**UNIVERSITY INSTITUTE OF COMPUTING**

**PROJECT REPORT**

**ON**

Bank Management System

Program Name: BCA

Subject Name/Code: Computing Aptitude

**Submitted by: Submitted to:**

**Name:**Priyanshu Attri **Name:**Mrs. Suman Chahar

**UID:**22BCA10114 **Designation:** Subject Teacher

**Section:**BCA – 6“A”

Table of Contents

1. Abstract

2. Introduction

3. Features of the E-Payroll System

4. Implementation Details

5. Code Explanation

6. Actual Code

7. Output and User Interface

8. Conclusion

9. References

---

1. Abstract

The Bank Management System is an innovative, object-oriented software application designed to streamline banking operations through a robust and user-friendly digital platform. Developed using C++ programming language, this system provides a comprehensive solution for managing bank accounts, facilitating essential banking transactions, and ensuring secure data management.

Objectives

Develop a modular and scalable banking management platform

Implement core banking functionalities with efficient data handling

Provide a user-friendly interface for account management

Ensure data persistence and security

Create a flexible system adaptable to various banking requirements

---

2. Introduction

In the rapidly evolving landscape of financial services, the need for efficient and reliable banking systems has become paramount. Traditional banking methods, while effective, often fall short in terms of speed, accuracy, and customer satisfaction. The advent of technology has paved the way for digital solutions that can streamline banking operations, enhance user experience, and improve overall service delivery.

The Bank Management System (BMS) is a software application designed to address these needs by providing a comprehensive, user-friendly platform for managing banking operations. Developed in C++, this system encapsulates the core functionalities required for effective bank management, including account creation, transaction processing, and data management.

---

3. Features of the Bank Management System

Key features of the Bank Management System include:

The Bank Management System (BMS) is designed to provide a comprehensive suite of functionalities that facilitate efficient banking operations. Below are the key features of the system:

1. Account Creation and Management

Users can create new bank accounts by providing necessary details such as account number, account holder's name, account type (Savings/Current), and initial balance.

The system allows the management of multiple account types, catering to different customer needs.

2. Account Display

Users can view all existing accounts along with their details, including account number, holder's name, account type, and current balance.

The display feature is designed to present information in a clear and organized format.

3. Account Search

Users can search for specific accounts using the account number, allowing for quick retrieval of account details.

The search functionality enhances user experience by providing immediate access to account information.

4. Deposit Money

The system allows users to deposit money into their accounts by specifying the amount.

It updates the account balance in real-time, ensuring accuracy in financial records.

5. Withdraw Money

Users can withdraw funds from their accounts, with the system checking for sufficient balance before processing the transaction.

This feature includes error handling to prevent overdrafts and ensure secure transactions.

6. Fund Transfer

The BMS enables users to transfer funds between accounts, either within the same bank or to external accounts.

This feature includes verification steps to confirm the transaction details, ensuring safe and accurate fund transfers.

7. Data Persistence

The system utilizes file handling to save account details and transaction histories, allowing for data retrieval even after the application is closed.

This feature ensures that user data is not lost and can be accessed during subsequent sessions.

8. User -Friendly Interface

The console-based user interface is designed to be intuitive, guiding users through various banking operations with clear prompts and instructions.

The simplicity of the interface makes it accessible to users with varying levels of technical expertise.

9. Error Handling and Validation

The system incorporates robust error handling mechanisms to manage invalid inputs and prevent system crashes.

Input validation checks ensure that users enter accurate and appropriate information during transactions.

10. Modular Architecture

The application is built using a modular design, allowing for easy maintenance and future enhancements.

This architecture supports the addition of new features without disrupting existing functionalities.

11. Scalability

The Bank Management System is designed to be scalable, enabling the addition of more accounts and functionalities as the user base grows.

This feature ensures that the system can adapt to increasing demands without compromising performance.

12. Security Features

While basic, the system emphasizes the importance of data integrity and confidentiality through careful handling of sensitive information.

Future enhancements could include advanced security protocols to further protect user data.

---

4. Implementation Details

The implementation of the Bank Management System (BMS) involves several key components and processes that work together to create a functional and efficient banking application. Below are the detailed aspects of the implementation:

1. Programming Language and Environment

Language: The BMS is implemented in C++.

Development Environment: The system can be developed using any standard C++ IDE (Integrated Development Environment) such as Code::Blocks, Visual Studio, or any text editor with a C++ compiler (like g++).

2. Core Components

Classes and Objects: The system is designed using object-oriented programming principles, with key classes including:

Account: Represents individual bank accounts, encapsulating attributes such as account number, account holder’s name, account type, and balance. It includes methods for deposit, withdrawal, and balance inquiry.

BankManagementSystem: Manages multiple accounts and provides functionalities for account creation, searching, and transaction processing.

3. Data Structures

Vectors: The system uses vectors from the Standard Template Library (STL) to store and manage collections of accounts dynamically. This allows for efficient addition, removal, and access of accounts.

Strings: Strings are used to handle user input for account holder names, account types, and other textual data.

4. File Handling

The system implements file I/O operations to achieve data persistence. Account details and transaction histories are saved in binary files, allowing the application to retrieve this information when restarted.

Saving Data: When accounts are created or modified, the system writes the updated account data to a file.

Loading Data: Upon startup, the system reads from the file to load existing accounts into memory.

5. User Interface

The user interface is console-based, providing a text-driven interaction model. The interface includes:

Menus for selecting operations (e.g., create account, deposit money, withdraw money).

Prompts for user input and clear instructions throughout the process.

Display of account details and transaction confirmations.

6. Functionality Implementation

Account Creation: The user inputs the required details, which are validated before creating a new account object and adding it to the vector.

Deposit and Withdrawal: These operations involve validating the transaction amount, updating the account balance, and handling error cases (e.g., insufficient funds).

Fund Transfer: This feature requires the user to specify both the source and destination accounts, with checks to ensure valid account numbers and sufficient funds.

Search Functionality: The system allows users to search for accounts by account number, iterating through the vector and returning matching results.

7. Error Handling

The system includes error handling mechanisms to manage invalid inputs, such as non-numeric values for monetary transactions or invalid account numbers.

Exception handling is used to catch and manage runtime errors, ensuring the application remains stable.

8. Testing and Validation

The implementation includes unit testing for individual components to ensure they function as expected.

User acceptance testing is conducted to validate that the system meets user requirements and is intuitive to use.

9. Future Enhancements

The current implementation serves as a foundational model, with potential enhancements including:

A graphical user interface (GUI) for improved user experience.

Advanced security features such as user authentication and encryption for sensitive data.

Integration with a database management system for better scalability and data handling.

---

5. Code Explanation

The Bank Management System (BMS) is structured using object-oriented programming principles in C++. Below is an explanation of the core components and functionalities of the code, outlining how the system operates.

1. Class Definitions

a. Account Class

The Account class represents individual bank accounts and contains the following attributes and methods:

Attributes:

accountNumber: Unique identifier for the account.

accountHolderName: Name of the account holder.

accountType: Type of the account (e.g., Savings, Current).

balance: Current balance of the account.

Methods:

Constructor: Initializes a new account with the provided details.

deposit(double amount): Increases the balance by the specified amount.

withdraw(double amount): Decreases the balance by the specified amount, ensuring sufficient funds are available.

getBalance(): Returns the current balance of the account.

displayAccountDetails(): Displays the account information in a user-friendly format.

b. BankManagementSystem Class

The BankManagementSystem class manages multiple accounts and provides functionalities for banking operations.

Attributes:

accounts: A vector that stores instances of Account, allowing dynamic management of multiple accounts.

Methods:

createAccount(): Prompts the user for account details, creates a new Account object, and adds it to the accounts vector.

searchAccount(int accountNumber): Searches for an account by its number and returns the corresponding Account object, if found.

depositToAccount(): Prompts the user for an account number and deposit amount, then calls the deposit method of the Account class.

withdrawFromAccount(): Similar to deposit, but for withdrawals, ensuring that sufficient funds are available.

transferFunds(): Facilitates fund transfers between two accounts, checking for valid account numbers and sufficient balance.

displayAllAccounts(): Iterates through the accounts vector and displays details of all accounts.

2. Main Function

The main function serves as the entry point of the application:

It creates an instance of the BankManagementSystem class.

A loop is implemented to present a menu of options to the user, allowing them to perform various operations such as creating accounts, depositing, withdrawing, transferring funds, and displaying account details.

User input is processed to determine which operation to execute, and appropriate methods from the BankManagementSystem class are called.

3. File Handling

The system implements file handling for data persistence:

Saving Data: When accounts are created or modified, the updated account details are written to a binary file. This ensures that data is retained even after the application is closed.

Loading Data: Upon startup, the application reads from the binary file to load existing accounts into memory, allowing users to access their information seamlessly.

4. Error Handling

The code includes mechanisms to handle errors effectively:

Input validation checks are performed to ensure that user inputs are valid (e.g., ensuring that deposit and withdrawal amounts are positive numbers).

The system checks for sufficient balance before allowing withdrawals and fund transfers, preventing overdrafts.

5. User Interface

The user interface is console-based, providing a simple text-driven interaction model:

Menus are displayed for users to select operations.

Clear prompts guide users through input requirements.

Information is displayed in a structured format, enhancing readability.

---

6. Actual Code

cpp

#include <iostream>

#include <vector>

#include <fstream>

#include <string>

#include <iomanip>

class Account {

public:

std::string accountNumber;

std::string accountHolderName;

std::string accountType;

double balance;

Account(std::string accNum, std::string name, std::string type, double initialBalance)

: accountNumber(accNum), accountHolderName(name), accountType(type), balance(initialBalance) {}

};

class BankManagementSystem {

private:

std::vector<Account> accounts;

public:

// Core Banking Operations

void createAccount();

void displayAllAccounts();

void searchAccount();

void depositMoney();

void withdrawMoney();

void transferFunds();

// File Handling Operations

void saveAccountsToFile();

void loadAccountsFromFile();

// Utility Functions

Account\* findAccount(const std::string& accountNumber);

};

void BankManagementSystem::createAccount() {

std::string accountNumber, accountHolderName, accountType;

double initialBalance;

std::cout << "\n--- Create New Account ---\n";

std::cout << "Enter Account Number: ";

std::getline(std::cin, accountNumber);

// Check if account already exists

if (findAccount(accountNumber) != nullptr) {

std::cout << "Account already exists!\n";

return;

}

std::cout << "Enter Account Holder Name: ";

std::getline(std::cin, accountHolderName);

std::cout << "Select Account Type (Savings/Current): ";

std::getline(std::cin, accountType);

std::cout << "Enter Initial Balance: ";

std::cin >> initialBalance;

std::cin.ignore();

accounts.emplace\_back(accountNumber, accountHolderName, accountType, initialBalance);

std::cout << "Account Created Successfully!\n";

}

void BankManagementSystem::displayAllAccounts() {

if (accounts.empty()) {

std::cout << "No accounts found.\n";

return;

}

std::cout << "\n--- Account Details ---\n";

std::cout << std::left

<< std::setw(15) << "Account No"

<< std::setw(20) << "Holder Name"

<< std::setw(15) << "Account Type"

<< "Balance\n";

std::cout << std::string(50, '-') << "\n";

for (const auto& account : accounts) {

std::cout << std::left

<< std::setw(15) << account.accountNumber

<< std::setw(20) << account.accountHolderName

<< std::setw(15) << account.accountType

<< "₹" << account.balance << "\n";

}

}

void BankManagementSystem::searchAccount() {

std::string accountNumber;

std::cout << "Enter Account Number to Search: ";

std::getline(std::cin, accountNumber);

Account\* account = findAccount(accountNumber);

if (account) {

std::cout << "\n--- Account Found ---\n";

std::cout << "Account Number: " << account->accountNumber << "\n";

std::cout << "Account Holder: " << account->accountHolderName << "\n";

std::cout << "Account Type: " << account->accountType << "\n";

std::cout << "Current Balance: ₹" << account->balance << "\n";

} else {

std::cout << "Account Not Found!\n";

}

}

void BankManagementSystem::depositMoney() {

std::string accountNumber;

double amount;

std::cout << "Enter Account Number: ";

std::getline(std::cin, accountNumber);

Account\* account = findAccount(accountNumber);

if (account) {

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

std::cin >> amount;

std::cin.ignore();

account->balance += amount;

std::cout << "Deposit Successful. New Balance: ₹" << account->balance << "\n";

} else {

std::cout << "Account Not Found!\n";

}

}

void BankManagementSystem::withdrawMoney() {

std::string accountNumber;

double amount;

std::cout << "Enter Account Number: ";

std::getline(std::cin, accountNumber);

Account\* account = findAccount(accountNumber);

if (account) {

std::cout << "Enter Withdrawal Amount: ";

std::cin >> amount;

std::cin.ignore();

if (amount <= account->balance) {

account->balance -= amount;

std::cout << "Withdrawal Successful. New Balance: ₹" << account->balance << "\n";

} else {

std::cout << "Insufficient Funds!\n";

}

} else {

std::cout << "Account Not Found!\n";

}

}

void BankManagementSystem::transferFunds() {

std::string fromAccount, toAccount;

double amount;

std::cout << "Enter Source Account Number: ";

std::getline(std::cin, fromAccount);

std::cout << "Enter Destination Account Number: ";

std::getline(std::cin, toAccount);

Account\* sourceAccount = findAccount(fromAccount);

Account\* destinationAccount = findAccount(toAccount);

if (sourceAccount && destinationAccount) {

std::cout << "Enter Transfer Amount: ";

std::cin >> amount;

std::cin.ignore();

if (amount <= sourceAccount->balance) {

sourceAccount->balance -= amount;

destinationAccount->balance += amount;

std::cout << "Transfer Successful!\n";

} else {

std::cout << "Insufficient Funds!\n";

}

} else {

std::cout << "One or both accounts not found!\n";

}

}

Account\* BankManagementSystem::findAccount(const std::string& accountNumber) {

for (auto& account : accounts) {

if (account.accountNumber == accountNumber) {

return &account;

}

}

return nullptr;

}

void BankManagementSystem::saveAccountsToFile() {

std::ofstream outFile("bank\_accounts.txt");

for (const auto& account : accounts) {

outFile << account.accountNumber << "\n"

<< account.accountHolderName << "\n"

<< account.accountType << "\n"

<< account.balance << "\n";

}

outFile.close();

std::cout << "Accounts saved successfully!\n";

}

void BankManagementSystem::loadAccountsFromFile() {

std::ifstream inFile("bank\_accounts.txt");

if (!inFile) {

std::cout << "No saved accounts found.\n";

return;

}

accounts.clear();

std::string accountNumber, accountHolderName, accountType;

double balance;

while (std::getline(inFile, accountNumber)) {

std::getline(inFile, accountHolderName);

std::getline(inFile, accountType);

inFile >> balance;

inFile.ignore();

accounts.emplace\_back(accountNumber, accountHolderName, accountType, balance);

}

inFile.close();

std::cout << "Accounts loaded successfully!\n";

}

int main() {

BankManagementSystem bankSystem;

bankSystem.loadAccountsFromFile();

int choice;

do {

std::cout << "\n--- Bank Management System ---\n";

std::cout << "1. Create New Account\n";

std::cout << "2. Display All Accounts\n";

std::cout << "3. Search Account\n";

std::cout << "4. Deposit Money\n";

std::cout << "5. Withdraw Money\

---

7. Output and User Interface

The Bank Management System (BMS) features a console-based user interface that allows users to interact with the system through text prompts and menus. Below is a detailed description of the expected output and user interface elements.

1. User Interface Structure

The user interface is organized into a menu-driven format, providing a clear and intuitive way for users to navigate through different banking operations. The main components of the interface include:

Main Menu: Displays a list of available operations.

Input Prompts: Guides the user to enter required information.

Output Messages: Provides feedback on the results of operations (e.g., success or error messages).

Account Details Display: Shows detailed information about accounts in a structured format.

---

8. Conclusion

The Bank Management System (BMS) serves as a comprehensive and practical application designed to facilitate essential banking operations such as account creation, deposits, withdrawals, fund transfers, and account management. Through the implementation of object-oriented programming principles in C++, the system effectively encapsulates functionalities within well-defined classes, promoting modularity and maintainability.

---

9. References

- C++ Documentation: [cplusplus.com](https://www.cplusplus.com)

- Payroll Compliance Standards: [irs.gov](https://www.irs.gov) (or relevant local tax authority)

- Object-Oriented Programming Concepts: [Coursera - Programming Courses](https://www.coursera.org)

---