**NANJING UNIVERSITY OF AERONAUTICS AND ASTRONAUTICS**

**Project Report**

**For**

**Fund Management System**

1. **Project Introduction:**
   1. **Purpose**

The purpose of our proposed system is to provide players within the financial business a simple, transparent and straightforward general-purpose software to computerize the processes and management of their core working principals in a practical, clean and organized approach based on the values of timely service, reliable operation, paramount safety and security practices, and the continuous improvement to adapt along the rigors of the economic world.

* 1. **What problem does your software solve?**

The modern-day life of the current 21st century has evolved so substantially that it is easy to take for granted everything we currently have available on offer. No longer the dreary and painstaking endeavor mentality of years’ past, the world of today places an undeniable emphasis on convenience at the forefront of everything engraved in the daily lives of modern people. This shift of perception in attitude means that service-based systems have become increasingly important and major economies are ramping up their focus towards practicality-geared systems to compete.

The global economy and trade ramifications consequently brought along by accelerative technological improvements had major implications in the way people perceive and establish normative standards nowadays. Similarly, as the pace and workflow of the modern-day people increases, it necessitates the use of a practicality-based approach undertaken to the completion of even the most menial of tasks. Therefore, our proposed system is meant to bridge this gap of providing fast, accurate and reliable financial services wherever and whenever the customer of our consumer demands.

* 1. **Why are you developing this system?**

We are developing this system in order to fill in the gap in the operation scheme of that the former traditional financial model utilize by which the availability of their services is accessible only on-site. Through the use of a personalized financing application tied to their database providing services wherever and whenever on demand, it enables them to provide a starting point for faster-paced workflow interaction with their consumers and therefore provide better customer experience whilst appealing to a wider practicality-based audience.

1. **Problem Solution for Proposed System**

Our proposed system for this problem is a console application constructed to be able to perform in a multitude of desktop computing environments such as Windows PC, Mac, Linux and so on. Serving as a means for finances to start with their basic operating service, the software provides cash management functionality bundled with the release. This assortment of basic operating service provided includes the ability for respective finance account holders to access their accounts and manage their dividend balance. Included as well is the ability to, without exception, systematically catalogue all the registered accounts in the database of the respective corporation.

1. **Related System Analysis/Literature Review**

The related systems that are discussed below are selected based on domain similarity in which this project’s domain are

|  |  |  |
| --- | --- | --- |
| **Application Name** | **Weakness** | **Proposed Project Solution** |
| Bank Management System | Specialized as a means for solely banks to process funds and manage them in such a way that is not really optimized for use in other application environments and therefore not extensible | The proposed system will be designed and implemented in such a way that emphasizes modularity and code readability for easy customization by our respective users |
| Local Places | Doesn’t provide In-App interaction between consumers and service providers | The proposed system will target this problem by providing services In-App bundled from the release |

1. **Advantages/Benefits of Proposed System**

The advantages of the proposed system are listed below

* Our customers will be able to provide rapid service deployment
* Our customers will be able to increase their income through gaining competitive edge
* Our customers will be able to manage their transaction flows with increasing efficiency

1. **Scope**

The proposed system is a C++ program designed to be a fund management system to give small banks and/or small companies a cheaper, more reliable but simple way to organize and collect customers data and store it offline. Named Gucci bank its goal is to assist small time banks/companies with a startup foundation fund-credit system

* 1. **Consumers**

The proposed system will allow customers of the user to deposit money into the system and carry out transaction in the business, resulting in cashless easy transactions done with just telling the attendant you have an account followed by their name then entering the password on the device used to run the system.

* 1. **Service Provider**

The service provider will be the main operator of the system, with the only interaction their customer get with the system will be inputting the password whenever required. Mobile text message updates to the client’s customers will if any non-administration or fund and password related operations are done to their accounts with a one phone call name verification freeze asset call center available. This system allows the client more financial freedom than banks and credit cards, more security with the system being offline and mobile text updates whenever a user name and password is used, and ease of implementation into current work space setting. It would however be the client’s responsibility to access customers who would like to register to the system if they are eligible for a credit account or a debit account using available means i.e. credit score minimum requirements.

1. **Modules**

The following are the core modules of the proposed system

* 1. **Module 1: User Welcome Module**

This Module handles the account creation and verification process for the client’s customers, those functions being sign up, sign in, forgot account and exit (to exit and close the system).

* 1. **Module 2: Main Menu module**

This module handles the main operations of the system, the first view would be of the users account information. The main reason for this is to allow the attendant accessing the system to be able to advice the client’s customers on available balance and their available actions. The functions under this module are: Deposit amount, withdraw amount, balance enquiry, all account holder list, close an account, modify an account and log off.

* 1. **Module 3: Admin Module**

The admin has special administration functions that even the attendant does not have access to. To help the company better organize their funds and have an understanding of how much of their funds can be liquidated, for this the admin module can view all account holder’s information such as their name and current balance.

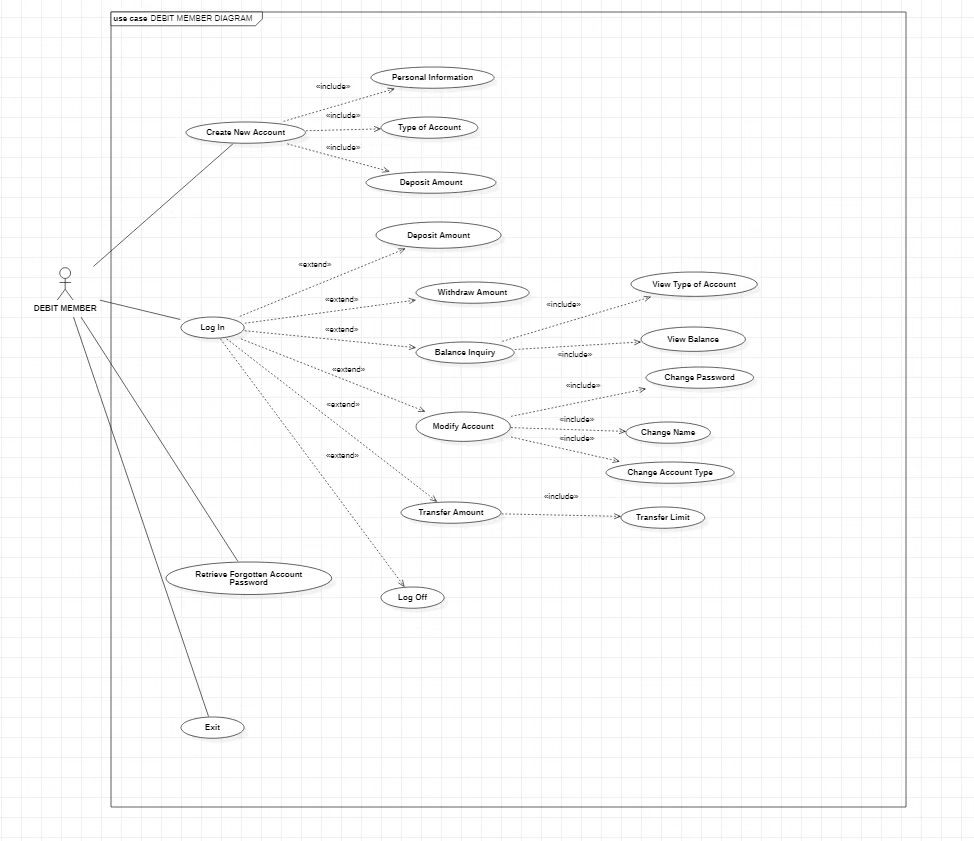
1. **Tools and Technologies**

The following tools along with the rationale are mentioned below will be used during the time line of our project.

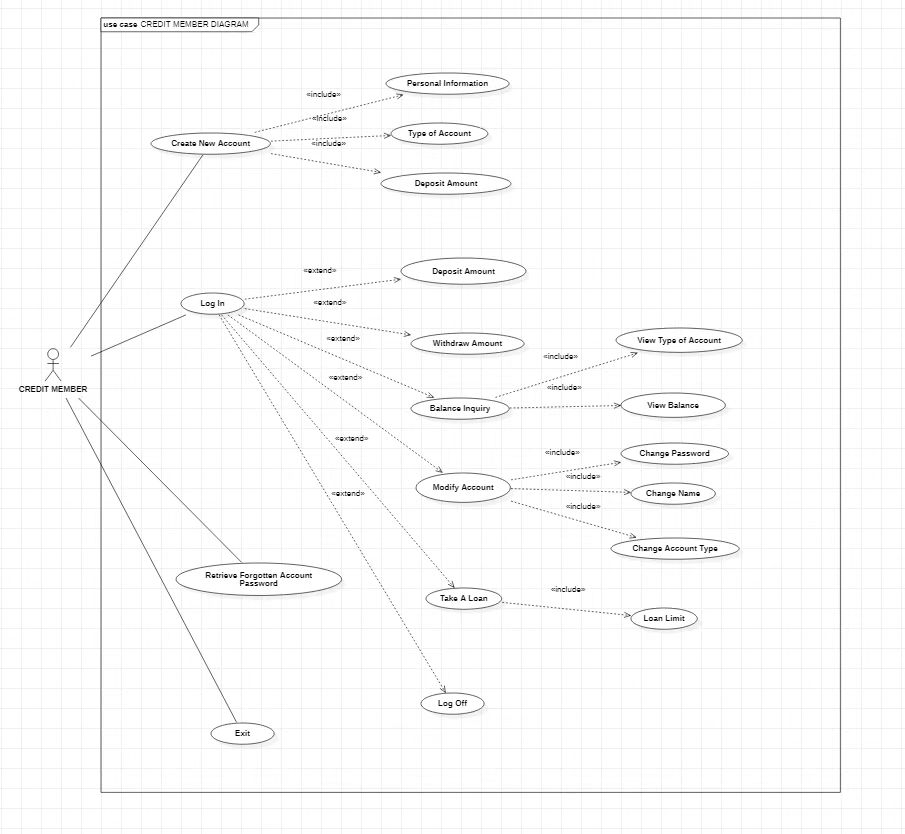
|  |  |  |  |
| --- | --- | --- | --- |
| **Tools and Technologies** | **Tools** | **Version** | **Rationale** |
| Ms. Word | 2016 | Documentation |
| Ms. Excel | 2016 | Documentation |
| DEV C++ | 5.11 | IDE |
| Star UML | - | UML diagrams |
| **Technology** | **Version** | **Rationale** |
| C++ | 2015 | Programming language |

**USE CASE DIAGRAMS:**

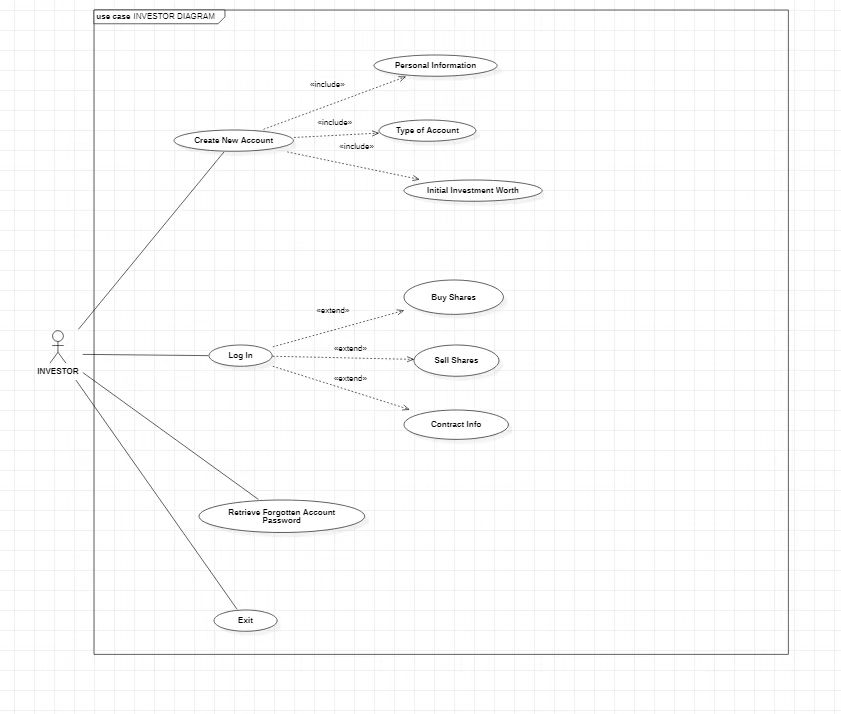
**Debit members**



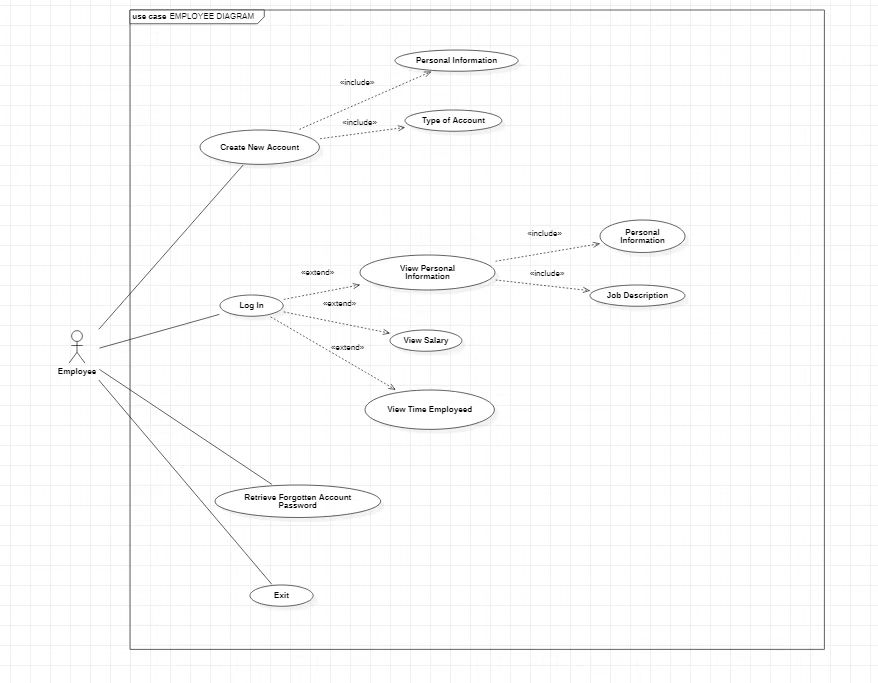
**Credit members**



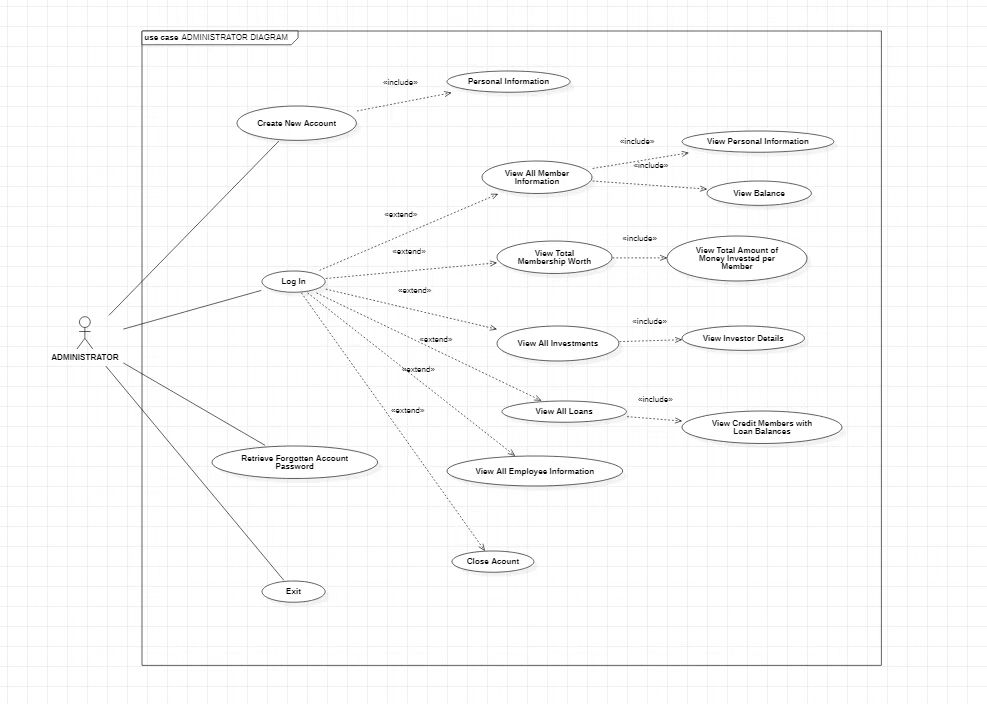
**Investors**



**Employee**



**Administrator**



**USE CASE DESCRIPTIONS:**

|  |  |  |
| --- | --- | --- |
| **Use Case ID:** | UC-1 | |
| **Use Case Name:** | Create New Account | |
| **Actors:** | | Consumer, Business Owner |
| **Description:** | | User would like to create an account |
| **Trigger:** | | User selects option create an account, system input: 2, Enter |
| **Preconditions:** | | User has been issued an account number by the business owner, and agreed on a paid a first deposit amount. |
| **Post conditions:** | | User has created their account successfully, and has submitted his number to the automatic text generator. |
| **Normal Flow:** | | 1. User selects create new account option. 2. User selects their type of Member. 3. System will automatically input the users Type 4. User enters his issued 4 digits account number 5. User provides first name and last name 6. User enters account password 7. User inputs their initial deposited amount 8. System displays account created successfully and logs in the user. |
| **Alternative Flows:** | | None |
| **Exceptions:** | | * If anything is left blank the user cannot proceed to the next step. * Step 2: if Credit or Debit is selected the system will allow additional member options that such as Silver, Gold, Platinum or normal debit, Vip debit respectively. * Step 2: if the entered account number is less than 1000 the user will be prompted that they need a bigger account number. * Step 6: if the input amount is less than the amount required to open the account the user will be required deposit a bigger amount. |
| **Includes:** | | None |
| **Assumptions:** | | The user is signing up with the authority of the business owner and has an attendant assisting them with the sign up. |
| **Notes and Issues:** | | Entering the wrong data value causes a run-time error |

|  |  |  |
| --- | --- | --- |
| **Use Case ID:** | UC-2 | |
| **Use Case Name:** | Administrator Request display all member records | |
| **Actors:** | | Administrator |
| **Description:** | | Administrator would like the system to print out member record information |
| **Trigger:** | | Administrator logs in and selects an option, system input: 1, Enter |
| **Preconditions:** | | None |
| **Post conditions:** | | System Displays member account information |
| **Normal Flow:** | | 1. Administrator select option 1: Log in 2. Administrator Enter their account number and password 3. System verifies the Administrators identity using the account class 4. System Displays the administrator menu 5. Administrator Select option 1: View all member information 6. System Displays all the system information of the members: Account number, type, password, current balance |
| **Alternative Flows:** | | None |
| **Exceptions:** | | * User enters the wrong account number or password: system displays error message and returns the to the welcome menu. |
| **Includes:** | | None |
| **Assumptions:** | | User has already created an account with our system |
| **Notes and Issues:** | | None |

|  |  |  |
| --- | --- | --- |
| **Use Case ID:** | UC-3 | |
| **Use Case Name:** | Administrator Request display all Credit member records | |
| **Actors:** | | Administrator |
| **Description:** | | Administrator would like the system to print out member record information |
| **Trigger:** | | Administrator logs in and selects an option, system input: 1, Enter |
| **Preconditions:** | | None |
| **Post conditions:** | | System Displays Credit member account information |
| **Normal Flow:** | | 1. Administrator select option 1: Log in 2. Administrator Enter their account number and password 3. System verifies the Administrators identity using the account class 4. System Displays the administrator menu 5. Administrator Select option 4: View all Loans 6. System Displays all the system information of the Credit members: Account number, type, loaned balance, Boolean Value: elegibility |
| **Alternative Flows:** | | None |
| **Exceptions:** | | * User enters the wrong account number or password: system displays error message and returns the to the welcome menu. |
| **Includes:** | | None |
| **Assumptions:** | | User has already created an account with our system |
| **Notes and Issues:** | | None |

|  |  |  |
| --- | --- | --- |
| **Use Case ID:** | UC-4 | |
| **Use Case Name:** | Administrator Request display all Debit member records | |
| **Actors:** | | Administrator |
| **Description:** | | Administrator would like the system to print out member record information |
| **Trigger:** | | Administrator logs in and selects an option, system input: 1, Enter |
| **Preconditions:** | | None |
| **Post conditions:** | | System Displays Debit member account information |
| **Normal Flow:** | | 1. Administrator select option 1: Log in 2. Administrator Enter their account number and password 3. System verifies the Administrators identity using the account class 4. System Displays the administrator menu 5. Administrator Select option 4: View All Debit Member Worth 6. System Displays all the system information of the Debit members: Account number, type, Pending transfer amount , Transfer Limit |
| **Alternative Flows:** | | None |
| **Exceptions:** | | * User enters the wrong account number or password: system displays error message and returns the to the welcome menu. |
| **Includes:** | | None |
| **Assumptions:** | | User has already created an account with our system |
| **Notes and Issues:** | | None |

|  |  |  |
| --- | --- | --- |
| **Use Case ID:** | UC-5 | |
| **Use Case Name:** | Retrieve forgotten account information | |
| **Actors:** | | User: Credit Member |
| **Description:** | | User will be able to retrieve forgotten password and/or account number |
| **Trigger:** | | User selects forget account |
| **Preconditions:** | | None |
| **Post conditions:** | | User has created their account successfully |
| **Normal Flow:** | | 1. User selects forget password option 2. User provide the account holders credentials 3. User enters their current balance and their name (current balance can be requested from admin or the user can check their sms system updates) 4. System displays all the user account Information: account number, password, name, type. |
| **Alternative Flows:** | | None |
| **Exceptions:** | | * If any of the entered information is wrong: user will receive and error message and will be sent back to the welcome menu. * If from the registered phone number sends back a freeze code: all operations are frozen until the end of an investigation. |
| **Includes:** | | None |
| **Assumptions:** | | The account holder is either there in person or is aware of the operation |
| **Notes and Issues:** | | None |

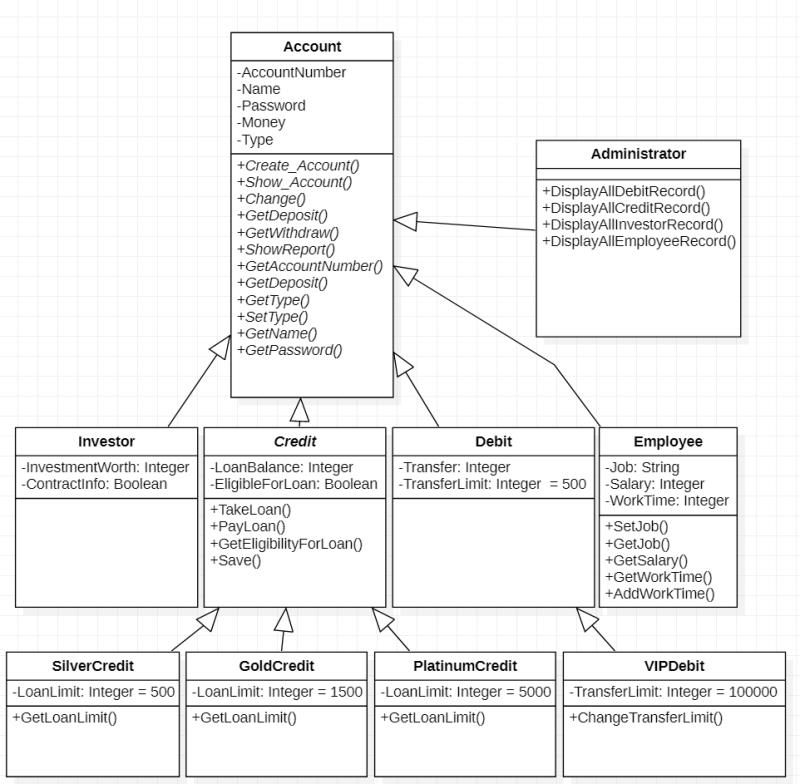
|  |  |  |
| --- | --- | --- |
| **Use Case ID:** | UC-5 | |
| **Use Case Name:** | Debit Member: Withdraw amount | |
| **Actors:** | | User: Debit Member or Vip Debit Member |
| **Description:** | | User would like to withdraw an amount (most likely to purchase or transfer) |
| **Trigger:** | | User selects withdraw amount |
| **Preconditions:** | | User has an active account |
| **Post conditions:** | | Amount withdrawn successfully |
| **Normal Flow:** | | 1. User logs in 2. User selects withdraw amount 3. System displays user’s information 4. User enters agreed upon amount 5. System updates the debit members current balance |
| **Exceptions:** | | * If the users current balance is less than the requested withdrawn amount and is not a credit type account the transaction is denied |
| **Includes:** | | None |
| **Assumptions:** | | None |
| **Notes and Issues:** | |  |

|  |  |  |
| --- | --- | --- |
| **Use Case ID:** | UC-6 | |
| **Use Case Name:** | Close an account | |
| **Actors:** | | User: Credit Member or Employee Member or Debit Member or Investor |
| **Description:** | | User would like to close their account |
| **Trigger:** | | Users requests to close their account |
| **Preconditions:** | | 1. User does not have a credit balance |
| **Post conditions:** | |  |
| **Normal Flow:** | | 1. User selects log in 2. User enters their account number 3. User enters their password 4. User selects close an account 5. System deletes the users account and returns back to the log in page |
| **Alternative Flows:** | | None |
| **Exceptions:** | | None |
| **Includes:** | | None |
| **Assumptions:** | | None |
| **Notes and Issues:** | | None |

|  |  |  |
| --- | --- | --- |
| **Use Case ID:** | UC-7 | |
| **Use Case Name:** | Debit Member: Deposit amount | |
| **Actors:** | | Debit Member or Vip Debit Member |
| **Description:** | | Account holder would like to deposit money into their account. |
| **Trigger:** | | User logs in and clicks on deposit amount. System input: 1 |
| **Preconditions:** | | User has an account and has already paid money at the cash register. |
| **Post conditions:** | | User has successfully deposited money |
| **Normal Flow:** | | 1. User logs into their account 2. User selects deposit amount 3. System displays Users information and asks for the amount they wish to deposit. 4. User enters the amount that they want deposited. |
| **Alternative Flows:** | | None |
| **Exceptions:** | | None |
| **Includes:** | | None |
| **Assumptions:** | | User has already signed up on our system |
| **Notes and Issues:** | | If any different data type is entered the system has a run-time error. |

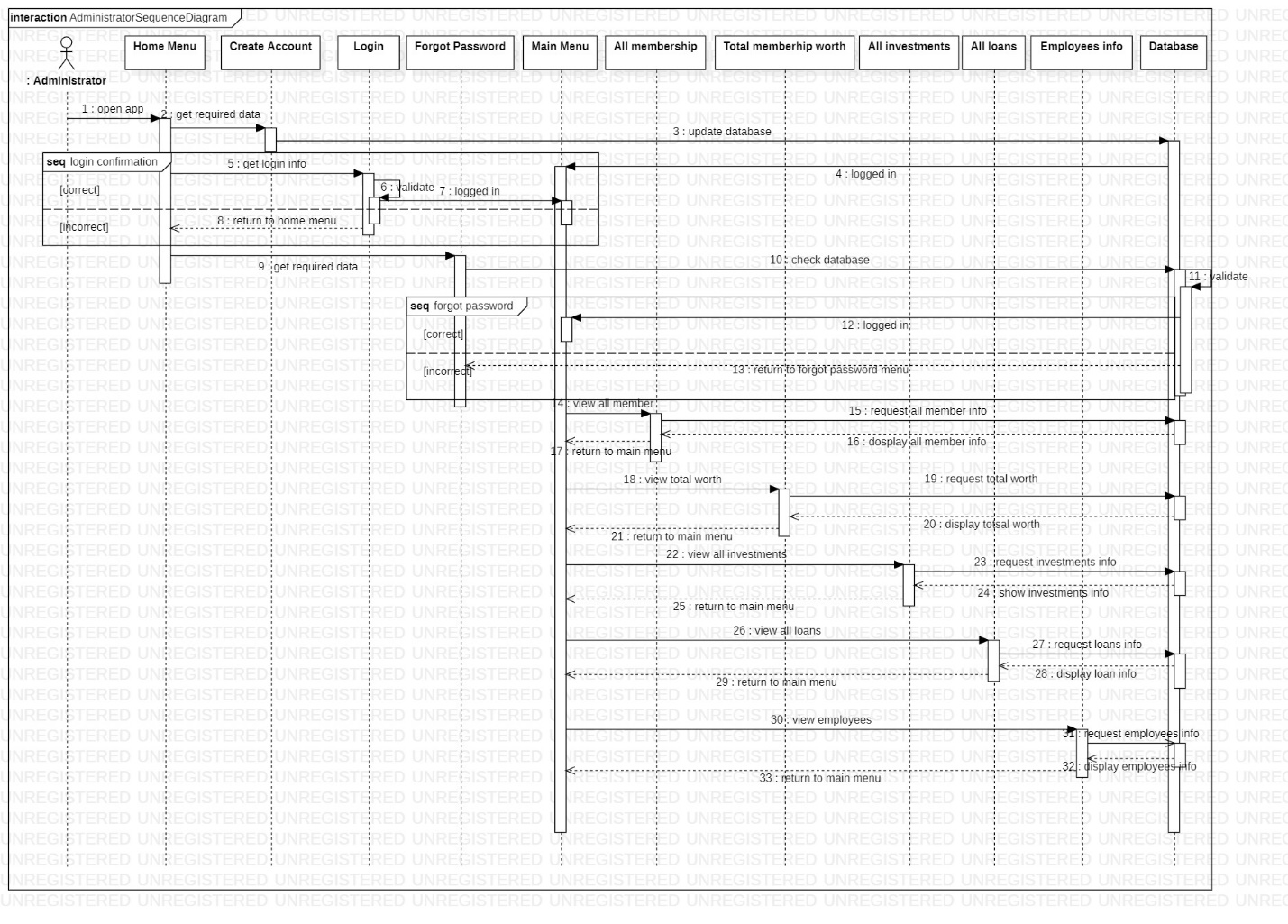
|  |  |  |
| --- | --- | --- |
| **Use Case ID:** | UC-8 | |
| **Use Case Name:** | Credit Member: Take a loan | |
| **Actors:** | | Credit Member |
| **Description:** | | Account holder would like to take a loan and deposit money into their account. |
| **Trigger:** | | User logs in and clicks on deposit amount. System input: 1 |
| **Preconditions:** | | User has an account and has already paid money at the cash register. |
| **Post conditions:** | | User has successfully deposited money |
| **Normal Flow:** | | 1. User logs into their account 2. User selects Take a loan 3. System displays Users information and asks for the amount they wish to loan. 4. User enters the amount that they want. 5. System verifies if the amount is with the loan limit of the credit members level: Silver, Gold, Platinum 6. Loan approved 7. The system updates the users loan balance 8. The system changes the users Boolean Eligibility for a loan 9. The system updates the users current balance |
| **Alternative Flows:** | | None |
| **Exceptions:** | | None |
| **Includes:** | | None |
| **Assumptions:** | | User has already signed up on our system |
| **Notes and Issues:** | | If any different data type is entered the system has a run-time error. |

**CLASS DIAGRAM**

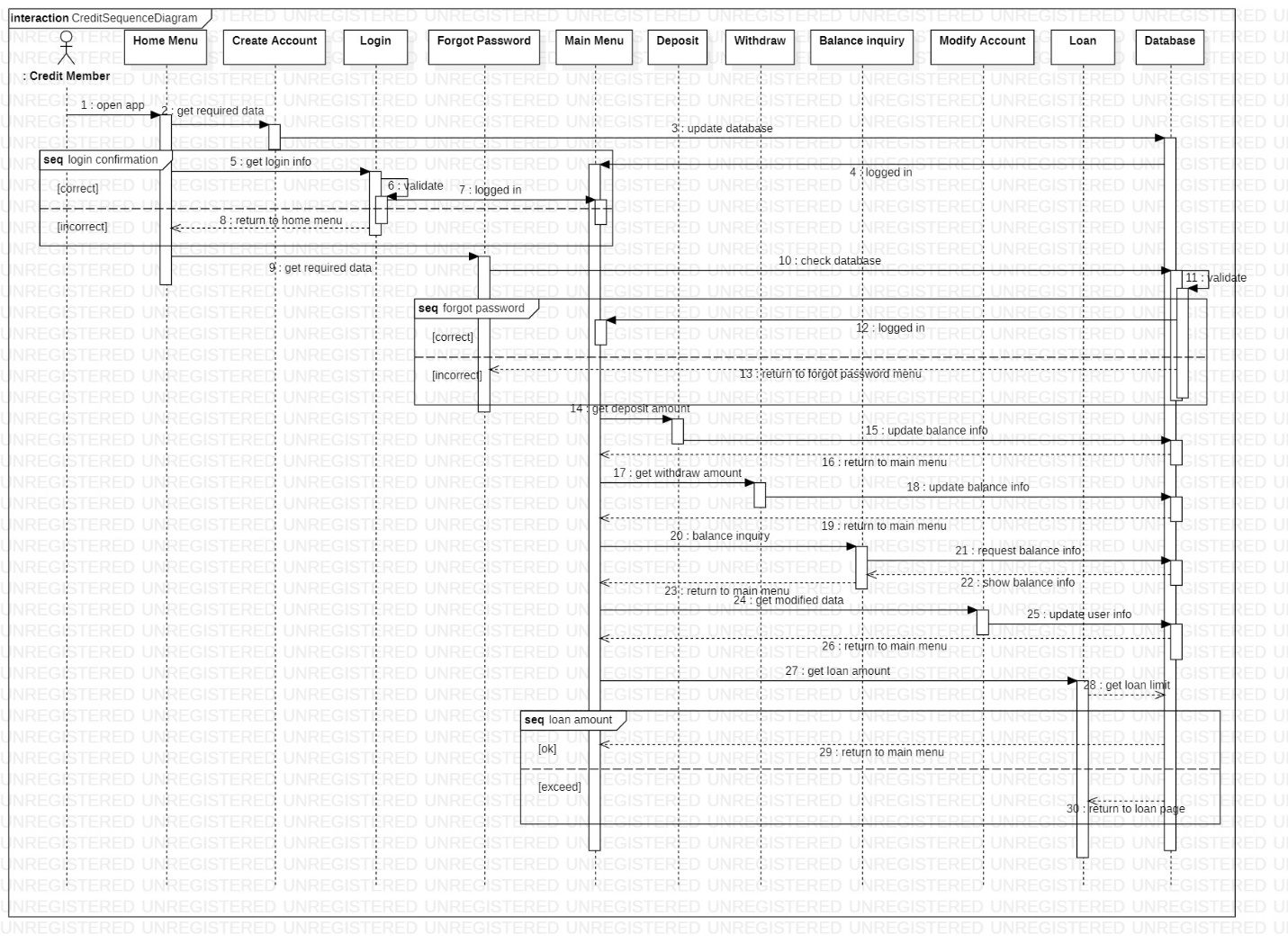


**SEQUENCE DIAGRAM:**

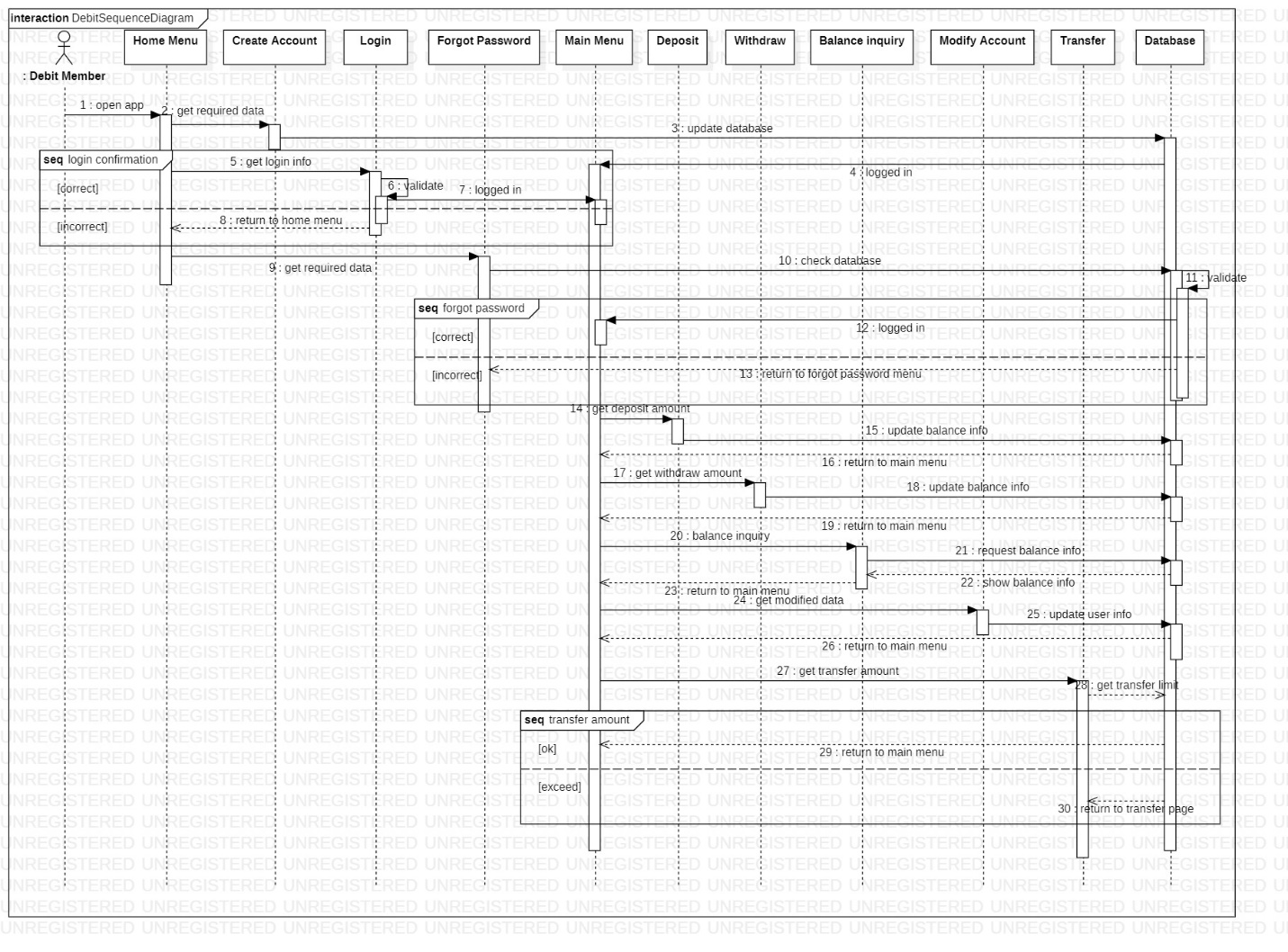
**Administrator**



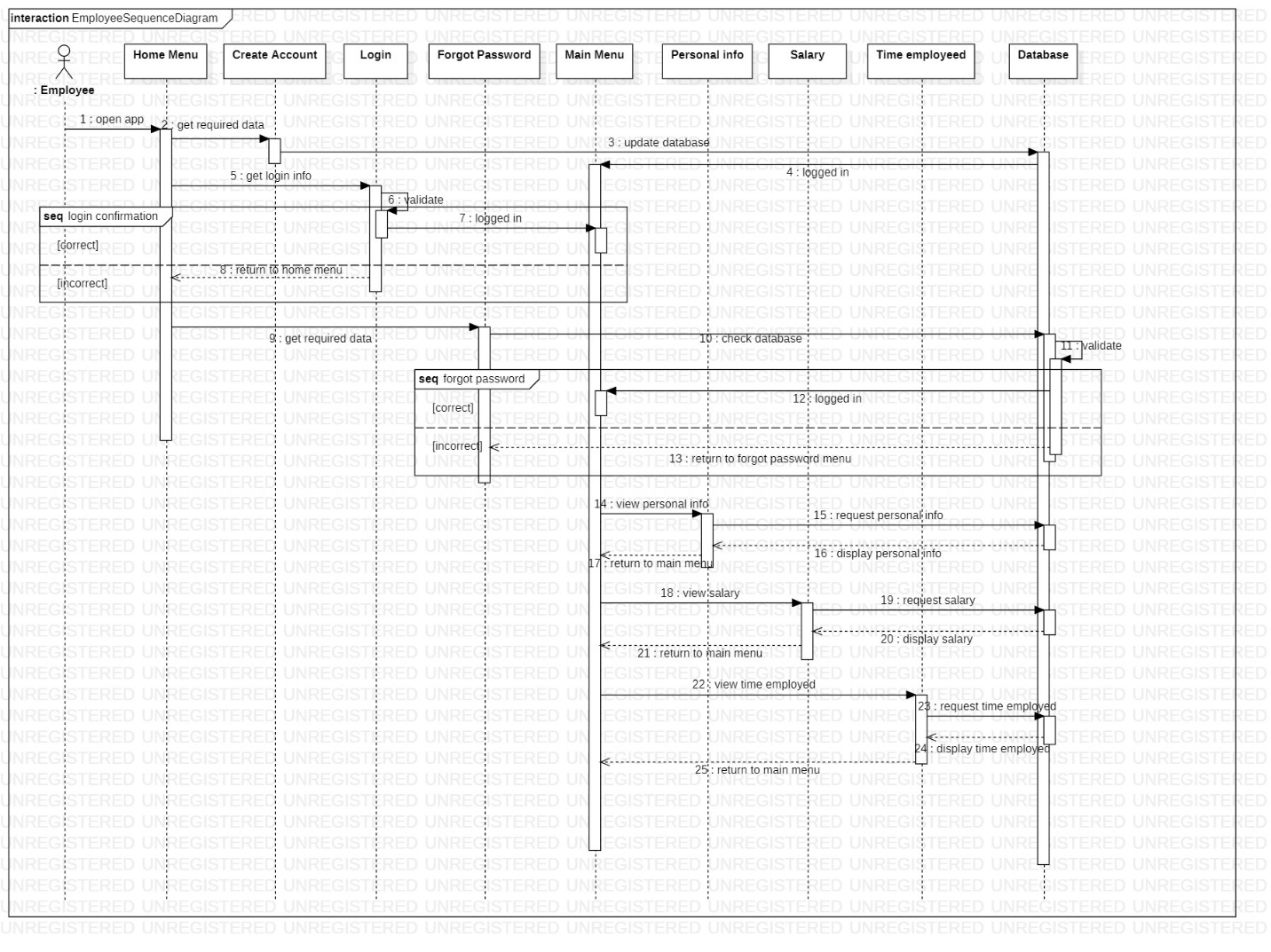
**Credit**



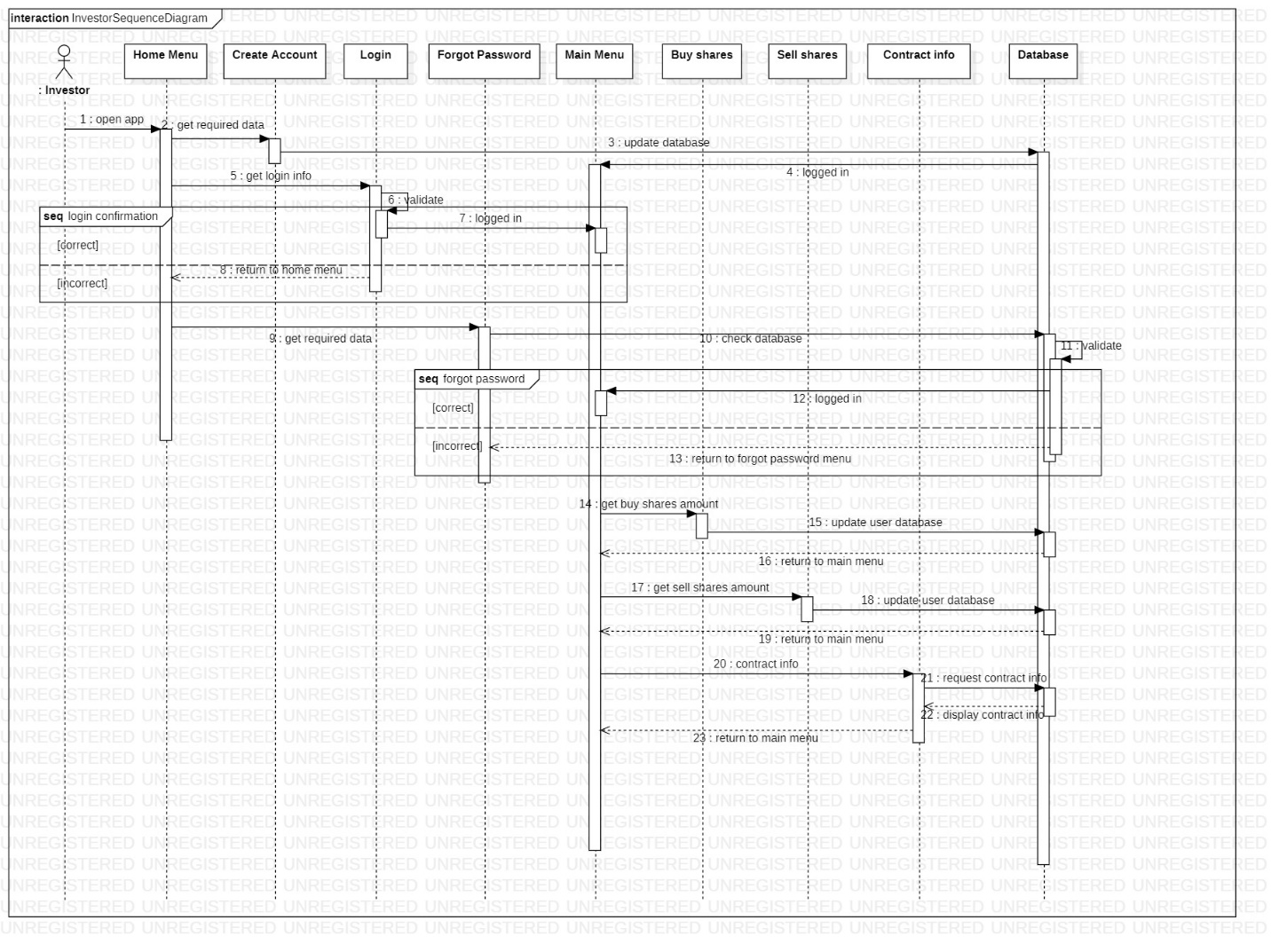
**Debit**



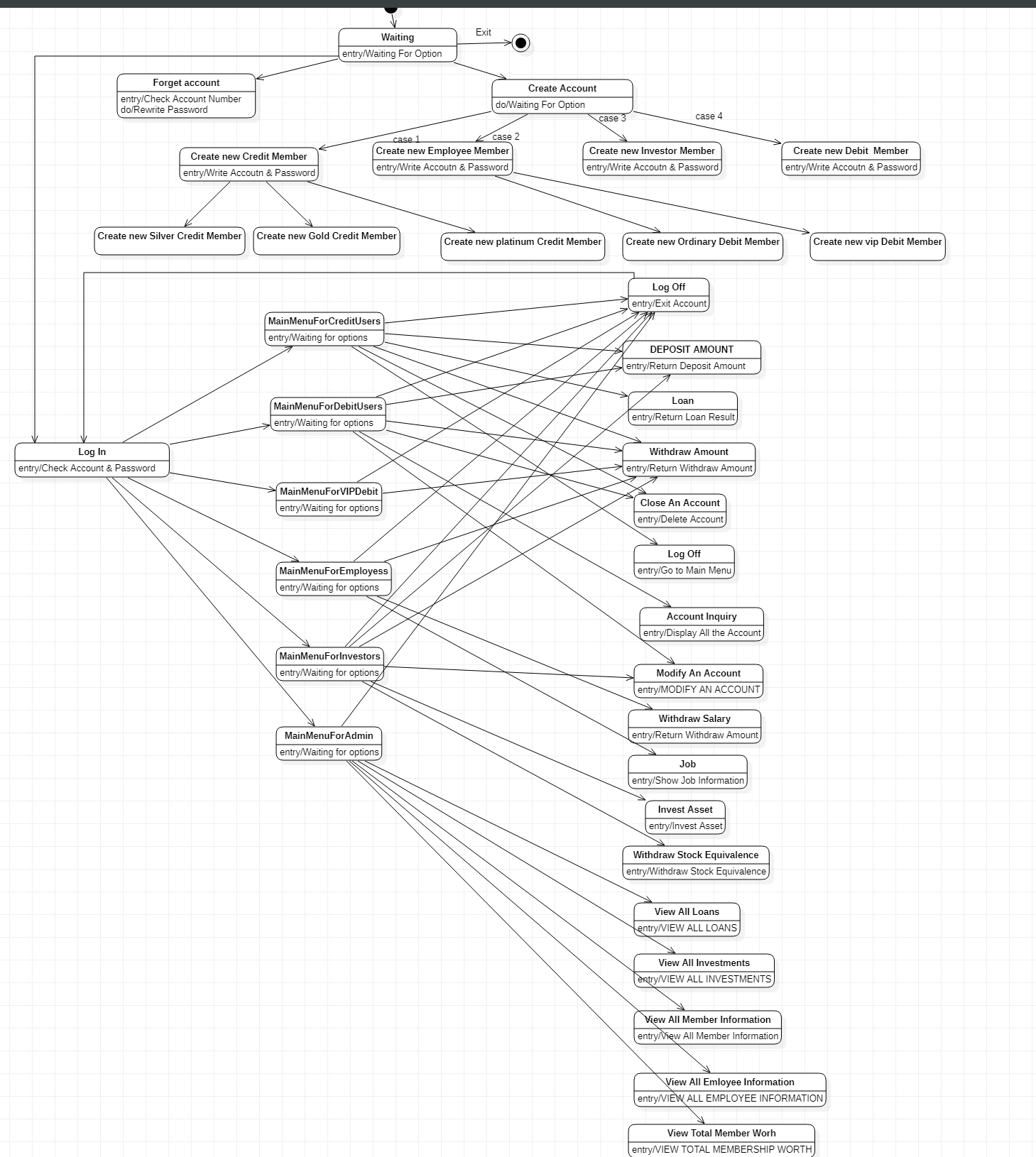
**Investor**



**Employee**



**STATE CHART DIAGRAM**



# Source Code

## Account.h

#pragma once

class account

{

public:

virtual void Create\_account(int tye); //Get data from user to create a bank account

virtual void Show\_account() const; //Show the accounts created on screen

virtual void Change(); //Change the account data

virtual void getDeposit(int); //Get the amount of money from user and add to balance

virtual void getWithdraw(int); //Get the amount of money from user and deduce from balance

virtual void showReport() const; // Show the accounts in the bank

virtual int getAccountnumber() const; //Gets the account number

virtual int getDeposit() const; //Gets the balance amount

virtual int getType(); //Gets the type of account

virtual void setType(int s);

virtual char\* getName(); //Gets the name of the account

virtual char\* getPassword(); //Gets the password of the account

void putdata();

private:

int accountnumber;

char name[100];

char password[250];

int money;

int type;

};

## Account.cpp

#include "account.h"

#include <iostream>

#include<string>

#include<string.h>

#include<fstream>

#include<cctype>

#include<iomanip>

#include<cstdlib>

#include<stdio.h>

using namespace std;

void account::Create\_account(int tye)

{

int numtemp;

Start:

cout << "\n\tPlease enter 4 DIGITS for your account number (DIGITS starts from 1000): ";

cin >> numtemp;

while (numtemp < 1000 || numtemp>9999)

{

cout << "\n\t\tERROR: Invalid DIGITS! " << endl;

cout << "\n\tPlease enter 4 DIGITS for your account number: ";

cin >> numtemp;

}

account ac;

ifstream inFile;

inFile.open("Member.dat", ios::binary);

if (!inFile)

{

cout << "File could not be open !! Press any Key...";

return;

}

while (inFile.read(reinterpret\_cast<char\*> (&ac), sizeof(account)))

{

if (ac.getAccountnumber() == numtemp)

{

cout << "The account number you have entered already exists!";

goto Start;

}

}

inFile.close();

accountnumber = numtemp;

cout << "\n\tEnter the FIRST and LAST name of The account Holder : ";

cin.ignore();

cin.getline(name, 100);

cout << "\n\tEnter the PASSWORD for The account : ";

cin.getline(password, 250);

cout << "\n\tEnter Account openning deposit : ";

cin >> money;

type = tye;

ofstream outFile;

outFile.open("Member.dat", ios::out | ios::binary);

outFile.write(reinterpret\_cast<char\*> (&ac), sizeof(account));

outFile.close();

cout << "\n\n\n\t..........Account Created.........." << endl;

cout << "\n\n\n\t..........Press Enter to Continue.........." << endl;

}

void account::Show\_account() const

{

cout << "\nAccount No. : " << accountnumber;

cout << "\nPassword : " << "\*\*\*\*\*";

cout << "\nAccount Holder Name : " << name;

cout << "\nBalance amount :" << money;

cout << "\nType of Account : " << type;

}

void account::Change()

{

cout << "\nAccount Number : " << accountnumber;

cout << "\n\nModify Account Holder Name : ";

cin.ignore();

cin.getline(name, 100);

cout << "\n\nModify Password : ";

cin.getline(password, 250);

}

void account::getDeposit(int x)

{

money = money + x;

}

void account::getWithdraw(int x)

{

money = money - x;

}

void account::showReport() const

{

cout << accountnumber << setw(10) << " " << name << setw(10) << " " << password << setw(15) << " " << type << setw(20) << " " << money << endl;

}

int account::getAccountnumber() const

{

return accountnumber;

}

int account::getDeposit() const

{

return money;

}

int account::getType()

{

return type;

}

void account::setType(int s)

{

type = s;

}

char\* account::getName()

{

return name;

}

char\* account::getPassword()

{

return password;

}

void account::putdata()

{

cout << accountnumber << setw(10) << " " << name << setw(10) << " " << password << setw(15) << " " << type << setw(20) << money << setw(20) << "\n";

}

## Credit.h

#pragma once

#include "account.h"

class Credit :public account

{

public:

virtual void takeloan(int x);

virtual void payloan(int y);

virtual bool getElegibilityForLoan();

virtual void showCreditMemberReport(Credit ob) const;

virtual void saveCreditMember(Credit ob);

void putdata();

private:

int LoanBalance;

bool ElegibleForLoan;

int accountnumber;

int type;

};

## Credit.cpp

#include <fstream>

#include<cstdlib>

#include<iomanip>

#include<conio.h>

using namespace std;

void Credit::takeloan(int x)

{

LoanBalance = x;

ElegibleForLoan = 0;

};

void Credit::payloan(int y)

{

LoanBalance = -y;

if (LoanBalance == 0)

{

ElegibleForLoan = 1;

};

}

bool Credit::getElegibilityForLoan()

{

return ElegibleForLoan;

};

void Credit::saveCreditMember(Credit ob)

{

ofstream outFile;

outFile.open("CreditMember.dat", ios::out | ios::binary);

outFile.write(reinterpret\_cast<char\*> (&ob), sizeof(Credit));

outFile.close();

}

void Credit::showCreditMemberReport(Credit ob) const

{

cout << "Outstanding Loan Balance: " << LoanBalance << endl;

bool flag = ob.getElegibilityForLoan();

if (flag == 0)

{

cout << "You are currently not elegibil for a loan" << endl;

}

}

void Credit::putdata()

{

cout << accountnumber << setw(10) << " " << type << setw(10) << " " << LoanBalance << setw(15) << " " << ElegibleForLoan << setw(20) << "\n";

}

## SilverCredit.h

#pragma once

#include "Credit.h"

class SilverCredit :public Credit

{

private:

int LoanLimit = 500;

public:

int getLoanLimit();

};

class GoldCredit :public Credit

{

private:

int LoanLimit = 1500;

public:

int getLoanLimit();

};

class PlatinumCredit :public Credit

{

private:

int LoanLimit = 5000;

public:

int getLoanLimit();

};

## SilverCredit.cpp

#include "SilverCredit.h"

int SilverCredit::getLoanLimit()

{

return LoanLimit;

}

int GoldCredit::getLoanLimit()

{

return LoanLimit;

}

int PlatinumCredit::getLoanLimit()

{

return LoanLimit;

}

## Debit.h

#pragma once

#include "account.h"

class Debit : public account

{

private:

int accountnumber;

int type;

int PendingTransfer;

int TransferLimit = 500;

public:

virtual int getTransferLimit();

virtual void saveDebitMember( Debit ob);

void showDebitReport() const;

void putdata();

};

## Debit.cpp

#include "Debit.h"

#include <iostream>

#include <cstring>

#include <fstream>

#include<iomanip>

using namespace std;

int Debit::getTransferLimit()

{

return TransferLimit;

}

void Debit::saveDebitMember(Debit ob)

{

ofstream outFile;

outFile.open("OrdinaryDebitMember.dat", ios::out | ios::binary);

outFile.write(reinterpret\_cast<char\*> (&ob), sizeof(Debit));

outFile.close();

}

void Debit::putdata()

{

cout << accountnumber << setw(10) << " " << type << setw(10) << " " << PendingTransfer << setw(25) << " " << TransferLimit << setw(20) <<"\n";

}

## VipDebit.h

#pragma once

#include "Debit.h"

class VipDebit :public Debit

{

private:

int TransferLimit = 100000;

public:

void ChangeTransferLimit(int change);

};

## VipDebit.cpp

#include "VipDebit.h"

#include <iostream>

using namespace std;

void VipDebit::ChangeTransferLimit(int change)

{

if (change < 100000)

{

TransferLimit = change;

}

else

{

cout << "Requsted change limit is out of range, please retry with a suitable limit" << endl;

}

}

## Administrator.h

#pragma once

#include "account.h"

#include "Credit.h"

#include "Debit.h"

#include "employee.h"

#include "Investor.h"

class Administrator :public account , Credit , Debit, employee ,Investor

{

public:

void DisplayAllDebitRecord();

void DisplayAllCreditRecord();

void DisplayAllInvestorRecord();

void DisplayAllemployeeRecord();

};

## Administration.cpp

#include "Administrator.h"

#include<iostream>

#include<fstream>

#include<cctype>

#include<iomanip>

#include<cstdlib>

#include<cstring>

#include<conio.h>

using namespace std;

void Administrator :: DisplayAllDebitRecord()

{

fstream f;

Debit Stu;

f.open("DebitMember.dat", ios::in | ios::binary);

cout << "\n\n\t\t DEBIT ACCOUNT HOLDER LIST\n\n";

cout << "========================================================================================\n";

cout << "Acc no. Type Pending Transfer Amount Transfer Limit\n";

cout << "========================================================================================\n";

while ((f.read((char\*)&Stu, sizeof(Stu))))

Stu.putdata();

f.close();

}

void Administrator :: DisplayAllCreditRecord()

{

fstream f;

Credit Stu;

f.open("CreditMember.dat", ios::in | ios::binary);

cout << "\n\n\t\t CREDIT ACCOUNT HOLDER LIST\n\n";

cout << "========================================================================================\n";

cout << "Acc no. Type Loaned Balance Eligibility \n";

cout << "========================================================================================\n";

while ((f.read((char\*)&Stu, sizeof(Stu))))

Stu.putdata();

f.close();

}

void Administrator :: DisplayAllInvestorRecord()

{

fstream f;

Investor Stu;

f.open("InvestorMember.dat", ios::in | ios::binary);

cout << "\n\n\t\t INVESTORS ACCOUNT HOLDER LIST\n\n";

cout << "========================================================================================\n";

cout << "Acc no. Type Invested Net Worth Access to contract\n";

cout << "========================================================================================\n";

while ((f.read((char\*)&Stu, sizeof(Stu))))

Stu.putdata();

f.close();

}

void Administrator :: DisplayAllemployeeRecord()

{

fstream f;

employee Stu;

f.open("EmployeeMember.dat", ios::in | ios::binary);

cout << "\n\n\t\t EMPLOYEE ACCOUNT HOLDER LIST\n\n";

cout << "========================================================================================\n";

cout << "Acc no. Type Job Salary Work Time\n";

cout << "========================================================================================\n";

while ((f.read((char\*)&Stu, sizeof(Stu))))

Stu.putdata();

f.close();

}

## Employee.h

#pragma once

#include "account.h"

class employee :public account

{

private:

int accountnumber;

int type;

char Job[100];

int Salary;

int Worktime;

public:

void SetJob(char s[100]);

char\* GetJob();

int GetSalary();

int getWorkTime();

void Addworktime(int x);

void save(account obj, employee ob);

void saveEmployee(employee ob);

void putdata();

};

## Employee.cpp

#include "employee.h"

#include <iostream>

#include <cstring>

#include <fstream>

#include<iomanip>

using namespace std;

void employee::SetJob(char s[100])

{

Job[100] = s[100];

}

char\* employee::GetJob()

{

return Job;

}

int employee::GetSalary()

{

return Salary;

}

int employee::getWorkTime()

{

return Worktime;

}

void employee::Addworktime(int x)

{

Worktime =+ x;

}

void employee::save(account obj, employee ob)

{

ofstream outFile;

outFile.open("EmployeeMember.dat", ios::out | ios::binary);

outFile.write(reinterpret\_cast<char\*> (&obj, &ob), sizeof(account));

outFile.close();

}

void employee::saveEmployee(employee ob)

{

ofstream outFile;

outFile.open("DeditMember.dat", ios::out | ios::binary);

outFile.write(reinterpret\_cast<char\*> (&ob), sizeof(employee));

outFile.close();

}

void employee::putdata()

{

cout << accountnumber << setw(10) << " " << type << setw(10) << " " << Job << setw(15) << " " << Salary << setw(20) << " " << Worktime << setw(20) << "\n";

}

## Investor.h

#pragma once

#include "account.h"

class Investor : public account

{

private:

int accountnumber;

int type;

int InvestmentWorth;

bool contractInfo;

public:

void saveInvestor( Investor ob);

void showInvestorReport(Investor obj );

void putdata();

};

## Investor.cpp

#include "Investor.h"

#include <iostream>

#include <cstring>

#include <fstream>

#include<iomanip>

using namespace std;

void Investor::saveInvestor( Investor ob)

{

ofstream outFile;

outFile.open("InvestorMember.dat", ios::out | ios::binary);

outFile.write(reinterpret\_cast<char\*> ( &ob), sizeof(Investor));

outFile.close();

}

void Investor::showInvestorReport(Investor obj)

{

cout << InvestmentWorth << " "<< contractInfo << endl;

}

void Investor::putdata()

{

cout << accountnumber << setw(10) << " " << type << setw(10) << " " << InvestmentWorth << setw(15) << " " << contractInfo << setw(20) << "\n";

}

## Main.cpp

#include<iostream>

#include<fstream>

#include<cctype>

#include<iomanip>

#include<cstdlib>

#include<cstring>

#include <windows.h>

#include "account.h"

#include "Credit.h"

#include<iostream>

#include<fstream>

#include<cctype>

#include<iomanip>

#include<cstdlib>

#include<cstring>

#include<conio.h>

#include"account.h"

#include "Credit.h"

#include "SilverCredit.h"

#include "Debit.h"

#include "Administrator.h"

#include "employee.h"

#include "Investor.h"

#include "VipDebit.h"

using namespace std;

void write\_account(int type);

void display\_sp(int);

void change\_account(int);

void delete\_account(int);

void display\_all\_accounts();

void deposit\_withdraw(int, int);

void main\_menu(account ac);

void log\_in(int, char\*);

void show\_greeting(int);

void forget\_account(char\*, int);

int main();

void loanloan(account ac);

string adminverif;

string adminpass = "locationnanhang";

bool flagf = false;

Debit debobj;

Investor invobj;

VipDebit vipobj;

SilverCredit Silobj;

GoldCredit Gldobj;

PlatinumCredit Pltobj;

Credit crdobj;

employee Empobj;

char ch1;

int number;

char ch2;

int numtemp;

// MAIN MENU FUNCTION //

void main\_menu(account ac)

{

char ch;

do

{

show\_greeting(number);

cout << "\n\n\n\tMAIN MENU";

cout << "\n\n\t1. DEPOSIT AMOUNT";

cout << "\n\n\t2. WITHDRAW AMOUNT";

cout << "\n\n\t3. BALANCE INQUIRY";

cout << "\n\n\t5. CLOSE AN ACCOUNT";

cout << "\n\n\t6. MODIFY AN ACCOUNT";

cout << "\n\n\t7. LOG OFF";

cout << "\n\n\tSelect Your Option (1-7) : ";

{

cin >> ch;

switch (ch)

{

case '1':

system("CLS");

deposit\_withdraw(number, 1);

break;

case '2':

system("CLS");

deposit\_withdraw(number, 2);

break;

case '3':

display\_sp(number);

break;

case '4':

display\_all\_accounts();

break;

case '5':

system("CLS");

delete\_account(number);

break;

case '6':

change\_account(number);

system("CLS");

break;

case '7':

main();

system("CLS");

break;

default:cout << "\a";

}

cin.ignore();

cin.get();

}

} while (ch != '7');

}

void MainMenuForCreditUsers(account ac)

{

char ch;

do

{

cout << "\n\tMAIN MENU";

cout << "\n\n\t1. DEPOSIT AMOUNT";

cout << "\n\n\t2. TAKE A LOAN";

cout << "\n\n\t3. WITHDRAW AMOUNT";

cout << "\n\n\t4. ACCOUNT INQUIRY";

cout << "\n\n\t5. CLOSE AN ACCOUNT";

cout << "\n\n\t6. MODIFY AN ACCOUNT";

cout << "\n\n\t7. LOG OFF";

cout << "\n\n\tSelect Your Option (1-7) : ";

cin >> ch;

switch (ch)

{

case '1':

system("CLS");

deposit\_withdraw(number, 1);

break;

case '2':

loanloan(ac);

break;

case '3':

system("CLS");

deposit\_withdraw(number, 2);

break;

case '4':

system("CLS");

display\_sp(number);

break;

case '5':

system("CLS");

delete\_account(number);

break;

case '6':

system("CLS");

change\_account(number);

break;

case '7':

main();

system("CLS");

break;

default:cout << "\a";

}

cin.ignore();

cin.get();

} while (ch != '7');

}

void MainMenuForVIPDebit(account ac)

{

char ch;

do

{

cout << "\n\tMAIN MENU";

cout << "\n\n\t1. DEPOSIT AMOUNT";

cout << "\n\n\t2. WITHDRAW AMOUNT";

cout << "\n\n\t3. ACCOUNT INQUIRY";

cout << "\n\n\t4. CLOSE AN ACCOUNT";

cout << "\n\n\t5. MODIFY AN ACCOUNT";

cout << "\n\n\t6. TRANSFER FUNDS";

cout << "\n\n\t7. LOG OFF";

cout << "\n\n\tSelect Your Option (1-7) : ";

cin >> ch;

switch (ch)

{

case '1':

system("CLS");

deposit\_withdraw(number, 1);

break;

case '2':

system("CLS");

deposit\_withdraw(number, 2);

break;

case '3':

display\_sp(number);

break;

case '4':

display\_all\_accounts();

break;

case '5':

system("CLS");

delete\_account(number);

break;

case '6':

system("CLS");

break;

case '7':

main();

system("CLS");

break;

default:cout << "\a";

}

cin.ignore();

cin.get();

} while (ch != '7');

}

void MainMenuForEmployess(account ac)

{

char ch;

do

{

cout << "\n\tMAIN MENU";

cout << "\n\n\t1. WITHDRAW SALARY";

cout << "\n\n\t2. JOB";

cout << "\n\n\t3. TIME EMPLOYEED";

cout << "\n\n\t7. LOG OFF";

cout << "\n\n\tSelect Your Option (1-7) : ";

cin >> ch;

switch (ch)

{

case '1':

system("CLS");

break;

case '2':

system("CLS");

break;

case '3':

break;

case '4':

break;

case '5':

system("CLS");

break;

case '6':

system("CLS");

break;

case '7':

main();

system("CLS");

break;

default:cout << "\a";

}

cin.ignore();

cin.get();

} while (ch != '7');

}

void MainMenuForInvestors(account ac)

{

char ch;

do

{

cout << "\n\tMAIN MENU";

cout << "\n\n\t1. DEPOSIT AMOUNT";

cout << "\n\n\t2. WITHDRAW AMOUNT";

cout << "\n\n\t3. INVEST ASSET";

cout << "\n\n\t3. WITHDRAW STOCK EQUIVALENCE";

cout << "\n\n\t4. REQUEST CONTRACT REVELUATION";

cout << "\n\n\t5. MODIFY AN ACCOUNT";

cout << "\n\n\t7. LOG OFF";

cout << "\n\n\tSelect Your Option (1-7) : ";

cin >> ch;

switch (ch)

{

case '1':

system("CLS");

break;

case '2':

system("CLS");

break;

case '3':

break;

case '4':

break;

case '5':

system("CLS");

break;

case '6':

system("CLS");

break;

case '7':

main();

system("CLS");

break;

default:cout << "\a";

}

cin.ignore();

cin.get();

} while (ch != '7');

}

void MainMenuForAdmin(account ac)

{

Administrator ad;

char ch;

do

{

cout << "\n\tMAIN MENU";

cout << "\n\n\t1. VIEW ALL MEMBER INFORMATION";

cout << "\n\n\t2. VIEW ALL DEBIT MEMBERSHIP WORTH";

cout << "\n\n\t3. VIEW ALL INVESTMENTS";

cout << "\n\n\t4. VIEW ALL LOANS";

cout << "\n\n\t5. VIEW ALL EMPLOYEE INFORMATION";

cout << "\n\n\t7. LOG OFF";

cout << "\n\n\tSelect Your Option (1-7) : ";

cin >> ch;

switch (ch)

{

case '1':

system("CLS");

display\_all\_accounts();

break;

case '2':

system("CLS");

ad.DisplayAllDebitRecord();

break;

case '3':

system("CLS");

ad.DisplayAllInvestorRecord();

break;

case '4':

system("CLS");

ad.DisplayAllCreditRecord();

break;

case '5':

system("CLS");

ad.DisplayAllemployeeRecord();

break;

case '6':

break;

case '7':

main();

system("CLS");

break;

default:cout << "\a";

}

cin.ignore();

cin.get();

} while (ch != '7');

}

// FUNCTION TO CREATE A NEW ACCOUNT //

void write\_account(int type)

{

account ac;

ofstream outFile;

outFile.open("Member.dat", ios::binary | ios::app);

ac.Create\_account(type);

outFile.write(reinterpret\_cast<char\*> (&ac), sizeof(account));

outFile.close();

number = ac.getAccountnumber();

main();

}

// FUNCTION TO DISPLAY ACCOUNT DETAILS //

void display\_sp(int n)

{

account ac;

bool flag = false;

ifstream inFile;

inFile.open("Member.dat", ios::binary);

if (!inFile)

{

cout << "File could not be open !! Press any Key...";

return;

}

cout << "\nBALANCE DETAILS\n";

while (inFile.read(reinterpret\_cast<char\*> (&ac), sizeof(account)))

{

if (ac.getAccountnumber() == n)

{

ac.Show\_account();

cout << "\n\n\t ..........Press Enter to Continue.......... " << endl;

flag = true;

}

}

inFile.close();

if (flag == false)

cout << "\n\n..........Account does not exist..........";

}

// FUNCTION TO CHANGE THE DETAILS OF THE ACCOUNT //

void change\_account(int n)

{

account ac;

bool found = false;

fstream File;

File.open("Member.dat", ios::binary | ios::in | ios::out);

if (!File)

{

cout << "File could not be open !! Press any Key...";

return;

}

while (!File.eof() && found == false)

{

File.read(reinterpret\_cast<char\*> (&ac), sizeof(account));

if (ac.getAccountnumber() == n)

{

ac.Show\_account();

cout << "\n\nEnter The New Details of account: " << endl;

ac.Change();

int pos = (-1) \* static\_cast<int>(sizeof(account));

File.seekp(pos, ios::cur);

File.write(reinterpret\_cast<char\*> (&ac), sizeof(account));

cout << "\n\n\t ..........Record Successfully Updated..........";

cout << "\n\n\t ..........Press Enter to Continue.......... ";

found = true;

}

}

File.close();

if (found == false)

cout << "\n\n ..........Record Not Found.......... ";

}

// FUNCTION TO DELETE AN ACCOUNT //

void delete\_account(int n)

{

account ac;

ifstream inFile;

ofstream outFile;

inFile.open("Member.dat", ios::binary);

if (!inFile)

{

cout << "File could not be open !! Press any Key...";

return;

}

outFile.open("Temp.dat", ios::binary);

inFile.seekg(0, ios::beg);

while (inFile.read(reinterpret\_cast<char\*> (&ac), sizeof(account)))

{

if (ac.getAccountnumber() != n)

{

outFile.write(reinterpret\_cast<char\*> (&ac), sizeof(account));

}

}

inFile.close();

outFile.close();

remove("Member.dat");

rename("Temp.dat", "Member.dat");

cout << "\n\n\t..........Account Successfully Closed..........";

main();

}

// FUNCTION TO DISPLAY ALL THE ACCOUNTS //

void display\_all\_accounts()

{

account ac;

ifstream inFile;

inFile.open("Member.dat", ios::binary);

if (!inFile)

{

cout << "File could not be open !! Press any Key...";

return;

}

cout << "\n\n\t\tACCOUNT HOLDER LIST\n\n";

cout << "========================================================================================\n";

cout << "Acc no. NAME Password Type Balance\n";

cout << "========================================================================================\n";

while (inFile.read(reinterpret\_cast<char\*> (&ac), sizeof(account)))

{

ac.showReport();

}

inFile.close();

cout << "\n\n\n\t ..........Press Enter to Continue.......... " << endl;

}

// FUNCTION TO DEPOSIT AND WITHDRAW MONEY //

void deposit\_withdraw(int n, int option)

{

int amount;

bool found = false;

account ac;

fstream File;

File.open("Member.dat", ios::binary | ios::in | ios::out);

if (!File)

{

cout << "File could not be open !! Press any Key...";

return;

}

while (!File.eof() && found == false)

{

File.read(reinterpret\_cast<char\*> (&ac), sizeof(account));

if (ac.getAccountnumber() == n)

{

ac.Show\_account();

if (option == 1)

{

cout << "\n\n\tTO DEPOSIT MONEY ";

cout << "\n\nEnter The amount of money to be deposited: ";

cin >> amount;

ac.getDeposit(amount);

}

if (option == 2)

{

cout << "\n\n\tTO WITHDRAW MONEY ";

cout << "\n\nEnter The amount of money to be withdrawn: ";

cin >> amount;

int balance = ac.getDeposit() - amount;

ac.getWithdraw(amount);

}

int pos = (-1) \* static\_cast<int>(sizeof(ac));

File.seekp(pos, ios::cur);

File.write(reinterpret\_cast<char\*> (&ac), sizeof(account));

cout << "\n\n\t ..........Record Successfully Updated..........";

cout << "\n\n\t ..........A TEXT MESSAGE has been sent to your PHONE about the Update" << endl;

cout << "\n\n\t ..........Press Enter to Continue.........." << endl;

system("pause");

found = true;

}

}

File.close();

if (found == false)

cout << "\n\n ..........Record Not Found.......... ";

}

// FUNCTION TO TAKE LOAN //

void loanloan(account ac) {

cout << "How much Would you like to loan from the club" << endl;

int x;

cin >> x;

Credit CredMem;

SilverCredit SilCred;

GoldCredit GoldCred;

PlatinumCredit platCred;

if (CredMem.getElegibilityForLoan())

{

if (ac.getType() == 10000001)

{

if (x < SilCred.getLoanLimit())

{

ac.getDeposit(x);

CredMem.takeloan(x);

cout << "Loan approved" << endl;

CredMem.saveCreditMember(CredMem);

MainMenuForCreditUsers(ac);

}

else

{

cout << "Requested loan amount rejected" << endl;

MainMenuForCreditUsers(ac);

}

}

else if (ac.getType() == 10000010)

{

if (x < GoldCred.getLoanLimit())

{

CredMem.takeloan(x);

cout << "Loan approved" << endl;

CredMem.saveCreditMember(CredMem);

MainMenuForCreditUsers(ac);

}

else

{

cout << "Requested loan amount rejected" << endl;

MainMenuForCreditUsers(ac);

}

}

else if (ac.getType() == 10000011)

{

if (x < platCred.getLoanLimit())

{

CredMem.takeloan(x);

cout << "Loan approved" << endl;

CredMem.saveCreditMember(CredMem);

MainMenuForCreditUsers(ac);

}

else

{

cout << "Requested loan amount rejected" << endl;

MainMenuForCreditUsers(ac);

}

}

}

else

{

cout << "You still have and outstanding loan, pay outstanding loan to become elegible for a new loan" << endl;

}

}

// LOG IN FUNCTION //

void log\_in(int n, char\* p)

{

account ac;

bool flag = false;

ifstream inFile;

inFile.open("Member.dat", ios::binary);

if (!inFile)

{

cout << "File could not be open !! Press any Key...";

return;

}

while (inFile.read(reinterpret\_cast<char\*> (&ac), sizeof(account)))

{

if (ac.getAccountnumber() == n && (strcmp(ac.getPassword(), p)) == 0)

{

int type = ac.getType();

int Silver = 10000001;

int Gold = 10000010;

int Platinum = 10000011;

int Debit = 10000100;

int Vip = 10000101;

int Employee = 10000110;

int investor = 1000010000;

int admin = 10000111;

system("CLS");

if ((type == Silver) || (type == Gold) || type == Platinum)

{

MainMenuForCreditUsers(ac);

}

else if (type == Debit)

{

main\_menu(ac);

}

else if (type == Vip)

{

MainMenuForVIPDebit(ac);

}

else if (type == Employee)

{

MainMenuForEmployess(ac);

}

else if (type == investor)

{

MainMenuForInvestors(ac);

}

else if (type == admin)

{

MainMenuForAdmin(ac);

}

flag = true;

}

}

inFile.close();

if (flag == false)

cout << "\n\tPress ENTER and input an existing account number and VALID PASSWORD";

}

// FUNCTION TO SHOW GREETINGS TO THE USER //

void show\_greeting(int n)

{

account ac;

ifstream inFile;

inFile.open("Member.dat", ios::binary);

if (!inFile)

{

cout << "File could not be open !! Press any Key...";

return;

}

cout << "\n\n\n\tHello There User\n";

while (inFile.read(reinterpret\_cast<char\*> (&ac), sizeof(account)))

{

if (ac.getAccountnumber() == n)

{

ac.Show\_account();

adminverif = ac.getName();

}

}

inFile.close();

}

// FORGET ACCOUNT FUNCTION //

void forget\_account(char\* name, int balance)

{

account ac;

bool flag = false;

ifstream inFile;

inFile.open("Member.dat", ios::binary);

if (!inFile)

{

cout << "File could not be open !! Press any Key...";

return;

}

while (inFile.read(reinterpret\_cast<char\*> (&ac), sizeof(account)))

{

if (strcmp(ac.getName(), name) == 0 && ac.getDeposit() == balance)

{

cout << "\n\n\tAccount Information";

ac.Show\_account();

flag = true;

cout << "\n\n\t ..........Press Enter to Continue.......... " << endl;

}

}

inFile.close();

if (flag == false)

cout << "\n\n..........Account does not exist..........";

}

int main()

{

char ch;

int num;

char name[100];

char password[250];

int balance;

int type;

account ac;

do

{

cout << "\n\n\n\t\tWelcome to Fund MGMT";

cout << "\n\n\t1. LOG IN";

cout << "\n\n\t2. CREATE A NEW ACCOUNT";

cout << "\n\n\t3. FORGOT ACCOUNT";

cout << "\n\n\t4. EXIT";

cout << "\n\n\tSelect Your Option (1-4) : ";

cin >> ch;

switch (ch)

{

case '1':

cout << "\n\n\tEnter The account Number : ";

cin >> num;

number = num;

cout << "\n\n\tEnter Password : ";

cin.ignore();

cin.getline(password, 250);

log\_in(num, password);

break;

case '2':

{

cout << "\n\n\t1. Create new Credit Member";

cout << "\n\n\t2. Create new Debit Member";

cout << "\n\n\t3. Create new Employee Member";

cout << "\n\n\t4. Create new Investor Member";

cout << "\n\n\tSelect Your Option (1-4) : ";

cin >> ch1;

switch (ch1) {

case '1': {

cout << "\n\n\t1. Create new Silver Credit Member";

cout << "\n\n\t2. Create new Gold Credit Member";

cout << "\n\n\t3. Create new platinum Credit Member";

cout << "\n\n\tSelect Your Option (1-3) : ";

cin >> ch2;

switch (ch2)

{

case '1':

type = 10000001;

ac.setType(type);

write\_account(type);

cout << "\n\n Welcome To the Membership Fund Managment System" << endl;

break;

case '2':

type = 10000010;

ac.setType(type);

write\_account(type);

cout << "\n\n Welcome To the Membership Fund Managment System" << endl;

main();

break;

case '3':

type = 10000011;

ac.setType(type);

write\_account(type);

cout << "\n\n Welcome To the Membership Fund Managment System" << endl;

main();

break;

}

}

break;

case '2': {

cout << "\n\n\t1. Create new Ordinary Debit Member";

cout << "\n\n\t2. Create new vip Debit Member";

cin >> ch2;

switch (ch2)

{

case '1':

type = 10000100;

ac.setType(type);

write\_account(type);

cout << "\n\n Welcome To the Membership Fund Managment System" << endl;

main();

break;

case '2':

type = 10000101;

ac.setType(type);

write\_account(type);

cout << "\n\n Welcome To the Membership Fund Managment System" << endl;

main();

break;

}

}

break;

case '3': {

type = 10000110;

ac.setType(type);

write\_account(type);

cout << "\n\n Welcome To the Membership Fund Managment System" << endl;

main();

}

break;

case '4': {

type = 10001000;

ac.setType(type);

write\_account(type);

cout << "\n\n Welcome To the Membership Fund Managment System" << endl;

main();

case '5': {

type = 10000111;

ac.setType(type);

write\_account(type);

cout << "\n\n Welcome To the Membership Fund Managment System" << endl;

main();

}

}

break;

}

system("CLS");

break;

case '3':

cout << "\n\tName : ";

cin.ignore();

cin.getline(name, 100);

cout << "\n\tBalance (Check your phone for the balance) : ";

cin >> balance;

forget\_account(name, balance);

break;

case '4':

cout << "\n\n\tThanks for using the Gucci Bank Managemnt System" << endl;

exit(0);

system("CLS");

break;

default:cout << "\a";

}

cin.ignore();

cin.get();

}

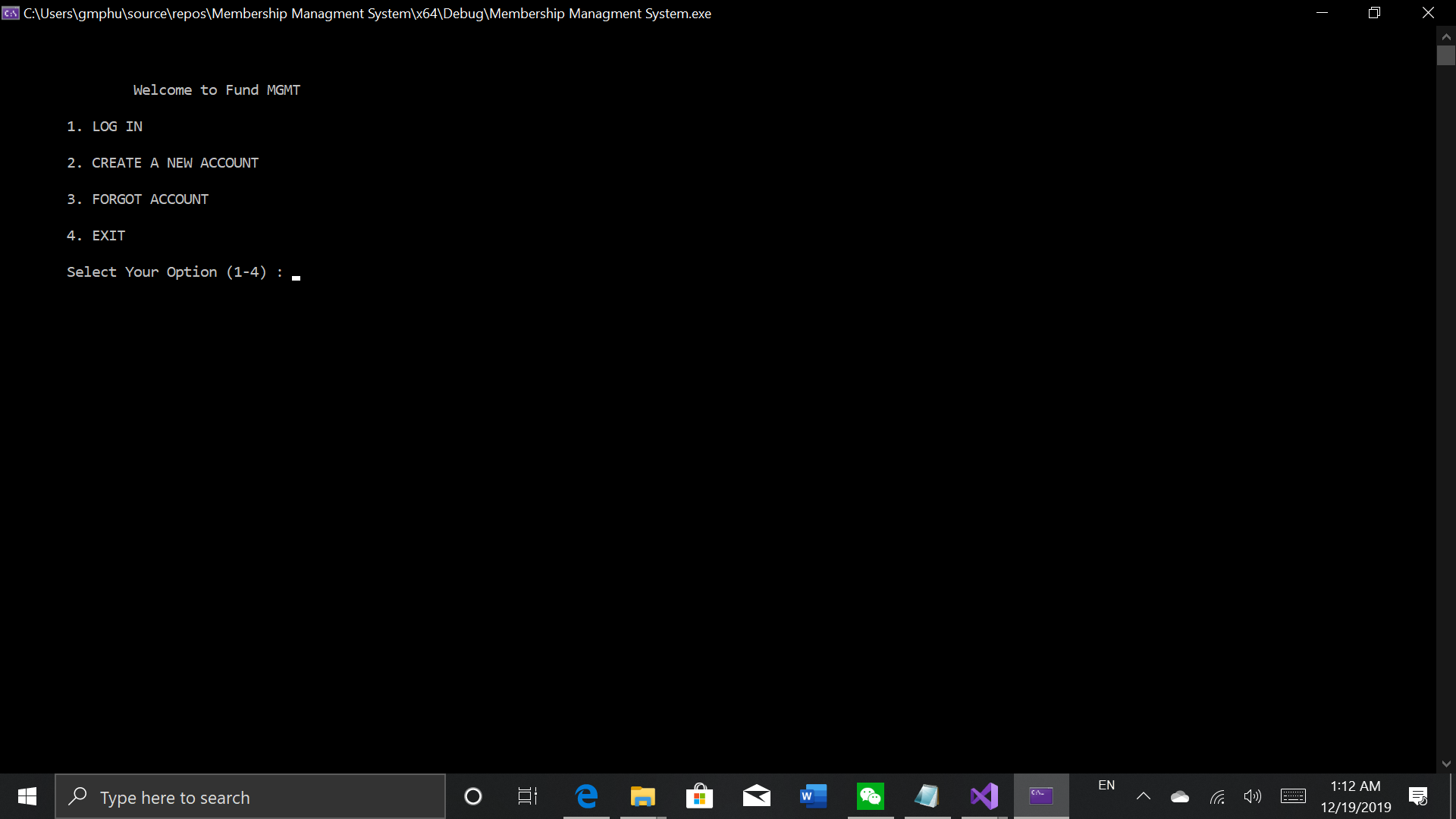
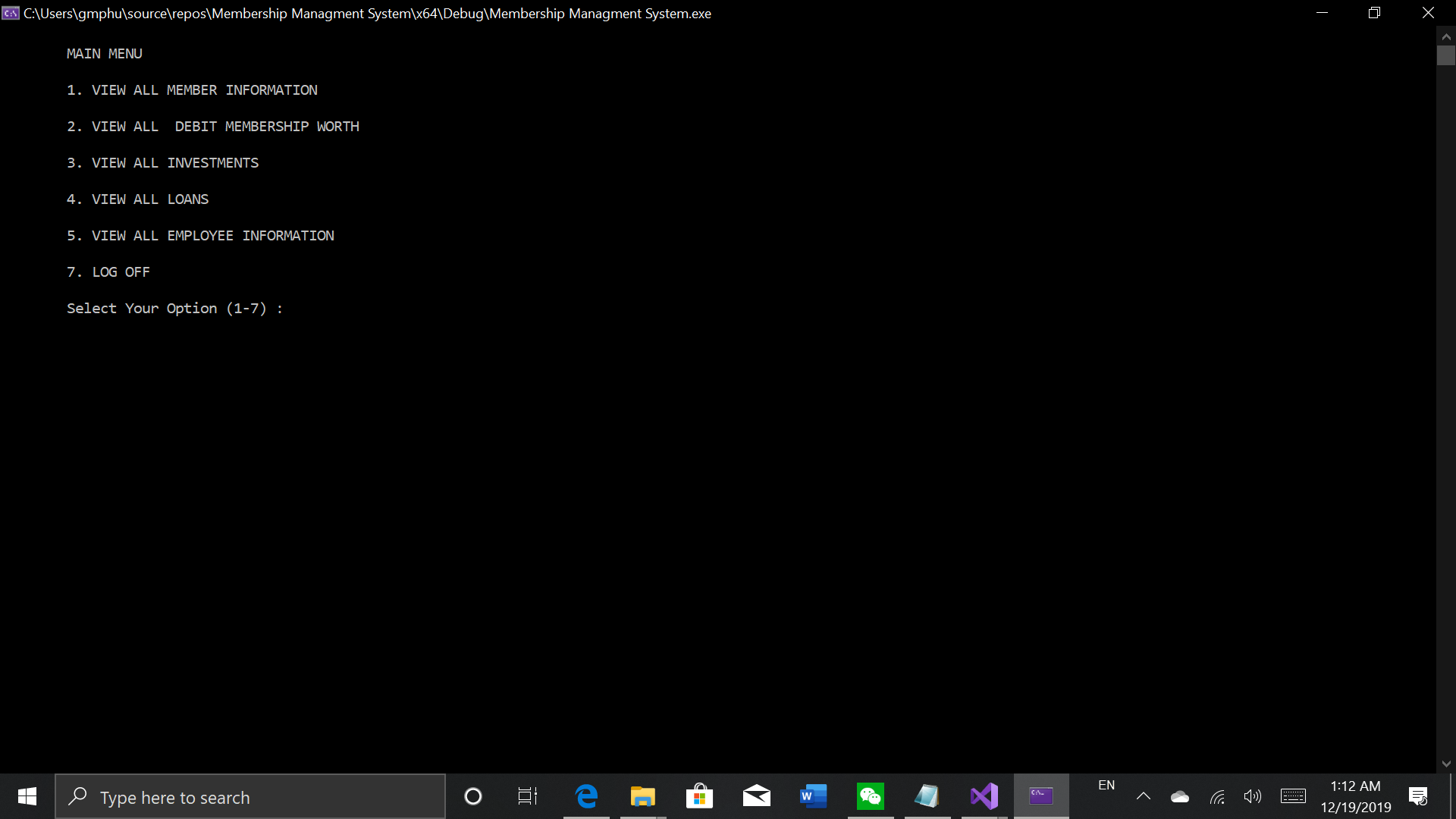
} while (1);

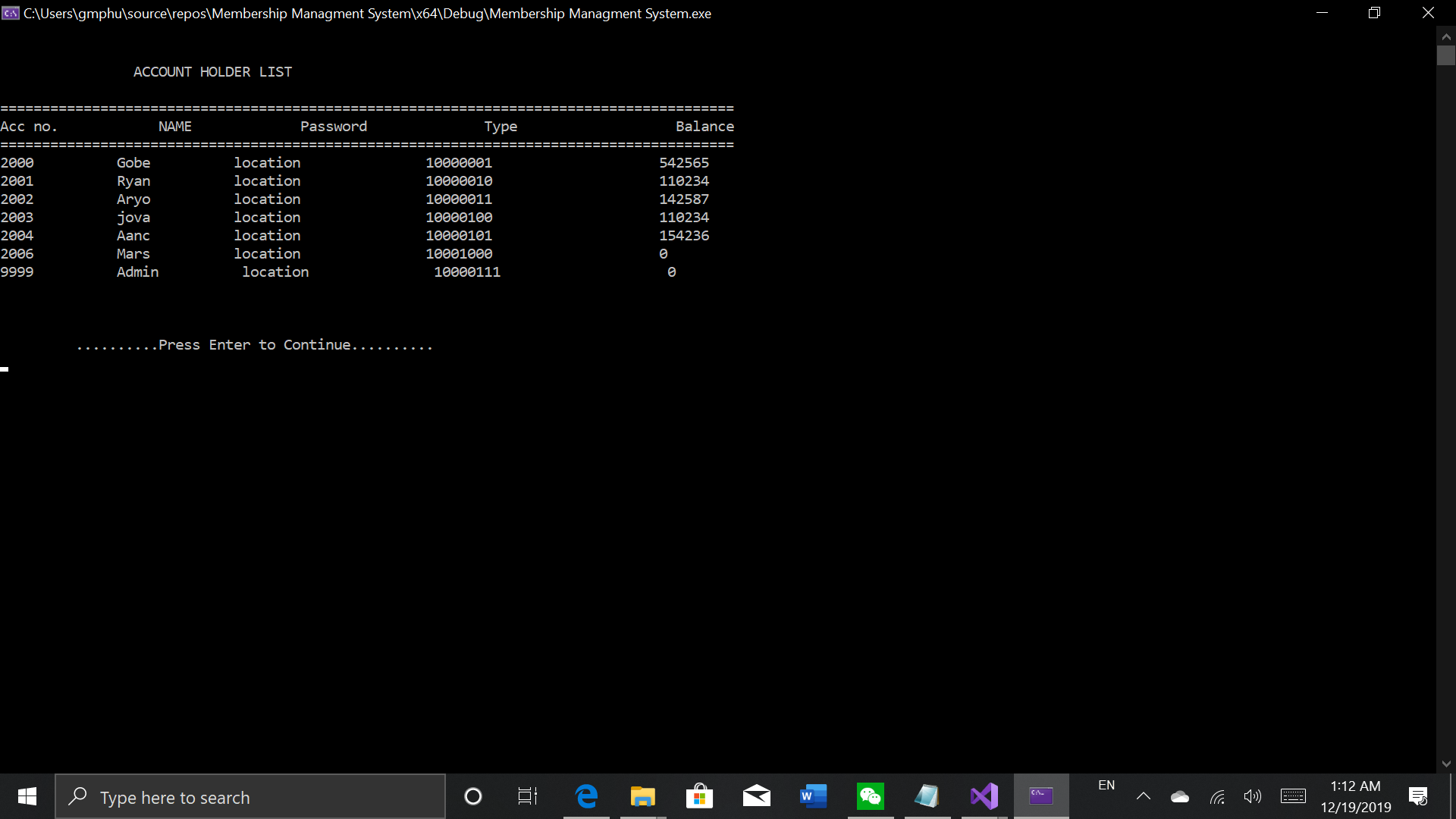
return 0;

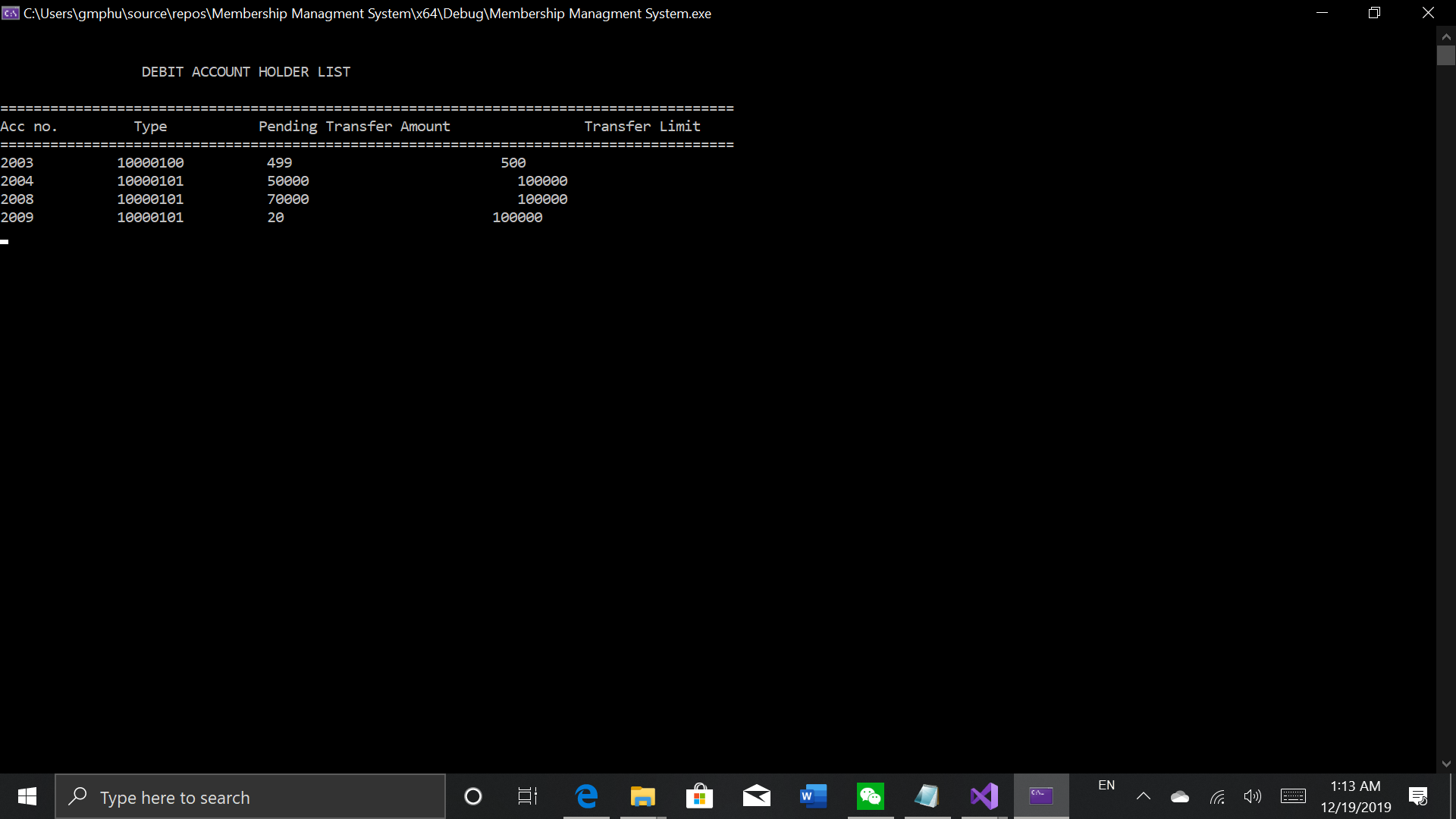
}

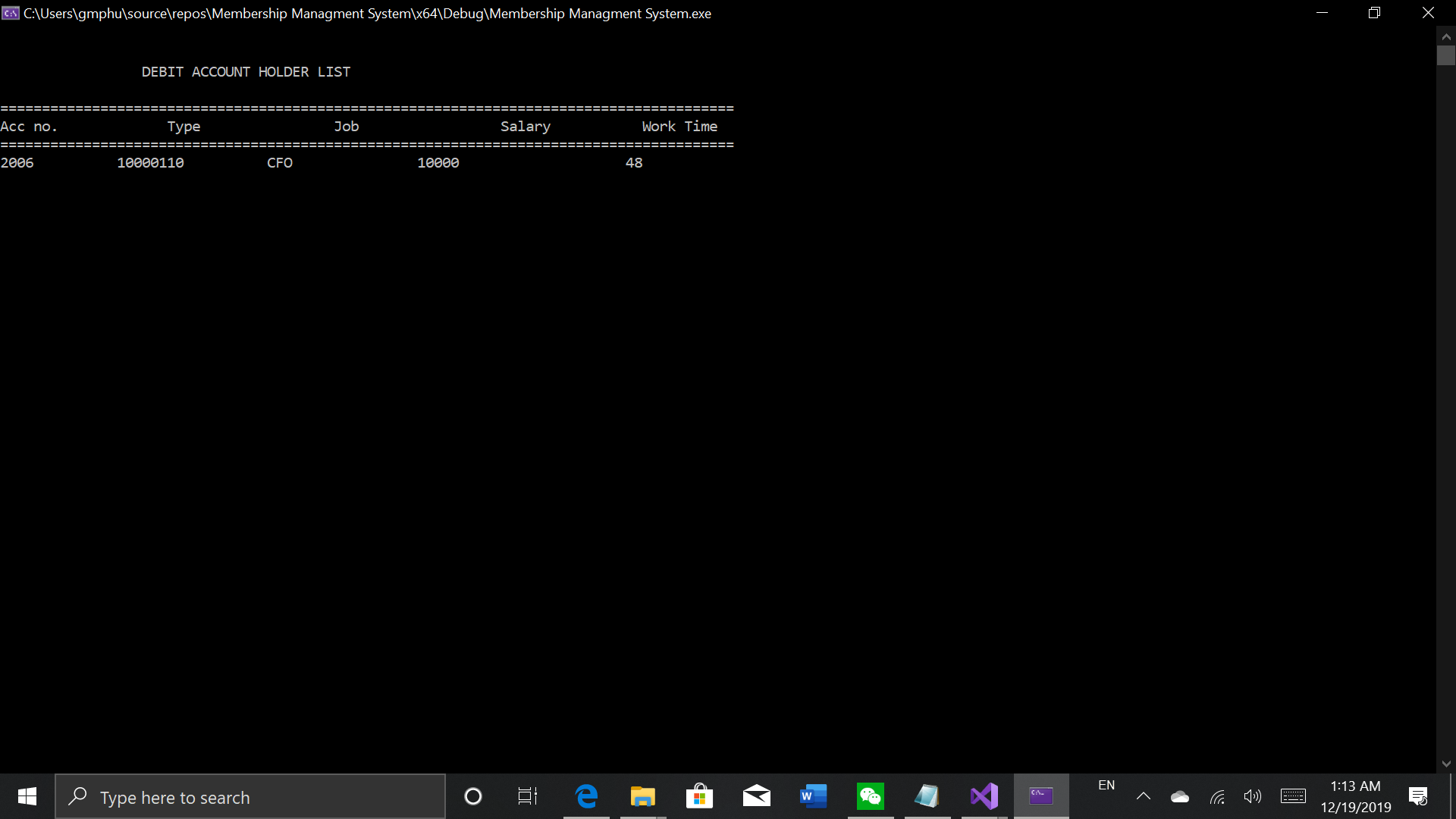
// END OF PROJECT //

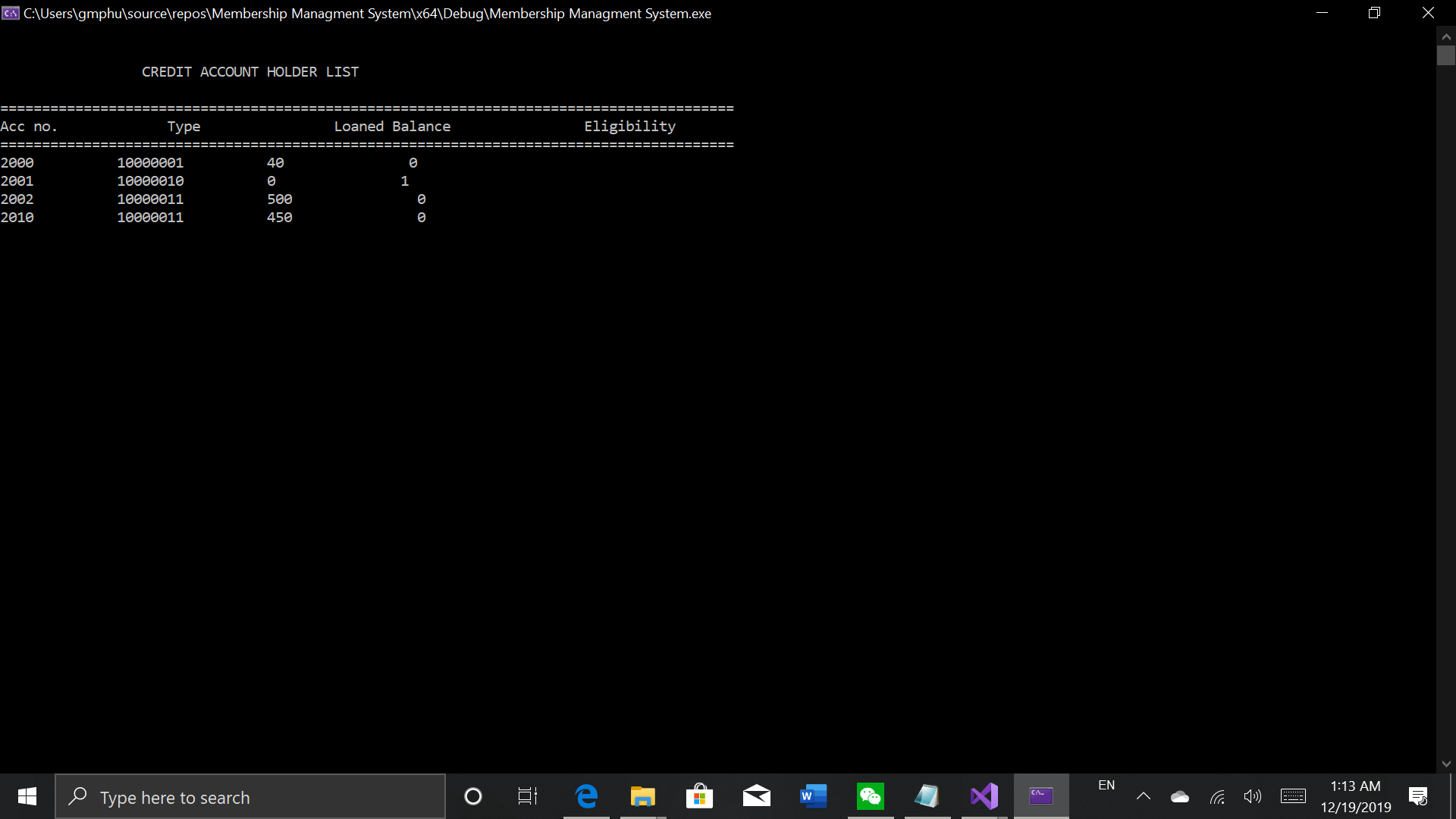
**USER INTERFACE:**

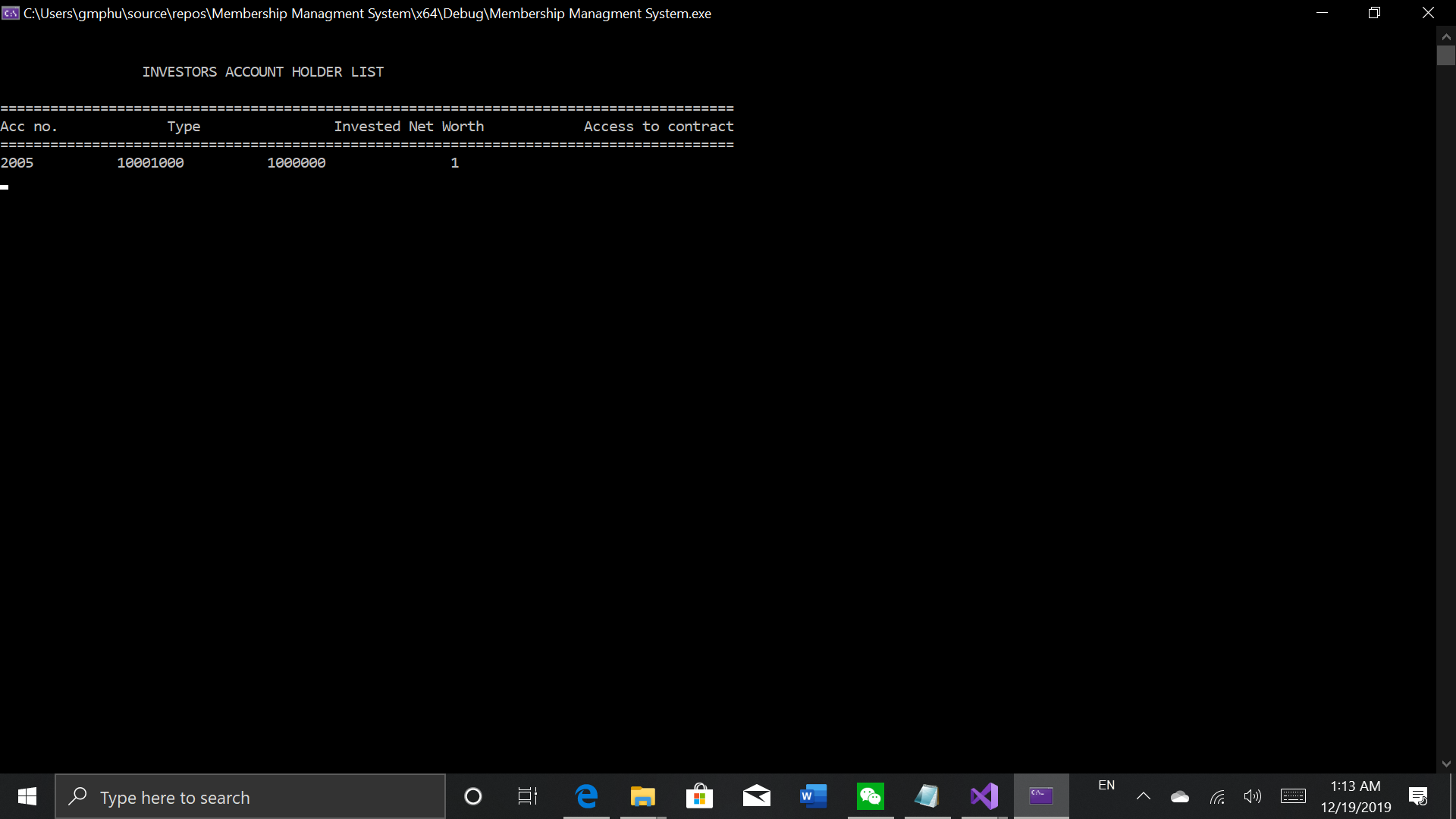
 

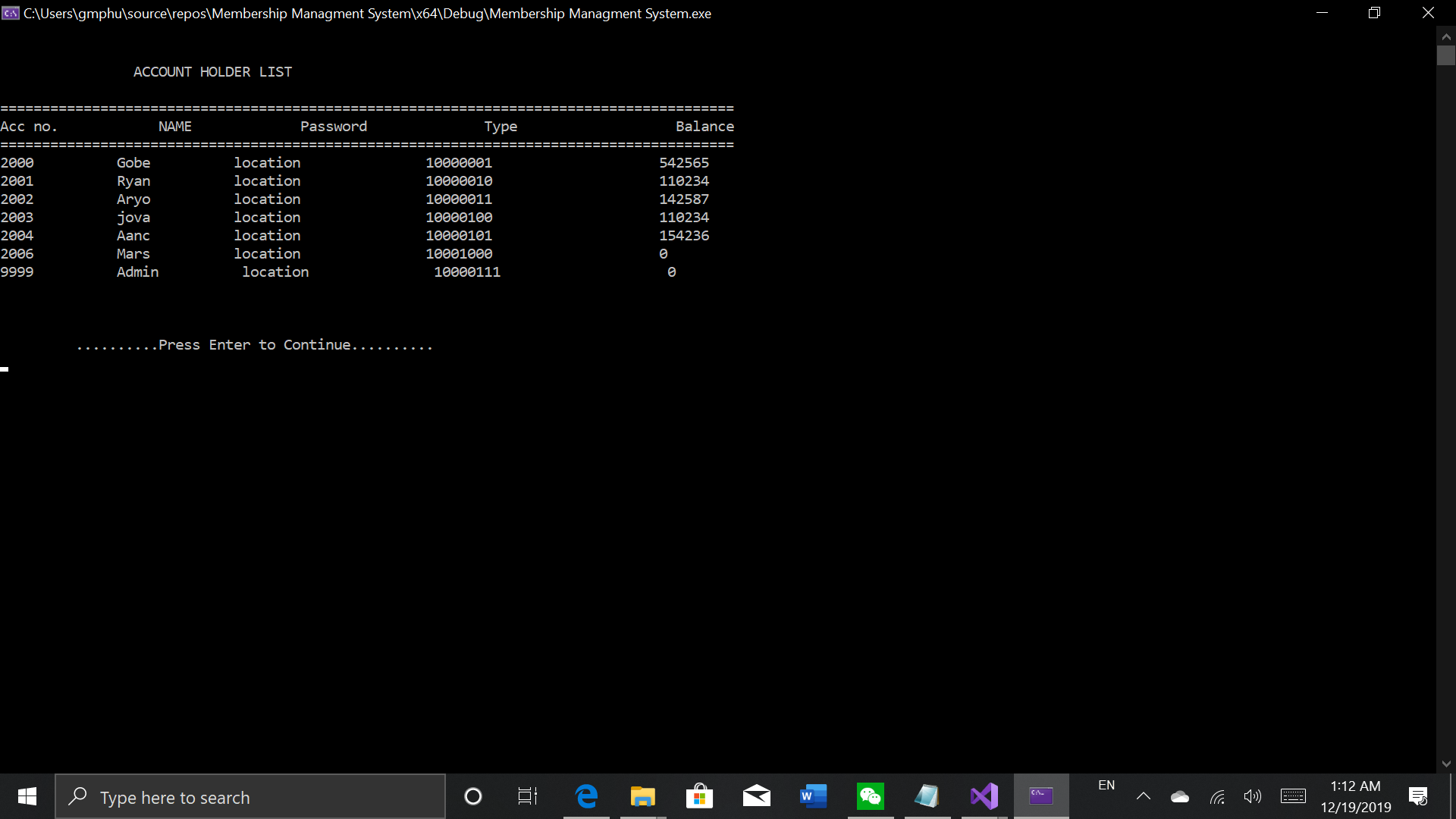
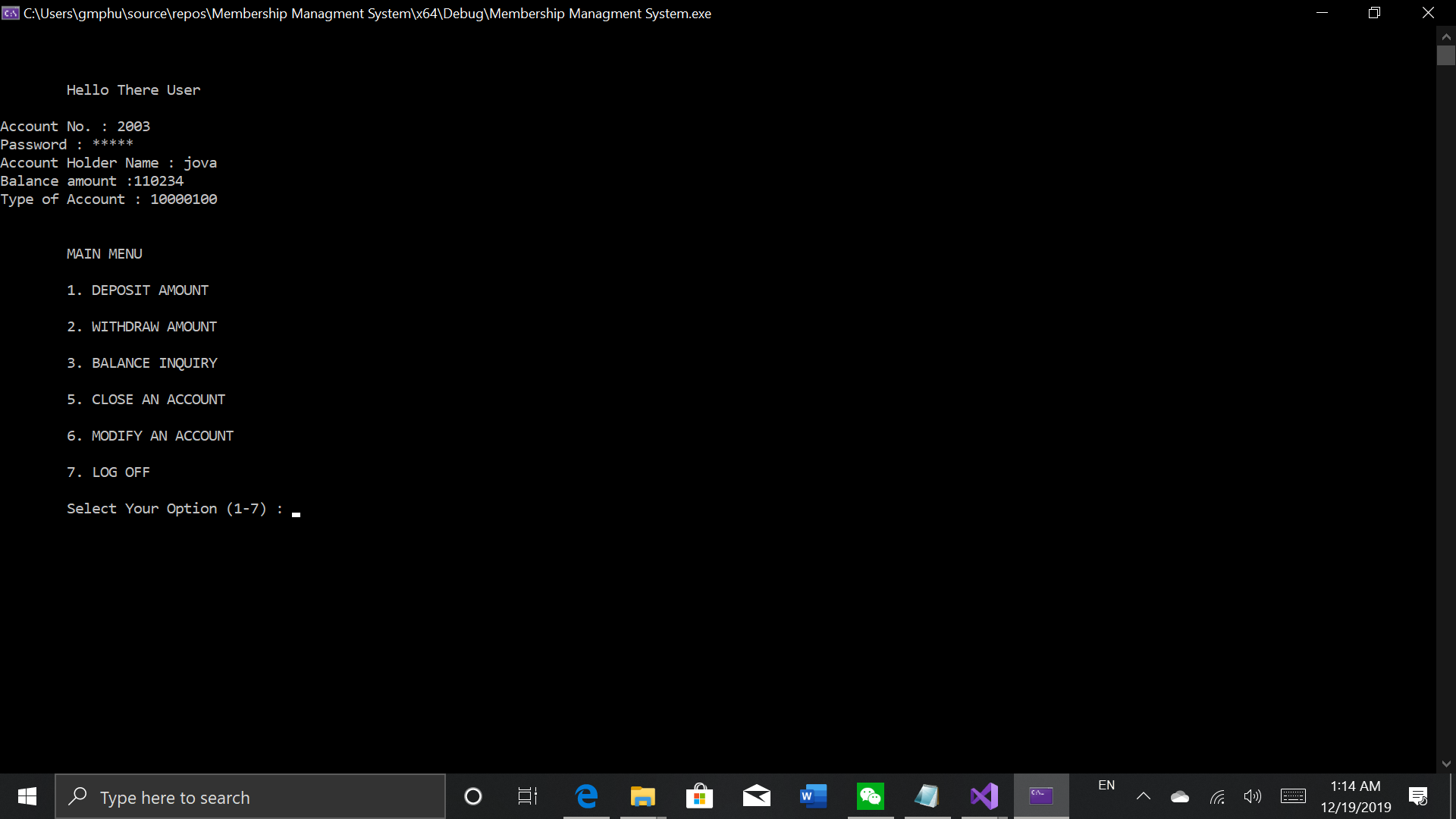












**THANK YOU!**