C Programming Major project

1. Title Page

- Project Title: Bank Management
- Course code: CSEG1032
- Team Members
    - o Shashwat Singh – 590022548
    - o Bhoomi Verma - 590021858

--SUBMISSION DATE: 30/11/2025

2. ABSTRACT

The Bank Management System is a C-based application designed to manage essential banking operations in a structured and secure manner. The system allows users to create new bank accounts, view account details, update personal information, perform deposits and withdrawals, and securely delete accounts when required. All records are permanently stored and retrieved using external files, ensuring data persistence even after program termination. The primary goal of this project is to provide a reliable, user-friendly interface for managing banking data while demonstrating practical application of C programming concepts learned throughout the course.

3. PROBLEM DEFINITION

-BACKGROUND

Banking institutions traditionally rely on manual record-keeping to manage customer accounts and transactions. This approach increases the chances of data errors, information loss, slow transaction processing, and difficulties in maintaining accuracy over time. With the growing need for digital solutions, banks require a system that can handle essential operations such as account creation, balance management, and financial transactions in a secure and efficient manner. A computerized solution developed using the C programming language offers a reliable method to store and manage banking data while reducing human intervention and operational delays.

-SCOPE AND CONSTRAINTS

The Bank Management System aims to automate core banking operations using the C programming language. The project also demonstrates the practical application of modular programming, data structures, and file input/output techniques to deliver an efficient and educational software solution. Despite automating essential operations, the project has certain limitations due to its academic scope.

4. SYSTEM DESIGN (FLOWCHART AND ALGORITHM)

System Design defines the structure and workflow of the Bank Management System, showing how each function operates and interacts with stored data. It includes the development of flowcharts and algorithms to ensure smooth processing of user requests and reliable handling of banking operations.

-DESIGN PHILOSOPHY

The design of the Bank Management System focuses on simplicity, reliability, and modular functionality. The system uses a structured programming approach to ensure that each banking operation—such as account creation, transactions, and record management—is handled through separate functions, improving maintainability and ease of understanding.

Function Group	Logical Responsibility
Account Search / Display	Retrieve stored account details based on account number and show accurate information
Account Creation	Collect customer details, validate inputs, and store new account records in the file
Deposit Operation	Accept deposit amount, update balance, ensure data integrity, and save updated record
Withdrawal Operation	Validate withdrawal amount against balance, prevent negative balance, and update data

Balance Inquiry	Display real-time balance information for a selected account
Account Modification	Edit and update existing customer details while maintaining consistency in file storage
Account Deletion	Remove unwanted or inactive accounts from records and rewrite file without deleted data
Data Storage & File Handling	Manage permanent storage, retrieval, and modification of accounts using file operations
Menu Navigation	Provide user-friendly interaction for accessing all system functionalities
Input Validation	Prevent invalid entries, incorrect formats, or operations on non-existing accounts
Program Exit & Save	Close program safely while ensuring all updated information is stored correctly

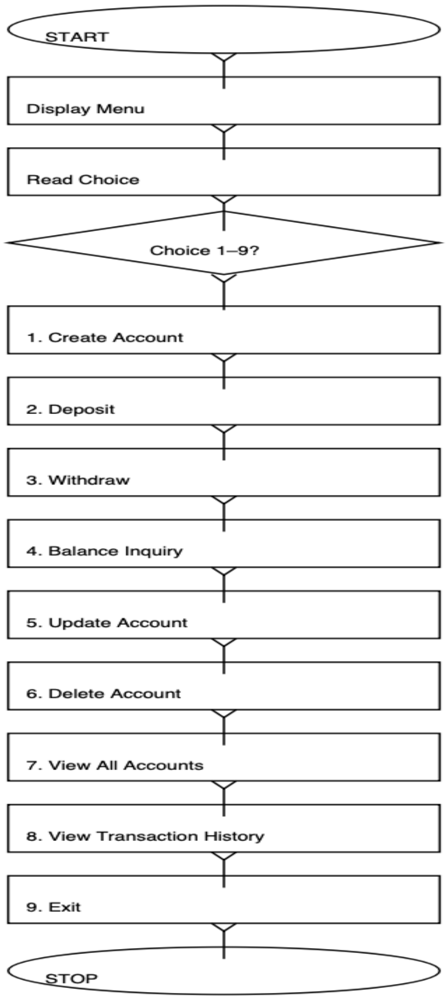
5. Implementation Details

Data persistence in the Bank Management System is achieved through the use of file handling in the C programming language. All customer account details, including account number, personal information, and updated transaction balances, are stored permanently in an external file. This ensures that data remains intact and accessible even after the program is closed, or the system is restarted.

```c
// Snippet for saving the data of all accounts
void view_all_accounts() {

FILE *fp = fopen("accounts.txt", "r");
struct Account acc;

printf("All Accounts:\n");
while (fscanf(fp, "%d %s %s %s %s %f %d %d %d %d %d %d", &acc.acc_no, acc.name, acc.address, acc.phone,
acc.acc_type, &acc.balance, &acc.dob_day, &acc.dob_month, &acc.dob_year, &acc.deposit_day, &acc.deposit_month,
&acc.deposit_year) != EOF) {
printf("Acc No: %d, Name: %s, Balance: %.2f\n", acc.acc_no, acc.name, acc.balance);
}

fclose(fp);

}
```

Bank System Flowchart:

The following flowchart shows the details of the algorithms steps implemented in the Bank System, including all conditional checks for successful execution.



6.Conclusion

The Bank Management System developed in C demonstrates how core banking operations can be managed efficiently using a structured programming approach. This project successfully implements essential functionalities such as account creation, customer record management, deposits, withdrawals, and balance inquiries. Through file handling, the system ensures data storage and retrieval, making it applicable for basic real-world scenarios.

7.Code

```
#include <stdio.h>
#include <stdlib.h>
```

```c
#include <string.h>
#include <time.h>

#define MAX_NAME 50
#define MAX_ADDRESS 100

struct Account {
int acc_no;
char name[MAX_NAME];
char address[MAX_ADDRESS];
char phone[15];
char acc_type[20];
float balance;
int dob_day, dob_month, dob_year;
int deposit_day, deposit_month, deposit_year;
};

void create_account();
void deposit();
void withdraw();
void balance_inquiry();
void update_account();
void delete_account();
void view_all_accounts();
void add_transaction(int acc_no, char *type, float amount);
void view_transaction_history(int acc_no);
void display_menu();
void delay(int number_of_seconds);

int main() {
int choice;
while (1) {
display_menu();
printf("Enter your choice: ");
scanf("%d", &choice);
switch (choice) {
case 1:
create_account();
break;
case 2:
deposit();
break;
case 3:
withdraw();
break;
case 4:
balance_inquiry();
break;
case 5:
update_account();
break;
case 6:
delete_account();
break;
case 7:
view_all_accounts();
break;
case 8:{
int acc_no;
printf("Enter account number: ");
scanf("%d", &acc_no);
view_transaction_history(acc_no);
break;
}
case 9:
printf("Thank you for using our banking system.\n");
exit(0);
default:
printf("Invalid choice. Please try again.\n");
}
delay(1);
}
return 0;
}

void display_menu() {
printf("\n=== Banking System Menu ===\n");
printf("1. Create Account\n");
printf("2. Deposit\n");
printf("3. Withdraw\n");
printf("4. Balance Inquiry\n");
printf("5. Update Account\n");
printf("6. Delete Account\n");
printf("7. View All Accounts\n");
printf("8. View Transaction History\n");
printf("9. Exit\n");
}

void create_account() {
FILE *fp = fopen("accounts.txt", "a");
if (fp == NULL) {
printf("Error opening file.\n");
return;
}
struct Account acc;
printf("Enter account number: ");
scanf("%d", &acc.acc_no);
printf("Enter name: ");
scanf("%s", acc.name);
printf("Enter address: ");
scanf("%s", acc.address);
printf("Enter phone: ");
scanf("%s", acc.phone);
printf("Enter account type (savings/current): ");
scanf("%s", acc.acc_type);
printf("Enter initial balance: ");
```

```
scanf("%f", &acc.balance);
printf("Enter date of birth (dd mm yyyy): ");
scanf("%d %d %d", &acc.dob_day, &acc.dob_month, &acc.dob_year);
time_t t = time(NULL);
struct tm tm = *localtime(&t);
acc.deposit_day = tm.tm_mday;
acc.deposit_month = tm.tm_mon + 1;
acc.deposit_year = tm.tm_year + 1900;
fprintf(fp, "%d %s %s %s %s %.2f %d %d %d %d %d %d\n", acc.acc_no, acc.name, acc.address, acc.phone, acc.acc_type,
acc.balance, acc.dob_day, acc.dob_month, acc.dob_year, acc.deposit_day, acc.deposit_month, acc.deposit_year);
fclose(fp);
add_transaction(acc.acc_no, "CREATE", acc.balance);
printf("Account created successfully.\n");
}

void deposit() {
FILE *fp = fopen("accounts.txt", "r+");
FILE *temp = fopen("temp.txt", "w");
struct Account acc;
int acc_no;
float amount;
printf("Enter account number: ");
scanf("%d", &acc_no);
printf("Enter amount to deposit: ");
scanf("%f", &amount);
int found = 0;
while (fscanf(fp, "%d %s %s %s %s %f %d %d %d %d %d %d", &acc.acc_no, acc.name, acc.address, acc.phone,
acc.acc_type, &acc.balance, &acc.dob_day, &acc.dob_month, &acc.dob_year, &acc.deposit_day, &acc.deposit_month,
&acc.deposit_year) != EOF) {
if (acc.acc_no == acc_no) {
acc.balance += amount;
found = 1;
add_transaction(acc.acc_no, "DEPOSIT", amount);
printf("Deposit successful. New balance: %.2f\n", acc.balance);
}
fprintf(temp, "%d %s %s %s %s %.2f %d %d %d %d %d %d\n", acc.acc_no, acc.name, acc.address, acc.phone,
acc.acc_type, acc.balance, acc.dob_day, acc.dob_month, acc.dob_year, acc.deposit_day, acc.deposit_month,
acc.deposit_year);
}
fclose(fp);
fclose(temp);
if (!found) {
printf("Account not found.\n");
remove("temp.txt");
} else {
remove("accounts.txt");
rename("temp.txt", "accounts.txt");
}
}

void withdraw() {
FILE *fp = fopen("accounts.txt", "r+");
FILE *temp = fopen("temp.txt", "w");
```

```
struct Account acc;
int acc_no;
float amount;
printf("Enter account number: ");
scanf("%d", &acc_no);
printf("Enter amount to withdraw: ");
scanf("%f", &amount);
int found = 0;
while (fscanf(fp, "%d %s %s %s %s %f %d %d %d %d %d %d", &acc.acc_no, acc.name, acc.address, acc.phone,
acc.acc_type, &acc.balance, &acc.dob_day, &acc.dob_month, &acc.dob_year, &acc.deposit_day, &acc.deposit_month,
&acc.deposit_year) != EOF) {
if (acc.acc_no == acc_no) {
if (acc.balance >= amount) {
acc.balance -= amount;
found = 1;
add_transaction(acc.acc_no, "WITHDRAW", amount);
printf("Withdrawal successful. New balance: %.2f\n", acc.balance);
} else {
printf("Insufficient balance.\n");
}
}
fprintf(temp, "%d %s %s %s %s %.2f %d %d %d %d %d %d\n", acc.acc_no, acc.name, acc.address, acc.phone,
acc.acc_type, acc.balance, acc.dob_day, acc.dob_month, acc.dob_year, acc.deposit_day, acc.deposit_month,
acc.deposit_year);
}
fclose(fp);
fclose(temp);
if (!found) {
printf("Account not found.\n");
remove("temp.txt");
} else {
remove("accounts.txt");
rename("temp.txt", "accounts.txt");
}
}

void balance_inquiry() {
FILE *fp = fopen("accounts.txt", "r");
struct Account acc;
int acc_no;
printf("Enter account number: ");
scanf("%d", &acc_no);
int found = 0;
while (fscanf(fp, "%d %s %s %s %s %f %d %d %d %d %d %d", &acc.acc_no, acc.name, acc.address, acc.phone,
acc.acc_type, &acc.balance, &acc.dob_day, &acc.dob_month, &acc.dob_year, &acc.deposit_day, &acc.deposit_month,
&acc.deposit_year) != EOF) {
if (acc.acc_no == acc_no) {
printf("Account Holder: %s\nAddress: %s\nPhone: %s\nAccount Type: %s\nBalance: %.2f\nDOB: %d/%d/%d\n",
acc.name, acc.address, acc.phone, acc.acc_type, acc.balance, acc.dob_day, acc.dob_month, acc.dob_year);
found = 1;
break;
}
}
if (!found) {
```

```
printf("Account not found.\n");
}
fclose(fp);
}

void update_account() {
FILE *fp = fopen("accounts.txt", "r+");
FILE *temp = fopen("temp.txt", "w");
struct Account acc;
int acc_no;
printf("Enter account number: ");
scanf("%d", &acc_no);
int found = 0;
while (fscanf(fp, "%d %s %s %s %s %f %d %d %d %d %d %d", &acc.acc_no, acc.name, acc.address, acc.phone,
acc.acc_type, &acc.balance, &acc.dob_day, &acc.dob_month, &acc.dob_year, &acc.deposit_day, &acc.deposit_month,
&acc.deposit_year) != EOF) {
if (acc.acc_no == acc_no) {
found = 1;
printf("Enter new address: ");
scanf("%s", acc.address);
printf("Enter new phone: ");
scanf("%s", acc.phone);
printf("Account updated successfully.\n");
}
fprintf(temp, "%d %s %s %s %s %.2f %d %d %d %d %d %d\n", acc.acc_no, acc.name, acc.address, acc.phone,
acc.acc_type, acc.balance, acc.dob_day, acc.dob_month, acc.dob_year, acc.deposit_day, acc.deposit_month,
acc.deposit_year);
}
fclose(fp);
fclose(temp);
if (!found) {
printf("Account not found.\n");
remove("temp.txt");
} else {
remove("accounts.txt");
rename("temp.txt", "accounts.txt");
}
}

void delete_account() {
FILE *fp = fopen("accounts.txt", "r");
FILE *temp = fopen("temp.txt", "w");
struct Account acc;
int acc_no;
printf("Enter account number: ");
scanf("%d", &acc_no);
int found = 0;
while (fscanf(fp, "%d %s %s %s %s %f %d %d %d %d %d %d", &acc.acc_no, acc.name, acc.address, acc.phone,
acc.acc_type, &acc.balance, &acc.dob_day, &acc.dob_month, &acc.dob_year, &acc.deposit_day, &acc.deposit_month,
&acc.deposit_year) != EOF) {
if (acc.acc_no != acc_no) {
```

```
fprintf(temp, "%d %s %s %s %s %.2f %d %d %d %d %d %d\n", acc.acc_no, acc.name, acc.address, acc.phone,
acc.acc_type, acc.balance, acc.dob_day, acc.dob_month, acc.dob_year, acc.deposit_day, acc.deposit_month,
acc.deposit_year);
} else {
found = 1;
}
}
fclose(fp);
fclose(temp);
if (!found) {
printf("Account not found.\n");
remove("temp.txt");
} else {
remove("accounts.txt");
rename("temp.txt", "accounts.txt");
printf("Account deleted successfully.\n");
}
}

void view_all_accounts() {
FILE *fp = fopen("accounts.txt", "r");
struct Account acc;
printf("All Accounts:\n");
while (fscanf(fp, "%d %s %s %s %s %f %d %d %d %d %d %d", &acc.acc_no, acc.name, acc.address, acc.phone,
acc.acc_type, &acc.balance, &acc.dob_day, &acc.dob_month, &acc.dob_year, &acc.deposit_day, &acc.deposit_month,
&acc.deposit_year) != EOF) {
printf("Acc No: %d, Name: %s, Balance: %.2f\n", acc.acc_no, acc.name, acc.balance);
}
fclose(fp);
}

void add_transaction(int acc_no, char *type, float amount) {
FILE *fp = fopen("transactions.txt", "a");
time_t t = time(NULL);
struct tm tm = *localtime(&t);
fprintf(fp, "%d %s %.2f %d/%d/%d %d:%d:%d\n", acc_no, type, amount, tm.tm_mday, tm.tm_mon + 1, tm.tm_year +
1900, tm.tm_hour, tm.tm_min, tm.tm_sec);
fclose(fp);
}

void view_transaction_history(int acc_no) {
FILE *fp = fopen("transactions.txt", "r");
int acc;
char type[20];
float amount;
int day, month, year, hour, min, sec;
printf("Transaction History for Account %d:\n", acc_no);
while (fscanf(fp, "%d %s %f %d/%d/%d %d:%d:%d", &acc, type, &amount, &day, &month, &year, &hour, &min, &sec) !=
EOF) {
if (acc == acc_no) {
printf("%s %.2f on %d/%d/%d at %d:%d:%d\n", type, amount, day, month, year, hour, min, sec);
}
}
```

```
fclose(fp);
}


void delay(int number_of_seconds) {
int milli_seconds = 1000 * number_of_seconds;
clock_t start_time = clock();
while (clock() < start_time + milli_seconds);
}
```