

PROJECT REPORT

Advanced Data Structures

Banking System using Blockchain in C

|  |  |  |
| --- | --- | --- |
| GROUP DETAILS : CS-A BATCH 1 GROUP 6 | | |
| NAME | PRN | ROLL NO. |
| Prasanna Bhalerao | 12210133 | 35 |
| Yash Bhalerao | 12211285 | 36 |
| Atharva Chavle | 12211811 | 50 |
| Chirag Belani | 12211398 | 51 |

|  |  |  |
| --- | --- | --- |
|  | **Table of Contents** |  |
| **1. Introduction** |  | **2** |
| **2. Problem Statement** |  | **3** |
| **3. Features** |  | **3** |
| **4. Concepts Used** |  | **3** |
| **5. Program Code** |  | **4** |
| **6. Output** |  | **15** |
| **7. Future Scope** |  | **17** |

# Introduction

The banking sector plays a pivotal role in the global economy, serving as a cornerstone for financial transactions, investments, and wealth management. With the advent of digital technologies, there has been a growing demand for secure, efficient, and transparent banking systems that can adapt to the evolving needs of customers while ensuring data integrity and confidentiality.   
  
Blockchain technology has emerged as a disruptive force in the financial industry, offering a decentralized and immutable ledger system that can revolutionize traditional banking practices. By leveraging cryptographic techniques and distributed consensus mechanisms, blockchain enables transparent and tamper-proof recording of transactions, thereby enhancing security and trust in financial transactions.   
  
The objective of this project is to develop a banking system using blockchain technology implemented in the C programming language. The project aims to address the shortcomings of traditional banking systems by providing a secure, transparent, and efficient platform for managing financial transactions.

Key Features

* Secure Transaction Management: Implementing blockchain data structures and algorithms to securely record and verify transactions.
* User Account Management: Facilitating account creation, authentication, and management functionalities for users.
* Transaction Processing: Enabling functionalities for deposit, withdrawal, and fund transfer operations, ensuring seamless transaction processing.
* Data Integrity and Confidentiality: Utilizing cryptographic techniques to ensure the integrity and confidentiality of user data and transaction records.

# Problem Statement

The project aims to develop a banking system using blockchain technology in C. The system will address challenges in traditional banking, including security vulnerabilities, inefficiencies, and lack of transparency. Key objectives include implementing robust security measures, streamlining transaction processing, ensuring transparency through blockchain's decentralization, and providing a user-friendly interface. The system will integrate with existing banking infrastructure, comply with regulations, and ensure data integrity through cryptographic techniques.

# Features

* Account Management
* Transaction Processing
* Blockchain Integration
* Security Measures
* User Authentication
* Data Integrity
* Scalability
* Compliance
* User-Friendly Interface
* Error Handling
* Backup and Recovery

1. **Concepts Used**

* Blockchain Technology
* Data Structures (e.g., Linked Lists, Binary Search Trees)
* File Handling (Input/Output operations)
* Dynamic Memory Allocation (malloc, free)
* String Manipulation (strcpy, strcat, sprintf)
* Time Handling (time, time\_t)
* Error Handling and Error Messages (perror)
* Memory Management and Resource Optimization
* Algorithm Design and Implementation (e.g., hashing)
* Input Validation and Sanitization
* Interfacing with External Libraries (e.g., stdio.h, stdlib.h)
* Implementation of Banking Operations (e.g., account creation, deposit, withdrawal, transfer)
* Cryptography and Hashing Algorithms (e.g., computeHash function)
* File I/O Operations (e.g., saving/loading user data and transactions to/from CSV files)
* User Authentication and Authorization

# Program Code

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#define TABLE\_SIZE 10 // Define the size of the hash table for simplicity

typedef struct Block {

int index;

char transactionID[65];

char previousHash[65];

time\_t timestamp;

char data[1024];

char hash[65];

struct Block \*next;

} Block;

typedef struct BSTNode {

char transactionID[65];

struct BSTNode \*left, \*right;

} BSTNode;

typedef struct {

Block \*head;

int length;

BSTNode \*root; // Root node for BST of transactions

} Blockchain;

typedef struct User {

char accountNumber[20];

char name[100];

char mobile[20];

char password[100];

float balance;

struct User \*next;

} User;

typedef struct {

User \*users;

int nextAccountNumber;

User \*hashTable[TABLE\_SIZE]; // Hash table for users

} BankDatabase;

Blockchain blockchain;

int hashFunction(char \*str) {

unsigned long hash = 5381;

int c;

while ((c = \*str++))

hash = hash \* 33 + c;

return hash % TABLE\_SIZE;

}

void computeHash(char \*str, char \*hash) {

unsigned long hashValue = 5381;

int c;

while ((c = \*str++))

hashValue = hashValue \* 33 + c;

sprintf(hash, "%lu", hashValue);

}

BSTNode\* insertBST(BSTNode \*node, char \*transactionID) {

if (node == NULL) {

BSTNode \*temp = malloc(sizeof(BSTNode));

strcpy(temp->transactionID, transactionID);

temp->left = temp->right = NULL;

return temp;

}

if (strcmp(transactionID, node->transactionID) < 0)

node->left = insertBST(node->left, transactionID);

else

node->right = insertBST(node->right, transactionID);

return node;

}

void addBlock(char \*data) {

Block \*newBlock = (Block \*)malloc(sizeof(Block));

newBlock->index = blockchain.length++;

time(&newBlock->timestamp);

strcpy(newBlock->data, data);

sprintf(newBlock->transactionID, "%s-%d", "TRX", newBlock->index);

computeHash(data, newBlock->hash);

newBlock->next = NULL;

if (blockchain.head == NULL) {

strcpy(newBlock->previousHash, "0");

blockchain.head = newBlock;

} else {

Block \*current = blockchain.head;

while (current->next != NULL) {

current = current->next;

}

strcpy(newBlock->previousHash, current->hash);

current->next = newBlock;

}

blockchain.root = insertBST(blockchain.root, newBlock->transactionID); // Insert into BST

}

void initBankDatabase(BankDatabase \*db) {

db->users = NULL;

db->nextAccountNumber = 1;

for (int i = 0; i < TABLE\_SIZE; i++) {

db->hashTable[i] = NULL;

}

}

User \*createUser(BankDatabase \*db, const char \*name, const char \*mobile, const char \*password, float initialDeposit) {

User \*newUser = (User \*)malloc(sizeof(User)); // Allocate memory for the new user

if (!newUser) {

// Handle memory allocation failure

return NULL;

}

sprintf(newUser->accountNumber, "CSAGRP6A%03d", db->nextAccountNumber++); // Generate an account number

strcpy(newUser->name, name); // Copy name to user struct

strcpy(newUser->mobile, mobile); // Copy mobile number to user struct

strcpy(newUser->password, password); // Copy password to user struct

newUser->balance = initialDeposit; // Set initial deposit as user balance

newUser->next = NULL; // Set the next pointer of the new user to NULL

// If the list is empty, make the new user the head

if (db->users == NULL) {

db->users = newUser;

} else {

// Traverse the list to find the last user

User \*current = db->users;

while (current->next != NULL) {

current = current->next;

}

// Append the new user at the end of the list

current->next = newUser;

}

// Calculate the index for the hash table and add the user to the appropriate bucket

int index = hashFunction(newUser->accountNumber);

newUser->next = db->hashTable[index];

db->hashTable[index] = newUser;

return newUser; // Return a pointer to the newly created user

}

User\* findUser(BankDatabase \*db, const char\* accountNumber) {

int index = hashFunction(accountNumber);

User \*current = db->hashTable[index];

while (current) {

if (strcmp(current->accountNumber, accountNumber) == 0) {

return current;

}

current = current->next;

}

return NULL;

}

void saveUsersToCSV(BankDatabase \*db, const char \*filename) {

FILE \*file = fopen(filename, "w");

if (!file) {

perror("Failed to open file");

return;

}

fprintf(file, "AccountNumber,Name,Mobile,Password,Balance\n");

User \*current = db->users;

while (current != NULL) {

fprintf(file, "%s,%s,%s,%s,%.2f\n", current->accountNumber, current->name, current->mobile, current->password, current->balance);

current = current->next;

}

fclose(file);

}

void loadUsersFromCSV(BankDatabase \*db, char \*filename) {

FILE \*file = fopen(filename, "r");

if (!file) {

file = fopen(filename, "w");

if (!file) {

perror("Failed to create file");

return;

}

fprintf(file, "AccountNumber,Name,Mobile,Password,Balance\n");

fclose(file);

printf("New file created: %s\n", filename);

return;

}

char line[256];

fgets(line, sizeof(line), file); // Skip header

while (fgets(line, sizeof(line), file) != NULL) {

char accountNumber[20], name[100], mobile[20], password[100];

float balance;

sscanf(line, "%[^,],%[^,],%[^,],%[^,],%f", accountNumber, name, mobile, password, &balance);

User \*user = createUser(db, name, mobile, password, balance);

if (user) {

strcpy(user->accountNumber, accountNumber); // Restore account number directly

}

}

fclose(file);

}

void saveTransactionsToCSV(const char \*filename) {

FILE \*file = fopen(filename, "w");

if (!file) {

perror("Failed to open file");

return;

}

fprintf(file, "Index,TransactionID,PreviousHash,Timestamp,Data,Hash\n");

Block \*current = blockchain.head;

while (current != NULL) {

fprintf(file, "%d,%s,%s,%ld,%s,%s\n", current->index, current->transactionID, current->previousHash, current->timestamp, current->data, current->hash);

current = current->next;

}

fclose(file);

}

void loadTransactionsFromCSV(const char \*filename) {

FILE \*file = fopen(filename, "r");

if (!file) {

file = fopen(filename, "w");

if (!file) {

perror("Failed to create file");

return;

}

fprintf(file, "Index,TransactionID,PreviousHash,Timestamp,Data,Hash\n");

fclose(file);

printf("New transactions file created: %s\n", filename);

return;

}

char line[2048];

fgets(line, sizeof(line), file); // Skip header

while (fgets(line, sizeof(line), file) != NULL) {

int index;

char transactionID[65], previousHash[65], data[1024], hash[65];

long timestamp;

sscanf(line, "%d,%64[^,],%64[^,],%ld,%1023[^,],%64s", &index, transactionID, previousHash, &timestamp, data, hash);

Block \*newBlock = (Block \*)malloc(sizeof(Block));

newBlock->index = index;

strcpy(newBlock->transactionID, transactionID);

strcpy(newBlock->previousHash, previousHash);

newBlock->timestamp = timestamp;

strcpy(newBlock->data, data);

strcpy(newBlock->hash, hash);

newBlock->next = blockchain.head;

blockchain.head = newBlock; // prepend to maintain order on reload

}

fclose(file);

}

void printUsers(BankDatabase \*db) {

User \*current = db->users;

printf("List of Users:\n");

while (current != NULL) {

printf("Account #%s: %s, Mobile: %s, Balance: Rs.%.2f\n", current->accountNumber, current->name, current->mobile, current->balance);

current = current->next;

}

}

int authenticateUser(BankDatabase \*db, const char\* accountNumber, const char\* password) {

User \*user = findUser(db, accountNumber);

if (user && strcmp(user->password, password) == 0) {

return 1; // Authentication successful

}

return 0; // Authentication failed

}

void transaction(BankDatabase \*db, const char\* accountNumber, float amount, int type) {

char password[100];

printf("Enter password for account %s: ", accountNumber);

scanf("%s", password); // Use scanf("%99s", password) to prevent buffer overflow

if (!authenticateUser(db, accountNumber, password)) {

printf("Authentication failed. Transaction aborted.\n");

return;

}

User \*user = findUser(db, accountNumber);

if (user == NULL) {

printf("Account number %s not found.\n", accountNumber);

return;

}

char data[1024];

if (type == 1) { // Deposit

user->balance += amount;

sprintf(data, "Deposited Rs.%.2f to %s. New Balance: Rs.%.2f", amount, user->accountNumber, user->balance);

printf("Rs.%.2f deposited to Account #%s. New Balance: Rs.%.2f\n", amount, user->accountNumber, user->balance);

} else if (type == 2) { // Withdrawal

if (user->balance >= amount) {

user->balance -= amount;

sprintf(data, "Withdrawn Rs.%.2f from %s. New Balance: Rs.%.2f", amount, user->accountNumber, user->balance);

printf("Rs.%.2f withdrawn from Account #%s. New Balance: Rs.%.2f\n", amount, user->accountNumber, user->balance);

} else {

printf("Insufficient funds for withdrawal.\n");

return;

}

}

addBlock(data);

}

void transfer(BankDatabase \*db, char\* fromAccount, char\* toAccount, float amount) {

char password[100];

printf("Enter password for account %s: ", fromAccount);

scanf("%s", password); // Use scanf("%99s", password) to prevent buffer overflow

if (!authenticateUser(db, fromAccount, password)) {

printf("Authentication failed. Transfer aborted.\n");

return;

}

User \*fromUser = findUser(db, fromAccount);

User \*toUser = findUser(db, toAccount);

if (fromUser == NULL || toUser == NULL) {

printf("One or both account numbers not found.\n");

return;

}

if (fromUser->balance >= amount) {

fromUser->balance -= amount;

toUser->balance += amount;

char data[1024];

sprintf(data, "Transferred Rs.%.2f from %s to %s", amount, fromUser->accountNumber, toUser->accountNumber);

printf("Rs.%.2f transferred from Account #%s to Account #%s\n", amount, fromAccount, toAccount);

addBlock(data);

} else {

printf("Insufficient funds in source account.\n");

}

}

void menu() {

BankDatabase db;

initBankDatabase(&db);

loadUsersFromCSV(&db, "D:\\YASH\\COLLEGE\\ADS\\CP\\Banking System using Blockchain\\users.csv");

loadTransactionsFromCSV("D:\\YASH\\COLLEGE\\ADS\\CP\\Banking System using Blockchain\\transactions.csv");

int choice;

char accountNumber[20];

float amount;

char name[100];

char mobile[20];

char password[100];

char confirmPassword[100];

do {

printf("\n--- Bank Menu ---\n");

printf("1. Create Account\n");

printf("2. Deposit Money\n");

printf("3. Withdraw Money\n");

printf("4. Transfer Money\n");

printf("5. View Accounts\n");

printf("6. Exit\n");

printf("Choose an option: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter name: ");

scanf("%s", name); // Use scanf("%99s", name) to prevent buffer overflow

printf("Enter mobile number: ");

scanf("%s", mobile); // Use scanf("%19s", mobile) to prevent buffer overflow

if (strlen(mobile) != 10) {

printf("Error: Mobile number must be exactly 10 digits long.\n");

break; // Break out of the switch-case

}

printf("Create password: ");

scanf("%s", password); // Use scanf("%99s", password) to prevent buffer overflow

printf("Confirm password: ");

scanf("%s", confirmPassword); // Use scanf("%99s", confirmPassword) to prevent buffer overflow

if (strcmp(password, confirmPassword) != 0) {

printf("Passwords do not match. Account creation failed.\n");

break;

}

printf("Initial deposit: ");

scanf("%f", &amount);

User \*user = createUser(&db, name, mobile, password, amount);

if (user) {

printf("Account created successfully. Account Number: %s\n", user->accountNumber);

char data[1024];

sprintf(data, "Created account for %s with initial deposit of Rs.%.2f. Account Number: %s", name, amount, user->accountNumber);

addBlock(data);

saveUsersToCSV(&db, "D:\\YASH\\COLLEGE\\ADS\\CP\\Banking System using Blockchain\\users.csv");

saveTransactionsToCSV("D:\\YASH\\COLLEGE\\ADS\\CP\\Banking System using Blockchain\\transactions.csv");

}

break;

case 2:

printf("Enter account number: ");

scanf("%s", accountNumber);

printf("Enter amount to deposit: ");

scanf("%f", &amount);

transaction(&db, accountNumber, amount, 1);

break;

case 3:

printf("Enter account number: ");

scanf("%s", accountNumber);

printf("Enter amount to withdraw: ");

scanf("%f", &amount);

transaction(&db, accountNumber, amount, 2);

break;

case 4:

{

char toAccount[20];

printf("Enter from account number: ");

scanf("%s", accountNumber);

printf("Enter to account number: ");

scanf("%s", toAccount);

printf("Enter amount to transfer: ");

scanf("%f", &amount);

transfer(&db, accountNumber, toAccount, amount);

saveUsersToCSV(&db, "D:\\YASH\\COLLEGE\\ADS\\CP\\Banking System using Blockchain\\users.csv");

saveTransactionsToCSV("D:\\YASH\\COLLEGE\\ADS\\CP\\Banking System using Blockchain\\transactions.csv");

}

break;

case 5:

printUsers(&db);

break;

case 6:

printf("Exiting and saving data...\n");

saveUsersToCSV(&db, "D:\\YASH\\COLLEGE\\ADS\\CP\\Banking System using Blockchain\\users.csv");

saveTransactionsToCSV("D:\\YASH\\COLLEGE\\ADS\\CP\\Banking System using Blockchain\\transactions.csv");

printf("Data saved. Exiting program.\n");

break;

default:

printf("Invalid option.\n");

}

} while (choice != 6);

}

int main() {

blockchain.head = NULL;

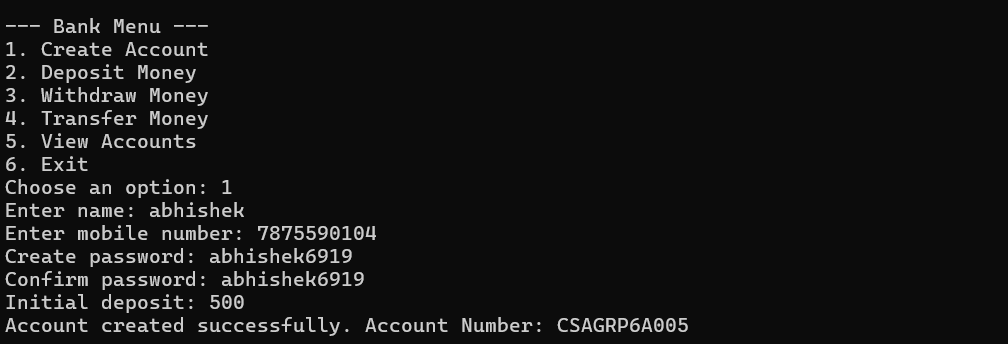
blockchain.length = 0;

menu();

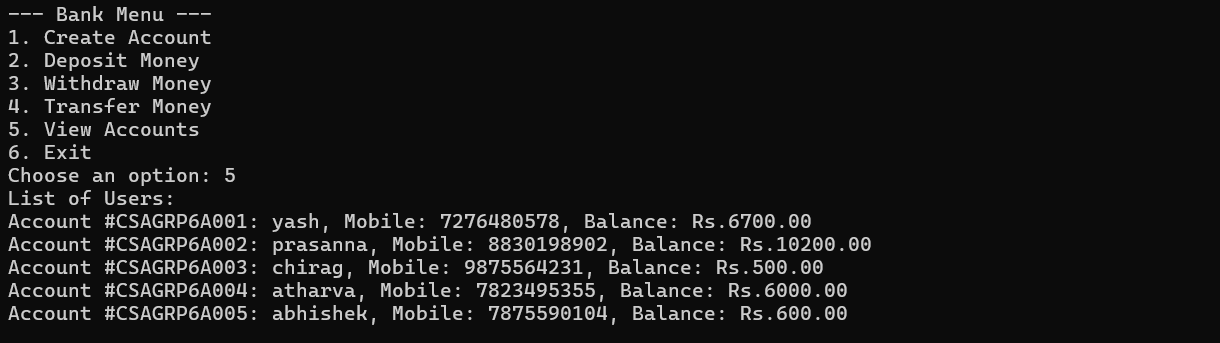
return 0;

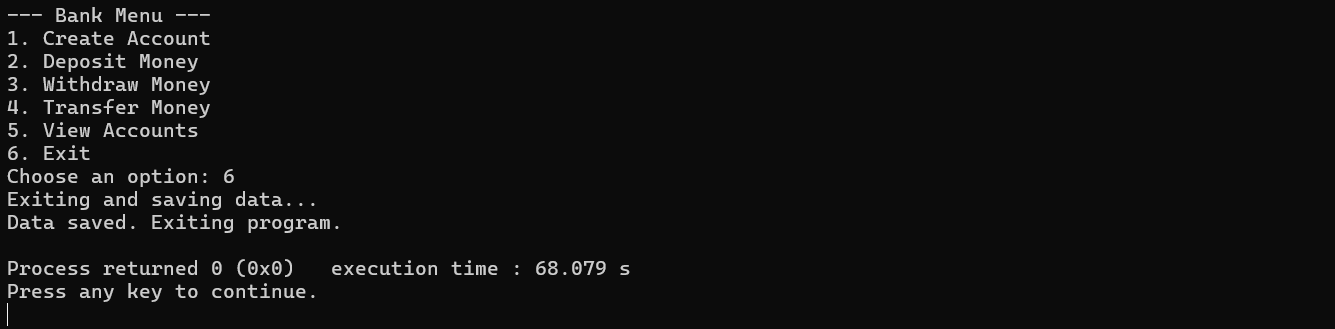
}

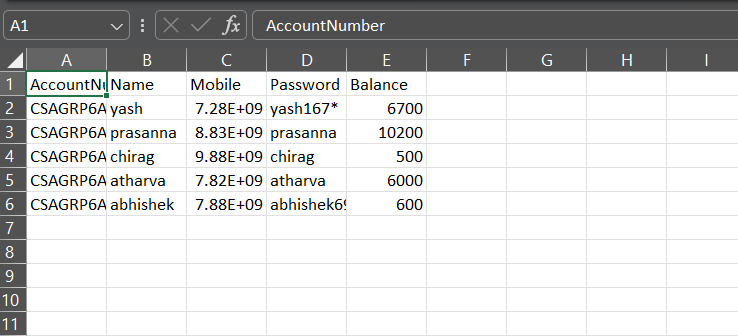
1. **Output**

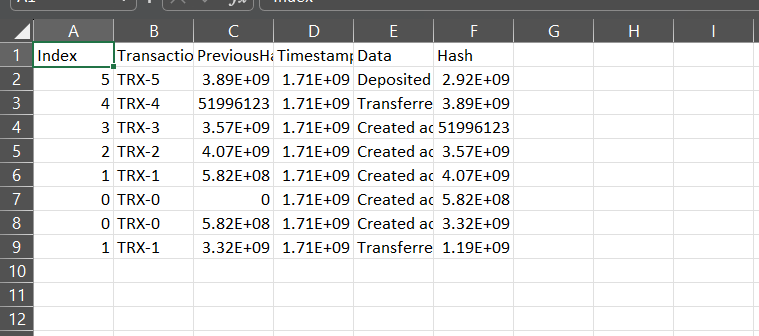


****

****







1. **Future Scope**

* **Smart Contract Integration**
* **Multi-Currency Support**
* **Enhanced Security Measures**
* **Integration with External Services**
* **Blockchain Interoperability**
* **Enhanced User Experience**
* **Regulatory Compliance Enhancements**
* **Data Analytics and Reporting**