Project: Toy Blockchain
Author: Thomas Gordon

Achievements:

- Design and implementation of data structures is completed,
  consisting of two records and a chain composed of blocks.
  One record, Trans, handles data the block contains--a sender, a receiver,
  and a message--it is a simple transactional ledger. Another record handles
  information about the block itself, including the Trans record.
  This is record is called simply Block since it models the block design.
  A block consists of a depth, a time stamp, a transactional record, a hash,
  the hash of its parent block, and a nonce which is integer that corresponds to the
  amount of work needed to generate the block.

- Design and implementation of the blockchain is nearing completion.
  Each block in the blockchain points to its parent block. Therefore there
  must be a root block that starts off the chain. This block is commonly
  referred to as the genesis block. Once the genesis block is generated,
  blocks can be mined one at a time, forming a sequence, or “chain”.
  The genesis block was fairly straightforward, although it lead to minor
  changes in the Block record. Mining the block proved more difficult.
  The difficulties arose out of how I handled filling in the Trans and getting a
  timestamp. The functions getTrans, whose type is IO Trans, and getStamp, with
  type IO int, meant I had to read about MonadIO. I ended up using liftIO
  to pull (or box?) the output of both functions into the block record.
  This feels hacky and I’ve also read the using MonadIO can be problematic.
  That is to say, mineBlock may be in a tentative form.

- Hashing.
  During the project proposal, we concluded that hashing would be
  extra. Yet, as stated above, a blockchain points to it’s parent hash. This can
  seen as an imperative feature. Not implementing a hash would mean
  implementing something very similar to a hash which felt unnecessary.
  Currently the hashing works. It took going through several libraries until I found
  one that worked, was maintained, and had the features I wanted. I’m currently
  using SHA256. Since this is a toy blockchain the flavor of hash is almost
  arbitrary. I chose SHA256 since it is arguably the most commonly known.
  Currently the digest is utterly simple, comprising of a hash of the parent block’s
  hash, the grandparent hash, and the timestamp of the parent. This feels a little
  too arbitrary for me. So, while it does work, I’d like to have it feel a little more
  meaningful.

Roadmap:

- Networking.
  This is the brunt of the work. I can create a blockchain
  locally, a blockchain is by definition a distributed ledger. Currently looking into a framework where this can be done while also handling concurrency issues. I'm aware that one solution is to use docker to provide a stable environment and automate networking (openVPN?). But I'd rather everything be handled using haskell. If I can run one instance on a VM and send a message, SUCCESS!
  This challenge can be further broken down into a few steps that will
  also help clear up any ambiguity about how the blockchain is intended to function.
  A. Query for a blockchain. If none are found, create a genesis block.
  B. Update blockchain metadata by querying all instances of the chain.
  C. If instances are not equal, update to the newest instance.
     Note:
     A,B,C can be reduced to "each node queries each node, checking the status of the chain so to keep a consistent, current chain"

- Refactoring code, fixing design problems and bugs.
  This is usually an ongoing process, there will be a point where I will
  look over my whole code and make improvements/fixes that I haven't
  gotten around to. Often I'll have a process that works but I am not satisfied with.

- CLI.
  This is probably going to be the last thing before a final review.
  Currently the blockchain has no interface. You can interact by calling
  mineBlock. Of course, the executable should run as a command-line utility, behaving something like this:
  ./<blockchain> -m "Hello World!" -r Marg.Atwood