peerIDList []
Log[] : array of struct Log {term, data}
majority : majority count for current state of distributed system

state: Leader

on **Append (data:[] byte):**

```
LogStore (sm.index, data []byte)
for all peerID i in peerID list {
        actionresult = Send (peerID[i], AppendEntriesReq (sm.ID, sm.term,
sm.index-1, sm.Log[sm.index-1].term, sm.Log[sm.index], sm.commitIndex ) )
        if actionresult == “OK”
                count = count + 1
}

if count >= majority
        Commit (sm.index, outputdata, 0: successful)
        sm.commitIndex = sm.index
        write Log[sm.commitIndex] to disk
else
        Commit (sm.index, outputdata, 1:unsuccessful)

sm.index = sm.index + 1
```

on **Timeout:**

```
for all peerID i in peerID list {
        Send (peerID[i], AppendEntriesReq (sm.ID, sm.term, sm.index-1,
sm.Log[sm.index-1].term, NIL , sm.commitIndex ) )
}
```

on **AppendEntriesReq (leaderID, term, previndex, prevterm, data , commitIndex)):**

```
if sm.term > msg.term,
        Send (msg.from, AppendEntriesResp (sm.term, sm.term, success = false) )
else
        state = follower
        sm.term = msg.term
        Send (sm.ID, AppendEntriesReq (msg.leaderID, msg.term, previndex,
msg.prevterm, msg.data , msg.commitIndex ) )
```

on **AppendEntriesResponse (from, term, success):**

```
if success == false,
        if msg.term > sm.term,
```

```
                    sm.term = msg.term
                    sm.state = “follower”
            else
                    nextIndex[msg.from] = nextIndex[msg.from]-1
                    Send(msg.from, AppendEntriesReq (sm.ID, sm.term,
nextIndex[msg.from], sm.Log[nextIndex[msg.from]].term, sm.Log[nextIndex[msg.from]:] ,
sm.commitIndex))
    else
            update sm.matchIndex[msg.from] to sm.index of msg.from


on VoteReq (from, term, candidateId, lastLogIndex, lastLogTerm):
    if sm.term >= msg.term,
            send (msg.from, VoteResp (sm.term, voteGranted = no))
    else
            sm.state = “follower”
            sm.term = msg.term
            if(sm.Log[sm.index].term < msg.lastLogTerm) or (sm.Log[sm.index].term ==
    msg.lastLogTerm && sm.index <= msg.lastLogIndex)
                    votedFor = msg.from
                    send (msg.from, VoteResp (sm.term, voteGranted = yes))
            else
                    send (msg.from, VoteResp (sm.term, voteGranted = no))

on VoteResp (term, voteGranted)
{
    //Received from minority servers. Nothing to do.
}


state: follower

on Append (data [] byte):
{
    Redirect Append to leader.
}


on Timeout:
    sm.term = sm.term + 1
    sm.state = “candidate”
    sm.votedFor = sm.ID
    Reset the election timer.

    for all peerID i in peerID list {
```

```
                Send (peerID[i], VoteReq (sm.ID, sm.term, sm.ID, sm.index,
sm.Log[sm.index]))
        }
on AppendEntriesReq (leaderID, term, previndex, prevterm, data , commitIndex ) ):
        if sm.term > msg.term,
                send (msg.leaderID, AppendEntriesResponse (sm.ID, sm.term, success =
false))
        else,
                sm.term = msg.term
                if (msg.prevterm == sm.Log[previndex].term)
                        insert data from sm.Log[previndex + 1] onwards
                        Modify sm.index according to previndex + no. of log entries in data
                        send (msg.leaderID, AppendEntriesResponse (sm.ID, sm.term,
success = true))
                else
                        send (msg.leaderID, AppendEntriesResponse (sm.ID, sm.term,
success = false))


on AppendEntriesResponse (from, term, success):
        if (sm.term < msg.term)
                sm.term = msg.term



on VoteReq(from, term, candidateId, lastLogIndex, lastLogTerm):
         if sm.term >= msg.term && sm.votedFor == nil or msg.candidateId,
                if(sm.Log[sm.index].term < msg.lastLogTerm) or (sm.Log[sm.index].term ==
        msg.lastLogTerm && sm.index <= msg.lastLogIndex),
                        sm.term = msg.term
                        sm.votedFor = msg.candidateId
                        action = Send(msg.from, VoteResp(sm.term, voteGranted =no))

        Otherwise, reject vote: action = Send(msg.from, VoteResp(sm.term, voteGranted
=no))


on VoteResp (term, voteGranted)
{
        if ( sm.term < msg.term )
                sm.term = msg.term
}
```