#include <iostream>

#include <string>

#include <vector>

#include <deque>

// Transaction class

class Transaction {

public:

std::string farmer\_id;

double amount;

std::string transaction\_type;

Transaction(std::string id, double amt, std::string type)

: farmer\_id(id), amount(amt), transaction\_type(type) {}

std::string toString() const {

return transaction\_type + ": UGX " + std::to\_string(amount);

}

};

// Farmer account structure

struct FarmerAccount {

std::string farmer\_id;

double balance;

std::deque<Transaction> transaction\_stack;

};

// SACCO class

class SACCO {

private:

std::vector<FarmerAccount> accounts;

FarmerAccount\* findAccount(const std::string& farmer\_id) {

for (auto& acc : accounts) {

if (acc.farmer\_id == farmer\_id) {

return &acc;

}

}

return 0;

}

public:

void deposit(const std::string& farmer\_id, double amount) {

FarmerAccount\* acc = findAccount(farmer\_id);

if (!acc) {

FarmerAccount new\_acc = {farmer\_id, 0.0, {}};

accounts.push\_back(new\_acc);

acc = &accounts.back();

}

acc->balance += amount;

acc->transaction\_stack.push\_front(Transaction(farmer\_id, amount, "Deposit"));

std::cout << "✅ Deposit successful: " << farmer\_id << " - UGX " << amount << "\n";

}

bool withdraw(const std::string& farmer\_id, double amount) {

FarmerAccount\* acc = findAccount(farmer\_id);

if (!acc || acc->balance < amount) {

std::cout << "Withdrawal failed: Insufficient balance\n";

return false;

}

acc->balance -= amount;

acc->transaction\_stack.push\_front(Transaction(farmer\_id, amount, "Withdrawal"));

std::cout << "✅ Withdrawal successful: " << farmer\_id << " - UGX " << amount << "\n";

return true;

}

double checkBalance(const std::string& farmer\_id) {

FarmerAccount\* acc = findAccount(farmer\_id);

return acc ? acc->balance : 0.0;

}

void getStatement(const std::string& farmer\_id, int N) {

FarmerAccount\* acc = findAccount(farmer\_id);

if (!acc) {

std::cout << "No transactions found for this farmer.\n";

return;

}

int count = std::min(N, (int)acc->transaction\_stack.size());

std::cout << "Last " << count << " transactions for " << farmer\_id << ":\n";

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

std::cout << " - " << acc->transaction\_stack[i].toString() << "\n";

}

}

};

int main() {

SACCO sacco;

int choice;

std::string farmer\_id;

double amount;

int N;

do {

std::cout << "\n===== SACCO Management Menu =====\n";

std::cout << "1. Deposit\n";

std::cout << "2. Withdraw\n";

std::cout << "3. Check Balance\n";

std::cout << "4. View Last N Transactions\n";

std::cout << "5. Exit\n";

std::cout << "Select an option (1-5): ";

std::cin >> choice;

if (choice == 1) {

std::cout << "Enter Farmer ID: ";

std::cin >> farmer\_id;

std::cout << "Enter Amount to Deposit: UGX ";

std::cin >> amount;

sacco.deposit(farmer\_id, amount);

}

else if (choice == 2) {

std::cout << "Enter Farmer ID: ";

std::cin >> farmer\_id;

std::cout << "Enter Amount to Withdraw: UGX ";

std::cin >> amount;

sacco.withdraw(farmer\_id, amount);

}

else if (choice == 3) {

std::cout << "Enter Farmer ID: ";

std::cin >> farmer\_id;

std::cout << "Current Balance: UGX " << sacco.checkBalance(farmer\_id) << "\n";

}

else if (choice == 4) {

std::cout << "Enter Farmer ID: ";

std::cin >> farmer\_id;

std::cout << "Enter number of transactions to view: ";

std::cin >> N;

sacco.getStatement(farmer\_id, N);

}

else if (choice == 5) {

std::cout << "Exiting... Thank you!\n";

}

else {

std::cout << "Invalid choice. Please select between 1 and 5.\n";

}

} while (choice != 5);

return 0;

}