Kyle Nguyen, Awet Fikadu, Brandon Flores, Miguel Logarta

Group Project: Bank for Customer

Updated: 5/12/23 at last page

Overview: This is going to be an object oriented bank program for a customer. The objects for the program are going to be the bank, the names of the users, the checking account, savings account, and credit account. The databases are going to be the names, account number, the balance of each account, user PIN, routing address, email, etc(Will see if we need to add more).

1. Stage One: Setting up file, printing menu, and setting up checking functions.
    a. Open up a file, see if we can read it, and then print it out
    b. Ask the user to input the email and password, and make sure it matches with a user file
        i. Linear Search
    c. Present the user with a menu, asking what account they want to access: Checking, Savings, and Credit, or to exit the program.
    d. In this stage, we will mainly be working on the checking account, but many things from the checking account work with the savings account.
    e. If it matches, we will present a menu to the user asking the following: Would you like to withdraw, check, deposit, transfer, send money, or disable/open a card, change your pin, go back .
        i. Withdraw: Ask how much the user wants to withdraw, and then subtract it from the user file and print out their amount after the withdrawal.
            1. If the user attempts to withdraw too much, print out an error message saying you do not have that much, and return them to the menu.
        ii. Deposit: Ask how much the user wants to deposit, and then add it to the user file and print out the amount after the deposit.
        iii. Check: Print out the amount of money in the userfile
        iv. Transfer money: Do Later
        v. Send money to another user: Do Later
        vi. Disable or open a card(?)
        vii. Change PIN(?)
        viii. Switch back to the first menu.
2. Stage Two: Savings account.
    a. For the most part, the menus for this will be the same as the checking account, but we will add one more option to see how much annual interest you have made.
        i. We will add a year

Summary of Features:

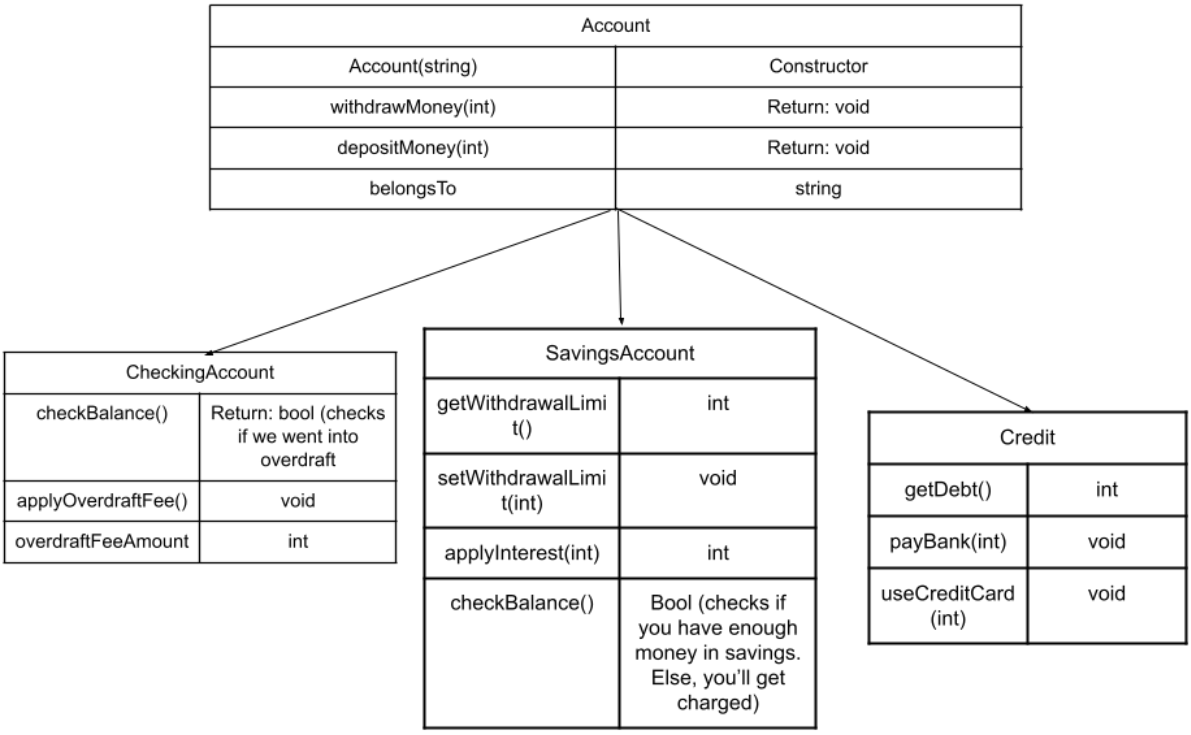
- Bank contains a database of users
- Each user contains customer information (username, email, real name, password, bank pin for ATMs)
    - We have to ask whether the user is at an ATM or the bank website (The difference matters since it'll ask you for your password or pin)
- Each user can have an unlimited amount of accounts, which are of type savings, credit, or checking.
- For each of these types of accounts, the user can withdraw, deposit, transfer money
- User can cash a check, disable or open a card, or change their pin, open / close an account

Bank Accounts:

Here are the types of accounts and their uses:

Checking	Savings	Credit
- You can withdraw often - If you go negative, you'll be charged an overdraft fee	- You have a withdrawal limit. Some banks even charge you for not having money in your savings. - You earn money from interest	- You can go into the negatives, but there's a floor (maximum amount that you can owe) - You have to pay back the bank

Inheritance Table:



Users

-

User	
username	string
accounts	vector<Account*> (This will hold a pointer to your different types of bank accounts)
email	string
password	string
pin	number
Real name	String (Your government name)

Database

- Have multiple files: Users.csv and a csv file for each bank account type

Filename	Contains
Users.csv	Username, email, password, pin number
CheckingAccounts.csv	Usernames, balances, current overdraft fees
SavingsAccounts.csv	Usernames, interests, balances
CreditAccounts.csv	Usernames, debts

- We're going to read each of these files and store each record / row into a linked list
- The Users linked list will be searched every time we go through a record in each banking account. For example, if we're scanning CheckingAccouns.csv, we find a record "Mary, 300, 0" which means that we have a checking account that belongs to user "Mary" with a balance of 300 dollars and 0 overdraft fees. We will search for "Mary" in the Users linked list, and if a match is found, give Mary a pointer to the account in the linked list.

Working Code so far:

main.cpp
Users.csv
checkingAccounts.csv
savingAccounts.csv
creditAccounts.csv

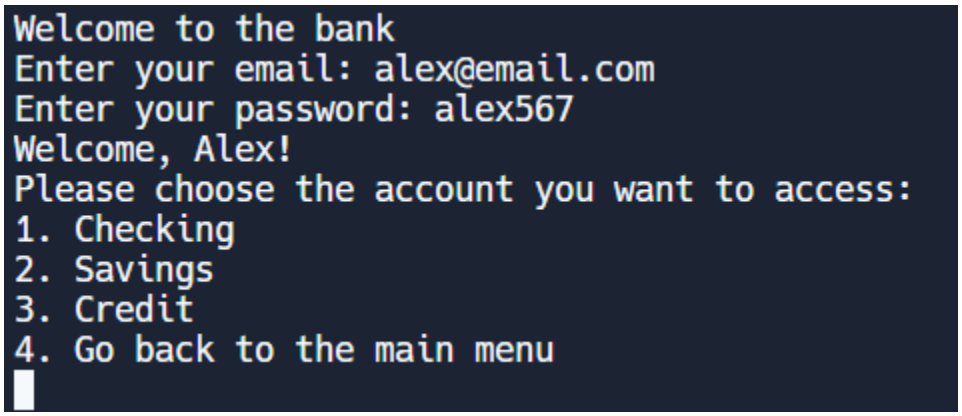
Update Halfway mark:

So far users are able to log in successfully into their accounts. Some banking accounts are accessible with some functions not yet fully implemented but the basic needs for an ordinary user using a banking machine is now being able to withdraw money, deposit and check their balances on both savings and checking accounts. In checkings, users can so far change their pin and pay their overdraft fees, while in savings users can check their savings interests from their

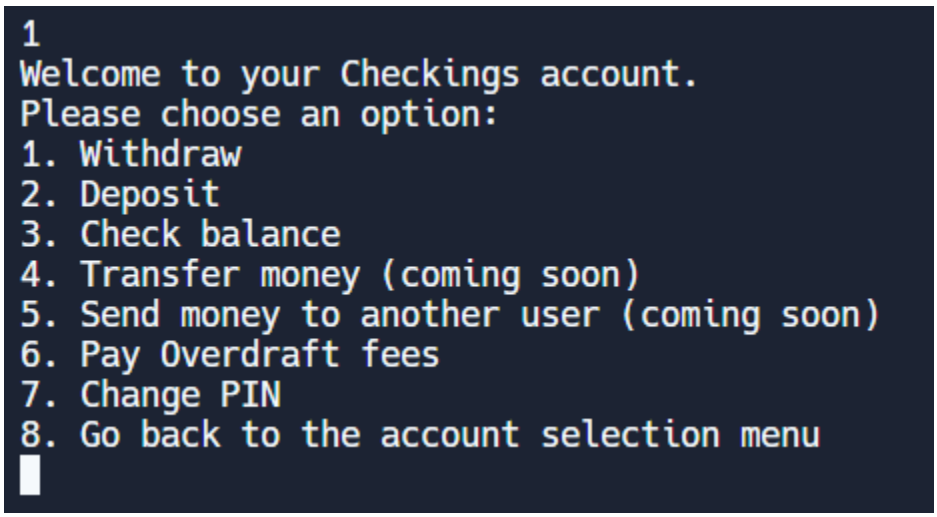
balance. We will continue to add more operational functions as we continue our work. Such as transferring money and finally adding credit card accounts.

Pictures:

User can Successfully log in



User Can choose accounts such as checkings



User Can Check Balance in checkings

```
Current balance: 2500
Please choose an option:
1. Withdraw
2. Deposit
3. Check balance
4. Transfer money (coming soon)
5. Send money to another user (coming soon)
6. Pay Overdraft fees
7. Change PIN
8. Go back to the account selection menu
```

The current balance corresponds to the same amount in the checkingsAccounts.csv file

Username:	Balance:	Overdraft Fee:
John	2500	35
Frank	465	35
Mary	10000	60
Bob	750	15
Phil	3000	45
Kate	1500	25
Meg	200	5
Chris	800	10
Mike	5000	50
Liv	100	0
Alex	2500	10
Sara	5000	50
Ryan	250	5
Emma	6000	75
David	2000	20
Lucas	300	5
Lily	800	10
Sam	1000	10
Mia	400	5

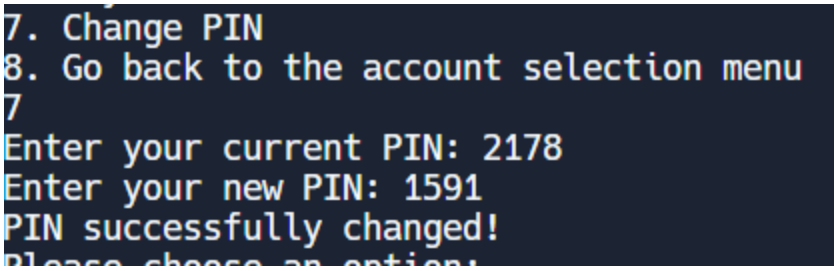
User can Deposit and Withdraw balance from that amount

```
Enter the amount to withdraw: 500
Withdrawal successful!
New balance: 2000
Please choose an option:
1. Withdraw
2. Deposit
3. Check balance
4. Transfer money (coming soon)
5. Send money to another user (coming soon)
6. Pay Overdraft fees
7. Change PIN
8. Go back to the account selection menu
2
Enter the amount to deposit: 43
Deposit successful!
New balance: 2043
Please choose an option:
1. Withdraw
2. Deposit
3. Check balance
4. Transfer money (coming soon)
5. Send money to another user (coming soon)
6. Pay Overdraft fees
7. Change PIN
8. Go back to the account selection menu
```

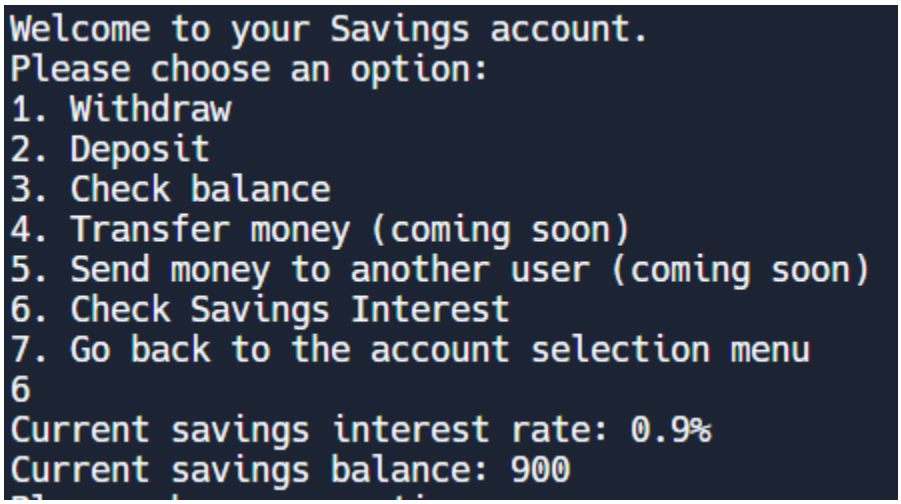
User can pay overdraft fees from their balance

```
6. Pay Overdraft fees
7. Change PIN
8. Go back to the account selection menu
6
Current overdraft fees: 10
Enter the amount you want to pay: 3
Deposit successful!
New balance: 2040
Payment successful!
New overdraft fees: 7
```

User can change pin from users.csv database



In savings users can check their interest rate as well



When anything in the database files changes everything will also be updated to new balances and pins that were adjusted during the execution for the next run. Files are also updated.

Code as Text:

```
#include <fstream>
#include <iostream>
#include <sstream>
#include <vector>

using namespace std;

struct User {
  string username;
  string email;
  string password;
  int pin;
```

```
  string realName;
};

bool authenticateUser(const string &email, const string &password, User &user) {
  ifstream file("users.csv");
  string line;

  // Read the file line by line
  while (getline(file, line)) {
    stringstream ss(line);
    vector<string> tokens;

    // Split each line into tokens using a comma separator
    while (ss.good()) {
      string token;
      getline(ss, token, ',');
      tokens.push_back(token);
    }

    // Compare the email and password entered by the user with the email and
    // password of each line in the file
    if (tokens[1] == email && tokens[2] == password) {
      user.username = tokens[0];
      user.email = tokens[1];
      user.password = tokens[2];
      user.pin = stoi(tokens[3]);
      return true;
    }
  }

  // If no match is found, return false
  return false;
}

// functions
void withdraw(double amount, User& currentUser) {
    string line;
    double currentBalance = 0.0;
    string currentUserUsername = currentUser.username;

    ifstream infile("checkingAccounts.csv");
    vector<string> updatedLines;
    while (getline(infile, line)) {
        // Split each line into tokens using a comma separator
        stringstream ss(line);
        vector<string> tokens;
        while (ss.good()) {
            string token;
            getline(ss, token, ',');
            tokens.push_back(token);
        }

        // If the line corresponds to the current user's account, update the balance
        if (tokens[0] == currentUserUsername) {
            currentBalance = stod(tokens[1]) - amount;
            tokens[1] = to_string(currentBalance);  // Update the second field of the CSV line
        }

        // Add the current line (possibly updated) to the lines to output
        updatedLines.push_back(tokens[0] + "," + tokens[1] + "," + tokens[2]);
    }

    // Save the updated lines to the file
    infile.close();
    ofstream outfile("checkingAccounts.csv");
    for (const string& line : updatedLines) {
```

```
        outfile << line << endl;
    }

    // Print the new balance
    cout << "Withdrawal successful!" << endl;
    cout << "New balance: " << currentBalance << endl;
}

void savingsWithdraw(double amount, User& currentUser) {
    string line;
    double currentBalance = 0.0;
    string currentUserUsername = currentUser.username;

    ifstream infile("savingsAccounts.csv");
    vector<string> updatedLines;
    while (getline(infile, line)) {
        // Split each line into tokens using a comma separator
        stringstream ss(line);
        vector<string> tokens;
        while (ss.good()) {
            string token;
            getline(ss, token, ',');
            tokens.push_back(token);
        }

        // If the line corresponds to the current user's account, update the balance
        if (tokens[0] == currentUserUsername) {
            currentBalance = stod(tokens[2]) - amount;
            tokens[2] = to_string(currentBalance);  // Update the third field of the CSV line
        }

        // Add the current line (possibly updated) to the lines to output
        updatedLines.push_back(tokens[0] + "," + tokens[1] + "," + tokens[2]);
    }

    // Save the updated lines to the file
    infile.close();
    ofstream outfile("savingsAccounts.csv");
    for (const string& line : updatedLines) {
        outfile << line << endl;
    }

    // Print the new balance
    cout << "Withdrawal successful!" << endl;
    cout << "New balance: " << currentBalance << endl;
}


void deposit(double amount, User &currentUser) {
  ifstream file("checkingAccounts.csv");
  string line;
  double currentBalance = 0.0;

  // Skip the header line
  if (file.good()) {
    getline(file, line);
  }

  // Open the output file and write the updated balances
  ofstream outfile("tmp.csv");   // Use a temporary file to avoid data loss
  outfile << "Username:,Balance:,Overdraft Fee:" << endl;  // Write the header line to the new file

  while (getline(file, line)) {
    stringstream ss(line);
    vector<string> tokens;
    string token;
```

```
    // Split each line into tokens using a comma separator
    while (getline(ss, token, ',')) {
      // Remove leading and trailing whitespace from the token
      size_t start = token.find_first_not_of(' ');
      size_t end = token.find_last_not_of(' ');
      tokens.push_back(token.substr(start, end - start + 1));
    }

    if (tokens[0] == currentUser.username) {
      currentBalance = stod(tokens[1]) + amount;
      outfile << tokens[0] << ", " << currentBalance << ", " << tokens[2] << endl;
    } else {
      outfile << line << endl;
    }
  }

  file.close();
  outfile.close();

  // Replace the original file with the temporary file
  remove("checkingAccounts.csv");
  rename("tmp.csv", "checkingAccounts.csv");

  cout << "Deposit successful!" << endl;
  cout << "New balance: " << currentBalance << endl;
}

void savingsDeposit(double amount, User &currentUser) {
    ifstream file("savingsAccounts.csv");
    string line;
    double currentBalance = 0.0;
    string currentUserUsername = currentUser.username;

    // Skip the header line
    if (file.good()) {
        getline(file, line);
    }

    // Open the output file and write the updated balances
    ofstream outfile("tmp.csv");  // Use a temporary file to avoid data loss
    outfile << "Username:,Savings Interest Rate:,Balance:" << endl;  // Write the header line to the
new file

    while (getline(file, line)) {
        stringstream ss(line);
        vector<string> tokens;
        string token;

        // Split each line into tokens using a comma separator
        while (getline(ss, token, ',')) {
            // Remove leading and trailing whitespace from the token
            size_t start = token.find_first_not_of(' ');
            size_t end = token.find_last_not_of(' ');
            tokens.push_back(token.substr(start, end - start + 1));
        }

        if (tokens[0] == currentUserUsername) {
            currentBalance = stod(tokens[2]) + amount;
            outfile << tokens[0] << ", " << tokens[1] << ", " << currentBalance << endl;
        } else {
            outfile << line << endl;
        }
    }

    file.close();
```

```
    outfile.close();

    // Replace the original file with the temporary file
    remove("savingsAccounts.csv");
    rename("tmp.csv", "savingsAccounts.csv");

    cout << "Deposit successful!" << endl;
    cout << "New balance: " << currentBalance << endl;
}

// case 3 function
void checkBalance(const User &currentUser) {
  ifstream usersFile("users.csv");
  string usersLine;
  string email = currentUser.email;

  // Find the user's username in users.csv based on their email
  while (getline(usersFile, usersLine)) {
    stringstream ss(usersLine);
    vector<string> usersTokens;
    string token;

    while (getline(ss, token, ',')) {
      usersTokens.push_back(token);
    }

    if (usersTokens[1] == email) {
      string currentUserUsername = usersTokens[0];

      // Find the user's current balance in checkingAccounts.csv based on their
      // username
      ifstream checkingAccountsFile("checkingAccounts.csv");
      string checkingAccountsLine;
      double currentBalance = 0.0;

      while (getline(checkingAccountsFile, checkingAccountsLine)) {
        stringstream ss(checkingAccountsLine);
        vector<string> checkingAccountsTokens;
        string token2;

        while (getline(ss, token2, ',')) {
          checkingAccountsTokens.push_back(token2);
        }

        if (checkingAccountsTokens[0] == currentUserUsername) {
          currentBalance = stod(checkingAccountsTokens[1]);
          break;
        }
      }

      checkingAccountsFile.close();

      // Print the user's current balance
      cout << "Current balance: " << currentBalance << endl;

      // Exit the function
      return;
    }
  }

  // If the user wasn't found in users.csv, print an error message
  cout << "Error: user not found" << endl;
}

void checkSavingsBalance(const User &currentUser) {
  ifstream usersFile("users.csv");
```

```
  string usersLine;
  string email = currentUser.email;

  // Find the user's username in users.csv based on their email
  while (getline(usersFile, usersLine)) {
    stringstream ss(usersLine);
    vector<string> usersTokens;
    string token;

    while (getline(ss, token, ',')) {
      usersTokens.push_back(token);
    }

    if (usersTokens[1] == email) {
      string currentUserUsername = usersTokens[0];

      // Find the user's current balance in savingsAccounts.csv based on their
      // username
      ifstream savingsAccountsFile("savingsAccounts.csv");
      string savingsAccountsLine;
      double currentBalance = 0.0;

      while (getline(savingsAccountsFile, savingsAccountsLine)) {
        stringstream ss(savingsAccountsLine);
        vector<string> savingsAccountsTokens;
        string token2;

        while (getline(ss, token2, ',')) {
          savingsAccountsTokens.push_back(token2);
        }

        if (savingsAccountsTokens[0] == currentUserUsername) {
          currentBalance = stod(savingsAccountsTokens[2]);
          break;
        }
      }

      savingsAccountsFile.close();

      // Print the user's current balance
      cout << "Current balance: " << currentBalance << endl;

      // Exit the function
      return;
    }
  }
  // If the user wasn't found in users.csv, print an error message
  cout << "Error: user not found" << endl;
}

void changePin(User &currentUser) {
  cout << "Enter your current PIN: ";
  int currentPin;
  cin >> currentPin;

  if (currentPin != currentUser.pin) {
    cout << "Incorrect PIN." << endl;
    return;
  }

  cout << "Enter your new PIN: ";
  int newPin;
  cin >> newPin;

  ifstream file("users.csv");
```

```
  // Use a temporary file to avoid data loss
  ofstream outfile("tmp.csv");

  string line;

  // Skip the first line
  getline(file, line);
  outfile << line << endl;

  while (getline(file, line)) {
    stringstream ss(line);
    vector<string> tokens;

    while (ss.good()) {
      string token;
      getline(ss, token, ',');
      tokens.push_back(token);
    }

    if (tokens[1] == currentUser.email && tokens[2] == currentUser.password) {
      currentUser.pin = newPin;
      outfile << tokens[0] << "," << tokens[1] << "," << tokens[2] << "," << currentUser.pin << endl;
    } else {
      outfile << line << endl;
    }
  }

  file.close();
  outfile.close();

  // Replace the original file with the temporary file
  remove("users.csv");
  rename("tmp.csv", "users.csv");

  cout << "PIN successfully changed!" << endl;
}

void payOverdraftFees(User &currentUser) {
  ifstream file("checkingAccounts.csv");
  string line;
  double currentOverdraft = 0.0;

  // Skip the header line
  if (file.good()) {
    getline(file, line);
  }

  // Find the current overdraft fees for the user
  while (getline(file, line)) {
    stringstream ss(line);
    vector<string> tokens;
    string token;

    // Split each line into tokens using a comma separator
    while (getline(ss, token, ',')) {
      // Remove leading and trailing whitespace from the token
      size_t start = token.find_first_not_of(' ');
      size_t end = token.find_last_not_of(' ');
      tokens.push_back(token.substr(start, end - start + 1));
    }

    if (tokens[0] == currentUser.username) {
      currentOverdraft = stod(tokens[2]);
      break;
    }
  }
```

```
  file.close();

  cout << "Current overdraft fees: " << currentOverdraft << endl;

  cout << "Enter the amount you want to pay: ";
  double payment;
  cin >> payment;

  if (payment > currentOverdraft) {
    cout << "You don't have that much in overdraft fees." << endl;
    return;
  }

  // Subtract the payment from the user's balance and overdraft fees
  deposit(-payment, currentUser);
  currentOverdraft -= payment;

  // Update the user's overdraft fees in the file
  ifstream infile("checkingAccounts.csv");
  vector<string> updatedLines;
  while (getline(infile, line)) {
    stringstream ss(line);
    vector<string> tokens;

    while (ss.good()) {
      string token;
      getline(ss, token, ',');
      tokens.push_back(token);
    }

    if (tokens[0] == currentUser.username) {
      tokens[2] = to_string(currentOverdraft);
    }

    updatedLines.push_back(tokens[0] + ", " + tokens[1] + ", " + tokens[2]);
  }

  infile.close();
  ofstream outfile("checkingAccounts.csv");
  for (const string& line : updatedLines) {
    outfile << line << endl;
  }

  cout << "Payment successful!" << endl;
  cout << "New overdraft fees: " << currentOverdraft << endl;
}

void checkSavingsInterest(const User &currentUser) {
  ifstream usersFile("users.csv");
  string usersLine;
  string email = currentUser.email;

  // Find the user's username in users.csv based on their email
  while (getline(usersFile, usersLine)) {
    stringstream ss(usersLine);
    vector<string> usersTokens;
    string token;

    while (getline(ss, token, ',')) {
      usersTokens.push_back(token);
    }

    if (usersTokens[1] == email) {
      string currentUserUsername = usersTokens[0];
```

```
      // Find the user's current balance and interest rate in savingsAccounts.csv based on their
      // username
      ifstream savingsAccountsFile("savingsAccounts.csv");
      string savingsAccountsLine;
      double currentBalance = 0.0;
      double interestRate = 0.0;

      while (getline(savingsAccountsFile, savingsAccountsLine)) {
        stringstream ss(savingsAccountsLine);
        vector<string> savingsAccountsTokens;
        string token2;

        while (getline(ss, token2, ',')) {
          savingsAccountsTokens.push_back(token2);
        }

        if (savingsAccountsTokens[0] == currentUserUsername) {
          currentBalance = stod(savingsAccountsTokens[2]);
          interestRate = stod(savingsAccountsTokens[1].substr(0, savingsAccountsTokens[1].size() - 1));
// Extract the interest rate value and convert it to a double
          break;
        }
      }

      savingsAccountsFile.close();

      // Print the user's interest rate and current balance
      cout << "Current savings interest rate: " << interestRate << "%" << endl;
      cout << "Current savings balance: " << currentBalance << endl;

      // Exit the function
      return;
    }
  }

  // If the user wasn't found in users.csv, print an error message
  cout << "Error: user not found" << endl;
}


int main() {
  User currentUser;
  cout << "Welcome to the bank" << endl;
  cout << "Enter your email: ";
  string email;
  cin >> email;

  cout << "Enter your password: ";
  string password;
  cin >> password;

  if (authenticateUser(email, password, currentUser)) {
    cout << "Welcome, " << currentUser.username << "!" << endl;

    while (true) {
      int accountChoice;

      cout << "Please choose the account you want to access:" << endl;
      cout << "1. Checking" << endl;
      cout << "2. Savings" << endl;
      cout << "3. Credit" << endl;
      cout << "4. Go back to the main menu" << endl;
      cin >> accountChoice;

    if (accountChoice == 2) { // User selected the savings account
    cout << "Welcome to your Savings account. " << endl;
```

```
    while(true) {
        int operationChoice;
        double amount;
        cout << "Please choose an option:" << endl;
        cout << "1. Withdraw" << endl;
        cout << "2. Deposit" << endl;
        cout << "3. Check balance" << endl;
        cout << "4. Transfer money (coming soon)" << endl;
        cout << "5. Send money to another user (coming soon)" << endl;
        cout << "6. Check Savings Interest " << endl;
        cout << "7. Go back to the account selection menu" << endl;
        cin >> operationChoice;

        switch (operationChoice) {
            case 1:
                cout << "Enter the amount to withdraw: ";
                cin >> amount;
                savingsWithdraw(amount, currentUser);
                break;
            case 2:
                cout << "Enter the amount to deposit: ";
                cin >> amount;
                savingsDeposit(amount, currentUser);
                break;
            case 3:
                checkSavingsBalance(currentUser);
                break;
            case 4:
                cout << "Transfer money feature coming soon." << endl;
                break;
            case 5:
                cout << "Send money to another user feature coming soon." << endl;
                break;
            case 6:
              checkSavingsInterest(currentUser);
                break;
            case 7:
                // Exit the nested loop and return to the account selection menu
                break;
            default:
                cout << "Invalid option. Please try again." << endl;
        }
        if (operationChoice == 7) {
            break;
        }
    }
}


      if (accountChoice == 3) { // User selected the savings account
    cout << "Welcome to your Credit account. " << endl;
        while(true) {
        int operationChoice;
        double amount;
        cout << "Please choose an option:" << endl;
        cout << "1. Withdraw" << endl;
        cout << "2. Deposit" << endl;
        cout << "3. Check balance" << endl;
        cout << "4. Transfer money (coming soon)" << endl;
        cout << "5. Send money to another user (coming soon)" << endl;
        cout << "6. Pay Credit Balances " << endl;
        cout << "7. Go back to the account selection menu" << endl;
        cin >> operationChoice;

        switch (operationChoice) {
            case 1:
```

```
            cout << "Enter the amount to withdraw: ";
            cin >> amount;
            break;
        case 2:
            cout << "Enter the amount to deposit: ";
            cin >> amount;
            break;
        case 3:
            break;
        case 4:
            cout << "Transfer money feature coming soon." << endl;
            break;
        case 5:
            cout << "Send money to another user feature coming soon." << endl;
            break;
        case 6:
            break;
        case 7:
            // Exit the nested loop and return to the account selection menu
            break;
        default:
            cout << "Invalid option. Please try again." << endl;
    }
    if (operationChoice == 7) {
        break;
    }
    }
  }


if (accountChoice == 1) { // User selected the savings account
 cout << "Welcome to your Checkings account. " << endl;
  while (true) {
       int operationChoice;
       double amount;

       cout << "Please choose an option:" << endl;
       cout << "1. Withdraw" << endl;
       cout << "2. Deposit" << endl;
       cout << "3. Check balance" << endl;
       cout << "4. Transfer money (coming soon)" << endl;
       cout << "5. Send money to another user (coming soon)" << endl;
       cout << "6. Pay Overdraft fees" << endl;
       cout << "7. Change PIN " << endl;
       cout << "8. Go back to the account selection menu" << endl;
       cin >> operationChoice;

       switch (operationChoice) {
       case 1:
         cout << "Enter the amount to withdraw: ";
         cin >> amount;
         withdraw(amount, currentUser);
         break;

         // Call the appropriate function to withdraw and print the remaining
         // balance Make sure to handle errors if the user tries to withdraw
         // too much
       case 2:
         cout << "Enter the amount to deposit: ";
         cin >> amount;
         deposit(amount, currentUser);
         // Call the appropriate function to deposit and print the new
         // balance
         break;
       case 3:
         checkBalance(currentUser);
```

```
            break;

            // Call the appropriate function to check the balance and print it
            break;
          case 4:
            cout << "Transfer money feature coming soon." << endl;
            break;
          case 5:
            cout << "Send money to another user feature coming soon." << endl;
            break;
          case 6:
            payOverdraftFees(currentUser);
            break;
          case 7:
            changePin(currentUser);
            break;
          case 8:
            break; // Go back to the account selection menu
          default:
            cout << "Invalid option. Please try again." << endl;
          }

          if (operationChoice == 8) {
            break;
          }
        }
      } else if (accountChoice == 4) {
        // Go back to the main menu or exit the program
        break;
      } else {
        cout << "Invalid option. Please try again." << endl;
      }
    }

  } else {
    cout << "User not found. Please try again." << endl;
  }
  return 0;
}
```