a) max = fn : int list → int

```
fun max(nil) = 0
  | max(y::ys) = if y > max(ys) then y
                 else y = max(ys)
```

a) gut, 1/1

b) nth = fn : int list * int → list

```
fun nth(nil, x) = 0
  | nth(y::ys, x) if x = 1 then y
                  else nth(ys, x-1)
```

x sollte vermutlich 0 sein, außer man Arbeitet mit Sprachen wie matlab wo Listen mit 1 beginnen 2/2

c) reverse = fn : 'a list → 'a list

```
fun reverse(y::ys) if length(l)>2 then reverse(ys) : y
                   else ys : y
```

den :: Operator kann man hier nicht verwenden, da muss die Liste immer rechts stehen, muss hier also reverse(xs) @ [x] sein, anstelle von length besser mit Patternmatching arbeiten

es fehlt noch mindestens das Match für die leere Liste

c) 1/2

d) insert = fn : 'a * 'a list * ('a * 'a → bool) → 'a list

```
fun insert(x, y:ys, op) = (x, y:ys, op(x, y))
  | insert(x, y:ys, op) if op(x, y) = true
                        then x::y::ys
                        else insert(x, ys, op)
```

ergibt für Rückgabe nicht so viel Sinn

hier fehlt wieder mindestens ein Match

```
| op(x, y) if x (op<) y then true
           else false
```

was ist das?

d) 1/2

(2)

```
haschild(alice, charlie).
haschild(bob, charlie).
haschild(edith, alice).
woman(alice).
woman(edith).
man(bob).
man(charlie).

parent(X) :- haschild(X, _), (woman(X); man(X)).
grandparent(X) :- haschild(X, Y), haschild(Y, _), (woman(X); man(X)).
father(X) :- haschild(X, _), man(X).
mother(X) :- haschild(X, _), woman(X).
grandfather(X) :- grandparent(X), man(X).
grandmother(X) :- grandparent(X), woman(X).
```

- ?- woman(alice). true.
- ?- man(alice). false.
- ?- haschild(X, charlie). X = alice; X = bob.
- ?- parent(X). X = alice; X = bob; X = edith.
- ?- grandmother(X). X = edith.
- ?- parent(charlie). X = alice; X = bob.

muss false sein, charlie hat kein Kind und kann daher kein parent sein

```java
package de.uni_oldenburg.inf.omp.rulebook.labyrinth.Aufgabe3;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

import com.deliveredtechnologies.rulebook.Fact;
import com.deliveredtechnologies.rulebook.FactMap;
import com.deliveredtechnologies.rulebook.Result;
import com.deliveredtechnologies.rulebook.annotation.Given;
import com.deliveredtechnologies.rulebook.annotation.Rule;
import com.deliveredtechnologies.rulebook.annotation.Then;
import com.deliveredtechnologies.rulebook.annotation.When;
import com.deliveredtechnologies.rulebook.model.runner.RuleBookRunner;

public class Labyrinth {
    @Rule
    public class RuleNavigation {
        private Chamber chamber;
        @Given
        private List<Chamber> chambers;
        @When
        public boolean when () {
            return chamber.hasAdjacent();
        }
        @Then
        public void then () {
            System.out.println("Moving to chamber" + chamber.getName() + ".");
            if (chamber.isTreasure()){
                System.out.println("Treasure found!");
                chamber.setTreasure(true);
            } else {
                chamber.getRandomAdjacent().addAdjacent(chamber);
            }
        }
    }
    public static void main(String[] args) {
        /*
         * A-B-C D
         * |   | |
         * E F-G-H
         * |   |
         * I-J-K-L
         *   |   |
         * M-N-O-P
         */
        Map<Character, Chamber> labyrinth = new HashMap<>();
        for (char n = 'A'; n <= 'P'; n++) {
            labyrinth.put(n, new Chamber(Character.toString(n)));
        }
        labyrinth.get('A').addAdjacent(labyrinth.get('B'));
        labyrinth.get('B').addAdjacent(labyrinth.get('C'));
        labyrinth.get('A').addAdjacent(labyrinth.get('E'));
        labyrinth.get('C').addAdjacent(labyrinth.get('G'));
        labyrinth.get('D').addAdjacent(labyrinth.get('H'));
        labyrinth.get('F').addAdjacent(labyrinth.get('G'));
        labyrinth.get('G').addAdjacent(labyrinth.get('H'));
```

chamber muss irgendwo gesetzt werden, Rückgabewert der Regel fehlt, 5/6

```
58          labyrinth.get('E').addAdjacent(labyrinth.get('I'));
59          labyrinth.get('G').addAdjacent(labyrinth.get('K'));
60          labyrinth.get('I').addAdjacent(labyrinth.get('J'));
61          labyrinth.get('J').addAdjacent(labyrinth.get('K'));
62          labyrinth.get('K').addAdjacent(labyrinth.get('L'));
63          labyrinth.get('J').addAdjacent(labyrinth.get('N'));
64          labyrinth.get('L').addAdjacent(labyrinth.get('P'));
65          labyrinth.get('M').addAdjacent(labyrinth.get('N'));
66          labyrinth.get('N').addAdjacent(labyrinth.get('O'));
67          labyrinth.get('O').addAdjacent(labyrinth.get('P'));
68          labyrinth.get('A').setEntrance(true);
69          labyrinth.get('P').setTreasure(true);
70          Chamber next = labyrinth.get('A');
71          FactMap<Chamber> facts = new FactMap<>();
72          RuleBookRunner ruleBook = new RuleBookRunner("de.uni_oldenburg.inf.
    omp.rulebook.labyrinth");
73          while (next != null) {
74              facts.clear();
75              facts.setValue("chamber", next);
76              ruleBook.run(facts);
77              next = (Chamber) ruleBook.getResult().orElse(new Result<>(null)).getValue
    ();
78          }
79
80      }
81
82  }
83
```