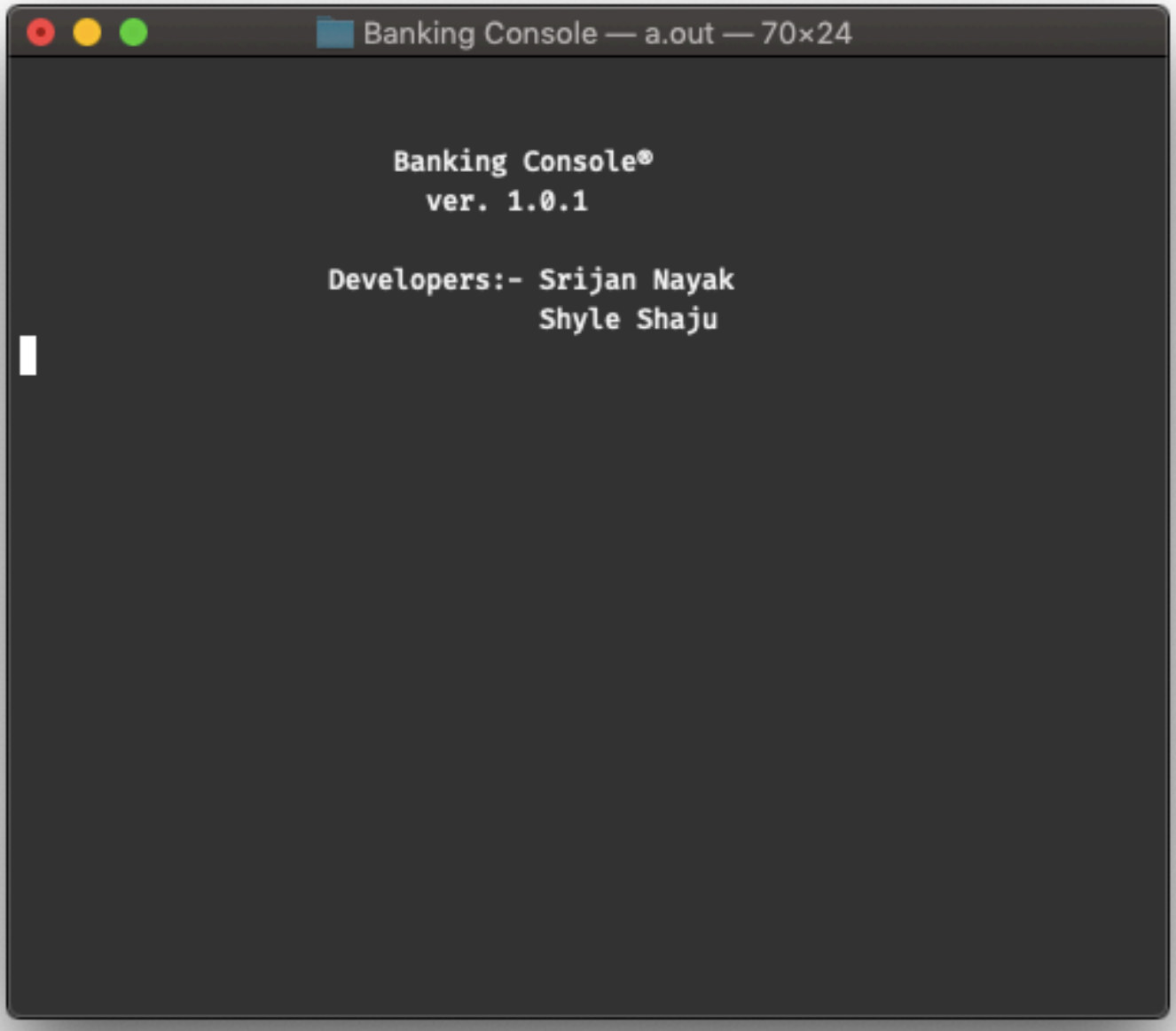
Comp. Sc. Investigatory Project

Banking Console



Acknowledgement

We would like to express our sincere gratitude to our principal Dr. G. Thangadurai as making of this project has been a great learning experience for us. I am thankful to our Computer Science teacher Mrs. Sathya for giving us valuable inputs.

We would also like to thank our parents who have constantly supported us. We also take this opportunity to thank my friends for providing us with constructive criticism.

Index

C++ Overview

C++ is a statically typed, compiled, general-purpose, case-sensitive, free-form programming language that supports procedural, object-oriented, and generic programming.

C++ is regarded as a middle-level language, as it comprises a combination of both high-level and low-level language features.

C++ was developed by Bjarne Stroustrup starting in 1979 at Bell Labs in Murray Hill, New Jersey, as an enhancement to the C language and originally named C with Classes but later it was renamed C++ in 1983.

C++ is a superset of C, and that virtually any legal C program is a legal C++ program.

C++ fully supports object-oriented programming, including the four pillars of object-oriented development :-

- Encapsulation
- Data hiding
- Inheritance
- Polymorphism

Project Overview

This project demonstrates a Banking Console. It is a program written in pure C++ which allows users to access basic features of a banking console.

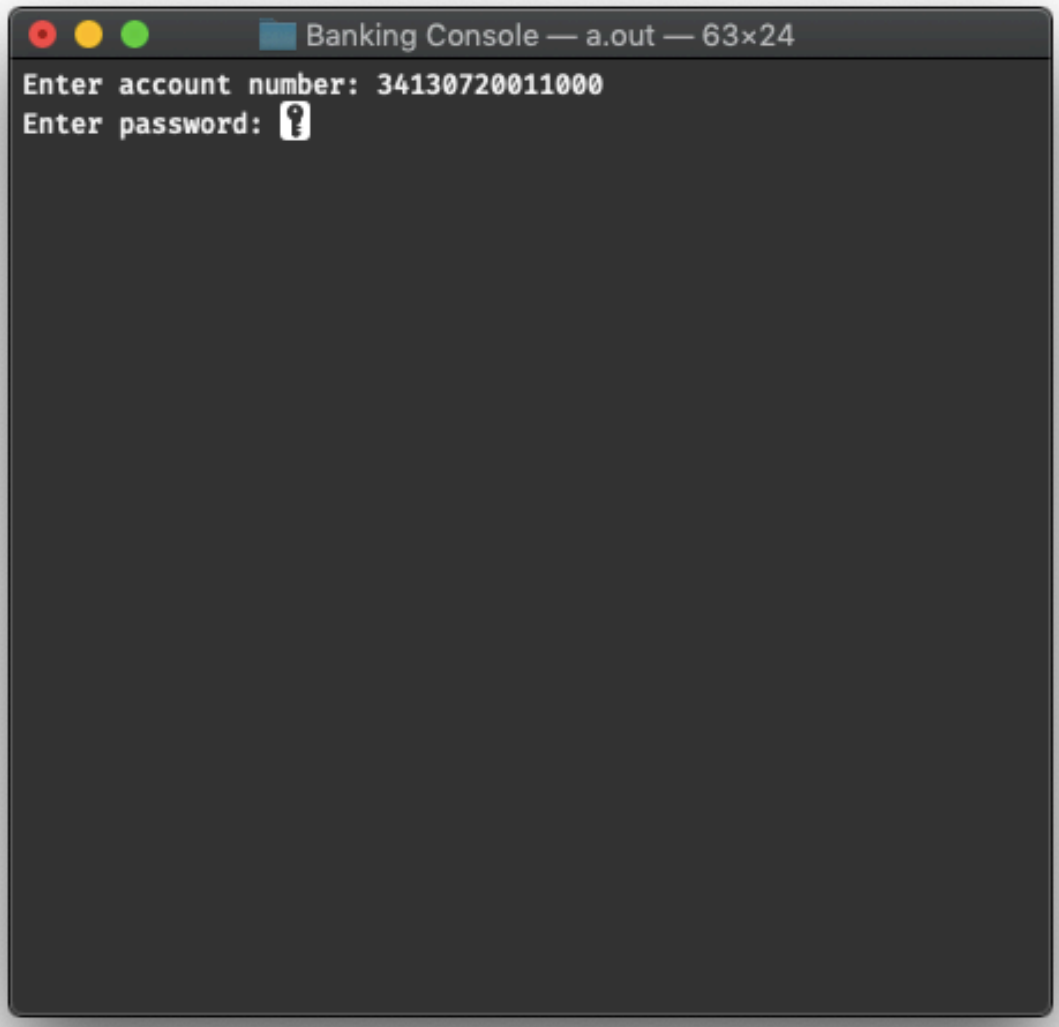
Users have the following options to choose from :-

- Open Account
- Deposit Amount
- Withdraw Amount
- Display All Account Information
- Transfer Amount
- Delete Account

There are two additional options that the users are not shown but the admin for the banking console should know of :-

- Reset Account Code Counter
- Exit Banking Console

Security is one of the main features in this console application. User passwords are never stored. Instead the entered password goes through the sha256 hashing algorithm and this 64 character long hash is stored. Since it is not possible to generate the original password input from a hash value, the console is very secure.



Module Details

Open Account (option: 1)

In this module, the user is prompted to enter necessary details such as his/her name, phone number, address, email id and date of birth, account type, password and balance for opening an account. Each input is checked for validity. The account number is generated by first converting birth date into a string without slashes (/). This string is the prefixed with '34' and suffixed with an account code count which is read from a binary file. The user's entered password undergoes the sha256 hashing algorithm and this hash is split up and stored in a binary file along with the rest of the details.

Deposit Amount (option: 2)

Using this module, the user can enter an amount to be deposited in his/her account. The user is then asked to enter his/her account number and password, both of which are validated with the current account holders' details. After the amount is deposited, the current balance in the account is displayed.

Withdraw Amount (option: 3)

This module allows the user to withdraw a given amount from his/her account after the correct account number and password are entered respectively. After these details are entered, the current account balance and amount to be withdrawn are checked, since the latter cannot exceed the former. If it is a savings account, the console ensures that there is a minimum of thousand rupees left in the account after withdrawal.

Forgot Password (option: 4)

This module allows the user to change his/her account password in case he/she forgot it. In order to do this, the user will have to verify his/her other account details such as account number, email address and date of birth.

Display Account Details (option: 5)

Using this module, the user can view his/her account details after entering the correct account number and password respectively. The details displayed are - name, account number, email address, address, date of birth and current balance.

Transfer Amount (option: 6)

This module allows the user to transfer a given amount from his/her account to another. The account number and password of the donor's account, and the account number of the recipient are required to be entered by the user. The console ensures that the transfer amount does not exceed the balance limitations of the donor's account.

Delete Account (option: 7)

If the user ever has to delete his/her account, he/she can do so with this module. First the user is asked to enter his/her account number, and if a matching account is found then the account is deleted after the user confirms the deletion with his/her password.

Reset Account Code Counter (option: c)

Resets the account code count used for account number generation to '1000'. **Execute this on the first run**.

Exit Banking Console (option: 0)

Use this hidden module to exit the console. **Password** is "30092015".

System Requirements

Hardware Requirements

Processor: Any 64-bit processor

RAM: At least 512 KB

Disk Space: At least 1MB

Software Requirements

Platform: macOS, Windows*, Linux*

Compiler: clang or g++ with support for c++17

Header Files

The following are the header files used in this project. sha256.hpp is a custom header file downloaded from the internet which provides the sha256 hash function.

```
#include "sha256.hpp"
#include <chrono>
#include <fstream>
#include <iostream>
#include <string>
#include <thread>
#ifdef WIN32
#include <windows.h>
#else
#include <termios.h>
#include <unistd.h>
#endif
```

Source Code

Account.hpp contains the Account class definition and includes all the necessary headers files required for the program. It also contains macro definitions and declarations for backend functions.

```
#ifndef BANKINGCONSOLE_ACCOUNT_HPP
#define BANKINGCONSOLE_ACCOUNT_HPP

#include "sha256.hpp"
#include <chrono>
#include <fstream>
#include <iostream>
#include <string>
#include <thread>

#ifdef WIN32
#include <windows.h>
#else
#include <termios.h>
#include <unistd.h>
#endif

using namespace std::this_thread;
using namespace std::chrono;
```

```cpp
#define sleep sleep_for(seconds(3))
#define clearScreen std::cout << "\033[2J\033[1;1H"

void setStdinEcho(bool);
void splitString(const std::string &longString, std::string *stringPart);
void combineString(std::string &longString, std::string *stringPart);

class Account {
  private:
    std::string name;
    std::string accountNumber;
    std::string phoneNumber;
    std::string email;
    std::string addressPart[8];
    std::string hashPart[8];
    std::string accountType;
    char dateOfBirth[11];
    char dateOfBirthString[9];

    void convertDate();
    void generateHash();
    void generateAccountNumber();
    bool notValidName() const;
    bool notValidDateOfBirth() const;
    bool notValidPhoneNumber() const;
    bool notValidEmail() const;

  public:
    double balance;

    const std::string &getAccountNumber() const;
    const std::string &getPhoneNumber() const;
    const std::string &getEmail() const;
    const char *getDateOfBirth() const;
    bool compareHash();
    void inputAccountDetails();
    void displayAccountDetails();
    void depositAmount();
    void withdrawAmount();
    void withdrawAmount(double);
    void changePassword();
};

#endif // BANKINGCONSOLE_ACCOUNT_HPP
```

Account.cpp contains function definitions of Account.hpp.

```
#include "Account.hpp"

/* Stops stdout so password can be entered
 * without the input being shown on the screen.
 */
void setStdinEcho(bool enable = true) {
#ifdef WIN32
    HANDLE hStdin = GetStdHandle(STD_INPUT_HANDLE);
    DWORD mode;
    GetConsoleMode(hStdin, &mode);

    if (!enable)
        mode &= ~ENABLE_ECHO_INPUT;
    else
        mode |= ENABLE_ECHO_INPUT;

    SetConsoleMode(hStdin, mode);

#else
    struct termios tty;
    tcgetattr(STDIN_FILENO, &tty);
    if (!enable)
        tty.c_lflag &= ~ECHO;
    else
        tty.c_lflag |= ECHO;

    (void)tcsetattr(STDIN_FILENO, TCSANOW, &tty);
#endif
}

/* Splits long strings into 8 equal parts so
 * that they can stored in a binary file without
 * exceeding the 80 char width limit.
 */
void splitString(const std::string &longString, std::string stringPart[]) {
    int n = static_cast<int>(longString.length());
    int pre;
    int at;
    for (int i = pre = 0; i < 8; i++) {
        at = (n * i + n) / 8;
        stringPart[i] = longString.substr(static_cast<unsigned long>(pre),
```

```
                                        static_cast<unsigned long>(at - pre));
        pre = at;
    }
}

/* Combines 8 strings into one single string
 * so that the data can be used as normal string
 * in the program.
 */
void combineString(std::string &longString, std::string stringPart[]) {
    for (int i = 0; i < 8; i++) {
        longString += stringPart[i];
    }
}

/* Returns the account number stored in
 * the current object.
 */
const std::string &Account::getAccountNumber() const { return accountNumber; }

/* Returns the user's phone number stored in
 * the current object.
 */
const std::string &Account::getPhoneNumber() const { return phoneNumber; }

/* Returns the user's email address stored in
 * the current object.
 */
const std::string &Account::getEmail() const { return email; }

/* Returns the users date of birth stored in
 * the current object.
 */
const char *Account::getDateOfBirth() const { return dateOfBirth; }

/* Takes in a password input from the user
 * and then generate a hash with the sha256
 * hash function.
 */
void Account::generateHash() {
    std::string passwordHash;
    char pass[17];
    setStdinEcho(false);
    std::cin.getline(pass, 17);
    setStdinEcho(true);
```

```
    std::cout << std::endl;
    passwordHash = sha256(pass);
    splitString(passwordHash, hashPart);
}

/* Converts a date string of the format
 * "dd/mm/yyyy" to a string of the format
 * "ddmmyyyy" which can be used to generate
 * the account number.
 */
void Account::convertDate() {
    int i = 0;
    int j = 0;
    while (i < 11) {
        while (j < 9) {
            if (isdigit(dateOfBirth[i])) {
                dateOfBirthString[j] = dateOfBirth[i];
                j++;
                break;
            } else {
                break;
            }
        }
        i++;
        dateOfBirthString[8] = char(0);
    }
    generateAccountNumber();
}

/* Generates the account number by prefixing
 * the converted date of birth string with
 * "34" and suffixing it with the current
 * ACCOUNT_CODE that is stored in accCode.dat
 */
void Account::generateAccountNumber() {
    int ACCOUNT_CODE;
    std::fstream file;
    file.open("data/accCode.dat", std::ios::binary | std::ios::in);
    file.read((char *)&ACCOUNT_CODE, sizeof(ACCOUNT_CODE));
    file.close();
    accountNumber = "34";
    accountNumber += std::string(dateOfBirthString);
    accountNumber += std::to_string(ACCOUNT_CODE);
    ACCOUNT_CODE++;
    file.open("data/accCode.dat", std::ios::binary | std::ios::out);
```

```
    file.write((char *)&ACCOUNT_CODE, sizeof(ACCOUNT_CODE));
    file.close();
}

/* Takes in a password input from the user
 * and then generate a hash with the sha256
 * hash function and compares with the hash
 * stored in the current object.
 */
bool Account::compareHash() {
    bool confirmation;
    char pass[17];
    std::string passwordHash;
    std::string hash;
    combineString(passwordHash, hashPart);
    setStdinEcho(false);
    std::cin.getline(pass, 17);
    setStdinEcho(true);
    std::cout << std::endl;
    hash = sha256(pass);
    confirmation = hash == passwordHash;
    return confirmation;
}

/* Takes in the input of all necessary
 *  details for creating an account
 *  from the user and validates them.
 */
void Account::inputAccountDetails() {
    std::cout << "Enter name or enter 'e' to exit to menu: ";
    std::cin.ignore();
getName:
    std::getline(std::cin, name);
    if (name == "e") {
        return;
    }
    if (notValidName()) {
        std::cout << "Please enter a valid name or enter 'e' to exit to menu: ";
        goto getName;
    }

    std::cout << "Enter date of birth in dd/mm/yyyy format: ";
getDate:
    std::cin.getline(dateOfBirth, 11);
    if (notValidDateOfBirth()) {
```

```
        std::cout << "Please enter date in dd/mm/yyyy format: ";
        goto getDate;
    }
    convertDate();

    std::cout << "Enter phone number without any prefix: ";
getPhoneNumber:
    std::getline(std::cin, phoneNumber);
    if (notValidPhoneNumber()) {
        std::cout << "Please enter valid phone number without any prefix: ";
        goto getPhoneNumber;
    }

    std::cout << "Enter email address: ";
getEmail:
    std::getline(std::cin, email);
    if (notValidEmail()) {
        std::cout << "Please enter a valid email address: ";
        goto getEmail;
    }

    std::cout << "Enter address: ";
    std::string address;
getAddress:
    std::getline(std::cin, address);
    if (address.empty()) {
        std::cout << "Please enter a valid address: ";
        goto getAddress;
    }
    splitString(address, addressPart);

    std::cout << "Enter password (maximum 16 characters): ";
getPassword:
    generateHash();
    std::cout << "Confirm password: ";
    if (!compareHash()) {
        std::cout << "Passwords don't match!" << std::endl;
        std::cout << "Please re-enter password: ";
        goto getPassword;
    }

    std::cout << "Enter account type (Current/Savings): ";
getAccountType:
    std::getline(std::cin, accountType);
    if (!(accountType == "Current" || accountType == "Savings")) {
```

```
        std::cout << "Invalid input! Please enter valid account type: ";
        goto getAccountType;
    }

    std::cout << "Enter starting balance: ";
getAmount:
    std::cin >> balance;
    if (balance < 0) {
        std::cout << "Please enter a valid balance: ";
        goto getAmount;
    }

    clearScreen;
    std::cout << "Account created successfully" << std::endl << std::endl;
    std::cin.ignore();
    displayAccountDetails();
}

/* Provides validation for name.
 * Returns true if name is not
 * valid.
 */
bool Account::notValidName() const {
    bool notValid = false;
    for (char ch : name) {
        if (!(isalpha(ch) || isspace(ch))) {
            notValid = true;
        }
    }
    return notValid;
}

/* Provides validation for date of birth.
 * Returns true if date of birth is not
 * valid.
 */
bool Account::notValidDateOfBirth() const {
    return (strlen(dateOfBirth) != 10) || (dateOfBirth[2] != '/') ||
           (dateOfBirth[5] != '/') || (!isdigit(dateOfBirth[0])) ||
           (!isdigit(dateOfBirth[1])) || (!isdigit(dateOfBirth[3])) ||
           (!isdigit(dateOfBirth[4])) || (!isdigit(dateOfBirth[6])) ||
           (!isdigit(dateOfBirth[7])) || (!isdigit(dateOfBirth[8])) ||
           (!isdigit(dateOfBirth[9]));
}
```

```cpp
/* Provides validation for phone number.
 * Returns true if phone number is not
 * valid.
 */
bool Account::notValidPhoneNumber() const {
    bool notValid = false;
    for (char ch : phoneNumber) {
        if (!isdigit(ch)) {
            notValid = true;
        }
    }
    if (phoneNumber.length() != 10) {
        notValid = true;
    }
    return notValid;
}

/* Provides validation for email address.
 * Returns true if email address is not
 * valid.
 */
bool Account::notValidEmail() const {
    bool notValid = true;
    for (char ch : email) {
        if (ch == '@') {
            notValid = false;
        }
    }
    return notValid;
}

/* Displays all account details of the user.
 * Does not display sensitive info like the
 * password.
 */
void Account::displayAccountDetails() {
    std::string address;
    combineString(address, addressPart);
    std::cout << "\t\tAccount Details" << std::endl;
    std::cout << "\t\t---------------" << std::endl;
    std::cout << "Account number: " << accountNumber << std::endl;
    std::cout << "Name: " << name << std::endl;
    std::cout << "Date of birth: " << dateOfBirth << std::endl;
    std::cout << "Phone number: " << phoneNumber << std::endl;
    std::cout << "Email address: " << email << std::endl;
```

```
    std::cout << "Address: " << address << std::endl;
    std::cout << "Account type: " << accountType << std::endl;
    std::cout << "Current balance: Rs. " << balance << std::endl;
    std::cout << std::endl;
    std::cout << "Press ENTER after noting down the details." << std::endl;
    std::cin.get();
}

/* Allows the user to deposit funds in his/her
 * account.
 */
void Account::depositAmount() {
    double amount;
    std::cout << "Enter amount to be deposited: ";
    std::cin >> amount;
    std::cin.ignore();
    std::cout << "Enter password: ";
getPassword:
    bool confirmation = compareHash();
    if (!confirmation) {
        std::cout << "Incorrect password!" << std::endl;
        std::cout << "Please re-enter password: ";
        goto getPassword;
    }
    std::cout << "Rs. " << amount << " deposited successfully." << std::endl;
    balance += amount;
    std::cout << "Current balance: " << balance << std::endl;
    sleep;
}

/* Allows the user to withdraw funds from his/her
 * account. If the account type is 'Current', then
 * withdrawal request is rejected if final minimum
 * balance is less than or equal to Rs. 1000.
 */
void Account::withdrawAmount() {
    double amount;
    std::cout << "Enter amount to be withdrawn: ";
getAmount:
    std::cin >> amount;
    std::cin.ignore();
    if (accountType == "Savings" && (balance - amount) <= 1000) {
        std::cout << "Savings account must have at least a minimum balance of "
                     "Rs. 1000!"
                  << std::endl;
```

```
        std::cout << "Please renter amount to be withdrawn: ";
        goto getAmount;
    } else if (amount > balance) {
        std::cout << "Insufficient credits!" << std::endl;
        std::cout << "Please re-enter amount to be withdrawn: ";
        goto getAmount;
    }
    std::cout << "Enter password: ";
getPassword:
    bool confirmation = compareHash();
    if (!confirmation) {
        std::cout << "Incorrect password!" << std::endl;
        std::cout << "Please re-enter password: ";
        goto getPassword;
    }
    std::cout << "Rs. " << amount << " withdrawn successfully." << std::endl;
    balance -= amount;
    std::cout << "Current balance: " << balance << std::endl;
    sleep;
}

/* Used to withdraw any remaining balance
 * when account is being deleted.
 */
void Account::withdrawAmount(double lastBalance) {
    balance -= lastBalance;
    std::cout << "Remaining balance Rs. " << lastBalance
              << " withdrawn successfully." << std::endl;
    sleep;
}

/* Used to change the password for the
 * current account object.
 */
void Account::changePassword() {
    std::cout << "Enter new password: ";
getPassword:
    generateHash();
    std::cout << "Confirm password: ";
    if (!compareHash()) {
        std::cout << "Passwords don't match!" << std::endl;
        std::cout << "Please re-enter password: ";
        goto getPassword;
    }
}
```

main.cpp contains the main function. It also contains definitions for frontend functions like splashScreen() and menuDisplay(). The hash for the exit banking console password is also stored in this file.

```
/* main.cpp
 * Banking Console
 *
 * Created by Srijan Nayak on 16/04/2018.
 * Copyright © 2018 Srijan Nayak, Shyle Shaju. All rights reserved.
 */

#include "Account.hpp"

std::string masterKeyHash = // Hash of master key for exiting the console.
    "491203c64506797dc07ce337d86f0355b0660df01f12c6b5de0304aebbc01fbd";

/* Displays the splash screen briefly when
 * the console is launched.
 */
void splashScreen() {
    std::cout << std::endl;
    std::cout << std::endl;
    std::cout << "                          Banking Console®" << std::endl;
    std::cout << "                            ver. 1.0.1" << std::endl;
    std::cout << std::endl;
    std::cout << "                    Developers:- Srijan Nayak" << std::endl;
    std::cout << "                                 Shyle Shaju" << std::endl;
    sleep;
```

```
    std::cout.flush();
}

/* Displays the banking console main menu
 * and prompts the user to enter the option
 * number.
 */
void menuDisplay() {
    std::cout << std::endl;
    std::cout << "  1. Open Account     5. Display All Account Information"
              << std::endl;
    std::cout << "  2. Deposit Amount   6. Transfer Amount" << std::endl;
    std::cout << "  3. Withdraw Amount  7. Delete Account" << std::endl;
    std::cout << "  4. Forgot Password" << std::endl;
    std::cout << std::endl;
    std::cout << "Enter preferred option number: ";
    std::cout.flush();
}

int main() {
    Account a;
    Account a1;
    std::fstream file;
    std::fstream file1;
    char choice;
    char temp[25];
    std::string temp1;
    long pos = 0;
    long pos1 = 0;
    double amount;
    bool foundFlag = false;
    bool foundFlag1 = false;
    bool deletionConfirmation = false;
    const int ACCOUNT_CODE = 1000;

    clearScreen;
    splashScreen();
    while (true) {
        clearScreen;
        menuDisplay();
        std::cin >> choice;
        clearScreen;

        switch (choice) {
        case 'c': // Case for hidden menu option 'Reset Account Code Counter'.
```

```
    file.open("data/accCode.dat", std::ios::binary | std::ios::out);
    file.write((char *)&ACCOUNT_CODE, sizeof(ACCOUNT_CODE));
    file.close();
    break;

case '0': // Case for hidden menu option 'Exit Banking Console'.
    std::cout << "Enter master key to exit the console: ";
    setStdinEcho(false);
    std::cin.ignore();
    std::cin.getline(temp, 25);
    setStdinEcho(true);
    std::cout << std::endl;
    if (sha256(temp) == masterKeyHash) {
        clearScreen;
        return 0;
    } else {
        std::cout << "Incorrect password!" << std::endl;
    }
    break;

case '1': // Case for menu option 'Open Account'.
    file.open("data/records.dat",
              std::ios::binary | std::ios::out | std::ios::app);
    a.inputAccountDetails();
    file.write((char *)&a, sizeof(a));
    file.close();
    break;

case '2': // Case for menu option 'Deposit Amount'.
    file.open("data/records.dat",
              std::ios::binary | std::ios::in | std::ios::out);
    std::cout << "Enter account number: ";
    std::cin.ignore();
    std::getline(std::cin, temp1);
    while (file) {
        pos = file.tellg();
        file.read((char *)&a, sizeof(a));
        if (temp1 == a.getAccountNumber()) {
            a.depositAmount();
            file.seekp(pos);
            file.write((char *)&a, sizeof(a));
            foundFlag = true;
            break;
        }
    }
```

```
    if (!foundFlag) {
        std::cout << "No account with that account number!"
                  << std::endl;
        sleep;
    }
    file.close();
    break;

case '3': // Case for menu option 'Withdraw Amount'.
    file.open("data/records.dat",
              std::ios::binary | std::ios::in | std::ios::out);
    std::cout << "Enter account number: ";
    std::cin.ignore();
    std::getline(std::cin, temp1);
    while (file) {
        pos = file.tellg();
        file.read((char *)&a, sizeof(a));
        if (temp1 == a.getAccountNumber()) {
            a.withdrawAmount();
            file.seekp(pos);
            file.write((char *)&a, sizeof(a));
            foundFlag = true;
            break;
        }
    }
    if (!foundFlag) {
        std::cout << "No account with that account number!"
                  << std::endl;
        sleep;
    }
    file.close();
    break;

case '4': // Case for menu option "Forgot Password".
    file.open("data/records.dat",
              std::ios::binary | std::ios::in | std::ios::out);
    std::cout << "Enter account number: ";
    std::cin.ignore();
    std::getline(std::cin, temp1);
    while (file) {
        pos = file.tellg();
        file.read((char *)&a, sizeof(a));
        if (temp1 == a.getAccountNumber()) {
            foundFlag = true;
            std::cout
```

```
                << "Enter phone number associated with the account: ";
            std::getline(std::cin, temp1);
            if (temp1 == a.getPhoneNumber()) {
                std::cout << "Enter email address associated with the "
                             "account: ";
                std::getline(std::cin, temp1);
                if (temp1 == a.getEmail()) {
                    std::cout << "Enter date of birth: ";
                    std::cin.getline(temp, 11);
                    if (strcmp(temp, a.getDateOfBirth()) == 0) {
                        a.changePassword();
                        file.seekp(pos);
                        file.write((char *)&a, sizeof(a));
                        foundFlag1 = true;
                    }
                }
            }
            if (!foundFlag1) {
                std::cout << "Incorrect detail entered!" << std::endl;
                sleep;
            }
            break;
        }
    }
    if (!foundFlag) {
        std::cout << "No account with that account number!"
                  << std::endl;
        sleep;
    }
    file.close();
    break;

case '5': // Case for menu option 'Display All Account Information'.
    file.open("data/records.dat", std::ios::binary | std::ios::in);
    std::cout << "Enter account number: ";
    std::cin.ignore();
    std::getline(std::cin, temp1);
    while (file.read((char *)&a, sizeof(a))) {
        if (temp1 == a.getAccountNumber()) {
            foundFlag = true;
            std::cout << "Enter password: ";
            if (a.compareHash()) {
                clearScreen;
                a.displayAccountDetails();
                break;
```

```
            } else {
                std::cout << "Incorrect password!" << std::endl;
                sleep;
                break;
            }
        }
    }
    if (!foundFlag) {
        std::cout << "No account with that account number!"
                  << std::endl;
        sleep;
    }
    file.close();
    break;

case '6': // Case for menu option 'Transfer Account'.
    file.open("data/records.dat",
              std::ios::binary | std::ios::in | std::ios::out);
    std::cout << "Enter your account number: ";
    std::cin.ignore();
    std::getline(std::cin, temp1);
    while (file) {
        pos = file.tellg();
        file.read((char *)&a, sizeof(a));
        if (temp1 == a.getAccountNumber()) {
            foundFlag = true;
            std::cout << "Enter password: ";
            if (a.compareHash()) {
                break;
            } else {
                std::cout << "Incorrect password!" << std::endl;
                sleep;
                break;
            }
        }
    }
    if (!foundFlag) {
        std::cout << "No account with that account number!"
                  << std::endl;
        sleep;
    } else {
        std::cout << "Enter recipient's account number: ";
        std::getline(std::cin, temp1);
        file.seekg(file.beg);
        while (file) {
```

```
            pos1 = file.tellg();
            file.read((char *)&a1, sizeof(a1));
            if (temp1 == a1.getAccountNumber()) {
                foundFlag1 = true;
                std::cout << "Enter amount to be transferred: ";
            getAmount:
                std::cin >> amount;
                if (amount > a.balance) {
                    std::cout << "Insufficient credits!" << std::endl;
                    std::cout << "Enter amount again: " << std::endl;
                    goto getAmount;
                } else {
                    a1.balance += amount;
                    a.balance -= amount;
                    std::cout << "Rs." << amount
                              << " transferred successfully to "
                              << temp1 << std::endl;
                    file.seekp(pos);
                    file.write((char *)&a, sizeof(a));
                    file.seekp(pos1);
                    file.write((char *)&a1, sizeof(a1));
                    sleep;
                    break;
                }
            }
        }
        if (!foundFlag1) {
            std::cout << "No account with that account number!"
                      << std::endl;
            sleep;
        }
    }
    file.close();
    break;

case '7': // Case for menu option 'Delete Account'.
    file.open("data/records.dat", std::ios::binary | std::ios::in);
    file1.open("data/temp.dat", std::ios::binary | std::ios::out);
    std::cout << "Enter account number: ";
    std::cin.ignore();
    std::getline(std::cin, temp1);
    while (file.read((char *)&a, sizeof(a))) {
        if (temp1 == a.getAccountNumber()) {
            foundFlag = true;
            std::cout << "Enter password to confirm deletion of your "
```

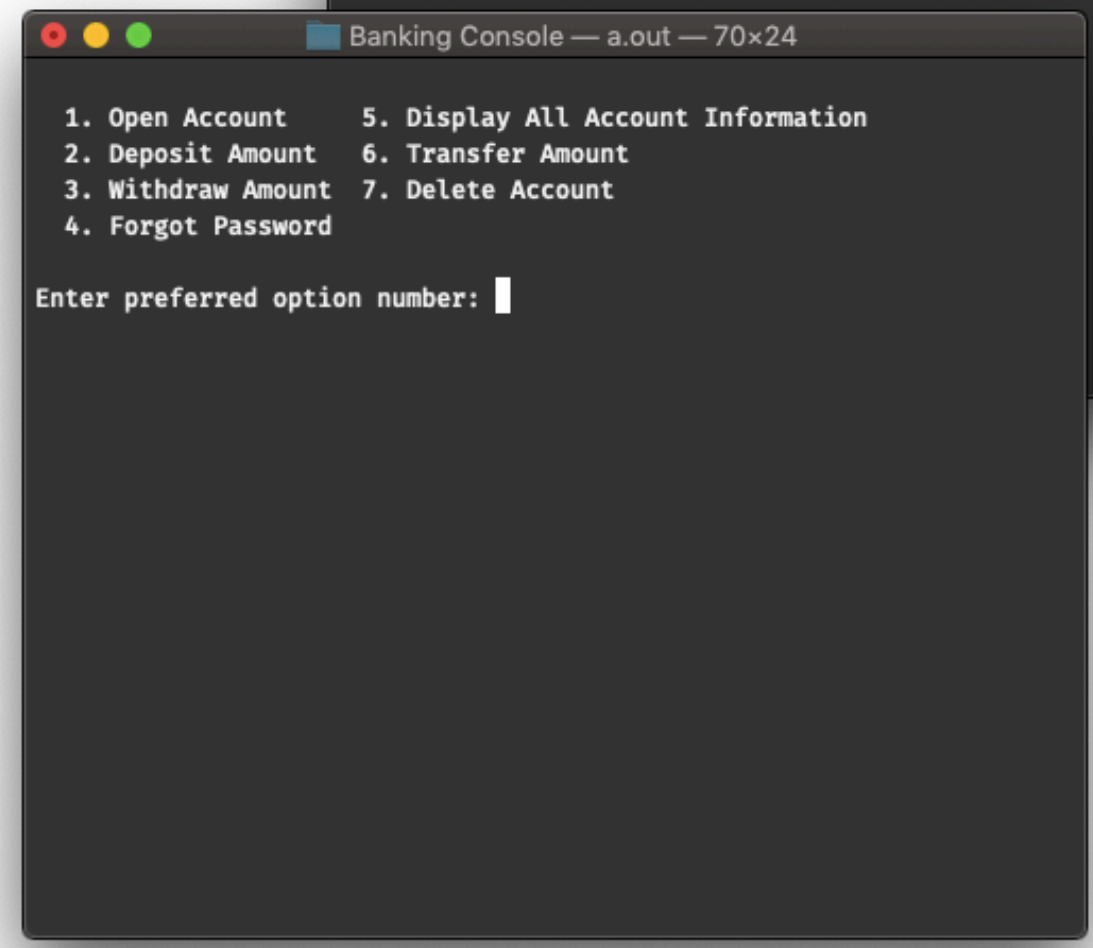
```
                                "account: ";
                    if (a.compareHash()) {
                        a.withdrawAmount(a.balance);
                        file.seekg(file.beg);
                        while (file.read((char *)&a, sizeof(a))) {
                            if (temp1 == a.getAccountNumber()) {
                                std::cout << "Account successfully deleted!"
                                          << std::endl;
                                sleep;
                            } else {
                                file1.write((char *)&a, sizeof(a));
                            }
                            deletionConfirmation = true;
                        }
                        break;
                    } else {
                        std::cout << "Incorrect password!" << std::endl;
                        sleep;
                        break;
                    }
                }
            }
            if (!foundFlag) {
                std::cout << "No account with that account number!"
                          << std::endl;
                sleep;
            }
            if (deletionConfirmation) {
                remove("data/records.dat");
                rename("data/temp.dat", "data/records.dat");
            } else {
                remove("data/temp.dat");
            }
            file.close();
            file1.close();
            break;

        default:
            std::cout << "Invalid choice!" << std::endl;
            sleep;
        }
        foundFlag = false;  // Resets the state of all boolean check flags to
        foundFlag1 = false; // false.
    }
}
```

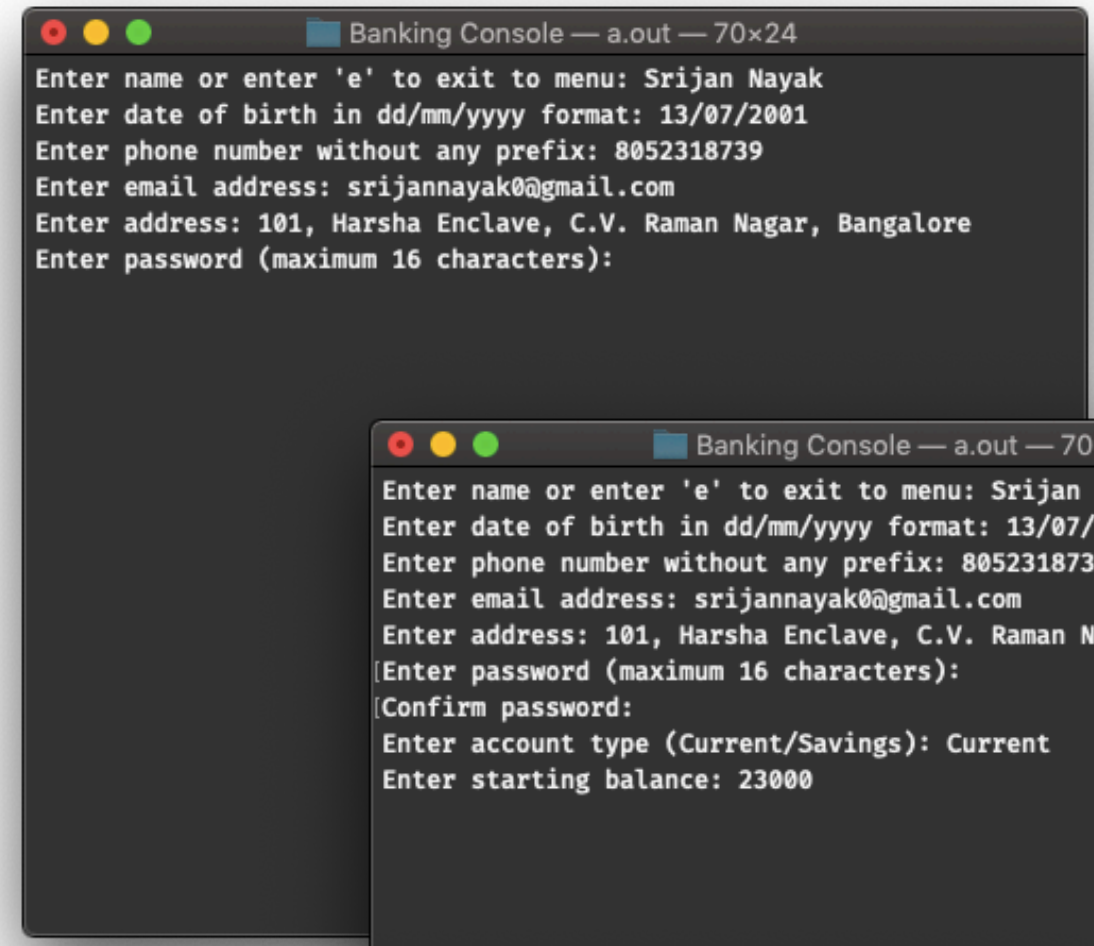
Console Screenshots

Splash Screen

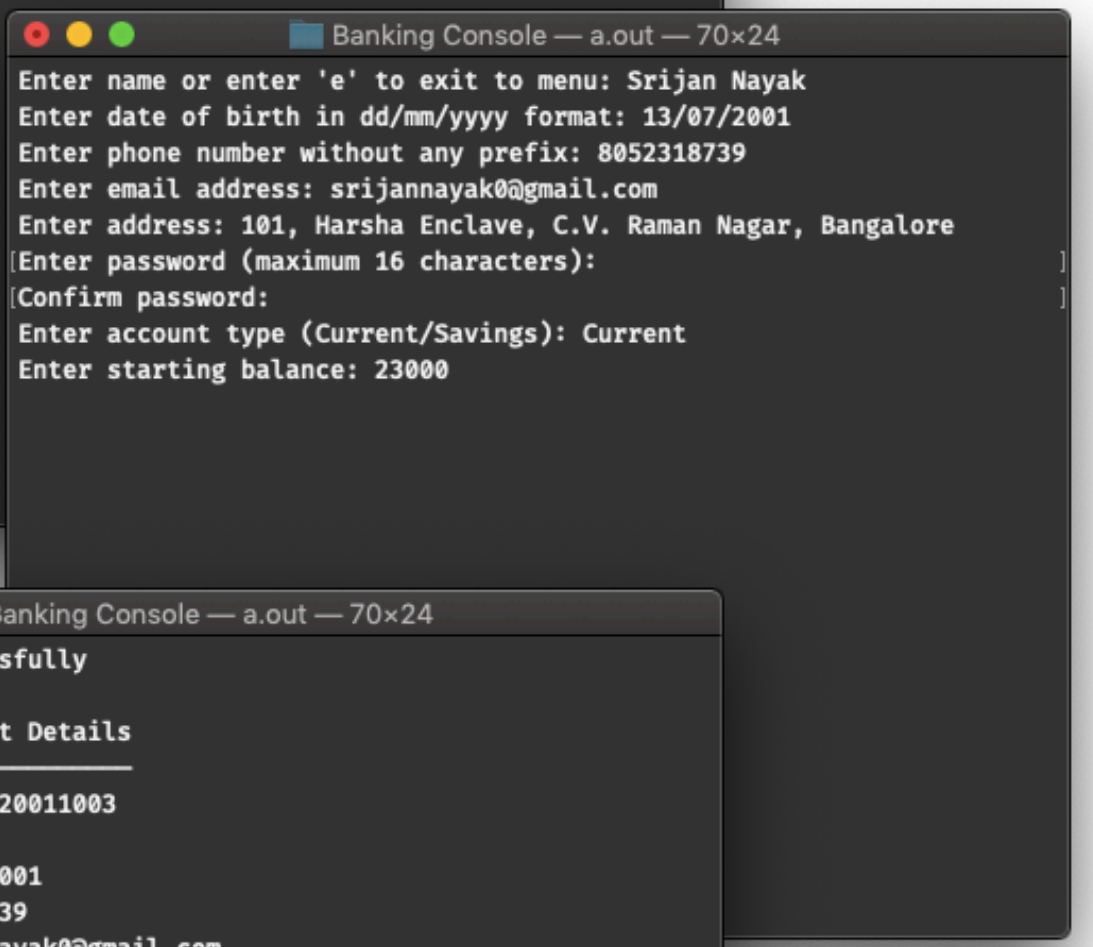
Main Menu

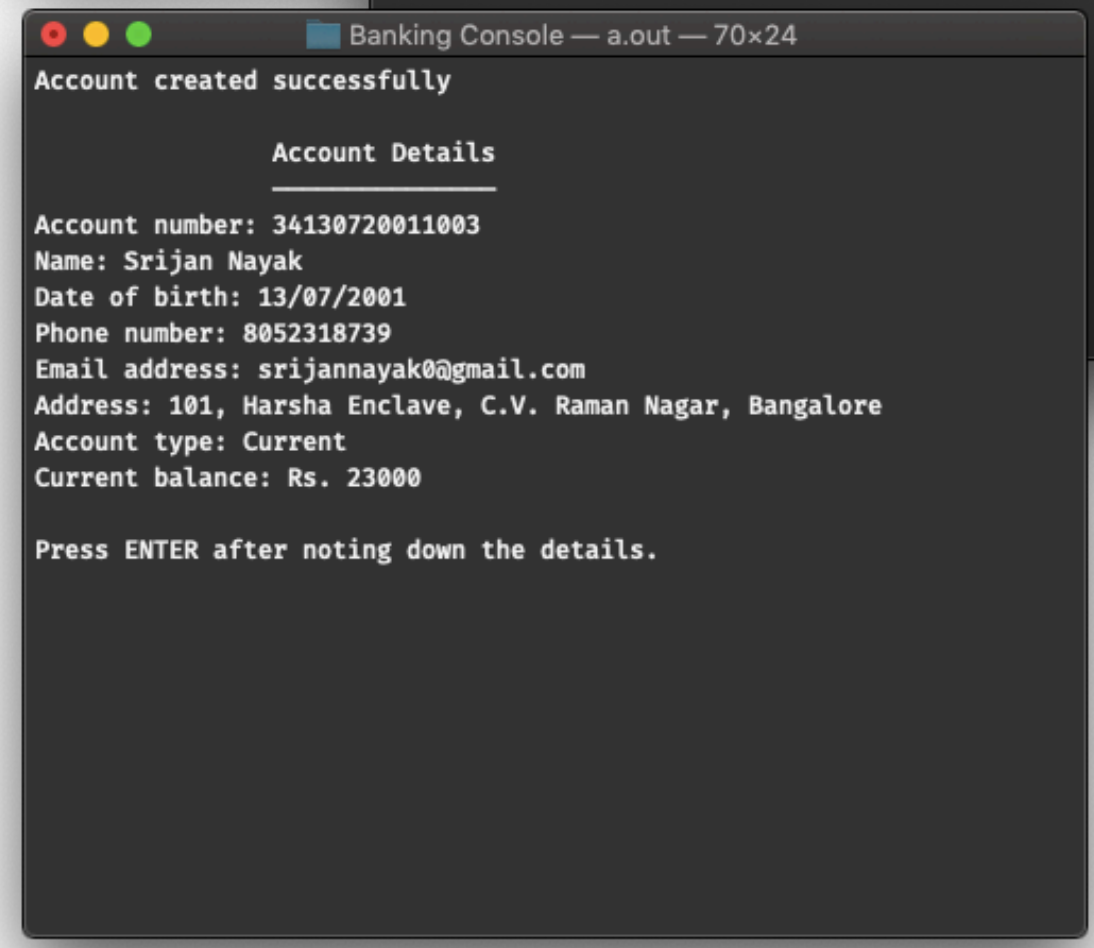


Open Account Module



Hidden password input for security

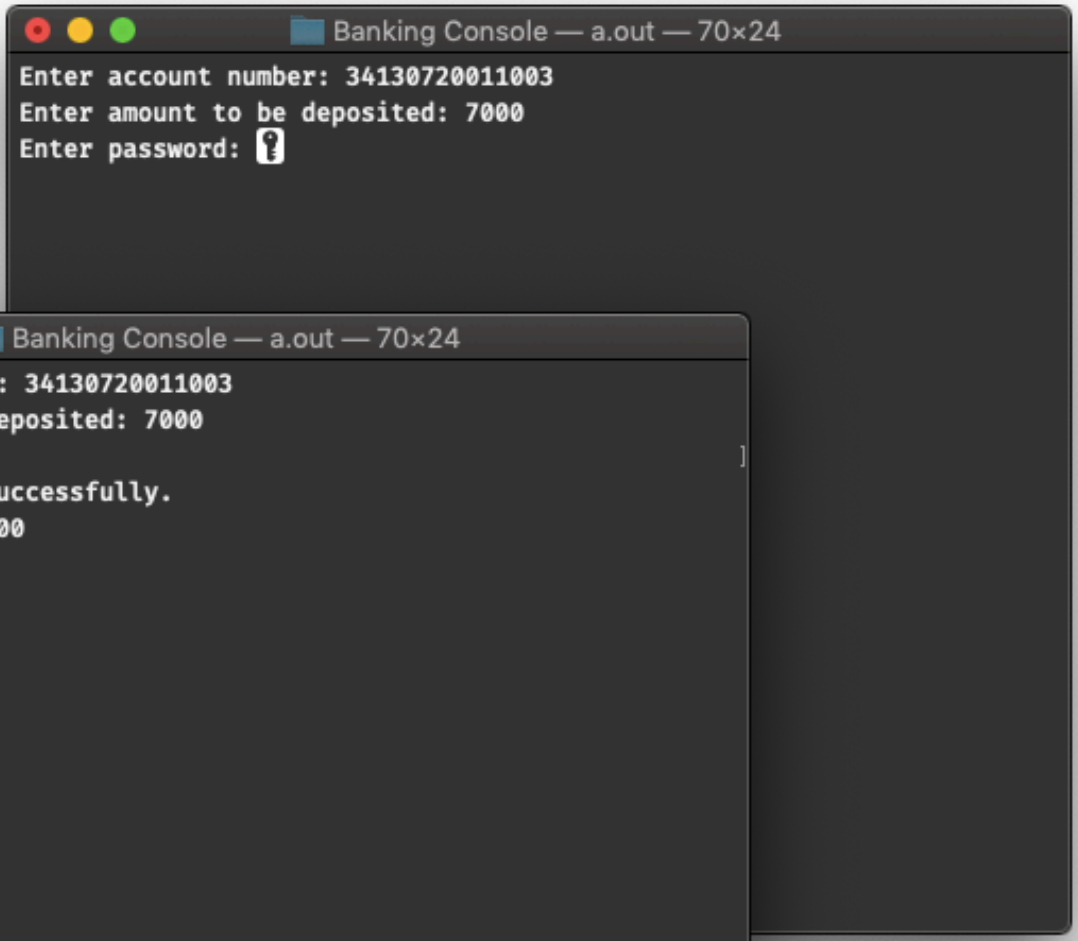


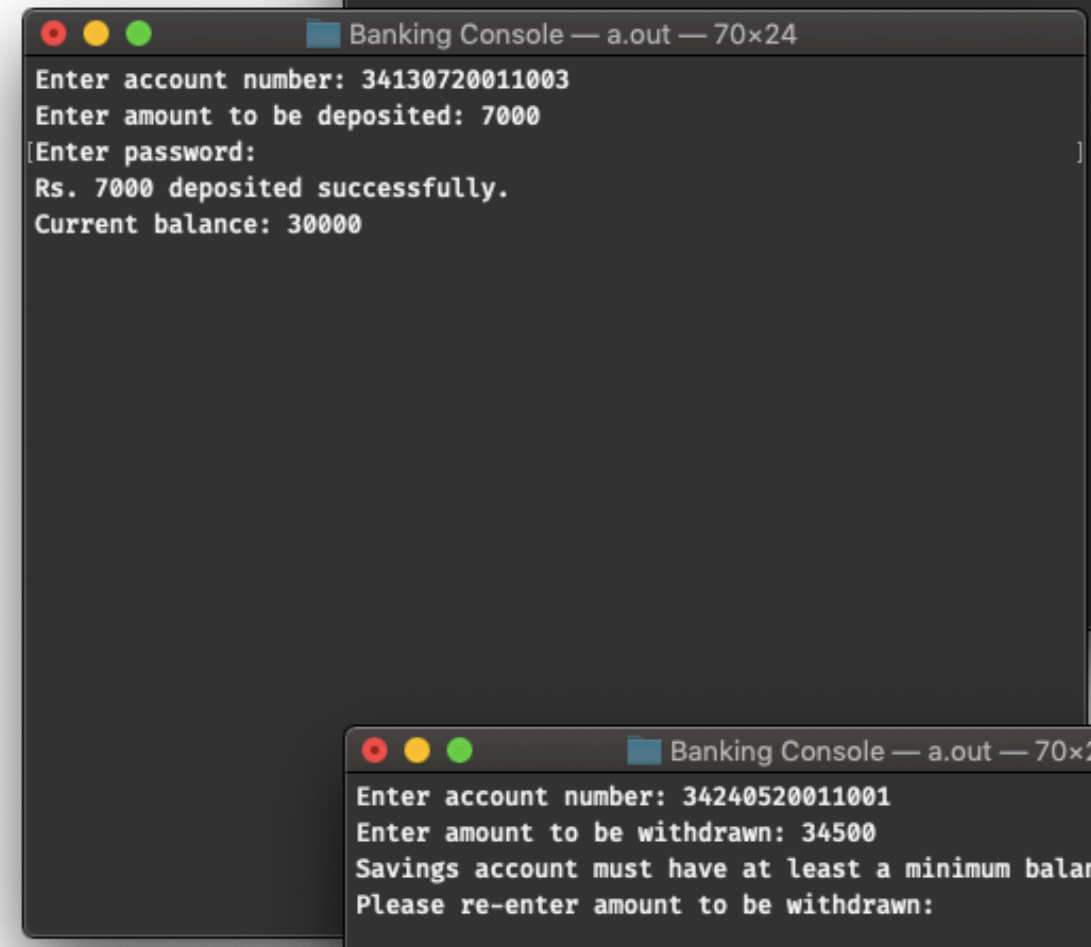


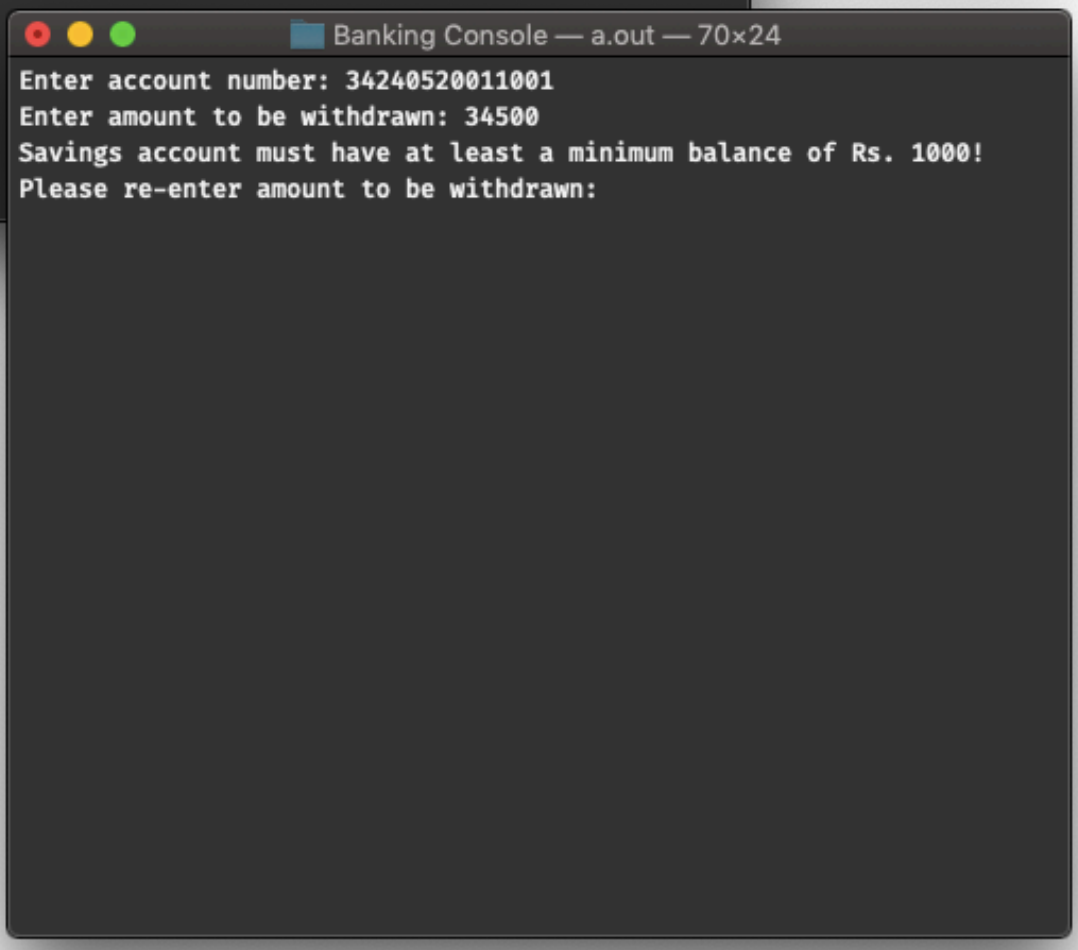
Details shown after opening of account

Deposit Amount Module

Password Authentication

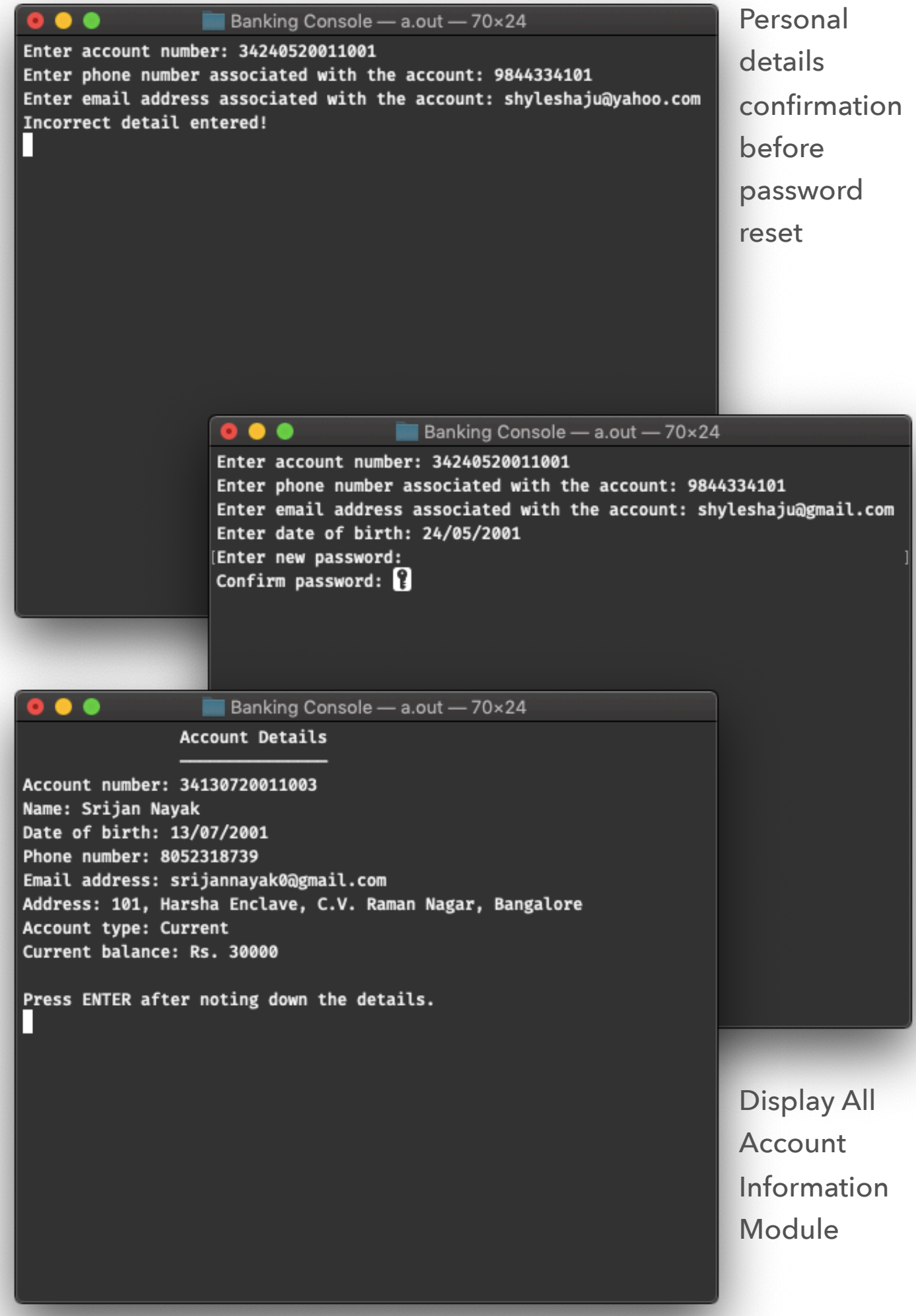






Withdrawal limit for savings accounts in Withdraw Amount Module

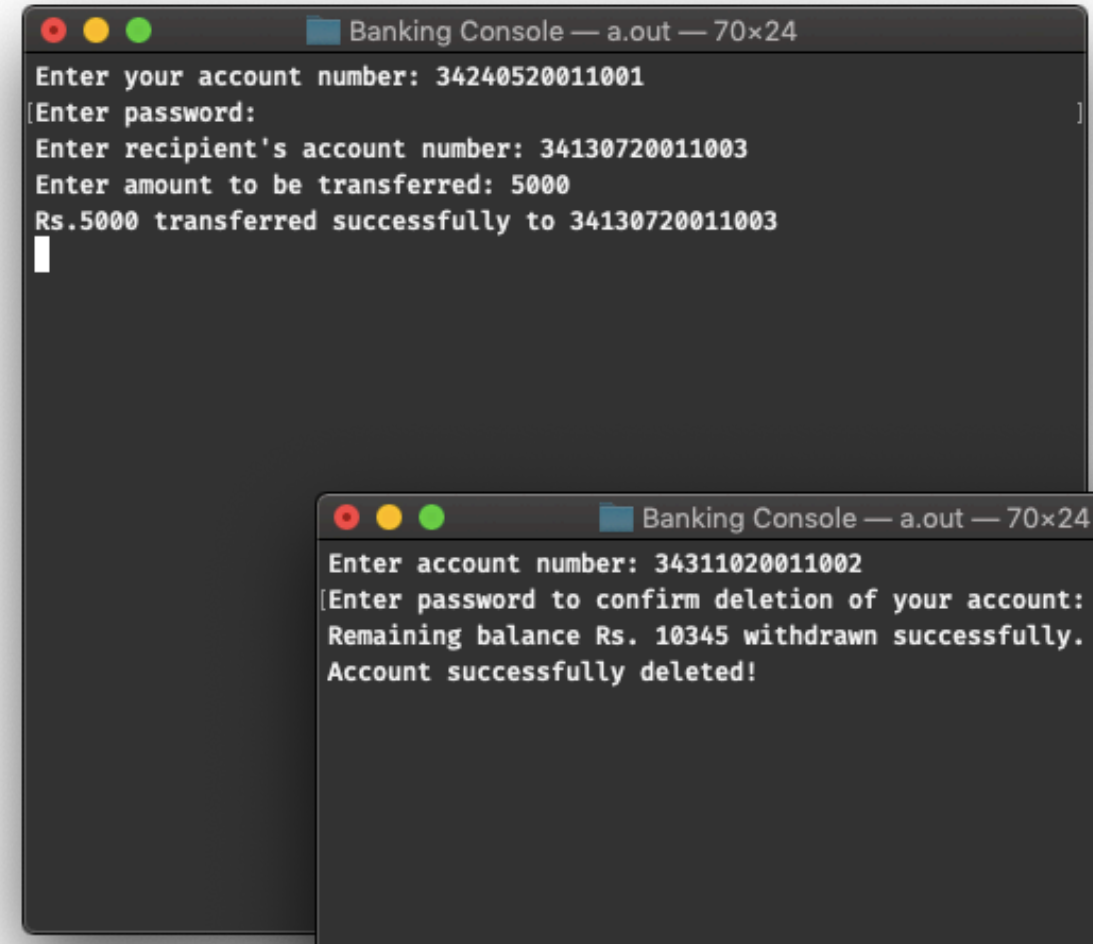
Forgot Password Module



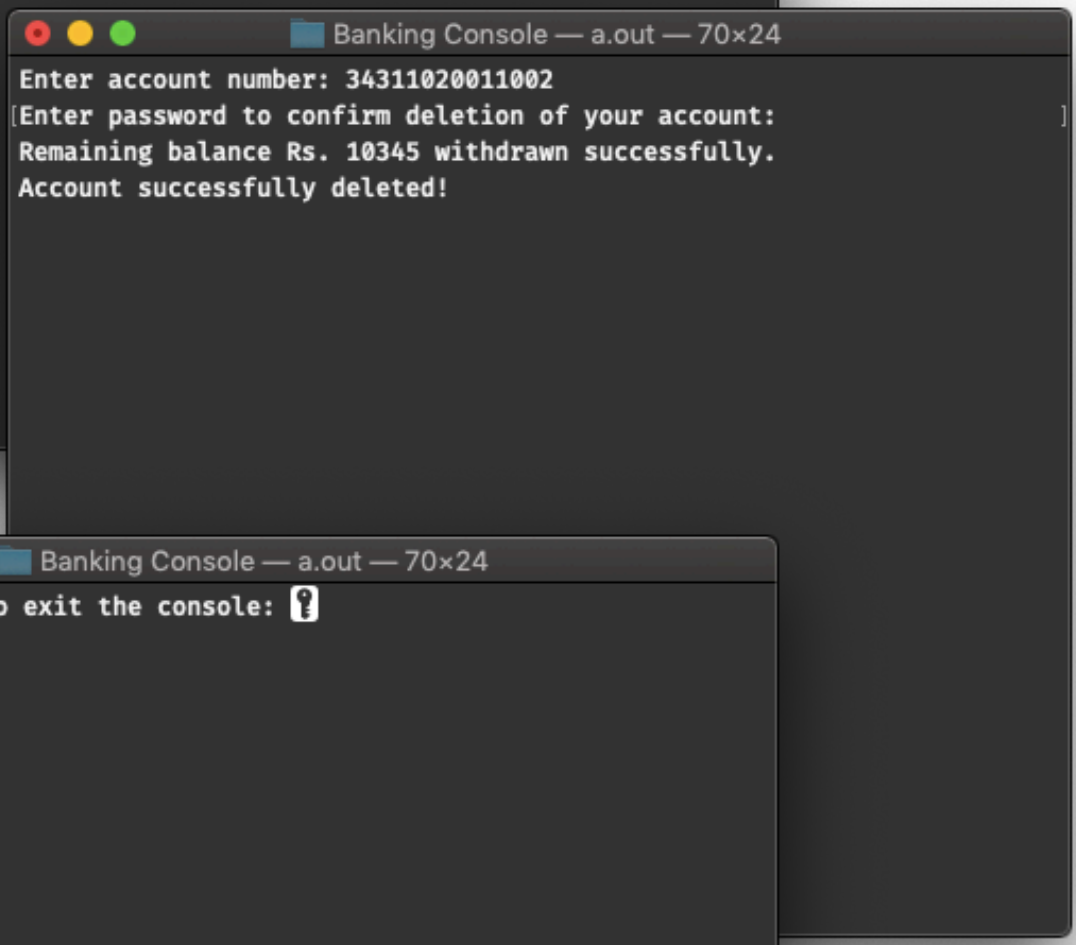
Personal details confirmation before password reset

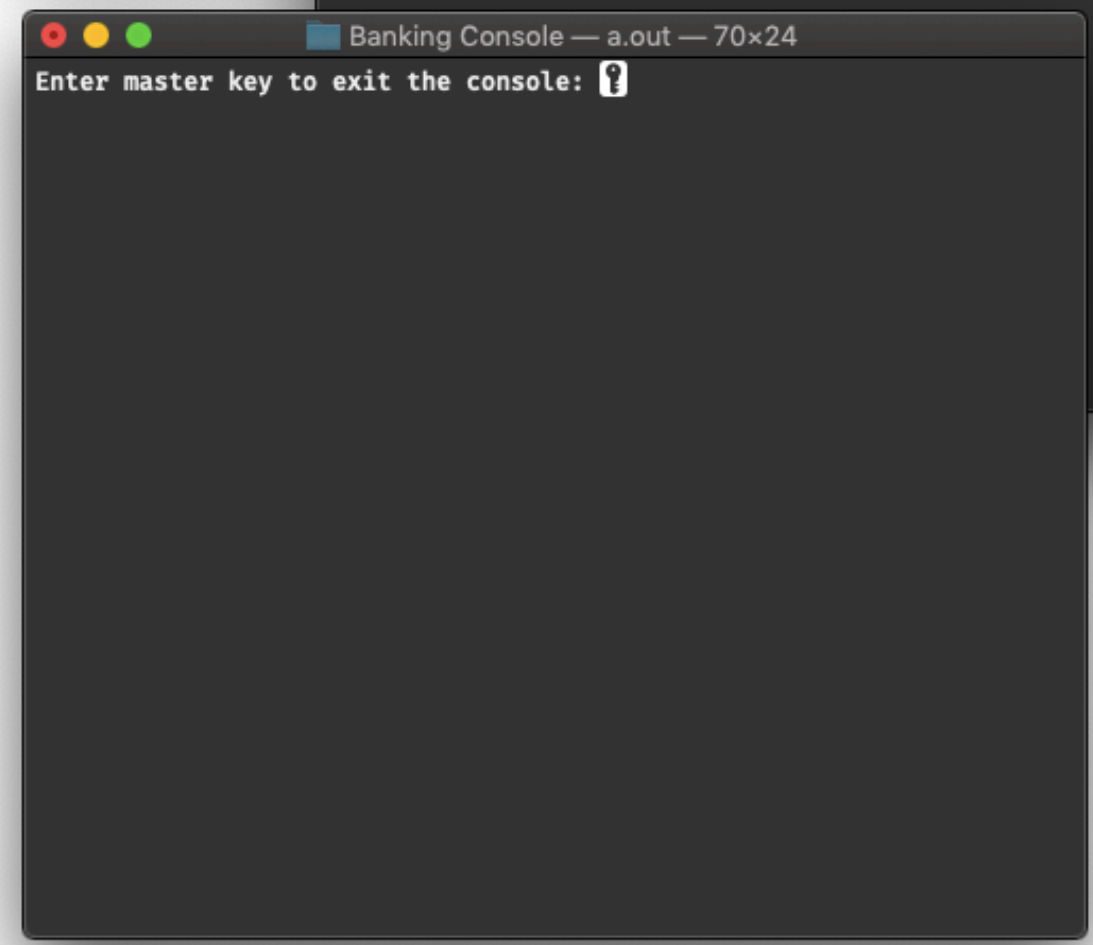
Display All Account Information Module

Transfer Amount Module



Delete Account Module





Exit Banking Console Module

Bibliography

http://www.zedwood.com/article/cpp-sha256-function

https://stackoverflow.com