CIS 200 - Lab/Project

1. Problem Statement

Given a struct clientData, build a program that can fill a file with instances of this data and read it back.

2. Requirements

2.1 Assumptions

- Some of the code is provided on the “instruction” sheet.
- Command Line Input Only
- User will only enter integers where applicable
- User will only enter doubles where applicable

2.2 Specifications

- Struct clientData will be used
- File will be accessed randomly
- File will accommodate 100 user inputs
- User can request specific accounts
- Program will print out all used accounts in order

3. Decomposition Diagram

- Program
    - Input
        - Integer Account Number
        - Character Array First Name
        - Character Array Last Name
        - Double Account Balance
    - Process
        - Create Default File
        - Push Respective Data to Output Stream
    - Output
        - Output Default File
        - Push Output Stream to File
        - Print Data to Console

4. Test Strategy

- Valid Data

- Invalid Data

5. Test Plan Version 1

Test Strategy	#	Description	Input	Expected Output	Actual Output	Pass/Fail
Valid Data	1	File Exists				
Valid Data	2	Set Account #				
Valid Data	3	Set Account #				
Valid Data	4	Set Account First Name				
Valid Data	5	Set Account Last Name				
Valid Data	6	Set Balance				
Valid Data	7	Set Balance				
Invalid Data	1	Set Account #				
Invalid Data	2	Set Account #				
Invalid Data	3	Set First Name Too Large				
Invalid Data	4	Set Last Name Too Large				
Invalid Data	5	Set Balance Negative				
Invalid Data	6	Asking for unused account				

6. Initial Algorithm

1. Create Struct clientData
    a. Integer accountNumber
    b. Character Array lastName[15]
    c. Character Array firstName[10]
    d. Float balance

```
2. Create Function createBlankFile
    a. Parameters: clientData blankClient
    b. Return: Void
    c. Method:
        i. Open Output Stream
        ii. Name File “client.dat”
        iii. Iterate 100 Times
            1. Fill with blankClient
        iv. Close Output Stream
3. Create Function writeToFile
    a. Parameters: clientData client
    b. Return: Void
    c. Method:
        i. Open Output Stream
        ii. Do While Account Number is not 0
            1. Prompt for initial Account Number
            2. IF 0
                a. Exit Function
            3. Else IF 1 >= Account Number >= 100
                a. Fill First Name
                b. Fill Last Name
                c. Fill Balance
                d. Seek Position of Account Number in file
                e. Write client data to file
            4. Else
                a. Alert Invalid Account Number
4. Create Function userRead
    a. Parameters: clientData client
    b. Return: Void
    c. Method:
        i. Create Input Stream
        ii. Do While
            1. Prompt for Account Number
            2. IF input = 0
                a. Return/Exit
            3. Else IF 1  <= input <= 100
                a. Seek Input
                b. Read Input
                c. IF Input != 0
                    i. Output Data
                d. Else
                    i. Alert Invalid Account Number
5. Create Function printAllRecords
```

```
        a. Parameters: clientData client
        b. Return: Void
        c. Method:
            i. Create Input Stream
            ii. Read Initial Data
            iii. While inCredit && !inCredit @ End Of File
                1. IF client.accountNumber != 0
                    a. Output Data
                2. Read Next Data
    6. Create MAIN
        a. Create and Initialize client and blankClient as {0,””,””,0.0}
        b. Perform createBlankFile(blankClient)
        c. Perform writeToFile(client)
        d. Perform userRead(client)
        e. Perform printAllRecords(client)
        f. Return 0;
```

7. Test Plan Version 2

Test Strategy	#	Description	Input	Expected Output	Actual Output	Pass/Fail
Valid Data	1	File Exists	createBlankFile	File Output Correct		
Valid Data	2	Set Account #	1	Account #1		
Valid Data	3	Set Account #	100	Account #100		
Valid Data	4	Set Account First Name	Bill	First Name: Bill		
Valid Data	5	Set Account Last Name	Gates	Last Name: Bill		
Valid Data	6	Set Balance	1234.56	Balance: 1234.56		
Valid Data	7	Set Balance	1	Balance: 1		
Invalid Data	1	Set Account #	-1	Out of Range		
Invalid Data	2	Set Account #	101	Out of Range		
Invalid Data	3	Set First Name Too Large	abcdefghijk	First Name Corrupted		

Invalid Data	4	Set Last Name Too Large	abcdefghijklmnop	Last Name Corrupted		
Invalid Data	5	Set Balance Negative	-100	Try Again		
Invalid Data	6	Asking for unused account	1 when unused	Account Not In Use		

8. Code

[source.cpp]

```
//Program Name: Client Data - RANDOM ACCESS
//Programmer Name: Arthur Aigeltinger IV
//Description: A toe dipped in the water of random file access that is still a little
over my head but I'm glad I made it this far. Thanks for coming to my TedTalk...
//Date Created: 10/29/18

#include <fstream>
#include <iostream>
#include <ostream>

//Define Struct
struct clientData
{
	int accountNumber;
	char lastName[15];
	char firstName[10];
	float balance;
};

//Function Prototypes
void createBlankFile(clientData);
void writeToFile(clientData);
void userRead(clientData);
void printAllRecords(clientData);

int main()
{

	//Initialize Blank and Holder Client
	clientData blankClient = { 0,"","",0.0 };
	clientData client = { 0,"","",0.0 };

	createBlankFile(blankClient);
	writeToFile(client);
	userRead(client);
	printAllRecords(client);

	system("pause");
	return 0;
}

//Description: Creates and Initialized a "blank" file with the size and space to fit
100 instances of struct [clientData]
//Pre-Condition: We need a file
//Post-Condition: WE'VE GOT A FILE
void createBlankFile(clientData blankClient)
{
	//Create and Open Output Stream
	std::ofstream outCredit("credit.dat", std::ios::out);

	//Fill File with blankClient
	for (int i = 0; i < 100; i++)
	{
		outCredit.write(reinterpret_cast<const char *>(&blankClient),
sizeof(clientData));
	}

	//Close File Stream
	outCredit.close();
}

//Description: Enters loop to ask user for client data to be pushed to the
pre-generated file
//Pre-Condition: File is at its default, empty state
//Post-Condition: File is filled with individual user accounts as determined bu the
user
void writeToFile(clientData client)
{
	//Open Client Output Stream
	std::ofstream outCredit1("credit.dat", std::ios::ate);

	//Announce Function to User
	std::cout << std::endl;
	std::cout << "Inputting Specific User Data" << std::endl;
	std::cout << std::endl;

	do //Filling Client Data until accountNumber = 0
	{
		//Initial Prompt
		std::cout << std::endl;
		std::cout << "Please Enter an Account Number (1-100)" << std::endl;
		std::cout << "Entering 0 Will Exit User Input" << std::endl;
		std::cout << "Account Number: ";
		std::cin >> client.accountNumber;

		if (client.accountNumber == 0)
		{
			outCredit1.close(); //Close Output Stream
			return; //Exit Function
		}
		else if (client.accountNumber >= 1 && client.accountNumber <= 100)
		{
			//Fill In Order
			std::cout << "First Name: ";
			std::cin >> client.firstName;
			std::cout << "Last Name: ";
			std::cin >> client.lastName;
			do
			{
				std::cout << "Balance: ";
				std::cin >> client.balance;
			} while (client.balance < 0);

			outCredit1.seekp((client.accountNumber - 1) *
sizeof(clientData));
			outCredit1.write(reinterpret_cast<const char*>(&client),
sizeof(clientData));
		}
		else
		{
			std::cout << "Invalid Account Number" << std::endl;
		}
	} while (client.accountNumber != 0); //Checking to repeat even though case [0]
is handled with initial "if"
}

//Description: Allows user to request specific client accounts and have them printed
to the screen
//Pre-Condition: Filled client data file via writeToFile()
//Post-Condition: Client account(s) requested are printed to console
void userRead(clientData client)
{
	//Holder Variable
	int userIn = 0;

	//Open Input Stream
	std::ifstream inCredit("credit.dat", std::ios::in);

	//Announce Function to User
```

```
        std::cout << std::endl;
        std::cout << "Request Specific User Data" << std::endl;
        //std::cout << std::endl;

        do
        {
                std::cout << std::endl;
                std::cout << "Enter 0 To Exit" << std::endl;
                std::cout << "Enter Account Number: ";
                std::cin >> userIn;

                if (userIn == 0)
                {
                        std::cout << "Exiting" << std::endl;
                        inCredit.close(); //Close Input Stream
                        return; //Exit Function
                }
                else if (userIn >= 1 && userIn <= 100)
                {
                        inCredit.seekg((userIn - 1) * sizeof(clientData));
                        inCredit.read(reinterpret_cast<char *>(&client),
sizeof(clientData));

                                if (client.accountNumber == userIn)
                                {
                                        //Output In Order
                                        std::cout << "Account Number: ";
                                        std::cout << client.accountNumber << std::endl;
                                        std::cout << "First Name:     ";
                                        std::cout << client.firstName << std::endl;
                                        std::cout << "Last Name:      ";
                                        std::cout << client.lastName << std::endl;
                                        std::cout << "Balance:       $";
                                        std::cout << client.balance << std::endl;
                                }
                                else //Checking for Invalid Within Range
                                {
                                        std::cout << "ACCOUNT NUMBER NOT IN USE" <<
std::endl;
                                        client.accountNumber = -1;
                                }
                }
                else //Checking for Invalid Outside Range
                {
                        std::cout << "ACCOUNT NUMBER OUT OF RANGE" << std::endl;
                        client.accountNumber = -1;
                }
```

```
	} while (client.accountNumber != 0);  //Checking to repeat even though case
[0] is handled with initial "if"
}

//Description: Prints all of the filled records within "credit.dat"
//Pre-Condition: A valid, filled "credit.dat" file
//Post-Condition: All filled client data is printed on the console
void printAllRecords(clientData client)
{
	//Open Input Stream
	std::ifstream inCredit("credit.dat", std::ios::in);

	//Announce Function to User
	std::cout << std::endl;
	std::cout << "Printing ALL RECORDS" << std::endl;
	std::cout << std::endl;

	inCredit.read(reinterpret_cast<char *>(&client), sizeof(clientData));

	while (inCredit && !inCredit.eof())
	{
		if (client.accountNumber != 0) //MAY CAUSE INFINITE LOOP (TRY 100?)
		{
			//Output In Order
			std::cout << "Account Number: ";
			std::cout << client.accountNumber << std::endl;
			std::cout << "First Name:     ";
			std::cout << client.firstName << std::endl;
			std::cout << "Last Name:      ";
			std::cout << client.lastName << std::endl;
			std::cout << "Balance:       $";
			std::cout << client.balance << std::endl << std::endl;
		}

		inCredit.read(reinterpret_cast<char *>(&client), sizeof(clientData));
	}

}
```

9. Updated Algorithm

1. Create Struct clientData
    - *a.* Integer accountNumber
    - *b.* Character Array lastName[15]
    - c. Character Array firstName[10]
    - d. Float balance

```
2. Create Function createBlankFile
    a. Parameters: clientData blankClient
    b. Return: Void
    c. Method:
        i. Open Output Stream
        ii. Name File “client.dat”
        iii. Iterate 100 Times
            1. Fill with blankClient
        iv. Close Output Stream
3. Create Function writeToFile
    a. Parameters: clientData client
    b. Return: Void
    c. Method:
        i. Open Output Stream
        ii. Do While Account Number is not 0
            1. Prompt for initial Account Number
            2. IF 0
                a. Exit Function
            3. Else IF 1 >= Account Number >= 100
                a. Fill First Name
                b. Fill Last Name
                c. Fill Balance
                d. Seek Position of Account Number in file
                e. Write client data to file
            4. Else
                a. Alert Invalid Account Number
4. Create Function userRead
    a. Parameters: clientData client
    b. Return: Void
    c. Method:
        i. Create Input Stream
        ii. Do While
            1. Prompt for Account Number
            2. IF input = 0
                a. Return/Exit
            3. Else IF 1  <= input <= 100
                a. Seek Input
                b. Read Input
                c. IF Input != 0
                    i. Output Data
                d. Else
                    i. Alert Invalid Account Number
                    ii. accountNumber = -1
                4. Else
                5. Alert Account Number Out Of Range
                6. accountNumber = -1
5. Create Function printAllRecords
    a. Parameters: clientData client
    b. Return: Void
    c. Method:
        i. Create Input Stream
        ii. Read Initial Data
        iii. While inCredit && !inCredit @ End Of File
            1. IF client.accountNumber != 0
                ~~a. Output Data~~
                b. Output Account Number
                c. Output First Name
                d. Output Last Name
                e. Output Balance
            2. Read Next Data
6. Create MAIN
    a. Create and Initialize client and blankClient as {0,””,””,0.0}
    b. Perform createBlankFile(blankClient)
    c. Perform writeToFile(client)
    d. Perform userRead(client)
    e. Perform printAllRecords(client)
    f. Return 0;
```

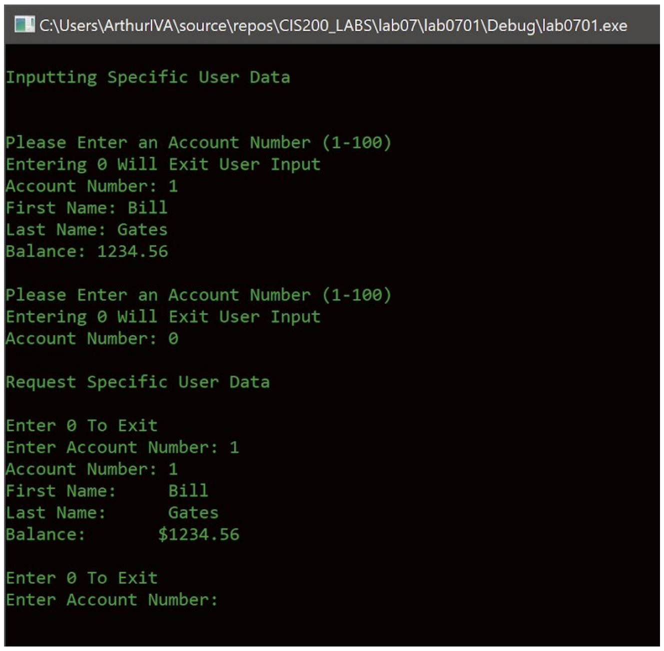
10. Test Plan Version 3

Test Strategy	#	Description	Input	Expected Output	Actual Output	Pass/ Fail
Valid Data	1	File Exists	createBlankFile	File Output Correct	File Output Correct	Pass
Valid Data	2	Set Account #	1	Account #1	Account #1	Pass
Valid Data	3	Set Account #	100	Account #100	Account #100	Pass
Valid Data	4	Set Account First Name	Bill	First Name: Bill	First Name: Bill	Pass
Valid Data	5	Set Account Last Name	Gates	Last Name: Bill	Last Name: Bill	Pass

Valid Data	6	Set Balance	1234.56	Balance: 1234.56	Balance: 1234.56	Pass
Valid Data	7	Set Balance	1	Balance: 1	Balance: 1	Pass
Invalid Data	1	Set Account #	-1	Out of Range	Out of Range	Pass
Invalid Data	2	Set Account #	101	Out of Range	Out of Range	Pass
Invalid Data	3	Set First Name Too Large	abcdefghijk	First Name Corrupted	First Name Corrupted	Pass
Invalid Data	4	Set Last Name Too Large	abcdefghijklmn op	Last Name Corrupted	Last Name Corrupted	Pass
Invalid Data	5	Set Balance Negative	-100	Try Again	Try Again	Pass
Invalid Data	6	Asking for unused account	1 when unused	Account Not In Use	Account Not In Use	Pass

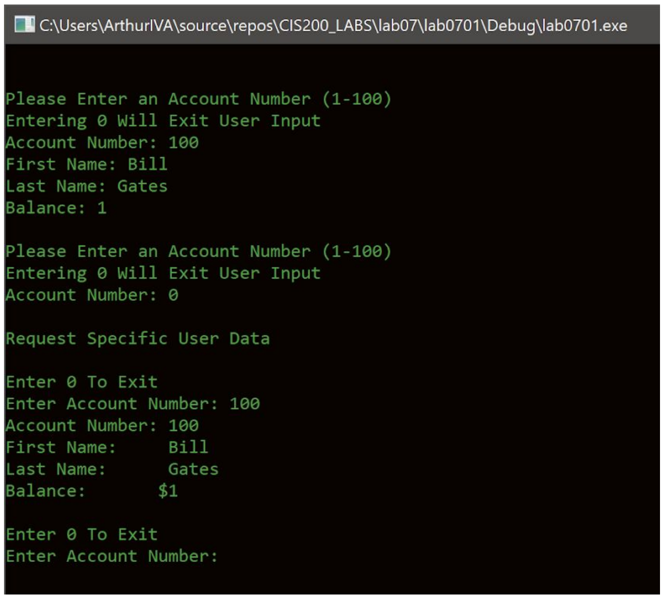
11. Screenshots

Valid 1, 2, 4, 5, 6.



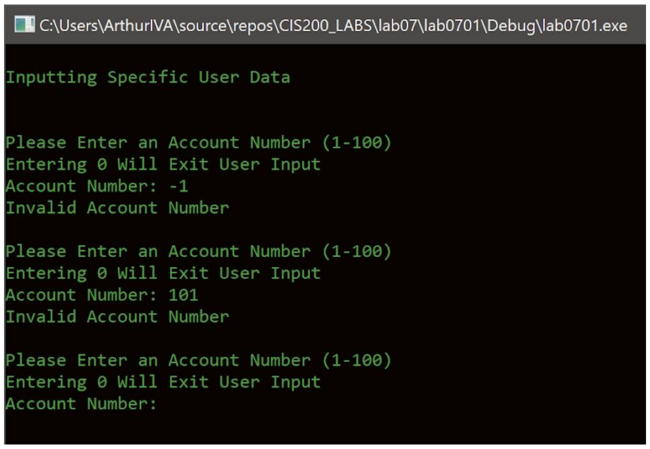
```
C:\Users\ArthurIVA\source\repos\CIS200_LABS\lab07\lab0701\Debug\lab0701.exe

Inputting Specific User Data


Please Enter an Account Number (1-100)
Entering 0 Will Exit User Input
Account Number: 1
First Name: Bill
Last Name: Gates
Balance: 1234.56

Please Enter an Account Number (1-100)
Entering 0 Will Exit User Input
Account Number: 0

Request Specific User Data

Enter 0 To Exit
Enter Account Number: 1
Account Number: 1
First Name:     Bill
Last Name:      Gates
Balance:       $1234.56

Enter 0 To Exit
Enter Account Number:
```

Valid 3, 4, 5, 7.



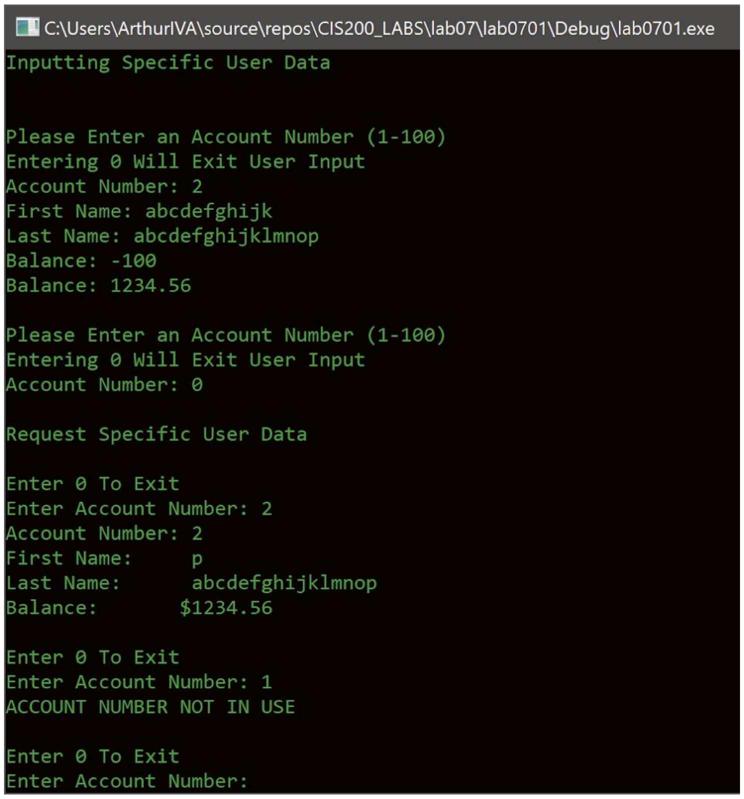
```
C:\Users\ArthurIVA\source\repos\CIS200_LABS\lab07\lab0701\Debug\lab0701.exe


Please Enter an Account Number (1-100)
Entering 0 Will Exit User Input
Account Number: 100
First Name: Bill
Last Name: Gates
Balance: 1

Please Enter an Account Number (1-100)
Entering 0 Will Exit User Input
Account Number: 0

Request Specific User Data

Enter 0 To Exit
Enter Account Number: 100
Account Number: 100
First Name:     Bill
Last Name:      Gates
Balance:       $1

Enter 0 To Exit
Enter Account Number:
```

Invalid 1, 2.



```
C:\Users\ArthurIVA\source\repos\CIS200_LABS\lab07\lab0701\Debug\lab0701.exe

Inputting Specific User Data


Please Enter an Account Number (1-100)
Entering 0 Will Exit User Input
Account Number: -1
Invalid Account Number

Please Enter an Account Number (1-100)
Entering 0 Will Exit User Input
Account Number: 101
Invalid Account Number

Please Enter an Account Number (1-100)
Entering 0 Will Exit User Input
Account Number:
```

Invalid 3, 4, 5, 6.



```
C:\Users\ArthurIVA\source\repos\CIS200_LABS\lab07\lab0701\Debug\lab0701.exe
Inputting Specific User Data


Please Enter an Account Number (1-100)
Entering 0 Will Exit User Input
Account Number: 2
First Name: abcdefghijk
Last Name: abcdefghijklmnop
Balance: -100
Balance: 1234.56

Please Enter an Account Number (1-100)
Entering 0 Will Exit User Input
Account Number: 0

Request Specific User Data

Enter 0 To Exit
Enter Account Number: 2
Account Number: 2
First Name:     p
Last Name:      abcdefghijklmnop
Balance:       $1234.56

Enter 0 To Exit
Enter Account Number: 1
ACCOUNT NUMBER NOT IN USE

Enter 0 To Exit
Enter Account Number:
```

12. Error Log

Error Type (Logic/Runtime)	Cause of Error	Solution to Error
Logic	Not Resetting accountNumber while checking for valid input.	Include resetting to -1 so that userIn can be compared correctly.

13. Status

Program is working. Just don’t stress it out too much, it has anxiety.