# Programming Project #9

## Assignment Overview

This assignment will give you experience with making your own class using maps, vectors, and algorithms. This is a bit more free-form as well as you have to write your own headers for the two classes.

This assignment is worth 50 points (5.0% of the course grade) and must be **completed and turned in before 11:59 on Monday, April 11th . That's two weeks because of the midterm.**

## Background

The Bank Of Evil (http://www.youtube.com/watch?v=SY94qvnJDdQ) wants to set up electronic bank records. Due to their supervillain clientele they want you to ensure only approved modifications are made to bank accounts. You need to make a `BankAccount` class which stores a client's `name`, `password_number`, and `current_funds`. You also need to make a `Bank` class, which stores a series of `BankAccounts` represented by your `Bank`, and each `BankAccount` in the `Bank` has a unique ID.

## Class Requirements

Create two classes. Note that `BankAccount` ***grants friendship status*** to the `Bank` class!

1. `BankAccount`, a class to hold the following fields, all of them ***private***!
   a. `name_` : a string, the client's name
   b. `password_` : a string, the client's password (only letters and numbers!)
   c. `funds_` : a double, the amount of money in the account.
2. `Bank`, a class to hold any number of `BankAccounts` where each `BankAccount` is indexed by its `account_num`. `Bank` has the following fields, all ***private***
   a. `name_` : a string, name of the bank
   b. `bank_id_` : a long, the id of the bank
   c. `accounts_` : a map of `BankAccounts (id` to `BankAccount)`
   d. `rand_eng_` : a default_random_engine.

`BankAccount`

- `default construct.` No parameters, use appropriate defaults values for each member variable**.** Don't rely on the compiler to set the defaults!
- 3-parameter `constructor`:
  - in order: `name (string), password (string), funds (double).`
  - all parameters required except `funds` which has a default of 0.
    - set to 0 if the amount is negative!
- overloaded `<< function.` This is a `friend` function that will print the `name` and the `funds` in an account (***not*** the password)

`Bank`

- 2-parameter `constructor`, a string that is the name of the bank and an integer seed for the random number generator. Seed has a default value 1234
  - a bank id is randomly generated and stored by the bank constructor. Its value is in the range (10000:99999).
- `bank_id`. member function. no parameters, returns bank_id (the id generated in the constructor).
- `bank_name`. member function. no parameters, returns bank_name
- `create_account.` member function. 3 args: `account_name (string), password (string), funds (double).` Returns a `long`, the randomly assigned account id (range 10000:99999).

    - an account number is automatically generated for the account and stored internally in the map `accounts_`. The number is 5 digits long (exactly) meaning the range is 10000-99999.
        - it is acceptable to have two accounts, two different `account_ids`, for the same account name.
        - no two accounts should have the same randomly assigned account id.
- `balance`. member function. 2 args: `long account_id, string password`. Returns the amount of money in the account if:
    - the `id` exists
    - the `password` is correct
    - upon error, returns `numeric_limits<double>::min()`, the smallest double (requires `#include<limits>`).
- `transfer.` member function. 5 args: `long from_id, string from_password, long to_id, string to_password, double amount`. Transfers the amount from the `from_id` account to the `to_id` account assuming:
    - both accounts exist
    - both passwords are correct
    - there is sufficient funds in the `from_id` account
    - the amount transferred is positive
    - returns Boolean, true if successful , false otherwise
- `print_account`. member function. 3 args: `long id, string password, ostream& out`. Prints to the passed stream the ID, the name and the funds in an account if:
    - account exists, if not prints `"No Such Account"`
    - password is correct, if not prints `"Bad Password"`
    - uses the overloaded `<< BankAccount` function.

## Deliverables

You must use handin to turn in the files: `proj09-bank.h, proj09-bank.cpp, proj09-bankaccount.h, proj09-bankaccount.cpp` these are your source code solution; **be sure to include your section, the date, the project number** and comments describing your code. Please be sure to use the specified file names, and save a copy of the files to your H drive as a backup.

## Other good information

1. You do not need to provide a main.cpp, but you need one to test your stuff.
    - there is a `main-09.cpp` in the Projects directory
2. If you need to put other members in your classes, data or function, feel free, but make sure that in so doing encapsulation is not broken. That is, the only access to bank information is through the provided/approved interface.
3. make a class a friend. In the granting class (in this case `BankAccount`) you place `friend class Bank`, saying the `Bank` has full access to `BankAccount` stuff. Remember, friend is a push not a pull
4. to check if something is in a map, a nice method is count. It counts the number of occurrences of a key. If it comes out 0, key isn't there and that is a wonderful result for a Boolean test.

## Example:

Using the example `main-09.cpp`, this is what I got for project output.

```
>./a.out
Bank Name : Bank of Evil
Bank ID   : 10869

Bad PASSWORD
No Such Account
ID:38587, Name:bill punch, funds:1000.00
ID:47528, Name:homer simpson, funds:0.00

Was it successful : true
Bill new amount   : 900.00
Homer new amount  : 100.00

Bad Balance Inqury
Not successful, insufficient funds, result was : false
Not successful, negative funds, result was     : false
```