Bank Management System

A Project Report

***Submitted by:***

**Mukesh Shrivastva (21401014)**

**Rohit Kumar (21401027)**

in partial fulfillment of the award of the degree

of

# BACHELOR OF COMPUTER APPLICATIONS

at



**SARDAR VALLABHBHAI PATEL COLLEGE (S.V.P)**

**BHABUA (KAIMUR)**

# TABLE OF CONTENTS

## Chapter 1: Project Profile

1.1 Project Definition

1.2 Scope & Objective of Project

## Chapter 2: System Study & Problem Formulation

2.1 Data Analysis

2.2 Proposed System

2.3 Advantages of Proposed System

2.4 System Requirements

2.5 Object-Oriented Analysis

## Chapter 3: Project Plan

3.1 Programming Languages & Development Tools

3.2 Reuse of Existing Software Components

## Chapter 4: Detailed Design

4.1 Design Strategy

4.2 Module Design

4.3 Interface Design

## Bibliography

#### **CHAPTER 1: PROJECT PROFILE**

* 1. **Project Definition:**

This project aims to develop a simple bank management system using C++ to simulate basic banking operations such as account creation, deposit, withdrawal, and balance inquiry.

This project targets students learning C++ and object-oriented programming concepts, as well as anyone interested in understanding the fundamentals of banking systems.

* 1. **Objectives & Scope of Project:**

**The system includes the following features:**

* Account creation with unique account numbers.
* Depositing funds into existing accounts.
* Withdrawing funds from existing accounts (with balance checks).
* Viewing account information (name, balance, etc.).
* Persistent data storage using text files.

**The objectives of this project are:**

* To demonstrate the application of C++ for creating a functional program.
* To implement basic banking operations using object-oriented programming principles.
* To utilize file handling for data persistence and data management.

#### **CHAPTER 2: SYSTEM STUDY AND PROBLEM FORMULATION**

**2.1 Data Analysis**

***Entities: Identify the key data elements and their relationships:***

**Account Attributes:** Account number (unique), Name, Father's name, CNIC, Phone number, Email, Balance.

**Customer Attributes:** Name, Father's name, CNIC, Phone number, Email.

**Transaction Attributes:** Account number, Transaction type (deposit or withdrawal), Amount, Date/Time.

**Relationships:** An Account belongs to a customer (if implemented). Transactions are associated with an Account.

**Data Storage:** Explains how data is stored (e.g., text file "data.txt").

***Example:*** Account data is stored in a text file named 'data.txt'. Each line represents an account and contains the following information separated by tabs: account number, name, father's name, CNIC, phone number, email, and balance.

**2.2 Proposed System**

**System Architecture:**

***Describes the main components (modules) of this system and their interactions:***

**User Interface Module:** Handles user input and output.

**Account Management Module:** Creates, updates, and retrieves account data.

**Transaction Processing Module:** Processes deposit and withdrawal transactions.

**Data Persistence Module:** Reads and writes data from/to files.

**2.3 Advantages of Proposed System**

**Benefits:**

***The proposed system offers the following advantages:***

**User-friendliness:** Provides a simple and intuitive interface for managing accounts.

**Data Security:** Uses file handling for storing account information, potentially with encryption with account number.

**Efficiency:** Permits quick processing of basic banking operations.

**2.4 System Requirements**

**Functional Requirements:**

***Create Account:*** The system must allow users to create new accounts with unique account numbers and store basic customer information.

***Deposit:*** The system must allow users to deposit funds into existing accounts and update the account balance accordingly.

***Withdraw:*** The system must allow users to withdraw funds from existing accounts, ensuring sufficient balance, and updating the balance.

***Check Information:*** The system must allow users to view account details (name, balance, etc.) by entering their account number.

***Data Persistence:*** The system must store account data persistently in a file for later retrieval.

**Non-Functional Requirements:**

***Security:*** Security measures implemented as data encryption, user authentication through account number.

***Usability:*** The system is easy to understand and use, with clear menus and instructions.

**2.5 Object-Oriented Analysis**

***bank class:*** Represents the overall banking system. It could contain methods for managing accounts, transactions, and data persistence.

***account class:*** Represents a bank account. It could have attributes like account number, name, balance, and methods for deposit, withdrawal, and checking balance.

**Relationships:**

account class might have a relationship with the bank class (e.g., an account might be associated with a specific bank).

transaction class: If you model transactions separately, it might have a relationship with the account class.

**Object-Oriented Principles:**

***Encapsulation:*** Explains how data and methods are hidden within classes to protect internal state.

#### **CHAPTER 3: PROJECT PLAN**

**3.1 Programming Languages & Development Tools**

**Programming Language:** The project was implemented in C++.

**Development Tools:**

***IDE:*** Visual Studio Code.

***Compiler:*** GCC/G++

**3.2 Reuse of Existing Software Components**

***Libraries:*** standard C++ libraries (like: iostream, limits, fstream, ctime, cstdlib) are used:

***iostream:*** For input-output stream.

***Limits:*** For maintain the values in a range.

***Fstream:*** For File read and write, i.e. for file handling.

#### **CHAPTER 4: STRUCTURED ANALYSIS & STRUCTURED DIAGRAM**

**4.1 Design Strategy**

***Account Management Module:*** Handles the creation, update, and retrieval of account data.

***Transaction Processing Module***: Handles deposit and withdrawal operations.

***Data Persistence Module:*** Handles reading and writing data from/to files.

***User Interface Module:*** Presents menus and prompts to the user and receives input.

***Create Account Submodule:*** Handles account creation.

***Update Account Submodule:*** Handles changes to account information.

**4.2 Module Design**

***Code Snippets:*** Include relevant code snippets to illustrate how each module functions.

**Example (Snippet from account class):**

class account {

private:

int accNo;

string name;

// ... other attributes

public:

// ... constructor, getter methods

void deposit(int amount) {

balance += amount;

// ... update transaction history (optional)

}

void withdraw(int amount) {

if (amount <= balance) {

balance -= amount;

// ... update transaction history (optional)

} else {

cout << "Insufficient balance!" << endl;

}

}};

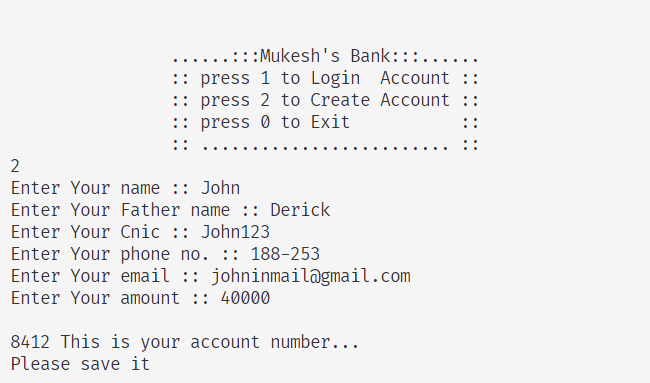
**4.3 Interface Design**

***Text-Based Menus:*** If you used a text-based menu, describe the menu structure. Provide an example of a menu with options like "Create Account", "Deposit", "Withdraw", "Check Info", "Exit".

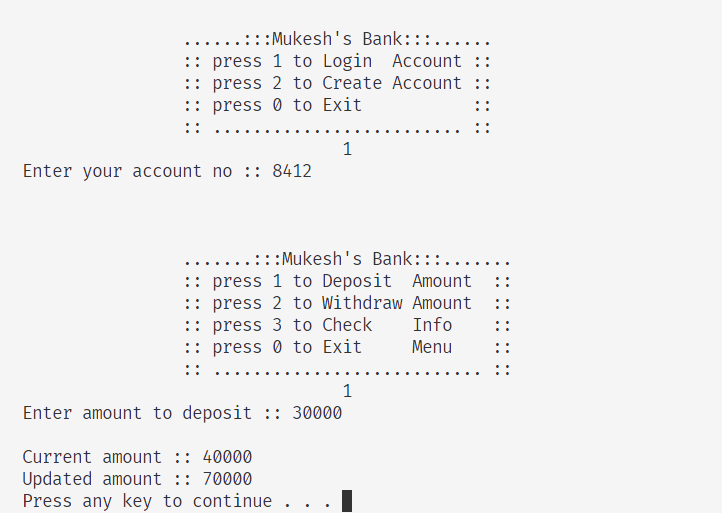
***Input Prompts:*** Describe the input prompts (e.g., "Enter account number", "Enter amount to deposit") used to guide the user.

***Output Displays:*** Describe how the system displays information to the user (e.g., confirmation messages, error messages, account details).

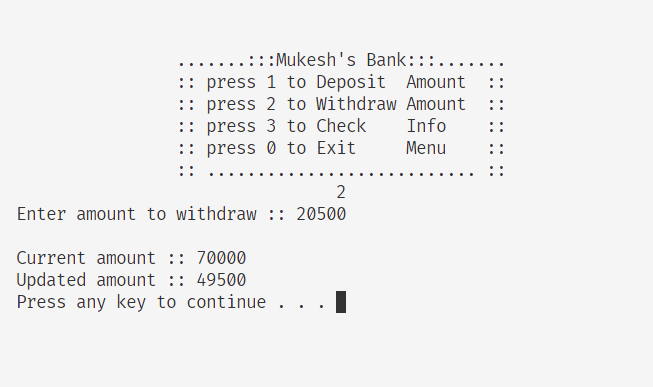
***User Interaction Flow:*** Provide a step-by-step explanation of how a user interacts with the system to perform a specific task (e.g., creating an account, making a deposit).



Account Creation



User Login & Amount Deposit



Amount Withdraw



Viewing User Info

#### **BIBLIOGRAPHY**

This report has presented a basic banking system designed to showcase fundamental programming concepts in C++. The system demonstrates basic file handling, OOP, and user interaction through a menu-driven interface. While functional, the system is limited in scope and lacks advanced features such as security, transaction history, interest calculations, and more sophisticated data storage methods. It serves as a foundation for further development, where more complex functionalities and real-world considerations can be incorporated to build a more robust and feature-rich banking application.

**List of References:**

* **The C++ Programming Language** by Bjarne Stroustrup.
* **ISO C++ Official Documentation**: isocpp.org.
* **cppreference.com**: cppreference.com.
* **C++ Core Guidelines**: isocpp.github.io.
* **C++ Primer** by Stanley B. Lippman, Josée Lajoie, and Barbara E. Moo.
* **The C++ Standard Library** by Nicolai M. Josuttis.
* **C++ Templates: The Complete Guide** by David Vandevoorde and Nicolai M. Josuttis.
* **Modern C++ Design** by Andrei Alexandrescu.
* **Accelerated C++** by Andrew Koenig and Barbara E. Moo.
* **Clean Code in C++** by Stephan Roth.
* **C++ Crash Course** by Josh Lospinoso.
* **Design Patterns** by Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides.