LAB 5

The following rules and questions are given:

Rules:

-If patient has cough and patient has infection then patient has pneumonia.
-If temperature is more than 38 then patient has fever.
-If patient has muscle pain and patient has fever then patient has flu.
-If patient was sick for at least 2 days and patient has fever then patient has infection.

Questions:

-What is patient temperature? (the answer is a number)
-For how many days has the patient been sick? (the answer is a number)
-Has patient muscle pain? (the answer is yes/no)
-Has patient cough? (the answer is yes/no)

The goal:

-The patient has pneumonia.

First approach:

1. The program interface should address the questions to the user. Based on the answers, the system will know whether or not the patient has cough, the temperature is more than 38 and patient was sick for at least 2 days and the patient has muscle pain.

2. The knowledge must be expressed as positive Horn propositional clauses, in the form [[n(w), s, n(p)], [a, n(w), n(r), n(t)], [q]]. With n(p) the negation of p was noted (this is provided just for example).

3. Only the user's answers will be given at the console; the KB expressed as a list of lists is read from a file.

4. Based on KB and the answers provided by the user, the system should say whether or not "patient has pneumonia" can be logically entailed. The output (of both reasoning algorithms) is written on the console.

5. The reasoning mechanisms are the backward chaining and forward chaining algorithms. Both procedures will be implemented in the versions presented at the course (from Ronald Brachman, Hector Levesque. Knowledge representation and reasoning, Morgan Kaufmann 2004).

6. The program should run (that means asking for user's answers and providing the output) repeatedly until "stop" is read.

Second approach:

Write the rules and the answers for the questions directly in Prolog, thus using the backward chaining mechanism already implemented in Prolog to solve the goal.

The predicate **repeat** is defined as following:

```
repeat.
repeat:-repeat.
```

and we can use it to simulate a loop.

```
A:-repeat, B, !.
```

If B is true then it terminates; otherwise repeats B.

Exercise: read different values from the console and write them on the screen until ‘stop’ is provided as input.