James Sigler

Nicholas Paladino

Section 502

banksystem.c Pseudo

* DEFINE admin constant as 1
* DEFINE customer constant as 2
* DEFINE MAX\_CUSTOMERS as 100
* DEFINE function prototypes for menu display and loading data from file
* START main with int argc and char\*
  + DECLARE strings for ID and password
  + DECLARE file pointer data
  + INITIALIZE string fileName to argv[1]
  + DECLARE int numAccounts
  + DECLARE string temp
  + DECLARE Account variable accountCurr
  + DECLARE Account array accounts with MAX\_CUSTOMERS length
  + DECLARE int IDnumberForNewAccount
  + DISPLAY welcome screen
  + LOGIN\_ID label
  + PROMPT user to enter ID
  + STORE in temp
  + IF length of temp isnt 5
    - PRINT error message
    - GOTO LOGIN\_ID label
  + SET ID string to contents of temp
  + LOGIN\_PASS label
  + PROMPT user to enter password
  + STORE in temp
  + IF length of temp is greater than 6
    - PRINT error message
    - GOTO LOGIN\_PASS label
  + SET password string to contents of temp
  + OPEN fileName for reading, set to data
  + Call DownloadCustomers function with data, accounts and address of numAccounts and IDnumberForNewAccount as parameters
  + CLOSE data file
  + DECLARE int i
  + LOOP through accounts to find with account matches ID number
    - IF password and ID entered matches password and IDof account stored
      * Set accountCur to account found
  + IF i is greater or equal to numAcounts
    - PRINT no account found
    - GOTO LOGIN\_ID label
  + DECLARE int choice
  + IF accountCur is admin
    - INFINITE LOOP
      * CALL DisplayMenu function with ADMIN value as parameter
      * OPEN fileName for reading, set to data
      * Call DownloadCustomers function with data, accounts and address of numAccounts and IDnumberForNewAccount as parameters
      * CLOSE data file
      * PROMPT user to enter choice
      * STORE in variable choice
      * CLEAR buffer
      * SWITCH choice
        + CASE 1

CALL CreateCustomerAccount with accounts, numAccounts, address of IDnumberForNewAccount as parameters

INCREMENT numAccounts

BREAK

* + - * + CASE 2

CALL ChangePassword with accountCurr parameter

BREAK

* + - * + CASE 3

CALL ViewCustomerInfo with accounts and numAccounts as parameters

BREAK

* + - * + CASE 4

CALL ChangeCustomerInfo with accounts and numAccounts as parameters

BREAK

* + - * + CASE 5

CALL DeleteCustomerAccount with accounts and numAccounts as parameters

BREAK

* + - * + CASE 6

CALL ShowTopFive with accounts as parameter

BREAK

* + - * + CASE 7

CALL ShowAccountsAlpha with accounts and numAccounts as parameters

BREAK

* + - * + CASE 8

PRINT Session terminated

EXIT program

* + - * + DEFAULT

PRINT not valid option

* + - * SET data to the file for writing
      * CALL UploadCustomers with data, accounts, and numAccounts as parameters
      * CLOSE data file
  + IF accountCur is customer
    - INFINITE LOOP
      * CALL DisplayMenu function with accounts status as parameter
      * OPEN fileName for reading, set to data
      * Call DownloadCustomers function with data, accounts and address of numAccounts and IDnumberForNewAccount as parameters
      * CLOSE data file
      * PROMPT user to enter choice
      * STORE in variable choice
      * SWITCH choice
        + CASE 1

CALL changePassword with accountCurr as parameter

BREAK

* + - * + CASE 2

CALL viewCustomerInfo with accountCurr as parameter

BREAK

* + - * + CASE 3

CALL viewBalence with accountCurr as parameter

BREAK

* + - * + CASE 4

CALL depositMoney with accountCurr as parameter

BREAK

* + - * + CASE 5

CALL transferMoney with accountCurr and accounts as parameter

BREAK

* + - * + CASE 6

CALL withdrawMoneywith accountCurr as parameter

BREAK

* + - * + CASE 7

PRINT error message

EXIT from program

* + - * + DEFAULT

PRINT not valid choice

* + - * SET data to the file for writing
      * CALL UploadCustomers with data, accounts, and numAccounts as parameters
      * CLOSE data file
  + RETURN 0 to main
* END main
* START DownloadCustomers function with file pointer input, Account array p, int pointer numAccounts, and int pointer accID
  + DECLARE int i
  + WHILE i is less than MAX\_CUSTOMERS
    - READ input from input file and store each field nto Account field in accounts in i
    - INCREMENT i
  + DECLARE int k
  + FOR k from 0 to MAX\_CUSTOMERS
    - IF account at k in p status doesnt equal ADMIN or CUSTOMER
      * BREAK
  + SET variable numAccounts is pointing to to value of k
  + SET variable accID is pointing to to the string to int conversion of the account ID of p at k - 1
* END DownloadCustomers
* START UploadCustomers with file pointer output, Account p array and int numAccounts as paramenters
  + DECLARE int i
  + WHILE i is less than numAccounts
    - WRITE output from output file from each spot in p into a seperate line in output
    - INCREMENT i
* END UploadCustomers
* START DisplayMenu with int mode as parameter
  + IF mode is ADMIN
    - PRINT admin options to user
  + IF mode is CUSTOMER
    - PRINT customer options to user
* END DIsplayMenu

Admin.h Pseudocode

* START header protection
* DEFINE struct Account with fields for status, first and last name, city, state, phone number, account number, password, and balance
* DEFINE function prototype for CreateCustomerAccount with Account array, int and int pointer as parameters
* DEFINE function prototype for ChangePassword with Account cur
* DEFINE function prototype for ViewCustomerInfo function with Account array and int as parameter
* DEFINE function prototype for DeleteCustomerAccount function with Account p[] and and int numAccounts as parameters
* DEFINE function prototype for ShowTopFive function with Account p[] as parameter
* DEFINE function prototype for ShowAccountsAlpha function with Account p[] and int numAccounts as parameter
* END header file protection

Admin.c Pseudo

* INCLUDE header files
* START StructCmp function with 2 void pointers a and b and int return type
  + INITIALIZE Account pointer ia to a casted to Account pointer
  + INITIALIZE Account pointer ib to b casted to Account pointer
  + RETURN casted to int: ia’s balence - ib’s balence
* START CreateCustomerAccount function
  + DECLARE string temp
  + SET status of new customer to CUSTOMER
  + PROMPT user to enter first name
  + STORE in temp
  + INPUT validate
  + STORE temp in first name field
  + PROMPT user to enter last name
  + STORE in temp
  + INPUT validate
  + STORE temp in last name
  + DO this for each field in account
* START changePassword function with Account pointer p
  + DECLARE string temp
  + PROMPT user to enter new password
  + STORE result in temp
  + INPUT validate
  + STORE temp in p.password
* END changePassword
* START ViewCustomerInfo with Account array data and int numAccounts as parameters
  + PROMPT user to enter ID num of account
  + VALIDATE input
  + LOOP through data going to numAccounts
    - IF input ID matches spot in data
      * PRINT p accounts status
      * PRINT p accounts firstname and last name
      * PRINT p accounts city
      * PRINT p accounts state
      * PRINT p accounts phone number
      * PRINT p accounts account number
      * PRINT p accounts password
      * PRINT p accounts balance
* END ViewCustomerInfo
* START DeleteCustomerAccount with Account p[] as parameter
  + PROMPT user to enter account number to delete
  + STORE in variable accountNum
  + FOR int i = 0; i < lengthOfPArray
    - If Account at i has same account number as accountNum
      * Delete Account
  + PRINT “Account deleted”
* END DeleteCustomerAccount
* START ShowTopFive function with Account data[] as parameter
  + DEFINE Account array bigs of length 5
  + DEFINE integer variable spot equals 0
  + DEFINE integer BalenceBig
  + FOR int k = 0; k < lengthOfPArray
    - IF Account at k has bigger balence than Balence big and bigs is not full
      * Add account at k to bigs
  + FOR int q = 0; q < 5
    - PRINT account at q in bigs
* END ShowTopFive
* START ShowAccountsAlpha
  + PROMPT user to enter letter of last name to sort by
  + STORE in variable letter
  + FOR int k = 0; k < lengthOfPArray
    - IF account holders last name begins with letter
      * PRINT account
* END ShowAccountsAlpha
* START ChangeCustomerInfo function with Account array data and numAccounts as parameters
  + PROMPT user to enter new info for each field, excluding password, balance, and account number
  + STORE entered values in each field of p respectively
* END ChangeCustomerInfo

Customer.h Pseudo

* DEFINE function prototype viewBalance using parameter accountCurr
* DEFINE function prototype depositMoney using parameter accountCurr
* DEFINE function prototype transferMoney using parameter accountCurr and accountSec (or p[] pointer)
* DEFINE function prototype withdrawMoney using parameter accountCurr
* DEFINE function prototype void viewAccountInfo using parameter accountCurr

Customer.c Pseudo

void viewBalance(Account\* accountCurr)

PRINT “Account Balance (accountCurr -> balance) ”

End viewBalance

void depositMoney(Account\* accountCurr)

DECLARE Double amountToDeposit

PRINT “Enter amount to deposit: “

SCAN user defined double amountToDeposit to current account

CHECK for invalid value for deposit

REPEAT if invalid

ADD amountToDeposit to accountCurr-> balance

END depositMoney

Void transferMoney (Acount\* accountCurr, Account\* p[])

DECLARE Double amountToTransfer

DECLARE int accountID

PRINT “Enter account number of second account”

SCAN user entered account number to int accountNum

CHECK that the account id is not invalid

CHECK that the account id is not the same the current user’s id

REPEAT if the id is invalid

PRINT “Enter amount to transfer:”

SCAN user defined double amountToTransfer to second account

CHECK that the amount to transfer is a positive number

CHECK that the amount is not too high

CHECK that the amount does not put the account balance as negative

REPEAT if the value is invalid

Going through every user to find the right id.

FOR int i = 0; i < AmountOfCustomers; i++

IF accountNum is equal to accounts.accountNumber

SUBTRACT amountToTransfer from accountCurr balance

ADD amountToTransfer to accountSec balance of matching account number

END IF

END FOR

END transferMoney

Void withdrawMoney (Account\* accoutCurr)

DECLARE double amountToWithdraw

PRINT “Enter amount of money to withdraw: “

SCAN user input for amountToWithdraw

CHECK that the input is positive

CHECK that the user has the amount in their account

REPEAT if invalid input

SUBTRACT amountToWithdraw from accountCurr.balance

END withdrawMoney

void viewAccountInfo(Account\* accountCurr)

PRINT First Name: , accountCurr->firstName

PRINT Last Name: , accountCurr->lastName

PRINT City: , accountCurr->city

PRINT State: , accountCurr->state

PRINT Phone Number: , accountCurr->phoneNumber

PRINT Account ID: , accountCurr->accountID

PRINT Balance: , accountCurr->balance

END viewAccountInfo