

**Arab Academy for Science, Technology & Maritime**

**Transport.**



**Computer Science department**

TRANSACTION HISTORY IN BANKING

**Presented By:**

**1- Jomana Amr Fawzy 221008668**

**2-Tasbeem Mostafa Adam 221008798**

**3- Hend Nabil Abo Elhassan 221008573**

**4-Yahya Mohamed Saadeldin 221018204**

**Presented To:**

**DR.TAMER ABDEL LATIF**

**Table of contents:**

**Table Of Figures**...............................................................................................................II

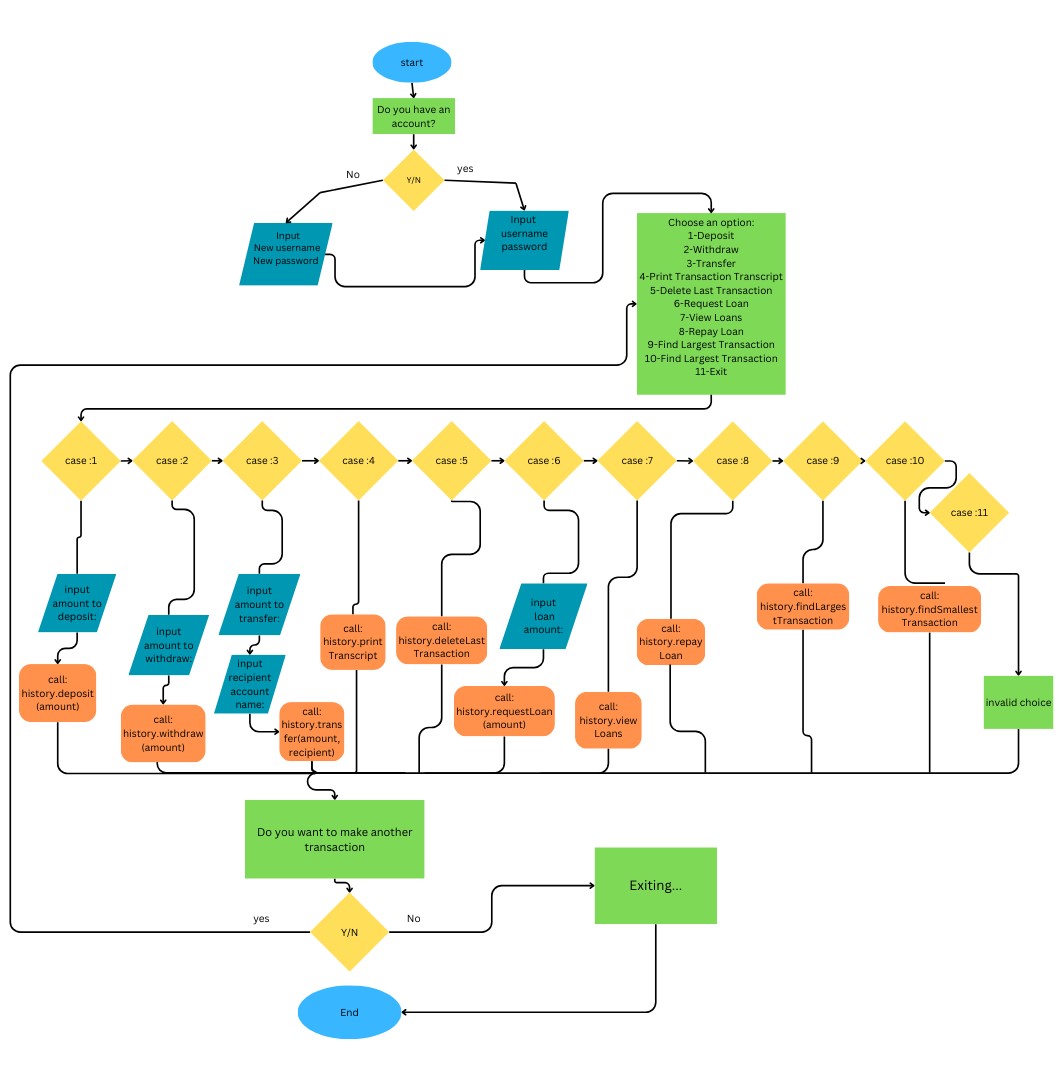
**Chapter 1:** Design…..........................................................................................................1

**Chapter 2**: Code..................................................................................................................2

**2.1 Sudo Code**…………………………………………………………………….2  
 **2.2 C++ Code**…………………………………………..…………………………3  
**References**.............................................................................................................................4

**Table of Figures:**

**Figure1.Flow Chart**...................................................................................................................1

** Chapter one: Design**

1

**Figure1**

**Chapter Two: Code**

**Psudo Code:**

Display the welcome text

Asking if the user has an account

If they do:

Ask for the password and the username

If they don’t:

Ask for a new password and username

Then ask them for the username and password

Once registered:

Display the menu:

1-deposit

2-withdraw

3-transfer

4-print transaction transcript

5-delete last transaction

6-request loan

7-view loans

8-repay loan

9-exist

Display the menu after each choice is finished until the user choose exit

If the user chose deposit:

User enters the deposit amount,

then updates the current balance, adds the deposit transaction to the transaction history

If the user chose to withdraw:

User enters the withdraw amount

the withdrawal amount then id subtracted from the current balance.

If the user chose to transfer:

User to enter the amount they wish to transfer and the recipient's account name.

the specified amount is taken from the user's account balance.

If the user chose to print Transaction Transcript:

the program displays a transcript of all recorded transactions.

If the user chose to Delete Last Transaction:

The program deletes the last transaction details from the history.

If the user chose to Request Loan:

user enters the loan amount.

The current balance is updated, and the loan request details are added to the transaction history stack.

Display the requested loan amount and the updated balance.

If the user chose to View Loans:

Display a list of all active loans, showing the loan amount for each.

If there are no active loans, display a message saying that to the user.

If the user chose to Repay Loan:

calculate the total amount due for repayment.

user enters the repayment amount, and the program updates the loan balances and the user's current balance accordingly.

If the user chose to exist:

the user exits the banking system.

The program terminates, and the repeated message is not displayed.

**C++ Code:**

#include <iostream>

#include <fstream>

using namespace std;

const int maxtrans = 100;

const int passlength = 9;

class Node {

public:

double amount;

double balance;

Node\* next;

public:

Node(double amt, double bal) {

amount = amt;

balance = bal;

next = nullptr;

}

double getAmount() { return amount; }

double getBalance() { return balance; }

Node\* getNext() { return next; }

void setNext(Node\* nextNode) { next = nextNode; }

void decreaseBalance(double amountPaid) {

balance -= amountPaid;

}

};

class Stack {

public:

string types[maxtrans];

double amounts[maxtrans];

double balances[maxtrans];

int top;

public:

Stack() {

top = -1;

}

bool isEmpty() {

return top == -1;

}

bool isFull() {

return top == maxtrans - 1;

}

void push(string type, double amount, double balance) {

if (isFull()) {

cout << "Transaction history is full" << endl;

}

else {

top++;

types[top] = type;

amounts[top] = amount;

balances[top] = balance;

}

}

void popLastTransaction() {

if (isEmpty()) {

cout << "Transaction history is empty" << endl;

}

else {

cout << "Last transaction removed:" << endl;

cout << "Type: " << types[top] << ", Amount: " << amounts[top] << ", Balance: " << balances[top] << endl;

top--;

}

}

void printTransactions() {

if (isEmpty()) {

cout << "Transaction history is empty" << endl;

}

else {

cout << "Transaction Transcript:" << endl;

for (int i = 0; i <= top; i++) {

cout << "Type: " << types[i] << ", Amount: " << amounts[i] << ", Balance: " << balances[i] << endl;

}

}

}

};

class BSTNode {

public:

double amount;

double balance;

BSTNode\* left;

BSTNode\* right;

BSTNode(double amt, double bal) {

amount = amt;

balance = bal;

left = nullptr;

right = nullptr;

}

};

class BST {

private:

BSTNode\* root;

public:

BST() {

root = nullptr;

}

void insert(double amount, double balance) {

root = insertRecursive(root, amount, balance);

}

void inOrderTraversal() {

inOrderTraversalRecursive(root);

}

double findLargestTransaction() {

BSTNode\* node = root;

while (node->right != nullptr) {

node = node->right;

}

return node->amount;

}

double findSmallestTransaction() {

BSTNode\* node = root;

while (node->left != nullptr) {

node = node->left;

}

return node->amount;

}

private:

BSTNode\* insertRecursive(BSTNode\* node, double amount, double balance) {

if (node == nullptr) {

return new BSTNode(amount, balance);

}

if (amount < node->amount) {

node->left = insertRecursive(node->left, amount, balance);

}

else if (amount > node->amount) {

node->right = insertRecursive(node->right, amount, balance);

}

return node;

}

void inOrderTraversalRecursive(BSTNode\* node) {

if (node != nullptr) {

inOrderTraversalRecursive(node->left);

cout << "Amount: " << node->amount << ", Balance: " << node->balance << endl;

inOrderTraversalRecursive(node->right);

}

}

};

class transhisto {

Stack t;

double currentBalance;

string username;

string password;

Node\* loansHead;

BST bst; // Binary Search Tree

public:

transhisto() {

currentBalance = 0.0;

loansHead = nullptr;

}

void setUsername(string name) {

username = name;

}

void setPassword(string pass) {

if (pass.length() == passlength) {

password = pass;

}

else {

cout << "Invalid password length." << endl;

}

}

bool login(string name, string pass) {

return (username == name && password == pass);

}

void deposit(double amount) {

currentBalance += amount;

t.push("Deposit", amount, currentBalance);

bst.insert(amount, currentBalance); // Insert transaction into BST

cout << "Deposited " << amount << ". New balance: " << currentBalance << endl;

}

void withdraw(double amount) {

if (currentBalance < amount) {

cout << "Error: Insufficient funds for withdrawal. Balance: " << currentBalance << endl;

return;

}

currentBalance -= amount;

t.push("Withdrawal", amount, currentBalance);

bst.insert(-amount, currentBalance);

cout << "Withdrawn " << amount << ". New balance: " << currentBalance << endl;

}

void transfer(double amount, string recipient) {

if (currentBalance < amount) {

cout << "Error: Insufficient funds for transfer. Balance: " << currentBalance << endl;

return;

}

currentBalance -= amount;

t.push("Transfer", amount, currentBalance);

cout << "Transferred " << amount << " to account: " << recipient << ". New balance: " << currentBalance << endl;

}

void deleteLastTransaction() {

t.popLastTransaction();

}

void printTranscript() {

t.printTransactions();

}

string getUsername() {

return username;

}

void requestLoan(double amount) {

Node\* newNode = new Node(amount, amount);

newNode->setNext(loansHead);

loansHead = newNode;

currentBalance += amount;

t.push("Loan Request", amount, currentBalance);

cout << "Loan of " << amount << " requested successfully. New balance: " << currentBalance << endl;

}

void viewLoans() {

Node\* current = loansHead;

bool hasActiveLoans = false;

while (current) {

if (current->getBalance() > 0) {

hasActiveLoans = true;

cout << "Loan Amount: " << current->getAmount() << endl;

}

current = current->getNext();

}

if (!hasActiveLoans) {

cout << "No active loans." << endl;

}

}

void repayLoan() {

double loanAmount = 0;

double totalRepayments = 0;

for (int i = 0; i <= t.top; i++) {

if (t.types[i] == "Loan Request") {

loanAmount += t.amounts[i];

} else if (t.types[i] == "Loan Repayment") {

totalRepayments += t.amounts[i];

}

}

if (loanAmount == 0 || totalRepayments == loanAmount) {

cout << "No loan to repay." << endl;

return;

}

cout << "Total loan amount: " << loanAmount << endl;

double remainingLoan = loanAmount - totalRepayments;

if (remainingLoan <= 0) {

cout << "Congratulations! You have successfully repaid the entire loan." << endl;

return;

}

cout << "Amount needed to be repaid: " << remainingLoan << endl;

double repaymentAmount;

cout << "Enter amount to repay: ";

cin >> repaymentAmount;

if (repaymentAmount <= 0) {

cout << "Invalid repayment amount. Please enter a positive value." << endl;

return;

}

if (repaymentAmount > remainingLoan) {

cout << "Error: Repayment amount exceeds remaining loan balance." << endl;

return;

}

currentBalance -= repaymentAmount;

t.push("Loan Repayment", -repaymentAmount, currentBalance);

cout << "Loan repayment of " << repaymentAmount << " processed successfully." << endl;

cout << "Remaining loan balance after payment: " << (remainingLoan - repaymentAmount) << endl;

}

double findLargestTransaction() {

return bst.findLargestTransaction();

}

double findSmallestTransaction() {

return bst.findSmallestTransaction();

}

};

bool isLoggedIn(string username, string password) {

ifstream read(username + ".txt");

if (!read) {

cout << "User does not exist." << endl;

return false;

}

string susername, spass;

getline(read, susername);

getline(read, spass);

if (susername == username && spass == password) {

return true;

}

else {

cout << "Invalid username or password." << endl;

return false;

}

}

void registerUser(string username, string password) {

ofstream file(username + ".txt");

if (file.is\_open()) {

file << username << endl << password;

file.close();

cout << "Registration successful. Please log in..." << endl;

} else {

cout << "Error: Unable to create user file." << endl;

return;

}

}

int main() {

int choice;

bool registered = false;

transhisto history;

cout << "\t\t\t\*\*\*\*\*Welcome to the banking system!\*\*\*\*\*" << endl;

while (!registered) {

cout << "Do you have an account? (1: yes / 2: no): ";

int ha;

cin >> ha;

if (ha == 1) {

string username, password;

cout << "Enter your username: ";

cin >> username;

cout << "Enter your password: ";

cin >> password;

if (isLoggedIn(username, password)) {

cout << "Logged in successfully." << endl;

registered = true;

}

else {

cout << "Invalid username or password. Please try again." << endl;

}

}

else if (ha == 2) {

string username, password;

cout << "Enter your desired username: ";

cin >> username;

cout << "Enter your desired password, must be 9 digits: ";

cin >> password;

if (password.length() == passlength) {

registerUser(username, password);

}

else {

cout << "Invalid password length. Please try again." << endl;

}

}

else {

cout << "Invalid choice. Please try again." << endl;

continue;

}

}

do {

cout << "Choose an option:" << endl;

cout << "1. Deposit" << endl;

cout << "2. Withdraw" << endl;

cout << "3. Transfer" << endl;

cout << "4. Print Transaction Transcript" << endl;

cout << "5. Delete Last Transaction" << endl;

cout << "6. Request Loan" << endl;

cout << "7. View Loans" << endl;

cout << "8. Repay Loan" << endl;

cout << "9. Find Largest Transaction" << endl;

cout << "10. Find Smallest Transaction" << endl;

cout << "11. Exit" << endl;

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1: {

cout << "Enter amount to deposit: ";

double amount;

cin >> amount;

history.deposit(amount);

break;

}

case 2: {

cout << "Enter amount to withdraw: ";

double amount;

cin >> amount;

history.withdraw(amount);

break;

}

case 3: {

cout << "Enter amount to transfer: ";

double amount;

cin >> amount;

cout << "Enter recipient account name: ";

string recipient;

cin >> recipient;

history.transfer(amount, recipient);

break;

}

case 4: {

history.printTranscript();

break;

}

case 5: {

history.deleteLastTransaction();

break;

}

case 6: {

cout << "Enter loan amount: ";

double amount;

cin >> amount;

history.requestLoan(amount);

break;

}

case 7: {

history.viewLoans();

break;

}

case 8: {

history.repayLoan();

break;

}

case 9: {

double largestTransaction = history.findLargestTransaction();

cout << "Largest Transaction Amount: " << largestTransaction << endl;

break;

}

case 10: {

double smallestTransaction = history.findSmallestTransaction();

cout << "Smallest Transaction Amount: " << smallestTransaction << endl;

break;

}

case 11: {

cout << "Exiting..." << endl;

return 0;

}

default: {

cout << "Invalid choice. Please try again." << endl;

break;

}

}

char anotherTransaction;

cout << "Do you want to perform another transaction? (Y/N): ";

cin >> anotherTransaction;

if (anotherTransaction != 'Y' && anotherTransaction != 'y') {

break;

}

} while (true);

return 0;

}

**References**

1- https://www.canva.com/online-whiteboard/flowcharts/