Bank Management System in C

A comprehensive banking application demonstrating advanced data structures including hash tables, stacks, and queues for efficient account management and transaction processing.

Rohan Suresh - PES1UG24CS383
S Mayur - PES1UG24CS391
Satvik L - PES1UG24CS423

System Architecture Overview

Hash Table Storage

Accounts stored using hash table with chaining for O(1) average lookup time and collision handling.

Transaction History

Stack-based structure maintains chronological transaction records with timestamp tracking for each account.

Service Requests

Queue implementation processes loan applications and service requests in FIFO order ensuring fairness.

Core Data Structures

Account Structure

- Account number (long integer)
- Customer name (string)
- Current balance (double)
- Transaction history stack
- Loan details and EMI tracker

Bank Structure

- Hash table array (SIZE 100)
- Linked list chains for collisions
- Service request queue
- Memory management utilities

Hash Table Implementation

01

Hash Function

Uses modulo operation (account_number % 100) to distribute accounts across 100 buckets for balanced distribution.

02

Collision Resolution

Employs separate chaining with linked lists, allowing multiple accounts per bucket without data loss.

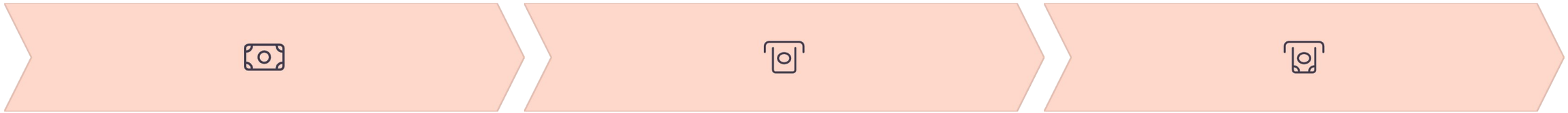
03

Account Lookup

Traverses the chain at hashed index to find matching account number with efficient linear search.

The hash table provides average O(1) insertion and lookup complexity, making account operations highly efficient even with thousands of accounts.

Transaction Operations



Deposit

Adds funds to account balance and logs transaction to history stack with timestamp.

Withdraw

Validates sufficient balance, deducts amount, and records withdrawal in transaction history.

Transfer

Executes inter-account transfer with dual logging for both source and destination accounts.

Transaction History Stack

LIFO Architecture

Each account maintains a transaction history using a stack data structure that stores:

- Transaction type (deposit, withdrawal, transfer, loan)
- Transaction amount (positive or negative)
- Unix timestamp for precise tracking

The most recent transaction appears first when viewing history, providing intuitive chronological order.

Loan Processing System

Application

Customer submits loan request with desired amount through menu interface.

Validation

Admin processes request, checking account balance meets 10% threshold requirement.

1

2

3

4

Queue Entry

Request added to service queue with account number and loan amount details.

Approval

If approved, loan disbursed with 10% interest calculated over 12-month EMI schedule.

Key Features Implemented

Interest Calculation

Simple interest computation based on principal, rate, and time period with automatic deposit to account balance.

Mini Statement

Complete transaction history display with formatted timestamps and running balance tracking for transparency.

Balance Validation

Comprehensive checks prevent overdrafts and ensure sufficient funds before executing withdrawals or transfers.

Memory Management

Proper cleanup routines free all dynamically allocated structures including accounts, nodes, and transaction stacks.

System Statistics

100

Hash Buckets

Array size providing balanced distribution for account storage and retrieval.

10

Core Operations

Complete banking functions from account creation to loan processing.

3

Data Structures

Hash table, stack, and queue implementations demonstrating algorithmic efficiency.

Technical Highlights

Efficient Algorithms

O(1) average case for account operations using hash table with proper collision handling through chaining.

Robust Design

Input validation, error handling, and memory safety ensure reliable operation under various conditions.

Practical Application

Real-world banking concepts including loans, EMI calculations, and interest computation demonstrate industry relevance.

This implementation showcases fundamental data structures and algorithms essential for systems programming, providing a solid foundation for understanding production-grade software architecture.