Transactions

Revised: v1.0 Sat Sep 22 13:29:27 MDT 2018 Revised: v1.1 Sat Feb 23 18:26:50 MST 2019

In bitcoin and Ethereum you can send funds from one account to another. This activity is captured in a transaction.

In this assignment we are going to add transactions to our blockchain.

You will need to implement code in `../04/bsvr/cli/cli.go` starting at line 183. The function is `func (cc *CLI) SendFundsTransaction( .`

Side Note: notice how line continuation works in go with the declaration of the function.

The function calls `cc.InstructorSendFundsTransaction` remove that. That is the instructors version of the code (The answer that I implemented).

Work through the pseudo code and implement the transaction.

Basically a transaction is finding all the outputs for an account that do not have any corresponding input. These are the unused outputs that represent the value of the account. Fortunately we have an index that tells us where to find these. Verify that there is sufficient funds in the account then create inputs for our new block/new transaction that collects all of the funds. Given that we have the `sum` of the funds, now create 1 or 2 output transactions. First a transaction output to the destination `to` account. If the amount of the funds is larger than the transferred amount then some “change” is owed back to the “from” account. If “change” is needed then create a transaction output with the “change”.

The pseudo code is in a comment in the file `.../cli/cli.go` and it is reproduced below.

```
//
// Pseudo Code:
// 1. Calcualte the total value of the account 'from'.  Call
//    this 'tot'.  You can do this by calling
//    `cc.GetTotalValueForAccount(from)`.
// 2. If the total, `tot` is less than the amount that is to be
//    transfered, `amount` then fail.  Return an error
//        "Insufficient funds".  The person is trying to bounce
//    a check.
// 3. Get the list of output tranactions.
//    Look in the file `.../transaactions/tx.go` for the
//    TxOutputType.  These need to be collected so that you.
//    Call this 'oldOutputs'.
// 4. Find the set of values that are pointed to in the index.
//    These are the values for the 'from' account.  Delete this
```

```
//    from the index.  These are the values that have been spent.
//    ((( To delete from the index use the from value.  Convert it
//              to a stirng (the key for the index
//              cc.BlockIndex.FindValue.AddrIndex  is a string.
//              Then use the builtin "delete" to remove this entire key.
//              "delete(cc.BlockIndex.FindValue.AddrIndex, fromConvertedToStrin
//              You may have to check that the key exists in the "AddrIndex"
//              first ))))
// 5. Create a new empty transaction.  Call `transctions.NewEmptyTx`
//    to create. Pass in the 'memo' and the 'from' for this
//    tranaction.
// 6. Convert the 'oldOutputs' into a set of new inputs.  The
//    type is `../transctions/tx.go` `TxInputType`.  Call
//    `transactions.CreateTxInputsFromOldOutputs` to do this.
// 7. Save the new inputs in the tx.Input.
// 8. Create the new output for the 'to' address.  Call
//    `transactions.CreateTxOutputWithFunds`.  Call this `txOut`.
//    Take `txOut` and append it to the tranaction by calling
//    `transactions.AppendTxOutputToTx`.
// 9. Calcualte the amount of "change" - if it is larger than 0 then
//    we owe 'from' change.  Create a 2nd tranaction with the change.
//    Append to the tranaction the `TxOutputType`.
// 10. Return
//
```

Steps that you will want to use

First you need to get the main program in `.../04/bsvr/main` to compile. This means having the solution to assignment 3 in the `.../04/bsvr/merkle` directory. Also you will need to fix `.../04/bsvr/cli.go` around line 190 to `return` and take out the code that is incomplete. Go to the `.../04/bsvr/main` directory and:

```
go build
```

You should end up with a `main` program or `main.exe` to run.

Now you need to have a genesis block. The code for this is already built. You just have to run it. In the `main` directory run:

```
./main --create-genesis
```

This should create a directory `.../04/bsvr/main/data` with 2 files in it. The file are in JSON format. You should be able to edit the files with a text editor (vi or vim for example) and take a look at them.

The file with the long name (it is the hash of the block) is the first block in the chain. This is the "genesis" block. A bunch of accounts have been created with 50000 tokens each. The other file is `index.json`. It is our index to finding stuff in the set of blocks. Take a look at both files.

If you need to run `./main --create-genesis` again (so that you are starting over with a fresh chain) then you will need to manually delete all the files in the `.../04/bsvr/main/data` directory. I had to do this a bunch of times before I got the transaction code to work.

Other commands that are useful.

You can take a look at the accounts that have been created with:

```
./main --list-accounts
```

and you can find out how much funds are in an account with:

```
./main --show-balance 0x00000SomeAccountNumber
```

When you start out you should have a bunch of accounts with 500000 tokens in each account. The account numbers should look familiar. They are specified in the `./main/cfg.json` file and match with the accounts that Sasha emailed out.

Search for Code

Go and find all the stuff that already exists.

1. in `.../04/bsvr/cli/cli.go` look for:
    1. GetTotalValueForAccount(acct)
    2. GetNonZeroForAccount(acct)
2. in `.../04/bsvr/transactions/tx.go` look for:
    1. NewEmptyTx(memo, acct)
    2. CreateTxInputsFromOldOutputs(oldOutputs)
    3. CreateTxOutputWithFunds(acct, amount)
    4. AppendTxOutputToTx(txToAppendTo, newTxOutputs)
3. Look at how the index is stored in `.../04/bsvr/main/data/index.json` and how this is stored in memory. Look at `.../04/bsvr/index/index.go`.
4. Assume that the author is not so good at writing comments. Add your own notes. Look at test cases and see how stuff works. Add your own test cases. Add print statements to the code to see how things work.