CTEC 12073 – Structured Programming II University of Kelaniya

Coursework Assignment

CT/2020/060 – Devasurendra R . W

Basic Banking Application

Introduction

The "Basic Banking Application" is a C programming project that aims to create a simple yet functional banking system. The application will allow users to perform essential banking operations such as creating new bank accounts, depositing and withdrawing money, checking account balance, and transferring funds between accounts. The project will focus on efficient and error-free implementation with proper error handling for various scenarios.

Features

1. **Create Bank Account:**
    - Users can open new bank accounts by providing their name and an initial deposit amount.

- Each account will be assigned a unique account number for identification.

2. **Money:**
   - Account holders can deposit money into their bank accounts.
   - The application will update the account balance accordingly.

3. **Withdraw Money:**
   - Account holders can withdraw money from their bank accounts.
   - The system will ensure that the account has sufficient funds before processing the withdrawal.

4. **Check Account Balance:**
   - Users can inquire about their account balance to know how much money they have in their account.

5. **Transfer Funds:**
   - Account holders can transfer money from one account to another.
   - The application will perform necessary checks, such as verifying sufficient funds and valid account numbers.

6. **Error Handling:**
   - The application will implement robust error handling for various scenarios, such as insufficient funds during withdrawals, invalid account numbers during transactions, and other possible errors.
   - Informative error messages will be displayed to guide users in resolving issues.

Project Implementation

1.

```
#include <stdio.h>
#include <stdbool.h>
#define MAX_ACCOUNTS 100

// Structure to represent a bank account
struct BankAccount {
    int accountNumber;
    char accountHolder[50];
    float balance;
};

// Array to hold bank accounts
struct BankAccount accounts[MAX_ACCOUNTS];
int totalAccounts = 0;

// Function prototypes
void createAccount();
void deposit();
void withdraw();
void checkBalance();
```

```
void transfer();

int main() {
    int choice;

    while (1) {
        printf("\nBanking Application\n");
        printf("1. Create Account\n");
        printf("2. Deposit\n");
        printf("3. Withdraw\n");
        printf("4. Check Balance\n");
        printf("5. Transfer\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                createAccount();
                break;
            case 2:
                deposit();
                break;
```

```
            case 3:
                withdraw();
                break;
            case 4:
                checkBalance();
                break;
            case 5:
                transfer();
                break;
            case 6:
                printf("Exiting the application. Thank you!\n");
                return 0;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}
```

Explanation:

- The code starts by including the necessary header files and defining constants such as '**MAX_ACCOUNTS'** to set the maximum number of accounts the application can handle.
- The project uses a '**struct BankAccount'** to represent each bank account. This structure contains an '**accountNumber**', **'accountHolder'** (account holder name), and '**balance**'.
- The array **'accounts'** is used to store multiple bank accounts, and '**totalAccounts**' keeps track of the number of accounts in the system.

2.

```
void createAccount() {
   if (totalAccounts == MAX_ACCOUNTS) {
      printf("Maximum number of accounts reached.\n");
      return;
   }

   struct BankAccount newAccount;
   printf("Enter account holder name: ");
   scanf("%s", newAccount.accountHolder);
   printf("Enter initial deposit amount: ");
   scanf("%f", &newAccount.balance);
```

```
        newAccount.accountNumber = totalAccounts + 1001; // Assuming
account numbers start from 1001

        accounts[totalAccounts++] = newAccount;



        printf("Account created successfully. Account number: %d\n",
newAccount.accountNumber);

    }
```

Explanation:

- The **‘createAccount()’** function allows users to create a new bank account.
- It first checks if the maximum number of accounts (‘**MAX_ACCOUNTS’**) has been reached. If so, it displays a message and returns.
- Next, the function prompts the user to enter the account holder's name and initial deposit amount.
- The account number is generated using **‘totalAccounts + 1001’** (assuming account numbers start from 1001) and assigned to the new account.
- The new account is added to the **‘accounts’** array, and the **‘totalAccounts’** count is incremented.
- Finally, a success message is displayed, showing the newly created account number.

3.

```
        void deposit() {
            int accountNumber;
            float amount;
            printf("Enter account number: ");
            scanf("%d", &accountNumber);
            for (int i = 0; i < totalAccounts; i++) {
                if (accounts[i].accountNumber == accountNumber) {
                    printf("Enter the amount to deposit: ");
                    scanf("%f", &amount);
                    if (amount > 0) {
                        accounts[i].balance += amount;
                        printf("Deposit successful. New balance: %.2f\n",
accounts[i].balance);
                    } else {
                        printf("Invalid deposit amount. Please enter a positive value.\n");
                    }
                    return;
                }
            }
            printf("Account not found. Please check the account number and try
again.\n");
```

```
    }
```

Explanation:

- The **‘deposit()’** function handles the deposit operation for a given account.
- The user is prompted to enter the account number they want to deposit money into.
- The function then searches for the account with the provided account number in the **‘accounts’** array.
- If the account is found, the user is prompted to enter the deposit amount.
- If the amount is greater than 0, it is added to the account's balance, and a success message with the updated balance is displayed.
- If the amount is not positive, an error message is shown.
- If the account is not found, an error message is displayed.

4.

```
void withdraw() {
   int accountNumber;
   float amount;
   printf("Enter account number: ");
   scanf("%d", &accountNumber);

   for (int i = 0; i < totalAccounts; i++) {
```

```
        if (accounts[i].accountNumber == accountNumber) {
            printf("Enter the amount to withdraw: ");
            scanf("%f", &amount);
            if (amount > 0) {
                if (amount <= accounts[i].balance) {
                    accounts[i].balance -= amount;
                    printf("Withdrawal successful. New balance: %.2f\n",
accounts[i].balance);
                } else {
                    printf("Insufficient funds. Cannot withdraw.\n");
                }
            } else {
                printf("Invalid withdrawal amount. Please enter a positive value.\n");
            }
            return;
        }
    }

    printf("Account not found. Please check the account number and try again.\n");
}
```

Explanation:

- The **‘withdraw()’** function handles the withdrawal operation for a given account.
- Similar to the **‘deposit()’** function, the user is prompted to enter the account number they want to withdraw money from.
- The function then searches for the account with the provided account number in the **‘accounts’** array.
- If the account is found, the user is prompted to enter the withdrawal amount.
- If the amount is greater than 0, the function checks if the account has sufficient funds (‘**amount <= accounts[i].balance’**) before processing the withdrawal.
- If the account has enough funds, the withdrawal is successful, and the updated balance is displayed.
- If the account does not have sufficient funds or the amount is not positive, appropriate error messages are shown.
- If the account is not found, an error message is displayed.

5.

```
void checkBalance() {
   int accountNumber;
   printf("Enter account number: ");
   scanf("%d", &accountNumber);
```

```
    for (int i = 0; i < totalAccounts; i++) {
        if (accounts[i].accountNumber == accountNumber) {
            printf("Account Holder: %s\n", accounts[i].accountHolder);
            printf("Account Balance: %.2f\n", accounts[i].balance);
            return;
        }
    }

    printf("Account not found. Please check the account number and try again.\n");
}
```

Explanation:

- The **‘checkBalance()’** function allows users to inquire about their account balance.
- The user is prompted to enter the account number they want to check the balance for.
- The function then searches for the account with the provided account number in the **‘accounts’** array.
- If the account is found, the account holder's name and balance are displayed.
- If the account is not found, an error message is displayed.

6.

```
void transfer() {
    int sourceAccountNumber, destinationAccountNumber;
    float amount;
    printf("Enter source account number: ");
    scanf("%d", &sourceAccountNumber);

    for (int i = 0; i < totalAccounts; i++) {
        if (accounts[i].accountNumber == sourceAccountNumber) {
            printf("Enter destination account number: ");
            scanf("%d", &destinationAccountNumber);

            for (int j = 0; j < totalAccounts; j++) {
                if (accounts[j].accountNumber == destinationAccountNumber) {
                    printf("Enter the amount to transfer: ");
                    scanf("%f", &amount);
                    if (amount > 0) {
                        if (amount <= accounts[i].balance) {
                            accounts[i].balance -= amount;
                            accounts[j].balance += amount;
                            printf("Transfer successful.\n");
                        } else {
```

```
                    printf("Insufficient funds. Cannot transfer.\n");
                }
            } else {
                printf("Invalid transfer amount. Please enter a positive value.\n");
            }
            return;
        }
    }


    printf("Destination account not found. Please check the account number and try again.\n");
    return;
  }
}


 printf("Source account not found. Please check the account number and try again.\n");
}
```

Explanation:

- The **‘transfer()’** function handles the transfer of funds between two accounts.
- The user is prompted to enter the source account number and the destination account number for the transfer.
- The function searches for both accounts in the **‘accounts’** array.
- If both accounts are found, the user is prompted to enter the transfer amount.
- If the amount is greater than 0, the function checks if the source account has sufficient funds **(‘amount <= accounts[i].balance’)** before processing the transfer.
- If the source account has enough funds, the transfer is successful, and the balances of both accounts are updated accordingly.
- If the source account does not have sufficient funds or the amount is not positive, appropriate error messages are shown.
- If the source or destination account is not found, an error message is displayed.

```
int main() {



}
```

Explanation:

- The **‘main()’** function is the entry point of the program and serves as the user interface.
- It presents a simple menu to the user, allowing them to choose various banking operations by entering a corresponding number.

- The user's choice is then used in a **‘switch’** statement to call the relevant functions for each operation.
- The user can exit the application by selecting option 6, which terminates the loop and ends the program.

- The Banking Application includes error handling for scenarios like insufficient funds during withdrawals, invalid account numbers during transactions, and other possible errors. When an invalid account number is provided, the application displays an error message informing the user to check the account number and try again. Similarly, if the account is not found during various operations, appropriate error messages are shown to the user.

Project Testing

- Extensive testing will be conducted to ensure the correctness and robustness of the banking application. Various test cases will be designed to cover all possible scenarios, including valid and invalid inputs, edge cases, and error conditions. The project will undergo rigorous testing to identify and fix potential bugs or issues.

Complete C ode

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

```
#include <stdbool.h>

#define MAX_ACCOUNTS 100

// Structure to represent a bank account
struct BankAccount {
    int accountNumber;
    char accountHolder[50];
    float balance;
};

// Array to hold bank accounts
struct BankAccount accounts[MAX_ACCOUNTS];
int totalAccounts = 0;

// Function prototypes
void createAccount();
void deposit();
void withdraw();
```

```
void checkBalance();
void transfer();

int main() {
    int choice;

    while (1) {
        printf("\nBanking Application\n");
        printf("1. Create Account\n");
        printf("2. Deposit\n");
        printf("3. Withdraw\n");
        printf("4. Check Balance\n");
        printf("5. Transfer\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
```

```
            createAccount();
            break;
        case 2:
            deposit();
            break;
        case 3:
            withdraw();
            break;
        case 4:
            checkBalance();
            break;
        case 5:
            transfer();
            break;
        case 6:
            printf("Exiting the application. Thank you!\n");
            return 0;
        default:
            printf("Invalid choice. Please try again.\n");
```

```
        }
    }

    return 0;
}

void createAccount() {
    if (totalAccounts == MAX_ACCOUNTS) {
        printf("Maximum number of accounts reached.\n");
        return;
    }

    struct BankAccount newAccount;
    printf("Enter account holder name: ");
    scanf("%s", newAccount.accountHolder);
    printf("Enter initial deposit amount: ");
    scanf("%f", &newAccount.balance);
```

```
    newAccount.accountNumber = totalAccounts + 1001; // Assuming account numbers start from 1001
    accounts[totalAccounts++] = newAccount;

    printf("Account created successfully. Account number: %d\n", newAccount.accountNumber);
}

void deposit() {
    int accountNumber;
    float amount;
    printf("Enter account number: ");
    scanf("%d", &accountNumber);

    for (int i = 0; i < totalAccounts; i++) {
        if (accounts[i].accountNumber == accountNumber) {
            printf("Enter the amount to deposit: ");
            scanf("%f", &amount);
            if (amount > 0) {
                accounts[i].balance += amount;
```

```
            printf("Deposit successful. New balance: %.2f\n",
accounts[i].balance);
        } else {
            printf("Invalid deposit amount. Please enter a positive
value.\n");
        }
        return;
    }
  }

  printf("Account not found. Please check the account number
and try again.\n");
}

void withdraw() {
  int accountNumber;
  float amount;
  printf("Enter account number: ");
  scanf("%d", &accountNumber);
```

```
    for (int i = 0; i < totalAccounts; i++) {
        if (accounts[i].accountNumber == accountNumber) {
            printf("Enter the amount to withdraw: ");
            scanf("%f", &amount);
            if (amount > 0) {
                if (amount <= accounts[i].balance) {
                    accounts[i].balance -= amount;
                    printf("Withdrawal successful. New balance: %.2f\n", accounts[i].balance);
                } else {
                    printf("Insufficient funds. Cannot withdraw.\n");
                }
            } else {
                printf("Invalid withdrawal amount. Please enter a positive value.\n");
            }
            return;
        }
    }
```

```
    printf("Account not found. Please check the account number and try again.\n");
}

void checkBalance() {
    int accountNumber;
    printf("Enter account number: ");
    scanf("%d", &accountNumber);

    for (int i = 0; i < totalAccounts; i++) {
        if (accounts[i].accountNumber == accountNumber) {
            printf("Account Holder: %s\n", accounts[i].accountHolder);
            printf("Account Balance: %.2f\n", accounts[i].balance);
            return;
        }
    }

    printf("Account not found. Please check the account number and try again.\n");
```

```
}

void transfer() {
    int sourceAccountNumber, destinationAccountNumber;
    float amount;
    printf("Enter source account number: ");
    scanf("%d", &sourceAccountNumber);

    for (int i = 0; i < totalAccounts; i++) {
        if (accounts[i].accountNumber == sourceAccountNumber)
{
            printf("Enter destination account number: ");
            scanf("%d", &destinationAccountNumber);

            for (int j = 0; j < totalAccounts; j++) {
                if (accounts[j].accountNumber ==
destinationAccountNumber) {
                    printf("Enter the amount to transfer: ");
                    scanf("%f", &amount);
                    if (amount > 0) {
```

```
                    if (amount <= accounts[i].balance) {
                        accounts[i].balance -= amount;
                        accounts[j].balance += amount;
                        printf("Transfer successful.\n");
                    } else {
                        printf("Insufficient funds. Cannot transfer.\n");
                    }
                } else {
                    printf("Invalid transfer amount. Please enter a positive value.\n");
                }
                return;
            }
        }

        printf("Destination account not found. Please check the account number and try again.\n");
        return;
    }
}
```

```
    printf("Source account not found. Please check the account number and try again.\n");
}
```