```
printMemory([]).
printMemory([[Key|Val]]) :- writef(Key), write(' = '), write(Val).
printMemory([[Key|Val]|T]) :- writef(Key), write(' = '), write(Val), nl, printMemory(T).

update(K,V,[[K|Val]|T], Store) :- Store = [[K|V]|T].
update(K,V,[[Key|Val]|T], Store) :- Store = [[Key|Val]|S], update(K,V,T,S).
update(K,V,[], Store) :- Store = [[K|V]].

          /* split Words into  ["while"] + E + [":"] + CL +  ["end"] : */
parseCmd(Words, while(ETree, CList)) :-  append(W1,W3,Words), append(["while"], ETree1, W1), append([":"],CList1,
            W2), append(W2, ["end"],W3), parseExpr(ETree1, ETree), parseCmd(CList1, CList).

          /* split up  Words  into   [I] + ["="] + E  :  */
parseCmd(Words, assign(I, ETree)) :-  append(I1, R1, Words), append(["="], E1, R1), parseExpr(I1, I), parseExpr(E1,
            Etree), nl.

         /* split up  Words  into  ["print"] + [I]  :  */
parseCmd(Words, print(I)) :- append(["print"], E1, Words), parseExpr(E1, I),nl.

lookup(K, leaf, V).
lookup(K, upd(Key, Value, DB), V) :- K = Key, print(Value).
lookup(K, upd(Key, Value, DB), V) :- lookup(K, DB, V).
lookup(K, node(Key, Value, L, R), V) :- K = Key, print(Value).
lookup(K, node(Key, Value, L, R), V) :- K < Key, lookup(K, L, V).
lookup(K, node(Key, Value, L, R), V) :- K > Key, lookup(K, R, V).

collect(leaf, L).
collect(upd(K, V, DB), L) :- remove(K, DB, RL), append([(K,V)], RL, L).
collect(node(K,V,Left,Right), L) :- remove(K, Left, T1), append([K, V], T1, Ans1), remove(K, Right, T2), append(Ans1,
            T2, L).

remove(K, [], Ans) :- !.
remove(K, leaf, Ans).
remove(K, [(K,V)|L], Ans) :- !, remove(K, L, Ans).
remove(K, [(Key,Val)|L], Ans) :- append([(Key, Val)], Ans, X), remove(K, L, X).
remove(K, node(K,V,Left,Right), Ans) :- remove(K, Left, Ans), remove(K, Right, Ans).
remove(K, node(Key,V,Left,Right), Ans) :- append([(Key, V)], Ans, X), remove(K, Left, X), remove(K, Right, X).

removedups(L, M) :- nodups(L, M), !.
nodups([], []).
nodups([(K,V)|R], [(K,V)|T]) :- nodups(R, S), select((K,_), S, T).
nodups([(K,V)|R], [(K,V)|T]) :- nodups(R, T).

lookup(K) :- upd(K, V), write(V), !.
lookup(K) :- mapsTo(K, V), write(V).

commit :- findall((K, V), upd(K, V),  List), findall((K, V), mapsTo(K, V), List2), append(List, List2, Total), write(Total),
            removedups(Total, Ans), retractall(upd(K,_)), retractall(mapsTo(K,_)), addDB(Ans).

hasFine(Who, Today, Howmuch) :- findall(X, (borrowed(Key, Who, _), fineOf(Key, Today, X)),List), sum(List, Howmuch).
```