to run Raft (part 1 and 2):

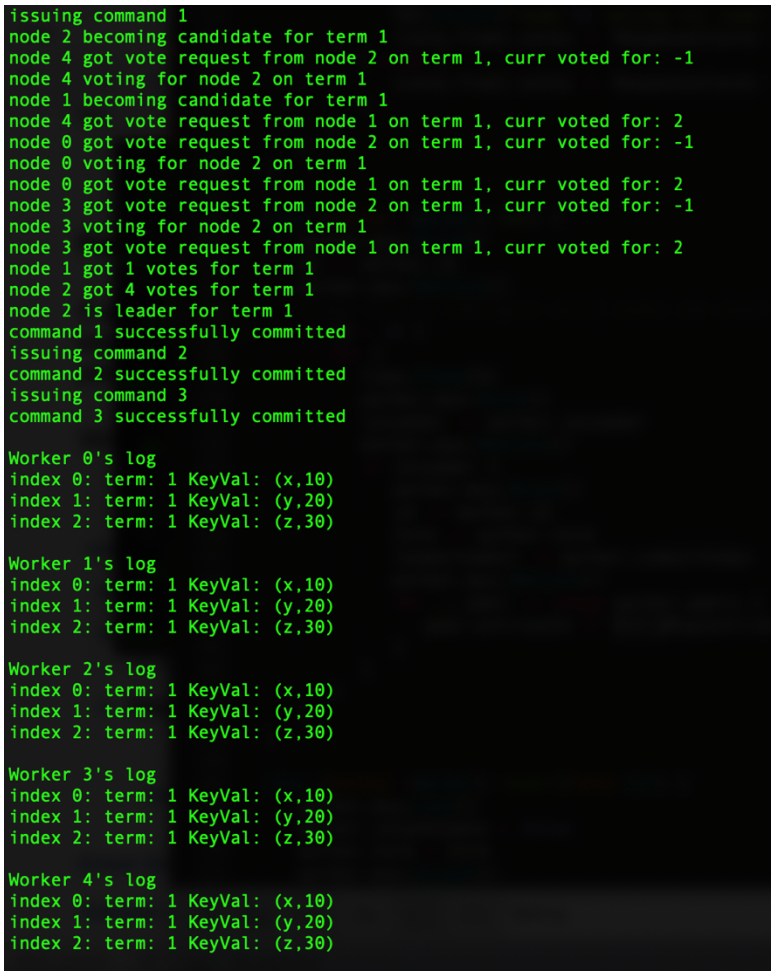
go build raft.go peer.go persister.go
./raft

To run consistent hashing (part 3):

go run hashing.go node.go
./hashing

Raft:
main will create 5 nodes using the Make() function and they will elect a leader.
main then issues three commands to an arbitrary worker to be committed to the log using the Start() function. This worker waits for a leader to be elected and then forwards the command to the leader to be committed. The leader then appends the new entry to its own log and sends append messages to the other nodes as specified by the Raft paper to ensure log consistency. If the leader gets a majority of successful log appends, it commits the entry, triggering ApplyMsg signals to be sent by the workers to the client. The implementation also saves the state of each worker to the persister object provided to the Make() function and uses this state to initialize if it is not empty.

sample run: node 1 and 2 both become candidates for term 1, node 2 is elected leader and begins append commands, then the final state of the logs are displayed



To show that a new leader will be elected if the current leader fails, on this run only node 3 was able to send heartbeat messages. When nodes with a different id are elected as leader, the other nodes receive no heartbeat and will start new elections until node 3 is elected. When the client doesn't receive an ApplyMsg after a period of time because the leader has failed before committing, it re-issues the failed command. The logs are kept consistent. In this case leader 1 is able to commit command 1 before the other nodes detect a failure. Node 4 is then elected and fails before committing the second command, so it is re-issued and committed by the new leader 3, which then also commits the third command.

```
issuing command 1
node 1 becoming candidate for term 1
node 4 got vote request from node 1 on term 1, curr voted for: -1
node 0 got vote request from node 1 on term 1, curr voted for: -1
node 3 got vote request from node 1 on term 1, curr voted for: -1
node 2 got vote request from node 1 on term 1, curr voted for: -1
node 1 got 5 votes for term 1
node 1 is leader for term 1
command 1 successfully committed
issuing command 2
node 3 becoming candidate for term 2
node 4 got vote request from node 3 on term 2, curr voted for: -1
node 4 becoming candidate for term 3
node 3 got vote request from node 4 on term 3, curr voted for: -1
node 0 got vote request from node 3 on term 2, curr voted for: -1
node 0 got vote request from node 4 on term 3, curr voted for: -1
node 2 got vote request from node 3 on term 2, curr voted for: -1
node 2 got vote request from node 4 on term 3, curr voted for: -1
node 1 got vote request from node 4 on term 3, curr voted for: -1
node 4 got 4 votes for term 3
node 4 is leader for term 3
node 3 becoming candidate for term 4
node 0 got vote request from node 3 on term 4, curr voted for: -1
node 1 got vote request from node 3 on term 4, curr voted for: -1
node 2 got vote request from node 3 on term 4, curr voted for: -1
node 3 got 3 votes for term 4
node 3 is leader for term 4
leader crashed before commit, re-issuing command
command 2 successfully committed
issuing command 3
command 3 successfully committed

Worker 0's log
id: 0, index 1: term: 1 KeyVal: (x,10)
id: 0, index 2: term: 4 KeyVal: (y,20)
id: 0, index 3: term: 4 KeyVal: (z,30)

Worker 1's log
id: 1, index 1: term: 1 KeyVal: (x,10)
id: 1, index 2: term: 4 KeyVal: (y,20)
id: 1, index 3: term: 4 KeyVal: (z,30)

Worker 2's log
id: 2, index 1: term: 1 KeyVal: (x,10)
id: 2, index 2: term: 4 KeyVal: (y,20)
id: 2, index 3: term: 4 KeyVal: (z,30)

Worker 3's log
id: 3, index 1: term: 1 KeyVal: (x,10)
id: 3, index 2: term: 4 KeyVal: (y,20)
id: 3, index 3: term: 4 KeyVal: (z,30)

Worker 4's log
id: 4, index 1: term: 1 KeyVal: (x,10)
id: 4, index 2: term: 4 KeyVal: (y,20)
id: 4, index 3: term: 4 KeyVal: (z,30)
```