

UNITEDWORLD INSTITUTE OF TECHNOLOGY(UIT)

Summative Assessment (SA): [ **Simple Banking System**]

Submitted by

[Heer Patel ]

(Enroll. No.: ku2407u532)

**Course Code and Name: [Eg: 24BSPC11 – Problem sloving and**

**Programming Using C]**

B.Sc. (Hons.) Computer Science / Data Science /

AIML

I Semester – July – Nov 2024



July/Nov 2024

Contents

[**1.** **Introduction** 2](#_Toc4695)

[**2.** **Understanding the Banking System** 2](#_Toc4696)

[**3.** **Using Structures to Govern Customer Information** 3](#_Toc4697)

[**4. Using Pointers for Dynamic Memory Management** 3](#_Toc4698)

[**5. Functions for Operations** 4](#_Toc4699)

[**6. Input Validation** 5](#_Toc4700)

[**7. Security Considerations** 6](#_Toc4701)

**[8. Conclusio](https://word-edit.officeapps.live.com/we/wordeditorframe.aspx?ui=en-US&rs=en-US&hid=CfEkTDV5w0iOxDwWKO%2Flfg.0.14.0&wopisrc=https%3A%2F%2Fwopi.onedrive.com%2Fwopi%2Ffiles%2FB88D40A01412836F!111&wdo=2&wde=docx&sc=host%3D%26qt%3DDefault&wdp=3&uih=onedrivecom&jsapi=1&jsapiver=v2&corrid=116708a2-5da2-4f64-86db-1f90c02256cf&usid=116708a2-5da2-4f64-86db-1f90c02256cf&newsession=1&sftc=1&uihit=editaspx&muv=1&cac=1&sams=1&mtf=1&sfp=1&sdp=1&hch=1&hwfh=1&dchat=1&wdorigin=Other&instantedit=1&wopicomplete=1&wdredirectionreason=Unified_SingleFlush" \l "_Toc182178468)**[**n** ................................................................................ 8](https://word-edit.officeapps.live.com/we/wordeditorframe.aspx?ui=en-US&rs=en-US&hid=CfEkTDV5w0iOxDwWKO%2Flfg.0.14.0&wopisrc=https%3A%2F%2Fwopi.onedrive.com%2Fwopi%2Ffiles%2FB88D40A01412836F!111&wdo=2&wde=docx&sc=host%3D%26qt%3DDefault&wdp=3&uih=onedrivecom&jsapi=1&jsapiver=v2&corrid=116708a2-5da2-4f64-86db-1f90c02256cf&usid=116708a2-5da2-4f64-86db-1f90c02256cf&newsession=1&sftc=1&uihit=editaspx&muv=1&cac=1&sams=1&mtf=1&sfp=1&sdp=1&hch=1&hwfh=1&dchat=1&wdorigin=Other&instantedit=1&wopicomplete=1&wdredirectionreason=Unified_SingleFlush#_Toc182178468)

# 1. Introduction

A simple banking system is one which can administrate details about the customer, account details and transactions in a banking scenario. This report discusses the implementation of a simple banking system in C, using concepts such as structures, pointers, and functions. The essence is that it should have the ability to manage accounts efficiently and it could be able to create an account, deposit funds, withdraw funds, and transfer funds. Input validation will be conducted, for which operations are correct and problems such as overdrafts do not arise.

# 2. Understanding the Banking System

The base operations that are part of a banking system include the management of customer data and the conduct of financial transactions. In our simple system, the following are supported:

Account Creation: The system should be able to support the creation of accounts by users using the following details:

personal information and account information. - Deposits: Money can be added to customer accounts. - Withdrawals: Money can be withdrawn from a customer's account with checks on balance being high.

- Fund Transfers: The money can be transferred between the various accounts.

The system will use structures and pointers for handling and accessing data for the customers. All operations will be validated for input to meet the predetermined constraints.

# 3. Using Structures to Govern Customer Information

In C, structures group related data under a single unit. In the banking system, a structure will represent the customer and his respective account. The general fields of a structure may include:

Customer Name: The name of the account holder.

Account Number: A unique identifier for the account.

Balance: The current balance in the account. Account Type: (e.g., savings, checking).

PIN/Password: For security purposes.

**Example of a structure definition in C:**

typedef struct { char name[100]; int accountNumber; float balance; char accountType[50]; char password[20];

} Account;

# 4. Using Pointers for Dynamic Memory Management

Pointers are essential memory management in C. We could dynamically allocate memory for customer accounts and manage more than one account in the banking system using pointers.

Dynamically Allocate Account: Instead of fixed-size arrays for accounts, we'll use the pointers for dynamic memory so that the system could scale depending on the number of customers.

**Example:**

Account \*createAccount(int accountNumber) {

Account \*newAccount = (Account \*)malloc(sizeof(Account));

newAccount->accountNumber = accountNumber; newAccount->balance = 0.0; return newAccount;

}

In this function, we assign some memory space for a new account using malloc. Now we insert the address of the newly created account in the pointer, newAccount.

# 5. Functions for Operations

The bank system must serve the following core functions to carry out basic operations.

**Create Account**:

Initializes a new account and stores the details.

**Example** function for creating an account: void createAccount(Account \*acc, char \*name, int accountNumber, char \*accountType) { strcpy(acc->name, name); acc>accountNumber = accountNumber; strcpy(acc->accountType, accountType); acc-

>balance = 0.0; // Initial balance is 0

}

**Deposit**:

Adds money to the account balance.

**Example** deposit function: void deposit(Account \*acc, float amount) {

if (amount > 0) { acc>balance += amount; printf("Deposited %.2f. New balance: %.2f\n", amount, acc->balance);

} else {

printf("Invalid deposit amount.\n");

}

}

**Withdraw**:

Subtracts money from the account, ensuring there are sufficient funds.

**Example** withdrawal function: void withdraw(Account \*acc, float amount) { if (amount <= acc->balance && amount > 0) { acc->balance -= amount; printf("Withdrew %.2f. New balance: %.2f\n", amount, acc->balance);

} else {

printf("Insufficient funds or invalid amount.\n");

}

}

**Transfer Funds**:

Allows a transfer of funds between two accounts.  **Example** fund transfer function: void transferFunds(Account

\*fromAcc, Account \*toAcc, float amount) { if (amount <= fromAcc->balance && amount > 0) { fromAcc->balance -= amount; toAcc->balance += amount; printf("Transferred %.2f. New balance of sender: %.2f, receiver: %.2f\n", amount, fromAcc->balance, toAcc>balance);

} else {

printf("Insufficient funds or invalid amount.\n");

}

}

# 6. Input Validation

Input validation is a part of the banking system to assure the integrity of operations. For instance:

Deposit / Withdrawal Limits: Note that the deposit/withdrawal limits are positive numbers. Withdrawing Balance Check While withdrawing, the amount for withdrawal cannot exceed the available balance in the account. Overdraft protection The system should never let overdrafts be created by withdrawals unless overdraft protection is first enabled.

**Example** of validating a withdrawal: float withdrawAmount; printf("Enter withdrawal amount: "); scanf("%f", &withdrawAmount); if (withdrawAmount <= 0 || withdrawAmount > acc->balance)

{

printf("Invalid withdrawal amount.\n");

} else {

withdraw(acc, withdrawAmount);

}

# 7. Security Considerations

Although the system discussed here is very simple, it is worth mentioning here that user information can be secured. For realworld applications, password encryption and secure access mechanisms will be used.

For example, in our simple system above, the password field is stored as plain text though the same in the actual production application would be hashed with SHA-256 or similar methods of hashing.

## 8. Conclusion

This paper outlines the basic elements needed to provide a simple banking system in C. The system could support potentially thousands of customers and transactions if \*structures\* are used for customer accounts and \*pointers\* for dynamic memory allocation. It will not only be workable, but it will also be secure for two reasons: \*functions\* may be used to carry out the operations on the accounts and because of input validation. This implementation is quite simple but serves as a basis for more complex banking systems in a real application. Even file handling for persisting data, the mechanisms of user authentication, and interest calculations or loan management could be added here as extensions for later use.