VIETNAM NATIONAL UNIVERSITY – HO CHI MINH CITY
INTERNATIONAL UNIVERSITY
SCHOOL OF ELECTRICAL ENGINEERING

EE057IU
PROGRAMMING FOR ENGINEERS

VALKORIA BANK

SUBMITTED BY
NGUYEN THANH DANH - EEACIU24018

HO CHI MINH CITY, VIETNAM
11 01 2026

TABLE OF CONTENTS

LIST OF TABLES

Make sure to select headings in the sidebar to see a table of contents.

LIST OF FIGURES

CHAPTER I - INTRODUCTION

1. Problem Statement

- Traditional manual bookkeeping methods for managing bank accounts are time-consuming, error-prone, and difficult to maintain over long periods. As the number of customers and transactions increases, manual records become inefficient and unreliable. Therefore, there is a strong need for a simple digital system that can store account information, process transactions accurately, and ensure data persistence.

2. Goal

- The goal of this project is to develop a console-based banking transaction simulation program in C that allows users to manage bank accounts and perform basic financial operations such as deposits and withdrawals using file-based data storage.

3. Objectives

- The main objectives of this project are:
- To design a structured data model for bank accounts using struct.
- To implement basic banking operations including account creation, deposit, and withdrawal.
- To ensure all account data is stored and updated using a text file.
- To validate user input and handle common error cases safely.

CHAPTER II - TECHNICAL CONTENT

1. Design Specifications

- This project is designed as a modular, console-based banking transaction system written in C. The overall architecture separates responsibilities into multiple modules, improving readability, maintainability, and scalability. Each module is defined by a header file (.h) and implemented by a corresponding source file (.c).
- The system follows procedural programming principles and focuses on clear data flow between modules rather than object-oriented abstractions.

2. Overall Program Architecture

- The program is divided into five main components:
- Application Layer (App.c)
    - Controls the program execution flow, displays menus, and handles user interaction.
- Core Logic Module (ValkoriaCore.c, core.h)

        - Implements the main banking logic, such as deposits, withdrawals, and account validation.
    - Database Module (ValkoriaDB.c, db.h)
        - Manages file input/output operations and ensures account data persistence using a text file.
    - Utility Library (ValkoriaLib.c, lib.h)
        - Provides helper functions such as input handling, string processing, and common utilities used across modules.
    - Logging Module (ValkoriaLogger.c, logger.h)
        - Handles system logging for debugging and status reporting.

→ This modular approach ensures that changes in one component (e.g., file storage format) do not significantly affect other parts of the program.

3. Data Structures Design

- All account-related data is represented using structured data types defined in type.h. A bank account is modeled as a structure containing:
    - A unique account identifier
    - Customer name
    - Account balance
- Using a struct allows related data fields to be grouped logically, simplifying function interfaces and improving code clarity. This design also supports future extensions, such as adding transaction history or account status flags.

4. Core Transaction Logic

- The core banking operations are implemented in the Core module:
    - Deposit Operation
        - Adds a positive amount to the selected account balance after validating user input.
    - Withdrawal Operation
        - Deducts a specified amount from the balance only if sufficient funds are available. If the balance is insufficient, the operation is rejected, and an error message is displayed.
    - Input Validation
        - The program prevents invalid operations such as negative values, non-numeric input, or overdraft attempts.
- By isolating transaction logic in a dedicated module, the program maintains a clear separation between the user interface and business rules.

5. File-Based Database Management

- Persistent storage is handled by the Database module. Account data is stored in a plain text file to ensure simplicity and transparency.
- The database workflow is as follows:
    - At program startup, account data is loaded from the file into memory.
    - All transactions are performed on in-memory data structures.

    - When the program exits, the updated account data overwrites the existing file.
- This approach ensures data consistency while keeping file handling efficient and easy to debug.

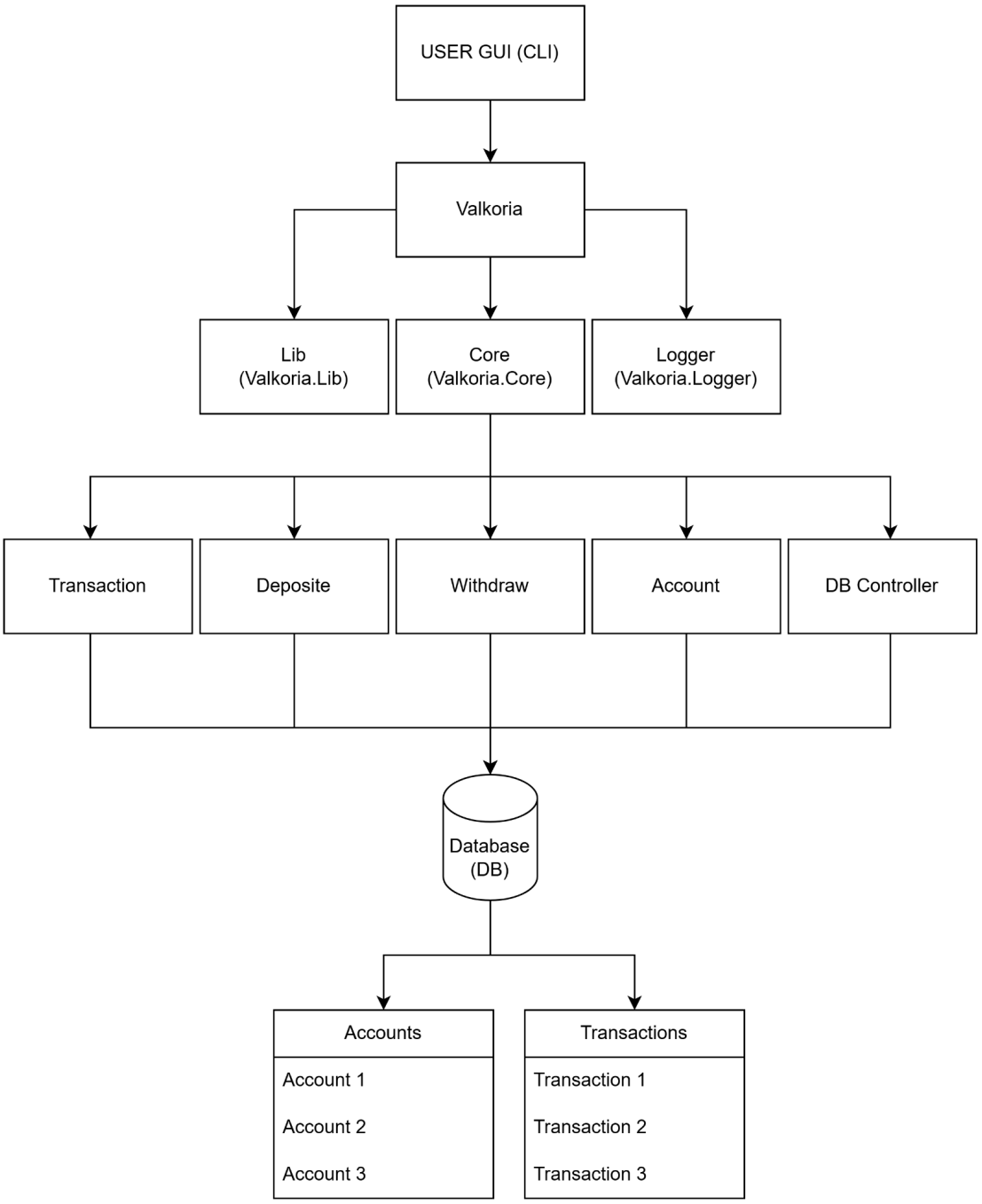
6. Logging Mechanism

- The logging module provides a centralized way to output system messages. Logging is used to:
    - Indicate successful operations
    - Report errors or invalid user actions
    - Assist with debugging during development
- By separating logging functionality into its own module, the program avoids redundant print statements and maintains cleaner core logic.

7. Program Execution Flow

- The program execution follows these steps:
    - Start the program and initialize the required modules.
    - Load account data from the database file.
    - Display the main menu and wait for user input.
    - Execute the selected banking operation.
    - Repeat until the user chooses to exit.
    - Save the updated data back to the file and terminate the program.

A flowchart can be used to visually illustrate this execution sequence and the interaction between modules.

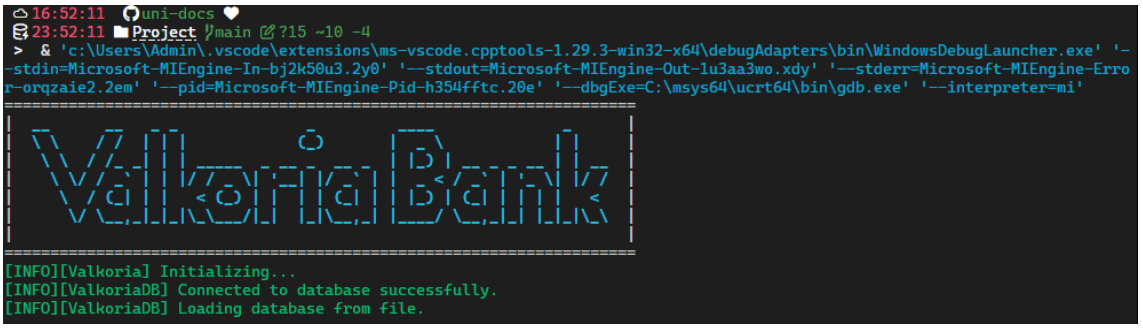


CHAPTER III - RESULTS

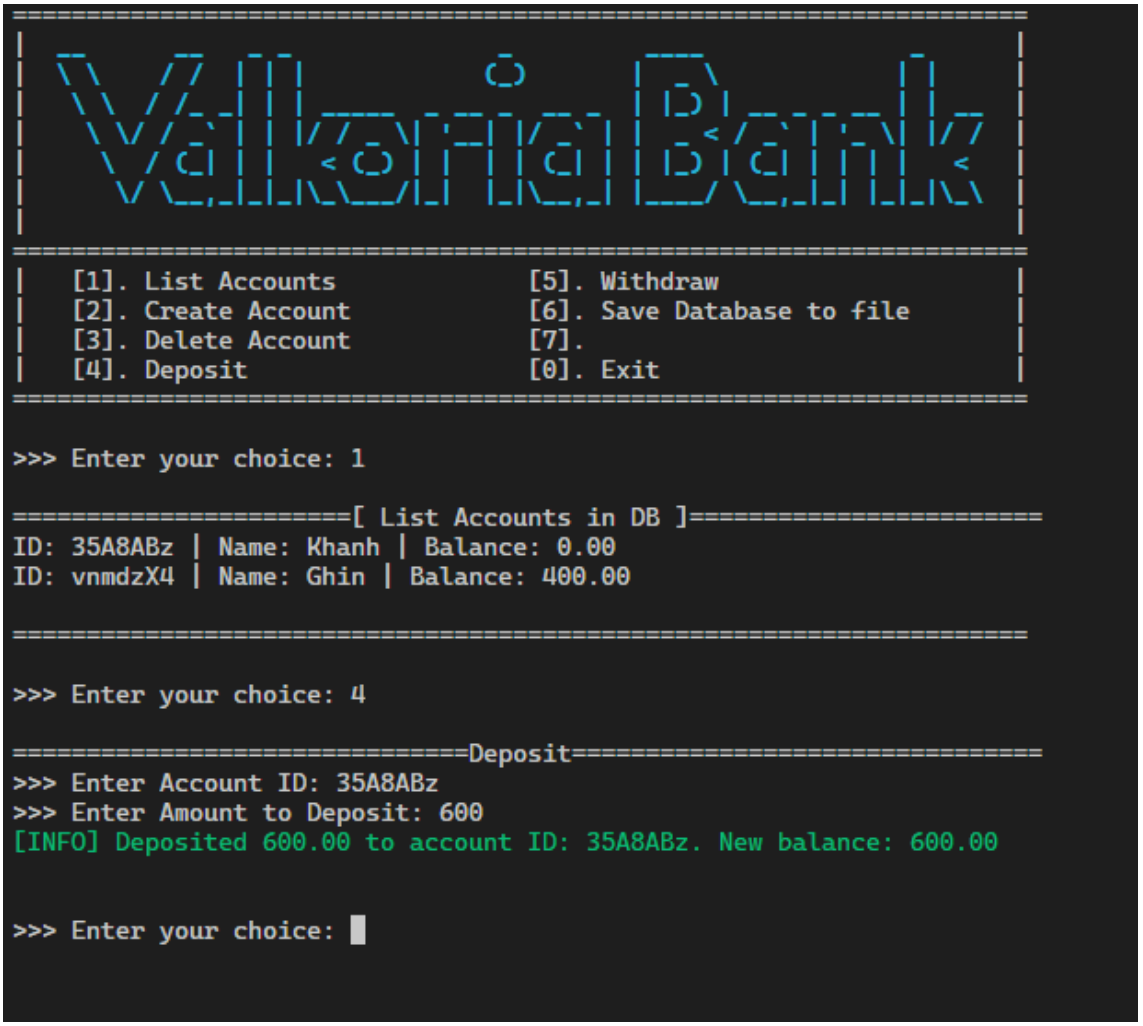
1. Successful Transaction Scenarios

- When valid input is provided, the system performs all banking operations correctly.
- ***Account Initialization***
    - At program startup, existing account data is successfully loaded from the text file into memory. All stored account numbers, customer names, and

balances are displayed correctly, confirming proper file reading functionality.

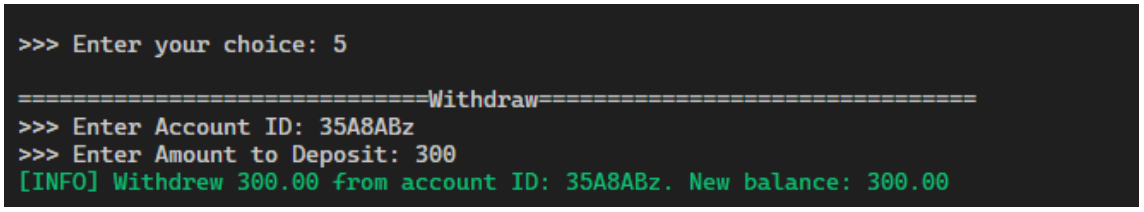


- Deposit Operation
    - When a user selects the deposit function and enters a positive amount, the system updates the account balance immediately. The new balance is displayed on the terminal, confirming that the deposit logic works as intended.



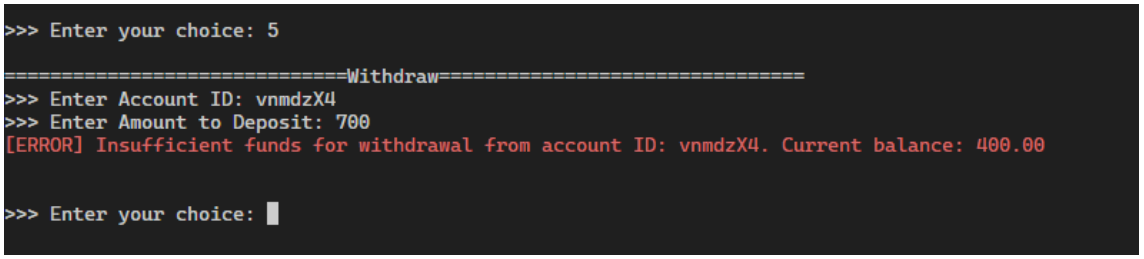
- Withdrawal Operation

- For withdrawals where the requested amount does not exceed the available balance, the transaction is processed successfully. The system deducts the amount and displays the updated balance, verifying the correctness of the core transaction logic.

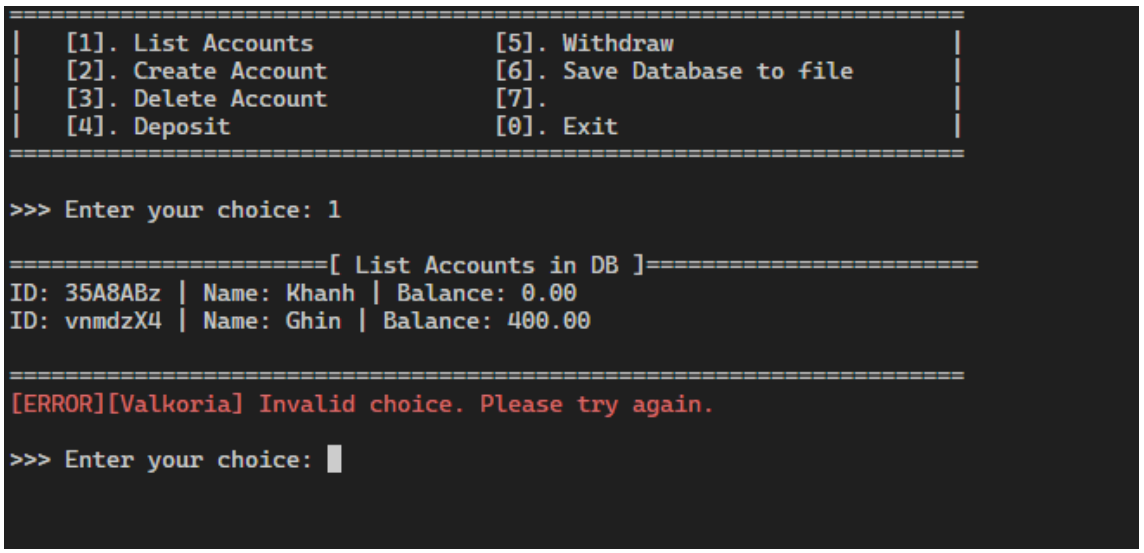


2. Error Handling and Validation Results

- The program effectively prevents invalid operations and handles error cases gracefully.
- Insufficient Balance Handling
  - If the user attempts to withdraw an amount greater than the current balance, the system rejects the transaction and displays an error message. The account balance remains unchanged, ensuring data integrity.

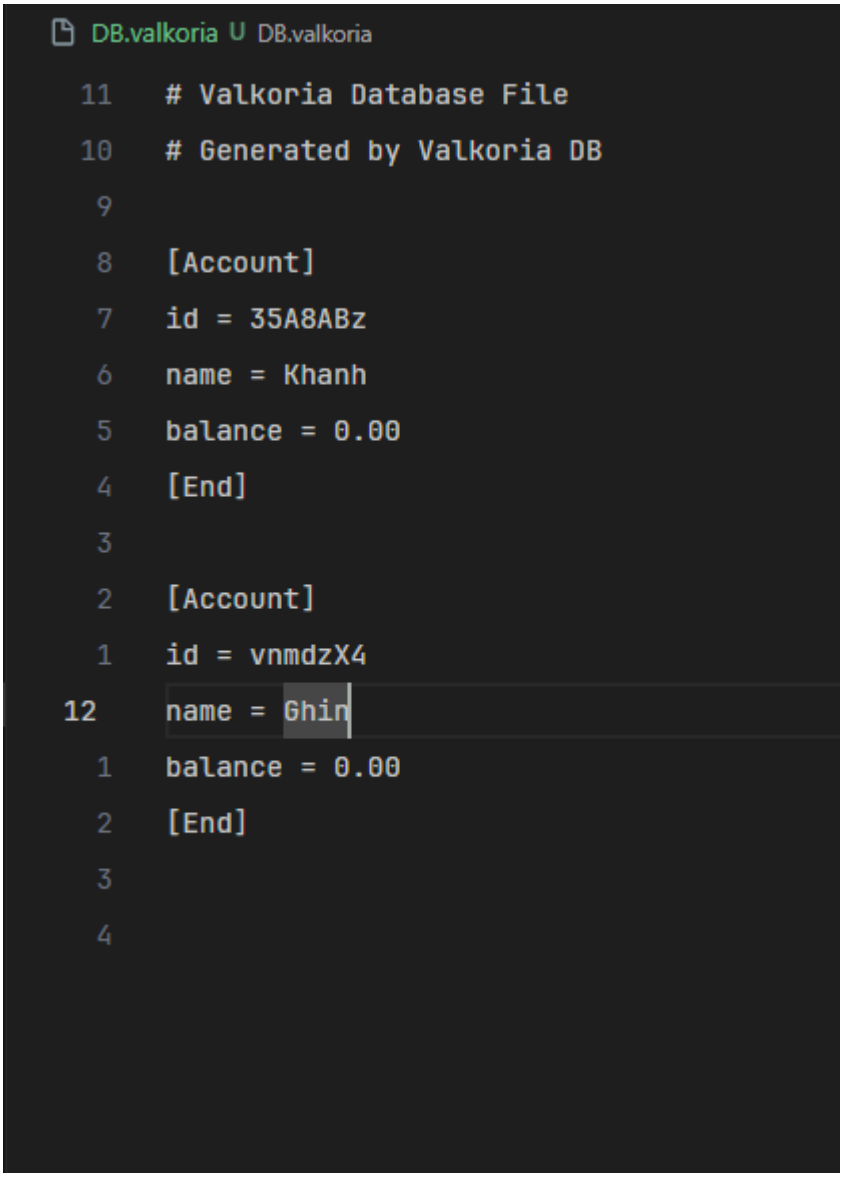


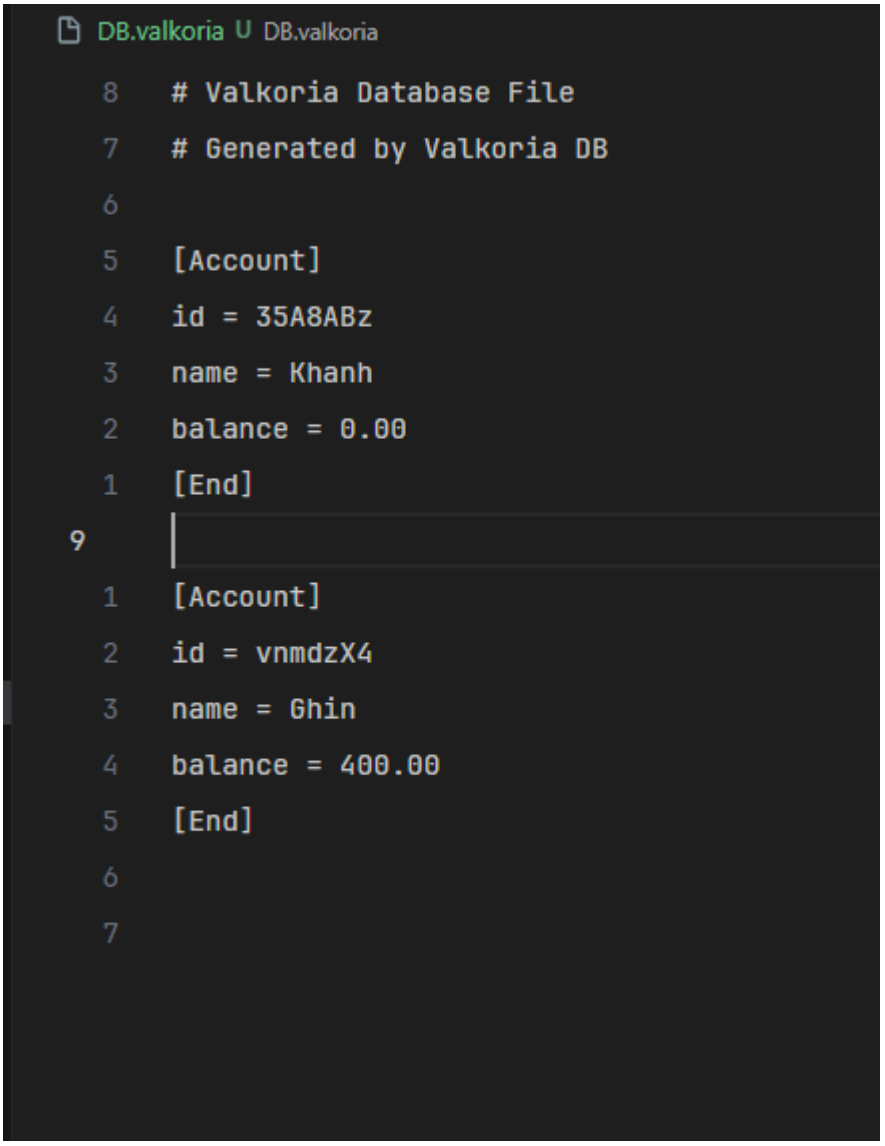
- Invalid Input Detection
  - The system correctly identifies invalid inputs such as negative values or non-numeric characters. In such cases, the user is prompted to re-enter valid input, preventing program crashes or undefined behavior.



3. File Data Verification

- To verify data persistence, the content of the account database file was examined before and after executing transactions.
    - Before execution, the file contained the original balances.
    - After deposits and withdrawals, the file was updated with the new balances upon program exit.
    - This confirms that file writing operations function correctly and that data is preserved across multiple executions of the program.





4. Discussion of Results

- The results confirm that the program meets all core project requirements. Transaction operations are accurate, input validation is effective, and file-based storage ensures persistent data management. The modular program structure also contributes to stable execution and ease of debugging.
- Overall, the system behaves reliably in both normal and error scenarios, demonstrating correct implementation of procedural programming and file handling techniques in C.

CHAPTER IV - CONCLUSION

This project successfully developed a console-based bank transaction simulation system using the C programming language. The primary objective of creating a simple yet

reliable banking application was achieved through the implementation of structured data management, core transaction logic, and file-based data persistence.

The program correctly handles essential banking operations such as account initialization, deposits, and withdrawals, while effectively preventing invalid transactions through input validation and error handling. The use of modular design, with separate components for core logic, database management, utilities, and logging, improved code organization and maintainability.

Although the system is limited to a text-based interface and basic functionality, it provides a solid foundation for future enhancements. Possible improvements include adding transaction history tracking, user authentication, and support for more advanced file formats or graphical user interfaces.

Overall, this project reinforces fundamental concepts of procedural programming, structured data design, and file handling in C, and demonstrates the practical application of these concepts in solving a real-world problem.

REFERENCES

[1] C. N. E. Anagnostopoulos, "License plate recognition: A brief tutorial," *IEEE Intelligent Transportation Systems Magazine,* vol. 6, no. 1, pp. 59–67, Jan. 2014.

[2] W. Jackson, *Learn Android App Development*, Apress, 2013.

[3] K. Aida-zade, E. Mustafaev, and J. Hasanov, "Intelligent reading system based on mobile platform," in *International Conference on Problems of Cybernetics and Informatics*, Baku, Azerbaijan, 2012, pp. 1-4.

[4] A. Mutholib, T. S. Gunawan, and M. Kartiwi, "Design and implementation of automatic number plate recognition on android platform," in *International Conference on Computer and Communication Engineering*, Kuala Lumpur, Malaysia, 2012, pp. 540–543.

[5] (A reference list of 20 or more sources is encouraged)

---

[1] *Footnote*