Rubric For Project 1

In all the testcases, we will first start the server by `./server [ipAddr] [port]` (assume the executable file for server is `server`). Note that, the support for parameters `ipAddr` (ip address) and `port` (port number) is optional, you can hardcode them in your server program. And during tests, if the corresponding port is occupied (Don't Worry), we will change to another available one.

In the following tests, we assume that server is running (on the same machine as client) and client executable file is `client`.

 For ease of grading, we fix the arguments of your client program. That is,

```
./client testaccount ipAddr port CMD [amount]
# testaccount is one of myChecking|mySavings|myRetirement|myCollege
# idAddr is the ip address for the sever (e.g., 127.0.0.1)
# port is the port number (e.g., 2017)
# CMD is one of BAL|WITHDRAW and when it is WITHDRAW, an amount is followed
```

Note that, you can assume that we will always input valid arguments according to your requirements.

Implementation Based on TCP (1 point)

 Please make sure you use TCP instead of UDP.

Balance Tests (3 points)

We will test balance function by using :

```
./client testaccount ipAddr port BAL
# testaccount is one of myChecking|mySavings|myRetirement|myCollege
```

Expected Outcome: An integer to indicate the amount of balance of the corresponding account.

Rubric

We might randomly test some or all of the four accounts.

- all fail ==> 0 point
- partially work ==> 2 point
- all work ==> 3 points

Correct Withdraw Tests (4 points)

We will test correct withdraw function by the following steps.

Step 1

```
./client testaccount ipAddr port WITHDRAW amount
# testaccount is one of myChecking|mySavings|myRetirement|myCollege
```

 You should tell the user withdraw succeeds (e.g., print out a message "Withdraw Succeeded!").

Expected Outcome: A message that tells the user withdraw succeeds.

Step 2

You will check the balance of the corresponding account again.

Expected Outcome: The balance should change accordingly.

Rubric

We might randomly test some or all of the four accounts.

- all fail ==> 0 point
- first step succeeds, second fails ==> 2 point
- all work ==> 4 points

Timeout Test (2 points)

In this part, we will randomly generate 10 testcases (5 timeouts, 5 not) (Do not worry about the boundary cases!).

An example of the possible testcases, where we should not see timeout error:

```
./client myChecking ipAddr port WITHDRAW amount
# after 5 seconds
./client myChecking ipAddr port WITHDRAW amount
# after 5 seconds
./client mySavings ipAddr port WITHDRAW amount
# after 10 seconds
./client mySavings ipAddr port WITHDRAW amount
# after 120 seconds
./client myChecking ipAddr port WITHDRAW amount
```

An example of the possible testcases, where we should see timeout error:

```
./client myChecking ipAddr port WITHDRAW amount
# after 5 seconds
./client myChecking ipAddr port WITHDRAW amount
# after 5 seconds
./client myChecking ipAddr port WITHDRAW amount
# after 10 seconds
./client mySavings ipAddr port WITHDRAW amount
# after 5 seconds
./client myChecking ipAddr port WITHDRAW amount
```

Rubric

- all fail ==> 0 point
- partially succeed ==> 1 point
- all work ==> 2 points

Notice

1. **You are suggested to test all the above functionalities on the shuttle server before trun in. We will test your codes on them!!**
2. **You are suggested to provide a README file (any format would be OK, e.g., README.txt) to tell TAs how to run your code.**
3. **Please put some comments in your codes!**