Project 02: Data Structure with User Defined Data

Nicholas Jones

Capitol Technology University

Prepared for Professor Mehri

CS-200

**Objectives**

* Complete the *Brokerage\_Client* class implementation and the main program
* An investment customer must have a broker, and a checking client doesn't need one
* The program must be able to generate several clients that hold checking accounts as well as investment accounts
* The program must be menu driven to allow for proper selection of operations

**Procedure**

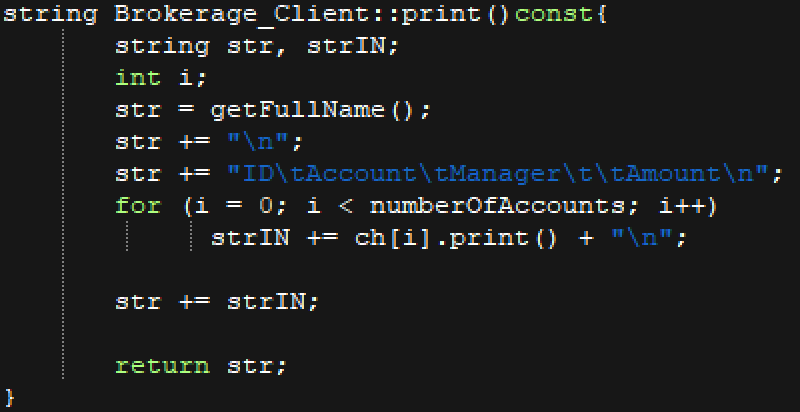
First, because this was a group project we had to divide the workload among the group members. We assigned two members, including myself, to completing the brokerage client part of the program, and the other three members were left to complete the menu and integrate all of the classes into the main program for functionality. Both teams within the group made a goal for a date that we would have each part done and then we began working when there was time.

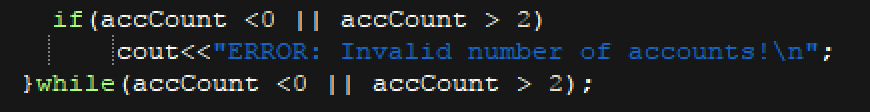
My team was tasked with completing the brokerage client functionality of the program to mirror the normal banking client that was already completed. The main differences that we had to change for the brokerage client were that a brokerage client could only have up to two investment accounts as opposed to ten checking accounts for a banking client. The other difference was that the brokerage client had an extra data member known as the account manager for each investment account which essentially was a broker for the brokerage client. First we began by copying the code from the banking client into the new brokerage client file, and to only allow two accounts per client we just had to change the value from 10 to 2 within the account creation loop. To add the functionality for the account manager data member we mostly had to focus on allowing the user to enter into this data member by using a setter function within the class. We also had to modify the overridden virtual print function to have an extra client to print out the account manager for each investment account. On our team we also changed some other little things throughout the program such as printing an error when the user tried to create an invalid amount of accounts.

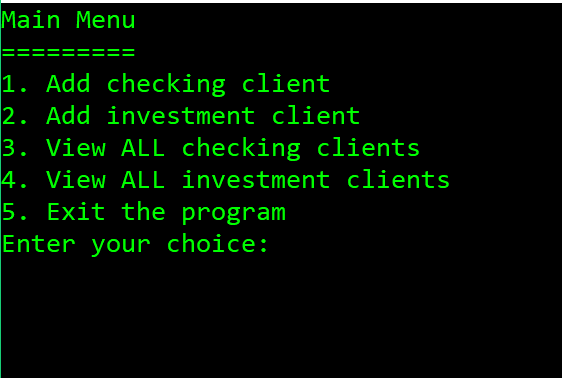
The other team within our group was tasked with creating the main program functionality that included a menu. They started by creating a simple menu function within the main file to print out the options. They then created a switch case within main that would take the selection from the menu function and act on it by running the associated function to either add a type of client, print a type of client, or exit the program. The team working on the menu could do most of what they did without our part being completed, but once we completed the brokerage side of the program, the menu team had a much smoother time completing the switch case for the menu.

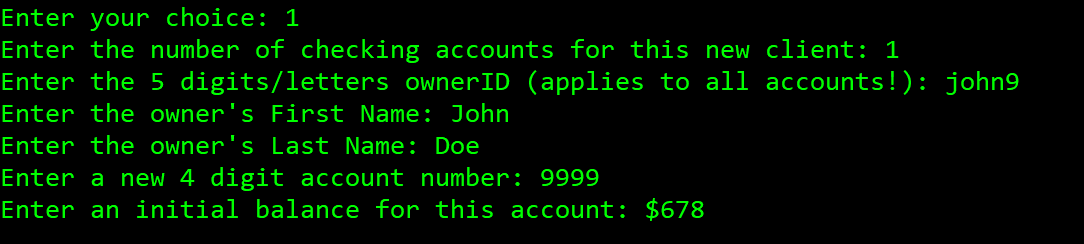
**Final Product**

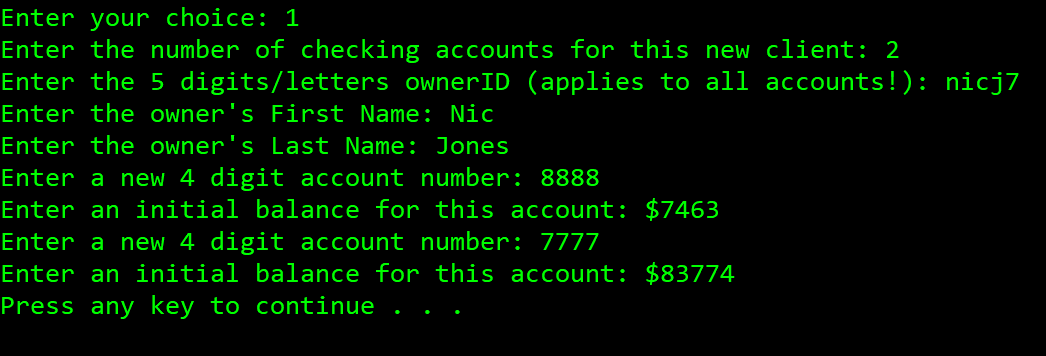
**Figure 1**

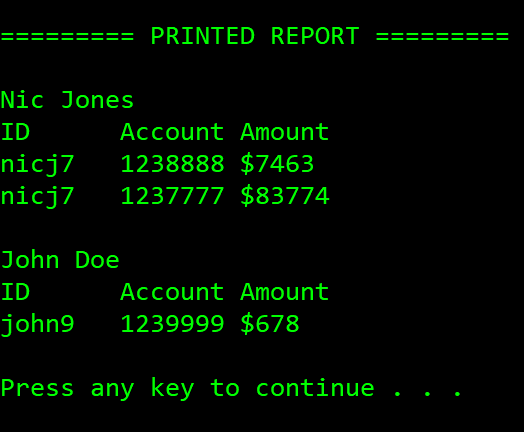
  
*This image shows the modified overridden virtual print function to support printing out the account manager data member to the user.*

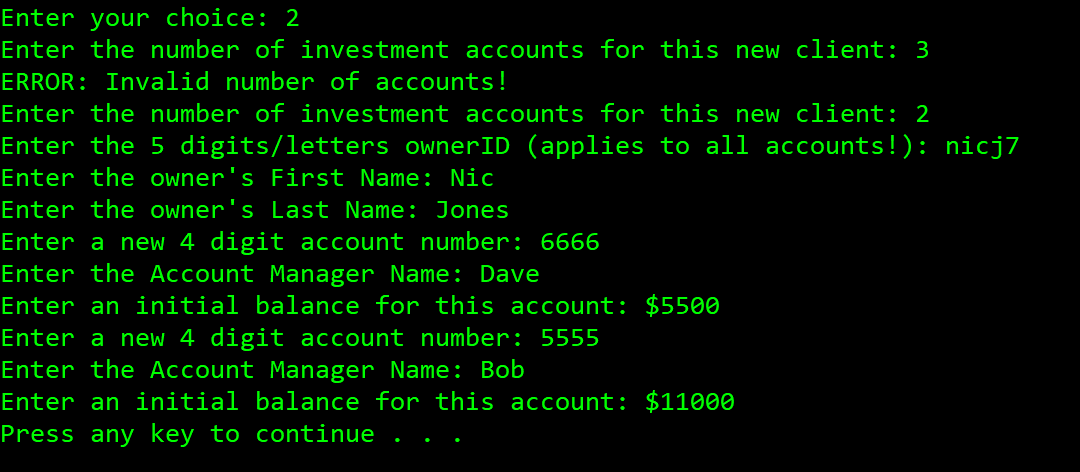
**Figure 2  
***This figure shows the modified allowed amount of investment accounts that only allows the loop to run 2 time and prints an error if the user tries to make more than 2 investment accounts.*

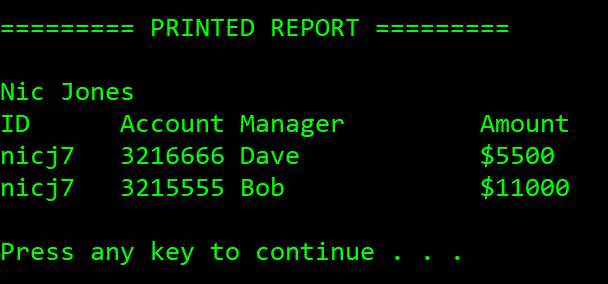
**Figure 3***This image shows the menu that is used to navigate the program and create the two types of clients as well and print them to view them and also exit.*

**Figure 4  
***This image shows the creation of a banking client where all of the data members are set including the OwnerID, Name, Account Number, and the initial balance.*

**Figure 5  
***This image shows the creation of another banking client with the difference that this client has two checking accounts rather than just one in Figure 4.*

**Figure 6  
***This image shows the printed report for the overridden print function for banking clients. The two separate clients that were set in Figure 5 & 4 are printed along with their specific checking accounts and the information that was provided. Note that checking account numbers have a prefix of “123”.*

**Figure 7  
*****This image shows the creation of a brokerage client that has two investment accounts. There was an attempt to create more than 2 investment accounts for the single client where an error was printed. Note that for each investment account a broker/account manager is specified.*

**Figure 8  
*****This image shows the printed report for the brokerage client that has two investment accounts. Take notice that the account manager for each investment account is also printed. Also note that investment accounts have a prefix of “321”.*

**Final Analysis**

Making this program had many important concepts and ideas that came along with it, and because it was a project one very good experience was working with a group that came together and split the workload to make the finished product. Another concept to learn from was being able to complete someone else’s code in the event that they could not do it themselves. In the real world its very likely that this would happen, and it requires adaptation and recognition of the original programmer’s approach to creating the program. It is important to be able to work after someone else on a program or else you’ll be right back at the start and have to complete the entire thing yourself.

The other advantage to this project is that it truly encompassed most of everything we learned this semester in a single project. Concepts such as encapsulation, inheritance, virtual functions, arrays and data structure were all included in this program making it very well rounded as we are ending the semester. The main goal of this project however, was to be able to create a data structure with different levels of classes that are being inherited, and allowing a user to add and view data within the structure of classes at runtime. Although, the main goal of this project was data structures with user interaction, I believe another very important concept was the idea of a pure virtual function in the base class of the program. “We can often write better code using polymorphism, i.e., using public inheritance, base class pointers (or references), and virtual functions” (Pitts, n.d.). The creation of the pure virtual print function within the client class allowed very easy modification of the print function down the inheritance hierarchy to match the derived classes. For example, the brokerage client’s print function needed to be different and have support for the account manager data, and therefore we were able to override the function and match it appropriately. Overall the concepts of polymorphism and object-oriented programming made this project very efficient to code rather than trying to code it sequentially.

**Conclusion**

In conclusion this project was very helpful in wrapping up this course for me. The inclusion of all of the concepts we’ve learned throughout the semester in a single program plus the aspects of working with a group after another programmer make this project one that I’ll remember. This project showed me more of how object oriented C++ programming will work in the real world and the concepts that I will use.

References

Pitts, R. I. (n.d.). Introduction to Polymorphism in C++. Retrieved November 30, 2017, from   
 https://www.cs.bu.edu/teaching/cpp/polymorphism/intro/