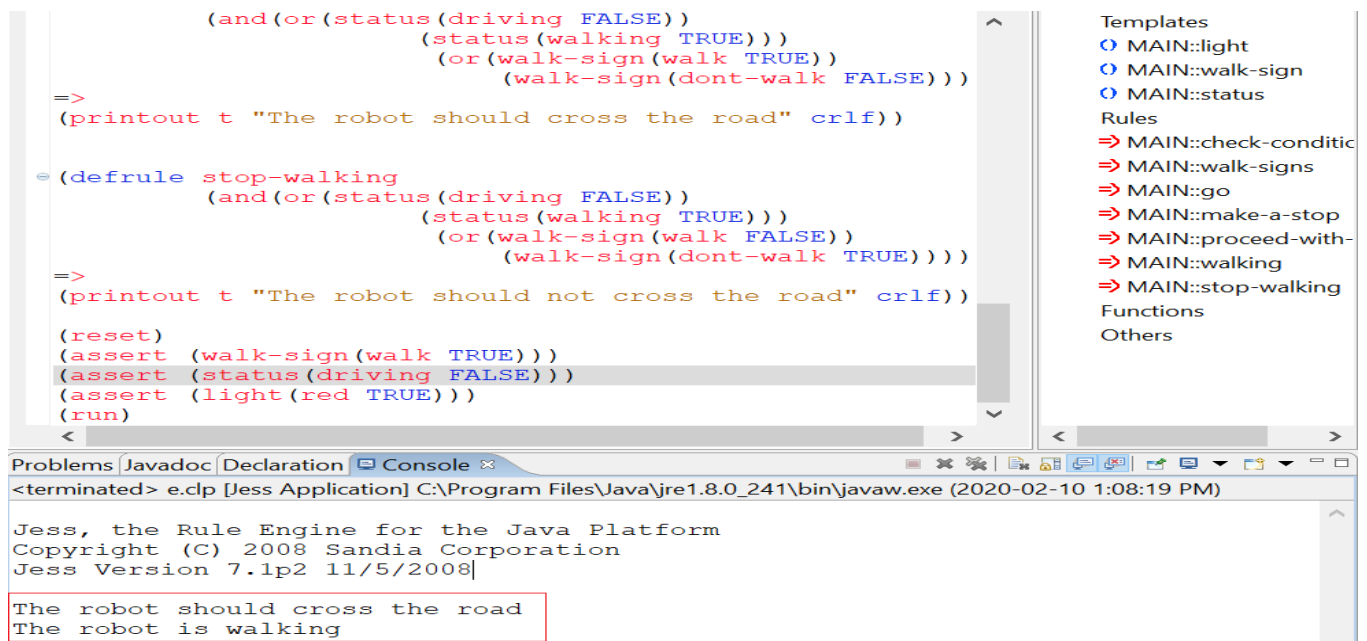


## Question 1:

The program for alerting a robot what to do when it encounters a traffic using JESS has been written and made available in appendix A1 stored as a *Question\_1 .clp* file. For this question, three templates were defined, namely, *light*, *walk-sign*, and *status* with various slots for the light which is indicated with the usual traffic lights. The status of the robot which is assumed to either be driving or walking (based on the way the code was written). Several rules were defined based on the color of the traffic light and the walk-signs (walk or don't-walk sign).

The code was written such that once the driving is *FALSE*, the status of the traffic light gets ignored whether it is *TRUE* or *FALSE*. The facts can be asserted individually by checking just the rules defined for the lights and/or for the walk sign alone.

Let's us consider a situation where the walk sign reads *walk* and the driving status is *FALSE*, the status of the light gets ignored. Whether it is red, green or whatever, the system just focuses on the walk-sign and ignore the light since we have assumed the robot is not driving. In this situation, we expect the output should tell us that the robot is walking and that the robot can cross the road. Figure 1i shows the output generated for this kind of scenario.



The screenshot displays a Java IDE with a JESS application. The main editor shows the following code:

```
(and(or(status(driving FALSE))
      (status(walking TRUE)))
  (or(walk-sign(walk TRUE))
      (walk-sign(dont-walk FALSE))))
=>
(printout t "The robot should cross the road" crlf))

(defrule stop-walking
  (and(or(status(driving FALSE))
        (status(walking TRUE)))
    (or(walk-sign(walk FALSE))
        (walk-sign(dont-walk TRUE)))))
=>
(printout t "The robot should not cross the road" crlf))

(reset)
(assert (walk-sign(walk TRUE)))
(assert (status(driving FALSE)))
(assert (light(red TRUE)))
(run)
```

On the right, a sidebar lists templates and rules. The 'Rules' section shows several rules, including 'MAIN::check-conditions', 'MAIN::walk-signs', 'MAIN::go', 'MAIN::make-a-stop', 'MAIN::proceed-with-caution', 'MAIN::walking', and 'MAIN::stop-walking'. The 'Functions' section lists 'Others'.

The bottom console window shows the output of the JESS application:

```
<terminated> e.clp [Jess Application] C:\Program Files\Java\jre1.8.0_241\bin\javaw.exe (2020-02-10 1:08:19 PM)

Jess, the Rule Engine for the Java Platform
Copyright (C) 2008 Sandia Corporation
Jess Version 7.1p2 11/5/2008

The robot should cross the road
The robot is walking
```

Fig 1i: A screenshot of JAVA environment displaying the JESS code and the output

Let us consider another case where the robot is driving and encounters any of the traffic light, say, blinking yellow, the robot should follow the pre-defined rule given to it, which in this case is, "The robot should proceed through the intersection with caution." If red, the rule defined for such is expected to alert the robot to stop driving. A screenshot of this output is shown below in figure 1ii.

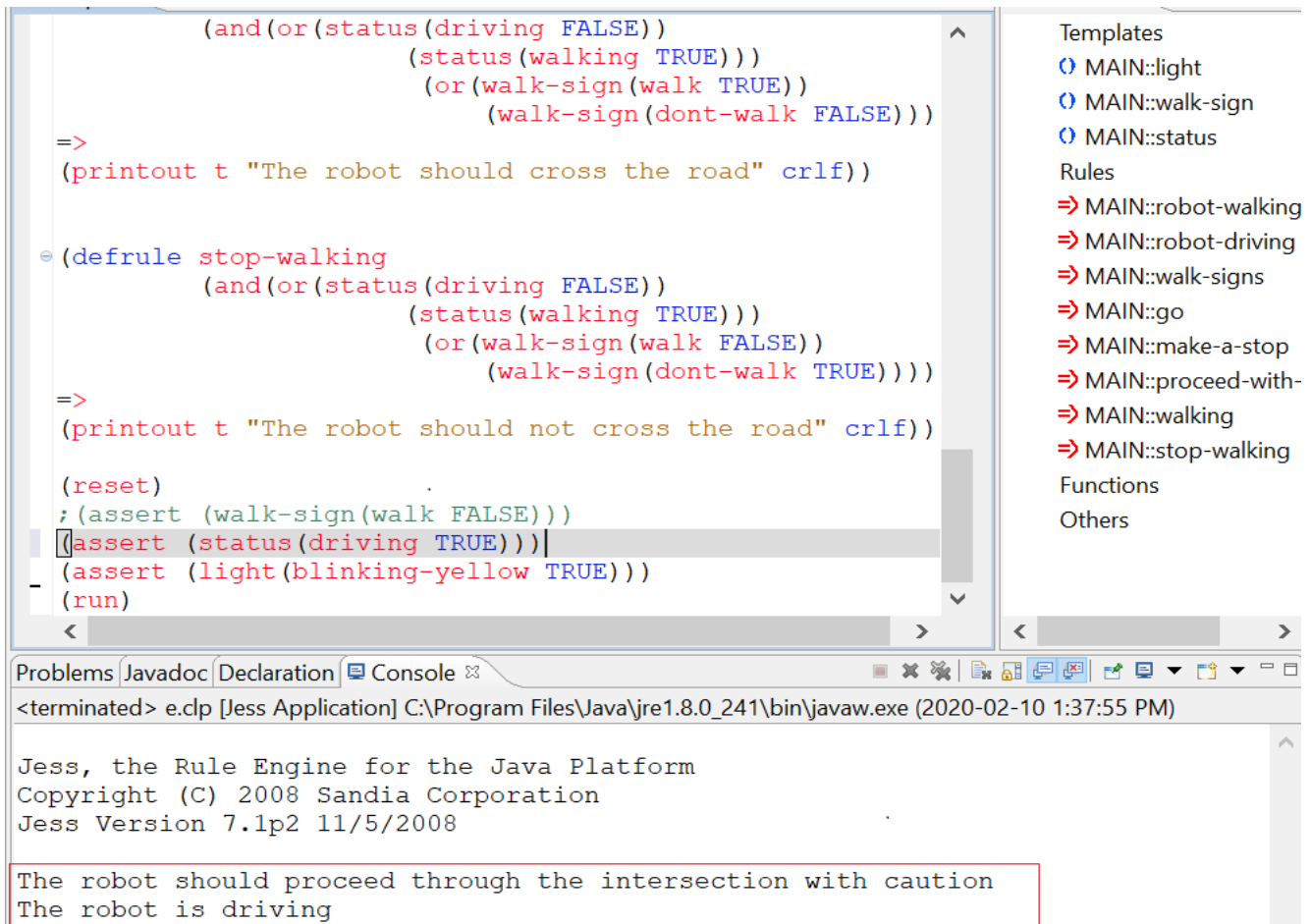


Fig 1ii: A screenshot of JAVA environment displaying the JESS code and the output

## Question 2:

The program for classifying the bar stocks according to their dimensioning using JESS has been written and made available in appendix A2 stored as a *Question\_2.clp* file. The program displays the stock ID and all stock dimensions that match the following definition

- length = width (square cross section)
- length= height (square side) and
- length= width = height (cubes)

Firstly, the *deftemplate* was constructed to create a template and each of the fact has a template (a fact gets its *name* and its list of *slots* from its template). The template has a name and a set of *slots*, and each fact gets these things from its template. For this question, a *stock\_ID* template was defined with slots *material*, *height*, *length* and *width*.

Typing separate assert commands for each of many facts is rather tedious. To make life easier in this regard, the *deffacts* construct was used. A *deffacts* construct is simply a named list of facts. The facts in all defined *deffacts* are asserted into the working memory whenever a reset command is issued. Some facts the *height*, *length* and *width* for each material for the query to work with and load them into working memory was defined.

Finally, the rules were defined using the *defrule* syntax. A variable was declared to refer to the contents of a slot. For example, look at the following pattern:

```
(stock_ID (material ?element) (length ?l) (width ?w) (height ?h))
```

This pattern will match any material fact. When the rule fires, Jess will assign the contents of the "length" slot to a variable "?l", the "width" slot to a variable "?w", and the "height" slot to "?h". The first rule matches any material whose length is the same as its width. The second rule matches any material whose length is the same as its height. The third rule matches any material whose length is the same as its width and height. The variable "?p" after *defrule cube* is a *pattern binding*; it's bound to the whole fact that matches this pattern.

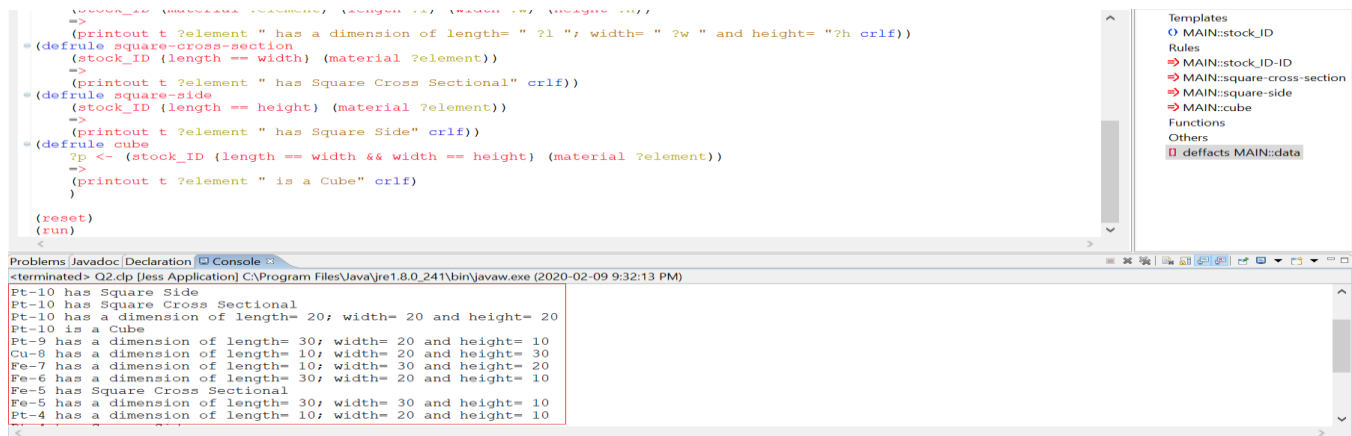


Fig 2. A screenshot of JAVA environment displaying the JESS code and the output

Although, the desired output was generated; I was able to print out the stock ID and all the stock dimensions that match the aforementioned dimensions, but the ordering of the output result was a challenge. I was unable to output all the stock IDs first before the stock dimensions was classified, the returned results were mixed together.

### Question 3:

The program for diagnosing a car by checking various items using JESS has been written and made available in appendix A3 stored as a *Question\_3.clp* file. The program displays all the possible reasons why a car will fail to run, with or without known facts.

For this question, two templates were defined, namely, *car* and *possible\_problem* with various slots for the *car*, which is the parts of the car, and the *possible\_problem* which are likely to occur with the car. Several rules were also defined for the engine, gas, oil, starter motor, battery, tyres, light, ignition and key.

Let's assume a possible scenario in which the car *fails* to run. Also, assume a case where you have *no* initial state for the gas and oil. Now, the system gets to ask you if the car is running or not. In this case, obviously, the answer is *no*. Then the system prompts you to know if the engine turns over when the ignition key is turned. Here, we assume the engine *does not* turn on. Since the operation of the starter motor is dependent on the battery status, and whether the engine turns over upon turning the ignition key or not, the system tells that the starter motor is not working. Here, we have declared lights to be *FALSE*, which means that the battery status is *bad*. If at *least one* of them fails (the battery and/or the engine), the starter motor will fail, and you would get to see this as the output. The system further asks, "does the fuel gauge of the gas read empty?". If the fuel gauge does not read empty, the system assumes that the gas is good. We have established the fact that the car is *not running*, that the engine and starter motor are *not working*, and that the fuel gauge for the gas is *not empty*. Afterwards, the system asks for the level of the oil dipstick. If empty, the system tells you to fill the oil. To be sure that the car is in a good condition, the system ask for the pressure level of the tyres in lbs. If the pressure level is not within the range of 28-32lbs, the system tells you to properly inflate the tyres within that range.



```
* (defrule tyres-yes[]
* (defrule tyre-no[]

* (defrule end-program[]

(reset)
(printout t "Is the car running? yes/no" crlf)
(assert (car (running (read))))
(assert (car (lights FALSE)))
; (assert (car (gas TRUE)))
(assert (car (key TRUE)))
; (assert (car (oil FALSE)))

(run)
```

Problem: Javadoc | Declaration | Console

<terminated> w.clp [JESS Application] C:\Program Files\Java\jre1.8.0\_241\bin\javaw.exe (2020-02-10 12:03:07 AM)

Is the car running? yes/no

no

Car is not Running.

Does the Engine Turn On? yes/no

no

Engine not working

Starter Motor is not Working

Does the fuel gauge of the gas read empty? yes/no

no

Oil dipstick indication? empty/full

empty

The oil needs to be filled

Charge the Battery

What is the tyre Pressure? lbs

24

Inflate Tyres properly between 28 and 32 lbs

Templates

- MAIN:car
- MAIN:problem

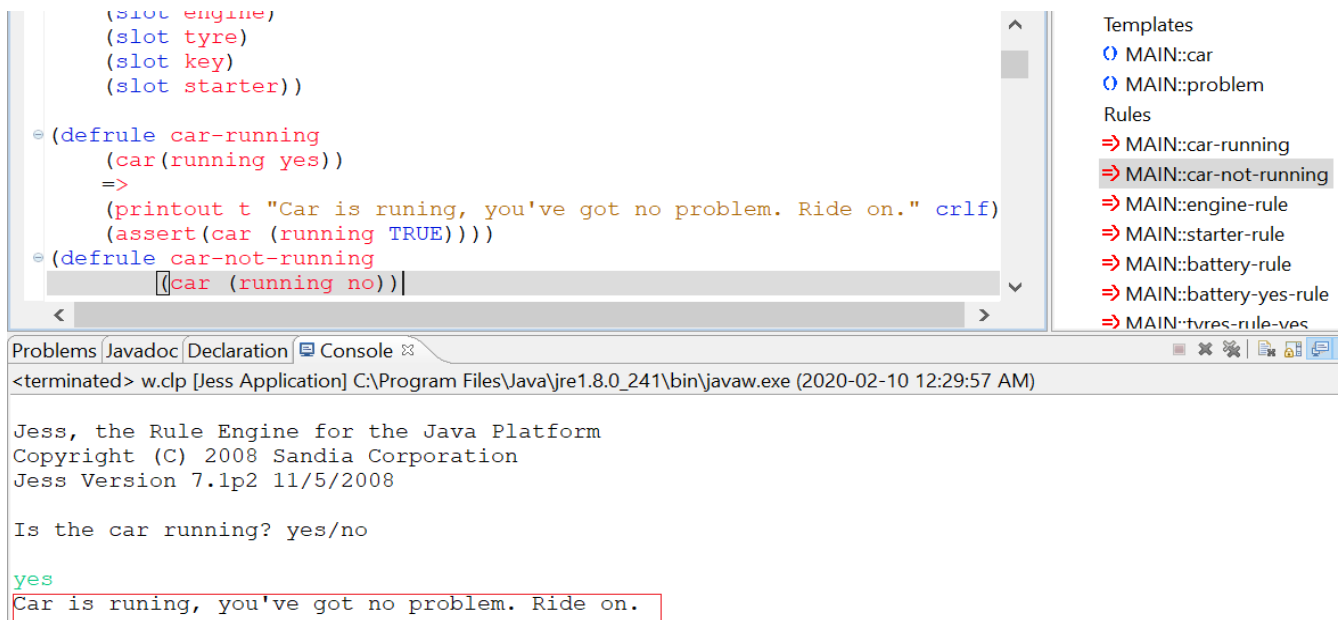
Rules

- MAIN:car-running
- MAIN:car-not-running
- MAIN:engine-rule
- MAIN:starter-rule
- MAIN:battery-rule
- MAIN:battery-yes-rule
- MAIN:tyres-rule-yes
- MAIN:tyres-rule-no

Fig 3i: A screenshot of JESS displaying an output when the car fails to run with a possible problem scenario

The basic idea of this program is to diagnose multiple problems without being terminated when one solution has been detected, as there may be more than one issue with the car. And the troubleshooting is done without a predefined order. A problem detected may unravel another problem that might be wrong with the car. In a situation where we initially fail to declare the state of all the parameters, we are expected to be asked more questions than when we declare a few facts. The system might even ask if you have a key or not. A *no* response would suggest you get your key and turn on the engine.

In a situation where your car is running, irrespective of the facts declared, the system assumes the car has no problem and suggest you should ride on.



The screenshot shows the JESS IDE interface. The main editor displays the following code:

```
(slot engine)
(slot tyre)
(slot key)
(slot starter))

= (defrule car-running
  (car (running yes))
  =>
  (printout t "Car is runing, you've got no problem. Ride on." crlf)
  (assert(car (running TRUE))))
= (defrule car-not-running
  [car (running no)]
```

On the right side, there is a 'Rules' panel listing several rules with arrows indicating their status. The rule 'MAIN::car-not-running' is highlighted.

The bottom console window shows the following output:

```
<terminated> w.clp [Jess Application] C:\Program Files\Java\jre1.8.0_241\bin\javaw.exe (2020-02-10 12:29:57 AM)

Jess, the Rule Engine for the Java Platform
Copyright (C) 2008 Sandia Corporation
Jess Version 7.1p2 11/5/2008

Is the car running? yes/no
yes
Car is runing, you've got no problem. Ride on.
```

Fig 3ii: A screenshot of JESS displaying an output when the car is running

# APPENDIX

## A1. Question\_1.clp

```
(deftemplate light
  (slot red)
  (slot green)
  (slot blinking-red)
  (slot blinking-yellow)
  (slot blinking-green)
  (slot none))

(deftemplate walk-sign
  (slot walk)
  (slot dont-walk ))

(deftemplate status
  (slot walking)
  (slot driving))

(defrule robot-walking
  (or(status (driving FALSE))
   (status(walking TRUE)))
=>
(printout t "The robot is walking" crlf))

(defrule robot-driving
  (or(status (driving TRUE))
   (status(walking FALSE)))
=>
(printout t "The robot is driving" crlf))

(defrule walk-signs
  (and(status(walking TRUE))
   (or(walk-sign(walk TRUE))
    (walk-sign(dont-walk FALSE))))
=>
(printout t "The robot can walk across the other side of the road" crlf))

(defrule go
  (and(or(status(driving TRUE))
   (status(walking FALSE)))
   (or(light(green TRUE))
    (light(blinking-green TRUE))))
=>
(printout t "The robot can drive across traffic signal" crlf))

(defrule make-a-stop
  (and(or(status(driving TRUE))
   (status(walking FALSE)))
   (or(light(red TRUE))
    (light(blinking-red TRUE))))
=>
```

```

(printout t "The robot should stop" crlf))

(defrule proceed-with-caution
  (and(or(status(driving TRUE))
        (status(walking FALSE))
        (or(light(blinking-yellow TRUE))
            (light(none TRUE))))))
=>
(printout t "The robot should proceed through the intersection with caution" crlf))

(defrule walking
  (and(or(status(driving FALSE))
        (status(walking TRUE))
        (or(walk-sign(walk TRUE))
            (walk-sign(dont-walk FALSE))))))
=>
(printout t "The robot should cross the road" crlf))

(defrule stoop-walking
  (and(or(status(driving FALSE))
        (status(walking TRUE))
        (or(walk-sign(walk FALSE))
            (walk-sign(dont-walk TRUE))))))
=>
(printout t "The robot should not cross the road" crlf))

(reset)
(assert (walk-sign(walk TRUE)))
(assert (status(driving FALSE)))
(assert (light(blinking-yellow FALSE)))
(run)

```



## A2. Question\_2.clp

```
(deftemplate stock_ID
  (slot material)
  (slot height)
  (slot length)
  (slot width))

(deffacts data
  (stock_ID (material Fe-1) (length 10) (width 10) (height 10))
  (stock_ID (material Cu-2) (length 20) (width 10) (height 10))
  (stock_ID (material Cu-3) (length 20) (width 10) (height 20))
  (stock_ID (material Pt-4) (length 10) (width 20) (height 10))
  (stock_ID (material Fe-5) (length 30) (width 30) (height 10))
  (stock_ID (material Fe-6) (length 30) (width 20) (height 10))
  (stock_ID (material Fe-7) (length 10) (width 30) (height 20))
  (stock_ID (material Cu-8) (length 10) (width 20) (height 30))
  (stock_ID (material Pt-9) (length 30) (width 20) (height 10))
  (stock_ID (material Pt-10) (length 20) (width 20) (height 20)))

(defrule stock_ID
  (stock_ID (material ?element) (length ?l) (width ?w) (height ?h))
  =>
  (printout t ?element " has a dimension of length= " ?l "; width= " ?w " and height= "?h
    crlf))

(defrule square-cross-section
  (stock_ID {length == width} (material ?element))
  =>
  (printout t ?element " has Square Cross Sectional" crlf))

(defrule square-side
  (stock_ID {length == height} (material ?element))
  =>
  (printout t ?element " has Square Side" crlf))

(defrule cube
  ?p <- (stock_ID {length == width && width == height} (material ?element))
  =>
  (printout t ?element " is a Cube" crlf)
  )

(reset)
(run)
```

### A3. Question\_3.clp

```
(deftemplate car
  (slot gas)
  (slot lights)
  (slot starter)
  (slot oil)
  (slot tyre)
  (slot running)
  (slot engine)
  (slot key)
  (slot battery))

(deftemplate possible_problem
  (slot oil)
  (slot gas)
  (slot lights)
  (slot engine)
  (slot tyre)
  (slot key)
  (slot starter))

(defrule car-running
  (car (running yes))
  =>
  (printout t "Car is running, you've got no possible_problem. Ride on." crlf)
  (assert (car (running TRUE))))

(defrule car-not-running
  (car (running no))
  =>
  (printout t "Car is not Running." crlf)
  (assert (car (running FALSE))))

(defrule engine-rule
  (car (key FALSE))
  =>
  (assert (car (engine FALSE))))

(defrule starter-rule
  (car (engine FALSE))
  (car (battery FALSE))
  =>
  (assert (car (starter FALSE))))

(defrule battery-rule
  (car (lights FALSE))
  =>
  (assert (car (battery FALSE))))

(defrule battery-yes-rule
  (car (lights TRUE))
  =>
```

```

(assert (car (battery TRUE))))

(defrule tyres-rule-yes
  (or(possible_problem (tyre 28))
    (or(possible_problem (tyre 29))
      (or (possible_problem (tyre 30))
        (or (possible_problem (tyre 31))
          (possible_problem (tyre 32))))))
=>
  (assert (car (tyre TRUE))))

(defrule tyres-rule-no
  (or(not(possible_problem (tyre 28)))
    (or(not(possible_problem (tyre 29)))
      (or (not(possible_problem (tyre 30)))
        (or (not(possible_problem (tyre 31)))
          (not(possible_problem (tyre 32)))))))
=>
  (assert (car (tyre FALSE))))

;FOR GAS
(defrule gas
  (car (running FALSE))
  (car (gas FALSE))
=>
  (printout t "Fill up the Gas" crlf))
(defrule ask-gas
  (car (running FALSE))
  (and(not (car (gas TRUE)))
    (not (car (gas FALSE))))
=>
  (printout t "Does the fuel gauge of the gas read empty? yes/no" crlf)
  (assert (possible_problem (gas (read))))))
(defrule gas-empty
  (possible_problem (gas empty))
=>
  (assert (car (gas FALSE))))
(defrule gas-full
  (or(car (gas TRUE))
    (possible_problem (gas full)))
=>
  (printout t "Gas is Full" crlf)
  )

;FOR OIL
(defrule oil
  (car (running FALSE))
  (car (oil FALSE))
=>
  (printout t "The oil needs to be filled" crlf))
(defrule ask-oil
  (car (running FALSE))
  (and(not (car (oil TRUE)))
    (not (car (oil FALSE))))
=>

```

```

    (printout t "Oil dipstick indication? empty/full" crlf)
    (assert (possible_problem (oil (read)))))
(defrule oil-empty
  (possible_problem (oil empty))
=>
  (assert (car (oil FALSE))))
(defrule oil-full
  (or(car (oil TRUE))
    (possible_problem (oil full)))
=>
  (printout t "Oil is Full" crlf)
)

```

;FOR BATTERY

```

(defrule lights
  (car (running FALSE))
  (car (lights FALSE))
=>
  (printout t "Charge the Battery" crlf))
(defrule ask-lights
  (car (running FALSE))
  (and(not (car (lights TRUE)))
    (not (car (lights FALSE)))))
=>
  (printout t "Are the lights turning on? yes/no" crlf)
  (assert (possible_problem (lights (read)))))
(defrule lights-no
  (possible_problem (lights no))
=>
  (assert (car (lights FALSE))))
(defrule lights-yes
  (or(car (lights TRUE))
    (possible_problem (lights yes)))
=>
  (printout t "Battery is charged" crlf)
)

```

;FOR IGNITION

```

(defrule ignition
  (car (running FALSE))
  (car (engine FALSE))
=>
  (printout t "Engine not working" crlf)
)
(defrule ask-ignition
  (car (running FALSE))
  (and (not (car (engine TRUE)))
    (not (car (engine FALSE)))))
=>
  (printout t "Does the Engine Turn On? yes/no" crlf)
  (assert (possible_problem (engine (read)))))
(defrule engine-no
  (possible_problem (engine no))
=>
  (assert (car (engine FALSE))))

```

```

(defrule engine-yes
  (possible_problem (engine yes))
=>
  (assert (car (engine TRUE))))

;FOR KEY
(defrule key
  (car (running FALSE))
  (car (key FALSE))
=>
  (printout t "Get your key and turn on the engine" crlf))
(defrule ask-key
  (car (running FALSE))
  (and (not (car (key TRUE)))
        (not (car (key FALSE)))))
=>
  (printout t "Do you have the key? yes/no" crlf)
  (assert (possible_problem (key (read))))))
(defrule key-no
  (possible_problem (key no))
=>
  (assert (car (key FALSE))))
(defrule key-yes
  (possible_problem (key yes))
=>
  (assert (car (key TRUE))))

;FOR STARTER
(defrule starter-no
  (car (running FALSE))
  (car (starter FALSE))
=>
  (printout t "Starter Motor is not Working" crlf)
  )

;FOR TYRES
(defrule tyres-ask
  (car (running FALSE))
=>
  (printout t "What is the tyre Pressure? lbs" crlf)
  (assert (possible_problem (tyre (read))))))
(defrule tyres-yes
  (car (tyre TRUE))
=>
  (printout t "Tyres are inflated properly." crlf))
(defrule tyre-no
  (car (running FALSE))
  (not (car (tyre TRUE)))
  (car (tyre FALSE))
=>
  (printout t "Inflate Tyres properly between 28 and 32 lbs"))

(defrule end-program

```

```

(or(possible_problem (engine end))
  (or(possible_problem (lights end))
    (or(possible_problem (gas end))
      (or(possible_problem (oil end))
        (or(possible_problem (tyre end))
          (or(possible_problem (key end))
            (car (running end))))))))))
=>
(printout t "end program")
(exit)
)

(reset)
(printout t "Is the car running? yes/no" crlf)
(assert (car (running (read))))
(assert (car (lights FALSE)))
;(assert (car (gas TRUE)))
(assert (car (key TRUE)))
;(assert (car (oil FALSE)))

(run)

```