CSCE 113 Project 1

Project Description

This project consisted of designing a bank teller system, allowing a user to do many of the functions you could do if you went to a bank. The user could create an account, close an account, withdraw/deposit money, or check their account information, including the past 10 transactions. If they were given an administrative password, they could also check the total amount deposited in the bank, the total amount of accounts and the average balance per customer.

In order for a customer to create an account, they would first choose the on-screen option. Then the system would ask for their first and last name and would randomly generate an account and PIN number for this customer. They must also have a minimum deposit of $1000 in order to create a new account.To close an account, the user would select the onscreen option. After confirming their credentials, the remaining balance would be withdrawn (up $500) and the account would be deleted. If a customer wants to deposit some money, the system would ask for user credentials and how much they wanted to deposit and then add that amount to their total balance. If a customer wanted to withdraw some money, the system would ask for credentials and how much they wanted to withdraw. It would also warn the customer of a $10 maintenance fee if their account balance fell below $1000 and if their account balance fell below $50, it would close their account.

Project Purpose

The purpose of this assignment is to demonstrate understanding of making classes in C++ as well as combining multiple header and implementation files to create a finished working program.

Program Organization/Description of Classes

The three classes we used were all put into their own header file and were named Customer.h, History.h and Bank.h, and were implemented in their respective .cpp files.

Bank class - Manages data between users. This class also holds a vector of all accounts in the bank.

Customer class - handles data for an individual customer, handles the withdraw/deposit amounts and transactions.

History class - A simple data structure that holds a single transaction event.

The Bank class relies on the Customer class because in order for the Bank to read in from the file, it needs to know how the customers are formatted. It also needs to access the customer's information in order to check the PIN/account numbers. The Customer class relies on the History class because the transaction history is a part of the customer class.

Compile/Run Instructions

In order to compile our project, using the terminal navigate to the project directory, then type the following:

```
g++ project1.cpp Bank.cpp History.cpp Customer.cpp -std=c++11 -o a
```

To run the program type:

```
./a
```

Input/Output Specifications

When the program first starts, it takes input from a file on the hard disk called “data.txt”. This file stores all account data for the bank. The data is read into a vector of “Customer” which is then stored in memory. Functions in the program can access data in the vector of Customers to perform banking actions. If account data is ever changed, the program will rewrite to the data.txt file. When the program is closed, the program will save all account data before exiting. The only other type of input that the program takes is user input from a keyboard. The user can navigate menus, make deposits, withdraws, create or close accounts, and view account information using the keyboard. There are 2 main instances where the program can behave unexpectedly. First, when there is no data.txt file, or if it is formatted incorrectly, the program may not run as intended. Second, if when creating an account, the user enters more than two names (a first and last name), the program will not create an account. To navigate, the user simply has to enter a number corresponding to the menu item number. The format of the data.txt file for a single customer is the following:

```
[ customer first_name last name account account_number
PIN pin_number
balance amount_number
transactions { (date amount) ... }
]
```

The program will catch unexpected user inputs and return to the main menu. Every time a user can input data, there is a function called that tests whether the input data is of the type that the program expects. For example, if the program asks for an amount to withdraw from the account, the program will test the input data to make sure that the user input an integer or double.

Logical Exceptions

There are no known logical exceptions in this program.

Object Oriented/Generic Programming Features

Because both Bank and History classes require the formatting of the Customer class, they can be considered an inheritance of the Customer class. We also used polymorphism in both the Customer.cpp and History.cpp files. We also made use of the time.h for the History.cpp file. In order to keep track of all the different customer accounts, we decided to put them in a vector rather than a linked list.

Tests

Correct user input in creating an account. After completion, the program returns to the main menu.

```
------- Create An Account -------

Enter your first and last name.
first last
Please enter initial deposit amount. (no dollar signs)
1000

account created

-------------
Account Number: 486276
PIN Number: 7806
-------------
```

Incorrect user input in creating an account. After the error code, the program returns back to the main menu.

```
------- Create An Account -------

Enter your first and last name.
test name
Please enter initial deposit amount. (no dollar signs)
a
error: invalid input
```

This error handling design is used in all points in the program where a user can input numerical data. There is no checking of user input on string data due to the numerous special cases on checking input. Since a string can hold any character type, there is no risk of a user crashing the program through invalid string inputs.

When trying to access account data, if the user inputs a valid account number and it's corresponding PIN number, they will be granted access to whatever account action they desire (they could be trying to access functions such as closing an account, depositing or withdrawing money, view account balance, or view recent transactions). After the function completes, the user will be returned to the main menu.

```
------- Account Balance -------

Enter your account number.
239716
Enter your PIN.
8922

-- ACCOUNT INFORMATION: --
Name:           Albert Einstein
Account Number: 239716
PIN:            8922
Balance:        6000.00

--  TRANSACTION HISTORY:  --

Date: 2015-03-13
Amount: 1000.00

Date: 2015-03-13
Amount: -345.00

Date: 2015-03-13
Amount: 234.00

Date: 2015-03-13
Amount: -23511.00

Date: 2015-03-13
Amount: 100000.00
```

```
Date: 2015-03-13
Amount: 2.00

Date: 2015-03-13
Amount: 1.00

Date: 2015-03-13
Amount: -5.00

Date: 2015-03-13
Amount: 14.00

Date: 2015-03-13
Amount: -6.00
```

When trying to access account data, if the user inputs an invalid account number or pin number, they will be notified which value was unacceptable and the user will be returned back to the main menu. Access to administrator functions is also handled in a similar manner.

```
------- Close An Account -------

Are you sure you want to close your account?
1: yes   2: no
1
Enter your account number.
14
Enter your PIN.
5
Invalid Account Number.
```

```
/*****************************
  Project #1 CSCE 113-502
      Names: Donohue, Brandon     Loyka, Kyle
      Login:  bdonohue             kloyka

*****************************/
/*
      This project entails designing a program that works like
a bank teller. Customers can add/delete accounts, withdraw/deposit
money or look up previous transactions. There is also an admin function
to look at the total amount in the bank and the list of current
customers.
*/

#include "Customer.h"
#include "Bank.h"
#include "History.h"
#include <iostream>
#include <string>
#include <stdexcept>

using namespace std;

void welcome();
void menu_new_account(Bank& b);
void menu_close_account(Bank& b);
void menu_withdraw(Bank& b);
void menu_deposit(Bank& b);
void menu_account_balance(Bank& b);
void menu_transaction_history(Bank& b);
void menu_statistics(Bank& b);
void checkInput();

int main() {
      int menu;
      Bank b;
      b.readData();
      while (1){
            try{
                  welcome();
                  cin >> menu;
                  checkInput();

                  if(menu == 1)
menu_new_account(b);                              // open new account
                  if(menu == 2)
menu_close_account(b);                     // close account
```

```
                  if(menu == 3) menu_withdraw(b);
            // withdraw from account
                  if(menu == 4) menu_deposit(b);
            // deposit to account
                  if(menu == 5) menu_account_balance(b);                //
display customer's account balance
                  if(menu == 6) menu_transaction_history(b);      // display
customer's transactions
                  if(menu == 7)
menu_statistics(b);                              // admin bank statistics
                  if(menu == 8) break;
                              // quits program
            }

            catch (exception& e) {
                  cerr << "error: " << e.what() << '\n';
                  cin.clear();
            }
      }
      b.writeData();
      return 0;
}



void welcome(){
      cout << "\n\n*****************************\n";
      cout << "*            MENU           *\n";
      cout << "*****************************\n";
      cout<<endl<<endl;
      cout << "This is your Piggy Bank. Welcome valued customer." << endl
            << "Please select one of these options:" << endl
            << "1: Open a new account\n"
            << "2: Close an existing account\n"
            << "3: Withdraw\n"
            << "4: Deposit\n"
            << "5: Account Balance\n"
            << "6: Last 10 transactions\n"
            << "7: Statistical Info\n"
            << "8: quit\n"<< endl;
}

void menu_new_account(Bank& b){
      cout << "\033[2J\033[1;1H";
      cout << "\n------- Create An Account -------\n\n";

      Customer user;
      string name1, name2;
```

```
	double balance;
	cout << "Enter your first and last name. " << endl;
	cin >> name1 >> name2;
	cout << "Please enter initial deposit amount. (no dollar signs)"<< endl;
	cin >> balance;
	checkInput();
	if (balance < 1000 ) {
		cout << "Minimum deposit of atleast $1000 required. Account was
not created."<<endl;
		return;
	}

	int pin = randomPIN();
	int acc = randomACC(b);

	user.newData(name1 + " " + name2, acc, pin, balance);

	b.new_account(user);
	cout <<"\naccount created\n\n";
	cout<<"-------------\n";
	cout <<"Account Number: "<< acc <<endl;
	cout <<"PIN Number: "<< pin<< endl;
	cout<<"-------------\n";
	b.writeData();


}

void menu_close_account(Bank& b){
	cout << "\033[2J\033[1;1H";
	cout << "\n------- Close An Account -------\n\n";
	int choice = 0;
	int acc = 0;
	int pin = 0;

	while(1){
	 cout << "Are you sure you want to close your account?" << endl
	 << "1: yes   2: no" << endl;
	 cin >> choice;
	 checkInput();

	 if(choice == 1) {
		cout << "Enter your account number." << endl;
		cin >> acc;
		cout << "Enter your PIN." << endl;
		cin >> pin;
		checkInput();
```

```
            Customer user;
            if ( b.check_acc(acc) ){
                        user = b.search(acc);
                  }

            else{
                        cout <<"Invalid Account Number."<<endl;
                        return;
                  }

            if (user.pin() == pin ){
                        double ddd = user.balance();
                        b.close_account( user.acc_num() );
                        if (ddd < 500) cout << "$"<< ddd << " withdrawn";
                        if (ddd > 500) cout << "$500 MAX withdrawn, please see
cashier";
                        cout << "\n\nAccount Closed.\n\n";
                        b.writeData();
                        return;
            }
            else {
                  cout << "Invalid PIN.\n";
                  return;
                  }
           } // if (choice == 1)

           else if (choice == 2) {
                  cout << "\n\nCancelled.\n\n";
                  return;
           } // else if (choice == 2)

           else{
                  cout<<"Invalid Entry.\n";
           }

      } // while(1)

} // menu_close_account

void menu_withdraw(Bank& b){
      cout << "\033[2J\033[1;1H";
      cout << "\n------- Withdraw -------\n\n";
      Customer user;
      int pin, acc;

      cout << "Notice: A $10 maintenance fee will be applied if your account
balance falls below $1000." << endl;
      cout << "Max withdraw of $500:\n\n";
```

```
	cout << "Enter your account number." << endl;
	cin >> acc;
	checkInput();
	cout << "Enter your PIN." << endl;
	cin >> pin;
	checkInput();
	if ( b.check_acc(acc) ){
		user = b.search(acc);
	}

	else{
		cout <<"Invalid Account Number."<<endl;
		return;
	}

	if (user.pin() == pin ){
		double amount = 0;
		cout << "Please enter the amount of money you would like to
withdraw (no dollar sign)\n";
		cin >> amount;
		checkInput();

		if ( -500 < amount && amount < 500 ){
			user.withdraw(amount);
			b.rewriteAccount(acc, user);
			b.writeData();
			cout << "Withdraw Complete.\n";
			if (user.balance() < 1000 ){
				cout<< "Account balance is below $1000. $10 fee was
charged.\n";
				user.withdraw(10);
				b.rewriteAccount(acc, user);
				b.writeData();
				}
			if (user.balance() < 50 ){
				cout << "Account balance is below minimum value.
Account is closed.\n";
				b.close_account( user.acc_num() );
				b.writeData();
			}
			return;
		}
		else {
			cout << "Invalid withdraw amount. Please see cashier.\n";
			return;
		}
	}
```

```
	else {
		cout << "Invalid PIN.\n";
		return;
	}
}

void menu_deposit(Bank& b){
	cout << "\033[2J\033[1;1H";
	cout << "\n------- Deposit -------\n\n";
	Customer user;
	int pin, acc;

	cout << "Enter your account number." << endl;
	cin >> acc;
	checkInput();
	cout << "Enter your PIN." << endl;
	cin >> pin;
	checkInput();
	if ( b.check_acc(acc) ){
		user = b.search(acc);
	}

	else{
		cout <<"Invalid Account Number."<<endl;
		return;
	}

	if (user.pin() == pin ){
		double amount = 0;
		cout << "Please enter the amount of money you would like to
deposit (no dollar sign)\n";
		cin >> amount;
		checkInput();
		if (amount < 0){
			cout<<"Invalid deposit.\n";
			return;
		}
		user.deposit(amount);
		b.rewriteAccount(acc, user);
		b.writeData();
		cout << "Deposit Complete.\n";
		return;
		}

	else {
		cout << "Invalid PIN.\n";
		return;
```

```
	}
	b.writeData();
}

void menu_account_balance(Bank& b){
	cout << "\033[2J\033[1;1H";
	cout << "\n------- Account Balance -------\n\n";
	Customer user;
	int pin, acc;
	cout << "Enter your account number." << endl;
	cin >> acc;
	checkInput();
	cout << "Enter your PIN." << endl;
	cin >> pin;
	checkInput();

	if ( b.check_acc(acc) ){
		user = b.search(acc);
	}
	else{
		cout <<"Invalid Account Number."<<endl;
		return;
	}

	if (user.pin() == pin ){
		print(cout, user);
	}
	else {
		cout << "Invalid PIN.\n";
		return;
	}
}

void menu_transaction_history(Bank& b){
	cout << "\033[2J\033[1;1H";
	cout << "\n------- Transaction History -------\n\n";
	Customer user;
	int pin,acc;
	cout << "Enter your account number." << endl;
	cin >> acc;
	checkInput();
	cout << "Enter your PIN." << endl;
	cin >> pin;
	checkInput();

	if ( b.check_acc(acc) ){
		user = b.search(acc);
	}
```

```
	else{
		cout <<"Invalid Account Number."<<endl;
		return;
	}

	if (user.pin() == pin ){
		user.printTransaction();
	}
	else {
		cout << "Invalid PIN.\n";
		return;
	}
	b.writeData();
}

void menu_statistics(Bank& b){
	cout << “\033[2J\033[1;1H"; // this clears the terminal window
	cout << "\n------- Admin Access -------\n\n";
	const int admin = 123;  // admin password
	int pass;
		cout << "Enter your admin password. (admin password: 123 )"
		     << endl;
		cin >> pass;
		checkInput();

		if(pass == admin){
			int choice2;

			cout << "Please choose an option:" << endl
				<< "1: Total number of customers\t"
				<< "2: Total amount of deposits\t"
				<< "3: Average current balance" << endl;
			cin >> choice2;
			checkInput();

			if(choice2 == 1){
				cout<< "Number of Customers: "
				    << b.numberOfCustomers() << endl;
			}
			if(choice2 == 2){
				cout<< "Total Deposits: " << b.totalDeposits() <<
endl;
			}
			if(choice2 == 3){
				cout << "Average Balance: " << b.avgBalance() << endl;
			}
		}
		else cout << "Incorrect Password.\n";
```

```
            b.writeData();
}

void checkInput(){
      if (cin.fail() ) {
            cin.clear();
            cin.ignore(1);
            throw invalid_argument("invalid input");
      }
}
```

```
// Kyle Loyka
// CSCE 113
// Project 1

// History.h

#ifndef HISTORY_H
#define HISTORY_H

#include <iostream>
#include <string>
using namespace std;

class History {
public:
      History(string ss, double dd):s(ss), d(dd){}
      History():
        s("--"),        //default constructor
        d(0){}

      string date() {return s;}   // basic functions to access private members
      double amount() {return d;}

      void newData(double dd);
      void loadData(string ss, double dd);

private:
      string s;          // date
      double d;          // transaction amount
};

ostream& operator<<(ostream& os, History& h);
istream& operator>>(istream& is, History& h);

string current_time();


#endif
```

```
// Kyle Loyka
// CSCE 113
// Project 1

// History.cpp

#include <string>
#include <ctime>
#include <iostream>
#include "History.h"


using namespace std;

// output overloading
ostream& operator<<(ostream& os, History& h){
  return os<<"Date: "<< h.date()<<"\n"
           <<"Amount: "<< h.amount()<<"\n";
}

// input overloading
istream& operator>>(istream& is, History& h){
  //temp objects
  string a,b,c,d;
  int n;

  //take input
  is >> a >> b >> c >> d >> n;

  //checking inputs
  if(!is) return is;
  if(a !="Date:" || d !="Amount:" ){
    is.clear(ios_base::failbit);
    return is;
  }

  h.loadData(b+c,n);
  return is;

}

// use for loading (prevous) data into a History object
void History::loadData(string ss, double dd){
  this->s = ss;
  this->d = dd;
}

// use for new transactions
```

```
void History::newData(double dd){
  this->s = current_time();
  this->d = dd;
}

// finds current system time
//http://stackoverflow.com/questions/997946/how-to-get-current-time-and-date-
in-c
string current_time(){
  time_t raw;
  struct tm * timedata;
  char buffer[80];

  time (&raw);
  timedata = localtime(&raw);

  // strftime(buffer,80,"%d-%m-%Y %I:%M:%S",timeinfo); //this gives date and
time
  strftime(buffer,80,"%Y-%m-%d",timedata);              // this only gives
date
  string output(buffer);

  return output;
}
```

Customer.h

```
// Kyle Loyka
// CSCE 113
// Project 1
// Customer.h

#ifndef CUSTOMER_H
#define CUSTOMER_H
#include <iostream>
#include <string>
#include <vector>
#include "History.h"
using namespace std;

class Customer {
public:
      Customer(string ss, int aa, int pp, double bb):s(ss),
       a(aa), p(pp), b(bb) {}
      Customer():                          //default constructor
        s("Customer Name"),
        a(0),
        p(0),
        b(0){}
      string name() { return s;}           //some basic functions to
      int acc_num() { return a;}           // access private objects
      int pin() {return p;}
      double balance() {return b;}
      void printTransaction();
      void withdraw(double n);
      void deposit(double n);
      void transactions();
      void newData(string ss, int aa, int pp, double bb);
      void newTransaction(double dd);
      void outputCustomer(ostream& os);
      void loadTransaction(string ss, double dd, int ii);
      vector<History> transactionList();
private:
      string s; // name
      int a;    // account number
      int p;    // pin
      double b; // balance
      History transaction[10];  // array of transaction histories
};
ostream& operator<<(ostream& os, Customer& c);
istream& operator>>(istream& is, Customer& cc);

void print(ostream& os, Customer c);
void inputCustomer(istream& is, Customer& c);
#endif
```

```
// Kyle Loyka
// CSCE 113
// Project 1

// Customer.cpp

// need function in set_val to test if account number is used by
// another person, this will prevent 2 people from having the same acccount #
// should probably include this function in the "bank" class.

#include "Customer.h"
#include "History.h"
#include <cmath>
#include <string>
#include <vector>
#include <iomanip>
#include <fstream>

using namespace std;

// output operator overloading
ostream& operator<<(ostream& os,Customer& d){
  if (d.name() == "Customer Name") return os<<"";
  return os<<"Name:           "<< d.name()<<"\n"
           <<"Account Number: "<<setfill('0')<<setw(6)<< d.acc_num()<<"\n"
           <<"PIN:            "<<setfill('0')<<setw(4)<<d.pin()<<'\n'
           <<"Balance:        "<<setprecision(2)<<fixed<< d.balance()<<'\n';
}

// input operator overloading
istream& operator>>(istream& is, Customer& cc){
  //make some temp variables
  string s;
  int i,p;
  double b;
  string ch1, ch2, ch3, ch4;

  //take input
  is>> ch1 >> s >> ch2 >> i >> ch3 >> p >> ch4 >> b;

  //checking inputs
  if(!is) return is;
  if(ch1 !="customer" || ch2 != "account" || ch3 !="PIN" || ch4 !="balance"){
    is.clear(ios_base::failbit);
    return is;
  }

  //putting user inputs into a "Customer"
```

```
  cc.newData(s,i,p,b);
  return is;

}

// function to output Customer data in an elegant format
void Customer::outputCustomer(ostream& os){
  if (this->name() == "Customer Name") return;
  os << "[ ";
  os << "customer " << this->name()<<'\n';
  os << "account "  <<setfill('0')<<setw(6)<< this->acc_num()<<'\n';
  os << "PIN " <<setfill('0')<<setw(4)<<  this->pin()<<'\n';
  os << "balance " << setprecision(2)<<fixed<< this->balance()<<'\n';
  os << "transactions { ";
  for (int i = 0; i<10; ++i){
    os <<"( "<< transaction[i].date() <<" "<< transaction[i].amount()<<" ) ";
  }
  os << "} " << '\n'<<"]"<<'\n';
}

// read helper function to load customer data from a file
void inputCustomer(istream& is, Customer& c){
  string s1,s2,s3,s4,s5,s6,s7,s8,s9,s10;
  string first_name, last_name;
  string d0[10];
  double j[10];
  char ch1, ch2, ch3, ch4;
  char ch5[20];
  int account_number, pin_number;
  double amount_number;



  is >> ch1 >> s1 >> first_name >> last_name >> s2 >> account_number >> s3
    >> pin_number >> s4 >> amount_number >> s5 >> ch2
    >> ch5[0] >> d0[0] >> j[0] >> ch5[1]
    >> ch5[2]>> d0[1] >> j[1]>> ch5[3]
    >> ch5[4]>> d0[2] >> j[2]>> ch5[5]
    >> ch5[6]>> d0[3] >> j[3]>> ch5[7]
    >> ch5[8]>> d0[4] >> j[4]>> ch5[9]
    >> ch5[10]>> d0[5] >> j[5]>> ch5[11]
    >> ch5[12]>> d0[6] >> j[6]>> ch5[13]
    >> ch5[14]>> d0[7] >> j[7]>> ch5[15]
    >> ch5[16]>> d0[8] >> j[8]>> ch5[17]
    >> ch5[18]>> d0[9] >> j[9]>> ch5[19]
    >> ch3 >> ch4;

  if(!is) return;
```

```
  if(ch1 !='['|| s1 != "customer" || s2 !="account" || s3 !="PIN" ||
      s4 !="balance" || s5!="transactions" || ch2!= '{' || ch3!='}' || ch4!
=']'){
    is.clear(ios_base::failbit);
    return;
  }

  c.newData(first_name+" "+last_name, account_number, pin_number,
amount_number);
  for (int i = 0; i < 10; ++i){
    c.loadTransaction( d0[i], j[i], i);
  }

}

// helper function to load transaction data from a file
void Customer::loadTransaction(string ss, double dd, int ii){
  transaction[ii].loadData(ss,dd);
}

// another print customer function. Just another formatting option....
// honestly a bit redundant
void print(ostream& os, Customer c){
  if (c.name() == "Customer Name") return;
  os << '\n';
  os << "-- ACCOUNT INFORMATION: --";
  os << '\n';
  os << c << endl;
  os << "--  TRANSACTION HISTORY:  --"<< '\n'<<'\n';
  c.printTransaction();
}

// withdraw from account
void Customer::withdraw(double m){
  this->b = this->b - abs(m);
  this->newTransaction(-abs(m) );
}

// deposit into account
void Customer::deposit(double m){
  this->b = this->b + abs(m);
  this->newTransaction(abs(m) );
}

// for writing data to a NEW customer
void Customer::newData(string ss, int aa, int pp, double bb){
  this->s = ss;
  this->a = aa;
```

```
  this->p = pp;
  this->b = bb;
  this->newTransaction(bb);
}

// updating transaction history
void Customer::newTransaction(double dd){
  History temp;
  temp.newData(dd);
  transaction[9] = transaction[8];
  transaction[8] = transaction[7];
  transaction[7] = transaction[6];
  transaction[6] = transaction[5];
  transaction[5] = transaction[4];
  transaction[4] = transaction[3];
  transaction[3] = transaction[2];
  transaction[2] = transaction[1];
  transaction[1] = transaction[0];
  transaction[0] = temp;
}

// print only transaction history for a customer
void Customer::printTransaction() {
  for (int i = 0; i<10; ++i){
    cout<<transaction[i]<<'\n';
  }
}

// return a list of all transactions
vector<History> Customer::transactionList(){
  vector<History>h;
  for (int i = 0; i<10; ++i){
    h.push_back(this->transaction[i]);
  }
  return h;
}
```

```
// Kyle Loyka Brandon Donohue
// CSCE 113
// Project 1

// Bank.h

#ifndef BANK_H
#define BANK_H

#include <iostream>
#include <string>
#include <vector>
#include "Customer.h"

using namespace std;

class Bank {
      private:
            vector<Customer> accounts;

      public:
            Bank(vector<Customer> c) {    // copy constructor
                  accounts = c;
            }
            Bank():
      // default constructor
            accounts() {}

            void readData();
            void writeData();
            void close_account(int num);
            bool check_acc(int ii);

            double totalDeposits();
            double avgBalance();
            void new_account(Customer c);
            void rewriteAccount(int num, Customer& c);
            Customer search(int num);
            int numberOfCustomers();
};

int randomPIN();
int randomACC(Bank& b);



#endif
```

```
// Kyle Loyka Brandon Donohue
// CSCE 113
// Project 1

// Bank.cpp

#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include "Customer.h"
#include "Bank.h"
#include "History.h"

using namespace std;

// creating new user account
void Bank::new_account(Customer c) {
      this->accounts.push_back(c);
}

// closing account, takes account number as an argument
void Bank::close_account(int num) {
      for(int n = 0; n < this->accounts.size(); ++n) {
            if(this->accounts[n].acc_num() == num)
                  this->accounts.erase( this->accounts.begin() + n );
      }
}

// random PIN number generator
int randomPIN(){
      int pin = rand() % 10000 + 999; // 4 digits long
      bool ready = false;
      while(!ready){                                                    //
sometimes rand() returns a number
            if (pin < 999) pin = pin*10;  //   that is not sutable, this will
            else if (pin > 9999) pin = pin/10;// manipulate it to be 4 digits
            else ready = true;
      }
      return pin;
}

// account number generator, ensures no duplicate account numbers
int randomACC(Bank& b){
      bool run = true;
      int acc=0;
      while(run){
      acc = rand() % 999999;  // 6 digits long
```

```
	run = b.check_acc(acc);
	}
	return acc;
}

// reading data from the the data file
void Bank::readData() {
	ifstream file;
	file.open ("data.txt");

	while(!file.eof() ){
	  Customer temp;
	  inputCustomer(file, temp);
	  this->accounts.push_back(temp);
	}// while

	file.close();
}

// writing data to the data file
void Bank::writeData() {
	ofstream file;
	//remove( "data.txt");
	file.open("data.txt", ofstream::out | ofstream::trunc);
	for (int i=0; i<this->accounts.size(); ++i ){
			this->accounts[i].outputCustomer(file);
	}

	file.close();
}

// overwrites an account's data with new data
void Bank::rewriteAccount(int num, Customer& c){
	for(int i=0; i < this->accounts.size(); ++i) {
		if(this->accounts[i].acc_num() == num){
				this->accounts[i] = c;
		}
	}
}

// search for a Customer account given the account number
Customer Bank::search(int num) {
		for(int i=0; i < this->accounts.size(); ++i) {
			if(this->accounts[i].acc_num() == num) return this-
>accounts[i];
		}
		Customer c;
		return c; // just to remove warnings, input account number
```

```
                                          // should be clean before this so
this return is not necessary.
                                          // however, just in case, it will
return a blank Customer class
      }

// computes total number of customers in Bank
int Bank::numberOfCustomers(){
      return this->accounts.size()-1;
}

// check to see if account number & account exists
bool Bank::check_acc(int ii) {
      for(int i=0; i < this->accounts.size(); ++i){
            if(this->accounts[i].acc_num() == ii) return true;
      } //for
      return false;
}//check_acc

// computes total number of deposits in Bank
double Bank::totalDeposits(){
      double deposit = 0;

      for (int j = 0; j < this->accounts.size() ; ++j){
            Customer user = this->accounts[j];
            vector<History>temp;
            temp = user.transactionList();

            for (int i = 0; i<10; ++i){
                  if (temp[i].amount() > 0) deposit = deposit +
temp[i].amount();
            }
      }
      return deposit;
}

// computes average balance of all bank customers
double Bank::avgBalance(){
      double balance = 0;

      for (int j = 0; j < this->accounts.size() ; ++j){
            balance = balance + this->accounts[j].balance();
      }
      balance = balance / this->numberOfCustomers();
      return balance;
}
```