1.
The change makes the Tiny language "forget" any change to occurred to the state, during the evaluation of E2. Instead, the final state of the evaluation of the \( \leq \) would be the same as the state that resulted after evaluating E1.

The only change that EE can effect upon any state (memory,input,output) is upon the input, i.e. EE cannot change the memory or the output (CC does that). So, our program must contain a 'read' as the right operand of a '\( \leq \)'. The following program will do:

```
program x:
    assign i := (1 \leq read);
    output(read);
end x.
```

Assuming the input is, say, (2,3), the program should assign 1 \leq 2 to i, and print 3. However, with the change proposed, after the assignment the state contains the input AS IT WAS BEFORE THE FIRST READ. With modified semantics, the program still assigns 1 \leq 2 to i, but prints 2

2.
a) EE(read)=(fn(m,i,o).Null i ->error |
               fn(m,i,o). (not Isinteger (Head i)) -> error
               | (Head i, (m, Tail i, o))
remove the original EE(read) semantics

b) EE(read)=(fn(m,i,o).Null i ->error |
              fn(m,i,o). (not Isinteger (Head i)) -> ((m,Tail i,o)=>EE(read))
              |(Head i, (m, Tail i, o))
remove the original EE(read) semantics

3. First, add an item to the C syntactic domain:

\[
C = \ldots \mid \langle \text{do } C \ E \rangle
\]

Then add a case in the definition of CC. The trick is to notice that
do C while E

is equivalent to

C;
while E do C;

So,

CC[<do C E>] = CC[<; C <while E C>]