**Project Name:** Design An ATM Prototype

**Developed by:**

Mehmet Akif VARDAR

You can view the source code on GitHub by clicking [here](https://github.com/devmehmetakifv/ATMPrototype).

1. **Introduction**

The purpose of this report is to provide an overview and documentation of the ATM prototype developed in C++. The project is aimed to simulate the basic system of an ATM with an authentication system, database simulation and various functional components.

This report outlines the key features, implementation details and potential areas for improvements.

1. **Features**

This ATM prototype offers several key features listed below:

**2.1 – Authentication System**

The authentication system is responsible for handling user authentications. When user enters their account number, system checks if it exists in the database. If it is, then the authentication system asks user to enter the PIN number related to that account number. If the PIN is correct, then the user becomes authorized and taken to main menu.

If the entered account number hasn’t found in the database, the authentication system then lets user know that the account number doesn’t exist and asks if they want to register.

If user wants to register, system asks them a few questions such as their name, surname, age, identity number, preferred PIN and so on.

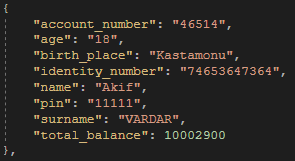
After getting these inputs, the authentication system creates a user object and registers the data to the database. After a successful registration, the generated account number and their PIN is displayed on the screen and user can use those credentials to login each time they run the software.

These functionalities will be explained with related code snippets in the later part of this report.

**2.2 – Database Simulation**

To simulate a database-server communication, a JSON file was used as a data storage mechanism. The JSON file acted as a representation of a database, containing user data such as account details and balances. This approach allowed for the management and retrieval of user information within the prototype.

The JSON file contains user columns and each user has some properties such as name or surname. Here’s a screenshot from the database:



As you can see from the screenshot, each user has an identical account number, an age, birth place, identity number, name, pin, surname and a total balance.

Throughout the development process, we’ll use looping techniques to pull data from these columns or update them. (increase or decrease balance etc.)

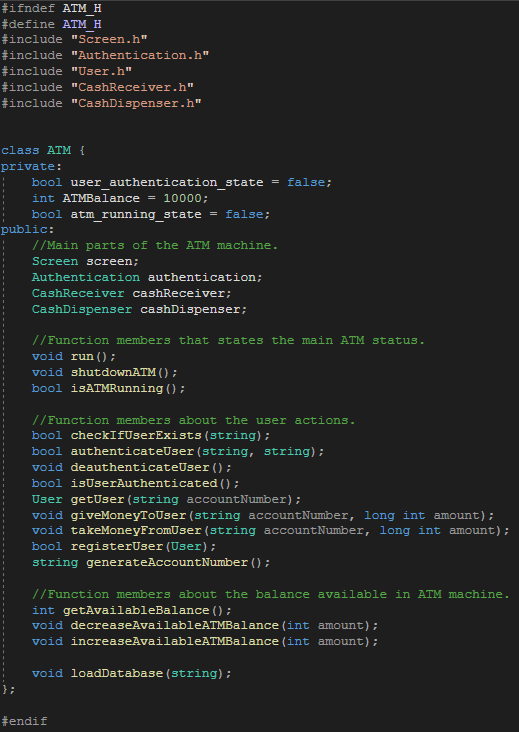
**2.3 – Class Representations**

Six main classes used for developing this ATM prototype. Let’s take a look at them one by one:

* + 1. **ATM**

ATM class was used for handling general functions and its considered as the main class of the prototype. It keeps some critical parameters such as user authentication state, current ATM balance and ATM running state. It also contains other classes.

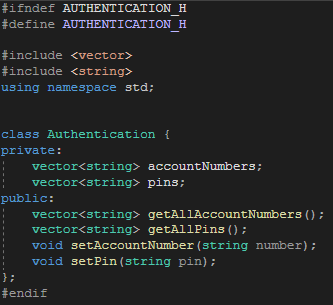
It has some public functions such as run(), shutdownATM() and isATMRunning() which are responsible for managing the general ATM functionalities. It also has some public functions for user management, such as checkIfUserExists(), authenticateUser(), deauthenticateUser()… You can see the general structure of the ATM class from the screenshot down below:



**2.3.2 Authentication**

Authentication class is used to store all account numbers and PINs in vector components received from the database. It also has some functions that lets ATM to get all the retrieved data in vector components to use in authentication process.

You can see the general structure of the Authentication class from the screenshot down below:

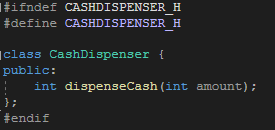


* getAllAccountNumbers() and getAllPins() returns the data inside the private accountNumbers and pins vectors.
* setAccountNumber() and setPin() is used to push given parameters into those private vectors.

**2.3.3 CashDispenser**

Cash dispenser class is used to return the amount of money to dispense. It is a basic class and it isn’t a vital class. The purpose of CashDispenser class is to represent a some sort of cash dispensing area of an ATM.

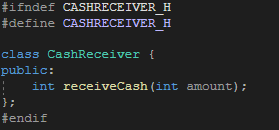
You can see the general structure of CashDispenser class from the screenshot down below:



**2.3.4 CashReceiver**

Cash receiver class is used to return the amount of money to withdraw. It is a basic class like CashDispenser and it isn’t vital for the functionality of the software. It’s purpose is to represent a some sort of cash receiving area of an ATM.

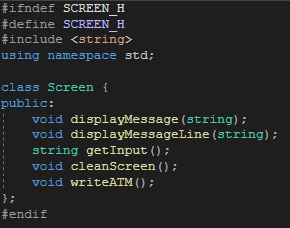
You can see the general structure of CashDispenser class from the screenshot down below:



* + 1. **Screen**

Screen class is a representation for an ATM’s screen and it’s used to display messages on the screen and get inputs from users. It also has some functionalities like cleaning the previous messages and write an ATM using keyboard symbols at the beginning of the program.

You can see the general structure of Screen class from the screenshot down below:



As you can see, there are 2 types of message displaying functions. This is because displayMessage() can display messages on one line, but displayMessageLine() can only display one message in one line.

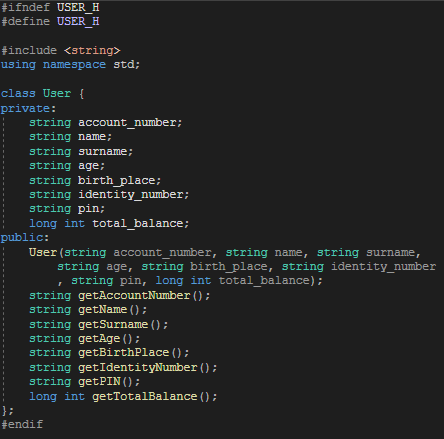
This functionality is useful in some cases, like when you have to display a string received from database etc.

* + 1. **User**

The User class within the program represents a user entity, encompassing properties such as identity number, age, name, total balance and much more.

When retrieving data of a specific user from the database, the values associated with these properties are assigned to an instance of the User class. This allows for the convenient utilization of the user’s data across various parts of the program.

You can see the general structure of the User class from the screenshot down below:



As you can see, the constructor of User class takes all neccessary parameters and assign them to the private members of the Class. Then, we can use these public functions to retrieve each specific data of the user object.

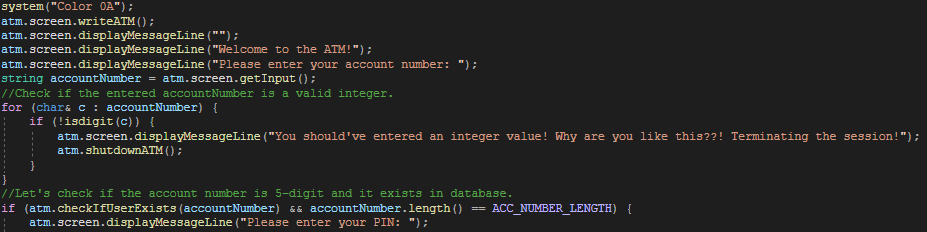
I’ll try to explain the usage of the user class more later in the report.

1. **Implementation Details**

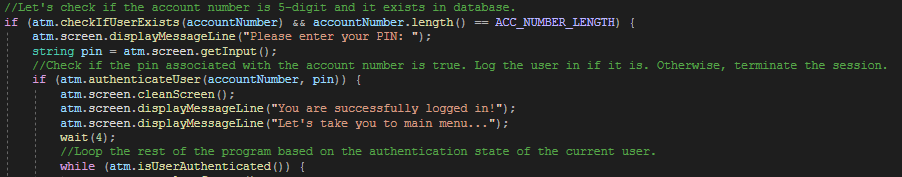
In this section, I’ll provide more details about the implementation of this ATM prototype and its various components.

* 1. **Implementation of the Authentication System**

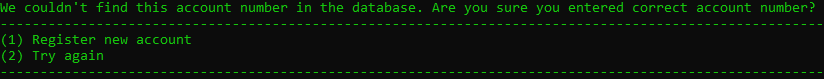
-In the beginning of the program, we prompt user to enter their account number. We first check if the account number is a proper integer. If it is, then we check if the entered account number is 5-digit and it exists in database:



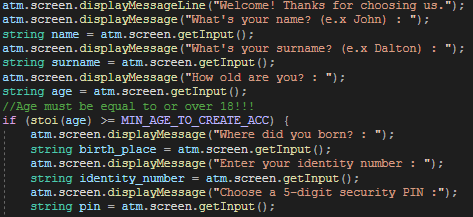
If the account number is exist and it is exactly 5-digit, then the program prompts user to enter their PIN. If the PIN matches with the account number they entered, program authenticaes the user and takes the them to main menu. Else, atm shutdowns itself and program closes:



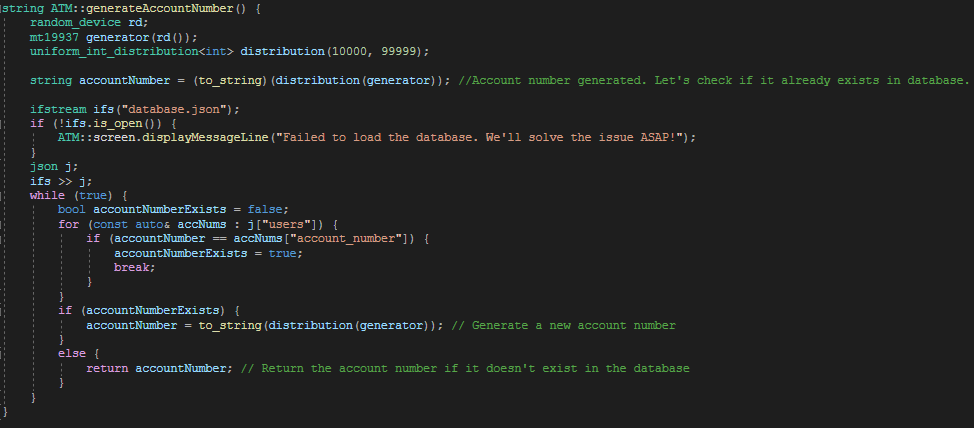
If the entered account number doesn’t exist in the database, the program asks user if they are a registered user. From this section, user can choose to register to the system:



If user chooses to register to the system, the program asks user bunch of questions for registration:



After these steps, the program generates a random 5-digit account number for the user. This is done by the following function:



As you can see from the function above, if the generated account number already exists in the database (which has a really low chance), it generates another number and keeps going until it generates a completely unique account number.

After the account number generation is completed, a new user object is created with the inputs received from the user during the registration process and new user data is inserted to the database using the registerUser() function:



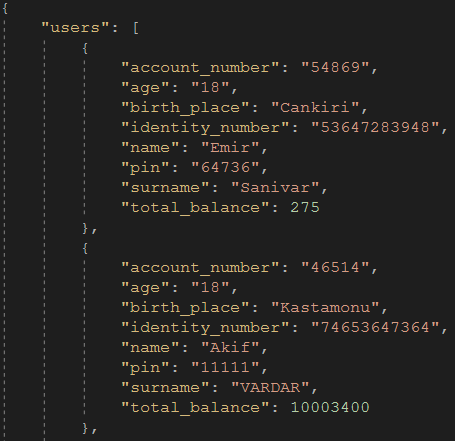
This function returns a boolean value, which then we can use to check if the registration process is completed successfully. If it is, then we display the account information on screen and ask user to login with their new credentials.

* 1. **Implementation of the Database System**

Simulating a database system means that we are keeping some important data like user login credentials and user identity informations in a static physical storage instead of keeping them in memory. That way, we can access them whenever we want.

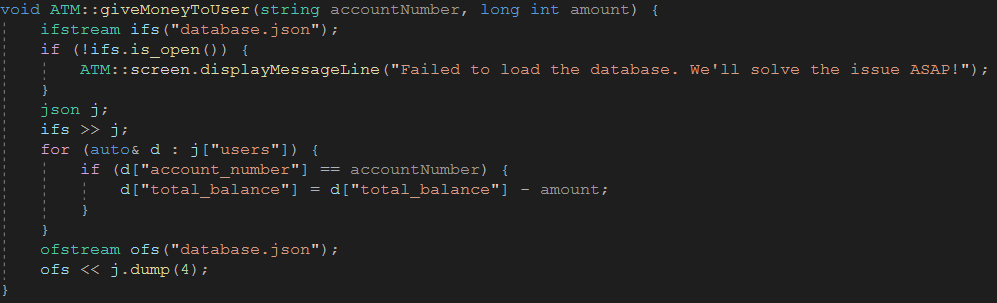
I used a JSON file to represent a database for this prototype. In this JSON file, there are user blocks being kept in “users” array and each user block has its own properties such as account number, age, name, surname and so on.

Let’s take a look at the database representation and explain what’s going on there:



You are looking at 2 user datas located in our JSON file. As you can see, they are seperated from each other with curly brackets and each block of data has its own unique property values.

Let’s say we want to give money to a user. To do that, we need to update the “total\_balance” property of the account number that we want to give money. We use the following function for this purpose:

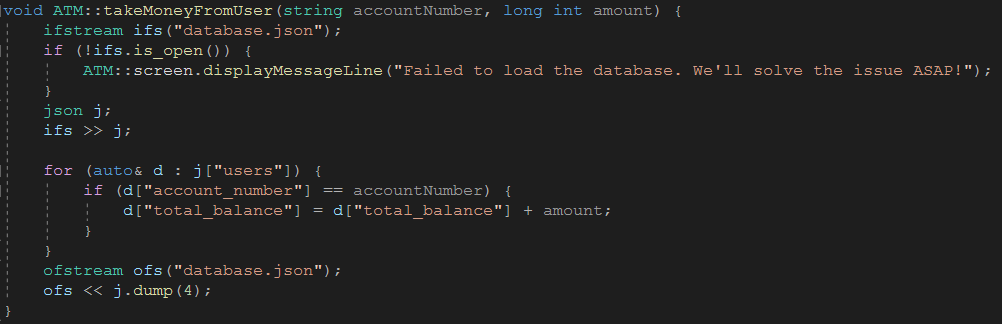


First of all, we open the database.json file. Assuming the file was opened successfully, we proceed to read the contents of database.json file into a ‘json’ object named ‘j’ using the >> operator.

Then, we proceed to iterate over each element in “users” array using a range-based for loop. Within the loop, we check if the “account\_number” of the current user object ‘d’ is equal to the given accounNumber. If it is, then we update the amount of “total\_balance” property by substracting the value “amount” from it.

Why do we subtract? It’s because we are giving money to the user physically, means that we should decrease the amount of money from their balance.

After updating the “total\_balance” property, we use an ofstream object to write all the json data to the “database.json” file, which basically means we are clicking “Save” button.



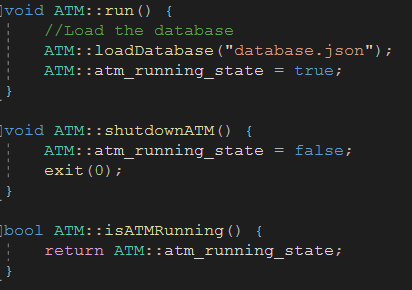
Here’s another very similiar function that uses the database simulation system to manage the data stored in the json file. It’s different because this time we add the amount value, instead of substracting.

* 1. **Implementation of Classes**

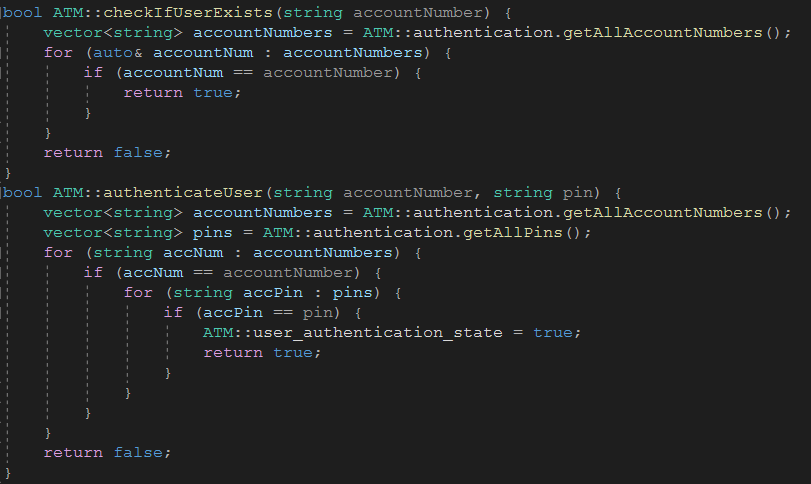
In this section, I’ll try to explain how I implemented each class’s public function member functionality is implemented. There are lots of functions, so I’ll explain them superficially.

* + 1. **ATM.cpp**

ATM class is the base class for this prototype and it contains most important functionalities.

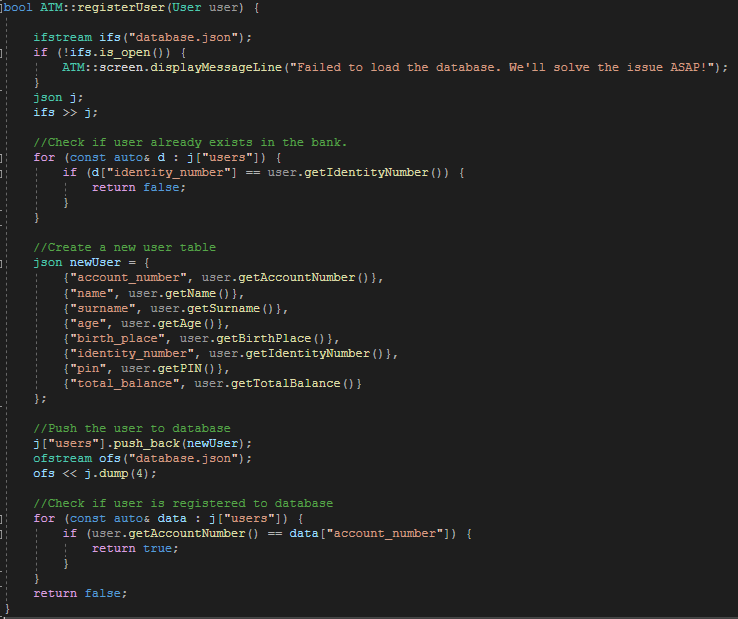


These 3 functions are responsible for starting or shutting down the ATM. There is also a function called “isAtmRunning()” which returns the running state of the ATM.



The function “checkIfUserExists()” is being used in logging system. It returns true if the account number given by user does exist in database. In this case, it returns true and the program asks for the PIN of this account number to user. Otherwise, it takes user to registration panel.

For “authenticateUser()” function, after we check for the account number in database and ask for the PIN, we take the input and check again if the given PIN matches with the one associated with the given account number in the database. If it is, the function authenticates the user by setting the “user\_authentication\_state” true and returns true.

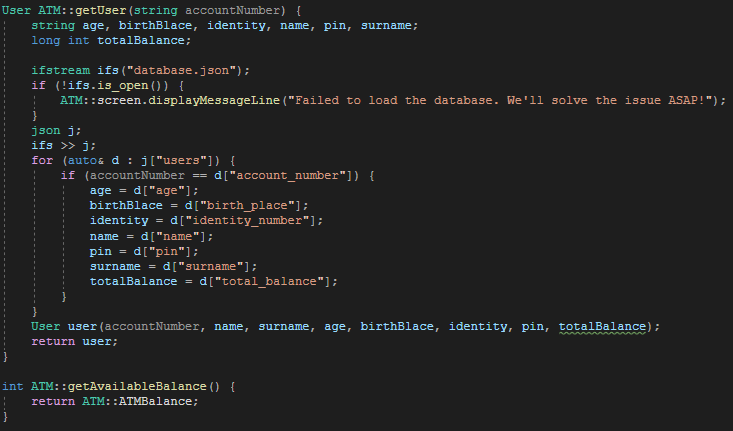


“registerUser()” function is used to register a user to the database. It takes a “user” object as the parameter and use it to read all the data. First, we loop through each user object in database and check if the user already exists in the bank by checking the identity numbers.

If they don’t exist in database, we create a new json object called “newUser” and pass all the neccessary property values from the “user” object we received as the function parameter.

Then, we push that json object to the “users” array located in our JSON file. After that, we save the file and close it.

In the last step, we check if the newly registered user’s account number exists in the database. If it does, the function returns true, means that registration process was successful and new user data is inserted to the database properly. Else, it returns a false, means that something went wrong and the process was unsuccessful.

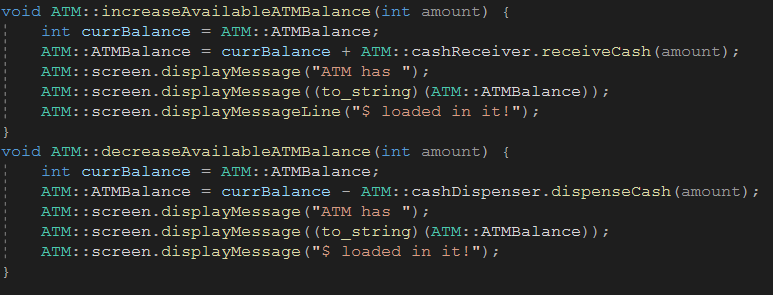


“getUser()” function is used to find a specific user by its account number in database and return it using the User model(class). We first define all required variables to model the data we are reading from database and open the JSON file.

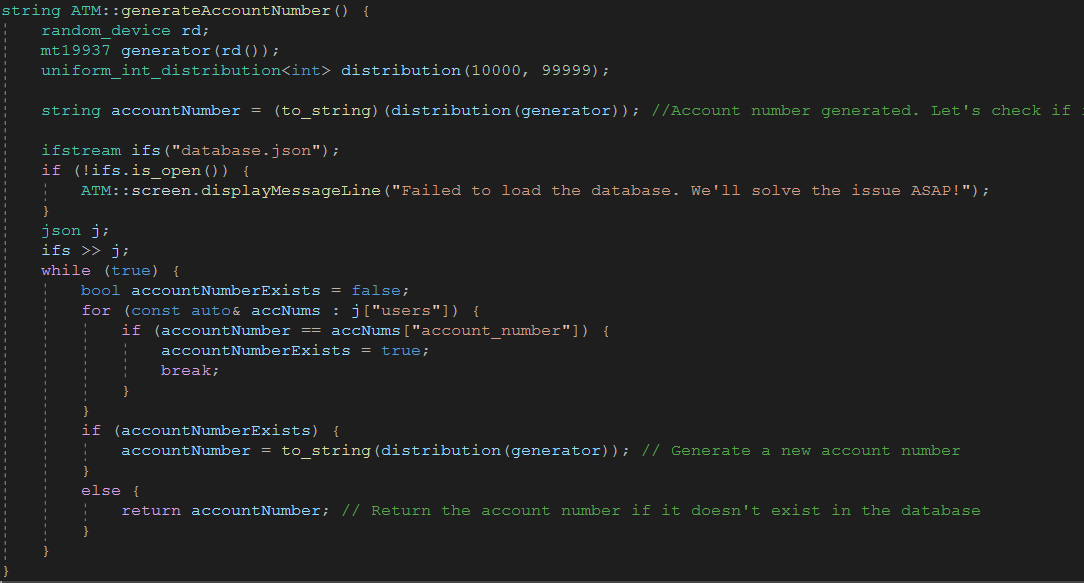
Then we iterate through each user object in “users” array and check if any of the account numbers matches with the account number given in as the function parameter. If an account number matches, we read the next properties and assign them to our newly initialized variables.

After that, we create the user object with those variables and return it.

“getAvailableBalance()” is used to return the current amount of money available in the ATM.



“increaseAvailableATMBalance()” and “decreaseAvailableATMBalance()” is used to increase and decrease the available balance in ATM. It takes an “amount” as parameter and updates the private ATMBalance property of the ATM class.

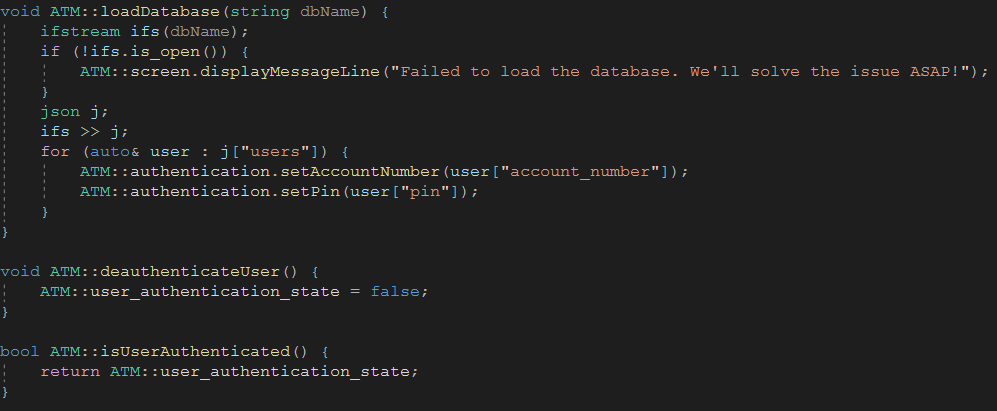


“generateAccountNumber()” is used to generate a 5-digit account number to assign an account number to a new user. It uses C++ built in random\_device, mt19937 and uniform\_int\_distribution classes to generate random 5-digit integer. Later, we transform the integer type to string type since we won’t use the account number in mathematical calculations, therefore we don’t need to define it as an integer.

After that, we once again open the database.json file to check if the generated account number already exists in the database. This is important since giving 2 different users same account numbers would cause a crucial flaw in the ATM’s system.

If it exists, we create another account number and check again. This loop goes forever since it’s in a while loop.

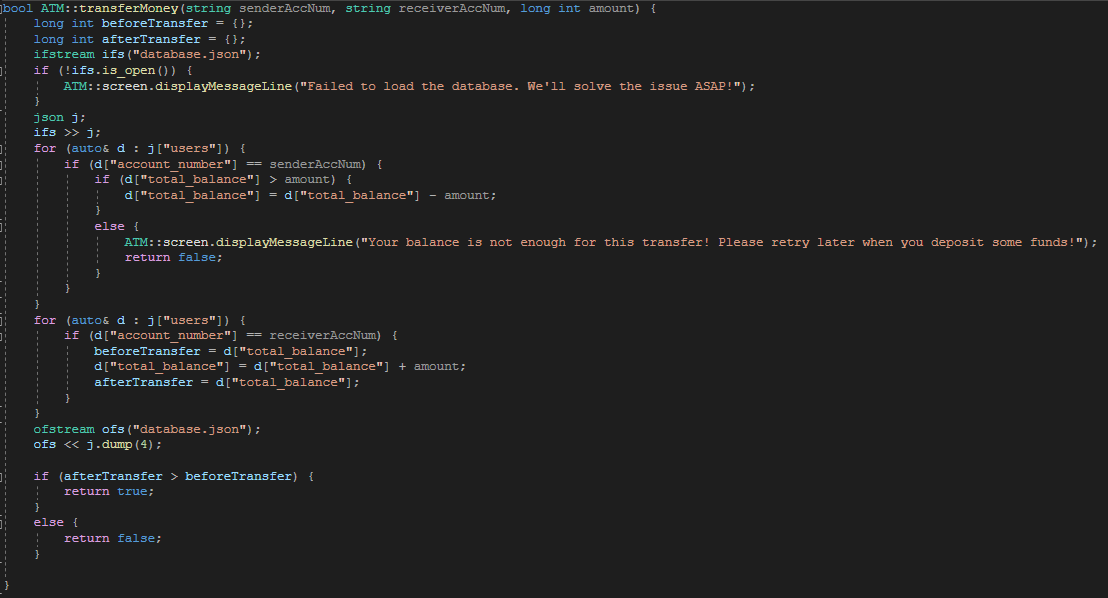
If it doesn’t exist, we just go ahead and return the account number.



“loadDatabase()” function is used to retrieve all of the account numbers and PINs in database and push them all to the related private vector class members of the Authentication class. Those datas will be used in authentication processes in the later of the program.

“deauthenticateUser()” is used to set the user\_authentication\_state to false, means that if the user somehow logs out runtime, this function is used to deauthenticate the user and terminate their session.

Similiarly, “isUserAuthenticated()” is used to check whether the user is authenticated or not. This function is useful in some cases as well.



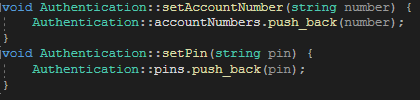
“transferMoney()” function is used to transfer an amount of money from one account to another using their account numbers. We first casually open the database.json file and read all the data into a json object.

We then look for the sender’s account number in the database by iterating through each user object in “users” array. When we find it, we simply decrease the amount from the “total\_balance” property.

After that, we do the same thing for the receiver’s account, but this time we add the amount to the “total\_balance” property, since the amount is being transferred into this account.

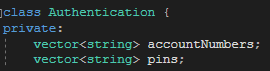
After all the changes and updates, we save the “database.json” file and if the amount after the transfer is bigger than the amount before the transfer, it means that the transfer is successful and the function returns true. Otherwise, the function returns false, means that an error ocurred during the transfer process.

* + 1. **Authentication.cpp**

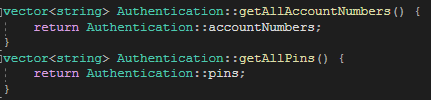


“setAccountNumber()” and “setPin()” functions are used to push data to the private vector class members called “accountNumbers” and “pins” of Authentication class. This is because those private vector class members will then be used in authentication processes.

You can see those members from the screen shot down below:



Vector types used because they are more flexible.



“getAllAccountNumbers()” and “getAllPins()” functions are used to return those private vector class members of the “Authentication” class. They return as vector<string>, so we can iterate through each element later when we need them.

* + 1. **CashDispenser.cpp**



CashDispenser class has only one public function named “dispenseCash” which simply returns the amount that has been passed in as the function parameter.

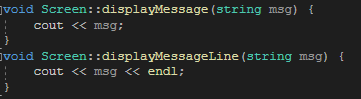
This looks like it doesn’t do anything and in reality, it actually doesn’t. But it is included since the task is to create this prototype using OOP techniques and I wanted to visualise an ATM by creating each part of it using classes.

* + 1. **CashReceiver.cpp**



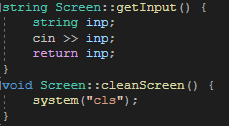
Just like CashDispenser.cpp, CashReceiver.cpp has the exact same purpose. These classes are only used to visualise the ATM by creating a class for every main part of it.

* + 1. **Screen.cpp**



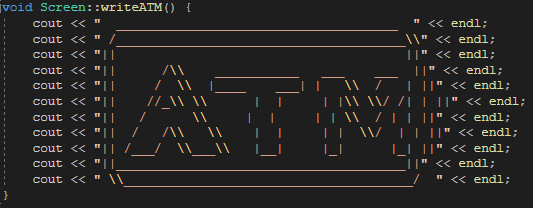
Screen.cpp is used to visualise the screen of an ATM. Therefore, it contains functions which are used to print messages or datas on the screen.

Functions displayMessage() and displayMessageLine() have similiar functionality. displayMessage() is used to display any amount of messages in one line. displayMessageLine() is used to display one message to one line.



getInput() function is used to get a string type of input from user and return it.

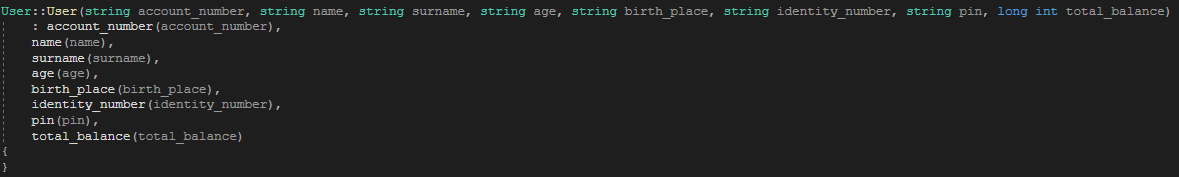
cleanScreen() function is used to clean previous messages displayed on the ATM screen, providing a fresh UI for the user.



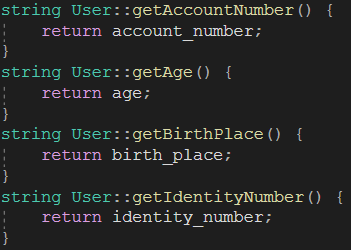
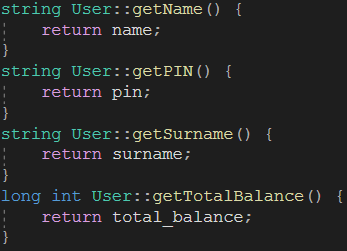
Function “writeATM()” is used to write an “ATM” on the screen using keyboard symbols. It looks weird like this but it looks wonderful in program, don’t worry : ) .

* + 1. **User.cpp**

User.cpp is used to construct a user object with all required parameters and return those parameters in need using public get functions.



You can see that the constructor of User class is used to instantiate a user object by passing in all required parameters. By that, we can initialize and model a user object easily.

With these public getter functions, we can easily access any property of a user object by just calling its get() function. This was very useful is some scenarios where a User object returned from a function and we can access the values of it by using these getter functions.

1. **Conclusion**

In conclusion, the ATM prototype project developed in C++ using Object-Oriented Programming (OOP) techniques has achieved its objectives by implementing a comprehensive set of functionalities.

The prototype incorporated an authentication system, a database simulation system and various functional components, including the user interface, cash receiver and cash dispenser.

Overall, the ATM project showcases the successful application of OOP techniques in developing a functional and efficient ATM prototype. It combines essential features such as authentication, database simulation, user interface, cash receiver and cash dispenser to provide a comprehensive and user-friendly experience.

While the project has achieved its primary objectives, there are potential areas for further improvement and expansion. These may include:

* Enhancements for error handling,
* Additional security measures,
* Improved user feedback

And the implementation of additional features such as:

* Balance inquiries,
* Transaction history,
* Account management options,

and so on.

In summary, the ATM project represents a basic accomplishment in implementing an OOP-based ATM system. It demonstrates the basic integration of various components and functionalities neccessary for a functional and secure ATM.