Mandatory assignment 1

Task 1

```
find(d) {
  i := head; # variable i is local to the current method instance
  while (i != null and i.val != d) { i := i.next; }
  # return i
}
```

```
insert(new) {
  if (tail == null) { # empty list
    head := new;
  } else {
    tail.next := new;
  }
  tail := new;
}
```

```
delfront() {
  if (head != null) { # list not empty
    if (head == tail) { tail := null; } # only 1 elem. in list
    head := head.next;
  }
}
```

(A)

- V(find(d)) = {head}
- W(find(d)) = Ø

(B)

- V(insert(new)) = {tail, head, new, tail.next}
- W(insert(new)) = {head, tail.next, tail}

(C)

- V(delfront()) = {head, tail, head.next}
- W(delfront()) = {head, tail}

Task 2

a

W_B disjoint V_A = {head} W_A disjoint V_B = Ø {head} union Ø = **{head}**

W_B disjoint V_C = {head, tail} W_C disjoint V_B = {head, tail} {head, tail} union {head, tail} = **{head, tail}**

W_C disjoint V_A = {head} W_A disjoint V_C = Ø {head} union Ø = **{head}**

find() in combination with insert() or delfront() has interference based on the results from task 2.a. In addition they do not follow the AMO-property for routines since the find process might return an item that should have been deleted or should have been added.

As for the combination of insert and delfront. There is two shared write and read variables. This might destroy the structure of the linked list and change the processes totally.

b

Based on the last task then every combination of processes should be done one at a time. find & insert:

- find; insert;
- find should return null if insert inserts d AFTER executing find
- Might return d if insert is faster than find can read head

find & delfront:

- find; delfront;

- find should return d if delfront deleted the head/d AFTER executing find
- Might return null if delfront is faster than find can read head

delfront & insert:

- delfront; insert;
- delfront should remove the head and insert should insert a new head AFTER delfront has deleted
- (case for 1 element) delfront updates head/tail to be null while insert has already read head to be not null and tries `tail.next := new` but ends up with a nullpointer exception because tail is null.