**程序设计实验报告**

**家庭财务管理系统**

**目录**

1. **题目……………………………………………3**
2. **前言……………………………………………5**
3. **系统设计思路………………………………6**
4. **UML类图……………………………………7**
5. **系统的设计过程……………………………9**
6. **程序源代码…………………………………15**
7. **系统测试情况……………………………28**
8. **系统的优点与改进………………………32**
9. **感想…………………………………………33**
10. **题目:家庭财务管理**

**目的：**收支管理是一个家庭日常生活中不可缺少一部分，也是财务管理的重要组成部分。开发一个家庭财务管理系统，实现家庭财务信息的自动化。

**系统主要功能：**

1.登陆管理 ：可以选一名家庭成员做管理员，普通用户只能查看自己的收支信息；

2. 家庭成员管理：成员的信息：姓名、性别、出身日期、年龄，身高，体重；

3. 收支项目管理

可以设置收支项目，修改或者删除。收支项目要求分为两层，比如“车辆维护-保养费”；任意一层次都可作为收支项目记账，用户在不清楚支出是否属于“保养费”的时候，可以选择大类“车辆维护”作为支出项；

4.家庭成员收支输出

显示每个成员每月的收支情况(时间、收支种类、金额、经手人)；

5.退出管理。

家庭财务管理系统

登陆管理

家庭成员管理

增加家庭成员

修改家庭成员

删除家庭成员

收支项目管理

查看收支项目

编辑收支项目

删除收支项目

家庭成员收支打印

退出管理

系统功能图

1. **前言**

本次设计的项目是家庭财务管理系统，实现的是简单的家庭财务管理功能。设计是面向对象编程，使用的语言为C++。编译环境是mingw-w64 8.1.0，编译器为编译环境内置的g++，版本8.1.0，编译器开启C++2a标准支持，开启发布版本O3优化。IDE是Qt5.15.0，并使用其中的Plain C++ Application项目。本项目的界面是DOS界面。系统支持在题目系统功能图中标明的全部功能。整个项目总计代码行数为2193，全部代码均由？？？一人编写完成。

1. **系统设计思路**

本系统设计满足题目要求。根据题目要求“可以选一名家庭成员做管理员，普通用户只能查看自己的收支信息”，本系统设计成有且仅有一名系统管理员。

本系统设计遵循分工明确的原则，每个类都有明确的分工。根据系统功能表中的表项精简提炼后，决定制作9个类，其中有3个强枚举类型类，有3个小类，3个大类。

3个强枚举类型类分别用来表示用户权限，性别和当前用户所处界面状态。

3个小类分别用来表示一级二级收支项目，成员个人信息和成员收支信息。

3个大类分别掌管成员数据存储，成员数据管理和成员数据显示的功能。

项目设计基本遵循MVC泛型，成员数据存储大类属于Model模型，成员数据管理大类属于Control控制，成员数据显示大类属于View显示。所有类的具体内容都在“系统的设计过程”部分，这里只简单介绍一下大类的主要设计思路。

在成员数据存储大类中存储个人信息，成员收支信息并提供获取和修改的接口，因为每个人都有各自的个人信息和收支信息，所以设计成普通类。

在成员数据管理大类中存放所有家庭成员，一级二级收支项目，并提供相关的管理函数接口以便进行管理，将所有家庭成员数据和其收支信息以及一级二级收支项目输出到文件和相关信息从文件的读入和初始化都是在此类内完成，因为数据管理是统一的，应该达到简单的目的，不用生成相应的对象就能使用其中提供的接口，故设计为静态类。

在成员数据显示大类主要负责UI界面的显示，以及读取用户从键盘中输入的操作而去调用数据成员管理大类的方法来执行相关的操作，权限检查也是在此类中完成，当用户选择了某个管理员所属操作时需要先检查其是否是管理员然后才执行成员数据管理大类中的相关的函数，因为UI界面的显示也不应该需要对象才能进行显示，为了简单易用起见也设计为静态类，可以直接调用其中的函数。

具体UI部分根据功能划分为成员管理，密码服务和收支管理三大块。

成员管理部分设置有基本的增删改查和管理员权限转让的选项。

密码服务部分有查看密码和修改密码两个选项。

收支管理部分有一二级收支项目的增删改查和成员收支信息的增删改查。

三个小类和三个强枚举类在三个大类中发挥各自的作用，而三个大类又通过函数调用和组合的方式进行相互影响，相互作用而实现相关的功能，从而构成整个完整的系统。

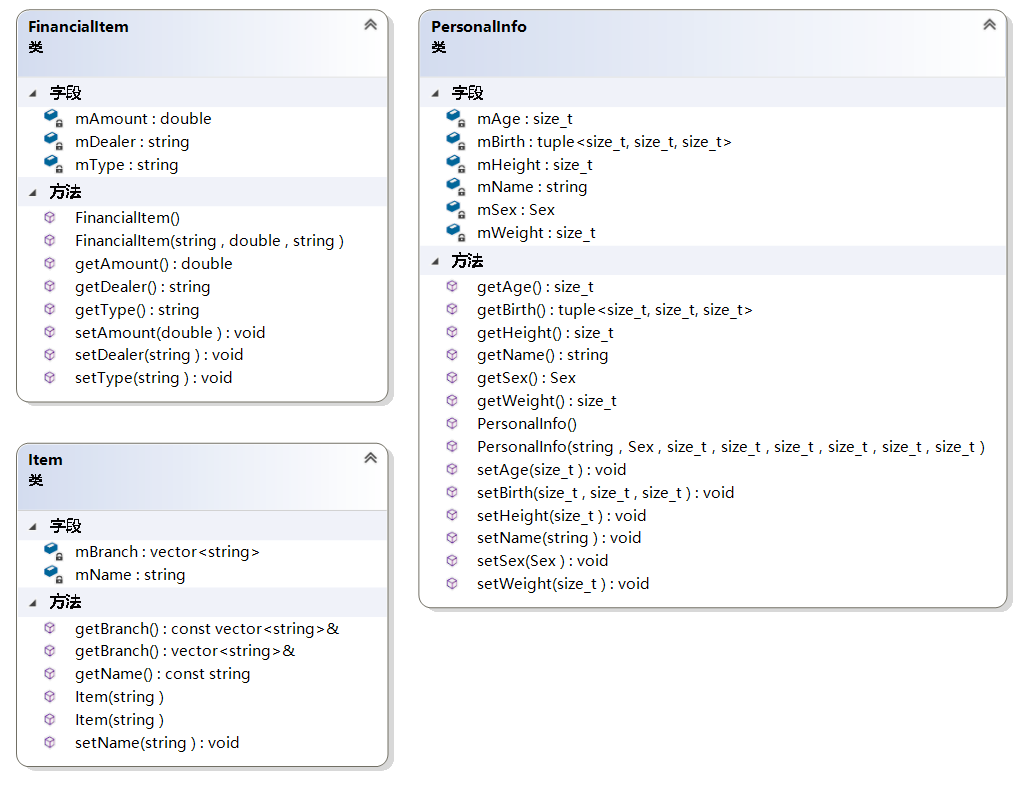
1. **UML图**

根据系统设计思路，首先画出了9个类的UML类图，下面将依次按照3个强枚举类型类，3个小类，3个大类的顺序呈现:

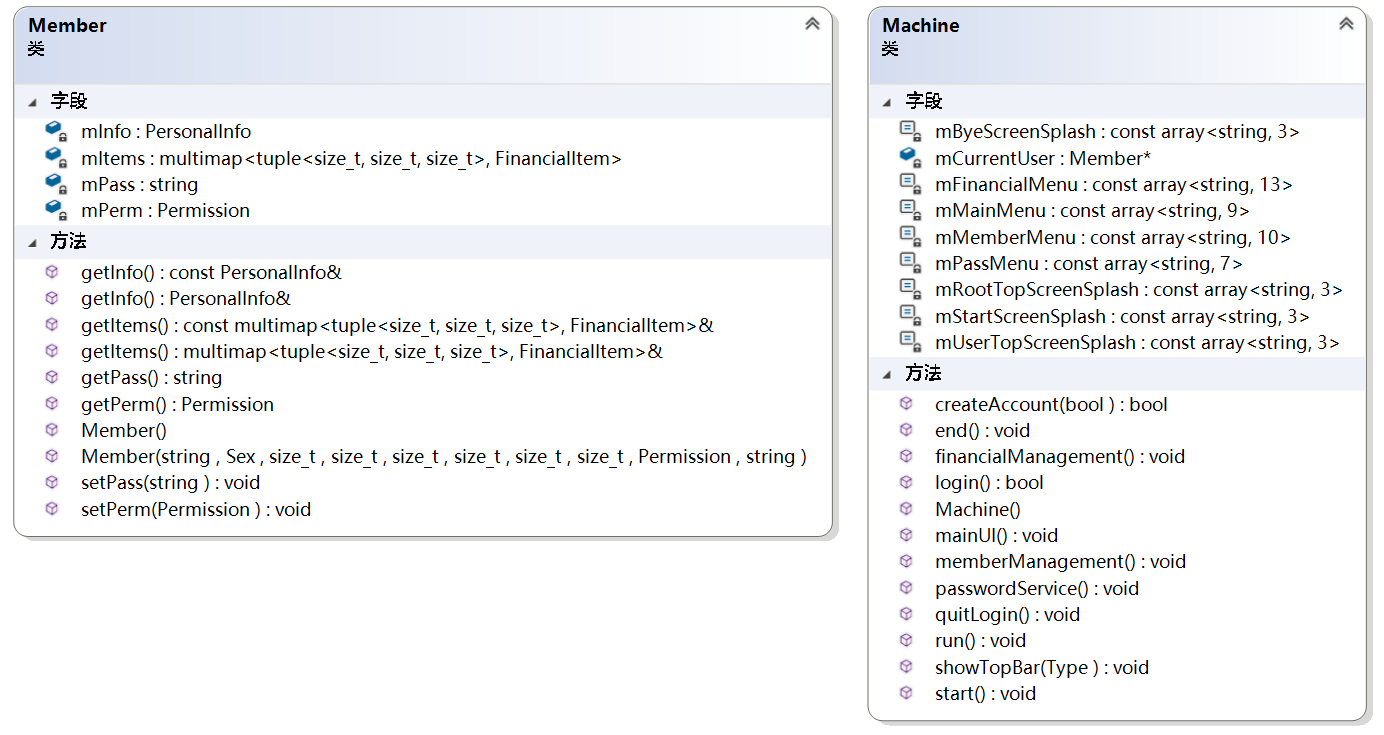
1. 强枚举类型

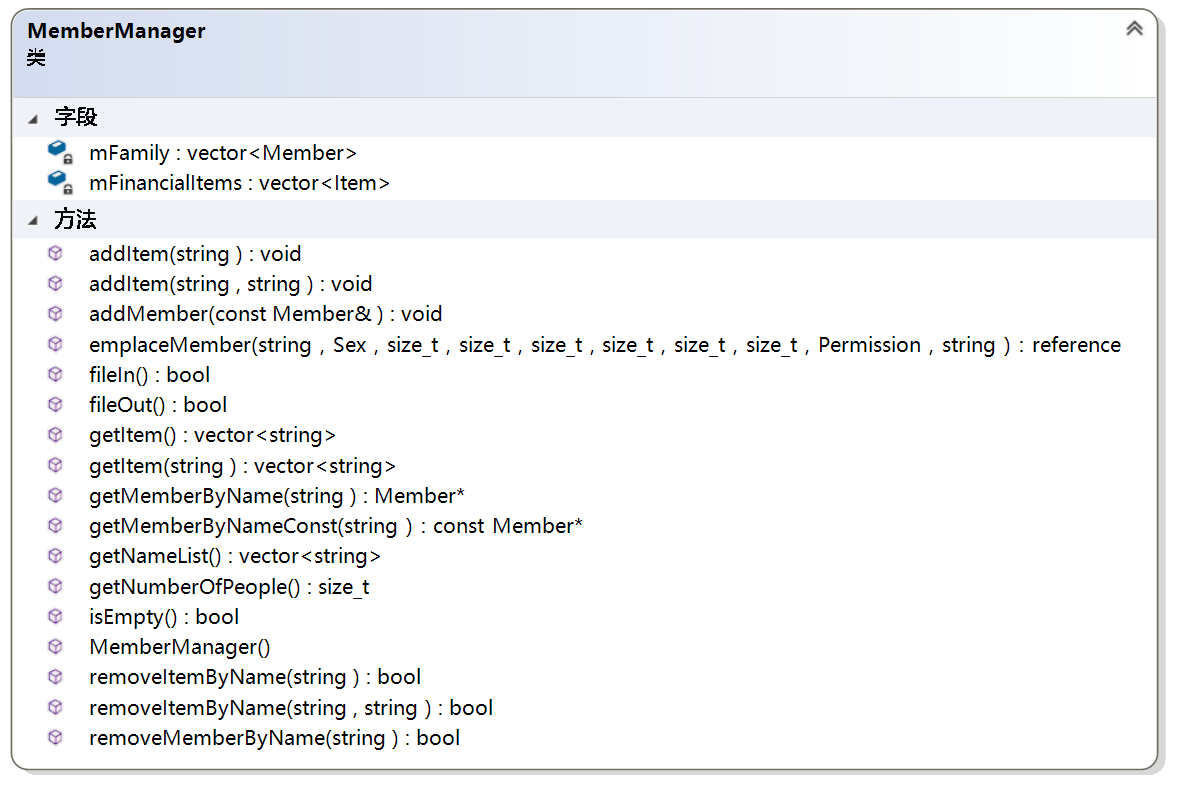


1. 小类



3、大类





1. **系统的设计过程**

根据上面的系统设计思路和UML图，这个部分将展示本项目中所有类的设计过程，展示顺序依次是3个枚举类型，3个小类和3个大类：

1、Sex类：

该类是性别枚举类型，类内包含male和female两个类型用于表示男女两种性别类型。该类将在PersonalInfo类中被使用，该类的对象会作为PersonalInfo类的私有成员，表示成员个人信息中的性别。

2、Permission类：

该类是操作权限枚举类型，类内包含root和user两个类型用于表示管理员和普通用户两种身份类型。该类将在Member类中被使用，该类的对象会作为Member类的私有成员，表示该成员的操作权限。

3、Type类：

该类是当前用户所处状态的枚举类型，类内包含start，top，bye三个类型用于表示程序开始执行，用户已登陆和程序结束三种状态。该类将在Machine类中被使用，该类的对象会作为参数传入Machine类的showTopBar函数中。showTopBar函数的作用将在介绍Machine类的部分中具体介绍。

4、Item类:

该类是一二级收支项目类，该类将在MemberManager类中被使用，作为mFinancialItems的模板参数，表示一个一二级收支项目。

该类的私有部分包含有一个string类的对象mName用于存放一级收支项目名称，一个vector<string>的对象mBranch用于存放该一级项目下所有的二级收支项目。

该类的公有部分提供了无参构造函数和带一个string类形参的构造函数用于初始化该一级收支项目的名称。

公有部分还提供了getName函数用于返回mName，提供了setName函数用于设置mName，提供了getBranch函数用于返回mBranch的引用用于对mBranch进行增删改查。

5、FinancialItem类：

该类是个人单条收支信息类，用于表示一条完整的收支信息。该类将在Member类中被使用，作为mItems的模板参数，表示该成员的一条收支信息。

该类的私有部分包含一个string类的对象mType表示收支种类，一个double类型的对象mAmount表示金额，一个string类的对象mDealer表示经手人。

该类的公有部分提供了无参构造函数和带有3个参数的构造函数用于初始化mType,mAmount和mDealer。

该类的公有部分还提供了三个getter函数getType，getAmount，getDealer用于获取三个私有成员的值，三个setter函数setType，setAmount，setDealer用于设置三个私有成员的值。

6、PersonalInfo类：

该类是成员个人信息类，用于存储一个家庭成员的完整个人信息。该类将在Member类中被使用，该类的对象会作为Member类的私有成员，表示该成员的个人信息。

该类的私有部分包含一个string类的对象mName表示姓名，一个Sex类型的对象mSex表示性别，一个size\_t类型的变量mAge表示年龄，一个tuple<size\_t,size\_t,size\_t>类型的对象表示生日的年月日(这里不自定义一个Date类而使用tuple元组，一是为了方便，二是为了与后面介绍的Member类私有成员中的mItems的键类型保持一致以维持程序代码的一致性)，一个size\_t类型的变量mHeight表示身高，一个size\_t类型的变量表示体重。

该类的公有部分提供了无参构造函数和带有8个参数的构造函数用于初始化mName，mSex，mAge，mBirth，mHeight和mWeight。

该类的公有部分还提供了六个getter函数getName，getSex，getAge，getBirth，getHeight，getWeight用于获取六个私有成员的值，六个setter函数setName，setSex，setAge，setBirth，setHeight，setWeight用于设置六个私有成员的值。其中getBirth返回的是tuple<size\_t,size\_t,size\_t>对象，在使用时用结构化绑定auto[x,y,z]来接受其返回值。setBirth函数接受三个size\_t类型的参数分别表示年月日，并通过初始化列表的方式赋值给mBirth。

7、Member类

该类是存储一个成员所有内容的大类、关键类。成员的个人信息，操作权限，登陆密码，所有收支信息均存储在该类中，该类将在MemberManager类中被使用，作为mFinancialItems的模板参数，表示一个家庭成员。

该类的私有部分有一个PersonalInfo类的对象mInfo表示成员的个人信息，一个Permission类的对象mPerm表示成员的操作权限，一个string类的对象mPass表示成员的登陆密码，一个multimap<tuple<size\_t,size\_t,size\_t>,FinancialItem>类的对象mItems用于存储所有的该成员的所有的收支信息，其中使用tuple<size\_t,size\_t,size\_t>作为键，FinancialItem类的对象作为值，形成multimap的键值对。

使用tuple<size\_t,size\_t,size\_t>作为键而不是自己手写一个Date类并重载比较运算符的原因一是tuple模板类本身就有比较完美的比较运算符重载，它可以根据模板参数的类型生成对应的字典顺序比较运算符重载函数进行比较，这个函数通常比自己手写的重载版本更加优秀并拥有较高的正确率，可以较好的配合multimap的排序功能，因为排序需要比较键的大小。二是可以缩短代码开发时间，不用自己重新实现一个Date类。

使用multimap作为存储容器一是因为multimap本身基于红黑树实现，并且可以自动根据键来排序，具有较高的查找效率，方便通过用户输入的日期并配合upper\_bound和lower\_bound函数快速查找收支信息，二是因为同一日可能又多条收支信息，刚好multimap同一个键可以对应多个值，符合设计需求。

该类的公有部分有无参构造函数和带有10个参数的构造函数用于初始化mInfo，mPerm，mPass，其中mInfo对象的初始化工作在Member类的构造函数的初始化列表中调用PersonalInfo类的构造函数并传参来完成，mItems默认留空，等待元素的插入。

该类的公有部分还提供了四个getter函数，其中getPerm返回一个Permission类的对象用于获得操作权限，getInfo返回mInfo的引用用于对mInfo进行操作，getPass返回mPass用于获得该成员的登录密码，getItems返回mItems的引用，用于对mItems进行增删改查等操作。提供了两个setter函数，其中setPerm用来设置mPerm，setPass用来设置mPass。对于另外两个私有成员不提供setter函数，因为在getter函数中已经返回了相关对象的引用，用户调用相关的getter函数即可拿到相关对象并对其进行操作。

8、MemberManager静态类

该类是静态成员管理类，主要用于管理一二级收支项目和所有的家庭成员。该类为什么设计为静态类在“系统设计思路”部分已经呈现，这里不再赘述。该类主要与Machine类构成联系，形成整个完整的系统。

该类的构造函数设置为delete，因为是静态类，默认不生成类的对象。

该类的私有部分有一个vector<Member>类的对象mFamily用于存放所有家庭成员，有一个vector<Item>类的对象mFinancialItems用于存放一二级收支项目。

该类的公有部分中的emplaceMember和addMember是添加成员的函数，前者接受参数通过传参直接在mFamily中构造一个Member类的对象并用传入的参数来初始化Member成员的值，后者则接受一个Member类的引用，用于直接使用mFamily的push\_back方法将传入的对象复制到mFamily的尾部。

getMemberByName函数接受一个string类的对象作为参数，通过名字查找并返回名字对应的Member类对象的引用。removeMemberByName函数接受一个string类的对象作为参数，通过名字查找并删除mFamily中的名字对应的Member类对象。

getNumberOfPeople函数返回当前mFamily中所拥有的成员的个数。getNameList函数返回所有成员名字列表，返回值是一个vector<string>，用于提供给Machine类做选择索引使用。isEmpty函数用于判断mFamily是否为空。

addItem函数有两个重载版本，第一个重载版本用于添加一级收支项目，接受一个string类的对象作为参数，并生成一个Item类的对象，用传入的string初始化该Item对象并将其放入mFinancialItems中。第二个重载版本用于添加二级收支项目，接受两个string类的对象作为参数，以第一个string类的对象作为值，在mFinancialItems中寻找对应的Item类的对象，找到后，在找到的Item类的对象上调用getBranch函数即可获得该一级收支项目下存放二级收支项目的容器，然后就可以在其中push\_back第二个string类的对象了。

removeItemByName函数也有两个重载版本，第一个重载版本用于删除一级收支项目，接受一个string类的对象作为参数，在mFinancialItems中使用getName逐个寻找对应名字的Item并删除。第二个重载版本用于添加二级收支项目，接受两个string类的对象作为参数，以第一个string类的对象作为值，在mFinancialItems中寻找对应的Item类的对象，找到后，在找到的Item类的对象上调用getBranch函数即可获得该一级收支项目下存放二级收支项目的容器，然后就可以在其中push\_back第二个string类的对象了。

getItem函数也有两个重载版本，第一个重载版本返回一个vector<string>类的对象，里面装着所有的一级收支项目。第二个重载版本接受一个string类的形参，并根据形参在mFinancialItems里面查找对应的一级收支项目，然后获取该一级收支项目下对应的二级收支项目并返回一个vector<string>类的对象，里面装着所有的该一级项目所对应的二级收支项目，如果给定的一级收支项目不存在则返回一个空的vector<string>。

FileIn和FileOut函数是文件的输入输出函数，用于将mFamily和mFinancialItems的内容全部保存到文件中。保存到文件的过程是，先输出总人数到文件，再依次输出每一个人的个人信息和收支信息。收支信息的输出先输出该成员总收支信息的条数再逐条输出收支信息。最后，输出收支项目，输出收支项目时先输出总一级收支项目的个数，逐个输出一级收支项目，输出一级收支项目时，先输出总二级收支项目的个数，再逐个输出二级收支项目。

文件读入的过程与上述保存的过程顺序相同，这里不再赘述。

9、Machine静态类:

该类是UI界面类，主要负责UI界面的显示，用户操作的读取与执行等。该类为什么设计为静态类在“系统设计思路”部分已经呈现，这里不再赘述。该类主要与MemberManager类构成联系，形成整个完整的系统。

该类的构造函数设置为delete，因为是静态类，默认不生成类的对象。

该类的私有部分有8个菜单UI string类的数组，用于存放一些命令行的UI图形。在需要显示相关的图形时直接在控制台上输出数组的内容即可达到目的。这8个菜单UI类的数组分别是: mStartScreenSplash表示开始信息标题栏，mUserTopScreenSplash表示普通用户信息标题栏，mRootTopScreenSplash表示管理员信息标题栏，mByeScreenSplash表示结束信息标题栏，mMainMenu表示主菜单，mMemberMenu表示成员管理菜单，mPassMenu表示密码服务菜单，mFinancialMenu表示收支项目菜单，这些UI的具体内容可以查看源代码获悉。该类的私有部分还保存了一个指向当前用户的Member类的指针mCurrentUser，保存该指针的目的是用该指针来指向当前登录进入系统的成员方便进行个人信息的管理，当前用户操作权限的检查和相关UI的显示等操作。

该类的公有部分有11个函数，其中1个辅助函数，4个流程函数，6个功能函数。其中有需要用户输入的部分都会对用户输入的数据做检查，下面将逐个介绍这些函数的功能。

辅助函数showTopBar接受一个Type类的对象作为参数，作用是显示一些界面UI。如果传入的Type是start则向控制台打印mStartScreenSplash中的内容，表示欢迎，如果Type是top，则根据mCurrentUser指针获得的操作权限来决定显示顶部的UI，如果是管理员则向控制台打印mRootTopScreenSplash中的内容，否则就向控制台中打印mUserTopScreenSplash中的内容。并且会展示出用户的姓名和权限。最后如果传入的Type是end，则向控制台打印mByeScreenSplash中的内容，表示欢迎下次使用。本函数中内置的清屏方法是system(“cls”)，该方法可以刷新控制台重新输出内容。

流程函数run是提供给main函数将整个系统开启用的，该函数将会按顺序调用start，mainUI和end其余三个流程控制函数。

流程控制函数start主要完成是文件读入和系统登录的工作。首先会输出欢迎UI并执行fileIn从文件中读入相关信息，如果文件不存在则转到createAccount函数中执行，然后判断是否账户个数是否为0，如果为0则转到createAccount函数中执行，否则进入login函数中执行登录操作。

流程控制函数mainUI主要完成主菜单的显示，和读取用户输入并转到对应的处理函数去执行相应的操作。该函数输出主菜单UI并读入用户输入，如果用户输入了1则转到memberManagement函数中执行，如果用户输入了2则转到passwordService函数中执行，如果用户输入了3则转到financialManagement函数中执行，如果用户输入了4则转到quitLogin函数中执行，如果用户输入了5则执行return，该函数返回。

流程控制函数end主要完成文件保存和退出系统的工作。该函数会执行fileOut函数来将当前的信息保存到文件，如果保存成功则显示欢迎下次使用的UI并结束程序，如果保存失败则会询问用户是否需要再次保存。

所有的流程控制函数中均由while(true)循环加break来控制界面和菜单UI的循环显示，用if(…)else if(…)…else分支结构来根据用户输入的内容来选择对应的操作。

功能函数login主要执行登录系统的功能。在用户输入用户名和密码后会调用MemberManager的getMemberByName函数获得对应用户名的Member类对象指针并赋给mCurrentUser，然后使用该对象的getPass函数并检查密码是否正确，如果密码正确则完成了登录功能，返回真，否则返回假。

功能函数createAccount主要执行创建账户的功能。该函数接受一个bool类型的参数用于说明当前是在一个账户都没有的情况下创建账户还是在原有账户的基础上继续添加。函数内部会从控制台读入用户输入的信息并检查信息的正确性，最后再调用MemberManager的emplaceMember方法来添加账户。

功能函数quitLogin主要执行退出登录的功能。在退出登录后会询问用户是否需要登录其他账户，是则进入login函数执行，否则进入end函数执行并退出程序。

由于剩下部分的功能函数中涉及到管理员身份验证和管理员选人，所以此处先介绍相关的实现思路。

1. 关于管理员的身份验证，使用的是mCurrentUser的getPerm方法，将返回的结果与Permission::root进行对比，如果一样，则是管理员，否则，就不是管理员而是普通人员。在菜单UI后面标记有“(管理员)”的操作均属于管理员操作，执行相关操作前都会进行管理员权限的验证，如果不是管理员则会提示“您不是管理员”的信息并返回，在后面的功能函数描述部分将不再赘述这一点，只需要知道每个管理员专属操作执行前都会进行身份验证，不会出现越权处理的现象。
2. 关于管理员选人功能，先定义一个Member类的指针target，然后使用MemberManager的getNameList方法获得成员名字列表，让管理员选择人员，选完之后根据选择的结果使用vector<string>的at方法获得管理员选择人员的名字，再将这个名字传入MemberManager的getMemberByName方法中即可获得相应的家庭成员Member类对象的指针，将其赋给target即可完成选人操作，之后即可对target进行一系列的操作。如果不是管理员则可以直接将target赋为mCurrentUser自身，之后便可对target进行一系列的操作。

功能函数memberManagement主要执行主菜单中的成员管理的功能，该函数内置有添加，删除成员，修改个人信息，查看当前成员，管理员权限转让的功能。

1. 添加成员调用createAccount函数完成。
2. 删除成员首先让管理员选人，然后再使用MemberManager的removeMemberByName函数完成。
3. 修改个人信息功能如果是管理员那么首先让管理员选人，然后打印出所有可供修改的个人信息供用户选择，选择完成后再使用Member类中的私有成员的相关公有setter函数完成修改，如果不是管理员则跳过管理员选人环节，直接将target指针设置为mCurrentUser，然后调用相关setter函数完成。
4. 管理员权限转让先让管理员选人，然后再使用Member类的setPerm完成。
5. 查看所有成员则使用MemberManager的getNameList函数完成。

功能函数passwordService主要执行主菜单中的密码服务的功能，该函数内置有查看密码和修改密码的功能。

1. 查看密码功能如果是管理员则先让管理员选人，之后再调用Member类的getPass函数完成，如果不是管理员则跳过管理员选人环节，直接使用getPass函数显示当前用户的密码。
2. 修改密码功能类似，如果是管理员则先让管理员选人，之后再调用Member类的setPass函数完成，如果不是管理员则跳过管理员选人环节，直接使用setPass函数修改当前用户的密码。

功能函数financialManagement主要执行主菜单中的收支管理的功能，该函数内置有收支项目的增删改查和个人收支信息的增删改查功能。

1. 添加收支项目的功能使用MemberManager类的addItem方法完成，添加一级收支项目使用该函数的第一个重载版本，添加二级收支项目使用第二个重载版本，使用前会询问用户将该二级收支项目添加到哪一个一级收支项目下。
2. 删除收支项目，在删除收支项目前，会对收支项目总数检测，看其是否为空，如果为空则提示为空并返回，如果不为空则使用MemberManager的getItem函数获取一二级收支项目并打印到控制台上让用户做选择。选择完成后使用MemberManager的removeItemByName方法移除对应的一级或二级收支项目。
3. 修改收支项目，在修改收支项目前，会对收支项目总数检测，看其是否为空，如果为空则提示为空并返回，如果不为空则使用MemberManager的getItem函数获取一二级收支项目并打印到控制台上让用户做选择。选择完成后让管理员先输入新的收支项目名字，然后使用MemberManager的removeItemByName方法配合MemberManager的addItem方法先移除旧的收支项目后添加新的收支项目来达到修改对应的一级或二级收支项目的目的。
4. 查看收支项目，在查看收支项目前，会对收支项目总数检测，看其是否为空，如果为空则提示为空并返回，如果不为空则使用MemberManager的getItem函数获取一二级收支项目并打印到控制台上供查看。
5. 添加收支功能，首先管理员选人，选择要给谁添加收支信息。选好后即可填写相关的收支信息项目，并选择该收支信息所属的收支项目，选择收支项目时使用MemberManager的getItem函数获取一二级收支项目并打印到控制台上供选择。选择完后就使用Member类的getItems方法先获得multimap对象的引用，然后再调用emplace方法添加收支信息。
6. 删除收支功能，首先管理员选人，选择要删除的收支信息所属的目标成员。选好后首先需要用户输入要删除的收支信息所属的日期（年月日），输入完后通过用户输入的年月日来调用target指针的equal\_range功能获得两个迭代器，首先判断两个迭代器是否相等，如果相等则证明这天没有收支信息，在屏幕上输入无收支并返回上一级操作，如果不相等则将第一个迭代器到第二个迭代器之间的收支信息全部输出，询问用户需要删除哪一条。选择完后就使用Member类的getItems方法先获得multimap对象的引用，然后再调用erase方法删除对应的收支信息。
7. 修改收支功能，首先管理员选人，选择要修改哪位成员的收支信息。选好后首先需要用户输入要修改的收支信息所属的日期（年月日），输入完后通过用户输入的年月日来调用target指针的equal\_range功能获得两个迭代器，首先判断两个迭代器是否相等，如果相等则证明这天没有收支信息，在屏幕上输入无收支并返回上一级操作，如果不相等则将第一个迭代器到第二个迭代器之间的收支信息全部输出，询问用户需要修改哪一条。选择完后就使用Member类的getItems方法先获得multimap对象的引用，然后输出所有可供修改的项并询问用户需要修改什么内容，如果需要修改日期，则采用先创建后删除的方法，先让用户输入日期，然后调用multimap的emplace方法用新的键创建对象，之后再删除旧的对象，完成修改过程。如果修改日期以外的项目则直接调用FinancialItem类的setter方法修改对应的收支信息。
8. 查看收支功能，首先管理员选人，选择要查看哪位成员的收支信息。选完后有3种方式可以查询到成员的收支信息，方式1是查看某一天，使用的方法是equal\_range方法，使用此方法要检查两个迭代器是否相同，相同则证明该日没有收支信息，要输出“该日没有收支信息”，否则就将两个迭代器之间的元素全部输出。方式2是查看全部，直接将multimap中的内容全部输出即可。方式3是查看一个时间段，使用的是upper\_bound配合lower\_bound方法，得到两个迭代器后首先检查是否相同，相同则证明该时间段没有收支信息，要输出“该时间段没有收支信息”，否则将两个迭代器之间的元素输出即可达成目的。

所有的功能函数中均由while(true)循环加break来控制界面和菜单UI的循环显示，用if(…)else if(…)…else分支结构来根据用户输入的内容来选择对应的操作。

1. **程序源代码**

1、machine.h

#ifndef MACHINE\_H

#define MACHINE\_H

#include "membermanager.h"

#include <iostream>

#include <string>

#include <array>

#include <thread>

#include <sstream>

enum class **Type**{start,top,bye};

class **Machine**//机器静态类

{

public:

**Machine**()=delete;//静态类不允许生成类的对象

static void **showTopBar**(Type);//显示顶部标题栏

static void **run**();//机器启动

static void **start**();//开始界面

static void **mainUI**();//主菜单UI

static void **end**();//结束界面

static bool **login**();//登录程序

static bool **createAccount**(bool=false);//创建账户

static void **memberManagement**();//成员管理

static void **passwordService**();//密码服务

static void **financialManagement**();//收支管理

static void **quitLogin**();//退出登录

private:

static Member\* mCurrentUser;//指向当前用户的指针

//各种标题菜单

static const std::array<std::string,3> mStartScreenSplash;//开始信息标题栏

static const std::array<std::string,3> mUserTopScreenSplash;//普通用户信息标题栏

static const std::array<std::string,3> mRootTopScreenSplash;//管理员信息标题栏

static const std::array<std::string,3> mByeScreenSplash;//结束信息标题栏

static const std::array<std::string,9> mMainMenu;//主菜单

static const std::array<std::string,10> mMemberMenu;//成员管理菜单

static const std::array<std::string,7> mPassMenu;//密码服务菜单

static const std::array<std::string,13> mFinancialMenu;//收支项目菜单

};

#endif // MACHINE\_H

2、member.h

#ifndef MEMBER\_H

#define MEMBER\_H

#include <string>

#include <tuple>

#include <map>

enum class **Sex**{male,female};//性别枚举类型

enum class **Permission**{user,root};//操作权限枚举类型

class **FinancialItem**//个人单调收支信息类

{

public:

**FinancialItem**();

**FinancialItem**(std::string,double,std::string);

std::string **getType**() const;//获得收支种类

double **getAmount**() const;//获得金额

std::string **getDealer**() const;//获得经手人

void **setType**(std::string);//设置收支种类

void **setAmount**(double);//设置金额

void **setDealer**(std::string);//设置经手人

private:

std::string mType;//收支种类

double mAmount;//金额

std::string mDealer;//经手人

};

class **PersonalInfo**//成员信息

{

public:

**PersonalInfo**();

**PersonalInfo**(std::string,Sex,size\_t,size\_t,size\_t,size\_t,size\_t,size\_t);

std::string **getName**() const;//获得家庭成员的姓名

Sex **getSex**() const;//获得家庭成员的性别

size\_t **getAge**() const;//获得家庭成员的年龄

std::tuple<size\_t,size\_t,size\_t> **getBirth**() const;//获得家庭成员的生日

size\_t **getHeight**() const;//获得家庭成员的身高

size\_t **getWeight**() const;//获得家庭成员的体重

void **setName**(std::string);//设置家庭成员的姓名

void **setSex**(Sex);//设置家庭成员的性别

void **setAge**(size\_t);//设置家庭成员的年龄

void **setBirth**(size\_t,size\_t,size\_t);//设置家庭成员的生日

void **setHeight**(size\_t);//设置家庭成员的身高

void **setWeight**(size\_t);//设置家庭成员的体重

private:

std::string mName;//家庭成员的姓名

Sex mSex;//家庭成员的性别

size\_t mAge;//家庭成员的年龄

std::tuple<size\_t,size\_t,size\_t> mBirth;//家庭成员的生日

size\_t mHeight;//家庭成员的身高

size\_t mWeight;//家庭成员的体重

};

class **Member**//成员类

{

public:

**Member**();

**Member**(std::string,Sex,size\_t,size\_t,size\_t,size\_t,size\_t,size\_t,Permission,std::string);

Permission **getPerm**() const;//获得家庭成员的操作权限

void **setPerm**(Permission);//设置家庭成员的操作权限

void **setPass**(std::string);//设置家庭成员的密码

PersonalInfo& **getInfo**();//获得家庭成员信息

const PersonalInfo& **getInfo**() const;//获得家庭成员信息(const)

std::string **getPass**() const;//获得家庭成员的密码

std::multimap<std::tuple<size\_t,size\_t,size\_t>,FinancialItem>& **getItems**();//获得家庭成员的所有收支项目

const std::multimap<std::tuple<size\_t,size\_t,size\_t>,FinancialItem>& **getItems**() const;//获得家庭成员的所有收支项目(const)

private:

PersonalInfo mInfo;//家庭成员的信息

Permission mPerm;//家庭成员的操作权限

std::string mPass;//家庭成员的登录密码

std::multimap<std::tuple<size\_t,size\_t,size\_t>,FinancialItem> mItems;//家庭成员的所有收支项目

};

#endif // MEMBER\_H

3、membermanager.h

#ifndef MEMBERMANAGER\_H

#define MEMBERMANAGER\_H

#include "member.h"

#include <fstream>

#include <vector>

#include <algorithm>

#include <unordered\_map>

class **Item**//一级收支项目类

{

public:

**Item**();

**Item**(std::string);

const std::string **getName**() const;//返回一级收支项目名字

std::vector<std::string>& **getBranch**();//返回二级收支项目总和

const std::vector<std::string>& **getBranch**() const;//返回二级收支项目总和Const

void **setName**(std::string);//设置一级收支项目名字

private:

std::string mName;//存放一级收支项目

std::vector<std::string> mBranch;//存放二级收支项目

};

class **MemberManager**//静态成员管理类

{

public:

//静态类不允许生成对象

**MemberManager**()=delete;

static std::vector<Member>::reference **emplaceMember**(std::string,Sex,size\_t,size\_t,size\_t,size\_t,size\_t,size\_t,Permission,std::string);//添加成员

static void **addMember**(const Member&);//添加成员

static void **addItem**(std::string);//添加一级收支项目

static void **addItem**(std::string,std::string);//添加二级收支项目

static bool **removeItemByName**(std::string);//通过名字移除一级收支项目

static bool **removeItemByName**(std::string,std::string);//通过名字移除一级收支项目

static bool **removeMemberByName**(std::string);//通过名字移除成员

static Member\* **getMemberByName**(std::string);//通过名字取得成员引用

static const Member\* **getMemberByNameConst**(std::string);//通过名字取得成员const引用

static size\_t **getNumberOfPeople**();//获得当前人数

static std::vector<std::string> **getNameList**();//获得成员名字列表

static std::vector<std::string> **getItem**();//获得一级收支项目

static std::vector<std::string> **getItem**(std::string);//获得二级收支项目

static bool **isEmpty**();//是非为空

static bool **fileIn**();//文件读入

static bool **fileOut**();//文件输出

private:

static std::vector<Member> mFamily;//存放所有家庭成员

static std::vector<Item> mFinancialItems;//一二级收支项目存放

};

#endif // MEMBERMANAGER\_H

4、machine.cpp(由于该文件有1715行代码，代码量过大，故在此处不做展示，请读者自行前往本报告配带的源代码文件观看)

5、main.cpp

#include "machine.h"

int **main**()

{

Machine::run();

return 0;

}

6、member.cpp

#include "member.h"

//FinancialItem类有关实现

FinancialItem::**FinancialItem**(){}

FinancialItem::**FinancialItem**(std::string type,double amount,std::string dealer)

:mType(type),mAmount(amount),mDealer(dealer){}

std::string FinancialItem::**getType**() const{return mType;}

double FinancialItem::**getAmount**() const{return mAmount;}

std::string FinancialItem::**getDealer**() const{return mDealer;}

void FinancialItem::**setType**(std::string type){mType=type;}

void FinancialItem::**setAmount**(double amount){mAmount=amount;}

void FinancialItem::**setDealer**(std::string dealer){mDealer=dealer;}

//FinancialItem类有关实现结束

//PersonalInfo类有关实现

PersonalInfo::**PersonalInfo**(){}

PersonalInfo::**PersonalInfo**(std::string name,Sex sex,size\_t age,

size\_t year,size\_t month,size\_t day,

size\_t height,size\_t weight)

:mName(name),mSex(sex),mAge(age),mBirth(*year*,*month*,*day*),

mHeight(height),mWeight(weight){}

std::string PersonalInfo::**getName**() const{return mName;}

Sex PersonalInfo::**getSex**() const{return mSex;}

size\_t PersonalInfo::**getAge**() const{return mAge;}

std::tuple<size\_t,size\_t,size\_t> PersonalInfo::**getBirth**() const{return mBirth;}

size\_t PersonalInfo::**getHeight**() const{return mHeight;}

size\_t PersonalInfo::**getWeight**() const{return mWeight;}

void PersonalInfo::**setName**(std::string name){mName=name;}

void PersonalInfo::**setSex**(Sex sex){mSex=sex;}

void PersonalInfo::**setAge**(size\_t age){mAge=age;}

void PersonalInfo::**setBirth**(size\_t year,size\_t month,size\_t day){

mBirth={*year*,*month*,*day*};

}

void PersonalInfo::**setHeight**(size\_t height){mHeight=height;}

void PersonalInfo::**setWeight**(size\_t weight){mWeight=weight;}

//PersonalInfo类有关实现结束

//Member类有关实现

Member::**Member**(){}

Member::**Member**(std::string name,Sex sex,size\_t age,

size\_t year,size\_t month,size\_t day,

size\_t height,size\_t weight,

Permission perm,std::string password)

:mInfo(name,sex,age,year,month,day,height,weight),mPerm(perm),mPass(password){}

Permission Member::**getPerm**() const{return mPerm;}

void Member::**setPerm**(Permission perm){mPerm=perm;}

void Member::**setPass**(std::string password){mPass=password;}

PersonalInfo& Member::**getInfo**(){return mInfo;}

const PersonalInfo& Member::**getInfo**() const{return mInfo;}

std::string Member::**getPass**() const{return mPass;}

std::multimap<std::tuple<size\_t,size\_t,size\_t>,FinancialItem>& Member::**getItems**(){return mItems;}

const std::multimap<std::tuple<size\_t,size\_t,size\_t>,FinancialItem>& Member::**getItems**() const{return mItems;}

//Member类有关实现结束

7、Membermanager.cpp

#include "membermanager.h"

//Item类有关实现

Item::**Item**(){}

Item::**Item**(std::string name):mName(name){}

const std::string Item::**getName**() const{return mName;}

std::vector<std::string>& Item::**getBranch**(){return mBranch;}

const std::vector<std::string>& Item::**getBranch**() const{return mBranch;}

void Item::**setName**(std::string name){mName=name;}

//Item类有关实现结束

//MemberManager类有关实现

std::vector<Member> MemberManager::mFamily{};

std::vector<Item> MemberManager::mFinancialItems{};

std::vector<Member>::reference MemberManager::**emplaceMember**(std::string name, Sex sex, size\_t age,

size\_t year, size\_t month, size\_t day,

size\_t height, size\_t weight,

Permission perm,std::string password){

return mFamily.emplace\_back(*name*,*sex*,*age*,*year*,*month*,*day*,*height*,*weight*,*perm*,*password*);

}

void MemberManager::**addMember**(const Member & member){

mFamily.push\_back(member);

}

void MemberManager::**addItem**(std::string name){

mFinancialItems.emplace\_back(*name*);

}

void MemberManager::**addItem**(std::string name1,std::string name2){

auto it=std::find\_if(begin(*mFinancialItems*),end(*mFinancialItems*),

[&](const Item& item){return item.getName()==name1;});

if(it!=end(*mFinancialItems*)){

it->getBranch().emplace\_back(*name2*);

}

}

bool MemberManager::**removeItemByName**(std::string name){

auto it=std::find\_if(begin(*mFinancialItems*),end(*mFinancialItems*),

[&](const Item& item){return item.getName()==name;});

if(it!=end(*mFinancialItems*)){

mFinancialItems.erase(it);

return true;

}

return false;

}

bool MemberManager::**removeItemByName**(std::string name1,std::string name2){

auto it=std::find\_if(begin(*mFinancialItems*),end(*mFinancialItems*),

[&](const Item& item){return item.getName()==name1;});

if(it!=end(*mFinancialItems*)){

auto it2=std::find\_if(begin(*it->getBranch()*),end(*it->getBranch()*),

[&](const Item& item){return item.getName()==name2;});

if(it2!=end(*it->getBranch()*)){

it->getBranch().erase(it2);

return true;

}

}

return false;

}

bool MemberManager::**removeMemberByName**(std::string name){

auto it = std::find\_if(cbegin(mFamily),cend(mFamily),

[&](const Member& member){

return member.getInfo().getName()==name;

});

if(it==std::cend(mFamily)){

return false;

}

mFamily.erase(it);

return true;

}

Member\* MemberManager::**getMemberByName**(std::string name){

auto it = std::find\_if(begin(*mFamily*),end(*mFamily*),

[&](const Member& member){

return member.getInfo().getName()==name;

});

return it==std::end(*mFamily*)?nullptr:&\*it;

}

const Member\* MemberManager::**getMemberByNameConst**(std::string name){

return getMemberByName(name);

}

size\_t MemberManager::**getNumberOfPeople**(){

return mFamily.size();

}

std::vector<std::string> MemberManager::**getNameList**(){

std::vector<std::string> temp;

for(const auto& member:mFamily){

temp.emplace\_back(member.getInfo().getName());

}

return temp;

}

std::vector<std::string> MemberManager::**getItem**(){

std::vector<std::string> temp;

for(const auto& item : mFinancialItems){

temp.push\_back(item.getName());

}

return temp;

}

std::vector<std::string> MemberManager::**getItem**(std::string name){

auto it=std::find\_if(begin(*mFinancialItems*),end(*mFinancialItems*),

[&](const Item& item){return item.getName()==name;});

std::vector<std::string> temp;

if(it!=end(*mFinancialItems*)){

for(const auto& item : it->getBranch()){

temp.push\_back(item);

}

}

return temp;

}

bool MemberManager::**isEmpty**(){

return mFamily.empty();

}

bool MemberManager::**fileIn**(){

std::fstream fio{R"(family.db)",std::ios::in};

if(!fio) return false;

//总人数的输入

size\_t totalMembers{};

fio>>totalMembers;mFamily.reserve(totalMembers);

for(size\_t i=0;i<totalMembers;++i){

std::string name,password,sex,perm;

size\_t age,year,month,day,height,weight;

//个人信息的输入

fio>>name>>sex>>age>>year>>month>>day>>height>>weight>>perm>>password;

//成员创建

emplaceMember(name,sex=="男"?Sex::male:Sex::female,age,year,month,day,height,weight,perm=="root"?Permission::root:Permission::user,password);

//收支信息的输入

size\_t totalItems{};

fio>>totalItems;//个数输入

for(size\_t j=0;j<totalItems;++j){

std::string type,dealer;

double amount;

fio>>year>>month>>day>>type>>amount>>dealer;

getMemberByName(name)->getItems().emplace(std::tuple<size\_t,size\_t,size\_t>{*year*,*month*,*day*},FinancialItem{type,amount,dealer});

}

}

//个数输入

size\_t size;fio>>size;mFinancialItems.reserve(size);

for(size\_t i=0;i<size;++i){

//一级收支项目名字输入

std::string name;fio>>name;

std::vector<Item>::reference ref=mFinancialItems.emplace\_back(*name*);

//该一级收支项目里包含的二级收支项目的个数输入

size\_t amount;fio>>amount;ref.getBranch().reserve(amount);

for(size\_t j=0;j<amount;++j){

//二级收支项目名字输入

fio>>name;

ref.getBranch().emplace\_back(*name*);

}

}

return true;

}

bool MemberManager::**fileOut**(){

std::fstream fio{R"(family.db)",std::ios::out};

if(!fio) return false;

//总人数的输出

fio<<mFamily.size()<<" ";

for(const auto& member:mFamily){

//个人信息的输出

auto[year,month,day]=member.getInfo().getBirth();

fio<<member.getInfo().getName()<<" "

<<(member.getInfo().getSex()==Sex::male?"男":"女")<<" "

<<member.getInfo().getAge()<<" "

<<year<<" "<<month<<" "<<day<<" "

<<member.getInfo().getHeight()<<" "

<<member.getInfo().getWeight()<<" "

<<(member.getPerm()==Permission::root?"root":"user")<<" "

<<member.getPass()<<" ";

//收支信息的输出

fio<<member.getItems().size()<<" ";//个数输出

for(const auto& item : member.getItems()){

auto[year,month,day]=item.first;

fio<<year<<" "<<month<<" "<<day<<" "

<<item.second.getType()<<" "

<<item.second.getAmount()<<" "

<<item.second.getDealer()<<" ";

}

}

//个数输出

fio<<mFinancialItems.size()<<" ";

for(const auto& i : mFinancialItems){

//一级收支项目名字输出

fio<<i.getName()<<" ";

//该一级收支项目里包含的二级收支项目的个数输出

fio<<i.getBranch().size()<<" ";

for(const auto &j : i.getBranch()){

//二级收支项目名字输出

fio<<j<<" ";

}

}

return true;

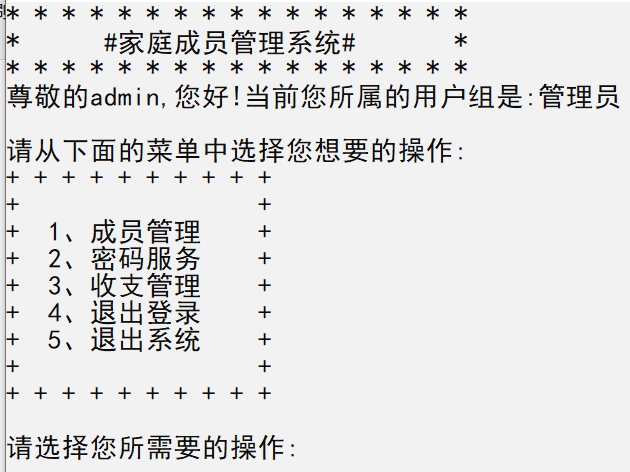
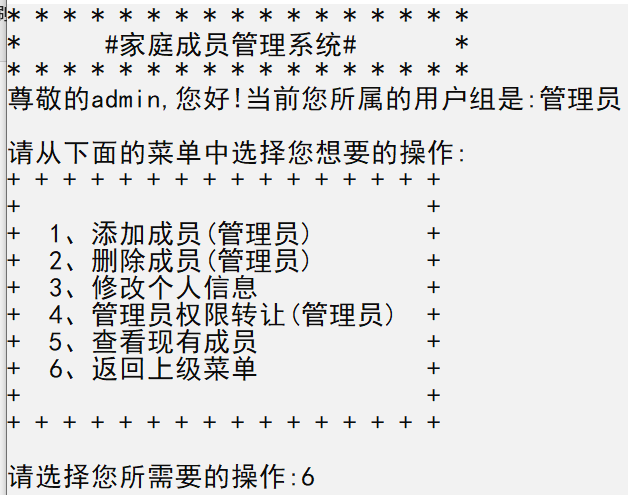
}

//MemberManager类有关实现结束

1. **系统测试情况**

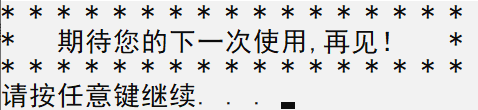
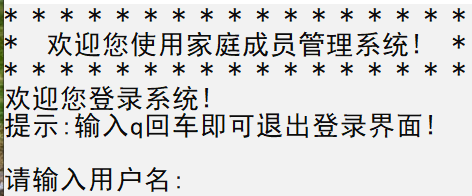
本次测试包括了系统的全部功能项目，都是由我本人进行测试，测试的结果将逐一呈现。

1. 返回功能方面，经测试，所有的“返回上级菜单”、“取消操作”、“退出登录”、“退出系统”之类的返回功能都已经通过测试，可以正常运作。



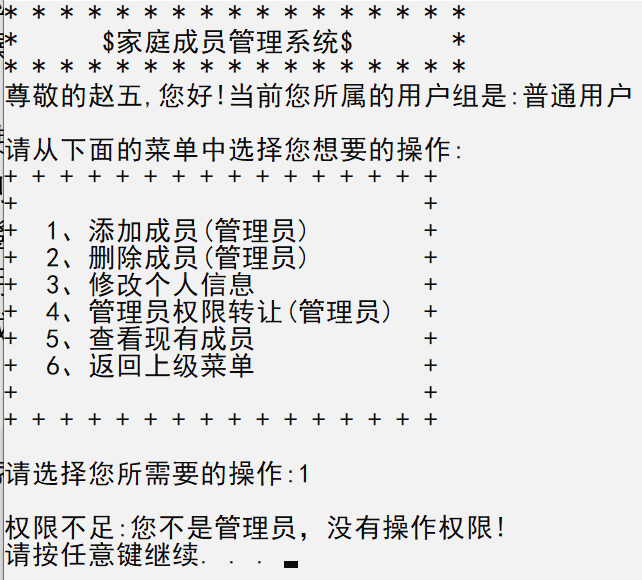
成功返回主菜单

1. 文件存储方面，经测试，系统的文件保存功能正常，可以从文件中读取用户信息，也可以将用户信息写出到文件。文件存储在软件所在目录下，名字为”family.db”。

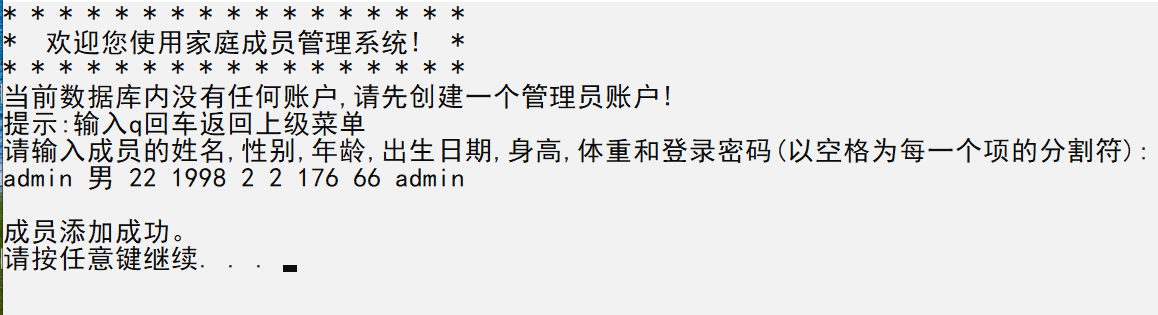


读取成功进入登录页面 成功保存到文件

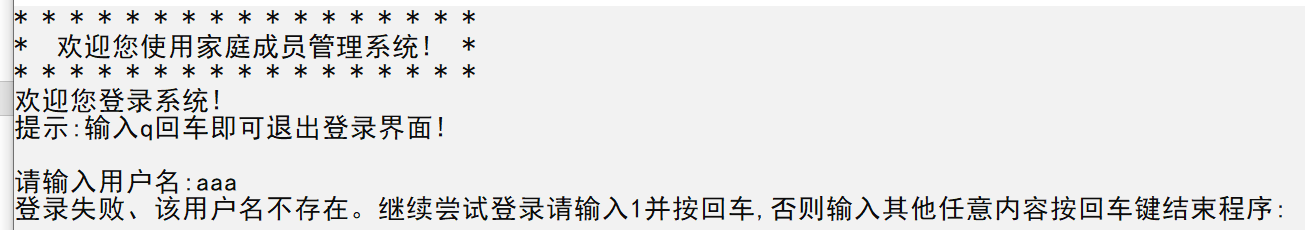
1. 权限检测方面，经测试，管理员的权限检测功能正常，普通用户无管理员权限无法操作管理员专属操作。

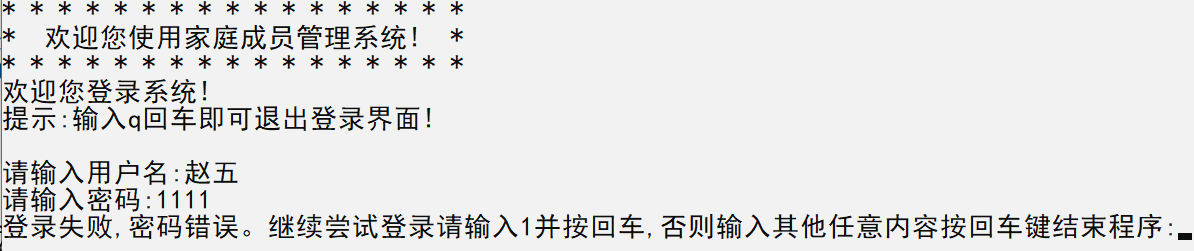


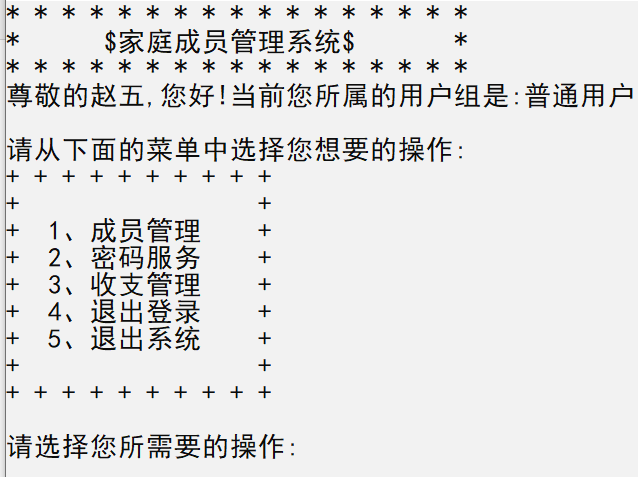
1. 首次使用系统方面，经测试，第一次使用系统时可以正常进入并可以成功引导用户创建系统管理员账户。



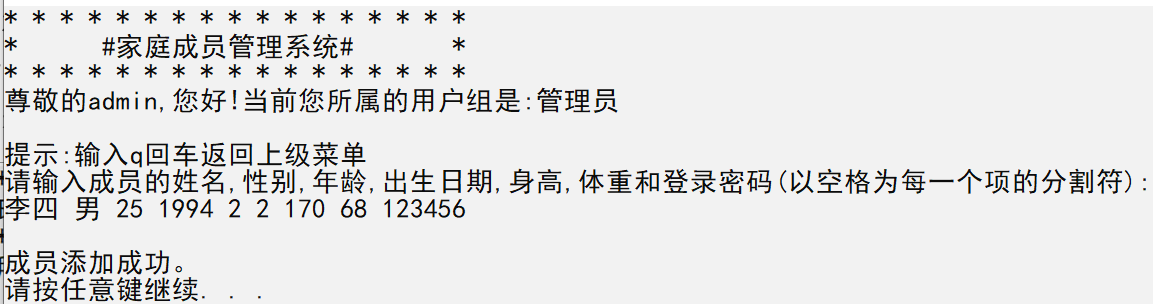
1. 系统登录方面，经测试，系统的登录功能正常，用户名或密码不正确时均会提示，用户名和密码都正确时成功进入系统主菜单界面。

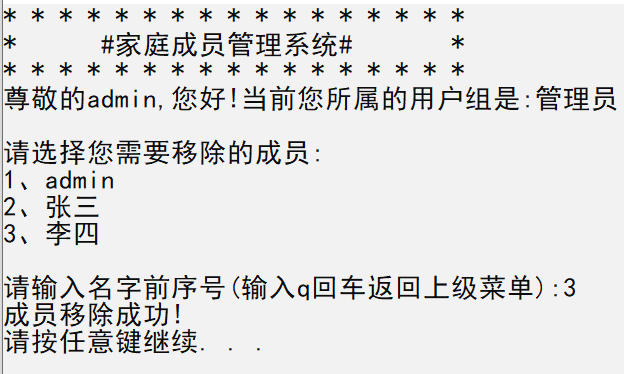
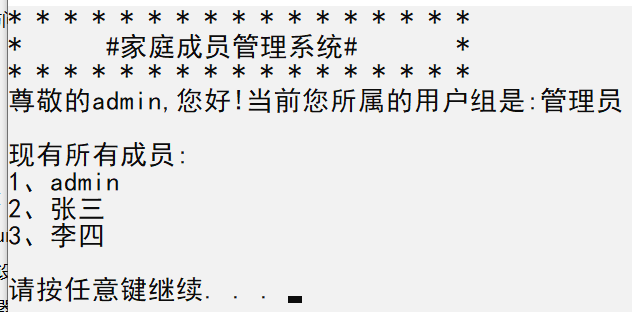


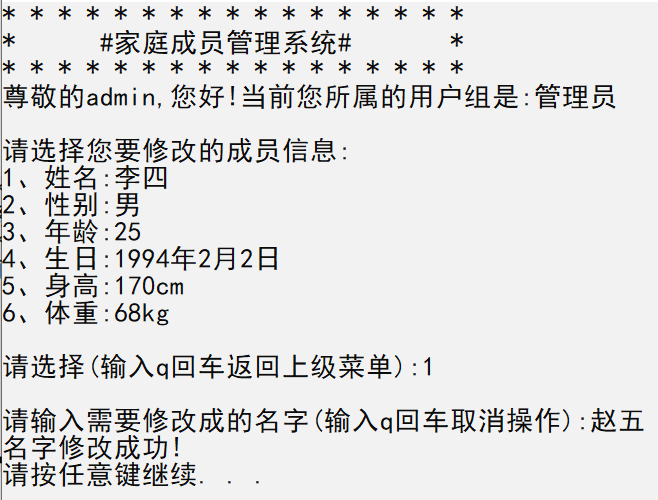




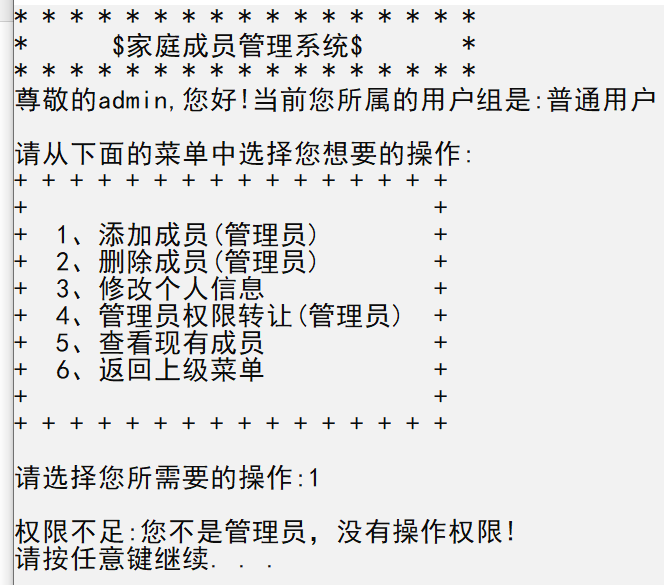
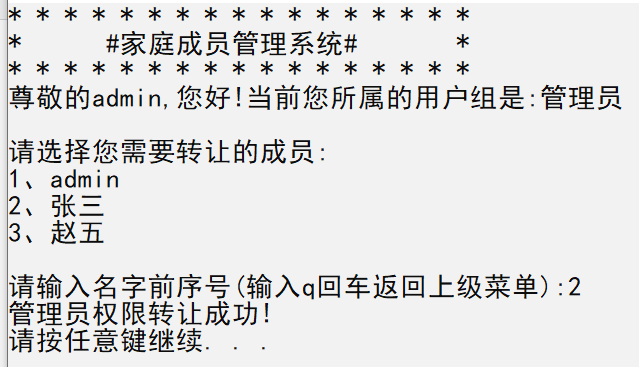
1. 家庭成员管理方面，经测试，家庭成员的增删改查功能工作正常，可以正常的添加，删除，查看成员，也可以很方便的修改家庭成员的个人信息。



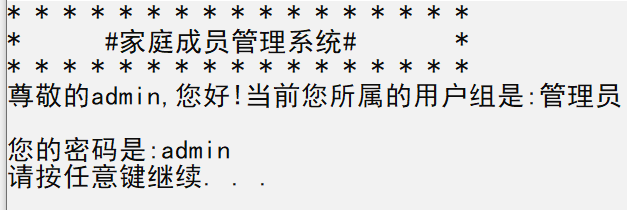
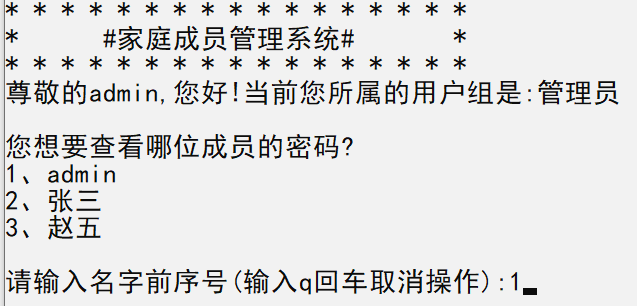




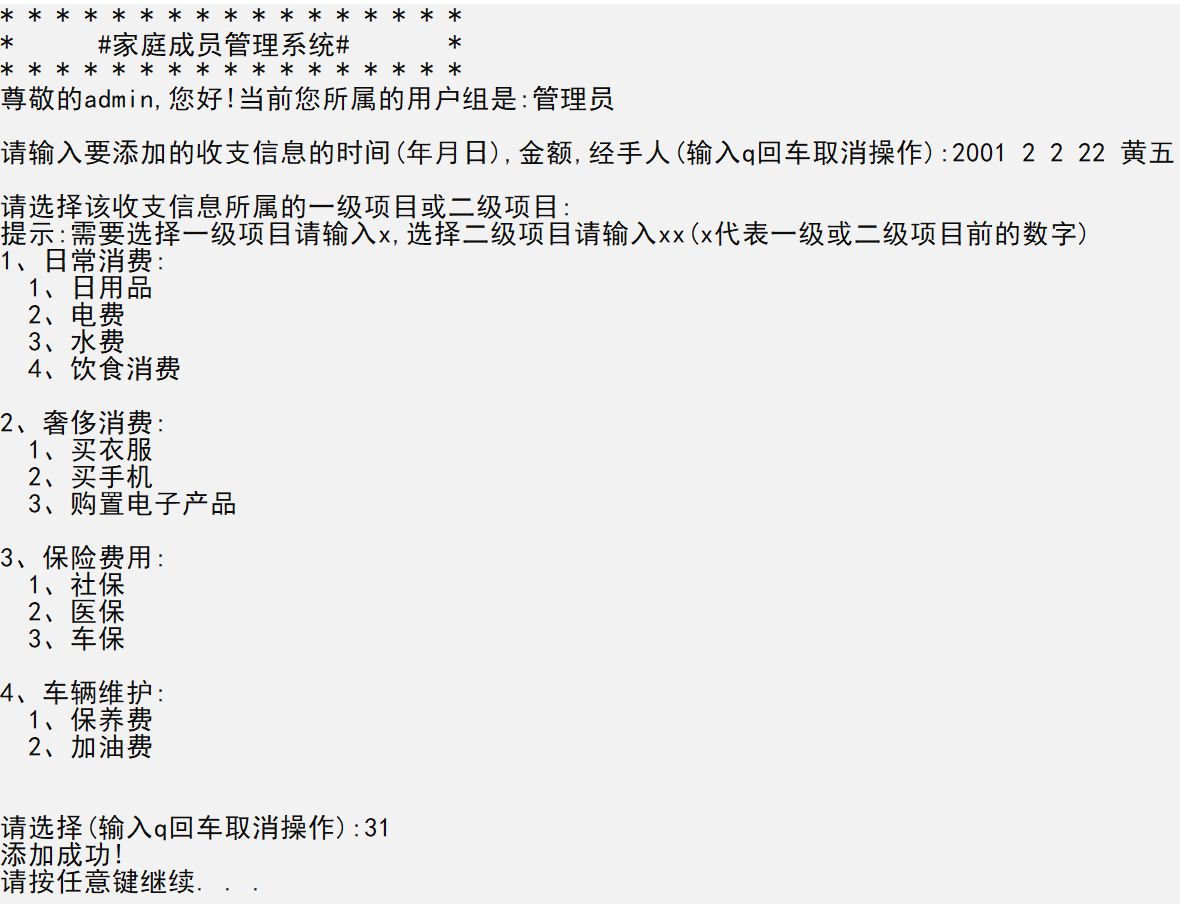
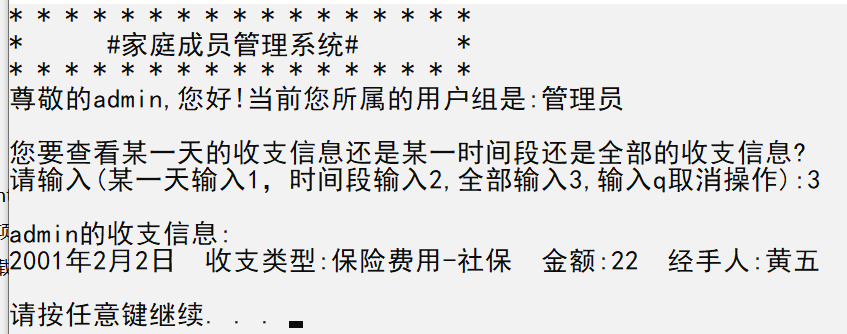
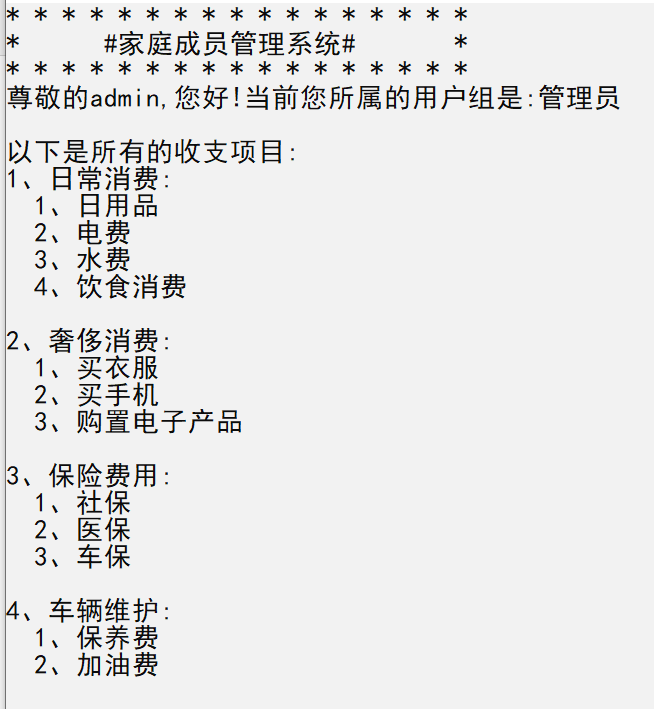
1. 管理员权限转让方面，经测试，管理员权限转让功能工作正常，可以正常的将管理员权限转让给他人，并且转让后自己无法再操作管理员专属操作。



1. 密码服务方面，经测试，管理员可以成功的查看和修改家庭成员的密码，其余普通用户可以查看和修改自己的密码。



1. 收支项目方面，经测试，收支项目的增删改查功能工作正常，成员收支信息的增删改查功能工作正常。能够正常的添加删除修改和查询。



至此，所有的测试已经通过，软件可以正常使用。

1. **系统的优点与改进**

系统的优点:

1. 该系统涵盖了比较丰富的功能，基本上能满足家庭收支管理的日常需求。
2. 有比较人性化的UI设计，在每一步怎么操作也有相关的引导信息。
3. 使用了multimap结构，由于其底层是红黑树的结构，所以在收支信息的查找上会比较快，以高效率的存储结构来节省查找时间，减少用户的等待时间，给用户最好的产品体验。
4. “返回”的设置很多，基本能够在任何时间返回上一层菜单或上一个操作，只要用户后悔了，随时都可以返回，用户的操作空间很大。

系统的可改进之处:

1. 由于本系统使用的是DOS设计，所以界面给人的感觉还是不够完美，如果可以使用MFC的话用户体验就可以更上一层楼了。
2. 家庭成员的文件保存不能自己指定，只能由程序员指定，对用户来说不太友好，如果使用MFC的话，可以弹出对话框让用户选择去读入哪个文件，写出到哪个文件，这样用户就可以获得更大的控制权，可以将家庭成员的文件保存到任何地方，还能写出到任何地方，这样就可以通过多文件来实现一个大家庭下多个小家庭的管理了。
3. 密码直接保存在文件中，比较容易被他人修改而登录进入系统，如果考虑安全性可以使用MD5算法加密一下密码，但是考虑到过于复杂，故没有这样处理。
4. **感想**

通过这次设计这个实验，我掌握了许多编码技巧，学习到了许多新的知识，同时还体验到了一个人开发一整个完整的系统的感觉。

和上个学期的团体设计来比较，这个项目的设计给我的感觉就是少了一些人帮你排忧解难，遇到问题只能靠自己不断的去摸索来解答，幸苦了不少，但是这却提升了我个人解决问题的能力，要知道绝大多数人，在绝大多数时候，都只能靠自己。

于此同时，我也学习到了软件测试要尽早完成，最好是做好一个模块就测试一个模块，不然等你整个系统都做完了才发现有问题的话就需要花费上不少的功夫去查询问题到底出在哪里了。

这次实验也让我掌握了许多的文献阅读技巧，提升了自己阅读文献的能力，知道要边看文献边跟着做，永远不能眼高手低，你不尝试一下永远不会知道自己到底掌握了多少东西，而这次实验学到最多的，掌握最多的就是关于STL方面的知识和algorithm头文件里面的算法的用法。

最后，我认为，我还需要继续提升我的这些相关的能力，因为以后的路还很长，身边的事例都在告诉我，自己还不够优秀，想要成为自己想成为的人就必须付出更多的努力，比别人掌握更多的东西，比别人拥有更多的能力。