A First Year Project Proposal on

Simple Banking System

Submitted in Partial Fulfillment of the Requirements for the Degree of **BCA** under Pokhara University

Submitted by:

Nischit Jha, 242052

Prabesh Shakya, 242049

Rajin Maharjan, 242026

Rohit Khaling Rai, 242030

Date:

06/05/2025

Department of Bachelors of Computer Application

NEPAL COLLEGE OF

INFORMATION TECHNOLOGY

Balkumari, Lalitpur, Nepal

Proposal

Simple Banking System

1. Abstract:

In this project, we propose to develop a simple banking system using the C programming language. As second semester students, we aim to apply the fundamental concepts we have learned so far in a practical and meaningful way. Our system will be console-based and include essential banking operations such as account creation, balance inquiry, deposits, withdrawals, and account deletion. We will implement the project using basic features of C, including loops, conditionals, functions, structures, and file handling, to manage data and perform operations effectively.

The main goal of our project is to strengthen our understanding of logical thinking, procedural programming, and structured code development. We plan to use file handling to ensure that user data is stored and remains accessible even after the program is closed. Through this, we hope to learn how to maintain data persistence in a simple application. We also intend to include proper input validation and basic error handling to improve the user experience and make the system more reliable.

Working on this project will give us an opportunity to collaborate, practice problem-solving, and understand the basics of software development from design to testing. While the system will be limited in scope due to our beginner-level knowledge, it will lay a strong foundation for more advanced programming projects in the future.

By completing this project, we aim to not only fulfill our course requirements but also gain hands-on experience and build confidence in applying programming skills to real-world problems.

2. Problem Statement:

People often face difficulties in keeping track of their basic banking activities, especially when everything is done manually. Tasks like creating an account, checking a balance, or recording deposits and withdrawals can become confusing or lead to mistakes. To help with this, we decided to create a simple banking system using the C programming language.

This system will run in the console and allow users to perform basic banking operations easily, while also helping us practice and apply what we've learned in programming.

This project is not meant to replace real banking software but to serve as a learning tool for understanding how programs can solve real-life problems.

By creating this system, we aim to improve our skills in basic programming concepts such as using variables, functions, structures, and file handling. It also gives us experience in designing a simple project from start to finish, which is an important step in becoming better programmers.

Through this project, we also hope to understand the importance of organizing code properly and thinking logically about how each part of a program works together. Even though it is a simple system, building it step by step teaches us how to solve problems, test our code, and fix errors. This hands-on experience is very valuable as we continue learning more advanced topics in computer science and software development.

3. Project Objectives:

The main objective of our project is to design and implement a simple banking system using the C programming language. Through this project, we aim to apply the programming concepts we have learned in class and develop a functional application that simulates basic banking operations.

The specific objectives of this project are:

- To create a console-based application that performs basic banking tasks such as account creation, balance inquiry, deposit, withdrawal, and account deletion.
- To implement the system using fundamental C programming features, including structures, functions, conditionals, loops, and file handling.
- To practice modular programming by organizing code into logical functions for better readability and maintainability.
- To apply file input/output operations to ensure user data is stored and can be accessed across multiple sessions.
- To handle user input effectively and include basic error-checking mechanisms to avoid crashes or invalid operations.
- To improve our problem-solving and debugging skills through real-world project development.
- To build a foundation for more advanced programming projects in future semesters.

4. Significance of the study:

This project is important for our academic growth and hands-on learning experience. It allows us to connect the programming theory we've studied with a real-world inspired application, helping us understand the practical use of coding skills.

- Practical Learning:

Helps us apply basic C programming concepts like loops, conditionals, structures, and file handling in a real-world inspired project.

- Understanding Real Systems:

Provides insight into how simple financial systems manage data, transactions, and user interaction.

- Skill Development:

Enhances problem-solving, teamwork, and logical thinking, which are essential for future academic and professional growth.

- Foundation for Future Projects:

Builds a strong base for more advanced systems involving databases, user interfaces, or networking.

5. Scope and Limitations:

5.1 Scope:

- The system will be developed using the C programming language. It will provide basic banking functionalities such as:
- Creating a new account
- Viewing account details
- Depositing and withdrawing money
- Checking balance

- The system will use file handling for storing and retrieving customer data.
- The program will run on a command-line interface (CLI) for simplicity.
- The focus will be on demonstrating core programming concepts: functions, structures, loops, and file I/O in C.

5.2 Limitations:

- The system will not include advanced banking features such as online transactions, loan processing, or credit scoring.
- There will be no graphical user interface (GUI); it will run in a terminal window.
- Security features such as password encryption or user authentication will be minimal or basic
- It will not support concurrent user access (no multi-user functionality).
- Data will be stored in flat files rather than a database, which limits scalability and data integrity.

6. Proposed Methodology/ Technical description of the Project:

6.1 Requirement Analysis:

- Identify core banking operations to be implemented: account creation, deposit, withdrawal, balance inquiry, and data storage.
- Define input/output structure for each operation.
- Decide on file structure for storing account information persistently.

6.2 System Design:

- Use structured programming principles to design the solution.
- Define necessary data structures using struct in C to represent account information.
- Design flowcharts or pseudocode for each module to clarify logic flow.

6.3 Module Development:

- Account Management Module: Create new accounts and assign unique account numbers.
- Transaction Module: Handle deposits and withdrawals with proper validation.
- Inquire Module: Allow users to view account details and balance.
- File Handling Module: Read/write/update customer data using standard file I/O operations in C.

6.4 Implementation:

- Write modular C code for each function using best practices.
- Ensure error handling (e.g., invalid inputs, file not found, insufficient balance).
- Keep code comments and documentation for maintainability.

6.5 Testing and Debugging:

- Perform unit testing for individual functions.
- Conduct integration testing to ensure proper flow between modules.
- Simulate test cases for various banking operations to verify correctness.

6.6 Final Integration:

- Combine all modules into a single application.
- Create a simple text-based menu for user interaction.
- Finalize documentation and ensure the program runs smoothly.

7. Proposed Performance Analysis Methodology and Validation Scheme:

7.1 Performance Metrics:

- Execution Time: Measure the time taken to perform basic operations like deposit, withdrawal, and balance check.
- Accuracy: Verify that all transactions are recorded correctly and balances are updated precisely.
- File Integrity: Ensure that the data stored in files is accurate and remains consistent after each operation.

7.2 Validation Approach:

- Functional Testing: Test each function/module (account creation, deposit, etc.) independently to ensure they meet the required behavior.

- Boundary Testing: Check the system's behavior when minimum and maximum values (e.g., zero balance, maximum withdrawal) are used.

- Error Handling Verification: Intentionally provide invalid input (e.g., wrong account number, negative deposit) to ensure the system responds appropriately.

7.3 User Simulation:

- Simulate different types of users (e.g., new users, repeat users) to verify how the system handles typical banking actions.
- Validate the user menu by navigating through all options and checking for logic errors or crashes.

7.4 Tools Used:

- GCC Compiler: To compile and run the C program.
- Embarcadero Dev C++ or Visual Studio Code: For writing and testing the C code.

8. Proposed Deliverables/Output:

8.1 Executable C Program:

- A fully functional console-based banking system application written in C.
- Capable of performing basic operations such as:
  - Account creation
  - Deposit and withdrawal transactions
  - Balance inquiry
  - Account detail viewing

8.2 Source Code Files:

- Well-structured and commented .c source code files.
- Organized into modules/functions for better readability and maintenance.

9. Project Task and Time Schedule:

Week	Task	Description
1	Project Planning & Requirement Analysis	Finalize topic, define scope, gather requirements, and plan basic features.
2	System Design	Design flowcharts, define struct for account data, plan file handling.
3	Module Development	Code functions for account creation, deposit, withdrawal, and balance check.
4	File Handling	Implement file reading/writing for storing and retrieving account information.
5	Testing & Debugging	Test all functions, handle edge cases, and improve code structure.
6	Finalization & Documentation	Prepare final report, finalize code, and submit all deliverables.

10. References:

- OpenAI. (2025). ChatGPT – AI Language Model. Used for drafting proposal content and refining project documentation. Retrieved from https://chat.openai.com/

- GitHub. (n.d.). Simple Banking System Projects in C. Retrieved April 30, 2025, from https://github.com/umairinayat/Banking-management-system

- Studocu. (n.d.). Sample C Programming Projects and Reports. Retrieved April 30, 2025, from https://www.studocu.com/row/document/mount-kenya-university/introduction-to-information-systems/banking-system-project-proposal/65987138