**BANKING RECORD SYSTEM**

***by***

**D.DANUSRI**

# (2403713820522010) First Year B.Tech(IT)



**Sri Ramakrishna Institute of Technology**

# Coimbatore - 641010 April 2025

**APPROVAL AND DECLARATION**

This project report titled **BANKING RECORD SYSTEM**

was prepared and submitted by **D.DANUSRI(Register Number: 2403713820522010)** and has been found satisfactory in terms of scope, quality and presentation as partial fulfillment of the requirement for the **Bachelor of Technology (Information Technology**) in Sri Ramakrishna Institute of Technology, Coimbatore (SRIT).

**Checked and Approved by**

**Ms.K.Malarvizhi**

**Course Coordinator / Assistant Professor**

**Department of Information Technology**

**Department of Information Technology**

**Sri Ramakrishna Institute of Technology, Coimbatore-10**

**April 2025**

# TABLE OF CONTENTS

|  |  |  |
| --- | --- | --- |
| S NO | CONTENT | PAGE NO |
| 1 | ABSTRACT | 4 |
| 2 | INTRODUCTION | 5 |
| 3 | OBJECTIVE | 6 |
| 4 | SCOPE | 6 |
| 5 | METHODOLOGY | 7 |
| 6 | ENTITY RELATIONSHIP DIAGRAM | 8 |
| 7 | SOURCE CODE | 9-13 |
| 8 | OUTPUT | 13-15 |
| 9 | FUNCTIONALITY | 16 |
| 10 | CONCLUSION | 17 |
| 11 | REFERENCE | 18 |

**BANKING RECORD SYSTEM**

# ABSTRACT

The **Banking Record System** is a console-based mini project developed using C++ that simulates fundamental banking operations. This system allows users to create and manage multiple bank accounts with features such as creating a new account, viewing account details, depositing and withdrawing money, deleting accounts, and displaying all existing accounts. It utilizes object-oriented programming concepts like classes, objects, and encapsulation to ensure organized code structure. The project also incorporates file handling to store account information persistently in a text file, enabling data retrieval even after the program is closed. Through a simple, menu-driven interface, users can interact with the system efficiently, making it a practical tool to demonstrate core C++ programming skills. This project not only enhances understanding of OOP and file I/O in C++ but also provides a foundational insight into how real-world banking systems operate at a basic level.

# CHAPTER 1

# INTRODUCTION

The **Banking Record System** is a simple yet efficient console-based application developed in C++ to demonstrate the practical implementation of object-oriented programming and file handling techniques. In today’s digital era, managing financial transactions through automated systems has become a necessity. Inspired by real-world banking operations, this project aims to replicate essential banking functionalities such as account creation, deposit, withdrawal, account deletion, and viewing account details. The system is designed with a user-friendly menu-driven interface that allows users to interact with the application in a straightforward manner. Each bank account is uniquely identified by an account number, and the associated details such as account holder’s name and balance are maintained using classes and stored in a text file for persistence.

This project plays a significant role in showcasing the fundamental concepts of **C++**, especially the use of **classes, objects, encapsulation**, and **file I/O operations**. Data is dynamically stored in memory during runtime and then saved in a file to ensure it remains available even after the program ends. The file acts as a basic database, storing records in a readable format. The use of vectors and standard input/output streams makes the code efficient and readable. Moreover, all operations are performed with input validation and simple feedback messages, ensuring a smooth user experience.

The **Banking Record System** is not only a useful learning project for students and beginners but also a stepping stone toward building more complex financial software. It emphasizes the importance of structured programming, modular design, and data integrity. While this version is a simplified model of actual banking systems, it effectively covers the core operations that lay the groundwork for more advanced features like authentication, interest calculation, and transaction history in future enhancements. Overall, this project serves as an ideal educational tool to understand how programming can be applied to solve real-life problems in the financial domain using C++

## 1.1.Objectives

* To develop a console-based banking system using C++ programming.
* To apply object-oriented programming (OOP) concepts such as classes, objects, and encapsulation.
* To implement basic banking operations like account creation, deposit, withdrawal, and deletion.
* To ensure persistent data storage using file handling techniques.
* To create a user-friendly, menu-driven interface for easy interaction.
* To manage multiple user accounts with unique account numbers and balance details.
* To enhance understanding of standard input/output operations and data validation.
* To organize account data using appropriate data structures like vectors.
* To provide a foundational system that can be expanded with advanced features in the future.
* To improve logical thinking and programming skills through real-world application development**.**

## 1.2.Scope

* The scope of the Banking Record System project is to develop a basic,console-based application that handles essential banking operations using the C++ programming language. The system allows users to create and manage multiple bank accounts, view account details, perform deposits and withdrawals, and delete accounts when needed. It emphasizes the practical implementation of object-oriented programming concepts such as classes, objects, and encapsulation, as well as the use of file handling to store account information persistently. With a simple menu-driven interface, users can easily navigate and perform transactions, making the application user-friendly and functional.
* This project is intended to provide a foundational understanding of how core banking functionalities can be implemented programmatically. While it is a simplified model compared to real-world banking systems, the system can be enhanced in the future with features like user authentication, graphical user interfaces (GUI), interest calculations, and transaction logs. The project is especially suitable for students and beginners who wish to gain hands-on experience in C++ programming and explore how real-life applications are structured and developed.

**CHAPTER 2**

# METHODOLOGY

The development of the **Banking Record System** follows a structured and step-by-step methodology to ensure clarity, modularity, and efficient implementation. The project begins with the **requirement analysis** phase, where the core functionalities of the system such as account creation, deposit, withdrawal, and deletion are identified. This helps in setting clear goals and planning the necessary features that align with the real-world needs of a basic banking system. The objective is to simulate common banking operations in a simple console environment using the C++ programming language.

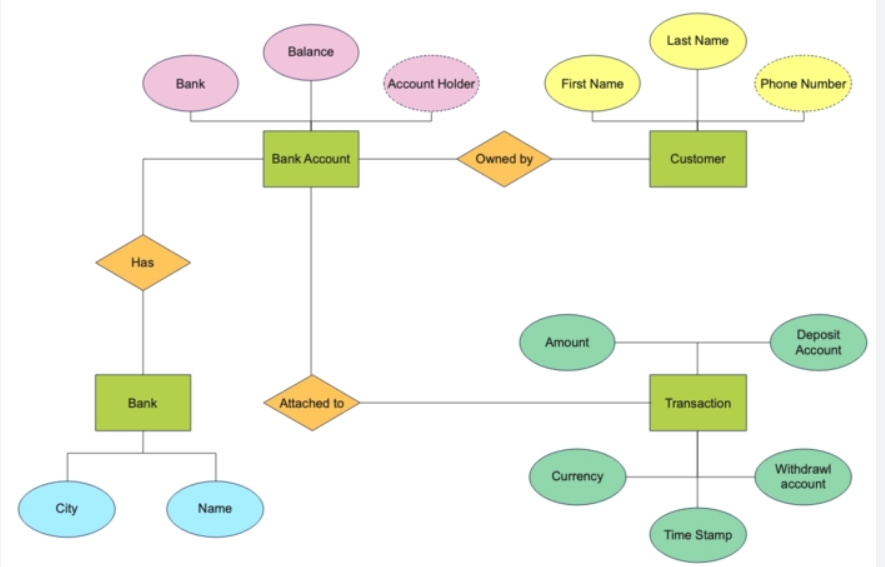
In the **design phase**, the system architecture is planned using object-oriented principles. A class named, for instance, BankAccount is designed to represent each account with data members such as account number, holder name, and balance, and member functions to perform operations like deposit, withdraw, and display details. The structure of the text file used for storing data is also defined during this phase to ensure smooth file input/output operations.

The **implementation phase** involves coding the functionalities in C++. The menu-driven interface is developed to allow users to interact with the system easily. File handling is implemented using C++ file streams to ensure that all user data is stored persistently in a text file, allowing retrieval and modification across multiple sessions. Input validation and error-handling mechanisms are incorporated to maintain the integrity of the data and provide feedback for incorrect operations.

Lastly, in the **testing and evaluation phase**, the system is tested thoroughly to check each functionality for correctness and reliability. Various use-case scenarios such as depositing insufficient funds or attempting to access a non-existent account are tested to ensure robust performance. The final output is evaluated to confirm that the system meets the original objectives and can serve as a functional prototype of a basic banking system.

## 2.1.Entity Relationship Diagram

An entity relationship diagram is also known as an entity relationship model. The key here is to graphically depict the relationship between people, objects, location, concepts or events in an IT system. It also represents the contacts in the set of entities stored in the database. In this section, an organization is referred to as an object, a data component. One of the similar entities is defined as a set of entities.



**CHAPTER – 3**

# PROGRAMMING

## 3.1.Source Code

#include <iostream>

#include <fstream>

#include <vector>

#include <iomanip>

using namespace std;

class Account {

public:

int accountNumber;

string name;

double balance;

void createAccount() {

cout << "Enter Account Number: ";

cin >> accountNumber;

cout << "Enter Name: ";

cin.ignore();

getline(cin, name);

cout << "Enter Initial Balance: ";

cin >> balance;

}

void showAccount() const {

cout << left << setw(10) << accountNumber

<< setw(20) << name

<< "Rs. " << fixed << setprecision(2) << balance << endl;

}

void deposit(double amount) {

balance += amount;

}

bool withdraw(double amount) {

if (amount > balance) {

cout << "Insufficient balance.\n";

return false;

}

balance -= amount;

return true; }

int getAccountNumber() const {

return accountNumber;

}

double getBalance() const {

return balance;

}

};

vector<Account> accounts;

void saveAccountsToFile() {

ofstream file("accounts.txt");

for (const auto& acc : accounts) {

file << acc.accountNumber << "," << acc.name << "," << acc.balance << endl;

}

file.close();

}

void loadAccountsFromFile() {

accounts.clear();

ifstream file("accounts.txt");

int accNum;

string name;

double balance;

string line;

while (getline(file, line)) {

sscanf(line.c\_str(), "%d,%[^,],%lf", &accNum, &name[0], &balance);

Account acc;

acc.accountNumber = accNum;

acc.name = name;

acc.balance = balance;

accounts.push\_back(acc);

}

file.close();

}

void createAccount() {

Account acc;

acc.createAccount();

accounts.push\_back(acc);

saveAccountsToFile();

cout << "Account Created Successfully.\n";

}

void displayAllAccounts() {

if (accounts.empty()) {

cout << "No accounts available.\n";

return;

}

cout << left << setw(10) << "Acc No" << setw(20) << "Name" << "Balance\n";

for (const auto& acc : accounts) {

acc.showAccount();

}

}

void viewAccount(int accNum) {

for (const auto& acc : accounts) {

if (acc.getAccountNumber() == accNum) {

acc.showAccount();

return;

}

}

cout << "Account not found.\n";

}

void depositMoney(int accNum, double amount) {

for (auto& acc : accounts) {

if (acc.getAccountNumber() == accNum) {

acc.deposit(amount);

saveAccountsToFile();

cout << "Amount Deposited.\n";

return;

}

}

cout << "Account not found.\n";

}

void withdrawMoney(int accNum, double amount) {

for (auto& acc : accounts) {

if (acc.getAccountNumber() == accNum) {

if (acc.withdraw(amount)) {

saveAccountsToFile();

cout << "Amount Withdrawn.\n";

}

return;

}

}

cout << "Account not found.\n";

}

void deleteAccount(int accNum) {

for (auto it = accounts.begin(); it != accounts.end(); ++it) {

if (it->getAccountNumber() == accNum) {

accounts.erase(it);

saveAccountsToFile();

cout << "Account Deleted.\n";

return;

}

}

cout << "Account not found.\n";

}

int main() {

loadAccountsFromFile();

int choice, accNum;

double amount;

do {

cout << "\n=== Banking System Menu ===\n";

cout << "1. Create Account\n";

cout << "2. View Account\n";

cout << "3. Deposit Money\n";

cout << "4. Withdraw Money\n";

cout << "5. Delete Account\n";

cout << "6. Display All Accounts\n";

cout << "7. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

createAccount();

break;

case 2:

cout << "Enter Account Number: ";

cin >> accNum;

viewAccount(accNum);

break;

case 3:

cout << "Enter Account Number: ";

cin >> accNum;

cout << "Enter Amount: ";

cin >> amount;

depositMoney(accNum, amount);

break;

case 4:

cout << "Enter Account Number: ";

cin >> accNum;

cout << "Enter Amount: ";

cin >> amount;

withdrawMoney(accNum, amount);

break;

case 5:

cout << "Enter Account Number to Delete: ";

cin >> accNum;

deleteAccount(accNum);

break;

case 6:

displayAllAccounts();

break;

case 7:

cout << "Exiting... Thank you!\n";

break;

default:

cout << "Invalid choice. Try again.\n";

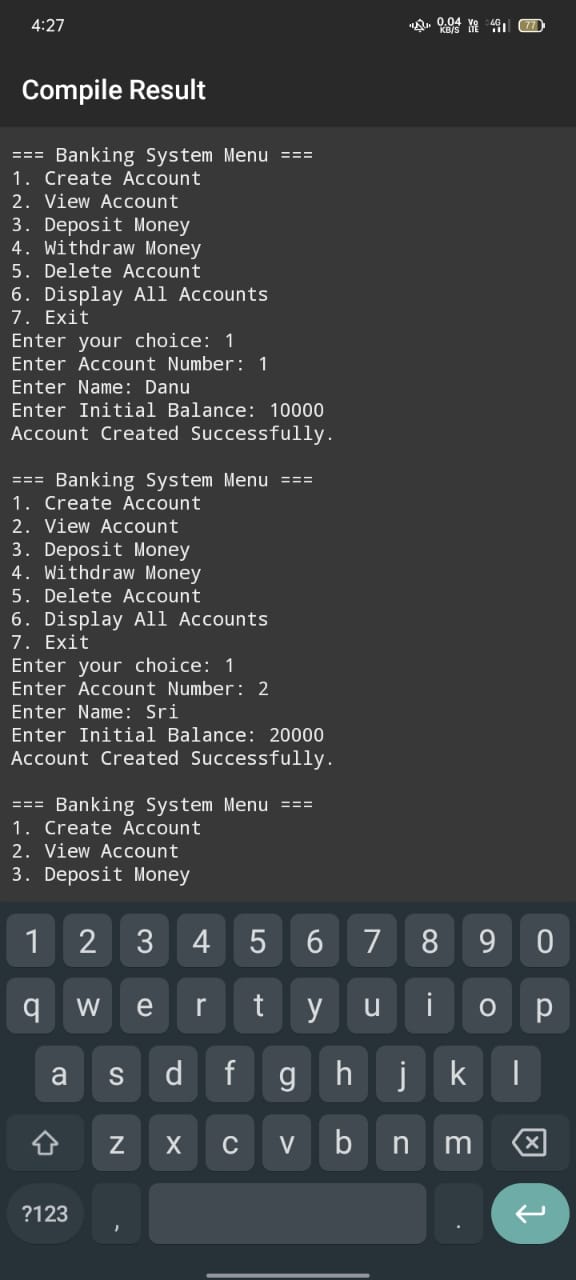
}

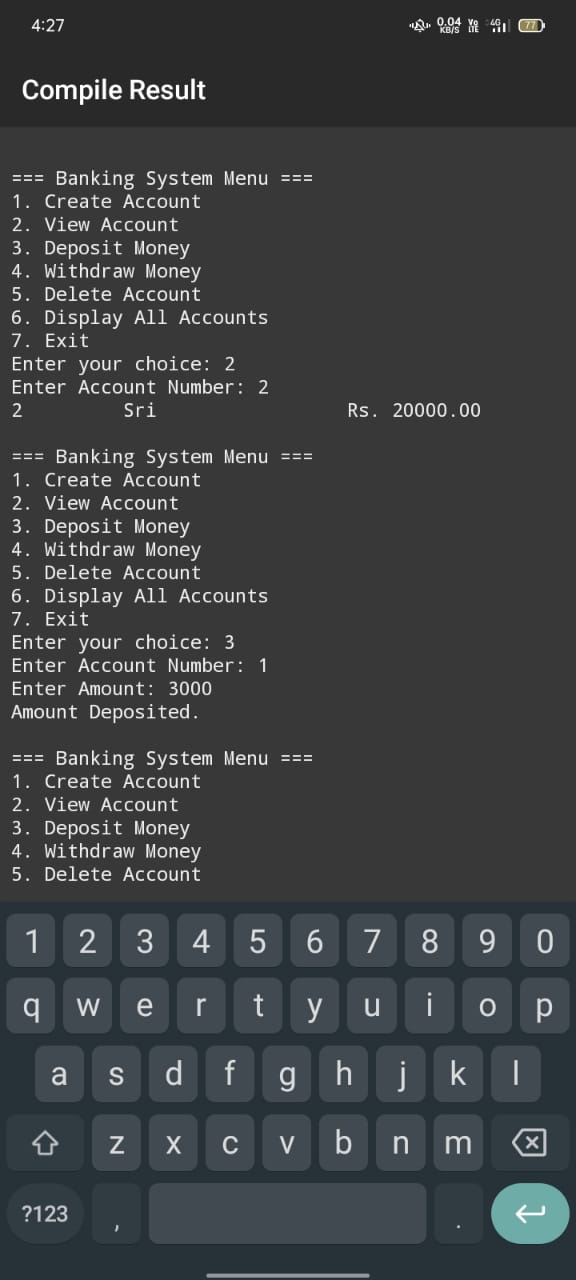
} while (choice != 7);

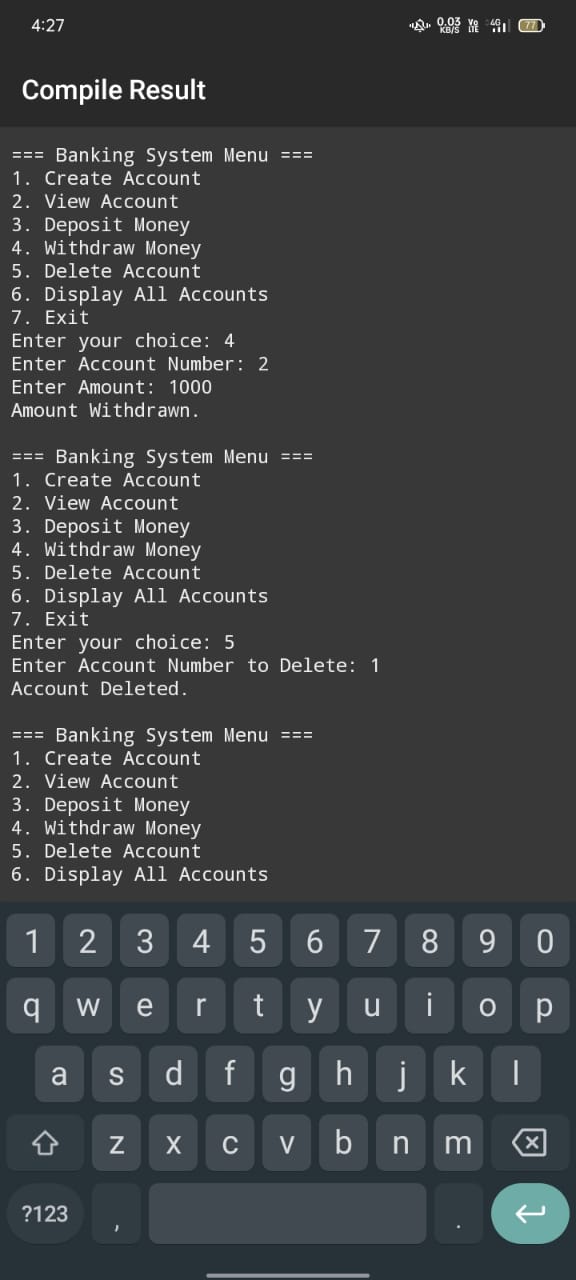
return 0;

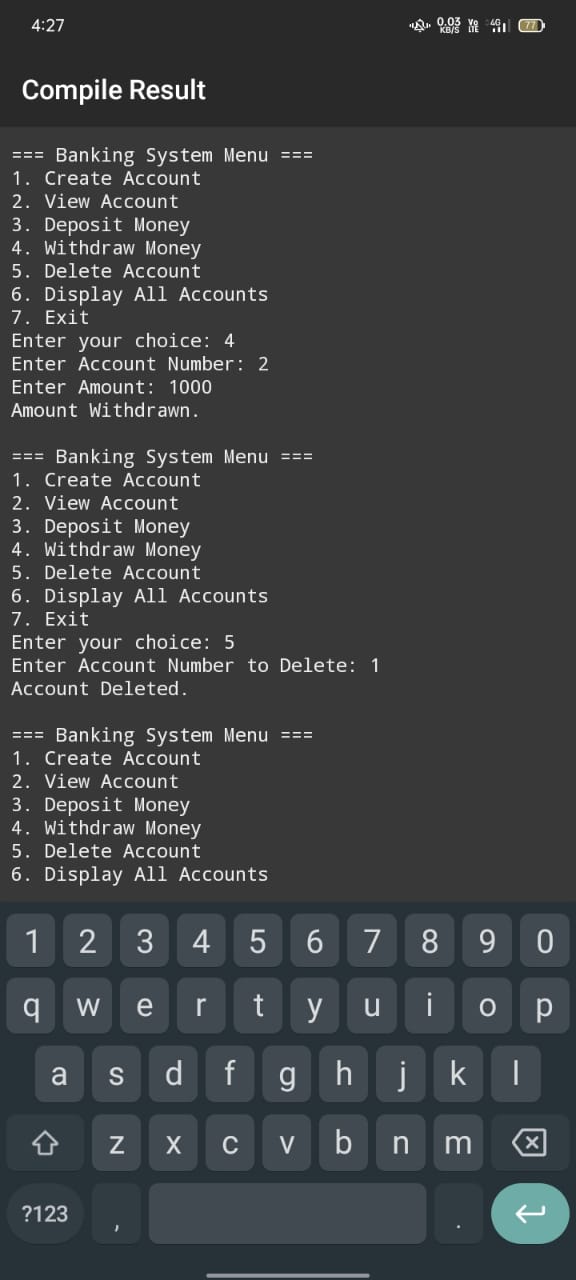
}

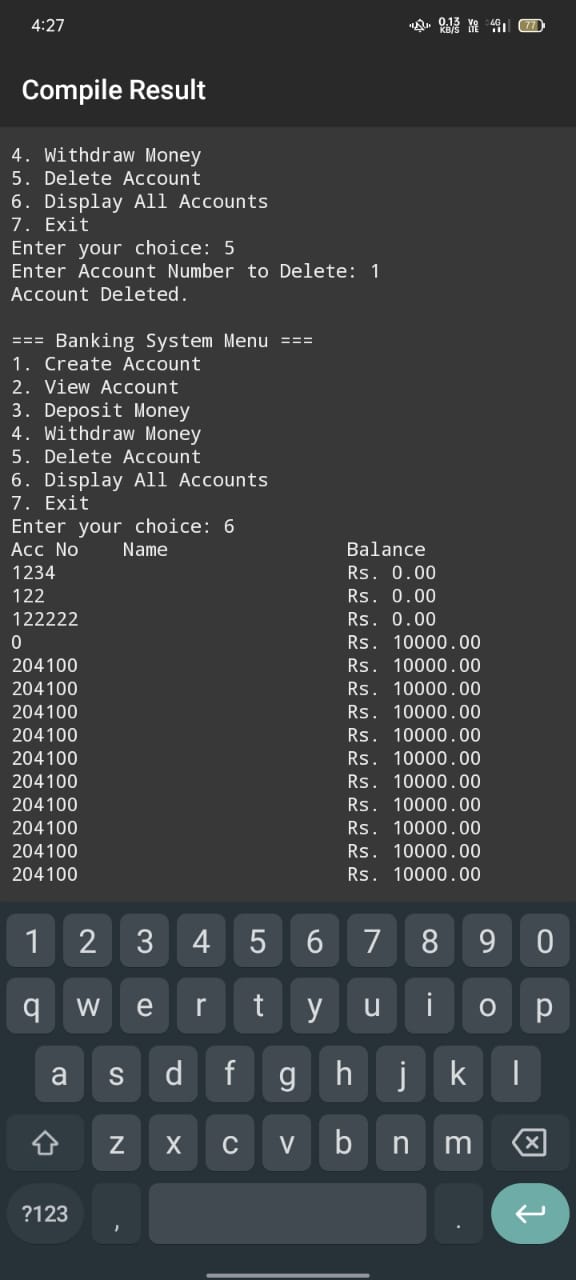
## 3.2.Output





****

****

****

**3.3 Functionality:**

* **Create New Account**  
  Allows the user to create a new bank account by entering details such as account number, account holder's name, and initial deposit amount. The details are stored in a file for future access.
* **Display Account Details**  
  Enables users to view complete details of a specific account by entering the account number. It retrieves and displays the data stored in the file.
* **Deposit Amount**  
  Facilitates depositing money into a selected account. The system updates the account balance accordingly and saves the changes to the file.
* **Withdraw Amount**  
  Allows withdrawal of money from a specific account. It checks for sufficient balance before updating the amount and prevents overdrawing.
* **Delete Account**  
  Provides the option to permanently remove an account from the system based on the account number. The file is updated to reflect this change.
* **Display All Accounts**  
  Lists all existing bank accounts with details such as account number, name, and balance. This helps in managing and monitoring records.

**CHAPTER – 4**

# CONCLUSION

In conclusion, the **Banking Record System** developed using C++ successfully demonstrates how fundamental banking operations can be implemented using object-oriented programming and file handling techniques. The system provides a simple yet effective way to manage bank accounts, including features like account creation, deposit, withdrawal, and deletion. By using a menu-driven interface and persistent file storage, the project offers a practical solution for handling account records in a basic yet functional manner. Through the development of this project, key programming concepts such as classes, objects, encapsulation, and data validation have been effectively applied. It not only serves as a useful educational tool for understanding the real-world application of programming but also lays the groundwork for future enhancements like user authentication, interest calculations, and GUI-based interactions. Overall, this project has helped in strengthening both technical and logical problem-solving skills while addressing a real-life scenario in the financial domain.

# CHAPTER – 5

# REFERENCES

* Balagurusamy, E.  
  Programming in C++  
  Tata McGraw-Hill Education, 7th Edition, 2017.
* Robert Lafore  
  Object-Oriented Programming in C++  
  Sams Publishing, 4th Edition, 2001.
* Bjarne Stroustrup  
  The C++ Programming Language  
  Addison-Wesley, 4th Edition, 2013.