**C++ PROJECT**

**\_\_\_\_ ATM WORKING SYSTEM\_\_\_\_**

****

**UDUMULA NEELA LOHITHA SUSMITHA REDDY**

**(AP23110010368)**

THE CODE BELOW INCLUDES ATM WORKING SYSTEM

THIS CODE IS USED TO DO THE FOLLOWING **:**

**Acknowledgments**

I would like to express my gratitude to everyone who contributed to the development of this ATM code project. This project was a valuable learning experience, and it would not have been possible without the support and guidance of our course coordinator.

I also wish to thank my team members who offered their assistance, shared resources, and engaged in constructive discussions. Their collaboration helped me overcome challenges and refine the code to enhance its functionality and user experience.

Finally, a special thanks to my friends for their encouragement and patience during the long hours of development. Their unwavering support was a source of motivation and perseverance.

Thank you all for your contributions and support in making this project a success.

**Table of Contents**

[Certificate 1](#_gjdgxs)

[Acknowledgements 3](#_30j0zll)

[Table of Contents 4](#_1fob9te)

[Abstract 6](#_3znysh7)

[Abbreviations 8](#_3dy6vkm)

[List of Tables 10](#_1t3h5sf)

[List of Figures 12](#_4d34og8)

[List of Equations 14](#_2s8eyo1)

[1. Introduction 1](#_17dp8vu)

[1.1 Program Structure and Features 1](#_3rdcrjn)

[2. Methodology 5](#_lnxbz9)

[1.1 Heading 2 5](#_35nkun2)

[1.1.1 Heading 3 5](#_1ksv4uv)

[Discussion 7](#_44sinio)

[Concluding Remarks 9](#_2jxsxqh)

[Future Work 11](#_z337ya)

[References 13](#_3j2qqm3)

**Abstract**

This C++ program is a simplified ATM simulation designed to demonstrate essential Object-Oriented Programming (OOP) principles such as encapsulation, inheritance, and polymorphism. The program features three main classes: AccountHolder, Admin, and ATM, each performing distinct roles to emulate a realistic ATM system. The AccountHolder class models individual bank users with secure PIN encryption, balance management, and encapsulated access methods. The Admin class inherits from AccountHolder to add administrative privileges, such as resetting user balances, showcasing inheritance and polymorphism. The ATM class serves as the interface, enabling users and admins to interact with the system securely through PIN verification and account access.

A simple XOR-based encryption function is used to protect PIN data, while the ATM class handles common transactions, including balance inquiry, deposit, and withdrawal. The program’s structure and interactive flow allow users to simulate realistic ATM operations, while showcasing fundamental OOP concepts and basic data security techniques. This project provides an illustrative example of how OOP principles and basic encryption can be applied to design a user-focused, secure ATM system simulation in C++.

**Abbreviations**

Abbreviations used in the code and their meanings:

1. **ATM** - Automated Teller Machine
   * A device that allows users to perform financial transactions (withdrawals, deposits, balance inquiries) without bank staff assistance.
2. **PIN** - Personal Identification Number
   * A numeric code used by the account holder to verify identity and access account features securely.
3. **AccNo** - Account Number
   * A unique identifier for each bank account, used to manage and secure transactions.
4. **OOP** - Object-Oriented Programming
   * A programming paradigm that structures code around objects and uses principles like encapsulation, inheritance, and polymorphism.

**1. Introduction**

This C++ program is a fully functional simulation of an ATM (Automated Teller Machine) system, designed to demonstrate key Object-Oriented Programming (OOP) principles, including encapsulation, inheritance, and polymorphism, as well as basic encryption techniques. The program is structured around three main classes—**AccountHolder, Admin,** and **ATM .**

## **1.1 Program Structure and Features**

The ATM program’s structure includes key classes and functions for secure transactions. AccountHolder manages user data and basic actions like balance inquiry, deposit, and withdrawal. The Admin class, inheriting from AccountHolder, adds functionality for resetting user balances. ATM handles user interactions, verifying PINs and guiding transactions. A helper function, encryptDecrypt, provides basic PIN encryption. Core features include secure access control, balance management, and session continuity. Encapsulation, inheritance, and polymorphism are applied, enabling a structured, role-based ATM experience with clear user and admin distinctions.

**1.2 Encryption Mechanism:**

This encryption function secures the account holders' PINs, demonstrating basic data protection techniques. The **encryptDecrypt** function uses encryption to both encrypt and decrypt PINs, simulating a secure data handling process. Though not intended for real-world security, this technique highlights fundamental encryption concepts.

**1.3 AccountHolder Class (Encapsulation and Inheritance)**:

This class represents a generic account holder in the ATM system, with essential attributes like **accountNumber**, **name**, **balance**, and **encryptedPin**.

The class also includes functions to:

* Verify PINs securely (by decrypting and comparing with the user input)
* Withdraw funds (with a balance check)
* Deposit funds
* Display account details.

These features demonstrate secure handling of sensitive data and realistic financial operations.

**1.4 Admin Class (Inheritance and Polymorphism):**

The Admin class inherits from **AccountHolder**, highlighting inheritance, with an added layer of authority to reset user balances. This admin feature is an example of access control and privilege differentiation within the system.

* The Admin class overrides the showDetails function from **AccountHolder**, showcasing polymorphism by customizing the display of details for an admin role.
* The admin can also perform a resetBalance operation, allowing administrative adjustments to users' account balances—an essential feature in real-world banking.

**1.5 ATM Class**:

* This class serves as the central interface for users and admins to interact with the ATM system.
* It includes:
  + An accessAccount function that handles basic ATM operations for users (view details, withdraw, deposit),
  + A findAccount function to locate account holders by account number or name, ensuring ease of access.
* Through the accessAccount function, the ATM interface ensures users can only access functions after PIN verification, simulating a secure transaction environment.

**1.6 Main Function (User Interaction)**:

* The main function provides a user interface, offering choices between user and admin access. It guides users through the options to enter an account number or name, validate PINs, and perform transactions.
* Admins, once verified, can view their own details and reset other users' balances. Users are also given the option to exit or restart the ATM session for further transactions.
* The loop in main allows repeated interactions with the ATM, giving users and admins an authentic ATM session experience.

**2. Methedology**

**2.1 Explaining the Class AccountHolder :**

Code snippet for Class AccountHolder:

class AccountHolder {

protected:

string accountNumber;

string name;

double balance;

string encryptedPin;

public:

AccountHolder(string accNo, string userName, double bal, string pin) {

accountNumber = accNo;

name = userName;

balance = bal;

encryptedPin = encryptDecrypt(pin); // PIN encryption

}

virtual void showDetails() { // Polymorphism (virtual function)

cout << "Account Number: " << accountNumber << endl;

cout << "Name: " << name << endl;

cout << "Balance: ₹" << balance << endl;

}

// Getter for name (Encapsulation)

string getName() const {

return name;

}

// Getter for account number (Encapsulation)

string getAccountNumber() const {

return accountNumber;

}

// Getter for balance (Encapsulation)

double getBalance() const {

return balance;

}

// Setter for balance (Encapsulation)

void setBalance(double newBalance) {

balance = newBalance;

}

// Encapsulated method to verify PIN

bool verifyPin(string pin) {

return encryptDecrypt(encryptedPin) == pin; // Decrypt and verify

}

// Encapsulated method to withdraw money

bool withdraw(double amount) {

if (amount <= balance) {

balance -= amount;

cout << "Withdrawal successful! New balance: ₹" << balance << endl;

return true;

} else {

cout << "Insufficient balance!" << endl;

return false;

}

}

// Encapsulated method to deposit money

void deposit(double amount) {

balance += amount;

cout << "Deposit successful! New balance: ₹" << balance << endl;

}

};

**Data Members:**

1. **Protected Members**:
   * accountNumber: Stores the account holder's unique account number.
   * name: Stores the name of the account holder.
   * balance: Stores the current balance in the account.
   * encryptedPin: Stores the encrypted form of the account holder's PIN for secure access.

**Constructor:**

* AccountHolder(string accNo, string userName, double bal, string pin): Initializes the account with account number, name, balance, and PIN. The PIN is encrypted using the encryptDecrypt function before storage, adding a layer of security.

**Member Functions:**

1. **showDetails** :
   * This virtual function displays account details (account number, name, and balance).
   * Using virtual enables polymorphism, allowing derived classes to override this function if needed.
2. **Encapsulation :**
   * getName, getAccountNumber, getBalance: These functions retrieve the name, account number, and balance, respectively.
   * setBalance: Updates the account balance.
   * These methods allow controlled access to private/protected data, adhering to encapsulation principles.
3. **Security - PIN Verification**:
   * verifyPin: Verifies the entered PIN by decrypting encryptedPin and comparing it with the input.
   * This method ensures that only users with the correct PIN can access the account.
4. **Transaction Methods**:
   * withdraw: Allows the user to withdraw a specified amount from the account. It checks if the balance is sufficient before deducting the amount. If successful, it updates the balance and displays the new balance.
   * deposit: Allows the user to deposit a specified amount into the account, updating the balance and displaying the new total.

**2.2 Explaining the class Admin :**

The Admin class is a specialized version of AccountHolder with added privileges:

1. **Constructor**: Initializes an admin account using the AccountHolder constructor for common account properties.
2. **Overridden showDetails()**: Displays admin-specific details (account number and name) instead of standard account info. This demonstrates polymorphism by customizing the showDetails() function for admin users.
3. **resetBalance()**: Allows the admin to reset the balance of any AccountHolder. This method is unique to Admin, showcasing its extra privileges over regular users.

**The code for Admin class is**

class Admin : public AccountHolder {

public:

    Admin(string accNo, string userName, double bal, string pin) : AccountHolder(accNo, userName, bal, pin) {}

    // Overridden function for showing admin details

    void showDetails() override { // Polymorphism

        cout << "Admin Account Number: " << accountNumber << endl;

        cout << "Admin Name: " << name << endl;

    }

    // Admin function to reset user balance

    void resetBalance(AccountHolder &user, double newBalance) {

        user.setBalance(newBalance);  // Use setter for balance

        cout << "Balance reset successful for " << user.getName() << ". New balance: ₹" << newBalance << endl;

    }

};

**2.3 Explaining Main Function :**

The main function in this code serves as the entry point for a simple ATM program that allows users and an admin to access and manage account-related tasks.

**Key Steps:**

1. **Create Account Holders and Admin**:

* A vector of AccountHolder objects (users) is created to simulate three user accounts, each initialized with an account number, name, balance, and PIN.
* An Admin object (admin1) is also created, with admin-specific privileges like resetting user balances.

1. **ATM Object Creation**:

* An ATM object (atmMachine) is created to handle account operations like accessing accounts, verifying PINs, and performing transactions.

1. **User Interaction Loop**:

* The program enters a loop to allow multiple sessions without restarting the program.
* The user is prompted to select their type: **1 for User** and **2 for Admin**.

1. **User-Specific Operations**:

If the user selects 1 (User):

* They enter either their account number or name to identify their account.
* The findAccount function of atmMachine searches for the matching account in users. If found, the accessAccount method allows the user to view details, withdraw, deposit, or exit.
* If the account is not found, a message is displayed, and they are redirected back to the main menu.

1. **Admin-Specific Operations**:

* If the user selects 2 (Admin):
* The admin must enter their PIN for verification. If the entered PIN matches the encrypted PIN, the admin gains access to additional functionalities:
* **View Admin Details**: Displays the admin’s account details.
* **Reset User Balance**: Allows the admin to reset a user’s balance.
* The admin enters an account number or name to identify the user, and if the user is found, the admin enters a new balance, which is updated.
* **Exit**: Ends the admin session and returns to the main menu.
* If the PIN is incorrect, an "Invalid Admin PIN" message is shown.

1. **Repeat Prompt**:

* After each session, the program asks if the user wants to restart the ATM process.
* If the user enters y or Y, the loop restarts, allowing another session. Any other input exits the loop and ends the program.

1. **Exit Message**:

* When the loop ends, a "Goodbye!" message is printed to indicate that the ATM session has ended.

**Code snippet for Main function:**

int main() {

    // Creating multiple account holders

    vector<AccountHolder> users = {

        AccountHolder("123456", "John Doe", 1000.0, "1234"),

        AccountHolder("789101", "Jane Smith", 2000.0, "5678"),

        AccountHolder("112233", "Alice Brown", 1500.0, "4321")

    };

    Admin admin1("000001", "Admin", 0.0, "admin");

    // Creating an ATM object

    ATM atmMachine;

    char repeat;

    do {

        int userType;

        cout << "Select user type: 1. User 2. Admin: ";

        cin >> userType;

        if (userType == 1) {

            string identifier;

            cout << "Enter Account Number or Name: ";

            cin.ignore();  // To handle newline after the previous input

            getline(cin, identifier);

            AccountHolder \*user = atmMachine.findAccount(users, identifier);

            if (user != nullptr) {

                atmMachine.accessAccount(\*user);

            }

        } else if (userType == 2) {

            string adminPin;

            cout << "Enter Admin PIN: ";

            cin >> adminPin;

            if (admin1.verifyPin(adminPin)) {

                int adminChoice;

                do {

                    cout << "\n1. View Admin Details\n2. Reset User Balance\n3. Exit\n";

                    cout << "Enter choice: ";

                    cin >> adminChoice;

                    if (adminChoice == 1) {

                        admin1.showDetails();

                    } else if (adminChoice == 2) {

                        string identifier;

                        cout << "Enter Account Number or Name: ";

                        cin.ignore();  // To handle newline after the previous input

                        getline(cin, identifier);

                        AccountHolder \*user = atmMachine.findAccount(users, identifier);

                        if (user != nullptr) {

                            double newBalance;

                            cout << "Enter new balance for user: ";

                            cin >> newBalance;

                            admin1.resetBalance(\*user, newBalance);

                        }

                    }

                } while (adminChoice != 3);

            } else {

                cout << "Invalid Admin PIN!" << endl;

            }

        }

        cout << "Do you want to restart the ATM process? (y/n): ";

        cin >> repeat;

    } while (repeat == 'y' || repeat == 'Y');

    cout << "ATM session ended. Goodbye!" << endl;

    return 0;

}

**CPP Code for ATM Mechanism:**

#include <iostream>

#include <string>

#include <vector>

using namespace std;

string encryptDecrypt(string toEncrypt) {

    char key = 'K';

    string output = toEncrypt;

    for (size\_t i = 0; i < toEncrypt.size(); i++) {

        output[i] = toEncrypt[i] ^ key;

    }

    return output;

}

class AccountHolder {

protected:

    string accountNumber;

    string name;

    double balance;

    string encryptedPin;

public:

    AccountHolder(string accNo, string userName, double bal, string pin) {

        accountNumber = accNo;

        name = userName;

        balance = bal;

        encryptedPin = encryptDecrypt(pin);

    }

    virtual void showDetails() {

        cout << "Account Number: " << accountNumber << endl;

        cout << "Name: " << name << endl;

        cout << "Balance: ₹" << balance << endl;

    }

    string getName() const {

        return name;

    }

    string getAccountNumber() const {

        return accountNumber;

    }

    double getBalance() const {

        return balance;

    }

    void setBalance(double newBalance) {

        balance = newBalance;

    }

    bool verifyPin(string pin) {

        return encryptDecrypt(encryptedPin) == pin;

    }

    bool withdraw(double amount) {

        if (amount <= balance) {

            balance -= amount;

            cout << "Withdrawal successful! New balance: ₹" << balance << endl;

            return true;

        } else {

            cout << "Insufficient balance!" << endl;

            return false;

        }

    }

    void deposit(double amount) {

        balance += amount;

        cout << "Deposit successful! New balance: ₹" << balance << endl;

    }

};

class Admin : public AccountHolder {

public:

    Admin(string accNo, string userName, double bal, string pin) : AccountHolder(accNo, userName, bal, pin) {}

    void showDetails() override {

        cout << "Admin Account Number: " << accountNumber << endl;

        cout << "Admin Name: " << name << endl;

    }

    void resetBalance(AccountHolder &user, double newBalance) {

        user.setBalance(newBalance);

        cout << "Balance reset successful for " << user.getName() << ". New balance: ₹" << newBalance << endl;

    }

};

// ATM Machine Class

class ATM {

public:

    void accessAccount(AccountHolder &user) {

        string pin;

        cout << "Enter PIN: ";

        cin >> pin;

        if (user.verifyPin(pin)) {

            int choice;

            do {

                cout << "\n1. View Details\n2. Withdraw\n3. Deposit\n4. Exit\n";

                cout << "Enter choice: ";

                cin >> choice;

                double amount;

                switch (choice) {

                    case 1:

                        user.showDetails();

                        break;

                    case 2:

                        cout << "Enter amount to withdraw: ";

                        cin >> amount;

                        user.withdraw(amount);

                        break;

                    case 3:

                        cout << "Enter amount to deposit: ";

                        cin >> amount;

                        user.deposit(amount);

                        break;

                    case 4:

                        cout << "Exiting...\n";

                        break;

                    default:

                        cout << "Invalid choice! Try again.\n";

                        break;

                }

            } while (choice != 4);

        } else {

            cout << "Invalid PIN!" << endl;

        }

    }

    AccountHolder\* findAccount(vector<AccountHolder> &users, const string &identifier) {

        for (auto &user : users) {

            if (user.getAccountNumber() == identifier || user.getName() == identifier) {

                return &user;

            }

        }

        cout << "Account not found!" << endl;

        return nullptr;

    }

};

int main() {

    vector<AccountHolder> users = {

        AccountHolder("123456", "John Doe", 1000.0, "1234"),

        AccountHolder("789101", "Jane Smith", 2000.0, "5678"),

        AccountHolder("112233", "Alice Brown", 1500.0, "4321")

    };

    Admin admin1("000001", "Admin", 0.0, "admin");

    ATM atmMachine;

    char repeat;

    do {

        int userType;

        cout << "Select user type: 1. User 2. Admin: ";

        cin >> userType;

        if (userType == 1) {

            string identifier;

            cout << "Enter Account Number or Name: ";

            cin.ignore();

            getline(cin, identifier);

            AccountHolder \*user = atmMachine.findAccount(users, identifier);

            if (user != nullptr) {

                atmMachine.accessAccount(\*user);

            }

        } else if (userType == 2) {

            string adminPin;

            cout << "Enter Admin PIN: ";

            cin >> adminPin;

            if (admin1.verifyPin(adminPin)) {

                int adminChoice;

                do {

                    cout << "\n1. View Admin Details\n2. Reset User Balance\n3. Exit\n";

                    cout << "Enter choice: ";

                    cin >> adminChoice;

                    if (adminChoice == 1) {

                        admin1.showDetails();

                    } else if (adminChoice == 2) {

                        string identifier;

                        cout << "Enter Account Number or Name: ";

                        cin.ignore();

                        getline(cin, identifier);

                        AccountHolder \*user = atmMachine.findAccount(users, identifier);

                        if (user != nullptr) {

                            double newBalance;

                            cout << "Enter new balance for user: ";

                            cin >> newBalance;

                            admin1.resetBalance(\*user, newBalance);

                        }

                    }

                } while (adminChoice != 3);

            } else {

                cout << "Invalid Admin PIN!" << endl;

            }

        }

        cout << "Do you want to restart the ATM process? (y/n): ";

        cin >> repeat;

    } while (repeat == 'y' || repeat == 'Y');

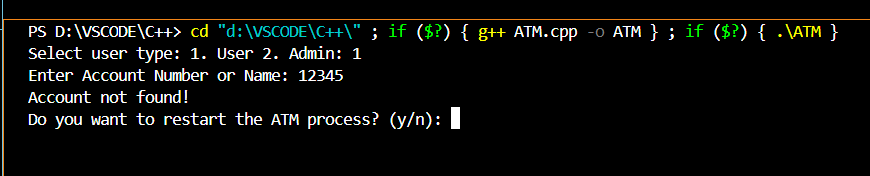
    cout << "ATM session ended. Goodbye!" << endl;

    return 0;

}

**Discussion/Outputs:**

In the first output sample user entered the incorrect Account Number or Name then the output gives like bellow



In 2nd output, the sample user entered the correct Account Number he got to verify with the encrypted PIN after verifying that he got 4 options to choose

1. View Details

2. Withdraw

3. Deposit

4. Exit

If users choose 1, they get their account details with name, account number & balance.

If users choose 2, they get asked how much they want to withdraw. The user enters that amount, and the money is deducted from the account.

If users choose 3 they get to add money to the account.

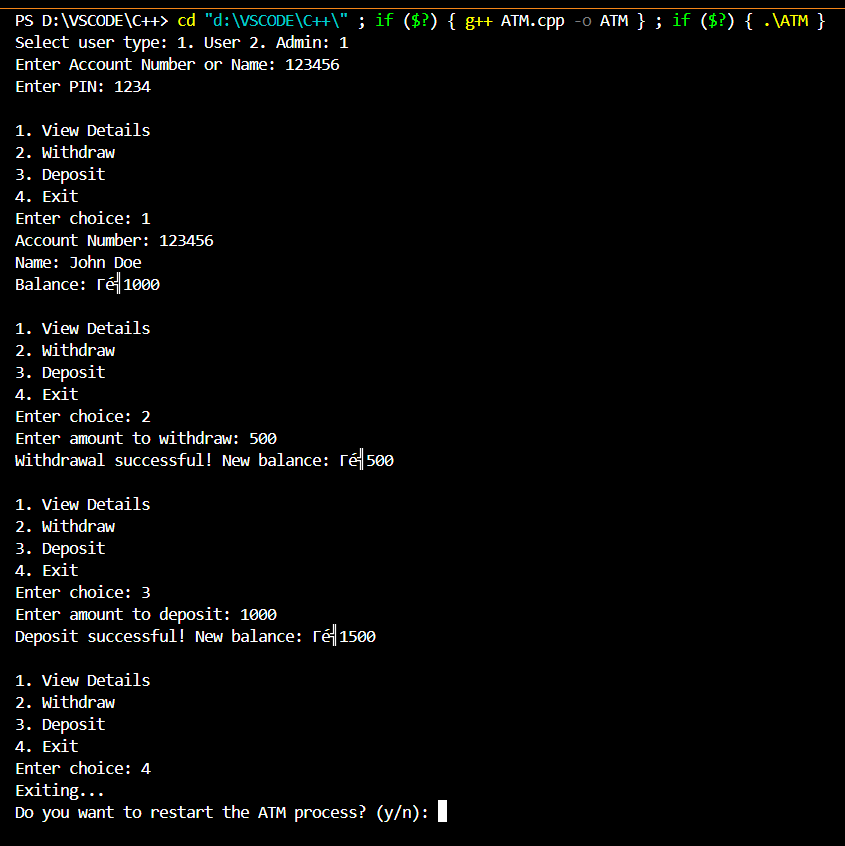
If the user chooses 4 the compiler terminates & asks to continue (or) exit.

If the user enters the Incorrect PIN then they get its Incorrect PIN & ask to continue (or) exit.

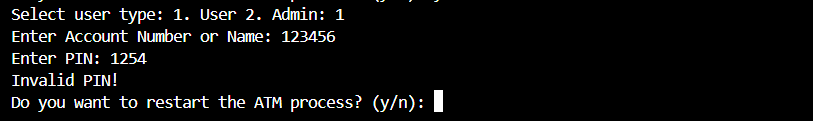
Here we use the While loop to continue the program or exit the program.

This is the given sample output for that user

This is the Output for all Entered details are correct:



This is the output for an incorrect PIN:



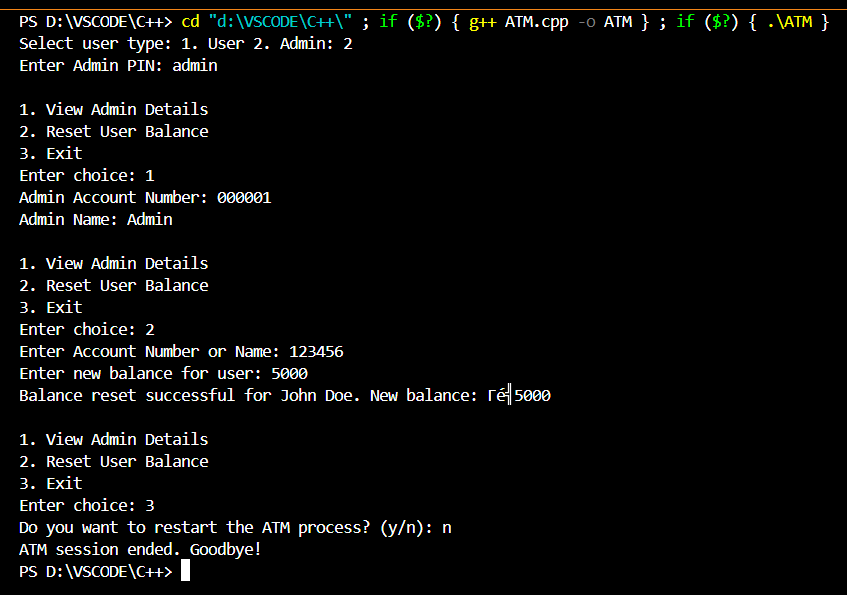
In 3rd Output if Admin can login with PIN

In the 1st option Admin check there own details

In 2nd option Admin can change the balance of the users.

In 3rd option they can exit from the code

This is the sample output is given below



**Concluding Remarks:**

In conclusion, this project provides a practical implementation of a basic ATM management system, demonstrating the power of object-oriented programming (OOP) in solving real-world problems. Using inheritance, encapsulation, and polymorphism, the code efficiently models distinct roles for users and administrators within the ATM system. The system allows users to securely access account functions, such as viewing account details, depositing, and withdrawing funds. Administrators, with elevated privileges, can manage user account balances, enhancing the flexibility of the ATM’s functionality.

The code provided demonstrates a simple ATM simulation program with functionality for both regular users and administrators. Using object-oriented programming principles like inheritance, encapsulation, and polymorphism, it efficiently models an ATM system.

Key features include:

* **User Account Access:** Users can securely access their accounts, view details, withdraw, and deposit money after PIN verification.
* **Admin Privileges:** The administrator can view admin details, reset user account balances, and manage user data with elevated privileges.
* **PIN Security:** The use of simple XOR encryption for PIN storage and verification provides basic security for sensitive information.
* **Encapsulation of Data:** Data like account balance and PIN are kept private and accessed only through controlled methods.

**References:**

1. E Balagurusamy, 8th Edition, year 2022 of **Object Oriented Programming with C++ .**

2. Paul Deitel, 9th Edition, **C++ How to Program.**