**University of Michigan – Dearborn**

**CIS 200 – Computer Science II**

**Lab# 8**

**Professor Robert Mann**

Demetrius Johnson

[meech@umich.edu](mailto:meech@umich.edu)

December 02, 2020

**Table Content**

Contents

[Question 1 (PROGRAM 1) 3](#_Toc57847623)

[Source Code –see program 1 source code in canvas 5](#_Toc57847624)

[Screenshots 5](#_Toc57847625)

# Question 1 (PROGRAM 1)

// LAB 8 - CIS 200

Question 1

-------------------------

Given:

struct clientData {

int accountNumber;

char lastName[ 16 ]; // c-style string, not string

char firstName[ 11 ];

float balance;

};

Create a random access file of 101 records named “credit.dat” using these code segments in a function to initialize the file to empty records:

ofstream creditFile( "credit.dat", ios::out);

clientData blankClient = { 0, "", "", 0.0 };  
  
for ( int i = 0; i <= 100; i++ )  
 creditFile.write( reinterpret\_cast<const char \*>( &blankClient ), sizeof( clientData ) );

creditFile.close();

<ostream> member function *write* - outputs a fixed number of bytes beginning at a specific location in memory to the specific stream (file).

The *write* function expects the first argument of type *const char \*,* hence the use of

*reinterpret\_cast <const char \*>* to convert the address of *blankClient* to *const char \*.*The second argumentof *write* is an integer specifying the number of bytes to be written.Thusthe use of *sizeof( clientData )*. Since size will never change it must be declared as a constant variable; sizeof must only appear once in your program: for the constant declaration.

The first entry in the file will be skipped so that record 1 is at position 1 and not position 0.

Write data into the file [minimum of 15 records, not in 15 consecutive account numbers] getting all data from the user. Truncate user input if it is too long for the data field.

Read data from the file. Loop, asking user for an account number to find (range 1 to 100, 0 to end input. error message if not a valid account number and try again). Print out all data fields for this account.

Update an account. Loop, asking user for an account number to update (error message if not a valid account number and try again) or 0 to quit (do several updates). Update firstname, lastname, or balance (but NOT accountNumber).

Print out all records that do not have accountNumber of 0, formatting output into columns (label columns).

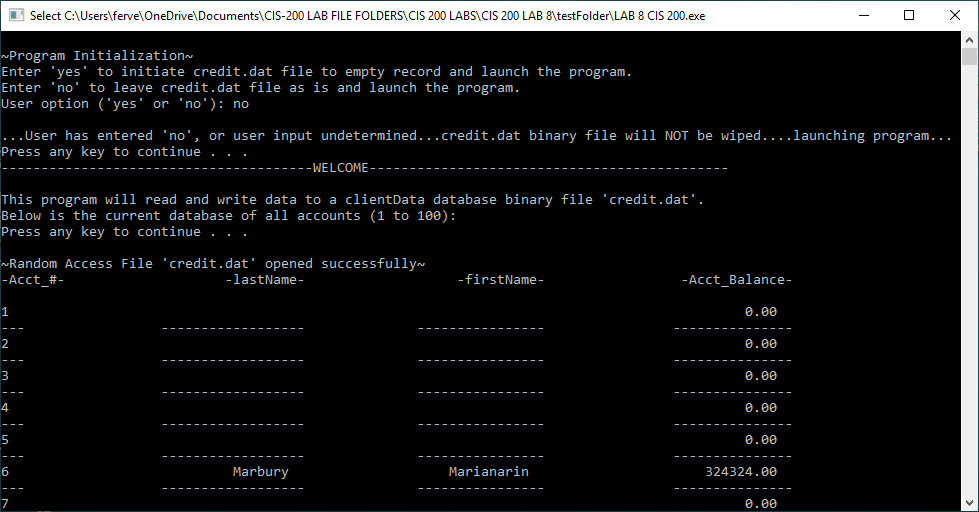
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test # | Valid / Invalid Data | Description of test | Input Value | Expected Output | Actual Output | Test Pass / Fail |
| 1 | VALID | Write data into the file [minimum of 15 records, not in 15 consecutive account numbers] getting all data from the user. Truncate user input if it is too long for the data field. | See test case screenshot; I looped through inputting data for 15 fields; including showing that data from user is truncated if too long | 15 lines of user input written to the file | See test case screenshots | pass |
| 2 | VALID | Read data from the file. Loop, asking user for an account number to find (range 1 to 100, 0 to end input. error message if not a valid account number and try again). Print out all data fields for this account. | See test case screenshot; I show what happens when looping through viewing multiple accounts, what happens for an invalid account, and what happens in the 0 case | Multiple accounts should output as the user specifies; error for nonexistent accounts, and exit for the 0-case | See screenshots | pass |
| 3 | VALID | Update an account. Loop, asking user for an account number to update (error message if not a valid account number and try again) or 0 to quit (do several updates). Update firstname, lastname, or balance (but NOT accountNumber). | See test case screenshot; loop and update several accounts, show that all fields have been updated show error for incorrect account or incorrect input for accountNumber; show 0 quit case. | 0-case expect to quit program; incorrect input expect error and re-try; valid data expected to update the account and get truncated if too long | See screenshots | pass |
| 4 | VALID | Print out all records that do not have accountNumber of 0, formatting output into columns (label columns | Select the viewing option B 🡪 display all accounts | All accounts should be output to the screen for user | See screenshot | pass |
| 5 | VALID | Show that the program is unbreakable from incorrect user input | Input false data in multiple ways for multiple menus/data input | Program should not crash; give error and retry | See screenshots | pass |

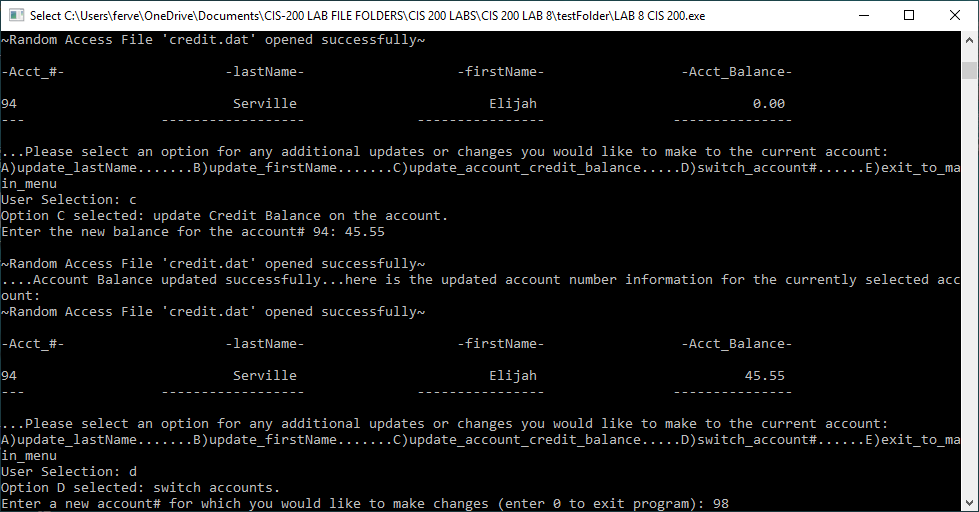
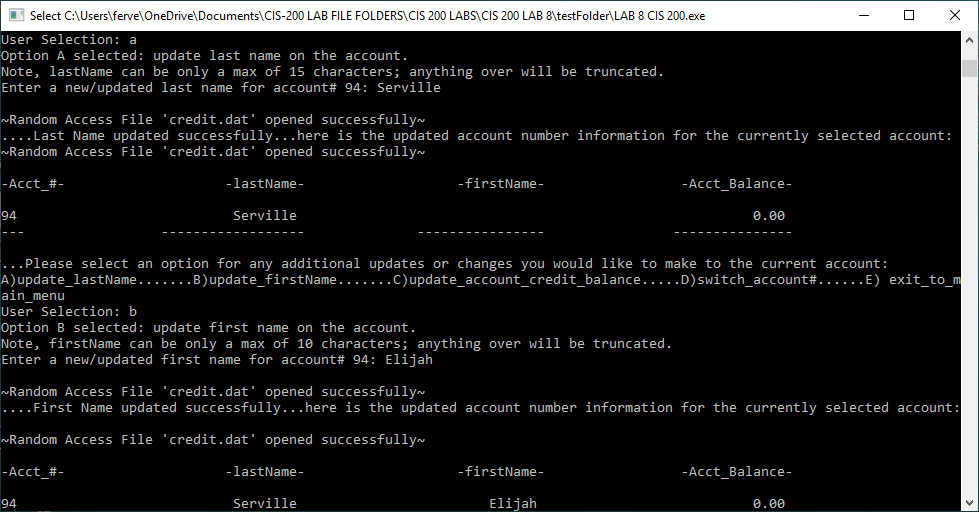
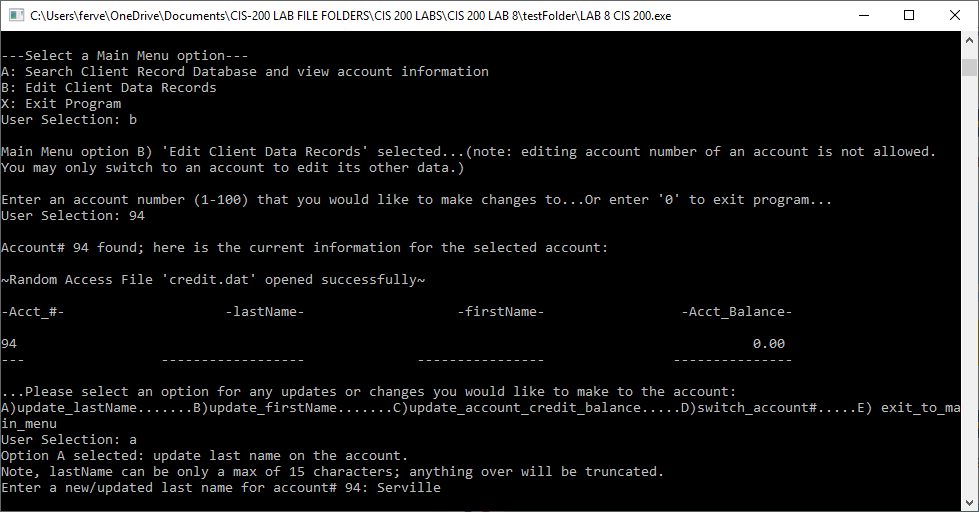
## Source Code –see program 1 source code in canvas

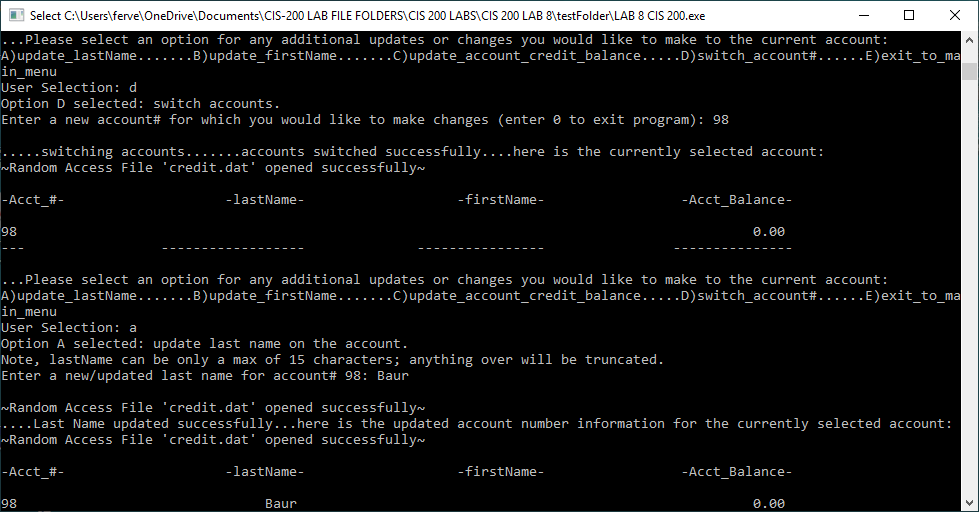
## Screenshots

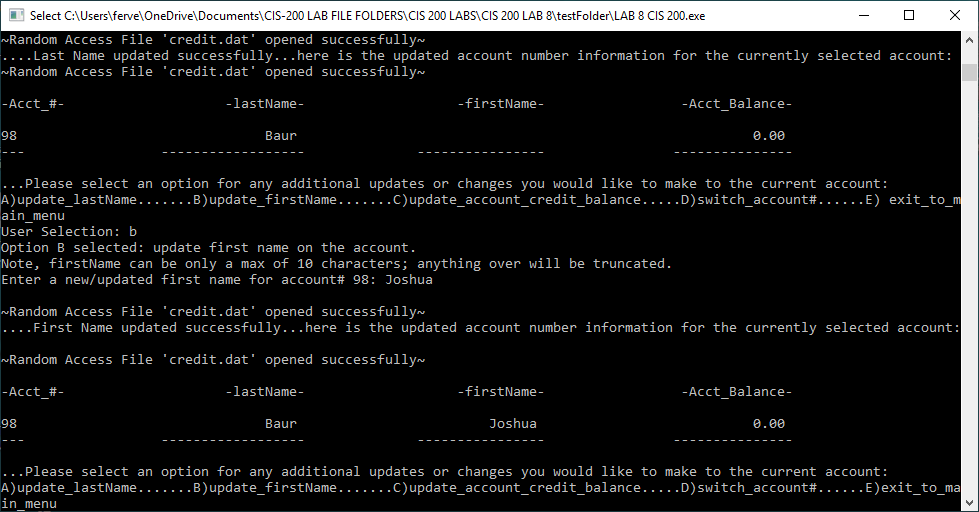
TEST 1 and – Write data into the file [minimum of 15 records, not in 15 consecutive account numbers] getting all data from the user. Truncate user input if it is too long for the data field.

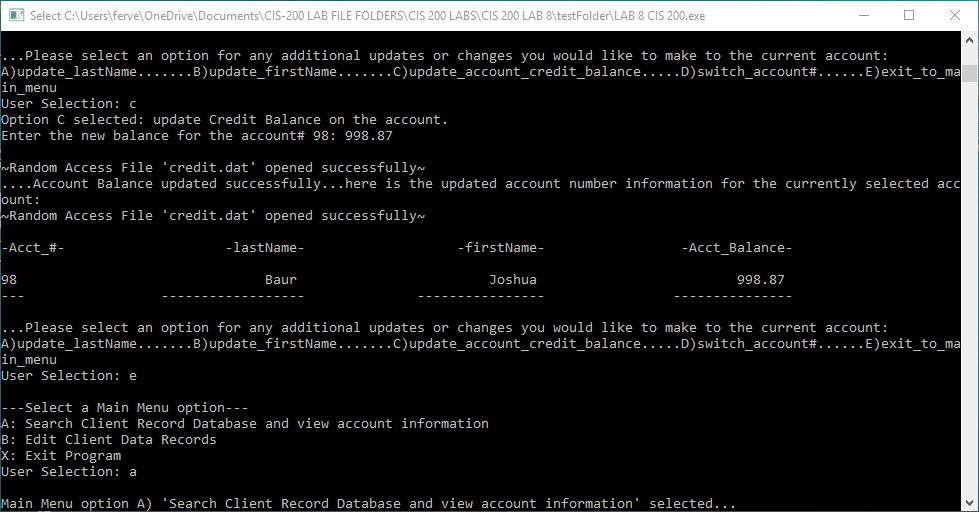
The following sequence of screenshots will show what I did for a few accounts (in total I did 15 as instructed, but I do not show screenshot for all 15, only for a few to demonstrate the full process):

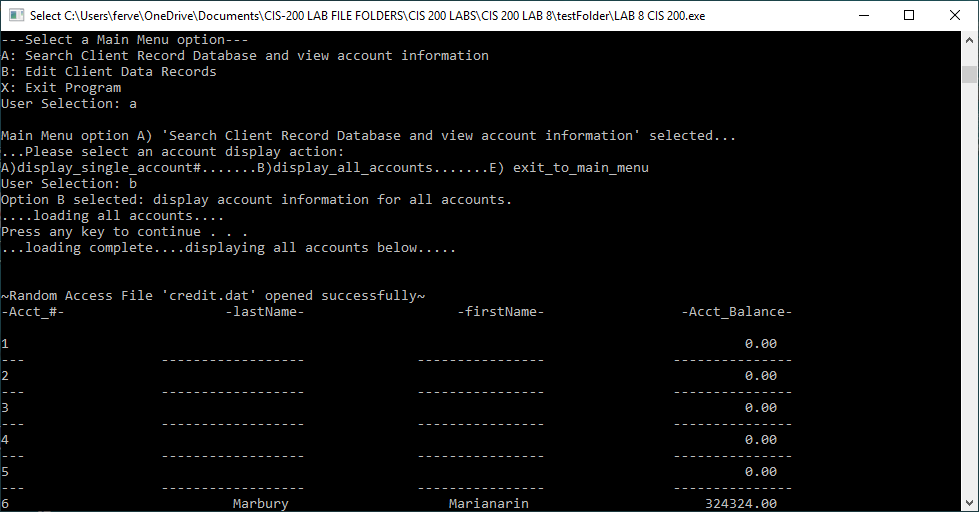


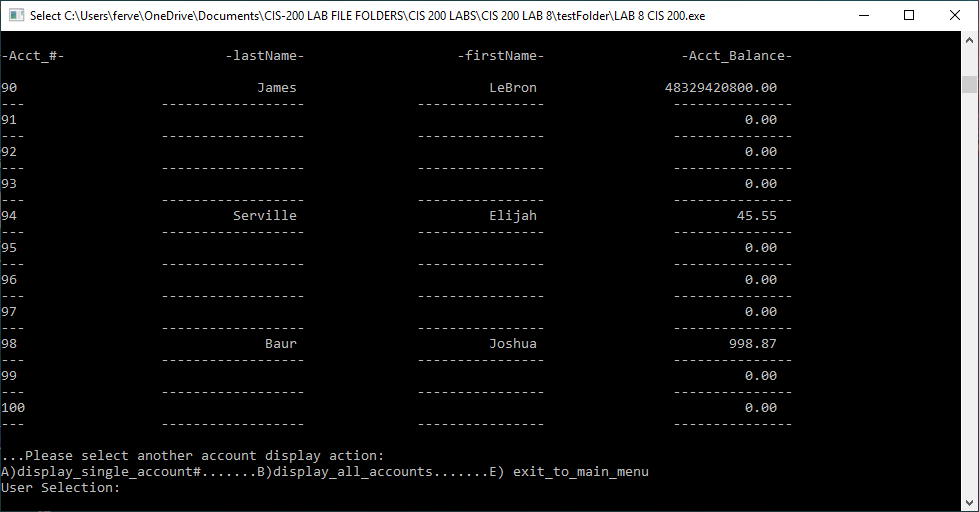




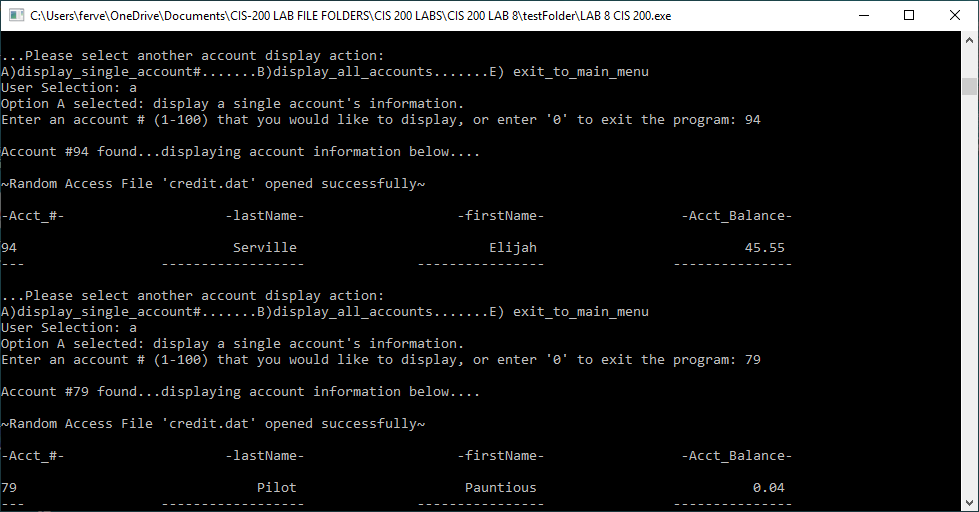


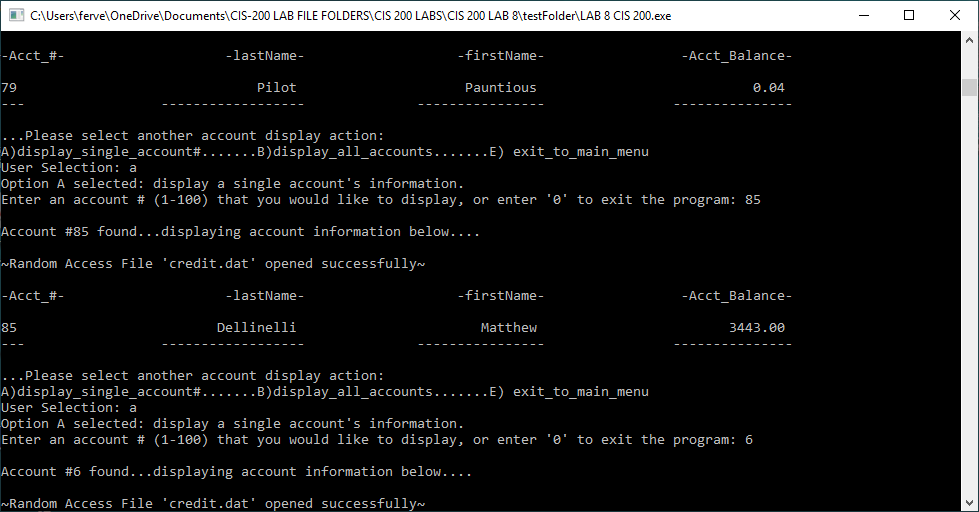


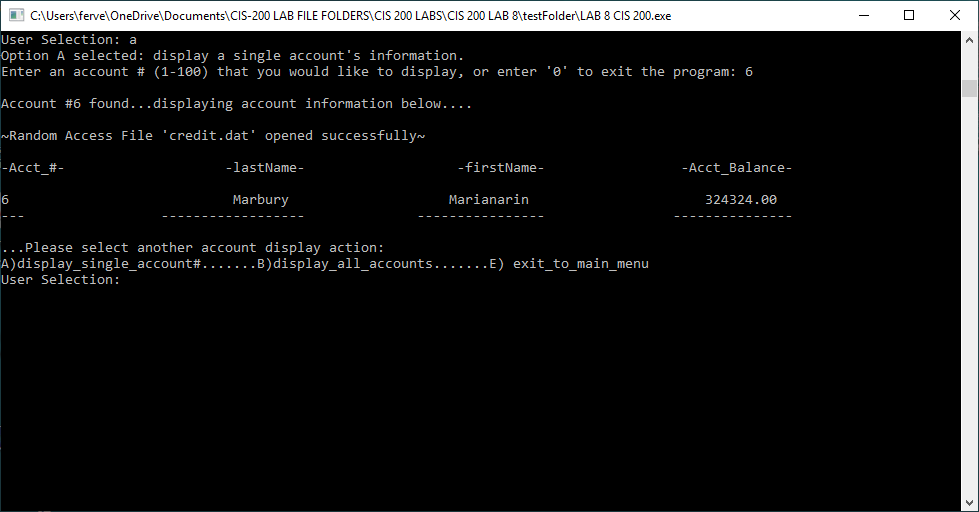




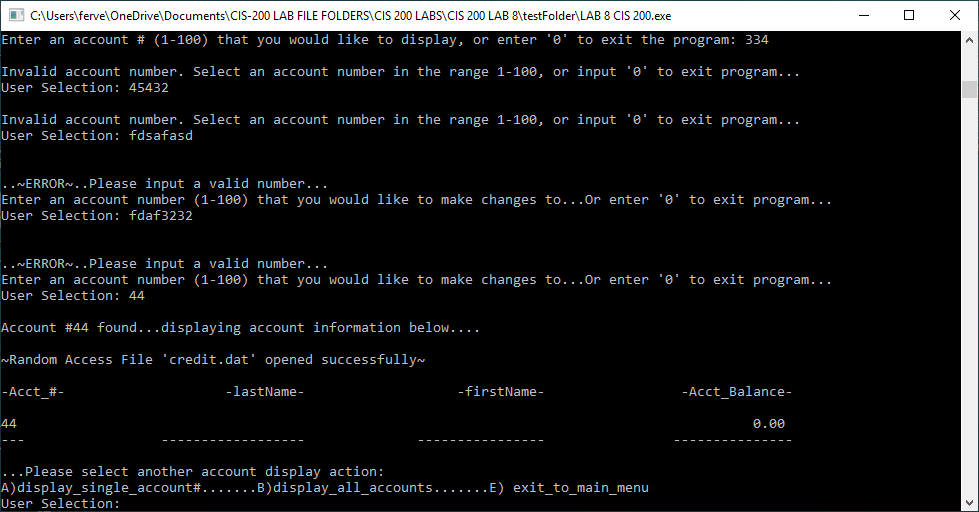
TEST 2 – Read data from the file. Loop, asking user for an account number to find (range 1 to 100, 0 to end input. error message if not a valid account number and try again). Print out all data fields for this account.



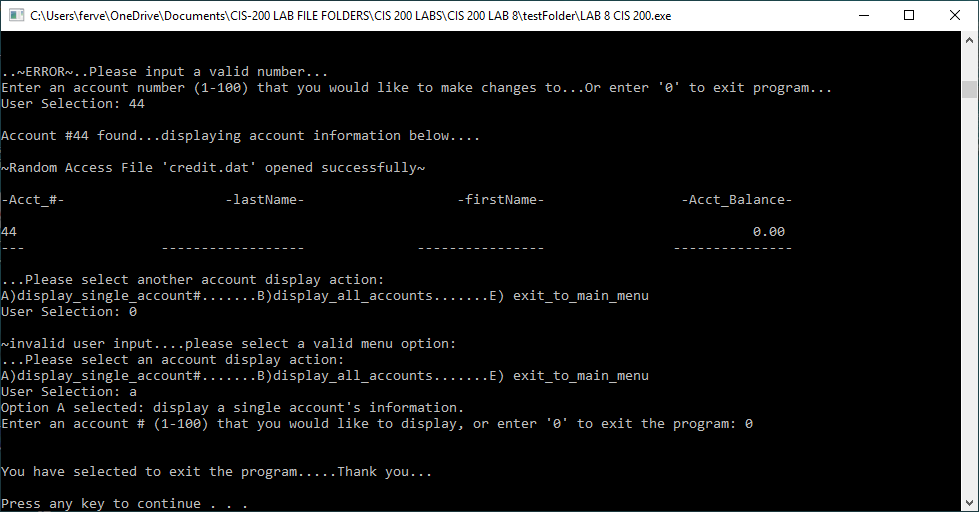




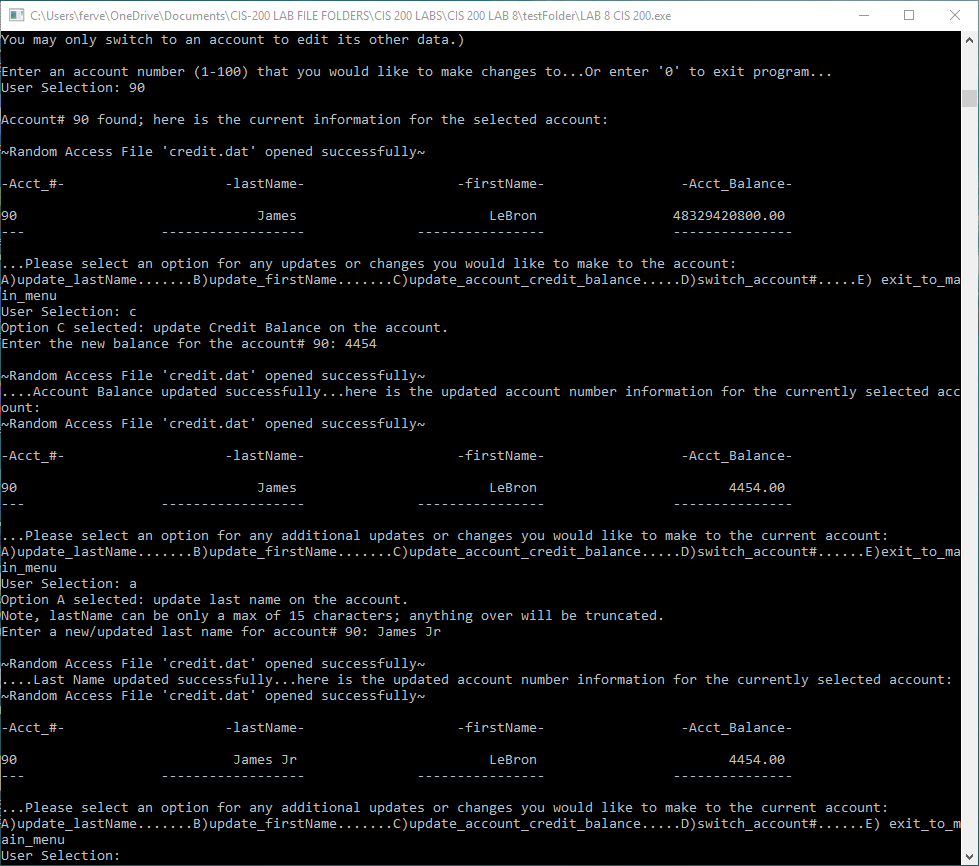
Error/invalid account num Case 🡪 then a final correct case:



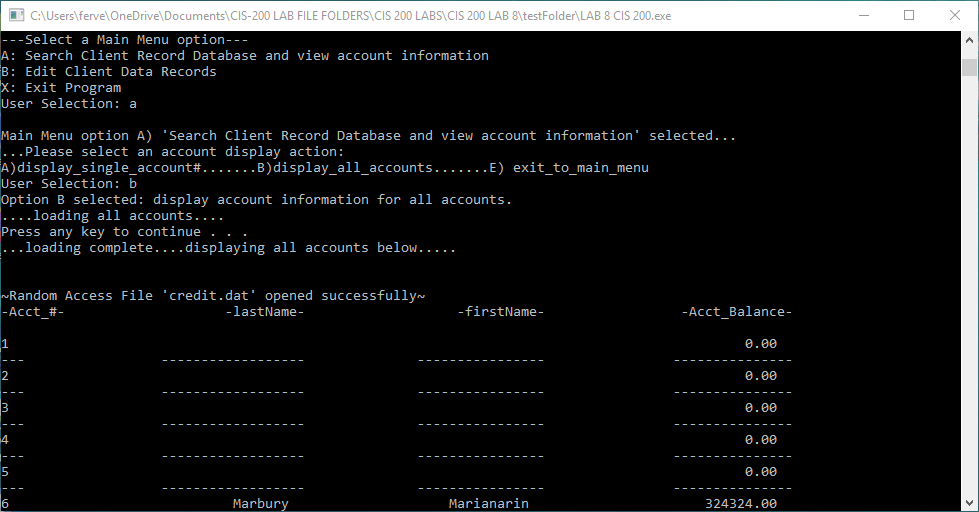
The 0-Case 🡪 exit program:

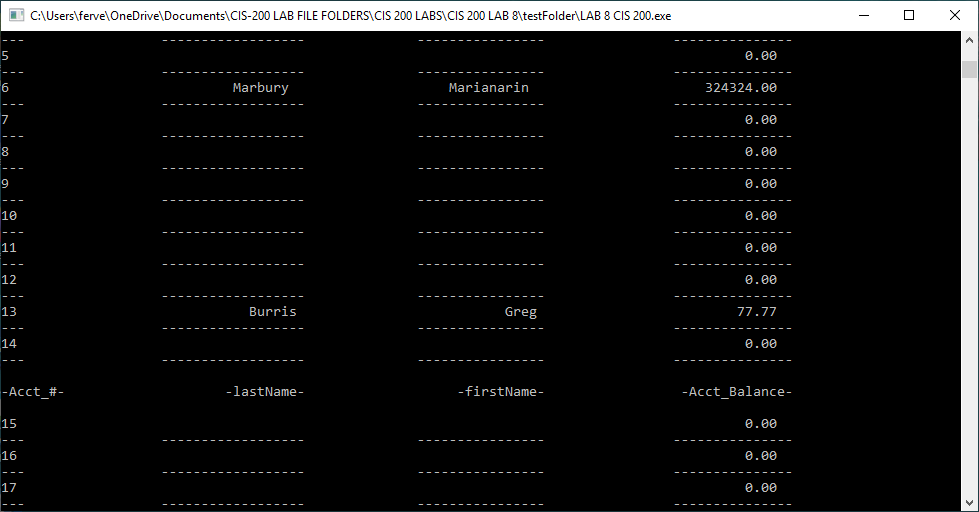


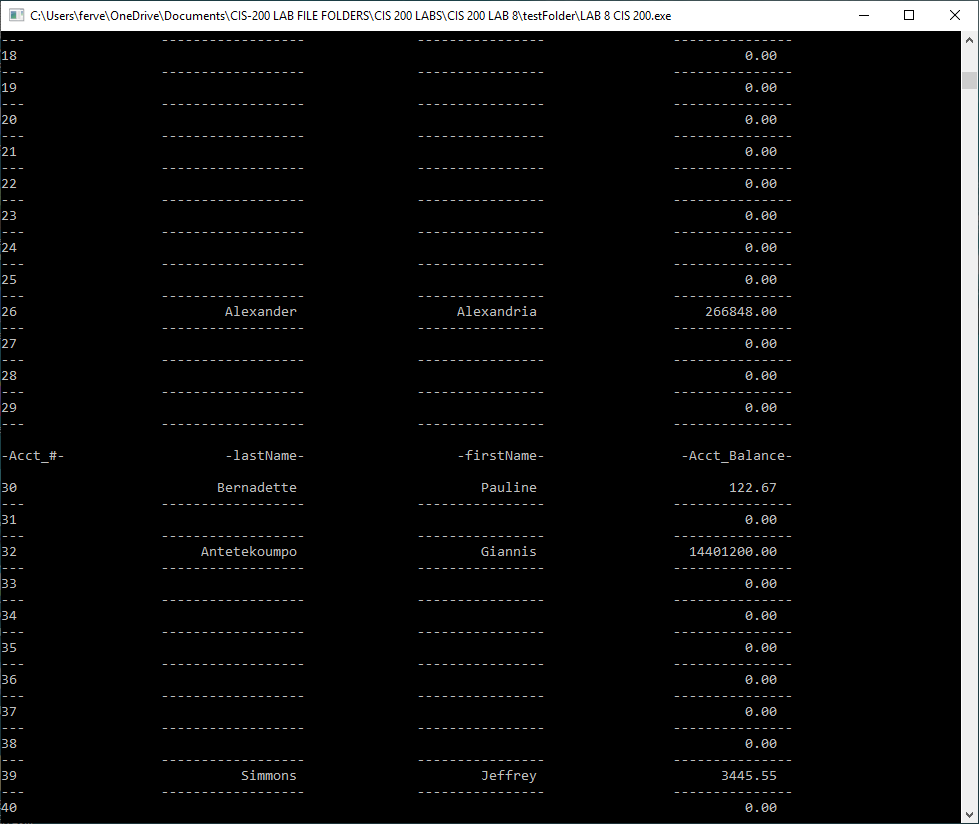
Test 3 – Update and account

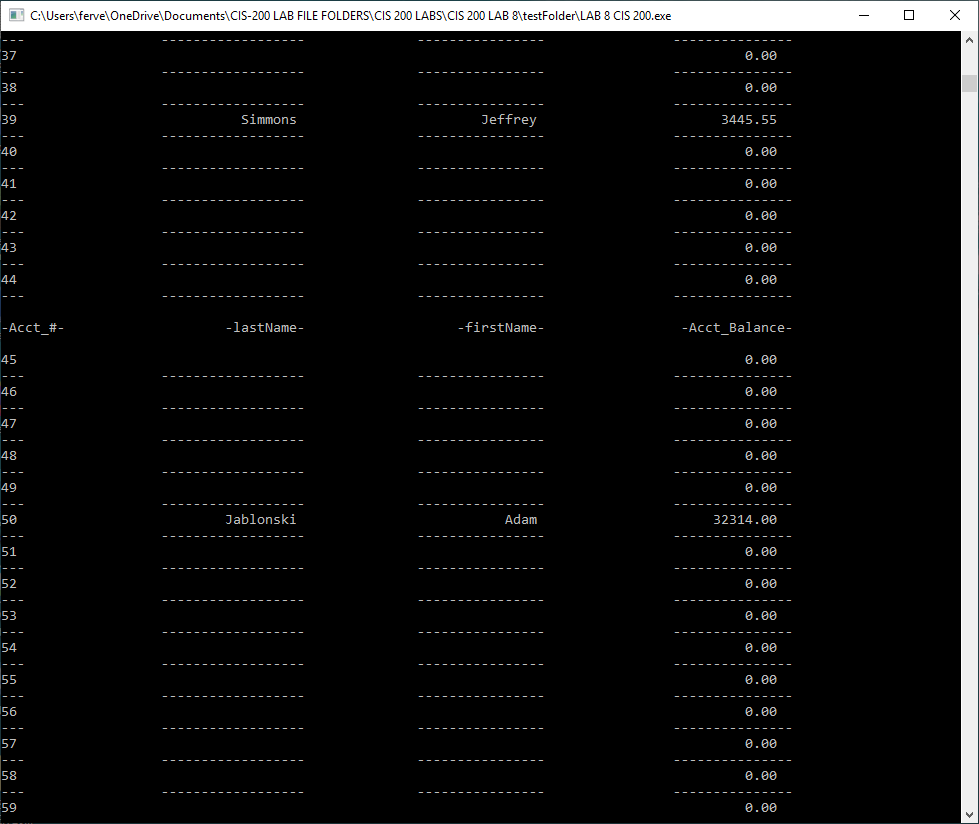


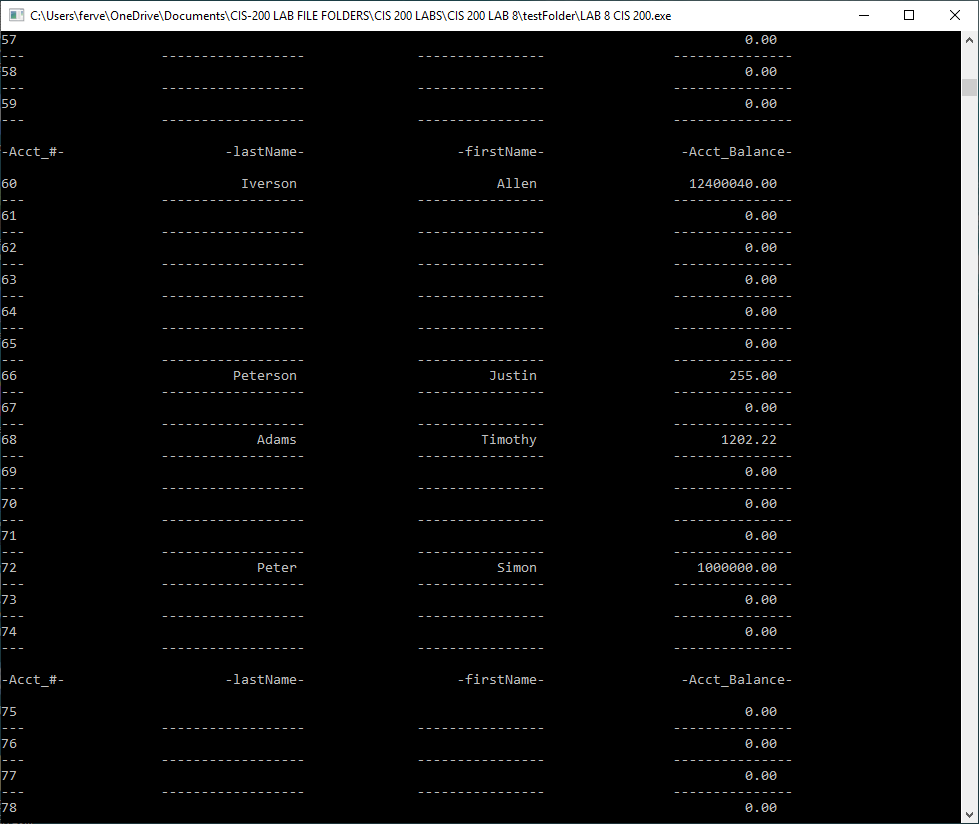
TEST 4 – Output all accounts:

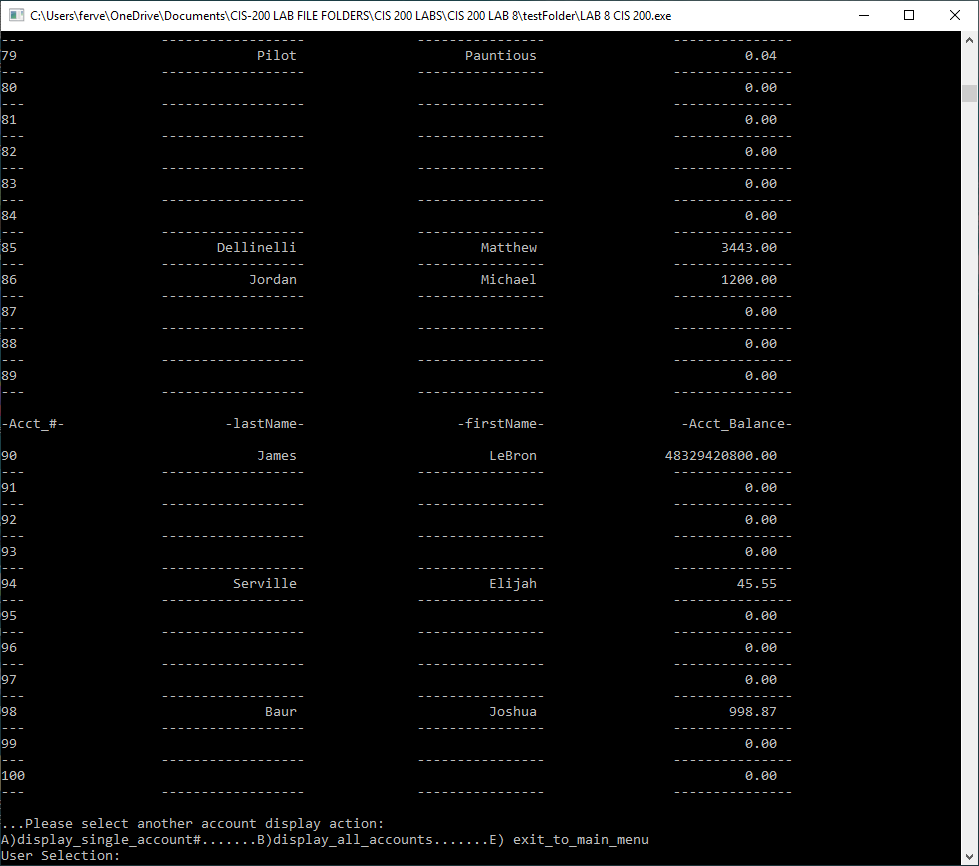


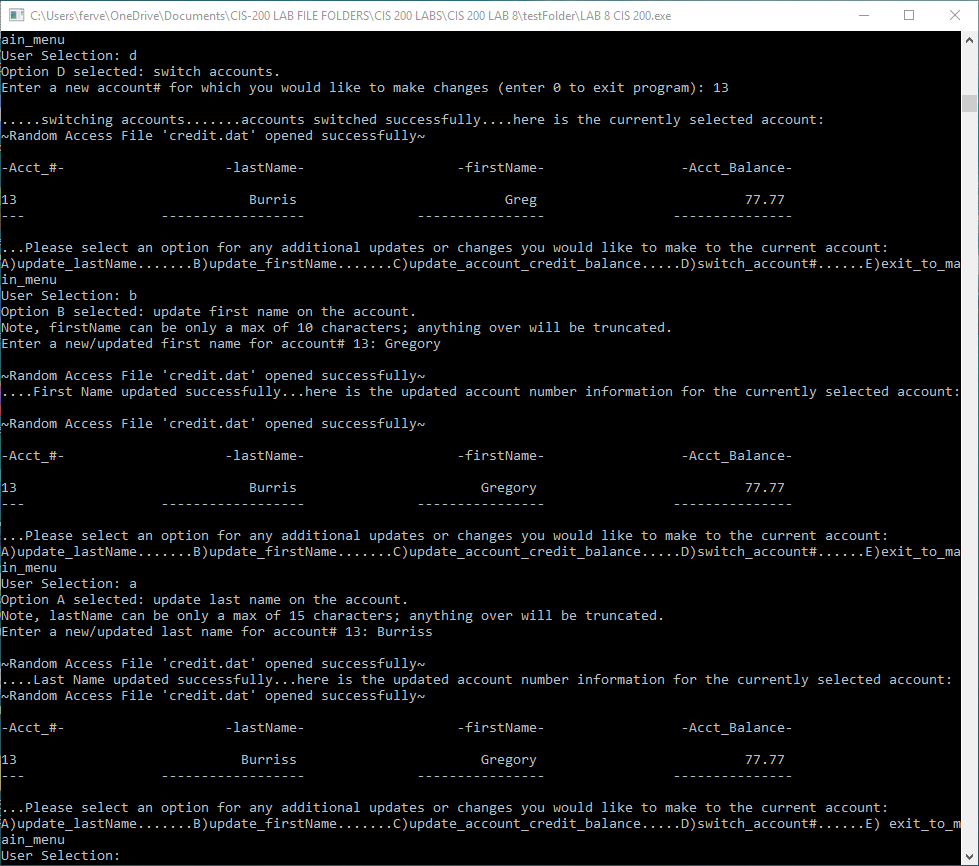












Test 5 – Show program cannot be broken with invalid data:

