CS 31 Worksheet 8

This worksheet is entirely **optional**, and meant for extra practice. Some problems will be more challenging than others and are designed to have you apply your knowledge beyond the examples presented in lecture, discussion or projects. All exams will be done on paper, so it is in your best interest to practice these problems by hand and not rely on a compiler.

Concepts: Operators

1. Consider the following class:

```
class bankAccount {
public:  // class member functions

//--constructors
    bankAccount();

    bankAccount(string initName, double initBalance);
    // post: A bankAccount with two arguments when called like this:
    //       bankAccount anAcct("Hall", 100.00);

//--modifiers

    void deposit(double depositAmount);
    // post: depositAmount is credited to this object's balance

    void withdraw(double withdrawalAmount);
    // post: withdrawalAmount is debited from this object's balance

//--accessors

    double balance() const;
    // post: return this account's current balance

    string name() const;
     // post return the account name

    void setName( string initName );
    // post updates the member variable my_name

private:
    string my_name;    // Uniquely identify an object
    double my_balance; // Store the current balance
   };
```

and its underlying implementation:

```
//--constructors

bankAccount::bankAccount()
{
  my_name = "?name?";
  my_balance = 0.0;
}

bankAccount::bankAccount(string initName, double initBalance)
{
  my_name = initName;
  my_balance = initBalance;
}

//--modifiers

void bankAccount::deposit(double depositAmount)
{
  my_balance = my_balance + depositAmount;
}

void bankAccount::withdraw(double withdrawalAmount)
{
  my_balance = my_balance - withdrawalAmount;
}

//--accessors

double bankAccount::balance() const
{
  return my_balance;
}

string bankAccount::name() const
{
  return my_name;
}

void bankAccount::setName( string initName )
{
  my_name = initName;
}
```

Define and implement the following operators so that the following pile of driver code will build, run and pass the various assert statements supplied below:

```
friend bankAccount operator + ( const bankAccount & left, const
bankAccount & right );

        bankAccount operator+ ( const bankAccount & left, const
        bankAccount & right ) {
```

```
                        bankAccount temp;
                        if (left.name() == right.name())
                        {
                                // only combine together accounts with the same
                    name…
                                temp = bankAccount ( left.name(),
                    left.balance() + right.balance() );

                        }
                        // if the names don't match, it might be better to
                    throw an exception…
                        return( temp );
                    }

          friend bankAccount operator - ( const bankAccount & left, const
bankAccount & right );

                    bankAccount operator- ( const bankAccount & left, const
                    bankAccount & right ) {
                        bankAccount temp;
                        if (left.name() == right.name())
                        {
                                // only combine together accounts with the same
                    name…
                                temp = bankAccount ( left.name(),
                    left.balance() - right.balance() );

                        }
                        // if the names don't match, it might be better to
                    throw an exception…
                        return( temp );
                    }


          friend bool   operator ==( const bankAccount & left, const
bankAccount & right );

                    bool operator== ( const bankAccount & left, const
                    bankAccount & right ) {
                        return( left.name() == right.name() && left.balance()
                    == right.balance() );
                    }


          friend std::ostream& operator <<( std::ostream& outs, const
bankAccount & b );


                    ostream& operator <<( ostream& outs,const bankAccount & b )
                    {
                        // prints a bankAccount by showing its name and its
                    balance
                        outs << "name = " << b.name( ) << " balance = $ " <<
                    b.balance() << endl;
                        return( outs );
```

```
        }

friend std::istream& operator >>( std::istream& ins, bankAccount & b
);


        istream& operator >>( istream& ins, bankAccount & b ) {
              // builds a new bankAcount by reading a string name
        and a double balance
              double d = 0;
              string s;
              ins >> s;
              ins >> d;
              b.setName( s );
              b.deposit( d );
              return( ins );
        }
```

Example:

```
bankAccount me( “Howard”, 100.00 );
bankAccount another( “Howard”, 50.00 );
bankAccount you( “You”, 100.00 );

bankAccount combined = me + another;
assert( combined.name() == “Howard” );
assert( combined.balance() == 150.00 );
if (me == you)
{
   asssert( false );
}
if (me == me)
{
   assert( true );
}

bankAccount less = me - another;
assert( combined.name() == “Howard” );
assert( combined.balance() == 50.00 );
cin >> combined;
cout << combined;
```