1. **Problem Statement**

This program creates a random access file to hold client records and be able to let the user add or print out the client data of user’s choice.

1. **Requirements**
   1. **Assumptions**
      1. User inputs letters for name without any space in the middle
      2. User inputs numbers for balance
   2. **Specifications**
      1. Input and output files
      2. Get all client names, account number, and their current balance.
      3. Function that converts the address to a constant char \*
      4. Struct that holds client data
2. **Decomposition Diagram**

|  |  |  |
| --- | --- | --- |
| **Main** | | |
| **Input** | **Process** | **Output** |
| Account number | Write the next inputs at this location in file | Ask for next input |
| First name | Check if it is less than 10 characters long and store it in struct variable | Ask for next input |
| Last name | Check if it is less than 15 characters long and store it in struct variable | Ask for next input |
| Balance | Store balance in account number’s location |  |
| User enters 0 | Quit the loop and move on to reading the data in | Ask user for account number to display data for it |
| Account number to read in data | Find the location with all the corresponding data | Display first and last name, and balance to screen |

1. **Test Strategy**
   1. Valid Data
   2. Invalid Data
2. **Test Plan Version 1**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| Valid | 1 | User enters account number between 0 and 100 |  |  |  |  |
| Valid | 2 | User enters first name with less than 11 characters |  |  |  |  |
| Valid | 3 | User enters last name less than 16 characters |  |  |  |  |
| Invalid | 1 | User enters account number outside of bounds 0 to 100 |  |  |  |  |
| Invalid | 2 | User enters first name more than 10 characters long |  |  |  |  |
| Invalid | 3 | User enters last name more than 15 characters |  |  |  |  |

1. **Initial Algorithm**
   1. In struct *clientData*
      1. Make int variable for account number, character variables for last name, to hold 15 characters, and for first name, to hold 10 characeters, and a float variable for balance.
   2. In main function
      1. Make an output file named *credit.dat*
      2. Make an instance of *clientData* and set all variables to initial values
         1. Account number = 0
         2. Last name = empty string
         3. First name = empty string
         4. Balance = 0.0
      3. For loop from i = 0 to 100, i++
         1. Call the function write using output file and convert the address of the instance of *clientData* to const char \*.
         2. Return size of *clientData*
      4. Write data into the file
         1. Make an instance of *clientData* called client.
         2. Make an output file named credit.dat, but this time variable is named different.
         3. Loop following until false
         4. Ask user for an account number and store it in temporary variable
            1. Loop until user enters a number between 0 and 100
         5. If account number is 0, break out the loop
         6. Else, assign temporary variable to client instance account number
            1. Loop again to ask for first name and keep looping until input is less than or equal to 10 characters long
            2. Loop again to ask for last name and keep looping until input is less than or equal to 15 characters long
            3. Ask for client’s balance and store it in client instance in balance variable
      5. Read data from the file
         1. Loop following until false
         2. Make an input file named credit.dat
         3. Ask user for an account number and loop until the number is in bound between 0 and 100
         4. If account number is 0, break out the loop
         5. Else, do the calculation to find the specific location of that account number
         6. After reading it, display account number following with client’s full name, and account balance.
      6. Print out all records
         1. Read in from the input file and while it is not end of file, keep printing data for account number and its corresponding client name, and balance.
2. **Test Plan Version 2**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| Valid | 1 | User enters account number between 0 and 100 | “50” | Program moves to asking for first name assuming this is correct |  |  |
| Valid | 2 | User enters first name with less than 11 characters | “Nisarg” | Program moves to asking for last name assuming this is correct |  |  |
| Valid | 3 | User enters last name less than 16 characters | “Patel” | Program moves to asking for balance assuming this is correct |  |  |
| Valid | 4 | User enters 0 to quit | “0” | Program moves to asking for account number to read and its data to be displayed to screen |  |  |
| Invalid | 1 | User enters account number outside of bounds 0 to 100 | “101” | “Invalid input. Account number can only be between 0 and 100.” |  |  |
| Invalid | 2 | User enters first name more than 10 characters long | “Verylongname” | “Invalid input. First name cannot exceed 10 characters” |  |  |
| Invalid | 3 | User enters last name more than 15 characters | “”Verylonglastname” | “Invalid input. Last name's length cannot exceed 15 characters” |  |  |

1. **Code**

#include "stdafx.h"

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

struct clientData {

int accountNumber;

char lastName[15];

char firstName[10];

float balance;

};

int main()

{

int tempAccountNum;

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

clientData bankClient = { 0,"","",0.0 };

clientData client;

for (int i = 0; i < 100; i++) {

outCredit.write(reinterpret\_cast<const char\*>(&bankClient), sizeof(clientData));

}

cout << "Welcome stranger. Here you can play around with Client Data." << endl;

cout << "...Write to file...Enter 0 to quit..." << endl;

while (true) {

do {

cout << "Enter Account Number: ";

cin >> tempAccountNum;

if (tempAccountNum < 0 || tempAccountNum > 100) {

cout << endl << "Invalid input. Account number can only be between 0 and 100." << endl << endl;

}

} while (tempAccountNum < 0 || tempAccountNum > 100);

if (tempAccountNum == 0) {

break;

}

else {

client.accountNumber = tempAccountNum;

do {

cout << "Enter First Name: ";

cin >> client.firstName;

if (string(client.firstName).length() > 10)

{

cout << endl << "Invalid input. First name cannot exceed 10 characters" << endl << endl;

}

} while (string(client.firstName).length() > 10);

do {

cout << "Enter Last Name: ";

cin >> client.lastName;

if (string(client.lastName).length() > 15)

{

cout << endl << "Invalid input. Last name's length cannot exceed 15 characters" << endl << endl;

}

} while (string(client.lastName).length() > 15);

cout << "Enter Client Balance: ";

cin >> client.balance;

cout << endl;

outCredit.seekp((client.accountNumber - 1) \* sizeof(clientData));

outCredit.write(reinterpret\_cast<const char\*>(&client), sizeof(clientData));

}

}

outCredit.close();

ifstream inCredit("credit.dat", ios::in);

cout << "...Read from file...Enter 0 to quit..." << endl;

while (true) {

do {

cout << "Enter Account Number: ";

cin >> tempAccountNum;

if (tempAccountNum < 0 || tempAccountNum > 100) {

cout << endl << "Invalid input. Account number can only be between 0 and 100." << endl << endl;

}

} while (tempAccountNum < 0 || tempAccountNum > 100);

if (tempAccountNum == 0) {

break;

}

else {

inCredit.seekg((tempAccountNum - 1) \* sizeof(clientData));

inCredit.read(reinterpret\_cast<char\*>(&client), sizeof(clientData));

cout << "Account Number: " << client.accountNumber << endl;

cout << "Client Name: " << client.firstName << " " << client.lastName << endl;

cout << "Account Balance: $" << client.balance << endl;

}

}

outCredit.close();

cout << endl << "Printing out all records that do not have account number 0..." << endl;

inCredit.seekg(0);

inCredit.read(reinterpret\_cast< char \*>(&client), sizeof(clientData));

while (inCredit && !inCredit.eof()) {

if (client.accountNumber != 0) {

cout << "Account Number: " << client.accountNumber << endl;

cout << "Client Name: " << client.firstName << " " << client.lastName << endl;

cout << "Account Balance: $" << client.balance << endl << endl;

}

inCredit.read(reinterpret\_cast<char \*>(&client), sizeof(clientData));

}

inCredit.close();

cout << "Hope you had fun, stranger. See you next time." << endl;

system("pause");

return 0;

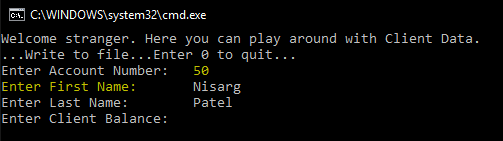
}

1. **Updated Algorithm**
   1. In struct *clientData*
      1. Make int variable for account number, character variables for last name, to hold 15 characters, and for first name, to hold 10 characeters, and a float variable for balance.
   2. In main function
      1. Welcome message
      2. Make an output file named *credit.dat*
      3. Make an instance of *clientData* and set all variables to initial values
         1. Account number = 0
         2. Last name = empty string
         3. First name = empty string
         4. Balance = 0.0
      4. For loop from i = 0 to 100, i++
         1. Call the function write using output file and convert the address of the instance of *clientData* to const char \*.
         2. Return size of *clientData*
      5. Write data into the file
         1. Make an instance of *clientData* called client.
         2. Make an output file named credit.dat, but this time variable is named different.
         3. Loop following until false
         4. Ask user for an account number and store it in temporary variable
            1. Loop until user enters a number between 0 and 100
         5. If account number is 0, break out the loop
         6. Else, assign temporary variable to client instance account number
            1. Loop again to ask for first name and keep looping until input is less than or equal to 10 characters long
            2. Loop again to ask for last name and keep looping until input is less than or equal to 15 characters long
            3. Ask for client’s balance and store it in client instance in balance variable
      6. Close output file
      7. Read data from the file
         1. Loop following until false
         2. Make an input file named credit.dat
         3. Ask user for an account number and loop until the number is in bound between 0 and 100
         4. If account number is 0, break out the loop
         5. Else, do the calculation to find the specific location of that account number
         6. After reading it, display account number following with client’s full name, and account balance.
      8. Print out all records
         1. Read in from the input file and while it is not end of file, keep printing data for account number and its corresponding client name, and balance.
      9. Thank You message
2. **Test Plan Version 3**

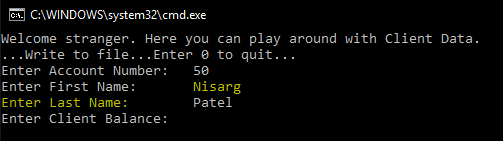
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| Valid | 1 | User enters account number between 0 and 100 | “50” | Program moves to asking for first name assuming this is correct | Program moves to asking for first name assuming this is correct | Pass |
| Valid | 2 | User enters first name with less than 11 characters | “Nisarg” | Program moves to asking for last name assuming this is correct | Program moves to asking for last name assuming this is correct | Pass |
| Valid | 3 | User enters last name less than 16 characters | “Patel” | Program moves to asking for balance assuming this is correct | Program moves to asking for balance assuming this is correct | Pass |
| Valid | 4 | User enters 0 to quit | “0” | Program moves to asking for account number to read and its data to be displayed to screen | Program moves to asking for account number to read and its data to be displayed to screen | Pass |
| Invalid | 1 | User enters account number outside of bounds 0 to 100 | “101” | “Invalid input. Account number can only be between 0 and 100.” | “Invalid input. Account number can only be between 0 and 100.” | Pass |
| Invalid | 2 | User enters first name more than 10 characters long | “Verylongname” | “Invalid input. First name cannot exceed 10 characters” | “Invalid input. First name cannot exceed 10 characters” | Pass |
| Invalid | 3 | User enters last name more than 15 characters | “”Verylonglastname” | “Invalid input. Last name's length cannot exceed 15 characters” | “Invalid input. Last name's length cannot exceed 15 characters” | Pass |

1. **Screenshots**

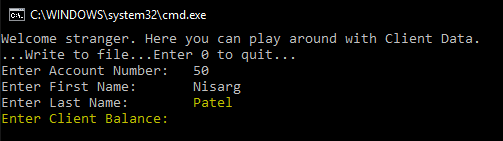
Valid Test Case 1:



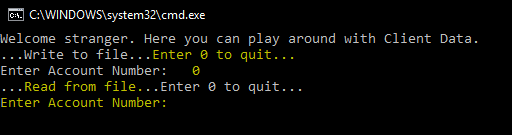
Valid Test Case 2:



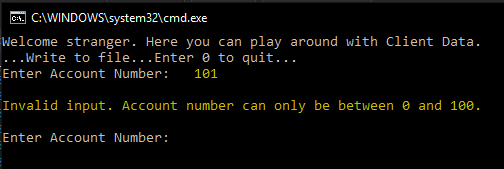
Valid Test Case 3:



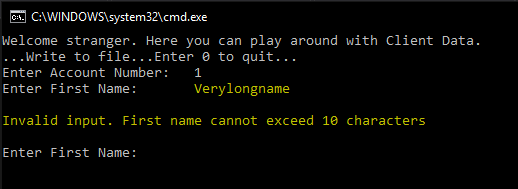
Valid Test Case 4:



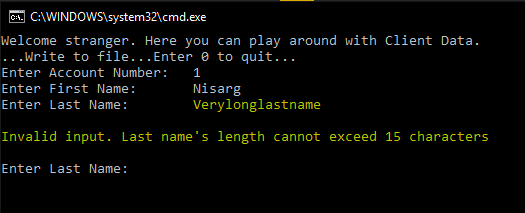
Invalid Test Case 1:



Invalid Test Case 2:



Invalid Test Case 3:



1. **Status**

Program works decent with all the assumptions in mind.