**\*\*All changes from original document are highlighted\*\***

**Compile information:**

To compile, have all files in the same folder. Use the following command:

g++ EmployeeManager.cpp ClientManager.cpp AccountManager.cpp System.cpp -o system

And then ./system to run, like normal.

If you wish to see/test each individual file, the file headers have compile information. Be sure to uncomment the #define UNIT\_TESTING to run the test functions. You \*may\* also need to move test files out of the “test files” folder and into the same folder as the project.

Project 2 Design

1. Analysis (Use Cases)
2. System should display a login screen when idle. If a user wishes to log in, they must type the username and password for their account. If they do not match, then user cannot log in (and displays error that one is wrong).
3. If a system admin logs in, they should have more options than a normal branch member. They will have the ability to display, add, and delete branch members.
   1. To add a branch member, they must create the username/password.
   2. To delete branch members, they must enter the username, and confirm the deletion. If the username doesn’t exist, an error is given.
4. Any employee should have the option to change their password after logging in. They must first validate their current password, and then enter the new one.
5. Any employee should be able to access and manage client data.
   1. Client data consist of: name, address, social security number, employer, income.
   2. Accounts can only be added to existing clients. Accounts consist of the account number, the type, and the balance, and the clients name.
   3. Client info can be edited, except the name.
   4. Accounts can be managed. The user can withdraw or deposit.
6. All client info, account info, and employee info, are saved to pre-designated files.
7. After any change to staff, client, or account info, save to corresponding file.
8. Design

**EmployeeManager Class**: The class that handles basic employees and their functionality

* **Variables:**
  + **Employee struct:** contains username, password, and job level
  + **Private vector<Employee> employeeList[]:** list of employees
  + **Private string employeeFile:** file for employees
  + **Private static const string DEFAULT\_FILE:** The default file for staff data.
* **Functions**
  + **EmployeeManager(string filename):** creates an employee manager using the employees found in the file given.
  + **EmployeeManager():** creates an employee manager using default file
  + **Private void loadEmployees():** update employee list with employee file
  + **Private void saveEmployees():** save employee list in employee file.
  + **Public bool verifyLogin(string username, string password):** verifies login information
  + **Public bool changePassword(string username, string newPass):** Verifies password of employee, then changes to new password.
  + **Public bool isAdmin(string username):** verifies employee job level
  + **Public bool addEmployee(string username, string password, bool isAdmin):** Creates an employee. Returns true if it works, false if employee exists.
  + **Public bool deleteEmployee(string username):** Deletes an employee. Returns true if successful, false if employee does not exist.
  + **Public void displayEmployees():** Displays employee list.
* **Class relation:** This class does not rely on any other classes. It is used by the System class to create and use employees.

**ClientManager Class**: The class that handles clients and their functionalities

* **Variables:**
  + **Client struct:** contains name, address, social security number, employer.
  + **Private vector<Client> clientList[]:** list of clients
  + **Private string clientFile:** file for clients
  + **Private static const string DEFAULT\_FILE:** the default file for client data
* **Functions**
  + **clientManager(string filename):** creates a client manager using the client found in the file given.
  + **clientManager():** creates a client manager using default file
  + **Private void loadClients():** update client list with client file
  + **Private void saveClients():** save client list in client file.
  + **Public bool addClient(string nameIn, string addressIn, string employerIn, string ssIn, string incomeIn):** Creates a client. Returns true if it works, false if client exists already.
  + **Public bool editClient(string nameIn, string addressIn, string employerIn, string ssIn, string incomeIn):** Finds client and displays prompts to edit client information. Returns true if successful, false if client does not exist.
  + **Public bool displayClient(string nameIn):** displays client information. Returns false if client does not exist.
  + **Public bool isClient(string nameIn):** Checks if nameIn is a client.
* **Class relation:** This class does not rely on any other classes. It is used by the System class to create and use clients

**AccountManager Class**: The class that handles accounts and their functionalities

* **Variables:**
  + **Account struct:** contains account number, client name, account type, and balance
  + **Private vector<Account> accountList[]:** list of accounts
  + **Private string accountFile:** file for accounts
  + **Private static const string DEFAULT\_FILE:** Default file for account information
* **Functions**
  + **accountManager(string filename):** creates an account manager using the accounts found in the file given.
  + **accountManager():** creates an account manager using default file
  + **Private void loadAccounts():** update account list with account file
  + **Private void saveAccounts():** save account list in account file.
  + **Public bool addAccount(int accNum, string clName, int accType, int bal):** Creates an account. Returns true if it works, false if account exists already.
  + **Public bool displayAccount(int accNum):** displays account information. Returns false if account does not exist.
  + **Public bool withdraw(int accNum, int withdrawAmount):** withdraws money from account. Returns false if account does not exist or if withdraw amount is invalid.
  + **Public bool deposit(int accNum, int depositAmount):** deposit money into account. Returns false if account does not exist or if deposit amount is invalid.
  + **Public bool isAccount(int accNum):** verifies account exists.
* **Class relation:** This class does not rely on any other classes. It is used by the System class to create and use accounts

**System Class:** Handles user input and overall system usage.

* **Variables:**
  + **Private EmployeeManager employeeManager:** Manages employees
  + **Private ClientManager clientManager:** Manages clients
  + **Private AccountManager accountManager:** Manages accounts
  + **Private bool idle:** true if system is idle, false if active
  + **Private int menuChoice:** holds the user’s menu choice.
  + **Private string currentUser:** holds the name of the user currently logged in.
* **Functions**
  + **Public void idleState():** When idle, display log in screen.
  + **Public void branchScreen():** Shows the screen for branch employees.
  + **Public void SysAdminScreen():** Shows the system admin screen
  + **Public void logout():** returns to log in screen.
  + **Public void clientManagement():** Displays client and account management screen and processes input.
  + **Public void manageAccount(int accountNumber):** Brings up withdraw/deposit screen for valid accounts.
  + **Public void changePassword():** Screen to allow employees to change passwords.
  + **Private void verifyInput(int numOfChoices):** used to verify input is valid for the menu displayed.

1. **Testing (See Project2\_test for results)**

Normal Usage:

* If wrong username/password is entered when logging in, system should display an error and return to the “idle” state.
* Branch Employees should never see the system admin screen. System admins should always see the system admin screen upon logging in.
* After adding an employee, an account, or a client, the data should immediately be saved into the corresponding file.
* Deleting an employee should result in the employee being removed from the employee list, and from the employee file.

Abnormal Usage:

* All input should be checked for valid input (integers need to be integers, etc)
* If a customer, account, or employee does not exist, appropriate errors must be displayed.
* If any file does not exist, it needs to be created.
* If the employee file is empty, it needs to be rewritten with a “default” admin.