Question 4

Part a)

As per the coursebook story is, Jenny painted the wall white.

Integrients are - color = {white, yellow}, wall = {w}, fluent = on{c,w} and action is paint(c,w). The story in AL will be based on action causes fluent and two colors cannot be on same wall or painting new color on wall will remove the old color.

Paint(c,w) **causes** *on(c,w)*
-on(c1,w) **if** *paint(c2,w), c1 ≠ c2*

Here- c: color, w: wall

Part b)

After applying part a in AL, I translated the program in ASP, including the initial condition given in the question and the action acted as painted with white color on the wall from yellow color.

The output is:

```
holds(on(yellow,w),0) holds(on(white,w),1)
```

The description of the program is given in comments, please refer comments-

```
% color:
color(yellow).
color(white).

%wall
wall(w).

#const n = 1.
step(0..n).

%fluent
fluent(inertial, on(C,W)) :- color(C), wall(W).
%action
action(paint(C,W)) :- color(C), wall(W).

% a causes f
holds(on(C,W), I+1) :- color(C), wall(W),
                       occurs(paint(C,W), I), I<n.

% painting new color on wall will remove the old color
-holds(on(C1,W), I+1) :- color(C1), color(C2), wall(W),
                         occurs(paint(C2,W), I), C1 != C2.
```

```
% Inertia:   standard
holds(F,I+1) :- fluent(inertial,F), holds(F,I), not -holds(F,I+1), I < n.
-holds(F,I+1) :- fluent(inertial,F),
                    -holds(F,I), not holds(F,I+1), I < n.

% CWA action
-occurs(A,I) :- action(A), step(I), not occurs(A,I).

% Initial condition as per the book
holds(on(yellow, w), 0).
%action given at step 0
occurs(paint(white,w), 0).

#show holds/2.
```

Part c)

Here as per question, one more color is added as BLACK and after painting the wall with white from yellow, it has again painted black, i.e. both actions are performed sequentially. Hence the program outputs as:

```
holds(on(yellow,w),0) holds(on(white,w),1) holds(on(black,w),2)
```

The description of the program is given in comments, please refer comments –

```
%color
color(yellow).
color(white).
%also have black paint
color(black).

%wall
wall(w).

#const n = 2.
step(0..n).

%fluent
fluent(inertial, on(C,W)) :- color(C), wall(W).
%action
action(paint(C,W)) :- color(C), wall(W).

% a causes f
holds(on(C,W), I+1) :- color(C), wall(W),
                    occurs(paint(C,W), I), I<n.
```

```
% painting new color on wall will remove the old color
-holds(on(C1,W), I+1) :- color(C1), color(C2), wall(W),
                    occurs(paint(C2,W), I), C1 != C2.

% Inertia:   standard
holds(F,I+1) :- fluent(inertial,F),
            holds(F,I),
            not -holds(F,I+1),
            I < n.

-holds(F,I+1) :- fluent(inertial,F),
                                        -holds(F,I),
             not holds(F,I+1),
             I < n.

% CWA action
-occurs(A,I) :- action(A), step(I), not occurs(A,I).

% Initial condition as per the book
holds(on(yellow, w), 0).
%action given at step 0
occurs(paint(white,w), 0).
%action given at step 1, i.e. painted wall black now
occurs(paint(white,w), 1).

#show holds/2.
```

Part d)

Now, a new person added to the sort, i.e. Jill and also the action type also changed as person action on paint with color c on wall w. Also, with the constraint that two people cannot paint the same wall at the same time, this is a concurrency situation for painting the walls

Hence new AL is :
Paint(PCW) **cause** *on (C,W)*
-on(C1,W) **if** *paint(P,C2,W), C1 != C2*
Imposibble *paint(P1,C1,W), paint(P2,C2,W)*

The program outputs as –

```
holds(on(yellow,w1),0) holds(on(white,w1),1) holds(on(black,w1),2)
holds(on(white,w2),1) holds(on(white,w2),2)
```

The description of the program is given in comments, please refer comments. The code is –

```
% color:
color(yellow).
```

```
color(white).
%also have black paint
color(black).

%wall
wall(w1).
wall(w2).

% two person jenny and jill
person(jenny).
person(jill).

#const n = 2.
step(0..n).

%fluent
fluent(inertial, on(C,W)) :- color(C), wall(W).
%action
action(paint(P,C,W)) :- person(P), color(C), wall(W).

% a causes f
holds(on(C,W), I+1) :- person(P),color(C), wall(W),
                       occurs(paint(P,C,W), I), I<n.

% painting new color on wall will remove the old color
-holds(on(C1,W), I+1) :- person(P), color(C1), color(C2), wall(W),
                         occurs(paint(P,C2,W), I), C1 != C2.

% Inertia:   standard
holds(F,I+1) :- fluent(inertial,F), holds(F,I), not -holds(F,I+1),I < n.

-holds(F,I+1) :- fluent(inertial,F), -holds(F,I),  not holds(F,I+1), I < n.
% CWA action
-occurs(A,I) :- action(A), step(I), not occurs(A,I).

%concurreny condition, two people can't paint the same wall at same time but can paint different wall
-occurs(paint(P1,C,W), I) | -occurs(paint(P2,C,W), I) :- person(P1), person(P2), color(C),
                                                         wall(W), step(I), action(paint(P1,C,W)),
                                                         action(paint(P2,C,W)), P1 != P2.
% Initial condition as per the book
holds(on(yellow, w1), 0).
%action given at step 0
occurs(paint(jenny,white,w1), 0).
%action jill painted another wall as white paint (given)
occurs(paint(jill,white,w2), 0).
%action given at step 1, i.e. painted wall black now
occurs(paint(jenny,black,w1), 1).
```

#show holds/2.

Question 5

Part a)

As per the coursebook story is, Claire always carries her cell phone with her. Claire is at the library

The signature integrients are – person, object, location, fluent as carried(Obj, Person) (hint in the book), ObjAt(Obj, location), personAt(person, location) and action is defined as went(person, location).

In the question it is also provided that the AL should support when the object type i.e. the phone is changed to a certain pet.

The terms used below are- P:person, L: location

From the above, the AL of the story can be formed as-
Went(P,L) **causes** *personAt(P,L)*
ObjAt(Obj,L) **if** *personAt(P,L)*
-carried(Obj1,P) **if** *carried(Obj2,P), Obj1 != Obj2*
-personAt(P,L1) **if** *personAt(P,L2), L1 != L2*
-personAt(P,L1) **if** *went(P, L2), L1 != L2.*

Above the *Obj* can be phone or pet.

Part b)

Designing part a in ASP, gave me the ouput as -

```
holds(carried(phone,claire),0) holds(personAt(claire,library),0)
holds(personAt(claire,home),1) holds(carried(phone,claire),1)
```

When I changed the object phone to chihuahua, the output came out as –

```
holds(carried(chihuahua,claire),0) holds(personAt(claire,library),0)
holds(personAt(claire,home),1) holds(carried(chihuahua,claire),1)
```

The description of the design of code is in the comments only, please refer comments –

% object:
obj(phone).
obj(chihuahua).

%location home and library
location(home).
location(library).

%person
person(claire).
#const n = 1.

```
step(0..n).

%fluent
fluent(inertial, carried(Obj,P)) :- obj(Obj), person(P).
fluent(inertial, personAt(P,L)) :- location(L), person(P).
fluent(defined, objAt(Obj,L)) :- obj(Obj), location(L).
%action
action(went(P,L)) :- person(P), location(L).

% a causes f from part a)
holds(personAt(P,L), I+1) :- person(P),location(L),
                    occurs(went(P,L), I), I<n.

% object is to the location if carried by the person on a location
-holds(objAt(Obj,P), I) :- person(P), location(L), obj(Obj),
                  holds(personAt(P,L),I), holds(carried(Obj,P),I).

% one object is carried to the location by person
-holds(carried(Obj1,P), I) :- person(P), obj(Obj1), obj(Obj2),
                    holds(carried(Obj2,P),I), Obj1 != Obj2 .

% person will be at one location only
-holds(personAt(P,L1), I) :- person(P), location(L1), location(L2),
                    holds(personAt(P,L2),I), L1 != L2 .

%constraint for defined and inertial
-holds(F,I) :- fluent(defined,F), step(I), not holds(F,I).

% Inertia:   standard
holds(F,I+1) :- fluent(inertial,F), holds(F,I), not -holds(F,I+1), I < n.

-holds(F,I+1) :- fluent(inertial,F), -holds(F,I),  not holds(F,I+1),  I < n.

% CWA action
-occurs(A,I) :- action(A), step(I), not occurs(A,I).

% Initial condition as per the book
holds(carried(phone,claire), 0).
% obj phone changed to chihuahua
%holds(carried(chihuahua,claire), 0).
holds(personAt(claire, library),0).
%action given at step 0
occurs(went(claire, home), 0).

#show holds/2.
```

Part c)

Now the question demands that a new person named Rod is added, who carries a towel with him everywhere. Also, Rod and Claire are never at same place at the same time, hence the phone and towel also cannot be at same location. Hence the AL of the program will be changed with different persons included.

impossible *went(P1,L), went(P2, L), P1 != P2*
-personAt(P1,L) **if** *personAt(P2,L), P1 != P2*
-ObjAt(Obj1,L) **if** *ObjAt(Obj2,L), Obj1 != Obj2*

Based on the above AL , the desired output of the program is –

```
holds(carried(phone,claire),0) holds(carried(towel,rod),0) holds(personAt(claire,library),0)
holds(personAt(claire,home),1) holds(personAt(rod,library),1) holds(carried(towel,rod),1)
holds(carried(phone,claire),1)
```

The description of the program is given in comments, please refer comments –

```
% object:
obj(phone).
obj(chihuahua).
obj(towel).

%location home and library
location(home).
location(library).

%person
person(claire).
person(rod).

#const n = 1.
step(0..n).

%fluent
fluent(inertial, carried(Obj,P)) :- obj(Obj), person(P).
fluent(inertial, personAt(P,L)) :- location(L), person(P).
fluent(defined, objAt(Obj,L)) :- obj(Obj), location(L).
%action
action(went(P,L)) :- person(P), location(L).

% a causes f
holds(personAt(P,L), I+1) :- person(P),location(L),
                    occurs(went(P,L), I), I<n.

% object is to the location if carried by the person on a location
-holds(objAt(Obj,P), I) :- person(P), location(L), obj(Obj),
                           holds(personAt(P,L),I), holds(carried(Obj,P),I).
```

```
% one object is carried to the location by person
-holds(carried(Obj1,P), I) :- person(P), obj(Obj1), obj(Obj2),
                              holds(carried(Obj2,P),I), Obj1 != Obj2 .

% person will be at one location only
-holds(personAt(P,L1), I) :- person(P), location(L1), location(L2),
                              holds(personAt(P,L2),I), L1 != L2 .

%constraint for defined and inertial
-holds(F,I) :- fluent(defined,F), step(I), not holds(F,I).

% Inertia:   standard
holds(F,I+1) :- fluent(inertial,F), holds(F,I), not -holds(F,I+1), I < n.

-holds(F,I+1) :- fluent(inertial,F), -holds(F,I), not holds(F,I+1), I < n.

% CWA action
-occurs(A,I) :- action(A), step(I), not occurs(A,I).

% Two persons cannot be at same location after moving from a location
-occurs(went(P1,L),I) | -occurs(went(P2,L),I) :- person(P1), person(P2), location(L), step(I),
                                                 action(went(P1,L)), action(went(P2,L)),  P1 != P2.

% person cannot be at same location at same time
-holds(personAt(P1,L),I) :- person(P1), person(P2), location(L), holds(personAt(P2,L),I), P1 != P2.

% two objects cannot be at same location
-holds(objAt(Obj1,L),I) :- obj(Obj1), obj(Obj2), location(L), holds(objAt(Obj2,L),I), Obj1 != Obj2.

% Initial condition as per the book
holds(carried(phone,claire), 0).
% obj phone changed to chihuahua
%holds(carried(chihuahua,claire), 0).

holds(carried(towel, rod), 0).
holds(personAt(claire,library),0).

%action given at step 0
occurs(went(claire, home), 0).
occurs(went(rod, library), 0).

#show holds/2.
```