Contents

[WARNING 2](#_Toc201882376)

[Project Overview 2](#_Toc201882377)

[Objectives 2](#_Toc201882378)

[Features 2](#_Toc201882379)

[Concepts Used 2](#_Toc201882380)

[Data Structure 3](#_Toc201882381)

[Code Breakdown 3](#_Toc201882382)

[Global variables 3](#_Toc201882383)

[LoggedUser Variable 3](#_Toc201882384)

[Filename Variable 3](#_Toc201882385)

[UserType 3](#_Toc201882386)

[Functions 4](#_Toc201882387)

[Withdraw Function 4](#_Toc201882388)

[Deposit() Function 4](#_Toc201882389)

[Deposit(vector<User>& users) Function 4](#_Toc201882390)

[Check Balance Function 4](#_Toc201882391)

[Change password Function 4](#_Toc201882392)

[Register User Function 4](#_Toc201882393)

[Login User Function 4](#_Toc201882394)

[Logout(const string & filename, vector<User>& users) Function 4](#_Toc201882395)

[Lowercase Text Function 5](#_Toc201882396)

[Getter Functions 5](#_Toc201882397)

[SaveData(vector<User> users) Function 5](#_Toc201882398)

[Load Data Function 5](#_Toc201882399)

[showMenu(vector<User>& users) Function 5](#_Toc201882400)

[Program Flow 5](#_Toc201882401)

# WARNING

This documentation has been made to cater for one who has basic knowledge in OOP i.e has knowledge on class objects, object constructors, access specifiers, member functions and enums. If you lack understanding in these kindly do your research before going through this document.

Try as best as you can to go through this and ask questions where needed. We will meet twice or thrice to go through it together and share knowledge.

# Project Overview

This project is a simple banking system made in Object Oriented Programming (OOP) style. It allows multiple users, admins and clients, to interact with the system through a console based interface.

The main operations include:

* Registering accounts (Admin function only)
* Depositing and withdrawing money
* Checking balances
* Changing account passwords
* Admins specific features such as depositing money for clients

All the program’s data is saved in **usersDB.txt.**

# Objectives

1. Implement a fully working console banking system using **OOP**.
2. Practice using **file I/O**, **string processing**, and **control structures** in C++.
3. Maintain **separation between admin and client functionality**.
4. Ensure **data persistence** between sessions via file saving/loading.
5. Provide a clean, understandable menu-driven interface.

# Features

|  |  |  |
| --- | --- | --- |
| **Feature** | **Admin** | **Client** |
| Register new account | Yes | No |
| Login | Yes | Yes |
| Deposit | Yes (for client) | Yes |
| Withdraw | No | Yes |
| Check balance | No | Yes |
| Change password | Yes | Yes |
| Logout | Yes | Yes |

# Concepts Used

* Classes
* Enums
* For loop
* Do while loop
* Fstream (File Stream, for manipulating files)
* Vector
* While loop

# Data Structure

All of the programs data is stored in a text file called usersDB in csv format i.e **item1,item2,item3.**

**A screenshot of a computer screen

Description automatically generated**

The data model is based on an object class called User that stores these properties:

* **ID**
* **Username**
* **Password**
* **Balance**
* **UserType** (admin/client).

The User class has the following methods/functions: withdraw, deposit, changePassword, showMenu, checkBalance and 5 getter functions.

When the program runs, this data is read from the text file and is formatted into an vector that stores the users inform of the User class.

**Note:** A vector is just a fancy array with better capabilities. They’re better than normal arrays. Itself is an object.

# Code Breakdown

## Global variables

### LoggedUser Variable

Int loggedUser = -1; - Tracks the currently logged-in user. -1 means no user is logged in. While any other value greater than -1 is the index of the user logged in the array of users.

### Filename Variable

String filename = “usersDB.txt”; - Name of the file where user data is saved.

### UserType

This is an enum with two states ADMIN and NORMAL. Think of an enum as a custom data type. We created a data type called UserType that assigns two states (admin and normal) as integer constants, making the code easier to read. In this case, ADMIN’s integer value is 0 and NORMAL’s value is 1. The values are automatically set by the compiler, so we don’t have to set them ourselves.

#### How is this useful?

You might have noticed that in usersDB.txt, 5 things are stored for a user: their ID, username, password, balance and a mysterious integer that’s either 0 or 1.

**A screenshot of a computer screen

Description automatically generated**



That fifth value is our enum UserType. When the code is compiled, ADMIN is turned to 0 and NORMAL to 1 for processing but for us it remains ADMIN and NORMAL which makes it easier for us.

We use this enum to distinguish between accounts. If an account belongs to a client, then that trailing integer will be 1 and 0 for an administrator account.

When reading this value from the file, we convert it from its integer form into its UserType enum equivalent so we can use it where fit.

## Functions

### Withdraw Function

The withdraw function lets the user withdraw cash from their account and updates their balance after.

The user will not be able to withdraw an amount higher than the available balance.

### Deposit() Function

The deposit function lets the user deposit cash into their account and updates their balance after.

### Deposit(vector<User>& users) Function

This deposit is an overloaded function of the original. It is invoked when an administrator user wants to deposit money for a client. In this, the administrator must enter the username of client who needs money deposited. The vector of users is passed by reference into the function, it’s then used in a loop to search for the client of said username and their deposit function is called i.e user.deposit(). Thereby the system prompts how much money is to be deposited and the transaction is carried out.

### Check Balance Function

Outputs the balance in the user’s account.

### Change password Function

Let’s the user change their password.

The user must get their old password right, then they’ll enter a new password and confirm the new password in order for the change to be successful.

### Register User Function

This function registers a new user into the system. However, only an administrator can register a new user. Once the user is registered, they’re automatically logged in by their index in the vector of users being set as the value of loggedUser.

The function does not allow duplicate usernames in the systems as each user has to be unique. It ensures that by looping through all the existing looking for a matching username, if it fails to find one then the username is valid, else otherwise.

### Login User Function

This function enables a user to login and use the system. It prompts them for their username and password. When successful they can proceed with the program.

### Logout(const string & filename, vector<User>& users) Function

This logs the user out of the program. This is achieved by resetting the loggedUser variable back to -1 instead of the user’s index in the vector of users. Then the saveData(filename, users) function is called hence saving the new data.

### Lowercase Text Function

This is a supporting function that just makes life easier. In the code it’s called lcText(). What it does is that it takes a string of text as a parameter such as “Admin” and converts it all into lowercase: “admin”. It works for any form of capitalization that is applied on the text.

### Getter Functions

These are custom member functions of the user class that allow us to retrieve private information in the user object because we cannot retrieve private information directly. This is basically a way of securing object information.

There are 5 getter functions: getID(), getUsername(), getPassword(), getBalance(), getUType().

As suggested by their names, their return value is as respect to their names i.e. getUsername() returns the username of the user e.g. sarahsun.

### SaveData(vector<User> users) Function

This void function saves the information of the vector of users into the text file in csv format. In this, getter functions are utilized in order to retrieve private information about the user for storing.

Storage is done using the ofstream. A foreach loop is run to store each user from the vector of users in the usersDB.txt file until it is exhausted.

### Load Data Function

This function is of type vector, its return value is a vector of users. It reads the usersDB.txt file and retrieves the information and creates the users based on the saved data. It creates a User object for each line of data stored and places the ID, username, password, balance and userType into the object which is then pushed/appended into the vector of users.

Retrieval is done using ifstream.

### showMenu(vector<User>& users) Function

This function is a member function of the class User. It is used to show either the admin menu or client menu. It does this by checking if the logged in user is an admin or a client; it checks uType (ADMIN/NORMAL) hence shows their respective menus.

## Program Flow

The main function continuously loops through the displayMenu() so the program doesn’t close immediately.

If no one is logged in it shows:

1. Admin login
2. Client login
3. Exit

If an admin is logged in it shows:

1. Register user
2. Deposit money for client
3. Change password
4. Logout

If a client is logged in it shows:

1. Withdraw
2. Deposit
3. Check balance
4. Change password
5. Logout

One key thing to note is that where the ‘vector<User> users’ is passed as a parameter, it is passed by reference not by value. This is because passing by value creates a copy of the data which is disadvantageous because:

* It is slower because passing by value must create a copy of the data first.
* It does not update the original vector hence any changes made will be lost.

Also, upon first time running of the program, a default admin account is set up with username: admin and password: admin123. This enables the first admin to be able to register new clients and admins.