Data Structure Lab Report

on

Blockchain Simulation using C++ Linked List

Submitted By

Nikhil Dinkar Pawar	**SY-Comp-A(68)**
Bhumi Anil Jha	**SY-Comp-A(70)**
Swayam Vijay Dusing	**SY-Comp-A(74)**
Jay Nitin Shinde	**SY-Comp-A(75)**

Under the Guidance of

Prof. Seema Gondhalekar

S. Y. Computer Engineering (2024-2025)

Department of Computer Engineering

K. K. Wagh Institute of Engineering Education & Research

Hirabai Haridas Vidyanagari, Amrutdham, Panchavati,

Nashik – 422 003

1. Abstract

This project implements a basic blockchain system in C++ with user management, transaction processing, and simple hashing for integrity verification. Using linked lists, users and blocks are dynamically maintained, simulating a real-world blockchain environment. It focuses on understanding how blockchain stores transactions securely and how data structures like linked lists can be used to manage such dynamic systems.

2. Introduction

Blockchain is a revolutionary technology that ensures secure, tamper-proof transaction records. In this project, we simulate a blockchain using C++ where users can create accounts, perform transactions, and verify the chain's integrity.
The project applies key concepts of data structures, such as linked lists, dynamic memory allocation, and hashing, making it an educational tool for understanding both blockchain principles and fundamental programming skills.

3. Problem Statement

The objective is to develop a simple blockchain prototype that allows user creation, balance management, secure transactions, and tamper detection using hashing and linked list structures.

a. Objective

- To implement a basic blockchain simulation with C++.
- To apply linked lists for dynamic storage of users and blocks.
- To validate transactions using hashing and signature verification.

b. Outcome

- A working blockchain that securely records user transactions.
- Hands-on understanding of how blockchains prevent data tampering.
- Better grasp of data structure operations like insertion, traversal, and validation.

4. Scope

This project focuses on educational simulation of blockchain. It covers basic functionalities like user management, transaction processing, blockchain creation, verification, and detection of unauthorized modifications. Advanced features like mining, cryptographic signatures, and consensus algorithms are not included.

5. Requirement Analysis

5.1 Functional Requirement

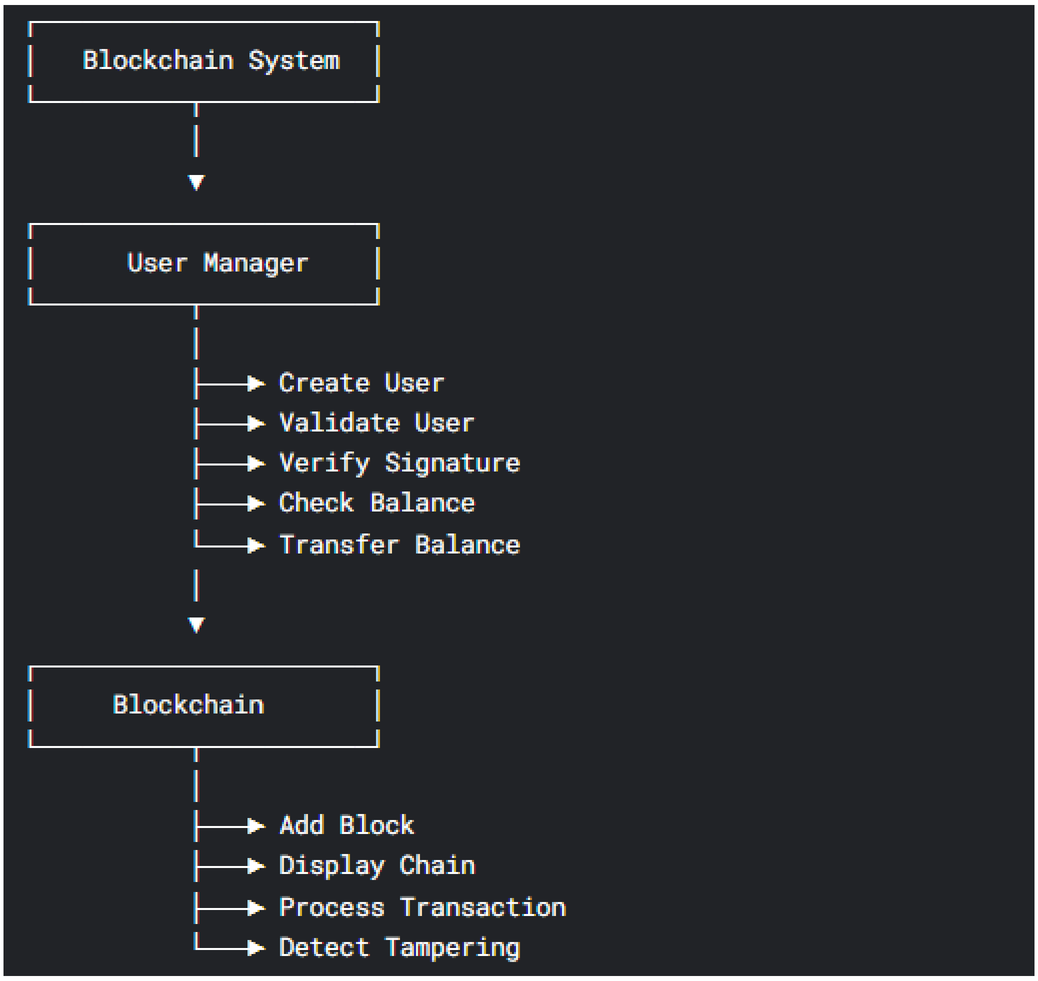
1. Create users with unique public and private keys.
2. Allow users to perform transactions.
3. Add blocks for every valid transaction.
4. Display all users and the entire blockchain.
5. Detect any tampering in the blockchain.

5.2 Non-Functional Requirement

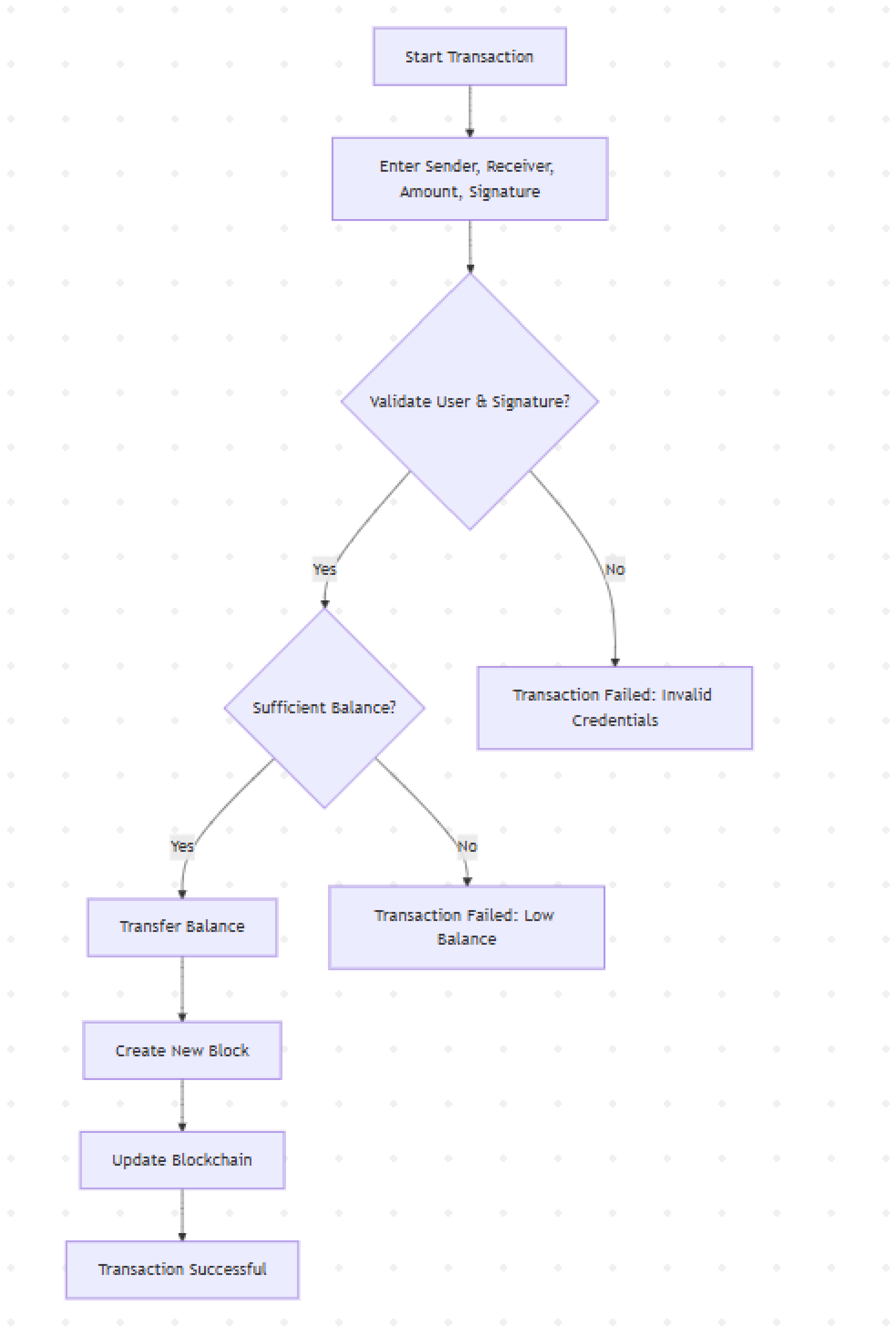
1. The system must provide a simple command-line interface.
2. The system should validate input properly and handle errors.
3. The blockchain should operate with basic integrity and fast response time.

6. UML Diagram

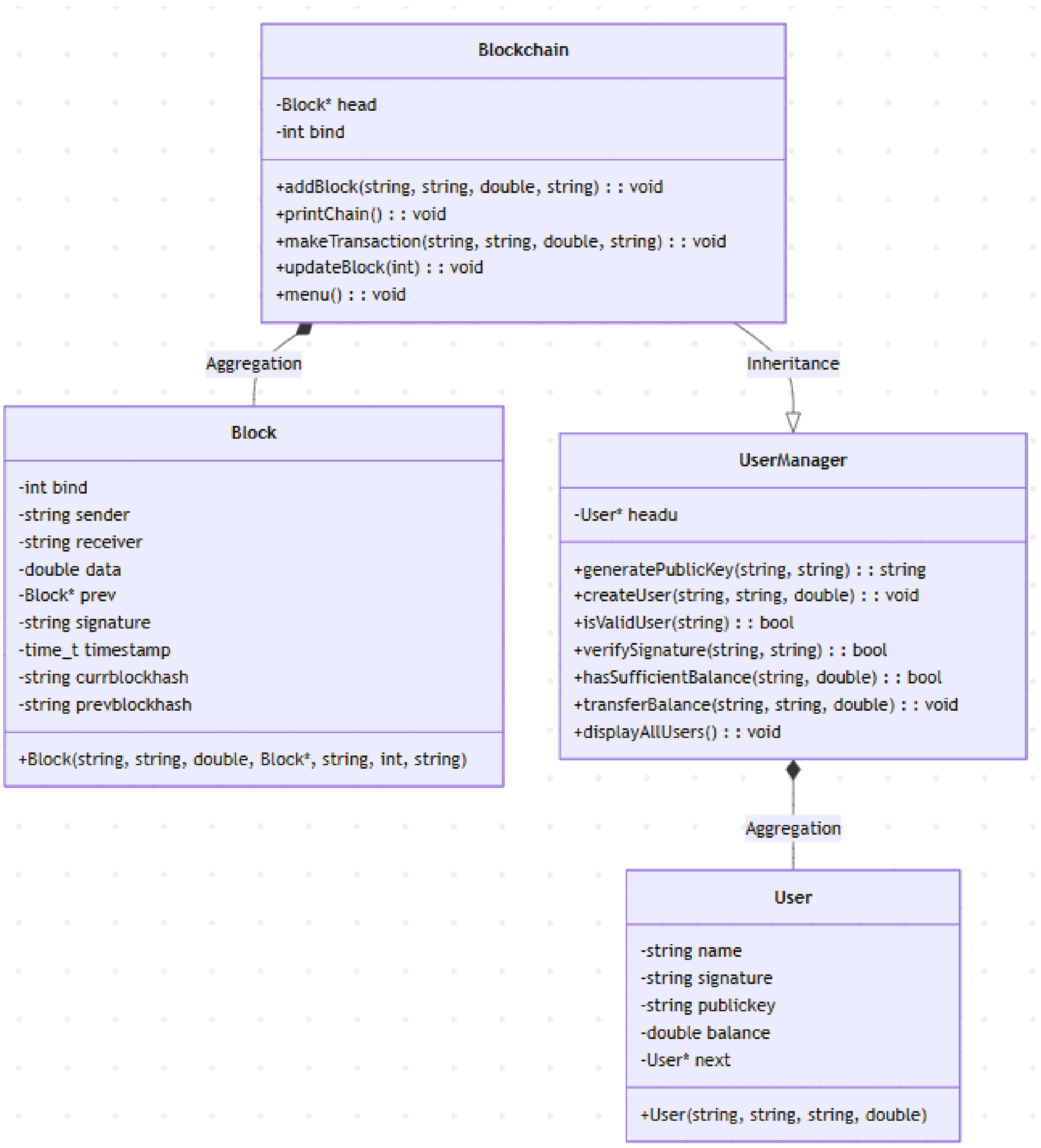
a. Use Case



b. Activity Diagram


Start Transaction
Enter Sender, Receiver, Amount, Signature
Validate User & Signature?
Yes
No
Sufficient Balance?
Transaction Failed: Invalid Credentials
Yes
No
Transfer Balance
Transaction Failed: Low Balance
Create New Block
Update Blockchain
Transaction Successful

c. Class Diagram


Blockchain
-Block* head
-int bind
+addBlock(string, string, double, string) : : void
+printChain() : : void
+makeTransaction(string, string, double, string) : : void
+updateBlock(int) : : void
+menu() : : void
Aggregation
Inheritance
Block
-int bind
-string sender
-string receiver
-double data
-Block* prev
-string signature
-time_t timestamp
-string currblockhash
-string prevblockhash
+Block(string, string, double, Block*, string, int, string)
UserManager
-User* headu
+generatePublicKey(string, string) : : string
+createUser(string, string, double) : : void
+isValidUser(string) : : bool
+verifySignature(string, string) : : bool
+hasSufficientBalance(string, double) : : bool
+transferBalance(string, string, double) : : void
+displayAllUsers() : : void
Aggregation
User
-string name
-string signature
-string publickey
-double balance
-User* next
+User(string, string, string, double)

7. Software Requirements/Architecture

a. Front End

Command Line Interface (CLI) based interaction using standard input and output (cin, cout).

Interface**:**

- A simple menu-driven interface lets users:
  - Create new users.
  - Perform transactions.
  - Display all users.
  - View the blockchain.
  - Test tampering detection.

b. Back End

The backend of the project is written entirely in C++ using Object-Oriented Programming. It handles user management, transaction validation, and blockchain operations.

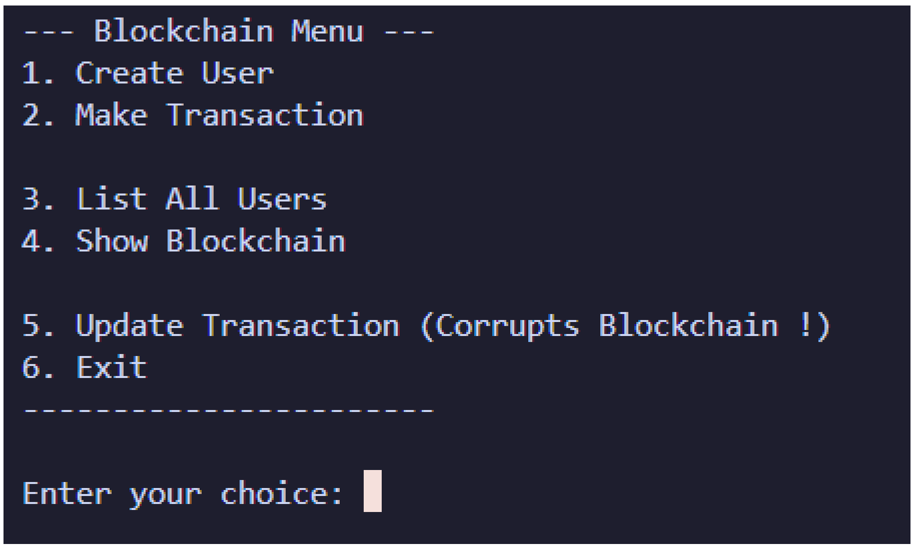
- **User Management:**
  - The `UserManager` class manages users using a linked list.
  - Each user has a name, a private signature, a public key (generated internally), and a balance.
  - It verifies users by checking their public key and signature and ensures they have enough balance before making a transaction.
- **Blockchain Management:**
  - The `Block` structure stores transaction data, timestamps, and hashes.
  - The `Blockchain` class links blocks together in a chain (using backward pointers).
  - Each transaction creates a new block, and a hash is generated for data integrity.
  - It also detects if someone tries to tamper with a block.
- **Transaction Processing:**
  - Before a transaction is accepted, the system checks:
    - Validity of sender and receiver.
    - Correct signature.
    - Sufficient balance.
  - If successful, balances are updated, and a new block is added.

c. Database Connectivity

No external database used. Data is stored dynamically in memory using linked lists.

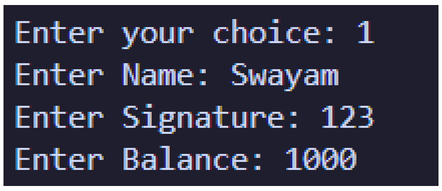
8. GUI

Menu



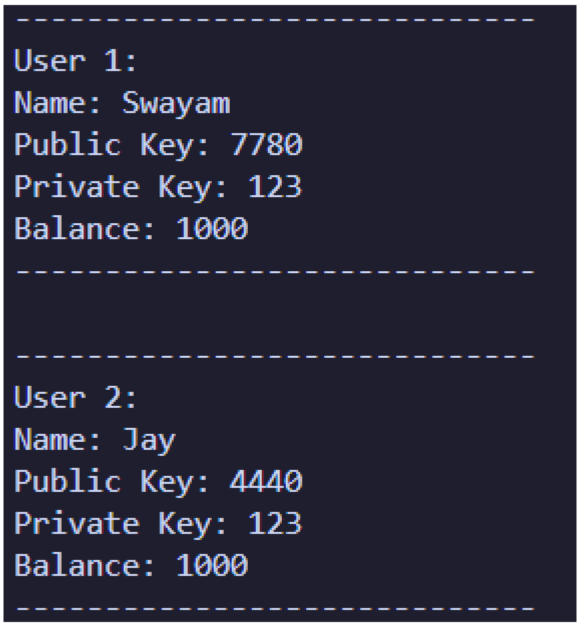
```
--- Blockchain Menu ---
1. Create User
2. Make Transaction

3. List All Users
4. Show Blockchain

5. Update Transaction (Corrupts Blockchain !)
6. Exit
-----------------------

Enter your choice:
```

User Creation



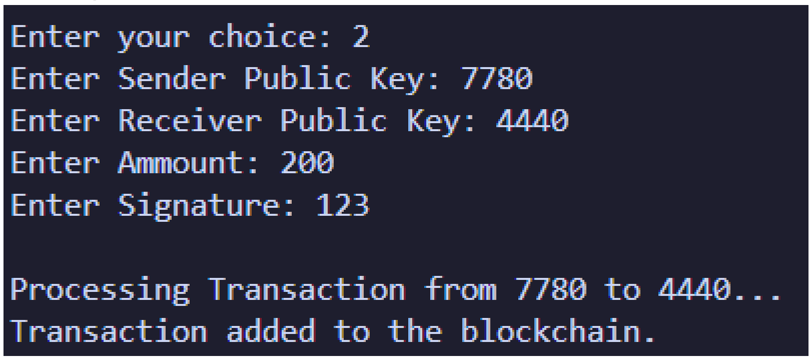
```
Enter your choice: 1
Enter Name: Swayam
Enter Signature: 123
Enter Balance: 1000
```

User List



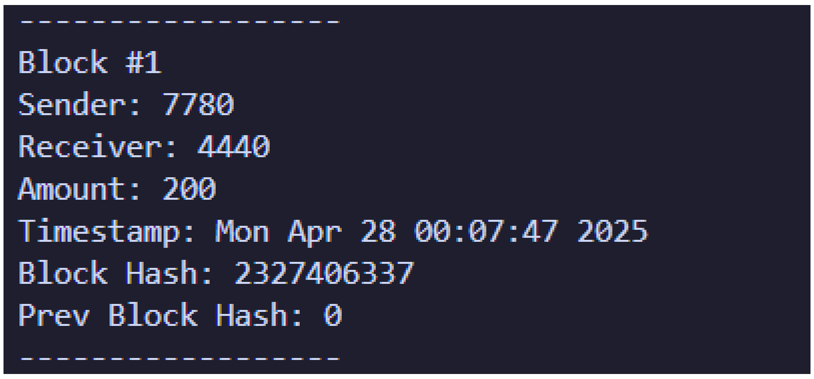
```
-----------------------------
User 1:
Name: Swayam
Public Key: 7780
Private Key: 123
Balance: 1000
-----------------------------


-----------------------------
User 2:
Name: Jay
Public Key: 4440
Private Key: 123
Balance: 1000
-----------------------------
```

Making Transaction



```
Enter your choice: 2
Enter Sender Public Key: 7780
Enter Receiver Public Key: 4440
Enter Ammount: 200
Enter Signature: 123

Processing Transaction from 7780 to 4440...
Transaction added to the blockchain.
```

A Block in Blockchain



```
------------------
Block #1
Sender: 7780
Receiver: 4440
Amount: 200
Timestamp: Mon Apr 28 00:07:47 2025
Block Hash: 2327406337
Prev Block Hash: 0
------------------
```

9. Source Code

https://drive.google.com/file/d/16uCrTdd4hXtK_ox0cm3HP3jDmF_p4yH6/view?usp=drive_link

10.Estimation of Project

Task	**Estimated Time**
Requirement Analysis	2 hours
Design of Classes	3 hours
Implementation (Coding)	6 hours
Testing and Debugging	4 hours
Documentation	2 hours
Total Estimated Time	**17 hours**
Task	**Estimated Time**

11.Testing

Testing is a critical phase that ensures the Blockchain Simulation application performs as intended and meets all functional and non-functional requirements. The C++ based blockchain project was systematically tested for correctness, integrity, and reliability.

Types of Testing Performed

1. Unit Testing

Each module — including user creation, public/private key generation, transaction validation, block creation, and blockchain linking — was individually tested. Manual testing was conducted to verify that each function behaves correctly under different conditions.

2. Integration Testing

Integration tests were performed to ensure that all modules interact correctly:

- Users created through `UserManager` were tested to perform transactions via `Blockchain`.
- Transactions successfully updated balances and created corresponding blocks.
- Blockchain links (`prev` pointers and hashes) were verified after each transaction.

3. Functional Testing

We verified each functionality against the specified requirements:

Functionality	Test Case Description	Result
User Creation	Verify users are created with correct keys and balance	Pass
User Verification	Check public key and signature validation before transaction	Pass
Transaction Validation	Ensure correct balance deduction and addition between users	Pass
Block Creation	Ensure each transaction creates a new block with correct data	Pass
Blockchain Integrity	Confirm that tampering detection works when a block is modified	Pass
Display Users	Verify user details are listed correctly	Pass
Show Blockchain	Display all blocks in correct reverse order	Pass

4. System Testing

The entire blockchain system was run as a full program. Multiple users were created, transactions made, and blockchain history displayed. Additionally, intentional tampering was tested to ensure the system detected hash mismatches correctly.

5. Performance Testing

The system was tested by:

- Creating large numbers of users (~100 users).
- Making many transactions sequentially (~200 transactions).
- Observing the program's speed, memory usage, and integrity checks.
- The system maintained fast response times even with increasing users and blocks.

6. Security Testing

Although advanced encryption is not implemented, basic security measures were tested:

- User verification using private signatures.
- Transaction authorization checks.
- Tampering attempts (via block data modification) were immediately detected and flagged.

Test Environment

- Operating System: Windows 11 and Ubuntu 22.04
- Compiler: g++ 9.4.0 / 10.2.0
- Development Tool: VS Code / g++ command-line
- Processor: Intel i5, 8 GB RAM
- Execution Environment: Terminal-based C++ Execution

Bug Tracking and Fixing

During testing, the following minor issues were identified and resolved:

- Block hash not updating correctly when block data was manually changed (fixed by updating timestamp properly).
- Balance update mismatch during rapid multiple transactions (fixed by correct order of balance deduction and addition).
- Minor formatting issues while displaying blockchain (improved output readability).

12. Conclusion

This mini-project demonstrates a practical blockchain simulation with secure transactions, linked list usage, and basic hashing techniques. It successfully meets the objectives by providing a functional, tamper-detecting blockchain system. The project strengthens understanding of data structures, memory management, and the core concepts behind blockchain technology.