***Report on***

***Bank Management System***

***Prepared For***

**Ali Akber Rudra**

**Course Code: SE133**

**Course Name: Software Development Capstone Project**

***Prepared By***

**Lab Section: C1**

**Group Name: Star Players**

**Md. Fardin Hasan, 0242310005341393**

**Maliha Farzana Muneyat, 0242310005341062**

**Erin Jahan Eshita, 0242310005341074**

**Date: 15th May 2024**

****

**Department of Software Engineering**

**Daffodil International University**

|  |  |
| --- | --- |
| **Table Of Contents** | |
| **Contents** | **Number Of pages** |
| *Context Of the Project* | **3-4** |
| *Project Summary* | **4-6** |
| *Feasibility Analysis* | **7-9** |
| *Requirement Analysis* | **9-11** |
| *Data Flow Diagram* | **12-14** |
| *Features Of the Project* | **14-16** |
| *Testing Of the Project* | **16-28** |
| *Features Checklist* | **28-29** |
| *Contributions* | **30** |
| *References* | **31** |

**Context of the Project: Bank Management System**

**Project Overview:**

The Bank Management System project is a software application designed to simulate basic banking operations using the C programming language. This project aims to provide a simple, user-friendly interface for bank customers to perform essential banking activities such as depositing money, withdrawing money, transferring money

and viewing account details. The system is built to handle the basic needs of a customer in a bank and ensure secure and efficient management of customer data and transactions.

**Objectives:**

* **Customer Registration and Authentication:** Allow new users to register by creating an account with a unique phone number and password. Existing users can log in using their credentials.
* **Deposit Money:** Enable users to deposit money into their account, updating their balance accordingly.
* **Withdraw Money:** Allow users to withdraw money from their account, ensuring sufficient balance before processing the withdrawal.
* **Transfer Money**: Facilitate money transfers between accounts, ensuring the sender has enough balance to complete the transfer.
* **Account and Transaction Details**: Provide users with the ability to view their account details and transaction history.

**Features**:

* **User Registration and Login:** Secure registration process that stores user data in individual files and a login mechanism to authenticate users.
* **Deposit Functionality:** Allows users to deposit a specified amount into their account and updates the balance. Withdrawal
* **Functionality:** Enables users to withdraw funds, ensuring that the account has sufficient balance.
* **Money Transfer:** Supports transferring money between accounts with balance validation.
* **Account Details:** Displays user's account information including name, phone number, account number and current balance.
* **Transaction History:** Records each transaction with timestamps and provides a summary of recent transactions.

**Technology Stack:**

* **Programming Language:** C
* **Libraries and Tools:** Standard C libraries for input/output operations, file handling and time functions. Windows-specific libraries for system commands and console input.

**Target Audience:** The project is intended for students, educators and novice developers interested in learning the basics of banking systems, file handling and user authentication using C programming. It serves as an educational tool to understand fundamental programming concepts and the practical application of C language in software development.

**Potential Use Cases:**

* **Educational Purposes**: Demonstrates practical application of C programming concepts and reinforces understanding of file handling, data structures and basic algorithms.
* **Bank Simulation:** Provides a basic simulation of banking operations, useful for small-scale demonstrations or prototypes**.**
* **Learning Tool:** Acts as a stepping stone for learners aiming to develop more complex systems or applications in the financial sector. This project is a simplified representation of a real-world banking system, focusing on core functionalities and providing a foundation for further enhancements and complexity**.**

This project is a simplified representation of a real-world banking system, focusing on core functionalities and providing a foundation for further enhancements and complexity.

**Project Summary:**

The Bank Management System project is a software application developed in C programming language to facilitate basic banking operations for users. This system aims to streamline various banking activities, making them more efficient and accessible. The project encompasses features for user registration, account management and transaction processing. The primary functionalities include depositing money, withdrawing money, transferring money, checking account details and viewing transaction history. Below is a concise summary of the project:

**Purpose and Scope**

The main purpose of the project is to provide a simple yet functional banking solution that allows users to manage their accounts efficiently. The scope includes: User registration and authentication. Handling deposits and withdrawals. Facilitating money transfers between accounts. Maintaining and displaying account and transaction details.

**Key Features**

* User Registration and Logic
* Deposit Money
* Withdraw Money
* Transfer Money
* Account Details
* Transaction Details

**Goal of the Bank Management Project**

The primary goal of the Bank Management System project is to develop a functional and efficient software application that simulates basic banking operations. The specific objectives are as follows:

* **Automation of Banking Operations:** To automate common banking tasks such as deposits, withdrawals and money transfers, reducing the need for manual processing and increasing efficiency.
* **User Account Management:** To provide a secure and user-friendly platform for customers to manage their bank accounts, including account registration, authentication and accessing account details.
* **Transaction Tracking:** To enable users to keep track of their financial transactions, ensuring transparency and ease of account management.
* **Data Security:** To implement security measures that protect user information and ensure the integrity of financial transactions.
* **Educational Objective**: To enhance the developer's understanding of C programming by applying theoretical knowledge to a real-world project, covering aspects such as file handling, user input, data structures and basic algorithms

**Specific Goals Include:**

1. **User Registration and Authentication:**

* Allow users to create new bank accounts by providing necessary details.
* Implement a login system that validates user credentials to ensure secure access to their accounts.

1. **Deposit Functionality:**

* Enable users to deposit money into their accounts and update the account balance accordingly.
* Record and maintain a log of deposit transactions.

1. **Withdrawal Functionality:**

* Allow users to withdraw money from their accounts, ensuring sufficient balance is available.
* Update the account balance and log the withdrawal transactions.

1. **Money Transfer Capability:**

* Facilitate the transfer of funds between user accounts.
* Ensure secure and accurate transaction processing and logging.

1. **Account and Transaction Details:**

* Provide users with access to their account details, including personal information and current balance.
* Maintain and display a history of all transactions, providing transparency and ease of tracking financial activity.

1. **File Management:**

* Use file handling to store user data and transaction logs, ensuring data persistence and retrieval.

By achieving these goals, the project aims to deliver a robust and practical banking system that can serve as a prototype for more complex financial software, while also providing valuable learning experience in software development and C programming.

**Feasibility Analysis for Bank Management System**

The feasibility analysis for your Bank Management System project will evaluate the technical, economic and operational aspects to ensure the project's viability and sustainability.

1. **Technical Feasibility**

Technical feasibility assesses whether the technical resources and expertise are available to develop and maintain the system.

**Technical Requirements**:

* **Programming Language:** The project uses C programming, a widely known and efficient language for system-level programming.
* **Development Environment:** Standard C development environments (e.g., GCC, Visual Studio) are available and sufficient for developing the project.
* **Libraries and Tools:** Utilizes standard C libraries and additional tools like windows.h and conio.h for handling Windows-specific functionalities and console I/O operations.
* **Platform Compatibility:** The software is designed to run on Windows OS, which is commonly available.

**Assessment:**

* **Availability of Resources:** Adequate C programming knowledge and development tools are available. Any necessary resources (libraries, IDEs) are easily accessible.
* **Technical Challenges:** Potential challenges include handling file I/O operations securely, managing user data efficiently and ensuring data integrity. These can be addressed with proper coding practices and testing

1. **Economic Feasibility**

Economic feasibility evaluates the cost-effectiveness of the project, including development, implementation and maintenance costs.

**Cost Analysis:**

* **Development Costs:**

**1. Software:** Open-source C compilers and IDEs are available, reducing software costs.

**2. Hardware:** Standard hardware (personal computer) is sufficient for development.

**3. Labor:** The primary cost is the developer's time. As this is a capstone project, the labor cost is typically part of the course curriculum.

* **Implementation Costs:**

1. **Deployment:** Minimal deployment costs if distributed as a software package for Windows.
2. **Training:** Minimal to moderate training costs for end-users, depending on their familiarity with similar systems.

* **Maintenance Costs:**

1. **Bug Fixes and Updates:** Regular updates and bug fixes require ongoing developer time.
2. **Support:** Providing user support could incur additional costs if scaled beyond a small user base.

**Assessment:**

* The project is economically feasible as it utilizes free development tools and incurs minimal additional costs. The primary investment is time, which aligns with educational objectives.

**3. Operational Feasibility**

Operational feasibility assesses how well the system will function in the intended operational environment and its acceptance by users.

**Operational Requirements:**

* **User Interface:** Simple text-based interface for basic banking operations like deposit, withdrawal, transfer and checking account details.
* **User Acceptance:** Designed for basic banking management, it should be user-friendly and cater to users with basic computer literacy.
* **Performance:** Expected to perform efficiently on standard hardware with fast execution times for basic operations.

**Assessment:**

* **Ease of Use:** The command-line interface is straightforward but may require minimal training for non-technical users.
* **Reliability and Security:** The system’s reliability hinges on proper handling of file operations and user data security. Implementing robust error handling and data validation will enhance reliability.
* **Scalability:** Suitable for small-scale operations (individual use or small businesses). Scaling up would require a more robust architecture and possibly a graphical user interface (GUI).

**Requirement Analysis**

Requirement analysis is a crucial phase in software development, as it involves understanding and documenting what is needed from the software. For your Bank Management System project, the requirement analysis should cover both functional and non-functional requirements. Here's a detailed breakdown:

|  |  |
| --- | --- |
| **Functional Requirements** | |
| **User Registration** | **Register New Account:**   * User can create a new account by providing their name, account number, phone number and password. * The system should store user details securely. |
| **Login** | **Login to account:**   * Users can log in using their phone number and password. * The system should validate the credentials and grant access to the account if correct. |
| **Account Management** | **Deposit Money:**   * Users can deposit a specified amount of money into their account. * The system updates the account balance and records the transaction. |
| **Withdraw Money:**   * Users can withdraw a specified amount of money from their account. * The system checks for sufficient balance and updates the account balance accordingly. |
| **Withdraw Money:**   * Users can withdraw a specified amount of money from their account. * The system checks for sufficient balance and updates the account balance accordingly**.** |
| **Transfer Money:**   * Users can transfer a specified amount of money to another account. * The system verifies the recipient account and updates both account balances**.** |
| **Check Account Details:**   * Users can view their account details, including current balance and personal information**.** |
|  | **View Transaction Details:**   * Users can view the details of their recent transactions, including deposits, withdrawals and transfers. | |

|  |  |
| --- | --- |
| **Non-Functional Requirements** | |
| **Security** | * The system must ensure that user data is securely stored and transmitted. * Passwords should be encrypted. * The system should prevent unauthorized access. |
| **Usability** | * The user interface should be simple and intuitive. * Users should be able to perform banking operations without extensive training. |
| **Performance** | * The system should handle multiple users simultaneously without performance degradation. * Transactions should be processed quickly, ideally in real-time. |
| **Reliability** | * The system should be highly reliable with minimal downtime. * Proper error handling mechanisms should be in place to manage failures gracefully. |
| **Scalability** | * The system should be scalable to accommodate a growing number of users and transactions. |
| **Compatibility** | * The system should be compatible with various operating systems, especially Windows, as indicated by the use of <windows.h> |

|  |  |
| --- | --- |
| **Technical Requirements** | |
| **Programming Language** | The system is developed using C programming language. |
| **Development Environment** | The project should be developed in a suitable Integrated Development Environment (IDE) that supports C programming. |
| **File Management** | User data should be stored in files with appropriate file handling mechanisms for reading and writing data. |

**Data Flow Diagram**

1. External Entities:

* User (Customer)
* Bank Database (not explicitly shown in your code but implied)

1. **Processes:**

* Register Account
* Login
* Deposit Money
* Withdraw Money
* Transfer Money
* View Account Details
* View Transaction Details

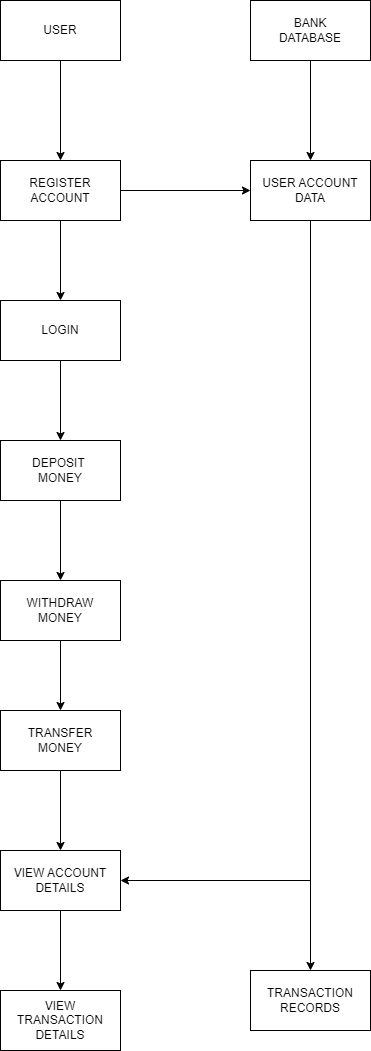
1. **Data Stores:**

* User Account Data
* Transaction Records

1. **Data Flows:**

* User inputs (e.g., registration details, login credentials)
* Processed data (e.g., account updates, transaction records)
* Outputs (e.g., confirmation messages, account balances)

**Level 1 DFD for Bank Management System**

****

**Explanation of the DFD:**

1. **User Interactions:**

* The User provides input to various processes like Register Account, Login, Deposit Money, Withdraw Money and Transfer Money.
* User can also request to View Account Details and View Transaction Details.

1. **Processes:**

* **Register Account:**

Takes user details and stores them in the User Account Data store.

* **Login:**

Validates user credentials against the data in the User Account Data store.

* **Deposit Money, Withdraw Money, Transfer Money:**

Update the User Account Data and create entries in the Transaction Records.

* **View Account Details:**

Retrieves user account information from the User Account Data store.

* **View Transaction Details:** Retrieves transaction details from the Transaction Records.
* **Data Stores:**

**User Account Data:** Stores user account details.

**Transaction Records:** Stores records of all transactions performed by users.

**Features of the Project**

1. **User Registration and Login:**

* **Register a new account:**

Users can create a new account by providing their name, account number, phone number and password.

* **Login to an existing account:**

Users can log in to their existing account using their phone number and password.

1. **Account Management:**

* **Deposit Money:**

Users can deposit a specified amount of money into their account.

* **Withdraw Money:**
* Users can withdraw a specified amount of money from their account, provided they have sufficient balance.
* **Transfer Money:**

Users can transfer a specified amount of money to another account, provided they have sufficient balance.

1. **Account Details:**

* **View Account Details:**

Users can view their account details including name, account number, phone number and current balance.

* **Transaction History:**

Users can view the history of transactions, including deposits, withdrawals and transfers.

1. **User-Friendly Interface:**

* **Menu Navigation:**

The system provides a clear menu-driven interface for users to navigate through different functionalities.

* **Input Validation:**

Basic input validation is implemented to ensure correct data entry.

**Short Description**

The Bank Management System is a console-based application developed in C programming language. It allows users to manage their bank accounts efficiently. The system supports user registration, login and various banking operations such as depositing money, withdrawing money, transferring money and viewing account details and transaction history. The application maintains a simple and user-friendly interface to facilitate easy navigation and operation.

**Pre-conditions**

1. **System Requirements:**

* **Operating System:**

Windows (as the program includes windows.h and conio.h headers).

* **Compiler:**

GCC or any other C compiler that supports Windows-specific headers.

* **Development Environment:**

A suitable IDE or text editor for writing and compiling C programs (e.g., Code::Blocks, Dev-C++).

1. **User Requirements:**

* **Registration:**

Users must register their account by providing their name, account number, phone number and creating a password.

* L**ogin Credentials:**

Users must remember their registered phone number and password to log in to their account.

1. **Initial Setup:**

* **File System Access:**

The application creates and reads user data files (e.g., phone.data), so it requires access to the file system for reading and writing these files.

**Testing Of the Features of Bank Management System**

1. **Testing Strategy**

The testing strategy involves verifying individual components (unit testing), interactions between integrated units (integration testing) and the complete system (system testing). Both manual and automated testing methods are utilized.

1. **Test Environment Hardware:**

Standard PC with a minimum of 4GB RAM and 500GB HDD. Software: Windows OS, GCC Compiler, Text Editor (e.g., Notepad++) and Command Prompt.

1. **Unit Testing**

Unit testing focuses on verifying that each individual module or function performs as expected.

**3.1 Deposit Money**

* Test Case 1: Valid Deposit

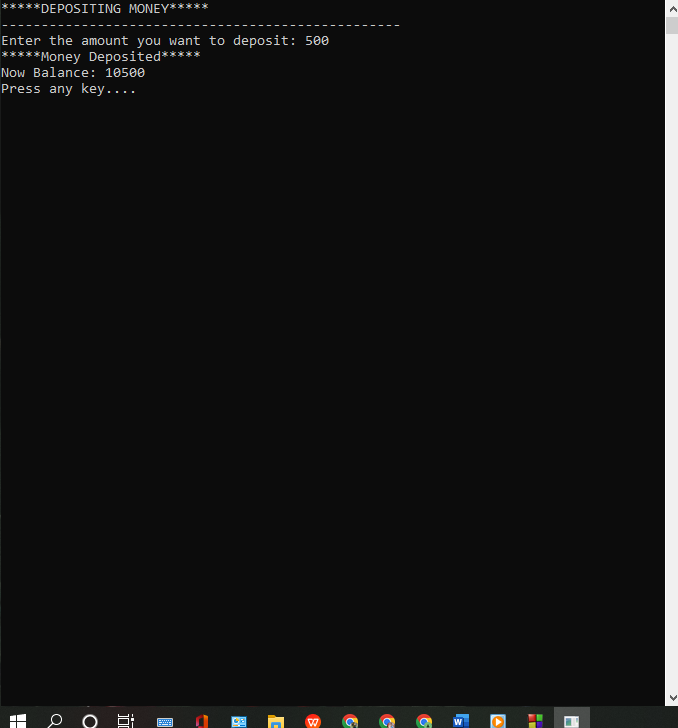
**Description:**

Deposit a valid amount into the account.

**Steps:**

1. Select "Deposit Money" from the menu.
2. Enter a valid amount (e.g., 500).

Expected Result: The amount should be added to the account balance and a success message should be displayed.



* **Test Case 2:** Invalid Deposit

**Description:**

Try to deposit an invalid amount (e.g., negative value).

**Steps:**

Select "Deposit Money" from the menu. Enter an invalid amount (e.g., -500).

**Expected Result:**

The system should display an error message indicating invalid input.

* 1. **Withdraw Money**
* **Test Case 1:**

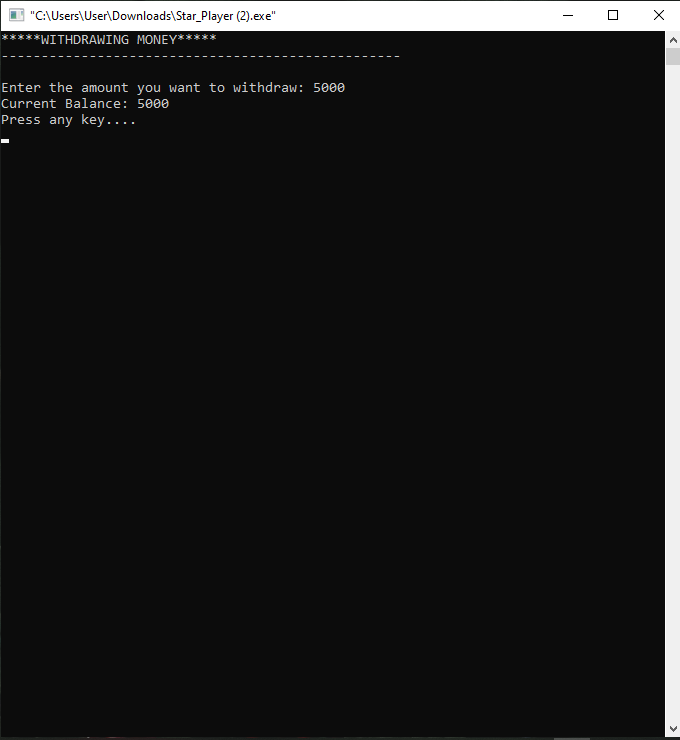
Valid Withdrawal

1. Description:

Withdraw a valid amount from the account.

1. **Steps:**
   1. Select "Withdraw Money" from the menu.
   2. Enter a valid amount that is less than or equal to the account balance (e.g., 500).
2. **Expected Result:**

The amount should be deducted from the account balance and a success message should be displayed.



* **Test Case 2:**

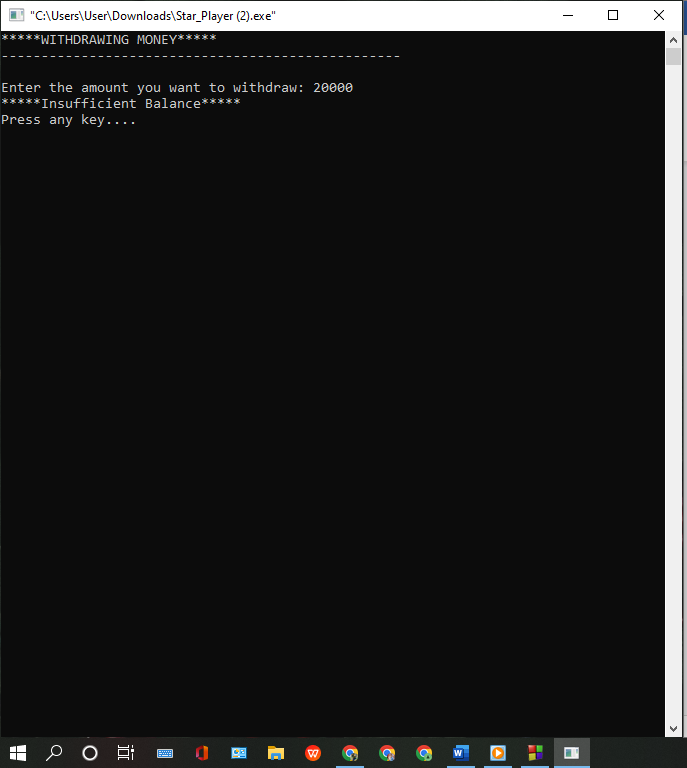
Insufficient Balance

1. **Description:**

Try to withdraw an amount greater than the account balance.

1. **Steps:**
   1. Select "Withdraw Money" from the menu. Enter an amount greater than the account balance (e.g., 15000).
2. **Expected Result:**

The system should display an error message indicating insufficient balance.



**3.3 Transfer Money**

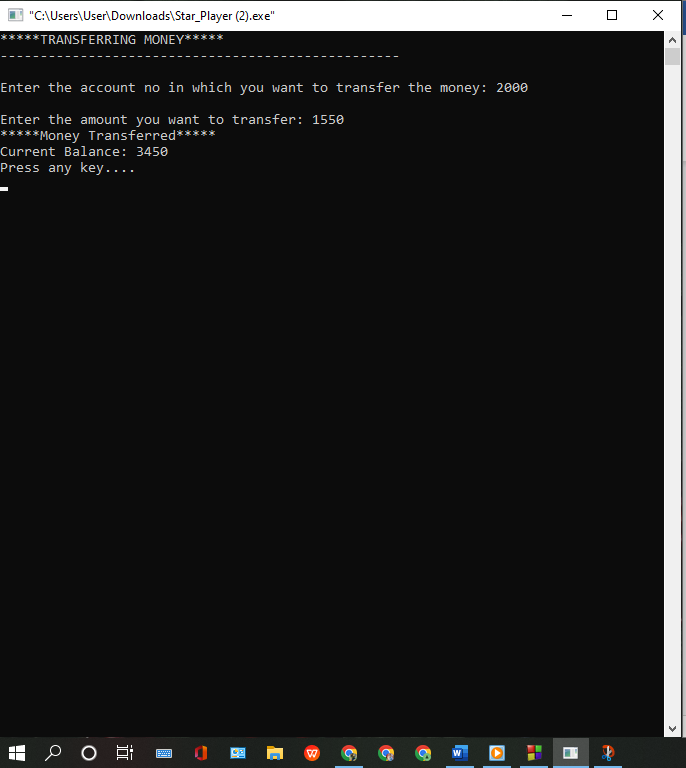
* **Test Case 1:** Valid Transfer

1. **Description:**

Transfer a valid amount to another account.

1. **Steps:** 
   1. Select "Transfer Money" from the menu.
   2. Enter a valid account number and amount.
2. **Expected Result:**

The amount should be deducted from the current account and a success message should be displayed.



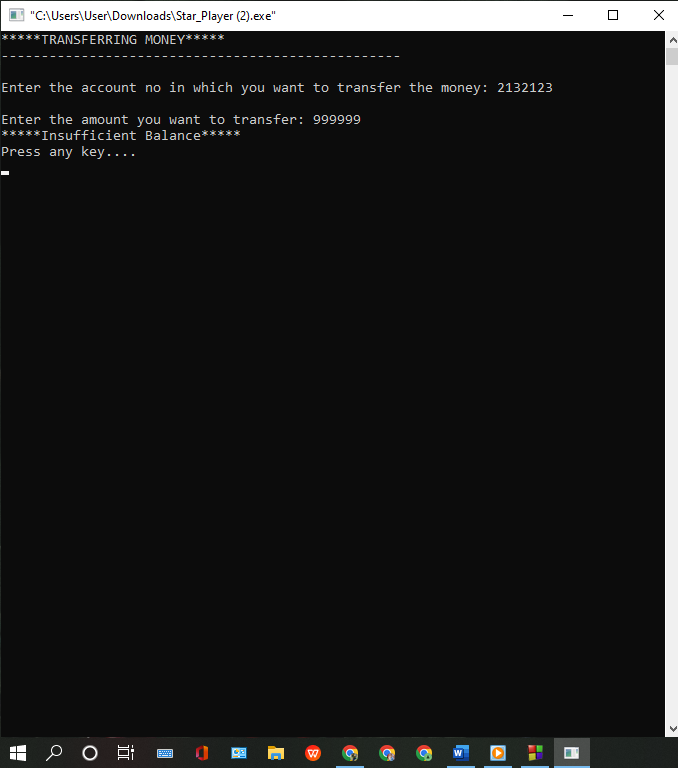
* **Test Case 2**: Insufficient Balance for Transfer

1. **Description:**

Try to transfer an amount greater than the account balance.

1. **Steps**:
   1. Select "Transfer Money" from the menu.
   2. Enter a valid account number and an amount greater than the account balance.
2. **Expected Result:**

The system should display an error message indicating insufficient balance.



**3.4 Check Account Details**

* **Test Case:** View Account Details

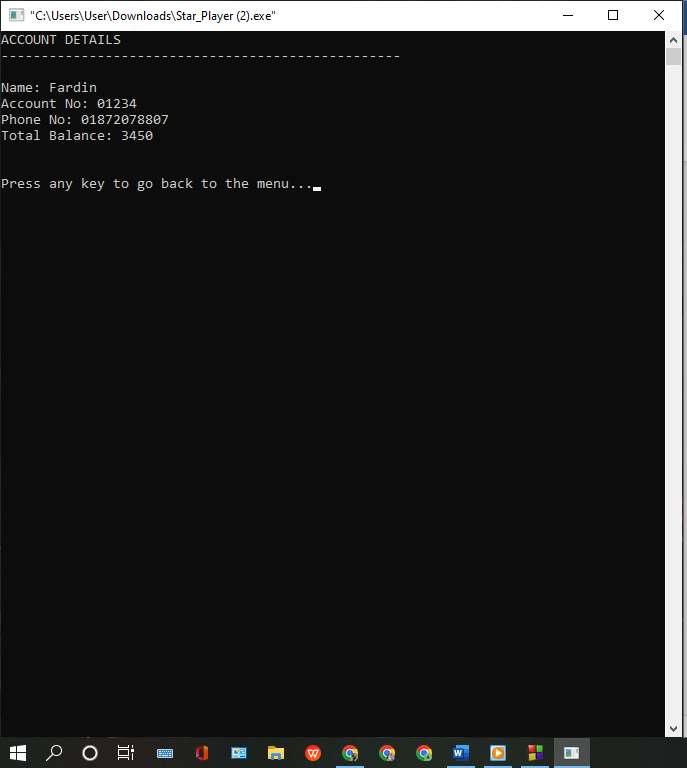
1. **Description:**

View the details of the account.

1. **Steps:**

Select "Account Details" from the menu.

1. **Expected Result:** The system should display the user's account details (name, account number, phone number and balance).



**3.5 Transaction Details**

* **Test Case:**

**View Transaction Details**

1. **Description:**

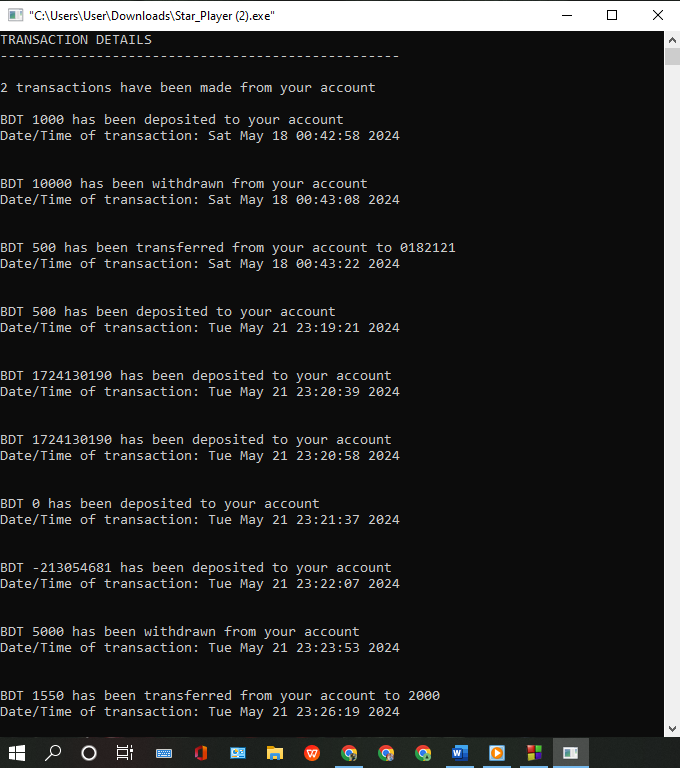
View the transaction history.

1. **Steps**:

Select "Transaction Details" from the menu.

1. **Expected Result:**

The system should display a list of all transactions performed by the user.



1. **Integration Testing**

Integration testing ensures that different modules interact correctly with each other.

* 1. **Deposit and Account Balance**
* **Test Case:**

Deposit and Check Balance

1. **Description:**

Deposit an amount and then check the updated account balance.

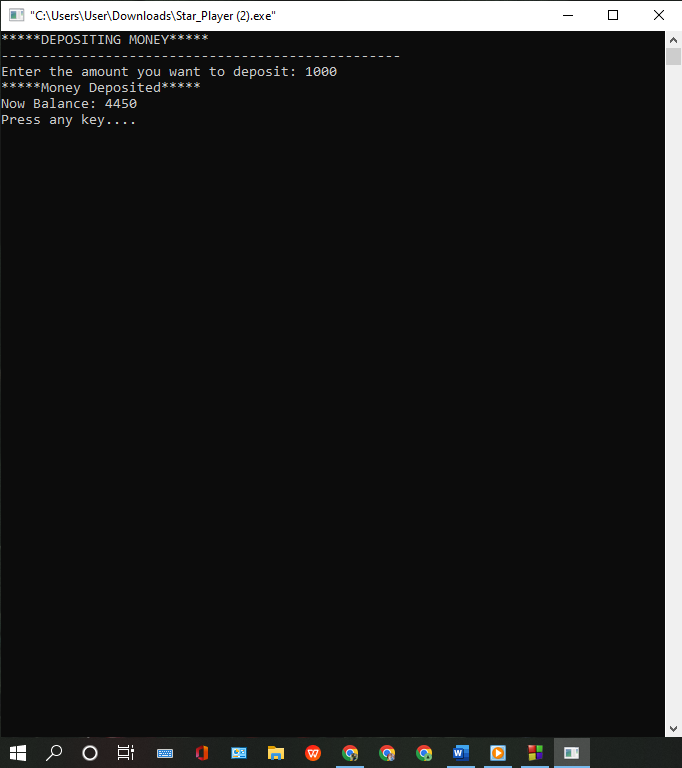
1. **Steps:**

Deposit a valid amount.

Check account details.

**Expected Result:**

The account balance should reflect the deposited amount.



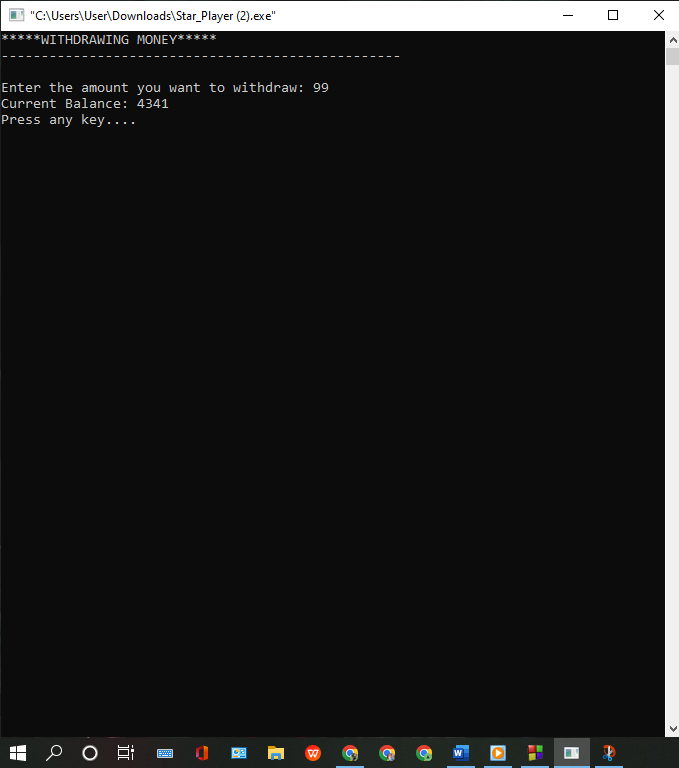
* 1. **Withdraw and Account Balance**
* Test Case: Withdraw and Check Balance

1. **Description:**

Withdraw an amount and then check the updated account balance.

1. **Steps:** 
   1. Withdraw a valid amount.
   2. Check account details.
2. **Expected Result:**

The account balance should reflect the deducted amount.

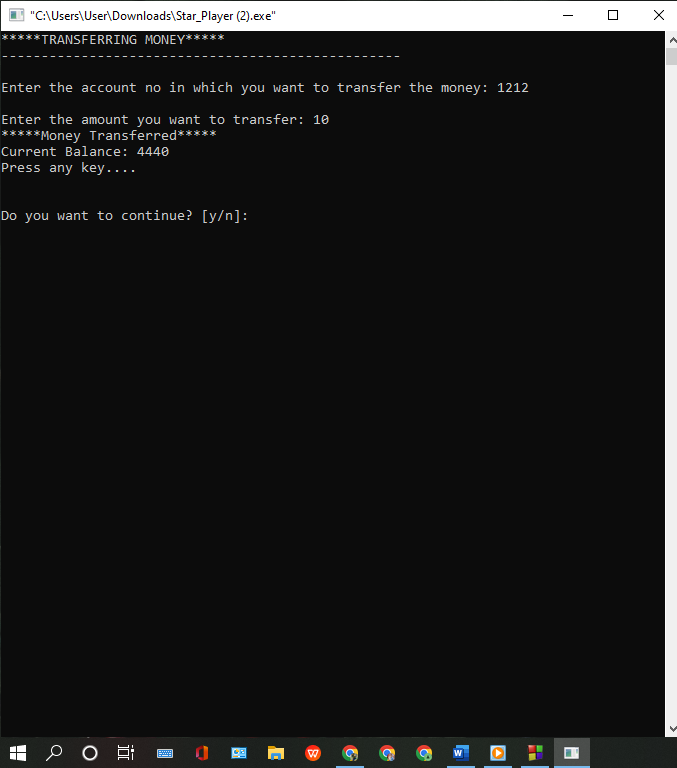


* 1. **Transfer and Account Balance**
* **Test Case:**

Transfer and Check Balance

1. **Description:**
2. Transfer an amount to another account and check the updated balance.
3. **Steps:** 
   1. Transfer a valid amount.
   2. Check account details.
4. **Expected Result:**

The account balance should reflect the transferred amount.



1. **System Testing**

System testing verifies the complete system's functionality in a production-like environment.

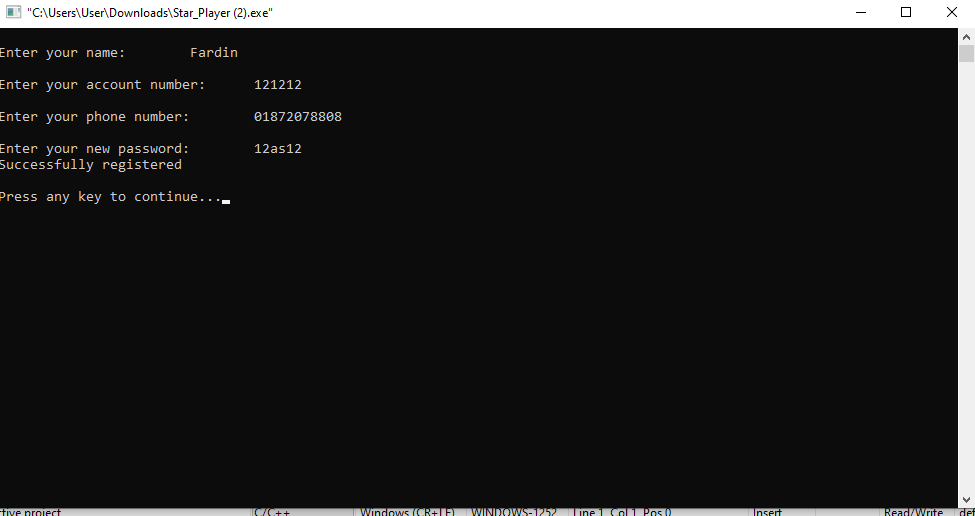
* 1. **User Registration and Login**
* **Test Case:**

Register and Login

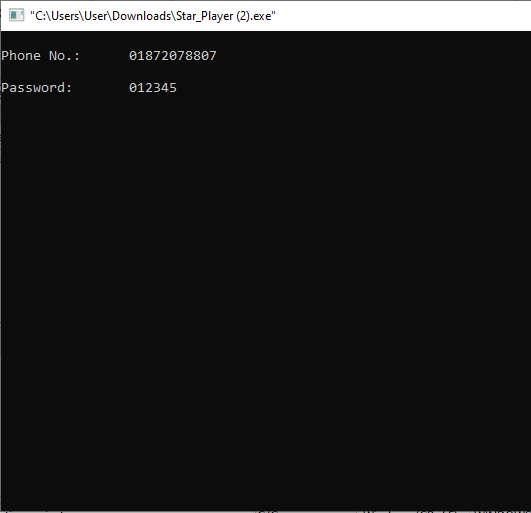
1. **Description:**

Register a new user and then log in with the new credentials.

1. **Steps:** 
   1. Register a new account.
   2. Log in with the new account details.
2. **Expected Result:** The user should be able to register and log in successfully.

****

**Logged in successfully**



* 1. **Complete Transaction Workflow**
* **Test Case**: Perform Full Transaction Workflow

1. Description: Perform a series of transactions and check final account details.
2. **Steps:** 
   1. Deposit an amount.
   2. Withdraw an amount.
   3. Transfer an amount.
   4. Check account details.
   5. View transaction details.
3. **Expected Result:**

All transactions should be performed successfully and account details should reflect all changes accurately.

1. **Bug Reporting and Tracking**

Any bugs found during testing are documented, reported and tracked until resolved. The bug report includes:

* **Bug ID:**

Unique identifier for the bug.

* **Description:**

Detailed description of the bug.

* **Severity:**

The impact level of the bug (Critical, Major, Minor).

* **Steps to Reproduce:**

Steps to recreate the bug.

* **Expected Result:**

What should have happened.

* **Actual Result:**

What actually happened.

* **Status:**

Current status of the bug (Open, In Progress, Resolved).

**Feature Checklist Of The Project**

|  |  |  |
| --- | --- | --- |
| **Bank Management System** | | |
| **Feature** | **Implemented** | **Not-Implemented** |
| **User Registration** | **Successful** |  |
| **User Login** | **Successful** |  |
| **Deposit Money** | **Successful** |  |
| **Transfer Money** | **Successful** |  |
| **View Account Details** | **Successful** |  |
| **Transaction History** | **Successful** |  |
| **User Profile Management** |  | **Failed** |
| **Customer Support Chat** |  | **Failed** |

**Contribution:**

Here is the contribution of each of our participants.

|  |  |  |
| --- | --- | --- |
| **Name** | **Id Number** | **Contribution** |
| **Md. Fardin Hasan** | **0242310005341393** | **33.33%** |
| **Maliha Farzana Muneyat** | **0242310005341062** | **33.33%** |
| **Erin Jahan Eshita** | **0242310005341074** | **33.33%** |

References

1. **Textbooks and Books**

- Kernighan, B. W., & Ritchie, D. M. (1988). \*The C Programming Language\* (2nd ed.). Prentice Hall.

1. **Online Tutorials and Articles**

- "Learn C Programming" (2023). \*TutorialsPoint\*. Retrieved from https://www.tutorialspoint.com/cprogramming/index.htm

1. **Official Documentation**

- Microsoft Documentation. (2023). \*<windows.h> header\*. Retrieved from <https://docs.microsoft.com/en-us/windows/win32/api/>

1. **Development Tools and Libraries**

- GCC (GNU Compiler Collection). Retrieved from https://gcc.gnu.org/

1. **Other Resources**

- "Introduction to Pointers in C" (2023). \*YouTube Video\*. Retrieved from

https://youtu.be/f2i0CnUOniA?si=t2tVULEgXzleQapX