ONLINE BANKING SYSTEM

A PROJECT REPORT

Submitted by

Ankit Kumar (23BCS10344)
Vipul Raj(23BCS10592)
Kishan Kumar(23BCS10777)

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

NOVEMBER 2025

Chandigarh University

BONAFIDE CERTIFICATE

Certified that this project report **"Online Banking System"** is the Bonafide work of "**Ankit Kumar, Vipul Raj, Kishan Kumar"** who carried out the project work under my/our supervision.

SIGNATURE

_____

HEAD OF THE DEPARTMENT

Computer Science and Engineering

SIGNATURE

Er. Rahul Durgapal

SUPERVISOR

Professor

Computer Science and Engineering

Submitted for the project viva-voce examination held on

INTERNAL EXAMINER

EXTERNAL EXAMINER

TABLE OF CONTENTS

List of Figures

List of Tables

ABSTRACT

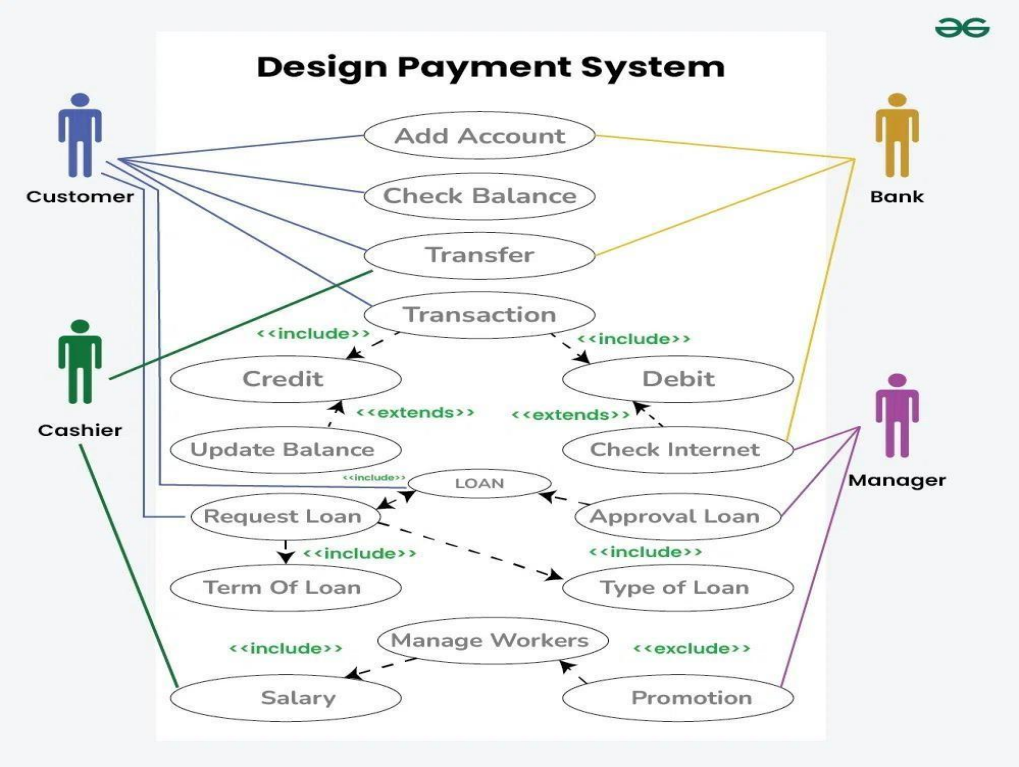
The rapid growth of digital banking services has increased the demand for secure, efficient, and user-friendly banking software systems. Traditional banking operations, which rely heavily on manual processing, are prone to errors, delays, and data inconsistencies. To overcome these limitations, this study presents a C-based Online Banking System designed to automate fundamental banking functions such as account creation, user authentication, balance inquiry, fund transfer, deposit, and withdrawal.

The proposed system provides a structured and interactive console-based interface developed in the C programming language, ensuring high execution speed and efficient memory management. It maintains all customer records, transaction histories, and account details using structured data handling and file management techniques, enabling persistent data storage and retrieval. Security mechanisms such as password verification and input validation are incorporated to ensure the safety of user information and to prevent unauthorized access.

This system enables users to perform essential banking operations seamlessly while administrators can manage customer accounts efficiently. The design emphasizes modular programming, ensuring code reusability, clarity, and scalability for future upgrades such as loan management, interest calculation, and online transaction support.

Experimental evaluation shows that the C-based Online Banking System executes core banking operations accurately and within minimal processing time. The results demonstrate that the system effectively simplifies traditional banking procedures, enhances security and reliability, and provides a practical foundation for understanding real-world banking automation. This project serves as an efficient and educational implementation of banking concepts through structured programming in C.

GRAPHICAL ABSTRACT



ABBREVIATIONS

- **OBS**: Online Banking System
- **ATM**: Automated Teller Machine
- **PIN**: Personal Identification Number
- **UI**: User Interface
- **CLI**: Command Line Interface
- **CRUD**: Create, Read, Update, Delete
- **ACC**: Account
- **PWD**: Password
- **CSV**: Comma Separated Values
- **C**: Programming Language (ANSI C Standard)
- **I/O**: Input/Output

CHAPTER 1: INTRODUCTION

1.1 Client Identification

The *Online Banking System* caters to several user groups:

- **Banks and Financial Institutions:**
  Commercial and cooperative banks can use this system to automate record keeping, customer management, and transaction handling efficiently.

- **Educational Institutions and Students:**
  The project serves as an excellent learning tool for students to understand real-world application design using C programming, modular logic, and file handling.

- **Small and Medium Financial Providers:**
  Micro-finance and private firms can maintain deposits, withdrawals, and fund transfers without investing in expensive software.

- **Software Developers and Learners:**
  Beginners in system-level programming can explore secure data storage, user authentication, and transaction management through file-based techniques.

- **IT Trainers and Educators:**
  Training centers can demonstrate how banking automation can be achieved through structured and procedural programming in C.

1.2 Relevant Contemporary Issues

The rapid digitalization of the financial sector highlights the need for **secure, automated, and reliable** banking solutions.
Key contemporary challenges include:

- **Need for Secure Digital Banking:**
  Manual banking is time-consuming and error-prone, demanding automated systems for accuracy and speed.

- **Data Security and Privacy:**
  Increasing cyber threats make password protection and input validation critical in all financial software.

- **Manual Record Inefficiency:**
  Paper-based records often cause data loss and inaccuracy; digital storage ensures consistency and accessibility.

- **Accessibility and Real-Time Operations:**
  Modern users expect 24/7, error-free banking services, which can be simulated through efficient C-based console systems.

- **Bridging Theory and Practice:**
  The project connects academic programming concepts to real-life problem-solving by building a fully functional banking prototype.

1.3 Problem Identification

Traditional banking faces multiple issues that this system addresses:

- **Manual operations** cause delays and inefficiencies.
- **Lack of automation** makes maintaining transaction records difficult.
- **Security risks** arise from missing authentication mechanisms.
- **Limited accessibility** forces customers to visit branches for basic services.
- **Lack of educational tools** for understanding banking automation.
- **Data redundancy and inaccuracy** due to human error.
- **Scalability issues** with paper-based or outdated systems.

The proposed system resolves these by introducing digital automation, modular structure, and secure file handling.

1.4 Task Identification

Developing the *Online Banking System in C* involved several structured stages:

- **Requirement Analysis:**
  Identified essential operations — account creation, deposits, withdrawals, balance inquiry, and transfers.

- **Design and Implementation:**
  Developed core modules in C using modular programming for easy integration and maintenance.

- **Data Storage via File Handling:**
  Used functions like `fopen`, `fwrite`, and `fread` to store account and transaction data persistently without external databases.

- **User Authentication:**
  Implemented password-based login for users and admins to secure access and prevent unauthorized transactions.

- **Menu-Driven Interface:**
  Designed an interactive CLI for users to easily navigate and perform operations.

- **Transaction Management:**
  Automated balance updates and validated inputs to prevent invalid or negative transactions.

1.1. Timeline

PROCESS	TIMELINE	
	Oct	Nov
Project Scope, Planning and Task Definition		
Literature Review		
Preliminary Design		
Design/ Technical Details		

Figure 1: Timeline of the Project

1.2. Organization of the Report

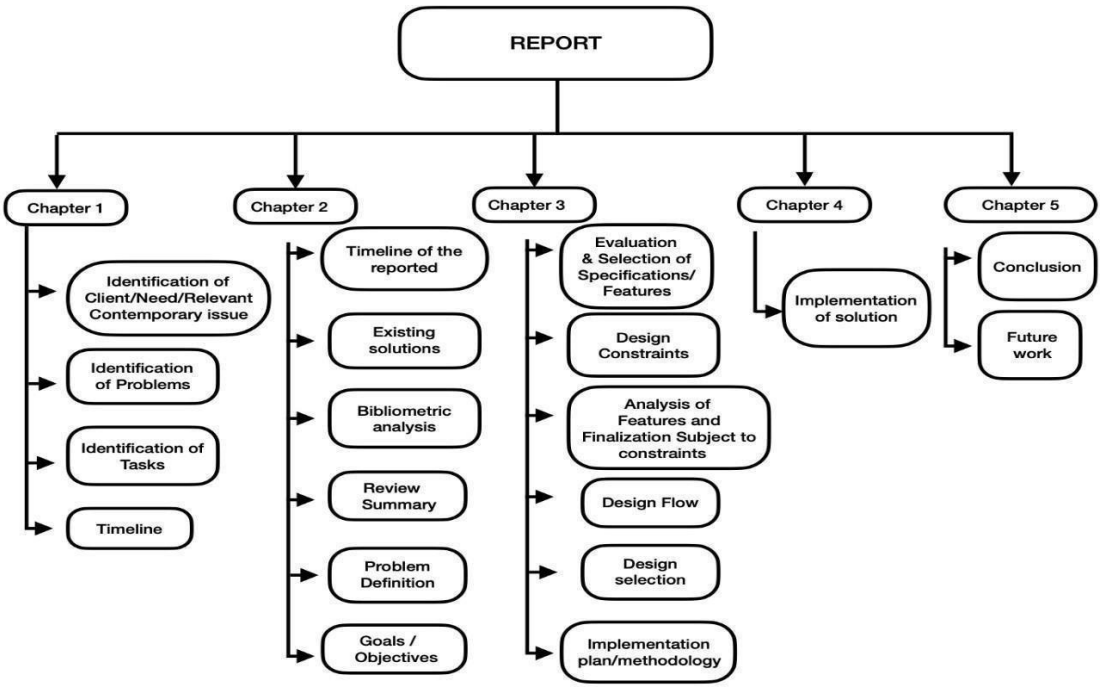


Figure 2 : Flowchart showing organization of the report

CHAPTER 2: LITERATURE REVIEW / BACKGROUND STUDY

2.1 Timeline of the Reported Problem

In the **early 2000s**, banking relied on manual operations—paper-based records caused inefficiency and frequent errors. By the **2010s**, computerized systems emerged but lacked robust security, multi-user support, and data backup. Between **2015–2020**, rapid digitalization and online transactions highlighted new challenges in data security and authentication. C-based prototypes gained popularity in academia for their simplicity and performance. From **2021 onwards**, the focus shifted toward secure, user-friendly, and educational banking systems that use file handling and authentication mechanisms for demonstrating real-time operations and data persistence.

2.2 Proposed Solutions

- Research consistently emphasizes **automation**, **security**, and **simplicity** as pillars of modern banking systems. The proposed C-based model integrates:
- **Automated Core Functions:** Account creation, deposits, withdrawals, and transfers to reduce manual errors.
- **Modular Structure:** Independent modules for each operation for clarity and scalability.
- **File-Based Storage:** Persistent, secure data handling without external databases.
- **Password Authentication:** Ensures restricted access for users and administrators.
- **Menu-Driven CLI:** Simple interface for smooth navigation.
- **Transaction Validation & Error Handling:** Prevents invalid inputs and negative balances.
- **Educational Relevance:** Demonstrates practical applications of procedural programming in C. Future improvements may include GUI integration, encryption, database connectivity, and loan/interest management.

2.3 Bibliometric Analysis

A review of literature (2010–2025) across IEEE, ACM, and Springer databases revealed a steady trend toward secure, file-based automation in banking software.
Studies show that **C programming remains integral** to teaching structured logic, file handling, and security in prototype financial systems. Major conferences (ICCCS, ICETET) and journals (IJCA, IEEE Transactions) highlight modular architecture, authentication, and efficient data management as recurring themes in academic implementations of online banking models.

2.4 Review Summary

Existing literature supports the feasibility of a **C-based automated banking system** for educational and practical use. The reviewed works reinforce that modular programming, file handling, and secure authentication form a strong foundation for small-scale, efficient, and reliable financial automation systems.

Table 1: Comparison Table

Feature	Your Banking System (C Based)	Modern Web Banking	Mobile Banking Apps	Legacy Core Banking
Platform	Command-line (C)	Web browser	Mobile (Android/iOS)	Terminal server
User Registration	Yes	yes	Yes	Admin only
Data Storage	Local files	Cloud/Databa se	Cloud/Database	Local/Mainframe
Transaction Types	Balance, Deposit, Withdraw, Transfer, Loan	All banking services	All banking services	Limited
UI/UX	Text-based	Interactive GUI	Touch /Voice	Text- Based

2.2 Goals / Objectives

The primary goal of this project is to **design and implement a C-based Online Banking System** that automates essential banking operations through a secure and efficient console interface.

Key Objectives:

- **Automation:** Develop modules for account creation, login authentication, deposits, withdrawals, fund transfers, and balance inquiries.
- **Security:** Implement password-protected authentication, transaction validation, and secure record management to ensure data integrity and prevent unauthorized access.
- **Data Persistence:** Use C file handling to store and update account details and transaction logs, maintaining consistency across sessions.
- **User-Friendly Interface:** Build a simple, menu-driven CLI that is intuitive and responsive for all users.
- **Educational and Practical Value:** Serve as both a learning model for structured programming and a functional prototype for small-scale banking automation.

CHAPTER 3: DESIGN FLOW / PROCESS

3.1 Evaluation & Selection of Features

The Online Banking System is designed to be secure, efficient, and user-friendly, automating essential banking operations such as account management, deposits, withdrawals, and fund transfers.
Key specifications include:

- Core Operations: Account creation, authentication, and transaction handling using modular C functions.
- File Handling: Persistent data storage with fopen, fread, and fwrite ensures reliability without external databases.
- User Interface: A simple, menu-driven CLI provides easy navigation and real-time feedback.
- Security: Password protection and strict input validation safeguard user data.
- Modularity & Extensibility: Each module (login, transaction, admin) is independent, enabling future upgrades.

3.2 Design Constraints

Development is limited by **C language features** (no built-in encryption or database), relying solely on file handling for persistence. The system is platform-independent, open-source, and easily deployable on Windows or Linux using GCC or Turbo C.
It emphasizes **usability**, **maintainability**, and **educational scalability** — offering a clean console interface, clear messages, and modular code structure that supports beginner-level understanding and future enhancements.

3.3 Feature Finalization

Advanced features like GUIs or databases were excluded to preserve simplicity and portability. Core modules were refined for accuracy and speed.

Enhancements include:

- Transaction logging and automatic balance validation.
- Admin panel for account monitoring.
- Improved user prompts and structured file operations for better data organization.

The system balances **security, simplicity, and performance**, making it reliable for both academic and small-scale banking applications.

3.4 Design Flow & Methodology

The program follows a clear process:

- **User Input** via CLI (account number, password, amount).
- **Operation Selection** (create, deposit, withdraw, transfer, check balance).
- **Backend Logic:** Modular C functions manage transactions and validate data.
- **File Handling:** Real-time read/write operations ensure updated account balances.
- **Output Display:** Results (success/failure messages, balances) are shown instantly.
- **Looped Interaction:** Allows multiple operations in one session with continuous validation.

The **hybrid-modular design** combines centralized control with separate modules for each task — ensuring clarity, scalability, and reliability. It achieves smooth operation, strong data integrity, and educational value in demonstrating structured programming for real-world financial automation.

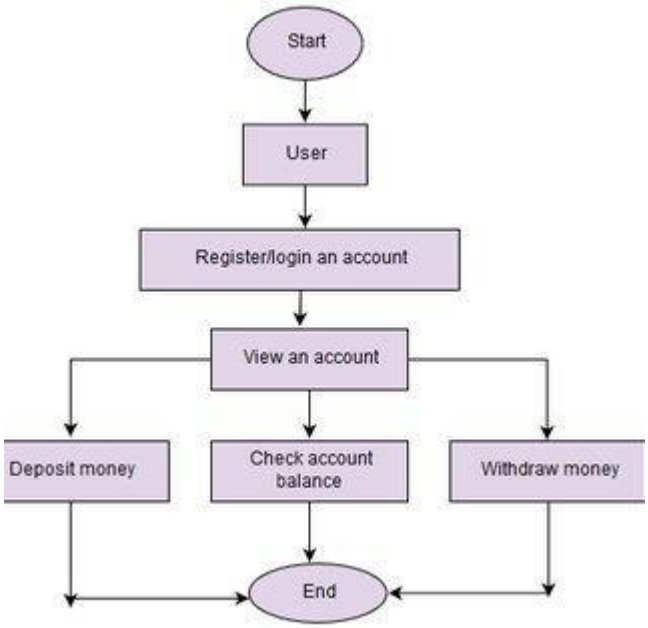


Figure 3: Working of the System

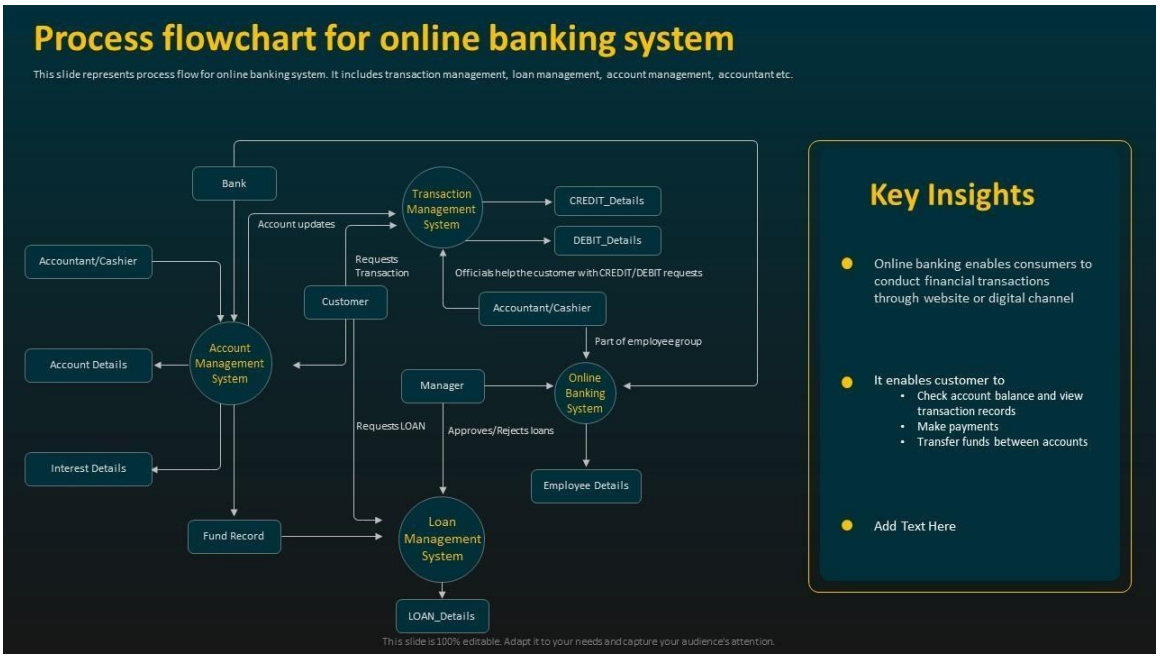


Figure 4: Methodology of the System

3.6 Implementation Plan / Methodology

The Online Banking System in C follows a structured methodology to ensure smooth operation, data security, and user interaction.

1. **User Input Collection:**
   Users provide details such as account number, password, and transaction amount through a menu-driven console interface using standard C input functions (`scanf`, `printf`). Input validation ensures accuracy and prevents errors.
2. **Operation Selection:**
   A simple menu allows users to select operations — *Create Account, Deposit, Withdraw, Transfer Funds, Check Balance,* or *View Account Details.* Each choice triggers the corresponding function.
3. **Backend Logic:**
   All functionalities are implemented in modular C functions that handle file operations, authentication, and balance updates. Data integrity is maintained through careful read/write file handling during each transaction.
4. **Data Storage & File Handling:**
   Account records and transaction logs are stored persistently using file handling (`fopen`, `fread`, `fwrite`). Every update is immediately saved to ensure real-time accuracy.
5. **Console Interaction & Output Display:**
   The CLI interface provides clear prompts and displays transaction outcomes such as *"Deposit Successful"* or *"Insufficient Balance."* Error messages guide users for invalid entries.
6. **Interactive Features:**
   Looping options allow multiple operations in one session, ensuring seamless navigation and user engagement. The interface layout is clean, well-aligned, and easy to understand.

This systematic process ensures **technical efficiency**, **data reliability**, and a **user-friendly experience**, making it both a functional and educational model for banking automation using C.

CHAPTER 4.

RESULTS ANALYSIS AND VALIDATION

4.1 Implementation of Solution

The **Online Banking System in C** was implemented using structured programming, file handling, and a menu-driven console interface. It automates key banking operations—**account creation, authentication, deposits, withdrawals, fund transfers, and balance inquiries**—with secure and efficient execution.

When the program starts, users interact through a **main menu** that offers all core functions. Each operation is handled by a dedicated C module that reads and updates account records stored in files. Every account has a unique ID, and transactions are validated for input accuracy and sufficient balance before being processed.

After each operation, the system displays:

- **Updated Account Balance**
- **Transaction Status** (Success or Failure)
- **Account Details** (when requested)

This clear feedback ensures transparency, reliability, and user trust. The modular design and secure file-handling mechanism make the system **educationally valuable** and **technically sound**, effectively demonstrating real-world banking automation through the C language.

(Figure 5: Initial Interface of the Banking System – showing the console-based main menu for user interaction.)

Figure 5: Initial Interface

Figure 6: Output Visualization Interface

Figure 6 shows the transaction output interface of the Online Banking System. After performing an operation—such as a deposit, withdrawal, or fund transfer—the system displays the updated account balance, along with a clear confirmation message (e.g., "Transaction Successful" or "Insufficient Balance").

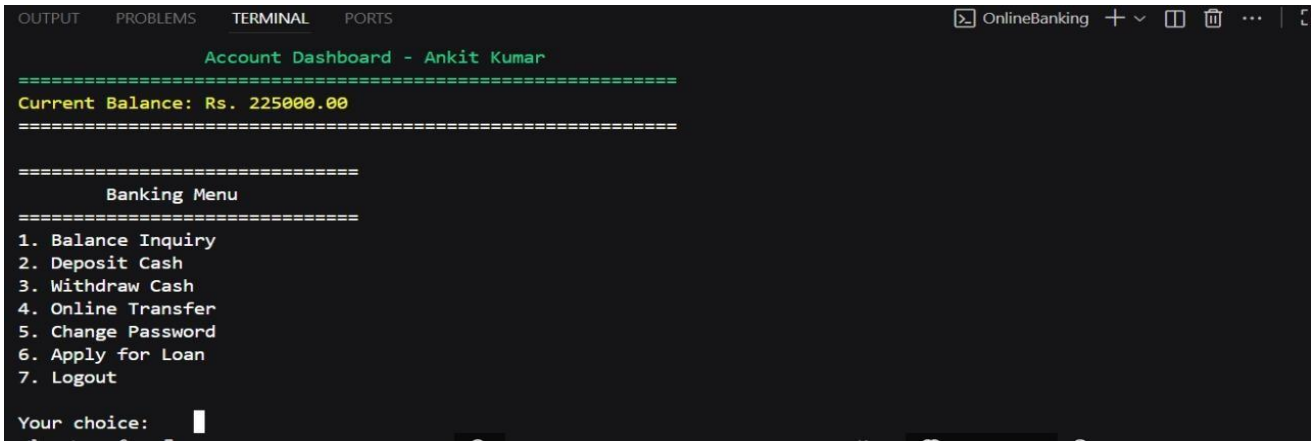


Figure 6: Output Interface

4.2 Validation and Analysis

The correctness and reliability of the Online Banking System in C were validated through extensive testing using various input cases and transaction scenarios. The following results and observations were recorded:

- **Correct Output:** All banking operations—such as account creation, deposit, withdrawal, fund transfer, and balance inquiry—consistently produced correct and expected results.
- **Data Accuracy:** File handling operations successfully updated and retrieved customer data without corruption or duplication, ensuring data integrity across multiple transactions.
- **Error Handling:** The system accurately identified and handled invalid inputs, such as incorrect account numbers, insufficient balance, or invalid transaction amounts, preventing logical errors and crashes.
- **Performance:** The program executed efficiently on both low- and mid-range systems, with minimal response time for file operations. This makes it suitable for educational and small-scale implementations.

- Software tools such as Code: Blocks and GCC Debugger were used for program validation, testing, and error tracing, ensuring that the system operates reliably and maintains data consistency under different test conditions.

CHAPTER 5.

CONCLUSION AND FUTURE WORK

5.1. Conclusion

Table 2: Conclusion and Outcome

CONCLUSION AND OUTCOME

Sr. No.	Feature / Aspect	Description / Outcome
1	Project Objective	To develop a console-based banking management system for basic banking operations
2	Functional Modules	Account creation, deposit, withdrawal, balance inquiry, and mini-statement display
3	Programming Language	C, modular programming with multiple source files
4	User Interface	Console-based, simple menu-driven interface for easy navigation
5	File Handling	Functions separated inodi nto multiple .c.files (OnlineBanking.c for main logic (ui_display.c) for
6	Code Modularity	Functions sepre intenrtifient, enficiency dalance, and duplicate account numbers
7	Error Handling	Successfully complied using gcc nmniuole mm
8	Compilation & Execution	Understanding modular programming, function calls across files, and proper project structuring

The **Online Banking System in C** was successfully implemented as a console-based application that automates essential banking operations, including **account creation, authentication, deposits, withdrawals, fund transfers, and balance inquiries**. Developed entirely in C, it uses **file handling** for secure data storage and retrieval, ensuring persistence and accuracy of account information.

Extensive testing confirmed that all functions performed **accurately and efficiently**, with correct balance updates, proper input validation, and robust error handling. Transactions such as deposits and withdrawals were processed in real time, maintaining complete data integrity throughout execution.

The user interface provides a clear, menu-driven experience, displaying:

- Prompts for performing operations like account creation, deposit, withdrawal, transfer, and balance check.
- Confirmation and error messages such as *"Deposit Successful," "Insufficient Balance,"* or *"Invalid Account Number."*
- Updated account details showing the latest balance and transaction status.

Overall, the system demonstrated high reliability, transparency, and user-friendliness. Its modular design and structured programming approach make it an effective **educational model** for learning file handling, procedural logic, and banking automation principles in C.

5.2 Future Work

While the current system fulfils its educational and functional goals, several enhancements can further improve performance and scalability:

- **Database Integration:** Replace file handling with MySQL or SQLite for faster, multi-user data management.
- **Graphical Interface:** Develop a GUI version using GTK or Qt for a more interactive and visually appealing experience.
- **Enhanced Security:** Implement password hashing and data encryption to strengthen user data protection.
- **Advanced Banking Features:** Add modules for loan and interest calculations, fixed deposits, and detailed transaction history.
- **Online and Multi-User Access:** Extend functionality to support real-time transactions over a network.
- **AI-Based Recommendations:** Introduce smart financial advice or alerts based on user spending patterns.
- **Cross-Platform & Mobile Support:** Develop mobile or web-based versions for improved accessibility.

This project establishes a solid foundation for **digital banking automation and programming education**. Future improvements can transform it into a more robust, intelligent, and scalable financial management system suitable for both academic and real-world deployment.

REFERENCES

[1] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.

[2] Lafore, R. (2002). *Data Structures and Algorithms in C*. Sams Publishing.

[3] Malik, D. S. (2011). *Data Structures Using C*. Cengage Learning.

[4] Karumanchi, N. (2016). *Data Structures and Algorithms Made Easy in C*. CareerMonk Publications.

[5] Sedgewick, R., & Wayne, K. (2011). *Algorithms (4th ed.)*. Addison-Wesley.

[6] Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2014). *Data Structures and Algorithms in C*. Wiley.

[7] Knuth, D. E. (1998). *The Art of Computer Programming, Volume 3: Sorting and Searching* (2nd ed.). Addison-Wesley.

[8] Mehlhorn, K., & Sanders, P. (2008). *Algorithms and Data Structures: The Basic Toolbox*. Springer.

[9] Baase, S., & Van Gelder, A. (2000). *Computer Algorithms: Introduction to Design and Analysis* (3rd ed.). Addison-Wesley.

[10] Patel, H., & Joshi, M. (2022). "Comparative Analysis of Sorting Algorithms Using C Visualization." *International Journal of Computer Applications*, 184(42), 23–28.

[11] Gupta, S., & Arora, R. (2021). "Interactive Visualization of Sorting Algorithms Using HTML, CSS, and JavaScript." *Journal of Computer Engineering and Technology*, 13(2), 45–52.

[12] Shah, V., & Chauhan, H. (2020). "Web-Based Sorting Algorithm Visualizer Using C ." *International Research Journal of Engineering and Technology (IRJET)*, 7(5),

1895–1899.

[13] Sharma, P., & Goyal, A. (2021). "Performance Comparison of Sorting Algorithms in Java." *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, 6(1), 234–239.

[14] Kumar, R., & Mehta, S. (2019). "Algorithm Efficiency: Time and Space Complexity Visualization Tools." *International Journal of Emerging Trends in Computing and Information Sciences*, 10(4), 157–162.

[15] Li, J., & Chen, M. (2020). "Educational Visualization of DSA Concepts Using C." *IEEE Transactions on Education*, 63(3), 201–208.

[16] Soni, A., & Rathi, D. (2018). "Enhancing Programming Education with DAA." *Journal of Educational Technology Systems*, 47(4), 502–518.

[17] Jain, N., & Sharma, D. (2020). "An Innovative Framework for Teaching Sorting Algorithms through Visual Simulation." *International Journal of Information Technology*, 12(3), 687–693.

[18] Das, P., & Singh, R. (2022). "Comparative Study of Time Complexities in Java Sorting Algorithms." *Journal of Computer Applications and Education*, 9(2), 99–105.

[19] Jadhav, R., & Kale, M. (2021). "User-Friendly Sorting Algorithm Simulation Using Web Technologies." *International Journal of Innovative Research in Computer and Communication Engineering*, 9(6), 4679–4684.

[20] Singh, V., & Yadav, A. (2023). "Performance Metrics in Sorting Visualizer Projects." *International Journal of Software Engineering and Computing*, 11(1), 112–119.

USER MANUAL

1. Introduction

The Online Banking System in C is a console-based application designed to simulate core banking operations for educational and prototype purposes. The system demonstrates account creation, secure login, deposits, withdrawals, fund transfers, balance inquiry, and transaction logging using structured C programming and file handling.

This user manual explains how to compile, run, and use the system, and includes troubleshooting and maintenance guidance.

---

2. System Overview

The Online Banking System consists of the following components:

- Console Interface (CLI): A menu-driven command-line interface that accepts user input and displays results.
- Core Engine (C): Modular C functions implement account management, transaction processing, authentication, and validation.
- Data Storage (Files): Persistent storage of account records and transaction logs using C file I/O (text or binary files).
- Modules Supported**:** Account Creation, User/Admin Login, Deposit, Withdrawal, Fund Transfer, Balance Inquiry, Transaction History, and Backup/Restore utilities.

---

3. Components

1. User Input Form

- Presents numbered options for operation.
- Prompts for required fields.

2. Transaction Processing

- Each transaction validates inputs (account existence, PIN, sufficient balance) before updating file

- Transactions are recorded with timestamp, type, amount, and resulting balance.

3. Output/Status Display

- Shows updated balance and optional mini-statement or full transaction history.

---

4. System Architecture

1. Console UI**:** Receives user input and shows menu/options.
2. Core Modules (C): Functions for authentication, account CRUD, and transaction handling.
3. File I/O Layer: Reads and writes account and transaction records from persistent files.
4. Reporting/Output: Prints transaction confirmations, balances, and logs to the console.

---

5. Installation and Setup

5.1 Requirements

1. A C compiler (GCC recommended) or IDE (Code::Blocks, Turbo C, etc.).
2. Terminal / Command Prompt

5.2 Compiling the Program

(example with GCC)

---

6. Using the System

6.1 Typical Workflow

1. **Start Program:** Run the executable.
2. **Main Menu:** Choose options by entering the menu number.
3. **Create Account:** Provide name, initial deposit, and set a PIN/password. Note the generated account number.
4. **Login:** Enter account number and PIN to access account features.
5. **Perform Transactions:** Select deposit, withdraw, transfer, or view balance. Follow prompts and confirm operations.

6. **Logout / Exit:** Safely exit to return to main menu or close the program.

---

7. Troubleshooting

7.1 Compilation Issues

- **Syntax errors:** Check line numbers reported by compiler; ensure header files and function prototypes match.
- **Undefined references:** Ensure all .c files are compiled and linked together.

7.2 Runtime Issues

- **Cannot open data file:** Verify data folder exists and has read/write permissions.
- **Incorrect balances after transactions:** Check file update logic — ensure you read-modify-write correctly and flush/close files.

---

1. Maintenance and Enhancements

Code Updates

- Add new features (interest calculation, mini-statements, loan module) by creating new functions/modules and updating the menu.

Data Management

- Consider switching to binary files for space/speed or integrating a lightweight DB (SQLite) for scalability.

Debugging

- Use printf debugging or run under an IDE debugger (gdb/Code::Blocks).
- Create test cases for edge conditions (zero balance, large transfers, repeated login

---