：结算利息。

**CreditAccount 类**

* **成员变量**：
  + Accumulator acc：辅助计算利息的累加器。
  + double credit：信用额度。
  + double rate：欠款的日利率。
  + double fee：信用卡年费。
* **成员函数**：
  + CreditAccount(const Date& date, const std::string& id, double credit, double rate, double fee)：构造函数。
  + void deposit(const Date& date, double amount, const std::string& desc)：存入现金。
  + void withdraw(const Date& date, double amount, const std::string& desc)：取出现金。
  + void settle(const Date& date)：结算利息和年费。
  + void show(std::ostream& out) const：显示账户信息。

**AccountException 类**

* **成员变量**：
  + const Account\* account：抛出异常的账户。
* **构造函数**：
  + AccountException(const Account \*account, const std::string msg)：构造函数。
* **成员函数**：
  + const Account\* getAccount() const：获取抛出异常的账户。

**date.cpp:**

该cpp文件实现了date.h中声明的getMaxDay函数与构造函数，并实现了日期类型的输入输出运算符重载。

**account.cpp:**

该 cpp 文件实现了账户管理系统的核心功能实现了账户的增删改查操作，通过记录每笔交易实现账户余额的更新和计算。支持储蓄账户和信用账户的不同操作，如计算利息、处理透支和信用额度等。通过异常处理（AccountException）处理错误情况，确保账户操作的安全性和稳定性。

**AccountRecord 类**

* **作用**：记录账目的具体信息，包括日期、账户、金额变动、余额和描述。
* **成员函数**：
  + - AccountRecord(const Date &date, const Account \*account, double amount, double balance, const std::string &desc)：构造函数，初始化账目记录。
    - void show() const：输出账目记录的详细信息。

**Account 类**

* **作用**：定义了账户的基本结构和操作，是一个抽象类，提供了接口供派生类实现具体的存取款和结算操作。
* **成员变量**：
  + - std::string id：账号。
    - double balance：当前余额。
    - static double total：所有账户的总余额。
* **成员函数**
  + - Account(const Date& date, const std::string& id)：构造函数，初始化账户信息。
    - void record(const Date& date, double amount, const std::string& desc)：记录一笔账目。
    - void show(std::ostream& out) const：显示账户信息。
    - void error(const std::string& msg) const：报告错误信息。
    - void query(const Date& begin, const Date& end)：查询指定时间范围内的账目记录（静态函数）。

**SavingsAccount 类**

* **作用**：派生自 Account 类，实现了储蓄账户的具体操作，包括存款、取款和结算利息。
* **成员变量**：
  + Accumulator acc：用于计算利息的累加器。
  + double rate：存款的年利率。
* **成员函数**：
  + SavingsAccount(const Date& date, const std::string& id, double rate)：构造函数，初始化储蓄账户。
  + void deposit(const Date& date, double amount, const std::string& desc)：存款操作。
  + void withdraw(const Date& date, double amount, const std::string& desc)：取款操作。
  + void settle(const Date& date)：结算利息。

**CreditAccount 类**

* **作用**：派生自 Account 类，实现了信用账户的具体操作，包括存款、取款、结算利息和年费。
* **成员变量**：
  + - Accumulator acc：用于计算利息的累加器。
    - double credit：信用额度。
    - double rate：欠款的日利率。
    - double fee：信用卡年费。
* **成员函数**：
  + - CreditAccount(const Date& date, const std::string& id, double credit, double rate, double fee)：构造函数，初始化信用账户。
    - void deposit(const Date& date, double amount, const std::string& desc)：存款操作。
    - void withdraw(const Date& date, double amount, const std::string& desc)：取款操作。
    - void settle(const Date& date)：结算利息和年费。
    - void show(std::ostream& out) const：显示账户信息。

**main.cpp:**

该文件是主程序入口，根据用户在命令行的输入执行对应操作。

switch (cmd)：根据不同的命令执行相应操作：

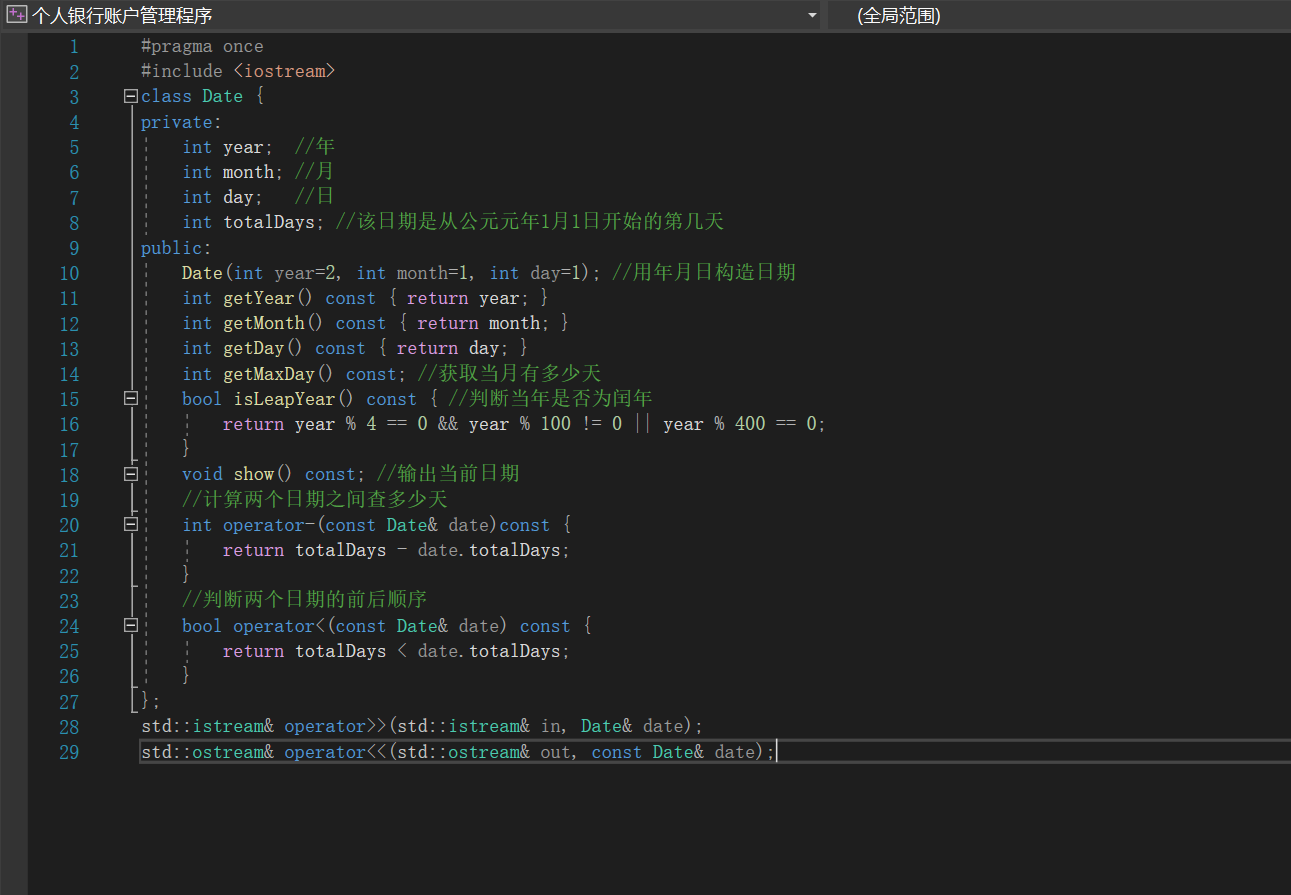
* 'a'：添加账户，根据账户类型创建储蓄账户或信用账户并加入账户列表。
* 'd'：存款，读取账户索引、金额和描述，并执行存款操作。
* 'w'：取款，读取账户索引、金额和描述，并执行取款操作。
* 's'：显示所有账户信息。
* 'c'：更改当前日期。
* 'n'：切换到下一个月，并结算所有账户。
* 'q'：查询指定日期范围内的交易记录。
* 'e'：结束程序。

该文件会创建一个名为command.txt的文本文件，保存每一次的操作，在下一次运行主程序时读取并执行这些操作，达到保存用户数据的作用。

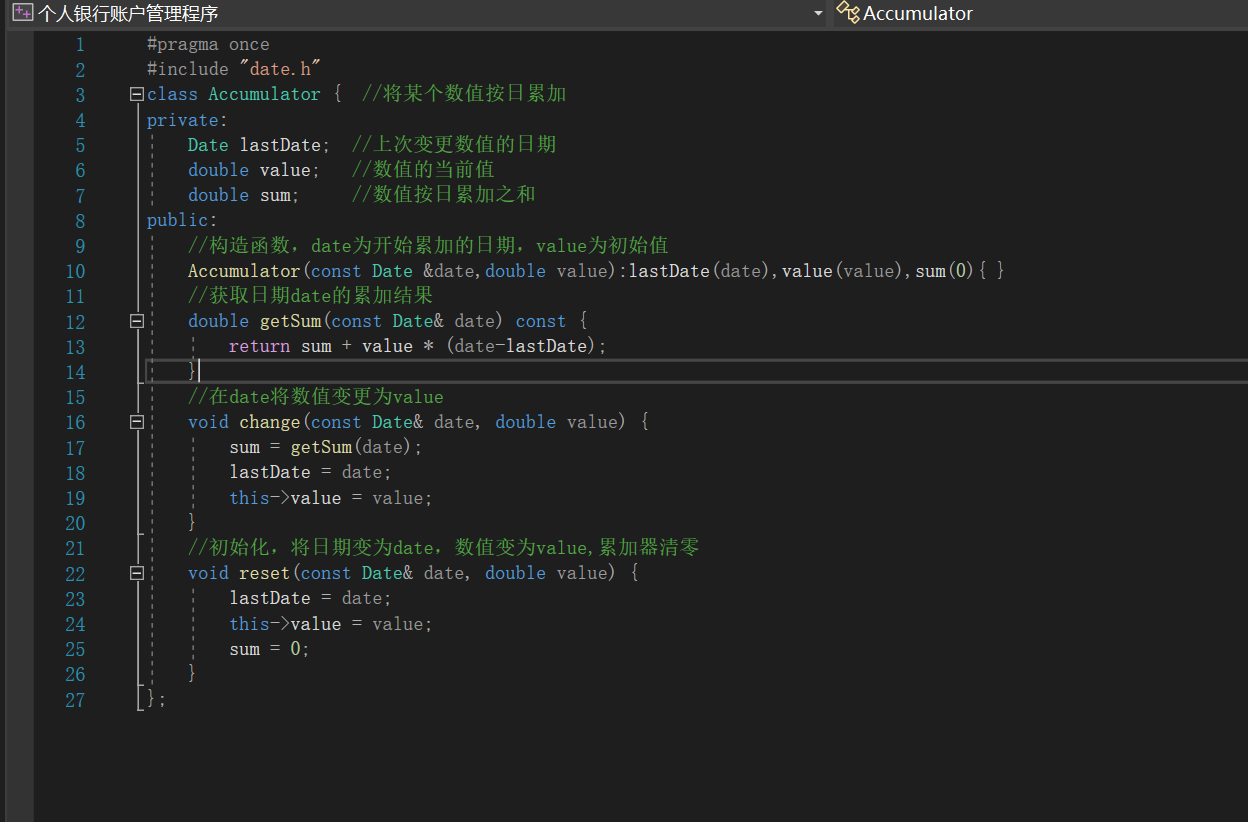
【源码】

题目一

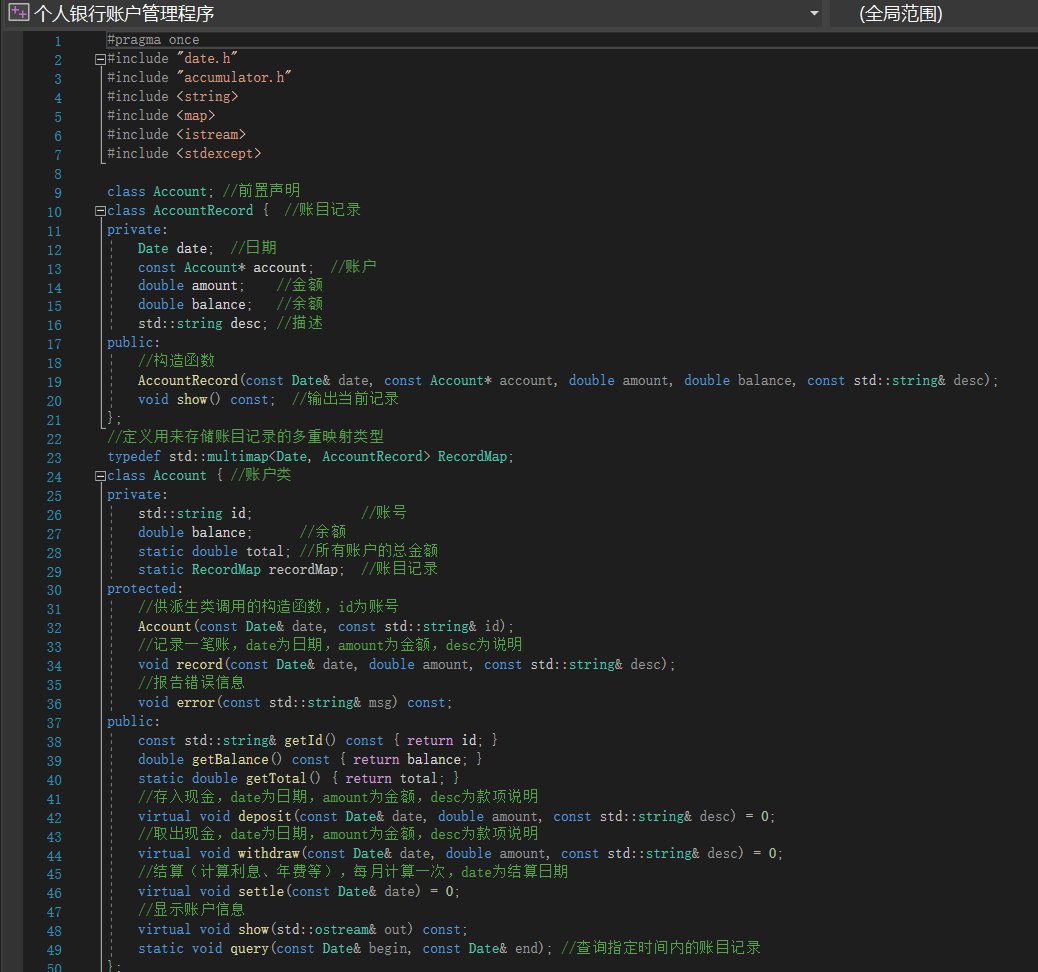
date.h:

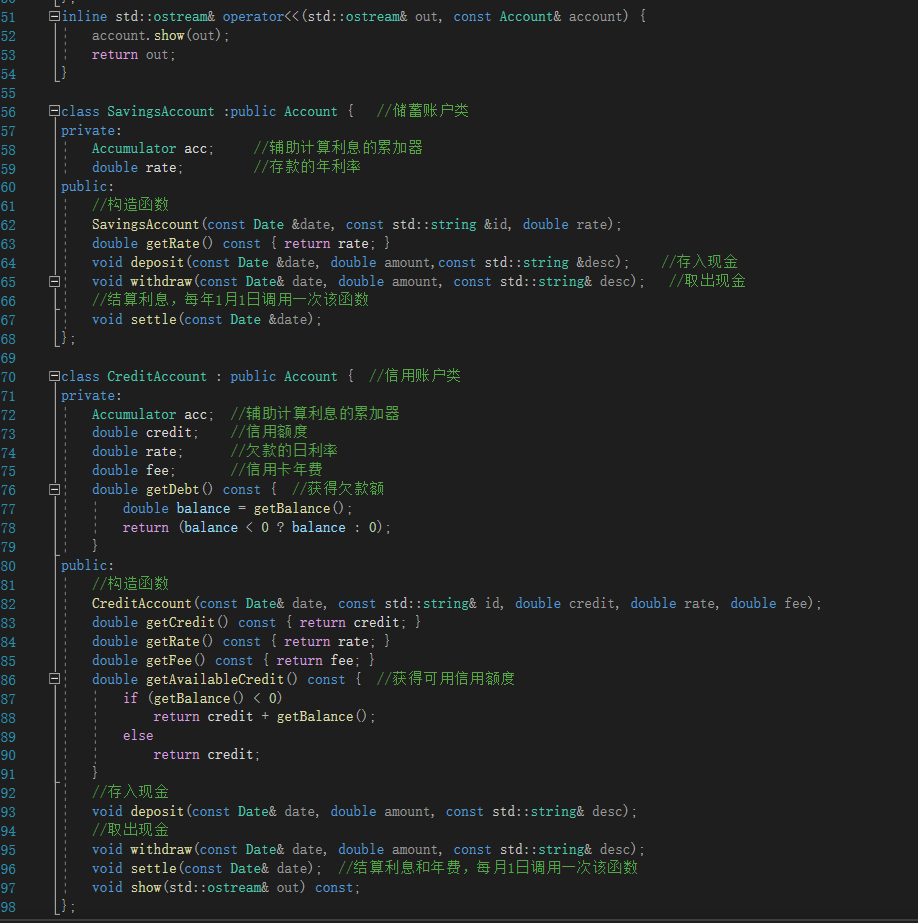


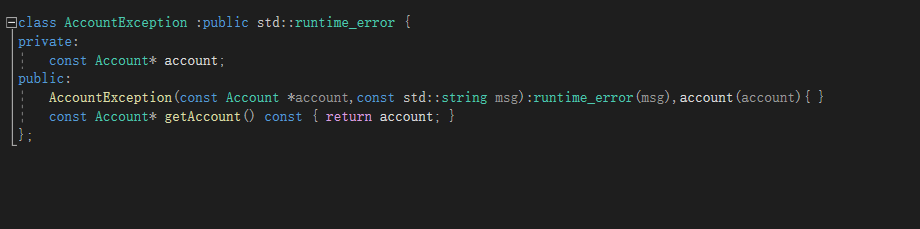
accumulator.h:



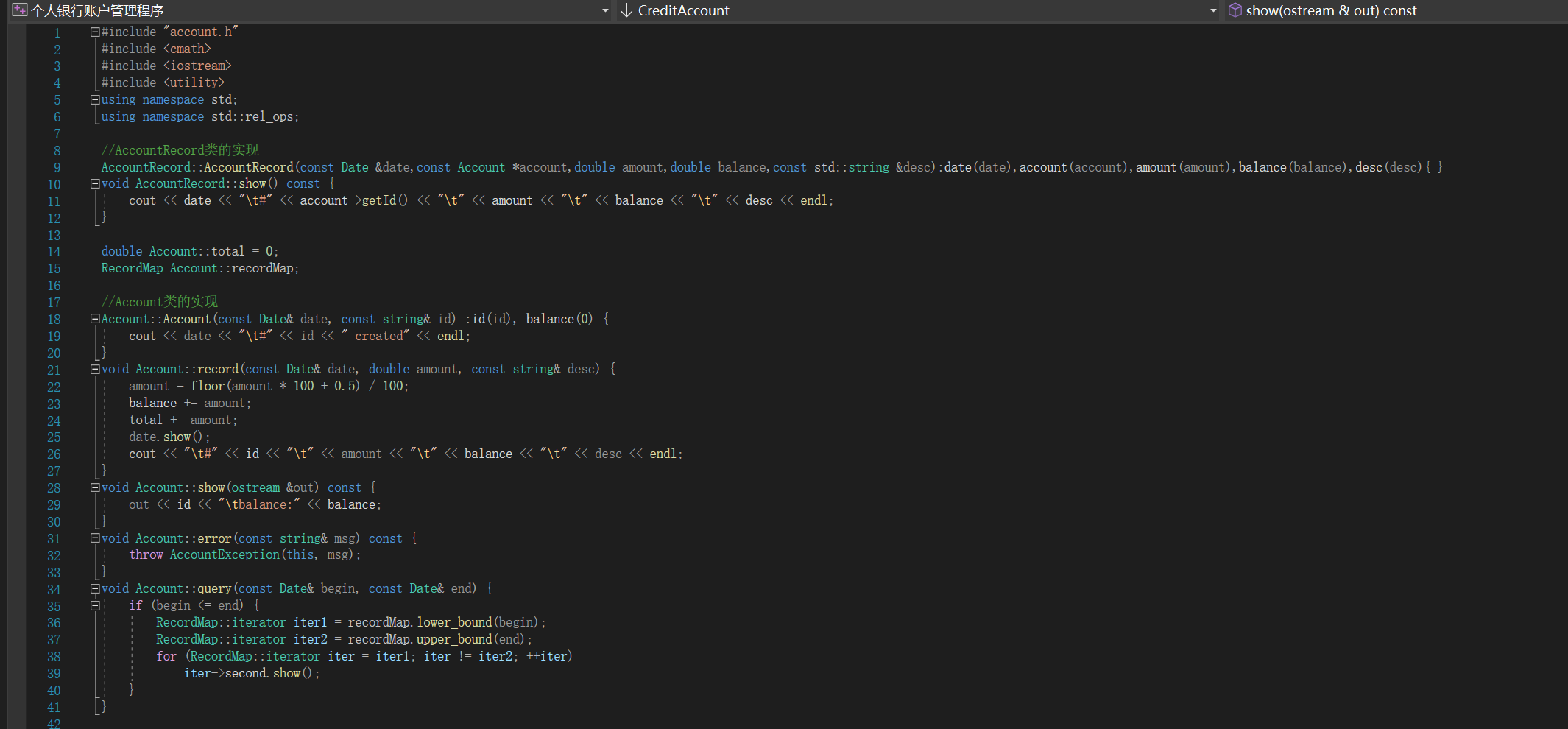
account.h

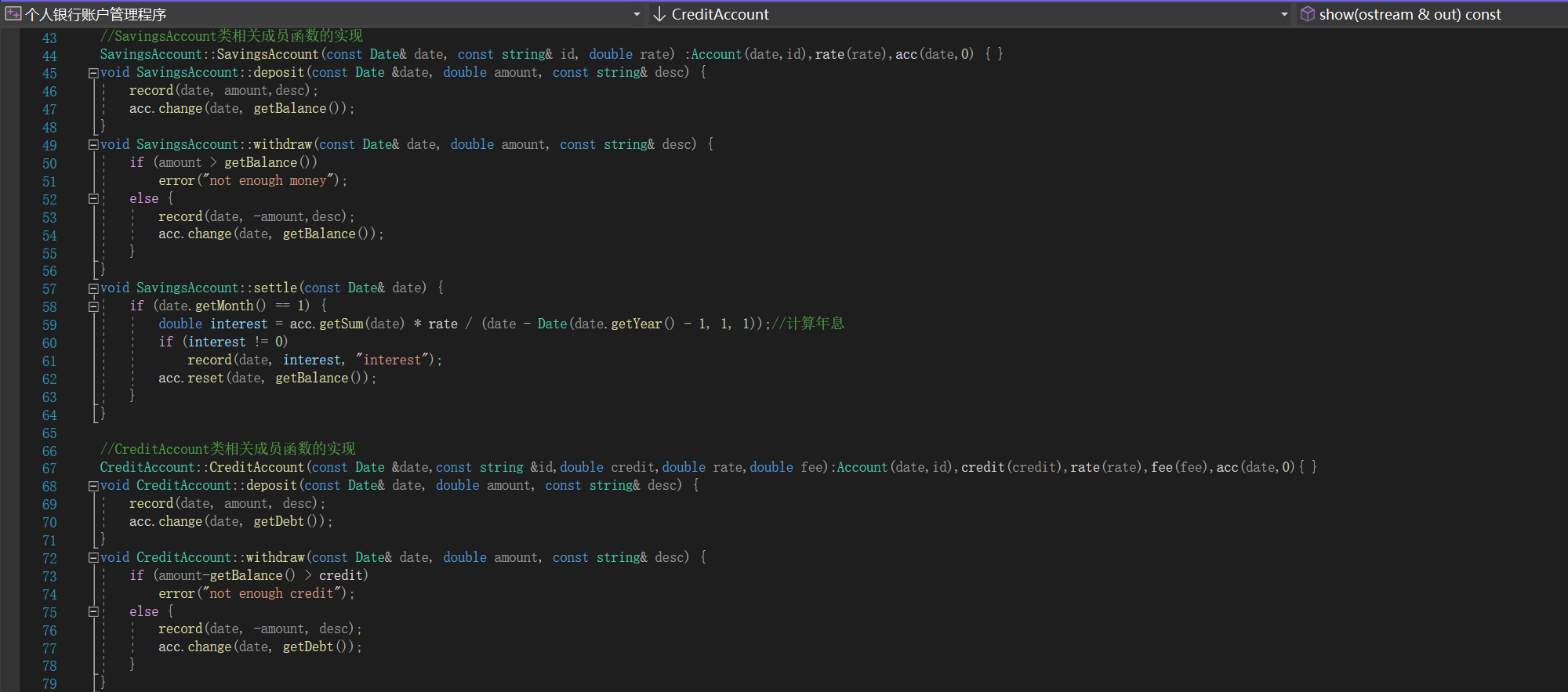


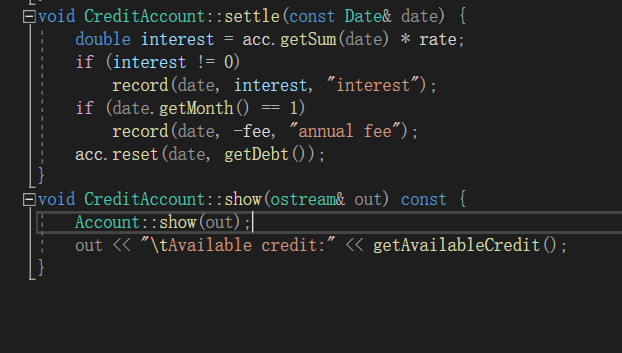




account.cpp

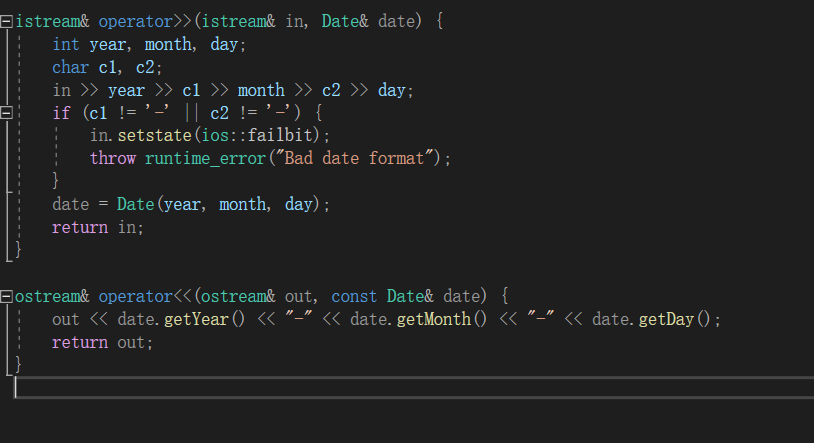




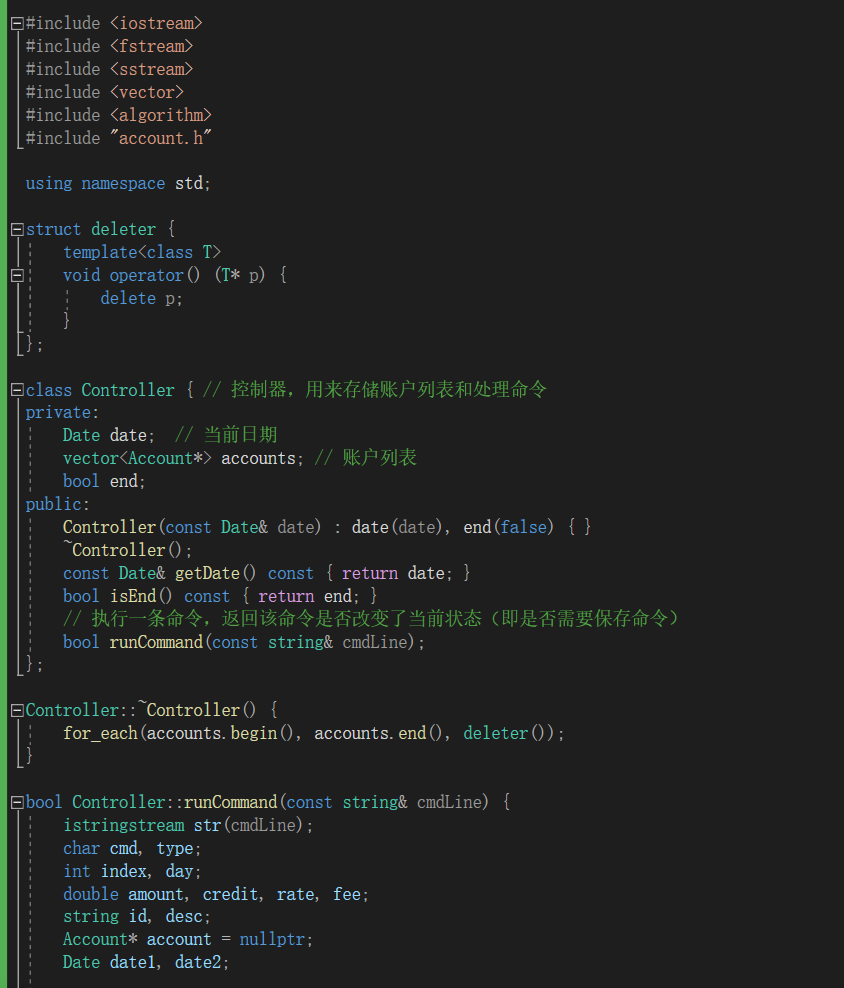


date.cpp

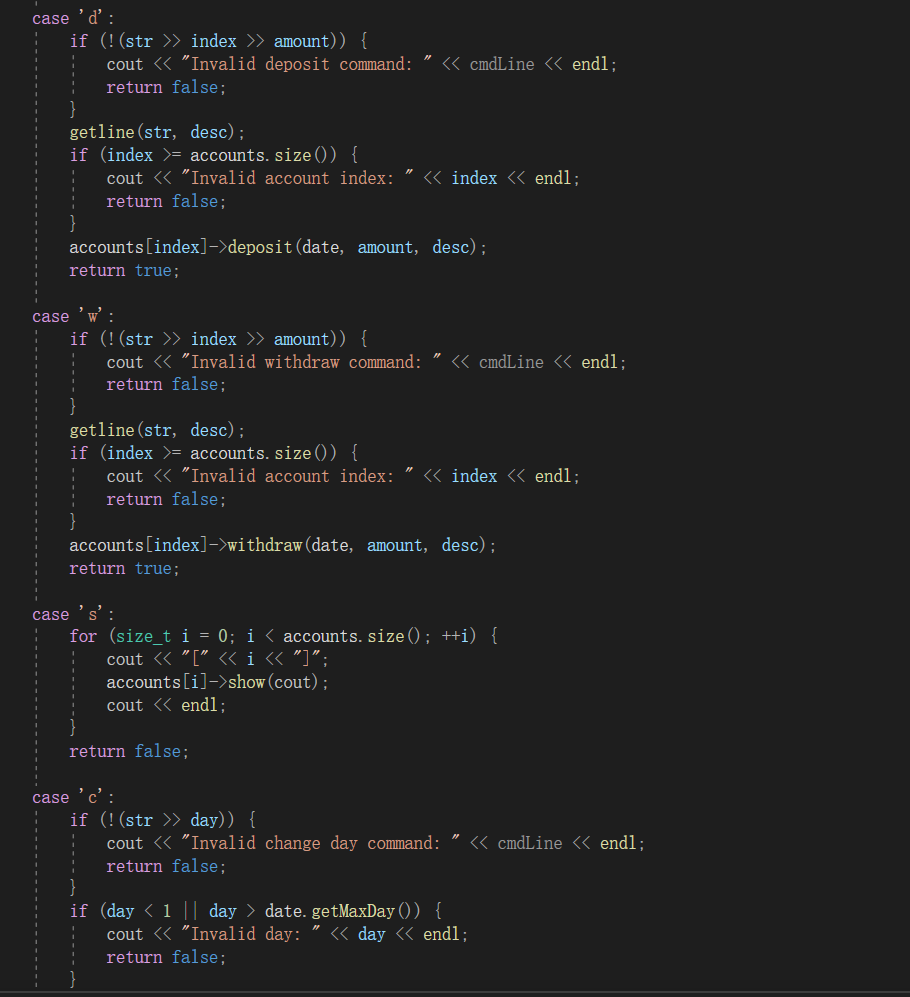


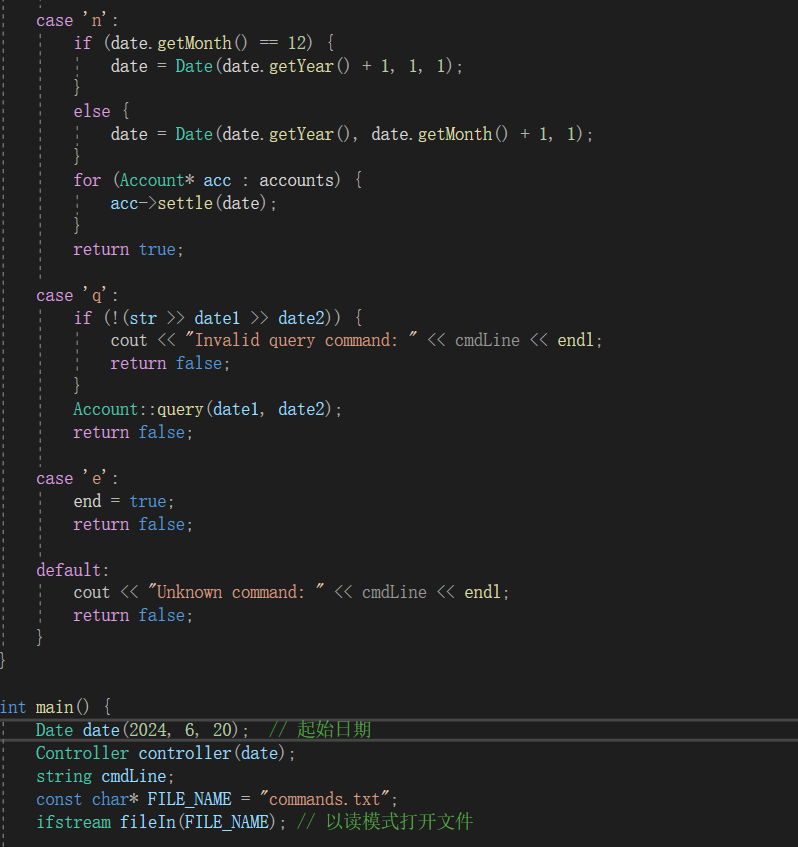


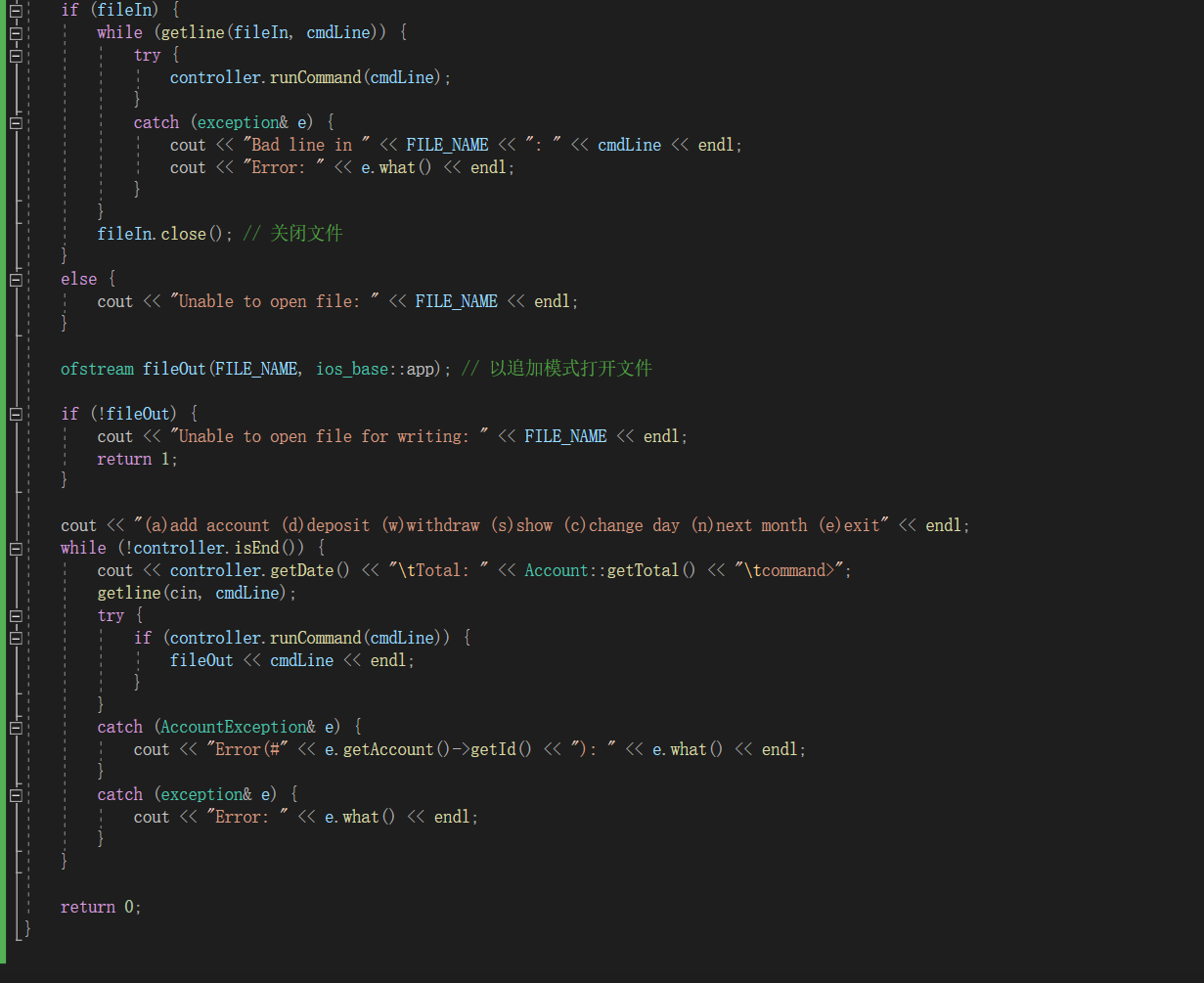
main.cpp



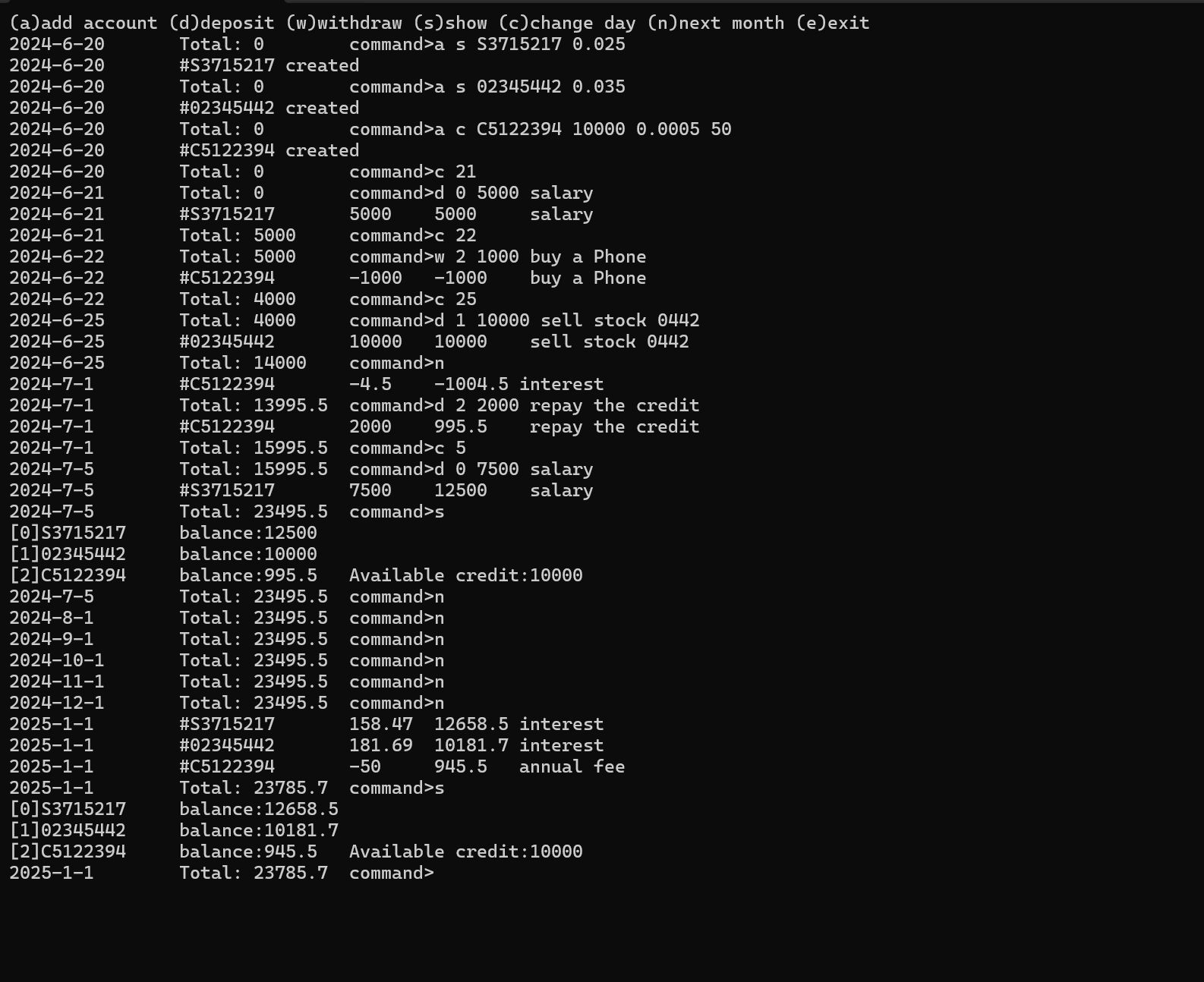




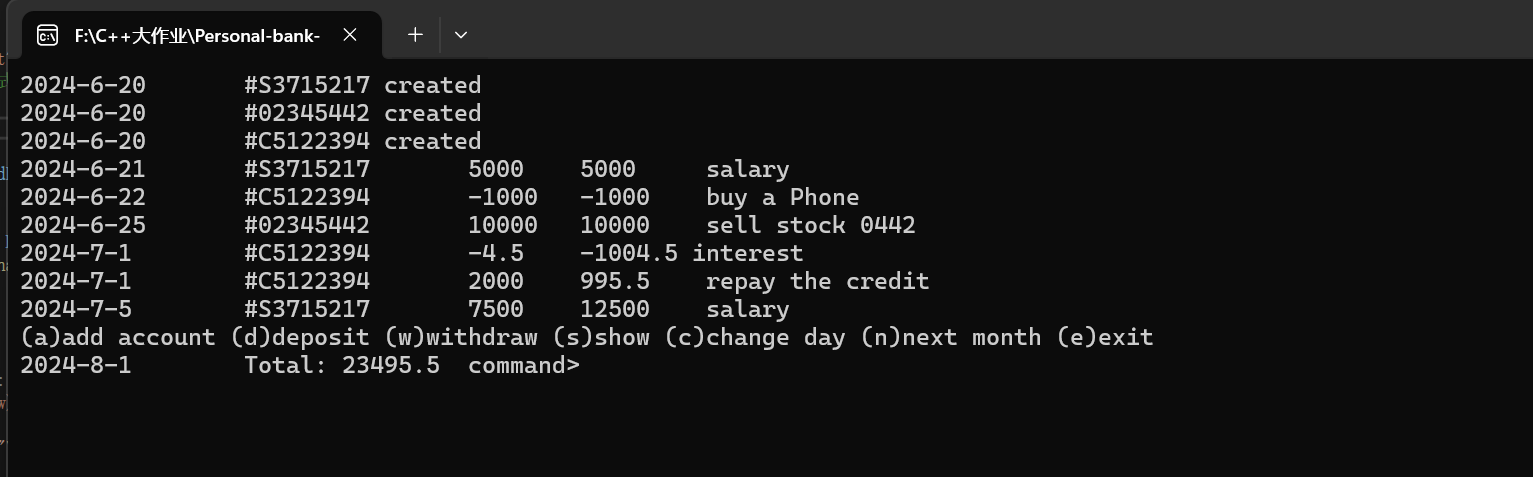




运行截图：



测试command.txt保存：



题目二：

代码：

总结：