****
Authors:
****

Kyle Chen - kc871 - Section 5
Bhavin Soni - bbs39 - Section 4

****
Usage:
****

(TERMINAL 0)
$ make all
$ ./bankingServer 74361 <--this is the port number we have used-->

(TERMINAL 1)
$ ./bankingClient cd.cs.rutgers.edu 74361 <--this is the machine and port number we used-->

****
bankingAccount.c
****
A method used to dictate actions that can be run by the user. Actions such
as debit, credit, balance, open account, withdraw, deposit,
and such are all accessible by the account. The accounts themselves are just structs
that are held in an array of pointers by the server.

****
bankingClient.c
****
The client program takes in the machine's hostname and port number. The client will first
attempt to connect with the server every 3 seconds until a connection can be made.
The client then activates two simultaneous threads that
both interact with the user. The first thread waits in the background and takes on
any input that the user gives it. It then sends the input
to the server. The other thread takes information from the server in response and
displays that for the user. This design of front-end
back-end helps keep everything organized when debugging, and also allows a far more
efficient processing time. This also allows a constant
exchange of information between the user and the server. That way, all the server
can shut down the clients down during error.

If the user ever gives the input end, the client will close both threads and close
out of the program. This will also send an indicator
to the server to close down as well, as all threads connected to it are then
terminated.

****
bankingServer.c
****
The server takes in the port number to start the client thread. The server's primary objective is to take and send input to the client program. It also keeps an array of pointers to all the
accounts saved on the server. At first the server waits with a clientAccept method
running to detect any clients connecting to it. It also
gives status checks to the clients depending on whether there are any errors or not.
The server then waits for any actions sent by the
client. Once an action has been sent, it goes to the action thread to determine what
action or command the user wants to use. The action
command thread changes the information saved by the server such as balances,
accounts, and other static information. This method also
keeps track of the mutex locking of the accounts for security. This works by setting
two arrays of both accounts and the mutexes for each of
those accounts. This mutext also allows multiple users to perform similar actions at
the same time. For instance, changing the same data is
prohibited since the mutex locks an account that is already in use.
In response to the server closing, the server goes through a linkedlist of connected
accounts and sends them each a flag to
end the clients. This multithreading process enables us to run multiple simultaneous
accounts at the same time.