

## CSCE 1030: Project 3

**Due: 11:59 PM on Sunday, April 18, 2021**

### PROGRAM DESCRIPTION

In this project, you have to write a C++ program to keep track of banking transactions in multiple disk files.

Your objective is to get transactions from a user and process the transactions for debiting or crediting the account. Each user holds two accounts – a business account and a personal account.

You are provided the following files. Download them and view their contents.

**account\_data**: This file contains the account name and the account number. The name and the number are separated by a comma.

**145268p**: This file contains the personal transactions made by the account number 145268. If you look at the account\_data.dat file, you will see this account belongs to John Smith.

**145268b**: This file contains the business transactions made by the same account.

You may choose to create your own transaction files with arbitrary data to test your program for different accounts.

This is an extension of Projects 1 and 2 so you may reuse any work you have done for Projects 1 and 2.

### PROGRAM REQUIREMENTS

1. As with all projects in this course, your program's output will display your name, your EUID, your e-mail address, the department name, and course number. This means that your program will print this information to the terminal (see the sample output).
2. Declare and initialize the following global parameters.
  - A global integer constant to store the length of account number and initialize it to 6.
  - An enumeration constant with values **Business** and **Personal** and assign integer values 0 and 1 to the data items, respectively. They represent the type of bank account.
  - Another enumeration constant with values **Add**, **Remove**, **Process**, **Display**, and **Quit**, and assign integer values 1, 2, 3, 4 and 5 to the data items, respectively. They represent menu choice presented to the user.
  - A structure named **Account** with the following members.
    - Name of the account
    - Number of the account
    - Type of the Account (i.e. Business or Personal)
      - You must use a suitable enum type variable for this member.
3. In your **main** function:
  - Display a menu for the user (SEE SAMPLE OUTPUT) that provides the user the following choices.
    - Add a new account
    - Remove an existing account
    - Process new transaction on an account
    - Display the stored transactions on an account.
    - Quit the program

- Using a suitable message, ask the user to make the menu choice using an integer variable.
  - Based on the value entered by the user for menu choice, design a switch-case block to implement the following requirements.
    - You must use the enumeration constants to set up your cases.
    - You must use a variable of your enumeration constant type for switching control.
    - If the user chooses to add an account
      - Call the function **addAccount**
    - If the user chooses to remove an account
      - Using a suitable message, ask the user the account number of the account to remove.
      - Call the function **removeAccount** and pass the account number entered by the user.
    - If the user chooses to process transactions for an account
      - Using a suitable message, ask the user the account number of the account to process transactions for.
      - Call the function **processAccount** and pass the account number entered by the user.
    - If the user chooses to display transactions for an account
      - Using a suitable message, ask the user the account number of the account to display transactions for.
      - Call the function **displayAccount** and pass the account number entered by the user.
    - If the user choose to quit the program.
      - Exit the program with a suitable message.
    - If the user chooses anything else, execute the default case to notify the user an incorrect choice.
      - Using a suitable loop, ask the use for the choice again.
      - Your program must keep on looping until the user enters the correct choice
4. Write a function named **getName** which gets the name on the bank account. Inside the function:
- Using a suitable message, prompt the user for the name on the account. The name can have multiple words.
  - Only alphabets (A-Z or a-z) and whitespaces are permitted in the account name.
    - If the user enters any other characters in the name, you need to generate an error message and ask for the name again.
    - Your program must keep on asking the user to enter the name until the user enters it correctly.
  - The user may type the name in either uppercase or lowercase, but you need to convert every initial to uppercase.
  - This function will be called by the **addAccount** function.
5. Write a function named **getAccountNumber** which get the account number. Inside the function:
- Using a suitable message, prompt the user for the number of the account.
  - The account number must be a 6-digit number.
  - If the user enters an account number with more than 6 digits generate an error message and ask the user to enter the number again.

- Only numbers 0-9 are permitted in the account number. If the user enters an account number with non-numeric characters, generate an error message, and ask the user to enter the number again.
  - Your program must keep on asking the user to enter the number until the user enters it correctly.
  - This function will be called by the **addAccount** function.
6. Write a function named **getNumberOfTransactions**. It accepts a string that stores the filename of a disk file and return an integer representing the number of transactions in the disk file.
- Inside the function, open the file for reading using the filename passed to the function.
  - Count the number of transactions in the file.
  - For example, the file **145268b** has **7** transactions. Hence the name of the file **145268b** has to be passed to this function and it has to return **7**. The return value will change after you edit the file.
7. Write a function named **getNumberOfAccounts**. It accepts a string that stores the filename of a disk file and return an integer representing the number of accounts in the disk file.
- Inside the function, open the file for reading using the filename passed to the function.
  - Count the number of accounts in the file.
  - For example, the file **account\_data** has **4** accounts. Hence the name of the file **account\_data** has to be passed to this function and it has to return 4. The return value will change after you edit the file.
8. Write a function named **addAccount**. It does not have any parameters.
- Inside the function, open the **account\_data** file for appending.
  - Call the function **getName** to get the name of the new account.
  - Call the function **getNumber** to get the number of the new account.
  - Add the new account information to the data file in the same **format (SEE SAMPLE OUTPUT)**.
9. Write a function named **removeAccount**. It accepts the account number to be removed.
- Inside this function, call function **getNumberOfAccounts** to get the current number of accounts in the data file **account\_data**.
  - Create a dynamic array of the structure type **Account**. Use the number of accounts in the data file for size of this dynamic array.
  - Open **account\_data.dat** file for reading.
  - In a loop, read one line of the file at a time and store appropriate data in the dynamic array. Note that each element of the structure array will be an account available on a line of the file.
    - While reading check if the account number being read from the file matches the account number to be removed.
    - If you find match, use a Boolean flag to store that you have found a match.
    - Copy the entire file to your dynamic array whether or not you find a match.
  - Close the file.
  - If you have found a match while reading:
    - Open **account\_data.dat** file for writing.
    - Write the contents of the dynamic array into the file, only if the account number does not match the account number to be removed.

- If you have not found a match while reading:
    - Notify the user with a suitable message to indicate the account number does not exist.
  - Close the file.
10. Write a function named **getCurrentBalance**.
- This function accepts two parameters: the account number and the type of account to get the balance of. The account type must be an enum data type.
  - This function returns the current balance of the account being read.
  - Inside the function, open the appropriate file for reading. For example, open the file **145268b** if the account number is **145268** and the account type is **Business**.
  - Inside a loop, read the transactions in the file to find their sum.
  - Return the computed sum representing the current account balance.
11. Write a function named **writeTransactions**.
- This function accepts two parameters: the account number and the type of account to get the balance of. The account type must be an enum data type.
  - Inside the function, open the appropriate file for appending. For example, open the file **145268b** if the account number is **145268** and the account type is **Business**.
  - Using a suitable message, ask the user for the transaction value.
  - Write the transaction value to the appropriate file.
12. Write a function named **displayAccount**. It accepts the account number of the account to be display the transactions for.
- Inside this function:
    - Using a suitable message, ask the user which account needs to be displayed – Business or Personal. Use an integer variable for this purpose.
    - Using a suitable message, ask the user if the display needs to be sorted. Use a character variable for this purpose.
    - Inside the function, call the function **getNumberOfTransactions** with appropriate arguments to get the number of transactions in the appropriate file. For example, if the account number is **145268** and the account type is **Business**, you must get the number of transactions from the file **145268b**.
    - Create a dynamic array of double type using the number of transactions in the file for the size of the array. You need an array since you may need to sort the transactions.
    - Inside the function, open the appropriate file for reading.
    - Using a loop, read the content of the file one item at a time and store in your dynamic array.
    - In a different loop, display the content of the array.
      - Your display must be sorted if the user has chosen to do so.
      - Your numeric data must have two numbers after the decimal point.
13. Write a function named **process\_account**. It accepts the account number of the account to process the transactions for. Inside this function:
- Ask the user to choose which account the user wants to access – Business or Personal. Use a suitable integer value to get the choice from the user.
  - Based on the choice of the user, design a switch case block to implement the following requirements.

- You must use the enumeration constants to set up your cases.
  - You must use a variable of your enumeration constant type for switching control.
  - If the user chooses a Business account
    - Call the function **writeTransaction** with appropriate arguments.
    - Call the function **getCurrentBalance** with appropriate arguments.
  - If the user chooses a Personal account
    - Call the function **writeTransaction** with appropriate arguments.
    - Call the function **getCurrentBalance** with appropriate arguments.
  - If the user enters a wrong choice, use the default case to provide an error message and ask the user to make the choice again.
    - Your program needs to keep on asking the user for the choice until the user chooses a correct choice.
  - Your numeric data must have two numbers after the decimal point.
- This function needs be able to process more than one transaction.
    - After successfully processing a transaction, ask the user if the user wants to process another transaction.
    - If the user chooses to process another transaction, use a suitable loop to ask the user about the type of account and the transaction to process.
14. Now that you've written and tested your program, make the following changes.
- Create a file named **euidProject3\_main.cpp** and copy the main function to this file.
  - Create a file named **euidProject3\_func.cpp** and copy all the functions (other than the main function) to this file.
  - Create a file named **euidProject3\_header.cpp** and copy all the global parameters as well as include fields and function headers to the header file. Make sure you include the header file in the .cpp files.
  - Compile the files together and make sure it works correctly.
  - You need to submit these three files to Canvas, where euid should be replaced by your EUID.
15. Your program will be graded based largely on whether it works correctly on the CSE machines (e.g., cse01, cse02, ..., cse06), so you should make sure that your program compiles and runs on a CSE machine.

## DESIGN (ALGORITHM):

On a piece of paper (or word processor), write down the algorithm, or sequence of steps, that you will use to solve the problem. You may think of this as a “recipe” for someone else to follow. Continue to refine your “recipe” until it is clear and deterministically solves the problem. Be sure to include the steps for prompting for input, performing calculations, and displaying output.

You should attempt to solve the problem by hand first (using a calculator as needed) to work out what the answer should be for a few sets of inputs.

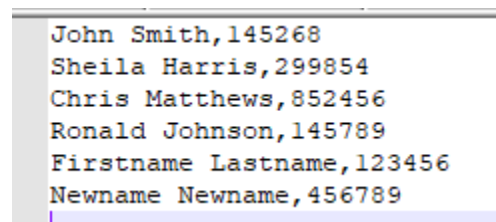
Type these steps and calculations into a document (i.e., Word, text, or PDF) that will be submitted along with your source code. Note that if you do any work by hand, images (such as pictures) may be used, but they must be clear and easily readable. This document shall contain both the algorithm and any supporting hand-calculations you used in verifying your results.

Here are some sample outputs to help you write the code. The items in bold are entered by the user on the terminal as input data.

### SAMPLE OUTPUT 1

```
$ ./a.out
+-----+
|           Computer Science and Engineering           |
|           CSCE 1030 - Computer Science I             |
|           Student Name EUID euid@my.unt.edu           |
+-----+
1. Add Account
2. Remove Account
3. Process Accounts
4. Display Account Information
5. Quit
Enter your choice:1
Enter your name:Firstname lastname
Enter your account number:123456
1. Add Account
2. Remove Account
3. Process Accounts
4. Display Account Information
5. Quit
Enter your choice:1
Enter your name:newname newname
Enter your account number:456789
1. Add Account
2. Remove Account
3. Process Accounts
4. Display Account Information
5. Quit
Enter your choice:5
Goodbye!!!
```

### Screenshot of the account\_data file after adding accounts



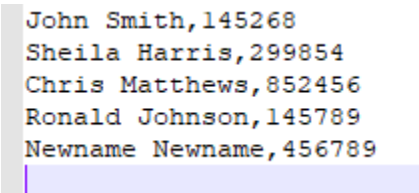
A screenshot of a text file named 'account\_data'. The file contains six lines of text, each representing an account. The first four lines are 'John Smith,145268', 'Sheila Harris,299854', 'Chris Matthews,852456', and 'Ronald Johnson,145789'. The next two lines are 'Firstname Lastname,123456' and 'Newname Newname,456789'. The text is displayed in a monospaced font with a light blue background.

```
John Smith,145268
Sheila Harris,299854
Chris Matthews,852456
Ronald Johnson,145789
Firstname Lastname,123456
Newname Newname,456789
```

## SAMPLE OUTPUT 2

```
$ ./a.out
+-----+
|           Computer Science and Engineering           |
|           CSCE 1030 - Computer Science I             |
|           Student Name EUID euid@my.unt.edu           |
+-----+
1. Add Account
2. Remove Account
3. Process Accounts
4. Display Account Information
5. Quit
Enter your choice:2
Enter account number to remove:000000
Account does not exist.
1. Add Account
2. Remove Account
3. Process Accounts
4. Display Account Information
5. Quit
Enter your choice:2
Enter account number to remove:123456
1. Add Account
2. Remove Account
3. Process Accounts
4. Display Account Information
5. Quit
Enter your choice:5
Goodbye!!!
```

## Screenshot of the account\_data file after removing account number 123456



```
John Smith,145268
Sheila Harris,299854
Chris Matthews,852456
Ronald Johnson,145789
Newname Newname,456789
```

## SAMPLE OUTPUT 3

```
$ ./a.out
+-----+
|           Computer Science and Engineering           |
|           CSCE 1030 - Computer Science I             |
|           Student Name EUID euid@my.unt.edu           |
+-----+
1. Add Account
2. Remove Account
3. Process Accounts
```

```

4. Display Account Information
5. Quit
Enter your choice:3
Enter account number to process:145268
What is your account type?0 for Business, 1 for Personal:0
Enter transaction:15000
Business Balance:$44581.15
Do you want to process another transaction? Y/N:y
What is your account type?0 for Business, 1 for Personal:1
Enter transaction:5000
Personal Balance:$27999.75
Do you want to process another transaction? Y/N:n
1. Add Account
2. Remove Account
3. Process Accounts
4. Display Account Information
5. Quit
Enter your choice:5
Goodbye!!!

```

#### Screenshot of the 145268p file after entering transaction

```

3000.00
5000.00
1000.50
-4000.75
10000
5000
1000
-5000
5000
2000
5000

```

#### Screenshot of the 145268b file after entering transaction

```

12000.75
-3000.85
14500.50
2580.75
500
5000
-2000
15000

```

#### SAMPLE OUTPUT 4

```

$ ./a.out
+-----+
|           Computer Science and Engineering           |
|           CSCE 1030 - Computer Science I             |

```



```

|           Student Name EUID euid@my.unt.edu           |
+-----+
1. Add Account
2. Remove Account
3. Process Accounts
4. Display Account Information
5. Quit
Enter your choice:4
Enter account number:145268
Which account to display? 0 for Business, 1 for Personal:0
Do you want to sort? Y/N:n
12000.75
-3000.85
14500.50
2580.75
500.00
5000.00
-2000.00
1. Add Account
2. Remove Account
3. Process Accounts
4. Display Account Information
5. Quit
Enter your choice:4
Enter account number:145268
Which account to display? 0 for Business, 1 for Personal:1
Do you want to sort? Y/N:y
10000.00
5000.00
5000.00
5000.00
3000.00
2000.00
1000.50
1000.00
-4000.75
-5000.00
1. Add Account
2. Remove Account
3. Process Accounts
4. Display Account Information
5. Quit
Enter your choice:5
Goodbye!!!

```

### TESTING:

Test your program to check that it operates as desired with a variety of inputs. Then, compare the answers your code gives with the ones you get from hand calculations.

### SUBMISSION:

- Your program will be graded based largely upon whether it works correctly on the CSE machines, so you should make sure your program compiles and runs on the CSE machines.
- Your program will also be graded based upon your program style. This means that you should use comments (as directed), meaningful variable names, and a consistent indentation style as recommended in the textbook and in class.
- We will be using an electronic homework submission on Canvas to make sure that all students hand their programming projects on time. You will submit four items -- (1) the two .cpp code files, (2) the header file, and (3) the algorithm design document -- to the **Project 3** dropbox on Canvas by the due date and time.
- Homework are meant to be problem-solving exercises and are designed to help you practice your coding on larger projects with various pieces of functionality. While the coding should be primarily your sole work, you are allowed to get assistance from classmates when working on these assignments. However, each student is required to report the name(s) of the students they worked with on the assignment. Cheating for these assignments is now defined as copying from a fellow student without reporting it or copying from the web. You should not copy someone else's code or let a classmate examine your code if you have not identified as working in a group for your homework.
- As a safety precaution, do not edit your program (using vim or nano) after you have submitted your program where you might accidentally re-save the program, causing the timestamp on your file to be later than the due date. If you want to look (or work on it) after submitting, make a copy of your submission and work on that copy. Should there be any issues with your submission, this timestamp on your code on the CSE machines will be used to validate when the program was completed.