NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA
NH 66, Srinivas Nagar, Surathkal, Mangaluru, Karnataka 575025, India

IT206 Odd 2024-25 Course Project Report

Bank Management System

Group Number: 12
Name of the Group Members
K.Pavan Teja (231IT030)
P.Pavan Kumar (231IT046)

<<DO NOT DELETE THIS TABLE FORMAT>>
<<The following points are mandatory; you may add additional points if needed>>
<<Upload the word file in the Moodle, not the PDF>>
<<Each of the Group members in the same group needs to submit the SAME report in Moodle, anyone will be evaluated>>

What is the Application? *(150-200 words approx.)*
The Bank Management System is designed to simulate core functionalities of modern banking systems, allowing users to securely manage their accounts, perform transactions, and maintain a transaction history. The application supports key operations such as creating accounts, depositing or withdrawing money, transferring funds, and viewing transaction histories. It emphasizes data security and efficiency, requiring authentication via unique tokens and PINs for account access. The inclusion of transaction history enhances accountability and transparency, enabling users to track their financial activity. By integrating robust data structures like stacks (or unordered maps for account management), the system handles multiple accounts and transactions effectively, ensuring both scalability and quick access times. This project replicates real-world banking needs, making it a valuable tool for learning and developing efficient banking applications for educational and professional environments.
What is a Possible Solution? *(Minimum 150 words or more, include figures and tables if needed)*

NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA
NH 66, Srinivas Nagar, Surathkal, Mangaluru, Karnataka 575025, India

IT206 Odd 2024-25 Course Project Report

Feature	Description
Account Creation	Unique tokens & account numbers generated dynamically.
Secure Login	Login using token & PIN for security.
Deposit/Withdraw	Add or withdraw funds with automatic balance updates.
Fund Transfer	Transfer money with sender/receiver validation.
Transaction History	Track recent transactions with a stack.
Data Persistence	Save details & history to a file for long-term storage.
User-Friendly Menu	Simple, easy-to-navigate interface for banking operations.
Efficiency	Fast lookups & memory usage with stacks and maps.

The proposed solution involves designing a robust system that efficiently manages accounts and transactions. The system uses an unordered map to associate unique tokens with account objects, ensuring O(1) average time complexity for account lookups. Each account maintains a stack of transactions to provide an accessible, chronological record of activities. Users authenticate themselves using tokens and PINs, which secures sensitive information. A key feature is file persistence, allowing account details and transaction histories to be stored for future retrieval.

The application includes features such as:

- **Account creation** with dynamic generation of unique tokens and account numbers based on current time.
- **Secure access** using tokens and PINs for authentication.
- **Deposit and Withdrawal**: Users can securely add or withdraw funds with balances automatically updated.
- **Fund Transfer**: Inter-account money transfers with sender and receiver validation.
- **History tracking**: a stack to store recent transactions for each account.
- **Data persistence**: saving account details and transaction histories to a file for long-term storage.
- **User-Friendly Menu**: An intuitive interface allows users to choose operations such as deposits, withdrawals, and transfers. The features focus on ease of access, security, and performance.

This design ensures security, efficiency, and scalability. Using stacks for transaction management and maps for account storage optimizes both memory usage and processing speed, addressing the challenges of managing large datasets in banking systems.

NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA
NH 66, Srinivas Nagar, Surathkal, Mangaluru, Karnataka 575025, India

IT206 Odd 2024-25 Course Project Report

Which Data Structure to Use for the Solution (Details of the Data Structure)
(Minimum 150 words or more, include figures and tables if needed)

Feature	Unordered Map	Stack
Time Complexity	O(1) for lookup/insert	O(1) for push/pop
Access Order	No order (unordered)	Last-In-First-Out
Memory Usage	Efficient, low overhead	Low overhead
Best For	Fast account lookups	Tracking recent transactions
Use in Banking	Quick account access	Manage transaction history

1. **Unordered Map (unordered_map<int, Account>)**:
   - Accounts are stored using unordered_map<int, Account>, where the key is a unique token, ensuring average **O(1)** access time. This structure allows the system to scale efficiently, even with a large user base. Unlike traditional arrays or linked lists, unordered_map provides rapid lookups and insertions, essential for handling account data.
   - **Advantages**:
     - Fast lookups with an average time complexity of **O(1)**.
     - Handles a large number of accounts efficiently.
2. **Stack for Transaction History**:
   - Transaction histories are maintained in a **stack** for each account. A stack follows **Last-In-First-Out (LIFO)** ordering, making it ideal for accessing the most recent transactions quickly. With constant-time push and pop operations, the stack is lightweight yet effective for chronological transaction management. .Each account maintains a stack of transactions (TransactionStack).
   - **Why Stack?**
     - Stacks provide **Last-In-First-Out (LIFO)** ordering, which makes it easy to retrieve the most recent transactions.
     - Adding a transaction (push) or removing/displaying the most recent

transaction (pop/top) is **O(1)**.

Key Features of These Data Structures:

- **Unordered Map** enables quick account access by token.
- **Stack** simplifies the management of transaction history, with efficient operations and minimal memory overhead compared to alternatives like linked lists

Justify Why This is the Best Data Structure for this Application.
(Minimum 150 words or more; it is better to list the justifications point-wise, including figures and tables if needed)

Feature	Map	Unordered Map
Order of Elements	Stores elements in a sorted order (by key).	Stores elements in no particular order.
Underlying Data Structure	Typically implemented as a Red-Black Tree (balanced BST).	Typically implemented as a Hash Table.
Search Time Complexity	O(log n)	O(1) on average (amortized time complexity).
Insertion Time Complexity	O(log n)	O(1) on average (amortized time complexity).
Memory Usage	Uses more memory due to the tree structure.	Generally uses less memory due to the hash table structure.
Performance Consistency	Performance is consistent, as it is sorted.	Performance can vary, depending on hash collisions.

Feature	Stack	Queue	Linked List
Data Structure Type	LIFO (Last In, First Out)	FIFO (First In, First Out)	Linear structure made up of nodes linked by pointers.
Access to Elements	Only the top element is accessible.	Only the front element is accessible.	Any element can be accessed via traversal.
Order of Elements	Last inserted element is the first to be removed.	First inserted element is the first to be removed.	No strict order, elements are linked sequentially.
Memory Allocation	Can be implemented using arrays or linked lists.	Can be implemented using arrays or linked lists.	Uses dynamic memory allocation with pointers.
Use Cases	Function call stack, backtracking, undo operations.	Task scheduling, buffering, message queues.	Dynamic data structures, implementation of stacks/queues.

Unordered Map vs Maps:

unordered_map allows near-instantaneous lookups for accounts, ensuring quick access times even with large datasets. unordered_map offers **O(1)** average lookup and insertion times, while map has **O(log n)** due to its tree-based implementation. For banking, where frequent account lookups are required, unordered_map ensures faster performance.

Stack vs Linked List and Queues:

Stacks provide **Last-In-First-Out (LIFO)** ordering, which makes it easy to retrieve the most recent transactions.Adding a transaction (push) or removing/displaying the most recent transaction (pop/top) is **O(1)**.

- **Stack vs Linked List**: Linked lists can dynamically grow and manage transactions efficiently, but each node incurs extra memory overhead for pointers. Stacks, implemented as dynamic arrays, save this memory while maintaining constant-time access to the most recent element.
- **Stack vs Queue**: Queues (FIFO) prioritize the earliest element, unsuitable for transaction histories that require access to the latest operations. Stacks (LIFO) handle

this better.

Implementation Details (Experimental Setup)
(150-200 words approx.)

Main Functions:

1. **createAccount()**: Generates unique tokens and account numbers using the current time, ensuring no collisions. It validates user input and initializes accounts.
2. **accessAccount()**: Authenticates users by verifying their token and PIN, preventing unauthorized access.
3. **deposit()**: Adds funds to the user's balance and logs the transaction in the stack.
4. **withdraw()**: Deducts money after validating sufficient balance, logging the transaction in the stack.
5. **transfer()**: Validates sender and receiver details, then processes the fund transfer while logging respective transactions.
6. **saveAccountToFile()**: Persists account and transaction details, ensuring data integrity across sessions.
7. **TransactionStack**: Handles pushing, displaying, and saving transactions to maintain order and accessibility.

The Bank Management System is implemented in C++ and utilizes key data structures such as unordered_map and stack to optimize performance. The code is modular and designed to simulate core banking operations, including account creation, deposits, withdrawals, transfers, and transaction history management.

1. **Language and Libraries**:
   - The system is written in **C++** and makes use of the <unordered_map>, <stack>, <string>, <fstream>, and <chrono> libraries. These libraries handle efficient data storage, transaction management, file I/O, and time-based token/account generation.
2. **Account Management**:
   - Accounts are represented by the Account class, which encapsulates private attributes such as the account holder's name, balance, PIN, and transaction

history (TransactionStack). Accounts are stored in an unordered_map<int, Account> for constant-time access, with unique tokens as keys.

3. **Transaction Management**:
   - Transactions (e.g., deposits, withdrawals, transfers) are stored in a stack (TransactionStack) associated with each account. This provides efficient Last-In-First-Out (LIFO) access for recent transactions.
4. **File Persistence**:
   - All account details and transaction histories are saved in a text file using ofstream for persistence across program sessions. The file is opened in append mode (ios::app), ensuring that new records do not overwrite existing data.
5. **User Interface**:
   - A **menu-driven approach** implemented with a switch statement in the main() function allows users to create accounts, perform transactions, view details, and transfer money.
6. **Testing and Validation**:
   - The system uses time-based tokens and account numbers to ensure uniqueness. PIN verification ensures secure access, and sufficient balance checks prevent unauthorized transactions.

Complexity Analysis
(100 words approx.)

The system uses **unordered_map** and **stack** to manage accounts and transactions effectively. **Unordered_map<int, Account>** stores account data, providing constant-time lookups and insertions, essential for managing a large number of users securely. Each account maintains a **stack** for transaction history, ensuring efficient access to the most recent transactions. The LIFO nature of stacks is ideal for chronological transaction tracking, with constant-time push and pop operations. The main() function ties these together, offering options like creating accounts, depositing, withdrawing, and transferring funds. It also handles file management, using ofstream to save data persistently. The program ensures security through token and PIN validation, while its modular design keeps functionalities separated for clarity and efficiency.

Observations and Conclusions

IT206 Odd 2024-25 Course Project Report

(100 words approx.)

Observations :
The secure authentication system ensures account safety by utilizing a unique token and PIN (based on the current date and time, streamlining the account creation process) for access. Efficient data storage is implemented through a stack structure, allowing swift retrieval of recent transactions. Account details and transaction histories are persistently saved to a file, guaranteeing data retention across sessions. Comprehensive banking features are provided, including deposits, withdrawals, and money transfers, ensuring user convenience. The system uses unordered map for constant-time account lookups and stack for efficient transaction management, optimizing both speed and memory usage. The system successfully handles essential banking functionalities while maintaining security and efficiency. These features, combined with modular design, make the system robust, scalable, and user-friendly.

Conclusion:
The Bank Management System effectively simulates core banking functionalities with a focus on security and data management. The integration of these data structures makes the application robust and capable of scaling for real-world usage, aligning with modern banking system requirements.