PROJECT

Cash flow minimizer in c using data structures

Submitted By:

NAME: Shivam Khurana

Paramvir Singh

Jaideep Singh

Submitted to:

Ms. Shilpi Garg

Designation:

Teacher

UID: 24BCA20020

24BCA20013

24BCA20021

SECTION: 24BCV-1

SUBJECT: Data Structure lab

Cash Flow Minimizer Using Data Structures

Abstract:

The primary goal of this C program is to minimize the number of transactions required to settle debts among a group of people. Given a matrix of who owes how much to whom, the program simplifies the settlement process by determining a minimal set of payments to resolve all debts.

This project presents a Cash Flow Minimizer system aimed at optimizing financial transactions within a group or organization.

It leverages efficient data structures and algorithms to reduce the number of cash transactions required to settle debts among participants.

Introduction:

Managing and minimizing cash flows in group transactions (e.g., friends splitting bills, business expense sharing) is a classic optimization problem. Rather than each individual paying others directly, a minimized set of transactions can significantly reduce complexity and transaction costs.

Data Structures and Algorithms (DSA) play a crucial role in solving this problem. It allows quick access to participant balances. These structures work together to simplify cash settlements. Beyond personal finance, this system can be extended to **corporate accounting**, **blockchain applications**, and **decentralized finance (DeFi)** where transactional efficiency is key.

Problem Statement

Designing a system to minimize the number of cash transactions requires addressing multiple challenges:

Optimal Transaction Reduction:

- Eliminate unnecessary transactions by offsetting debts.
- Minimize the number of payments to achieve balance.

Efficient Balance Calculation:

- Compute net balances for all participants after multiple transactions.
- Handle positive (creditor) and negative (debtor) balances.

Interactive and Scalable System:

- Provide real-time optimization suggestions.
- Adapt to growing group sizes or transaction volumes.

Data Structures Used

Data Structure	Type	Purpose
Graph [][]	2D Array (Matrix)	Represents the initial debts; graph[i][j] is the amount Person i owes to Person j.
Amount []	1D Array	Stores net amount each person owes or is owed. Positive = to receive, Negative = to pay.
Constants (MAX)	Integer Macro	Used to define the maximum number of people (10).
SettleDebts(int amount[],int n)	Recursion	To settle the debts of maximum number of people and to repeating use of function

Algorithm Explanation:

Step-by-Step Breakdown:

1. Input Collection

   A. Start the program.

   B. Ask the user to input the number of people n (2 ≤ n ≤ MAX).

   C. Create a 2D array graph[n][n] to store the amount each person owes to another.

   D. For each pair (i, j) where i ≠ j:

      - Prompt the user to enter the amount Person i owes to Person j.
      - Store this value in graph[i][j].
      - Set graph[i][i] to 0.

2. Calculate Net Amount for Each Person

   A. Create an array amount[n] and initialize all values to 0.

   B. For each person i from 0 to n-1:

      - For each person j from 0 to n-1:
         - Update amount[i] = amount[i] + (graph[j][i] - graph[i][j]).
      - This computes how much the person should receive or pay in total.

3. Settle Debts Recursively

Call the recursive function settleDebts(amount, n).

Inside settleDebts:

   A. Base Case:

      a. Find the person with the maximum credit (getMaxCredit) and the person with the maximum debit (getMaxDebit).

      b. If both the maximum credit and debit are 0, return (debts are settled).

B. Find Minimum Transfer Amount:

   a. Calculate minAmount = min(amount[maxCreditor], -amount[maxDebtor]).

C. Perform Transaction:

   a. Subtract minAmount from amount[maxCreditor].

   b. Add minAmount to amount[maxDebtor].

   c. Print: Person maxDebtor pays minAmount to Person maxCreditor.

D. Recursive Call:

   a. Call settleDebts(amount, n) again to process remaining balances.

4. End Program

   A. Once all debts are settled, print a final message:
   "All debts settled with minimum number of transactions."

   B. End the program.

Key Logic:

1. getMaxCredit(int amount[], int n)
   - Purpose: Finds the index of the person with the maximum net credit.
   - Returns: Index of person who should receive the most money.
2. getMaxDebit(int amount[], int n)
   - Purpose: Finds the index of the person with the maximum net debit.
   - Returns: Index of person who should pay the most money.
3. settleDebts(int amount[], int n)
   - Purpose: Recursively finds the maximum creditor and debtor, performs a transaction, and updates the balances until all are settled (i.e., all values in amount[] become zero).

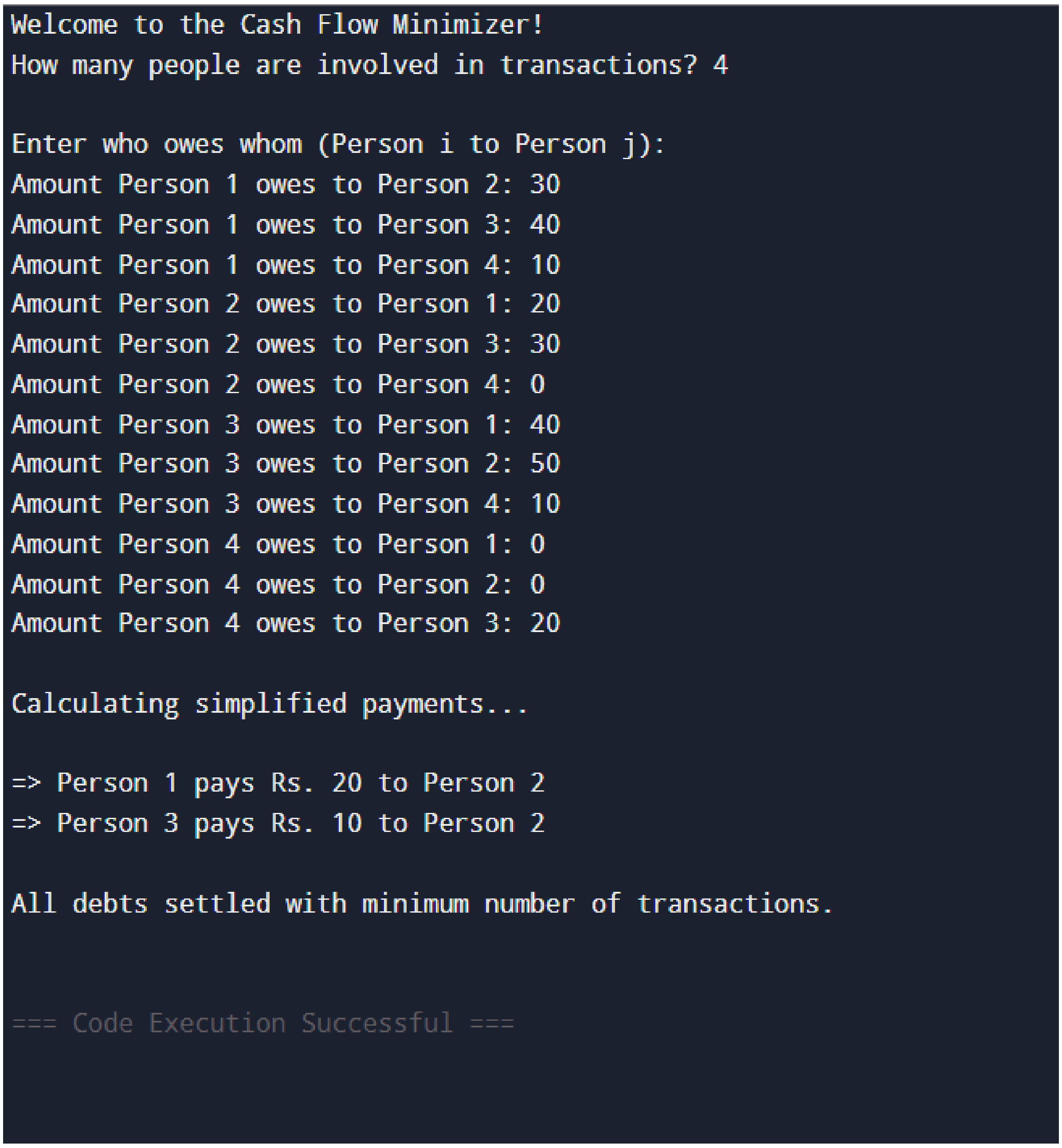
Base case:

if both max credit and debit are zero, debts are settled.

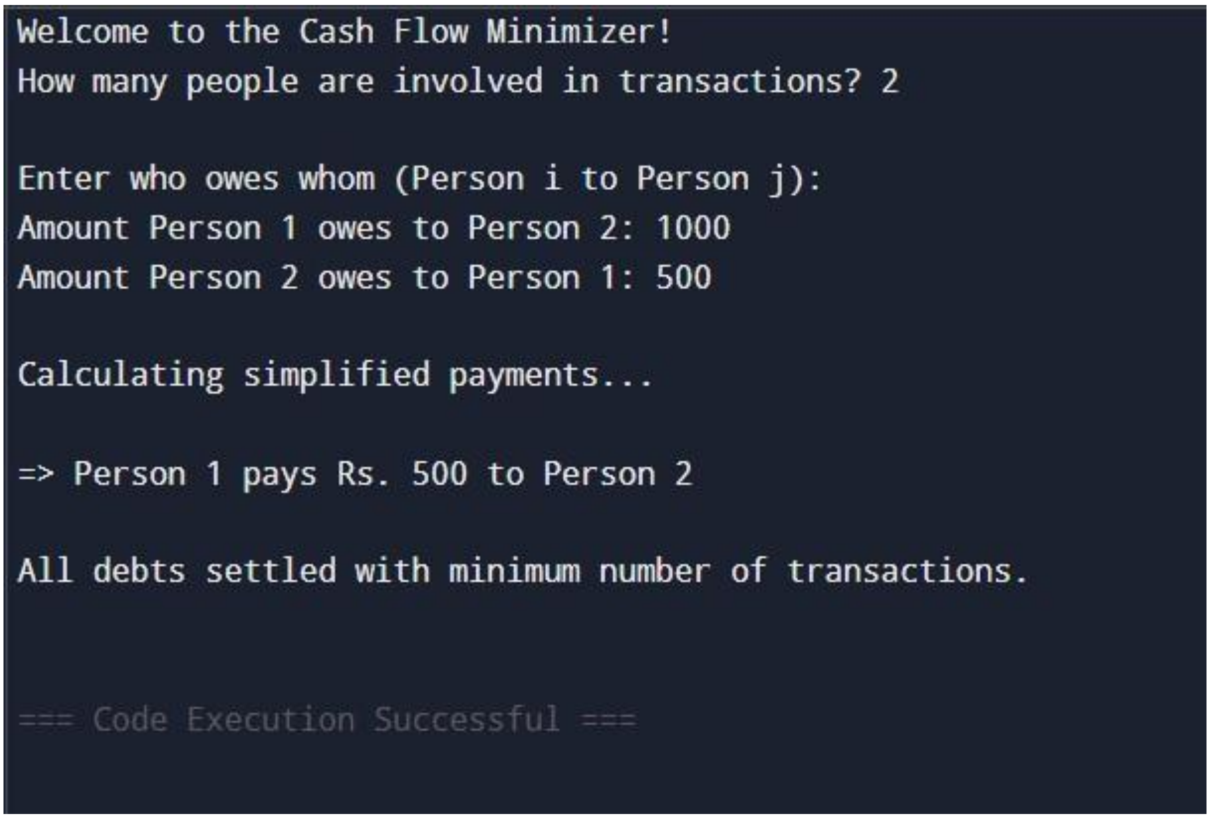
Transfer the minimum of the credit or absolute debit between the two.

Recursively call the function again to handle the next transaction.

Execution Screenshots:



```
Welcome to the Cash Flow Minimizer!
How many people are involved in transactions? 4

Enter who owes whom (Person i to Person j):
Amount Person 1 owes to Person 2: 30
Amount Person 1 owes to Person 3: 40
Amount Person 1 owes to Person 4: 10
Amount Person 2 owes to Person 1: 20
Amount Person 2 owes to Person 3: 30
Amount Person 2 owes to Person 4: 0
Amount Person 3 owes to Person 1: 40
Amount Person 3 owes to Person 2: 50
Amount Person 3 owes to Person 4: 10
Amount Person 4 owes to Person 1: 0
Amount Person 4 owes to Person 2: 0
Amount Person 4 owes to Person 3: 20

Calculating simplified payments...

=> Person 1 pays Rs. 20 to Person 2
=> Person 3 pays Rs. 10 to Person 2

All debts settled with minimum number of transactions.


=== Code Execution Successful ===
```



```
Welcome to the Cash Flow Minimizer!
How many people are involved in transactions? 2

Enter who owes whom (Person i to Person j):
Amount Person 1 owes to Person 2: 1000
Amount Person 2 owes to Person 1: 500

Calculating simplified payments...

=> Person 1 pays Rs. 500 to Person 2

All debts settled with minimum number of transactions.


=== Code Execution Successful ===
```

Github link:
https://github.com/paramvir-12/Minor-project
https://github.com/JAIDEEPx21/minor-project.git
https://github.com/shivam-9188/Minor-project

REFRENCE LINK –

GeeksforGeeks | A computer science portal for geeks

Conclusion:

This project successfully implements a **cash flow minimizer** that reduces the number of financial transactions among a group using efficient data structures. This program efficiently minimizes the number of transactions to settle debts using a greedy approach and simple data structures. It effectively models a real-world financial problem and showcases how algorithms can simplify human interactions involving money.

By applying core principles of DSA, this project highlights the power of algorithmic optimization in solving everyday problems, making it a valuable asset for fintech developers, accountants, and data science learners.