**北京科技大学实验报告**

学院：计算机与通信工程学院 专业：信息安全 班级：信安201

姓名：曾绎哲 学号：42024058 实验日期：2021年10 月 27 日

**实验名称：个人银行账户管理系统的改进**

**实验目的：**

完善银行账户系统，并实现以下功能：

1.加入异常处理机制，判断额度超出，日期不合法等问题。

2.加入注册函数，登录函数，登录时密码是否正确的判断函数，用于用户注册与登录。

3.若用户信用卡欠款，则登陆后自动提醒该用户信用卡欠款。

4.加入查询账目信息函数，用户可以自由选择按照交易金额排序或者日期排序查询用户想要的某一段时间内的账目信息。

**实验仪器：**

计算机：Huawei MateBook14

CPU：Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz 2.11 GHz

内存：16 GB DDR4 2666MHz

硬盘：512GB

显卡：NVIDIA GeForce MX350

操作系统：Ubuntu18.04（Window10/Ubuntu18.04双系统）

编译器：g++

**实验内容与步骤：**

**【**功能1：异常判断与处理**】**

1.设计

对于异常判断，我分为了日期不合法异常以及额度超出异常。（1）日期不合法异常直接抛至系统自带的runtime\_error中；（2）额度超出则由exception类派生出AccountException类。当储蓄卡超出余额时以及信用卡超出额度时都会将异常抛出至该类中。

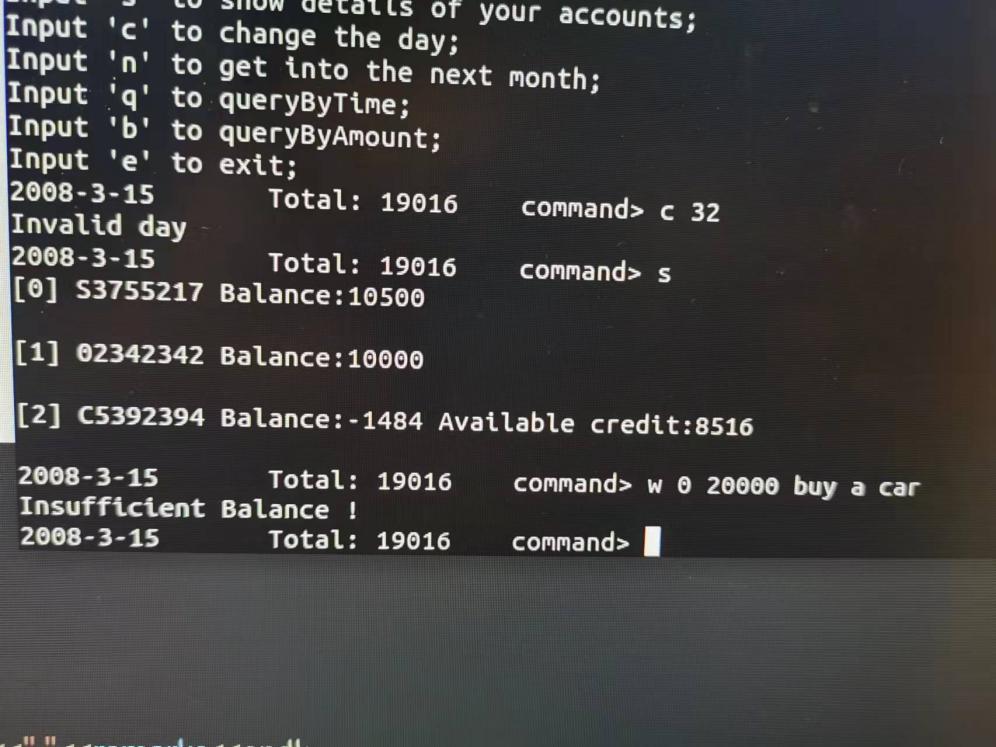
2.实现

|  |
| --- |
| class AccountException:public exception //派生异常类  {      public:      AccountException(string mes)      {          error=mes;      }      static string error;//利用构造函数存储异常信息  };  void SavingsAccount::withdraw(Date date,double amount,string remarks)  {      try{      if((balance-amount)<0)  {  flag=1;  throw AccountException("Insufficient Balance !");  }//储蓄类账户取款数目大于余额时抛出异常至派生类      else  {  ...      }      catch(AccountException)      {cout<<AccountException::error<<endl;}//处理异常  }  void CreditAccount::withdraw(Date date,double amount,string remarks)  {      try{      if((balance-amount+credit)<0)  {flag=1;  throw AccountException("Insufficient Credit !");//信用卡账户取款数目大于信用额度时抛出异常至派生类  }      else  {  ...      }      catch(AccountException)      {cout<<AccountException::error<<endl;}//处理异常  }  Date::Date(int y,int m,int d)  {      try{      year=y;      month=m;      day=d;      int Day[12]={31,28,31,30,31,30,31,31,30,31,30,31};      if((year%4==0 && year%100!=0) || (year%400==0))      Day[1]=29;      else      Day[1]=28;      if(day>Day[month-1])      throw std::runtime\_error("Illegal Date !");      }//日期不合法时抛出异常至系统的runtime\_error中      catch(std::runtime\_error){      cerr<<"Illegal Date !"<<endl;}}//处理异常 |

代码片段1 异常判断与处理的关键代码

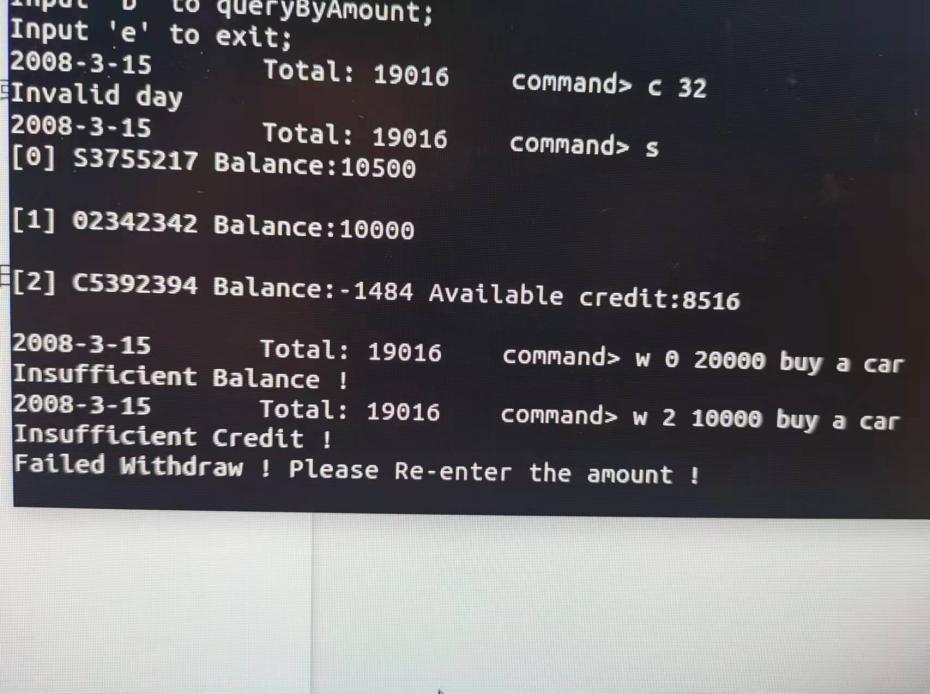
3.测试

（1）测试用例1：w 0 20000 buy a car（超出储蓄卡余额）

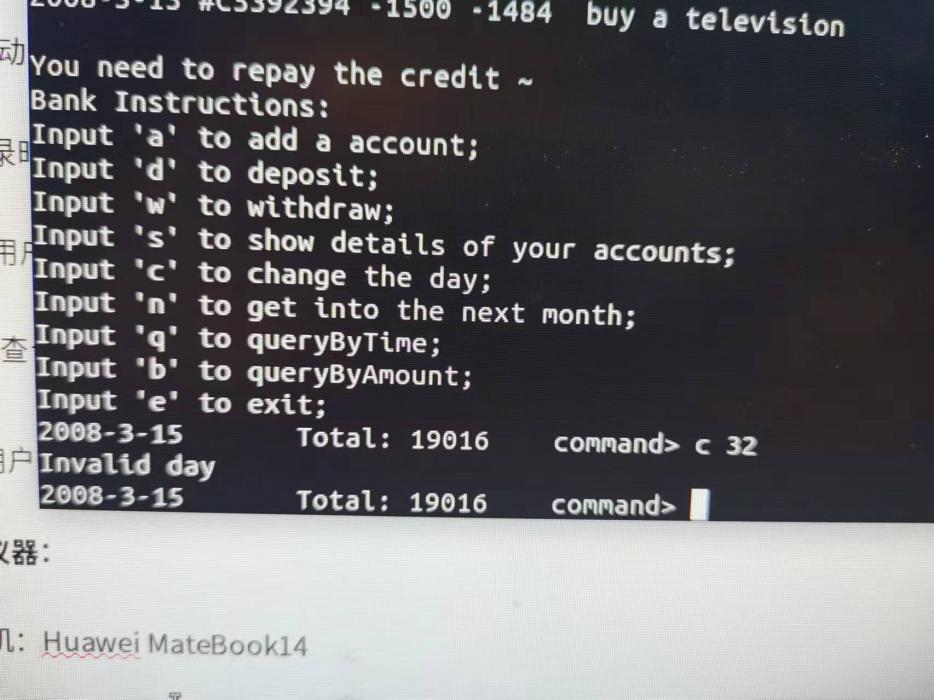
得到的运行截图如下：

（2）测试用例2：w 2 10000 buy a car（超出信用卡的信用额度）

得到的运行截图如下：

1. 测试用例3: c 32（用户企图更改日期为10月32日，为不合法日期）

得到的运行截图如下：



1. 分析

设立这些测试用例的目的是检测异常判断与处理是否正确执行，防止代码出现bug，导致运行结果错误。运行结果表明异常判断与处理执行正常，有效地实现了该功能。

**【**功能2：注册与登录**】**

1.设计

直接在主函数的main.cpp中编写Login以及Register函数以实现登录和注册两个功能。

对于Login函数，当用户选择登录时，便需要输入账户名和密码，随后系统会在allaccounts.txt中搜寻该用户的账户名，若存在：（1）密码正确即为登陆成功；（2）密码错误则会提示密码错误，随后用户可自由选择退出或者是重新输入密码，重新输入直至密码正确便可以成功登录。若不存在：则会提示用户该用户不存在，询问用户是否需要注册，用户可自由选择注册或者退出，一旦注册，便会生成以该用户账户名命名的txt文件，存放该用户的账号密码信息以及操作信息，并在allaccount.txt中加入该账户名，注册成功。

对于Register函数，用户可在银行最开始的界面选择是登录还是注册，若选择注册，便可以直接进行注册程序。

密码正确与否的判断方法为直接打开以该用户命名的txt文件读取密码信息，并将其与输入的密码进行比对，返回真值，从而判断密码是否正确。

1. 实现

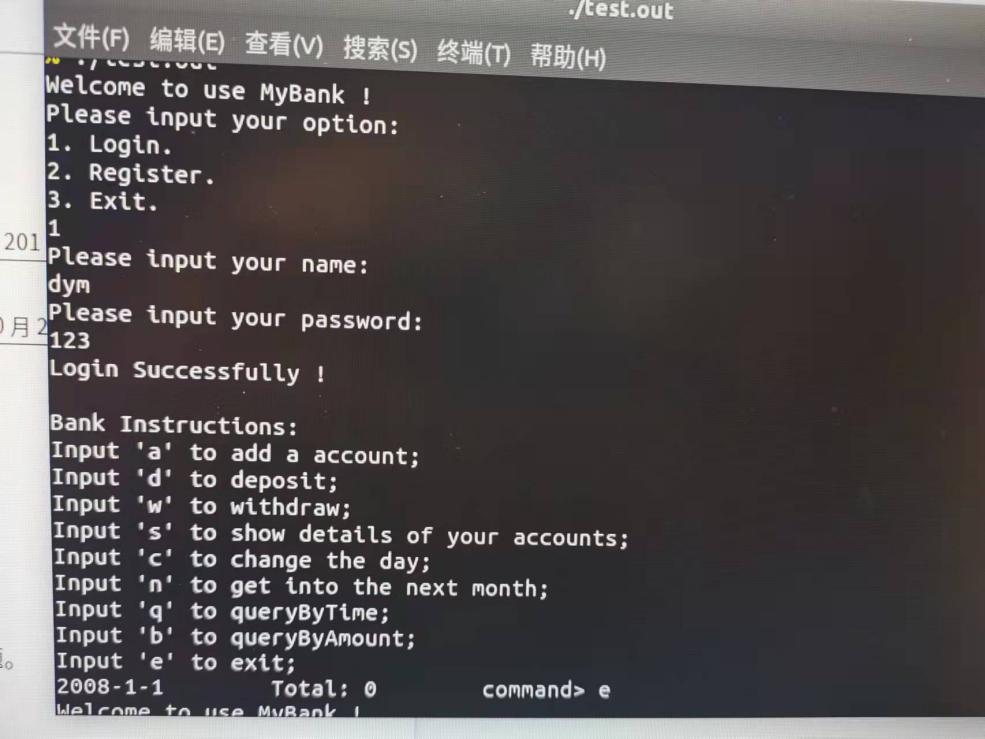
|  |
| --- |
| void Login()//登录函数  {      while (true)//无限循环用于判断是否登录成功      {          ofstream write;int judge=0;          ifstream fin;          cout<<"Please input your name:"<<endl;          cin>>name;          cout<<"Please input your password:"<<endl;          cin>>password;          write.open("allaccounts.txt",ios::app);write.close();          fin.open("allaccounts.txt",ios::in);          string s,Option1;          int Flag=0;          while(getline(fin,s))//遍历银行已注册人员名单          {              if(s==name)              {                  Flag=1;                  break;//找到即说明已注册，Flag状态由0变为1              }          }          fin.close();          if(Flag==0)//说明用户未注册          {              cout<<"UnRegistered !"<<endl;loginjudge=0;break;          }          if(Flag==1)//说明用户已注册          {              string str;              fin.open(name+".txt",ios::in);              getline(fin,str);              getline(fin,str);              string d=("password:"+password);              if(str==d)//密码正确，登录成功              {cout<<"Login Successfully !"<<endl;fin.close();break;}              else//密码错误，登录失败              {                  cout<<"User name or password is incorrect !"<<endl;                  cout<<"1.Retry"<<endl;                  cout<<"2.Exit"<<endl;                  cin>>Option1;//用户自由选择，退出 or 重试                  while(Option1=="1")//重试                  {                      cout<<"Please input your name:"<<endl;                      cin>>name;                      cout<<"Please input your password:"<<endl;                      cin>>password;                      d=("password:"+password);                      if(str==d)                      {cout<<"Login Successfully !"<<endl;judge=1;fin.close();break;}//密码正确，登录成功，退出循环                      else //密码错误，再次循环，用户自由选择                      {                          cout<<"User name or password is incorrect !"<<endl;                          cout<<"1.Retry"<<endl;                          cout<<"2.Exit"<<endl;                          fin.close();string Option1;                          cin>>Option1;                      }                  }                  if(Option1=="2")//用户选择退出，程序退出                  loginjudge=-1;              }          }          if(judge==1 || Option1=="2")          break;//密码输入正确 or 用户自主选择退出 都使得循环终止，登录函数结束      }  }  void Register()//注册函数  {      cout<<"Please input your name:"<<endl;      cin>>name;      cout<<"Please input your password:(If you do not want to register now,Please input 'q'  !)"<<endl;      cin>>password;      if(password=="q")          {loginjudge=0;}//若用户中途不想注册，输入q即可退出      if(password!="q")//继续注册      {ofstream write;      write.open("allaccounts.txt",ios::app);      write<<name<<endl;      write.close();      write.open(name+".txt",ios::app);      write<<"name:"<<name<<endl;      write<<"password:"<<password<<endl;      write.close();      cout<<"Register Successfully !"<<endl;loginjudge=1;}//写入账户以及密码信息，注册结束  } |

代码片段2 登录与注册的关键代码

3.测试

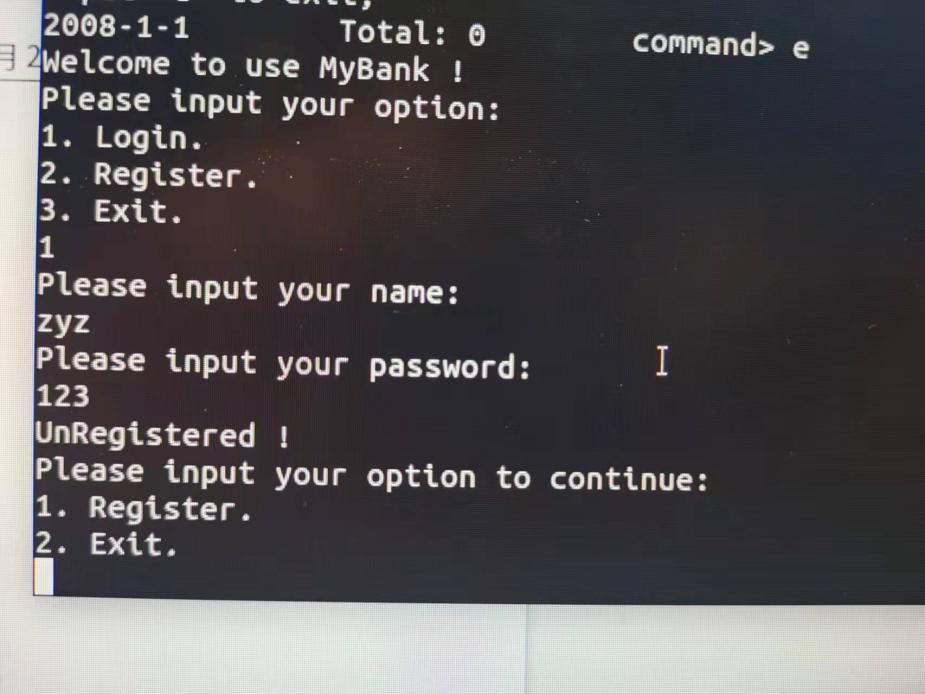
（1）测试用例1：1 【enter】 dym 123（用户dym进行登录操作，密码为123，他是已注册用户）

得到的运行截图如下：



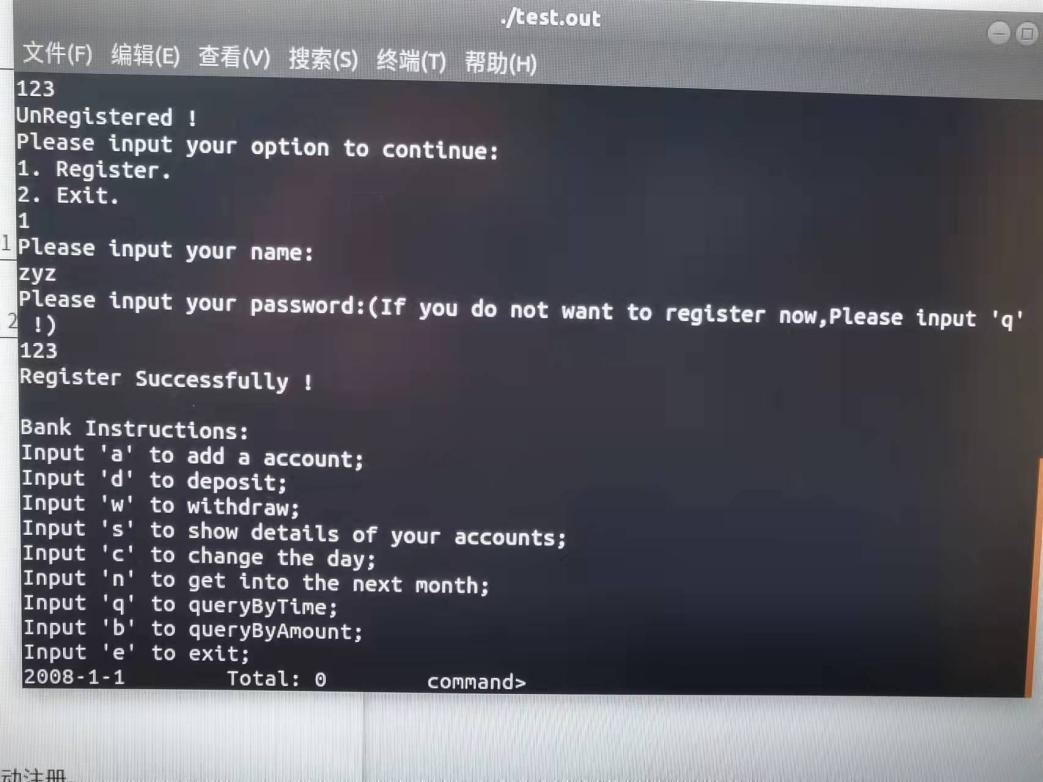
1. 测试用例2：1 【enter】 zyz 123 （用户zyz进行登录操作，密码为123，他是未注册用户）

得到的运行截图如下：

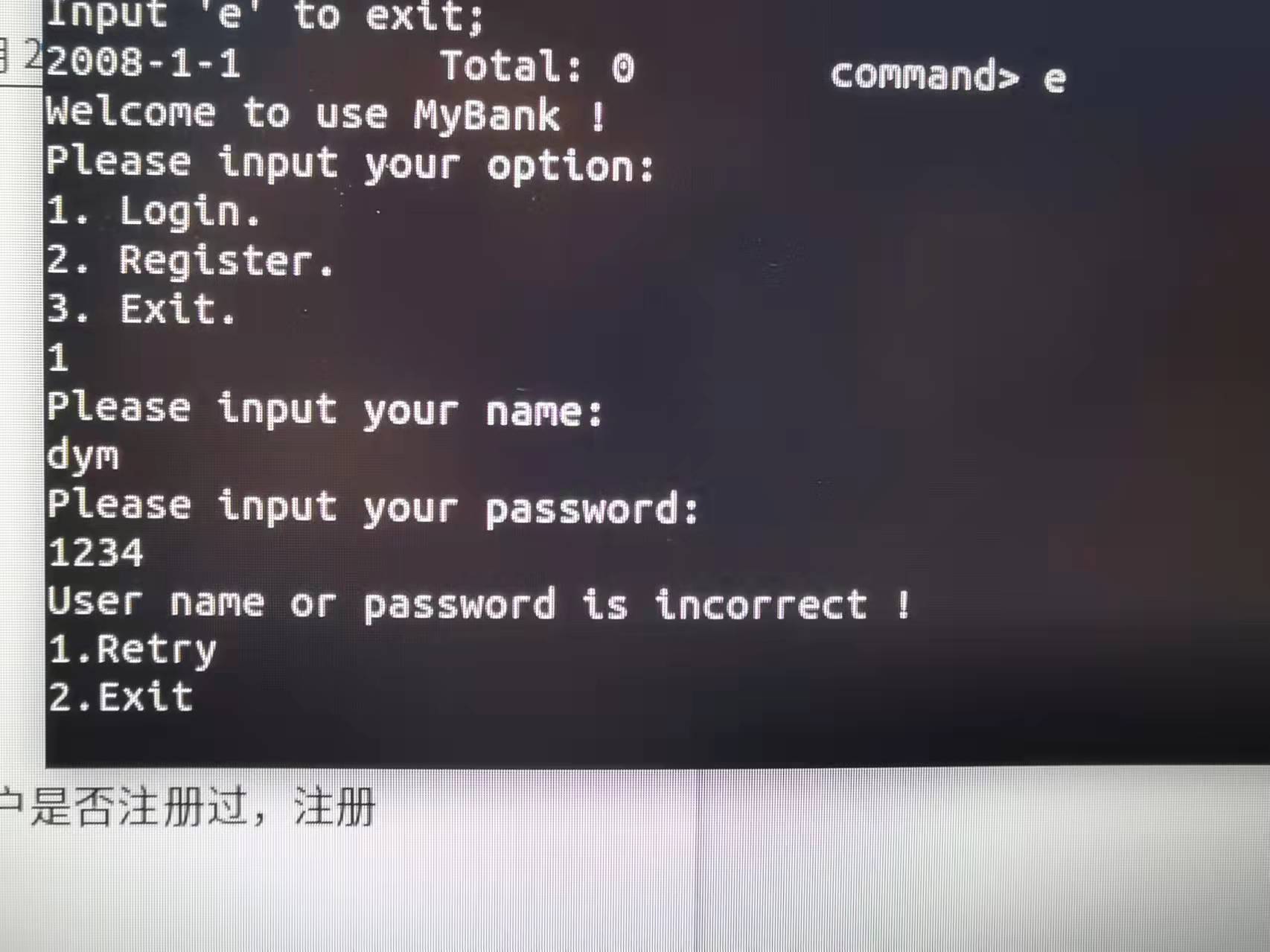


（3）测试用例3：1 【enter】 zyz 123 【enter】 1（用户zyz进行登录操作，密码为123，他是未注册用户，登陆失败后选择注册）

得到的运行截图如下：



（4）测试用例4：1 【enter】 dym 1234（用户dym进行登录操作，输入了错误的密码）

得到的运行截图如下：

4.分析

Login函数与Register函数的加入使得银行系统具有了最初步的UI界面效果，给予了用户一定的自主选择权，并且使得银行系统更加地真实，全面。加入这两个函数的目的是使银行系统具有一个初步的“系统”形态，可以让用户自主操作，贴近实际。而这些测试用例也表示这两个函数基本上实现了对于登录注册以及一系列相关操作的功能。

【功能3：登录后信用卡提醒还款】

1. 设计

每当用户登录后，会利用重定向自动读取“用户名+commands.txt”文件恢复该用户的所有操作数据，从而获得该用户所有的账户信息。一旦发现该用户的信用卡处于欠款状态，便会提醒该用户还款。

1. 实现

|  |
| --- |
| //Redirect read 恢复用户数据                 write.open(name+"commands.txt",ios::app);write.close();                  fin.open(name+"commands.txt",ios::in);                  streambuf \*oldcin;                  oldcin=cin.rdbuf(fin.rdbuf());                  char cmd;Account\* account;                  cin>>cmd;                  do {                  ...                  } while (cin>>cmd);  //Remind                  cout<<endl;                for (size\_t i = 0; i < accounts.size(); i++)//遍历所有账户                  {                      if((accounts[i]->balance)<0)//寻找余额小于0的账户                      cout<<"You need to repay the credit ~"<<endl;//找到并提醒还款                  }                  //Console input                  cin.rdbuf(oldcin);                  fin.close(); |

代码片段3 信用卡提醒还款的关键代码

1. 测试

（1）测试用例1：1 【enter】 dym 123 （用户dym进行登录，密码123，由于该账户信用卡余额小于0，故系统提醒用户还款）

得到的运行截图如下：



1. 分析

加入还款提醒的目的是为了更加接近实际，进一步完善银行系统，使得银行系统更加真实。而为了实现该功能，可以使用重定向以及账户指针的遍历，轻松实现这个简单的功能。该测试用例说明该功能正常执行，可以实现用户还款提醒。

【功能4：账目信息查询】

1. 设计

编写query函数以及queryByAmount函数，实现用户可以按照日期顺序或者按照交易金额大小查询想要查询的一段时间内的账户信息两个功能。

首先编写read函数，可以将输入的日期字符串转换为Date类型的数据并返回。

对于query函数：用户可以查询自己想要查询的一段时间内，按照时间顺序进行的账户信息。用户只需要输入两个不同的日期，系统就会从该用户所有的操作记录中筛选出处于两个时间之间的操作记录，日期大小的比较采用将日期转换为int类型的方法进行比较。筛选之后，便可以输出该条记录。

对于queryByAmount函数：采用与query函数中相同的比较方式，筛选出该时间段内的操作记录，然后按照交易金额的大小进行排序，最后输出即可。

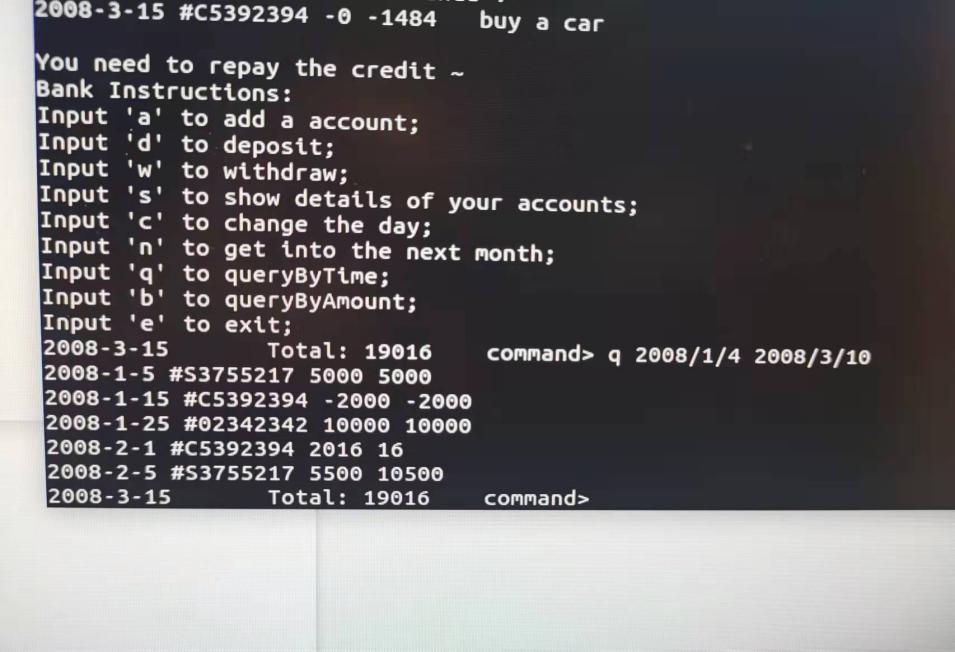
1. 实现

|  |
| --- |
| Date Date::read()  {     string date;      cin>>date;      int tempyear,tempmonth,tempday;      int length=date.length();int num=0;      string s="";      stringstream ss;//将string类型转换为int类型      ss<<date.substr(0,4);      ss>>tempyear;//存放年份      ss.clear();      for(int i=5;i<length;++i)//由于输入的日期形式为“year/month/day”的形式，因此需要去除“/”      {          if(date[i]!='/')          {s+=date[i];}          else          {              ss<<s;              ss>>tempmonth;              ss.clear();              s="";          }      }      ss<<s;      ss>>tempday;      ss.clear();      Date tempdate(tempyear,tempmonth,tempday);      return tempdate；//返回Date类型的日期  }  vector<string> Account::recordvector;  vector<double> Account::amountvector;  void Account::query(Date date1,Date date2)//按照日期顺序查询  {  stringstream ss;  int tempdate1=date1.year\*10000+date1.month\*100+date1.day;  int tempdate2=date2.year\*10000+date2.month\*100+date2.day;      int tempyear,tempmonth,tempday;//将日期转为int类型直接比较数字大小从而比较日期大小      for(int i=0;i<recordvector.size();++i)      {          string s="";          ss<<recordvector[i].substr(0,4);          ss>>tempyear;//得到该条记录的年份          ss.clear();int J=5;          for(int j=5;j<recordvector[i].length();++j)         {              if(recordvector[i][j]!='-')              {s+=recordvector[i][j];J++;}              else              {                  ss<<s;                  ss>>tempmonth;                  ss.clear();s="";                  break;              }          }//该循环是为了得到该条记录的月份          for(int j=J+1;j<recordvector[i].length();++j)          {              if(recordvector[i][j]!=' ')              s+=recordvector[i][j];              else              break;          }          ss<<s;          ss>>tempday;//得到该条记录的日子          ss.clear();          int tempdate=tempyear\*10000+tempmonth\*100+tempday;//整合，得到该条记录的数字日期以便于日期比较          if(tempdate>=tempdate1 && tempdate<=tempdate2)//判断该条记录的日期是否在用户所要查询的一段日期内          {              cout<<recordvector[i]<<endl;//是则输出该条记录          }      }  }  void Account::queryByAmount(Date date1,Date date2)//按照交易金额查询  {      vector<string>queryvector;      vector<double>amount;  stringstream ss;  int tempdate1=date1.year\*10000+date1.month\*100+date1.day;  int tempdate2=date2.year\*10000+date2.month\*100+date2.day;      int tempyear,tempmonth,tempday;//将日期转为int类型直接比较数字大小从而比较日期大小      for(int i=0;i<recordvector.size();++i)      {          string s="";          ss<<recordvector[i].substr(0,4);          ss>>tempyear;//得到该条记录的年份          ss.clear();int J=5;          for(int j=5;j<recordvector[i].length();++j)         {              if(recordvector[i][j]!='-')              {s+=recordvector[i][j];J++;}              else              {                  ss<<s;                  ss>>tempmonth;                  ss.clear();s="";                  break;              }          }//该循环是为了得到该条记录的月份          for(int j=J+1;j<recordvector[i].length();++j)          {              if(recordvector[i][j]!=' ')              s+=recordvector[i][j];              else              break;          }          ss<<s;          ss>>tempday;//得到该条记录的日子          ss.clear();          int tempdate=tempyear\*10000+tempmonth\*100+tempday;//整合，得到该条记录的数字日期以便于日期比较          if(tempdate>=tempdate1 && tempdate<=tempdate2)//判断该条记录的日期是否在用户所要查询的一段日期内          {//是则进入，存放在两个容器中，容器中的数据一一对应              queryvector.push\_back(recordvector[i]);              amount.push\_back(amountvector[i]);          }      }      string tempquery;double tempamount;      for(int i=0;i<queryvector.size()-1;++i)//利用冒泡排序，将操作记录按照交易金额进行单调排序      {          for(int j=i;j<amount.size();++j)          {              if(amount[i]<amount[j])              {                  tempamount=amount[i];                  amount[i]=amount[j];                  amount[j]=tempamount;                  tempquery=queryvector[i];                  queryvector[i]=queryvector[j];                  queryvector[j]=tempquery;              }          }      }//排序完成      for(int i=0;i<queryvector.size();++i)      cout<<queryvector[i]<<endl;//输出记录  } |

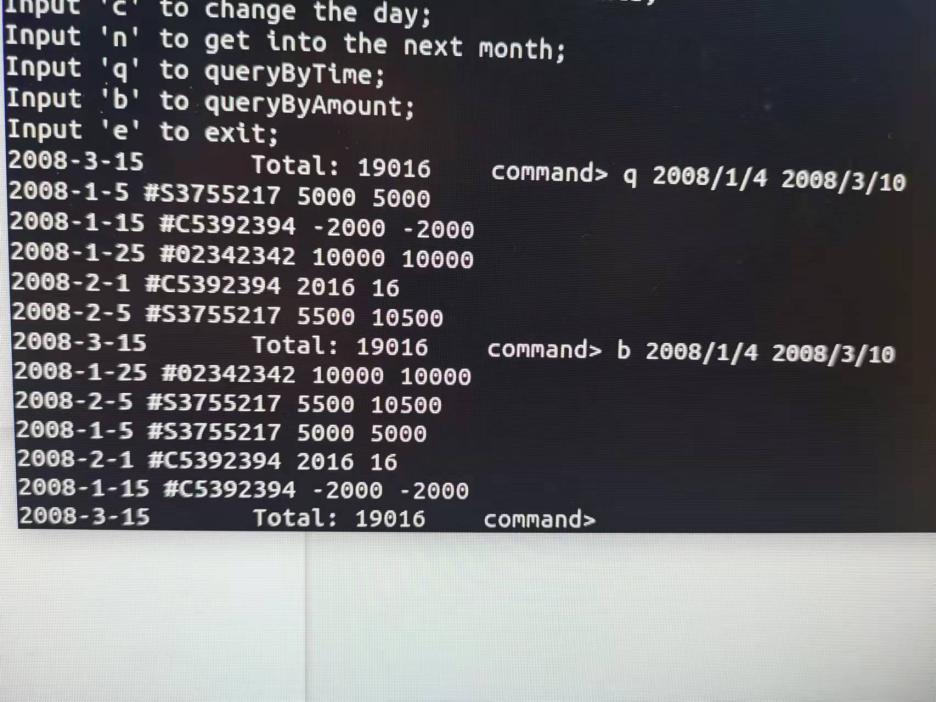
代码片段4 账目信息查询的关键代码

1. 测试

（1）测试用例1：q 2008/1/4 2008/3/10（按照时间顺序查询账目信息）

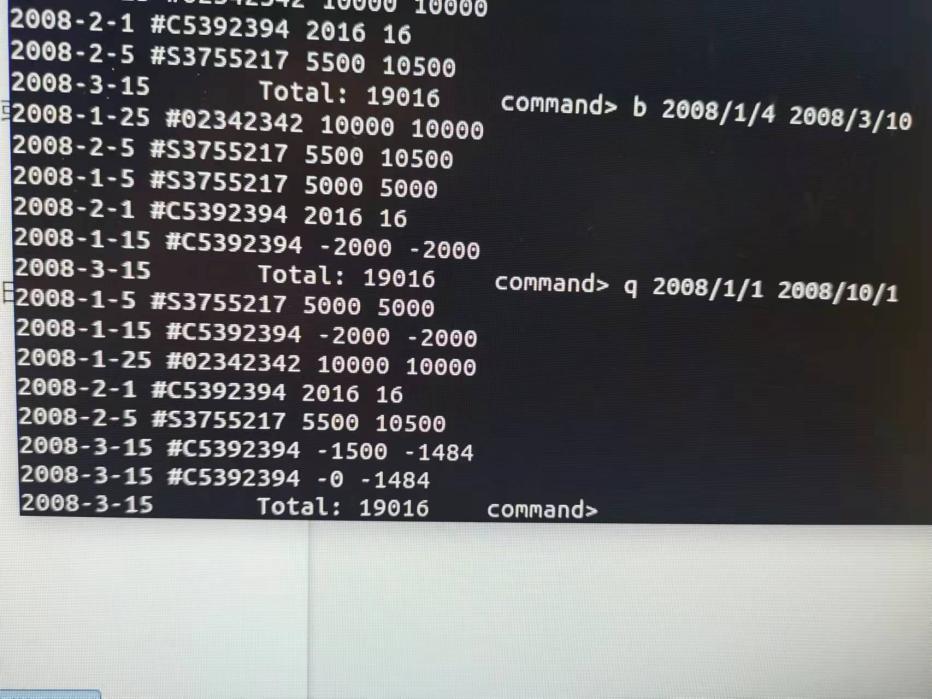
得到的运行截图如下：

（2）测试用例2：b 2008/1/4 2008/3/10（按照交易金额大小顺序查询账目信息）

得到的运行截图如下：

（3）测试用例3：q 2008/1/1 2008/10/1（按照时间顺序查询账目信息）

得到的运行截图如下：



1. 分析

加入账目信息查询函数，并且提供两种查询方式的目的是为了更进一步完善银行系统，使其更加贴近实际，更加真实。同时也在一定程度上给予了用户便利，使得用户可以按照自己的需求，按照自己想要的方式，查询自己想要查询的那一段日期内的账目信息。这三个测试说明两种查询方式均可以正常执行，两个功能基本完善，可以实现按照不同的方式查询账目信息。

**实验结论与感想：**

**实验结论：**该银行系统已经具有“系统”的初步形态，可以提供一些简单的UI操作界面，同时可以实现一些基本的，简单的功能，算是一个“银行系统”的雏形。

**实验感想：**整个银行系统的编写时间大概在一周以内，其基本架构可以快速地在几天内确立，随后基本上在一周内可以完成对各个bug的修复以及细节方面的完善。通过此次银行系统的作业，我对c++的类与对象有了更深一步的体会与理解，代码功底以及逻辑思维能力有了更进一步的提升。同时，在编写银行系统的过程中，我对编写一个小型的项目的步骤和条理有了更加清晰的认知，是一个感触极深，且确实学到众多知识的一个作业。

在编写的过程中经常性地会遇到很多从来都没有遇见过的问题，每一次都会使我陷入深思，然后在各大网络平台上寻找解决方法，求助于老师与同学。在这个过程中积累了较多的经验，也学习到了很多从不知晓的知识。

但这个作业也暴露了我的一些问题，比如代码风格，代码规范性以及思考方式等等。这个银行系统我并未按照老师给出的方案进行编写，而是根据自己的第一反应进行编写，从而导致后来的代码写的有些过于复杂，非常的不简洁，不规范，不具有美感。因此，通过这次作业，我也认识到自己应该多去看看规范的代码编写，同时也应该思考，如何在看到题目需求的时候第一感觉不要想偏？这些问题都值得我在接下来的学习中继续探索。