**Chapter:All**

**Assignment:** **SDEV 120 Final Project**

**Name: Makin**

**Overview:**

ATM OS ver 1.0 will provide all basic functionality for an ATM including depositing and withdrawing money, transferring money between checking and savings accounts, and checking account balances.

Users will be limited to three transactions per session. Security is accomplished by requiring a username and password. Initial program data is installed with a one-time use a side-program.

Maximum transaction amount is $500.

To initialize the ATM with current financial an customer data, there is a side-program that must be executed first called Create Table. This will create the flat-file database the main ATM program will use to save data across sessions.

Future versions of this project will allow the office computer to read and update this file so that all customer data stays in sync wither transactions happen in a branch or through the ATM.

When ATM OS 1.0 starts up, it will clear the console, declare/initialize all variables, and load all the customer data from the FinalProjectMakin.txt file.

**INPUT**:

All the relevant data is loaded via redirected input statement, as soon as the program opens, in the LoadData module.

There is a general welcome screen prompting the user to enter their username and password. They are allowed three attempts for each one. Successfully logging in will display a personalized welcome message using their name.

After that the user will see the main menu. This displays all the things they can do which includes depositing, withdrawing, or transferring money, as well as being able to check their balance. Each option will let them choose their checking or savings account. A third option is provided if they change their mind and wish to return to the main menu.

Withdrawing, depositing, transferring, and balance checking each represent a single transaction. Users are only allowed to make three transactions per visit. If they choose to return to the main menu using the third option mentioned above, this will not count as a transaction.

Input validation is very important since an error in this program could cause the bank or the user to lose money. Every time a user is asked to enter any information, we’ll check their input to make sure it is of the correct type. Negative numbers are not allowed at all. Input that requires a number will be checked to ensure there aren’t any letters and input that requires text will be checked to ensure it isn’t purely a number.

The first input from the user comes the user is greeted by the GeneralWelcome module and prompted to enter their username and password. This process is defined in the LogIn module. It runs the user’s input through the NumberCheck and NotNumberCheck modules.

The NotNumberCheck module is only used by the LogIn module. After the NumberCheck module has validated their input is, in fact, not a number that input is passed to the NotNumberCheck module. If it is a number, the NumberCheck module will continue to warn them they’ve entered the incorrect information up to three times.

The NotNumberCheck module, beginning with the Username[n], sequentially checks the entered username, stored in the variable test, against the list of allowed usernames. The variable CusIndex is used to increment the index of Username[n]. Once the password matches, CusIndex is never changed again during the user’s session and is used to align all the data for this customer.

Once the username has been established, the LogIn module moves forward to validate the format of the password in the NumberCheck module just the same as we did for the username. If valid, it proceeds to NotNumberCheck where it checks to see if it’s the correct password for that user.

As attempts are made and incorrect information is entered, show the user how many attempts they have remaining.

Three failures to enter either the password or the username will result in the LogInFail module telling the user they failed and should see a bank teller for assistance.

Once logged in, the MainMenu module writes the available transactions to the screen and calls the required modules as needed based on which option the user selects.

A “phase” dimensional variable is used to track which segment of code has called the NumberCheck module. In this case, phase[1] is set to “mainmenu”. The NumberCheck module uses various selection statements to direct the request to the section that checks to make sure the user has entered a number between 1 and 5. Any other results in telling the user their input was not allowed and cycling the loop for them to try again.

The Deposit and Withdrawal modules work essentially identically. The user chooses which account to work with. This is done in the ActionMenu module. Once an account is selected, the user enters the amount to withdraw or deposit which is handled in the NumberCheck module.

Phase[2] is defined in the Deposit, Withdrawal, and Transfer modules as deposit, withdrawal, and transfer respectively. This variable is used to alter the text displayed on the screen to the user even though the code itself is same.

For example, "How much would you like to " + phase[2] + " ($500 limit)?". If they came from the Deposit, Withdraw, or Transfer module, the value of phase[1] would be “amt” which would direct the program to ask above question.

However, if they came from the Deposit module, specifically, phase[2] would equal “deposit” so the user would actually see, How much would you like to deposit ($500 limit)?"

For the Deposit and Withdrawal modules, we also need to make sure the user isn’t trying to deposit or withdraw coins. If the test condition Amt > floor(Amt), where amount is the amount they entered to deposit or withdraw, is true, we know they have entered an amount with a decimal place. In this case, the NoCoins module is called and the user is told to use whole bill amounts.

The user may also check their balance. The output for that choice is shown in the module ShowBalance.

Transfer works the same as the Deposit and Withdraw modules but adds one question to the path- which account they want to transfer money from. Once we know that, the same pathway is used as in the other modules for the remaining input.

Lastly is the option to quit the program or to force the user out after making three transactions. TransCount is incremented after each successful transaction. Quitting can be done from the main menu by choosing option 5. In either case, phase[1] has been set to “mainmenu” and the program is directed to the appropriate section of the ActionMenu module via a series of selction statements.

The module also checks to see if TransCount is equal to 4. If it is, then the user is informed they have made 3 transactions, thanks them for using the ATM and exits. If it is not 4, we know they’ve just chose to exit and a similar message is displayed.

**PROCESS/ASSIGNMENT STATEMENTS**:

Much of the processing is mixed in with the input and output, but there are some important issues to note. NumberCheck is heavily used and part of every input gathered from the user. It ensures input is a number or is not a number as required by the calling module. It also checks for zeros, null, and negative numbers. It also enforces the $500 limit on transactions. This is the test condition when getting the amount of money the customer wants to work with: Is\_Number(Amt) AND ABS(Amt) = Amt AND Amt <= 500 AND Amt != 0. It requires the input to be a positive number and under $500.

The HasEnoughMoney module checks to see if the user has enough money to withdraw or transfer from the required account. It uses a Boolean variable of true when they do have enough money and false when they don’t.

Back in the calling module, a pre-test condition is used to determine how to proceed: NOT HasEnoughMoney = true OR Menu = 3

If Menu equals 3, the user has chosen to go back to the main menu and no further output is required. If HasEnoughMoney is false, meaning they don’t have enough money to do the transaction, the NOT statement is true and no further output is required since that was already handled in the HasEnoughMoney module.

**OUTPUT**:

Most of the output is distributed through the various modules and has already been explained but there are still some actions to point out.

The UserWelcome module greets the customer personally once they’ve successfully logged in.

The ShowBalance module is called when the user wants to specifically see their balance, but is also used by the Deposit, Withdrawal, and Transfer modules to show the balance after each transaction.

The UpdateServer modules uses a series of redirect output statements and a loop to output all the data, whether it’s been changed or not, back out to the flat-file database. This better allows for future software upgrades to add functionality such as allowing users to change their passwords.

Major output screens, such as the main menu, transaction menus, etc. are formatted with a row of asterisks on the top and bottom. In between these delineators, all lines start with an asterisk followed by two spaces. No asterisks are used at the end of lines because line length can vary. The resulting line down the right side looks very disjointed and sloppy so we’re not using it at all.

There is a requirement to pause and allow a user to read the output provided and proceed at their own process. This is accomplished using a dummy variable called “nothing”. Users will be prompted to click the OK button to continue. While that does collect data, it isn’t used and this is strictly in place to facilitate the user reading the output. This is the one place where no error checking is used.

**Variable List**:

Count, INT, counter for loops

Phase[2], STRING, used to direct input to specific sections of modules

Amt, FLOAT, the amount of money involved in a transaction

test, STRING, username and password input before it’s ben validated

CusIndex, INT, used to coordinate parallel array data

Menu, INT, menu selections from the main menu and transaction menus

TransCount, INT, tranks the number of transactions that have been made in a session

HasEnoughMoney, BOOLEAN, tracks whether enough money exists for a transaction

Customer[4], STRING, the customer’s name

Username[4], STRING, the customer’s username

Password[4], STRING, the customer’s password

SavBal[4], FLOAT, the customer’s savings account balance

CheBal[4], FLOAT, the customer’s checking account balance

**Modules**:

Main, controls top-level flow of program

Declare, declares and initializes all variables

LoadData, assigns all the data from the flat-file database to parallel arrays in the program

GeneralWelcome, opening screen asking to enter username and password

UserWelcome, a more personal greeting including the customer’s name

LogIn, controls the general process flow of validating and verifying log in input

LogInFail, output section informing the user they failed to log in

NumberCheck, validates input as being or not being a number, collects menu selection input, collects transaction amounts

NotNumberCheck, once validated as not being a number, determines if the username and password are correct and establishes CusIndex

ActionMenu, generates output when a user returns to the main menu, completes three transactions, or quits the program, and displays the prompt to enter an amount for a transaction

MainMenu, provides choices for available transaction types or to quit the program

Deposit, controls deposit transactions

Withdrawal, controls withdrawal transactions

Transfer, controls transferring money between accounts

ShowBalance, output that shows the available balance of both accounts

HasEnoughMoney, determines if accounts have a sufficient balance to support the chosen transaction

NoCoins, output telling the user they must use whole bill amounts

UpdateServer, sends all the data back out to the flat-file database to keep track of changes between sessions