**Module Code : FC723**

**Class/Group: Group A**

**Module Title: Programming Theory**

**Assessment Title: Portfolio Project 2**

**Tutor Name: Sophie Norman**

**Student GUID Number: P453118**

**Date of Submission: 18th May 2025**

“I confirm that this assignment is my own work.

Where I/we have referred to academic sources, I have provided in-text citations and included the sources in the final reference list. “

**PSEUDOCODE**

START

Display “Welcome to the bank Application”

#Loop

While True

Display menu:

1.new account

2. login to existing account

3.exit

Get user\_choice

If user\_choice is 1:

Call CreateAccount()

Elif user\_choice is 2:

Call Login()

Elif user\_choice is 3:

Display “Exiting Application”

Break

Else:

Display “Invalid choice. Please try again.”

END

Define CreateAccount():

Display “---- Create New Account ----”

Get new\_username

Get new\_password

#check length

While length(new\_password)< 8 or length(new\_password)>16:

Display “Password must be 8 to 16 charaters”

Prompt user re-enter new\_password

Display “Enter initial deposit amount:”

Get initial\_deposit

If initial\_deposit < 0:

Display “Invalid amount”

Return

Call store\_amount\_details(new\_username, new\_password, initial\_deposit)

Display “Account created successfully ^^”

END

Def Login():

Display “---- Account Login ----”

Set login\_attempts=0

Set max\_login\_attempts=3

While login\_attempts< max\_login\_attemptes:

Get username

Get password

If call verify\_account\_details(username, password) is True:

Display “Login successful.”

Set current\_account = username #track log-in account

Call BankingMenu(current\_account)

Return

Else:

Login\_attempts = login\_attempts +1

Display” Invalid username or password. Try again.

Display “Too many incorrect attempts. Account locked”

END

Def BankingMenu(account\_name):

While True:

Display “---- Menu ----”

Display”1. Check Balance”

Display”2. Deposit Money”

Display”3. Withdraw Money”

Display”4. Transfer Money”

Display”5. Exit”

Get user\_menu\_choice

If user\_menu\_choice is 1:

Call check\_balance(account\_name)

Elif user\_menu\_choice is 2:

Call deposit(account\_name):

Elif user\_menu\_choice is 3:

Call withdraw(accoumt\_name)

Elif user\_menu\_choice is 4:

Call transfer(account\_name)

Elif user\_menu\_choice is 5:

Display “Logging out…...”

Return

Else:

Display “Invalid option. Try again.”

END

Def check\_balance(account\_name):

Set balance\_2s\_complement= call get\_balance(account\_name)

#Convert 2’s compliment to decimal

Set balance\_decimal = convert\_to\_decimal(balance\_2s\_compliment)

Display “Your current balance is : £”+ balance\_decimal

END

Def deposit(account\_name)

Display “Enter amount to deposit : £”

Get deposit\_amount

If deposit\_amount<=0:

Display “Invalid deposit amount. ”

Return

Call update\_balance(account\_name, deposit\_amount)

Display “Deposit successful. ”

END

Def withdraw(account\_name):

Display “Enter amount to withdraw: £”

Get withdraw\_amount

Set balance = call get\_balance(account\_name)

Set balance-decimal = convert\_to\_decimal(balance)

If withdraw\_amount <= balance\_decimal or (balance\_decimal-withdraw\_amount)>=-1500:

Call update\_balance(account\_name, -withdraw\_amount)

Display “Withdrawal successful. ”

Else:

Display “Insufficient funds. ”

END

Def transfer(from\_account):

Display “Enter account to transfer : ”

Get to\_account

If call account\_exists(to\_account) is False:

Display “Account does not exist.”

Return

Display “Enter amount to transfer: £”

Get transfer\_amount

Set from\_balance = call get\_balance(from\_account)

Set from\_balance\_decimal= convert\_to\_decimal(from\_balance)

If transfer\_amount<=from\_balance\_decimal or (from\_balance\_decimal – transfer\_amount)>= -1500:

Call update\_balance(from\_account, -transfer\_amount)

Call update\_balance(to\_account, transfer\_amount)

Display “Transfer successful. ”

Else:

Display “Insufficient funds for transfer. ”

END

Def account\_exists(account\_name):

Return True or False

END

Def convert\_to\_decimal(twos\_compliment\_value):

If twos\_complement\_value represents a negative:

# Invert bits and add 1, then apply negative sign

Return negative decimal value

Else:

Return positive decimal value

END

Def store\_account\_details(username, password, initial\_deposit):

#Save account data securely

# Encrypt or hash the password before saving

END

Def vertify\_account\_details(username, password):

#Check if user and password match

Return True if they match, False otherwise

END

Def get\_balance(account\_name):

#Retrieves the amount balance in 2’s complement

Return twos\_complement\_balance

END

Def update\_balance(account\_name, amount):

Get current\_balance (2’s complement)

Convert to decimal

Add amount

Convert back to 2’s complement

Store updated value

END

Unit Testing Section

#All major modules will be tested using unit tests:

- Account creation

- Login system (success and fail)

- Deposit, Withdraw, Transfer

- 2’s complement conversion

- Balance updates