This assignment is designed to have you use Object Oriented Design to design accounts with transaction processing at a bank.

There will be the following 3 account types:

- Savings Account
- Checking Account
- Interest Checking Account

Your program must create classes for all the account types including a base class.

All account types and the base class must use .h files for headers and .cpp for class implementations. The names of these files should be the names of the class used in the program.

The .h files and .cpp files must all implement guards.

The .h files may use inline defining for small function body code. An example is the setters and getters are usually 1 line of code in the body, so are candidates for using in lining to do the function body implementation in .h files. More complex function should be implemented in a .cpp file.

You must have a file named main.cpp that contains the main program.

The following description is of the required classes for the project.
You must use the class and file names stated in this assignment.

BankAccountsClass

BankAccountsClass.h

This is the top level base class for all bank account types.

Class Members

This has the following protected class members with the specified names.

accountIdStr   is a string variable that is an account identifier
accountOwnerStr is a string that has the account owners name associated with the account identifier
accountBalanceFl is a float that is the account balance

This class has the following public class member functions with the specified names.

Constructor

The following default constructor must be provided for this class.

```
BankAccountsClass() : accountIdStr(""), accountOwnerStr(""), accountBalanceFl(0) {}
```

Class Member Functions

All the public mutators and accessors for all the protected variables in this class:

```
const string& getAcountIdStr () const
void          setAccountIdStr(string accountIdStr)

const string& getAccountOwnerStr () const
void          setAccountOwnerStr (string accountOwnerStr)

float         getAccountBalanceFl() const
void          setAccountBalanceFl(float accountBalanceFl)
```

Other Class Member Functions

```
void          deposit(float depositAmountFl)
```

This function handles deposit additions to the protected accountBalanceFl variable

```
void          virtual display()
```

This function handles displaying the information for all account types.
It is a virtual function that must be overridden in each of the account types.

BankAccountsClass.cpp

This file contains the implementations for this class.

```
// Displays the information for the BankAccountsClass.
void BankAccountsClass::display() {}
```

SavingsAccountClass

This class is derived from public BankAccountClass and it represents savings accounts.

SavingsAccountClass.h

Class Members

This class has the following protected class members with the specified names.

savingsInterestRate is a static float that contains the annual interest rate for all savings accounts

This class has the following public class member functions with the specified names.

Constructor

The following default constructor must be provided for this class.

```
  SavingsAccountsClass(string accountIdStr, string accountOwnerStr, float accountBalanceFl)
  : BankAccountsClass(        accountIdStr,        accountOwnerStr,       accountBalanceFl) { }
```

The public mutators and accessors for all the protected variables in this class:

```
static void  setSavingsInterestRate(float savingsInterestRate) { }

static float getSavingsInterestRate() { }
```

Other Class Member Functions

```
bool cashCheck(float);
void display();
```

SavingsAccountClass.cpp

This file contains the implementations for this class.

CheckingAccountsClass.h

This class is derived from public BankAccountClass and it represents checking accounts.

Class Members

This class has the following protected class members with the specified names.

```
static float checkingMinimumFl;
static float chargePerCheckFl;
```

This class has the following public class member functions with the specified names.

Constructor

The following default constructor must be provided for this class.

```
CheckingAccountsClass(
               string accountIdStr, string accountNameStr, float accountBalanceFl)
  : BankAccountsClass(accountIdStr,        accountNameStr,       accountBalanceFl)
  {}
```

The public mutators and accessors for all the protected variables in this class:

```
static void  setCheckingMinimumFl(float checkingMinimumFl )
{}

static float getCheckingMinimumFl(void)
{}

static void  setChargePerCheckFl(float chargePerCheckFl)
{}

static float getChargePerCheckFl(void)
{}
```

Other Class Member Functions

```
bool cashCheck(float);
void display();
```

CheckingAccountsClass.cpp

This file contains the implementations for this class.

```
//CheckingAccountsClass statics

float CheckingAccountsClass::checkingMinimumFl = 0,
      CheckingAccountsClass::chargePerCheckFl  = 0;

/*
if there is not enough money to cash the check display a message that states the problem
then return a false
otherwise cash the check, deduct per check fee if below the minimum balance
then return a true
*/

bool CheckingAccountsClass::cashCheck(float checkAmountFl)
{}
```

InterestCheckingAccountsClass.h

This class is derived from public CheckingAccountsClass and it represents interest bearing checking accounts.

Class Members

This class has the following protected class members with the specified names.

static float interestCheckingRate is the annual interest rate earned when bal > minbal credited monthly
static float minimmunBalanceRequired is the minimum balance required to receive interest
static float monthlyFeeChargeNoMinimumBalance is the monthly fee (only charged if minimum balance not met)

This class has the following public class member functions with the specified names.

Constructor

The following default constructor must be provided for this class.

```
InterestCheckingAccountsClass(
                    string accountIdStr, string accountNameStr, float accountBalanceFl)
  : CheckingAccountsClass (accountIdStr,        accountNameStr,       accountBalanceFl) { }
```

The public mutators and accessors for all the protected variables in this class:

```
static void  setInterestCheckingRate(float interestCheckingRate){ }
static float getInterestCheckingRate(){ }

static void  setMinimmunBlanceRequired(float minimmunBalanceRequired){ }
static float getMinimmunBlanceRequired(){ }

static void  setMonthlyFeeChargeNoMinimumBalance(float monthlyFeeChargeNoMinimumBalance){ }
static float getMonthlyFeeChargeNoMinimumBalance(){ }
```

Other Class Member Functions

```
  void calculateInterest();
  void display();
```

InterestCheckingAccountsClass.cpp

This file contains the implementations for this class.

```
//InterestCheckingAccountsClass statics

float InterestCheckingAccountsClass::interestCheckingRate             = 0,
      InterestCheckingAccountsClass::minimmunBalanceRequired          = 0,
      InterestCheckingAccountsClass::monthlyFeeChargeNoMinimumBalance = 0;

bool CheckingAccountsClass::cashCheck(float checkAmountFl)
{ }

void InterestCheckingAccountsClass::calculateInterest()
{ }
void InterestCheckingAccountsClass::display()
{ }

// InterestCheckingAccountsClass statics
float InterestCheckingAccountsClass::interestCheckingRate             = 0,
float InterestCheckingAccountsClass::minimmunBalanceRequired          = 0,
float InterestCheckingAccountsClass::monthlyFeeChargeNoMinimumBalance = 0;
```

TransactionsClass.h

This class represents the type of transactions that can be done on the account.

Class Members

This class has the following private class members with the specified names.

```
string   accountIdentifcationStr  is the account that the transaction is to be performed on.
string   transactionIdStr         is the type of transaction to execute on the account
unsigned transactionArgumentUns   is the value to be used by the type of transactions
```

This class has the following public class member functions with the specified names.

Constructor

The following default constructor must be provided for this class.

```
TransactionsClass()
  : accountIdentifcationStr(""), transactionIdStr(""),transactionArgumentUns (0)
  {}
```

The public mutators and accessors for all the protected variables in this class:

```
const string& getIdentifcationStr() const { }
void          setIdentifcationStr(string standIdentifcationStr) { }

const string& getTransactionIdStr() const { }
void          setTransactionIdStr(string transactionIdStr) { }

unsigned      getTransactionArgumentUns() const { }
void          setTransactionArgumentUns(unsigned transactionArgumentUns) { }
```

Other Class Member Functions
None

TransactionsClass.cpp

This file is not necessary if the **TransactionsClass.h** in lines all its functions.

Files
The following specifies the input and output files for the assignment.

All the files, unless noted, are comma separated value (csv) text files.

You must use the file names specified in this section.

You can assume that all the files are correctly formatted and contain valid entries. You do not have to worry about blank lines in the file or invalid data.

InfoBankConfig.txt

This file is an input file.

Each line in this file is used to contain one of the values that are used to configure the bank account properties used in the accounts. There is one configuration value per line.

Line 1 : Savings Account Interest Rate
Line 2 : Checking Account Minimum Balance
Line 3 : Checking Account Charge Per Check
Line 4 : Interest Checking Interest Rate
Line 4 : Interest Checking Minimum Balance
Line 5 : Interest Checking Monthly Fee Charge

The following values must be in the file:
3.5
1000.00
0.25
2.50
2500.00
10.00

InfoBankAccounts.txt

This file is both an input and output file.

This file has the following information and format that represents the banking account types.

account Identification String, Account Owner String, Current Account Balance Float

Bank Account Identification String

The first character indicates the account type:

s – savings account
c – checking account
i – interest bearing checking account

File content examples:

s847300,Deepika Padukone,0
c974423,Yu Nan,19.16
i555559,Aubrey Graham,1.00

The file must contain at least 6 entries with at least each account type having 2 entries each.

InfoBankAccountsTransactions.txt

This file is an input file.

Each line in this file is used to contain a transaction that the bank will process.

This file has the following information and format that represents the banking transactions.

account Identification String, Account Transaction Identifier String, Account Transaction Amount String (If Necessary)

Bank Account Identification String

The first character indicates the account type:

s – savings account
c – checking account
i – interest bearing checking account

Account Transaction Identifier String

The following are the only valid Transaction Identifiers:

deposit
withdraw
check
interest

If the Account Transaction Amount String is interest then there is not any Account Transaction Amount Float.

The file must contain at least 16 entries with at least each account type having 4 entries each.

File content examples:

The following accounts would have to exist in the InfoBankAccounts.txt file.

s847300,deposit,1200
s847300,withdrawal,120
s847300,interest
c974423,deposit,1000
c974423,check,512.39
s742100,deposit,1500.63
s742100,withdraw,500.63
c473892,deposit,20000
c473892,check,857.99
i555559,deposit,23000
i555559,check,1683.27
i555559,interest
i987654,deposit,13200
i987654,check,11200
i987654,interest
x000001,deposit,1600.69

main.cpp

The main .cpp is provided. You cannot modify the code that already exists there.

To correctly code up the program, you must put code in the sections that have /*$$*/ that indicate where you must identify areas that require coding.

In some function there are dummy returns that have values so that the code structure will compile. These dummy returns are marked with /*$$*/ so that you know you must replace these returns with correct values.

You must upload all the required .h and .cpp files along with all the .txt file that are used by the program.

General Design

You must use the arrays specified and use new to create the dynamic arrays.

The best approach is to get a function in main and its associated class to work one at a time, then proceed to the next function and associated class.