SUBJECT: EXPERT SYSTEM DESIGN-LAB

LIST OF EXPERIMENTS

Sr. No.	Name of Experiment
1	Introduction to expert systems.
2	To implement 8 puzzle A* algorithm.
3	Rule based system for problem solving
4	Write a program to implement Planning concept
5	Design a program / problem based on object-oriented analysis and for expert system (Representing typical objects and situations).
6	Design a program / problem based on knowledge Acquisition with respect to expert system shell.
7	Design a program based on Heuristic Classification.
8	Design a program demonstrating non-monotonic justification

[illegible]

Subject Teacher

Prof. M. S. Shegokar

Department of CSE

PRACTICAL NO. 1

Aim:- Introduction to expert systems.

- What is Expert System?

An expert system is a set of programs that manipulate encoded knowledge to solve problems in a specialized domain that normally requires human expertise. An expert systems knowledge is obtained from expert sources and coded in form suitable for the system to use in its inference or reasoning processes. The expert knowledge must be obtained from specialist or other sources of expertise, such as text, journal articles and data bases. This type of knowledge usually requires much training an experience in some specialized field such as medicine, geology, system configuration, or engineering design. Once a sufficient body of expert knowledge has been required, it must be encoded in some form , loaded in to a knowledge base, then tested and refined continually throughout the life of the system.

- Need of Expert System

- Expert systems have been proven to be effective in a number of problem domains, which normally require the kind of intelligence possessed by a human expert. The areas of application are almost endless.
- Wherever human expertise is needed to solve a problem, expert systems are most likely of the options sought. Application domain includes law, chemistry, biology engineering, manufacturing, aerospace military operations, finance, banking, meteorology, geology, geophysics and medicine.
- An expert system is a set of programs that manipulate encoded knowledge to solve problems in a specialized domain that normally requires human expertise
- The expert knowledge must be obtained from specialists or other sources of expertise, such as texts, journal articles, and databases.

- Advantages and Applications

  - Advantages of Expert System

    1. Training people to become an expert is expensive and very time consuming. Expert systems, though can be copied many times.
    2. They are always available unlike us that need holidays, get sick, change job etc.
    3. Contain the accumulated knowledge from many experts on a particular subject.

  - Applications of Expert System

Since the introduction of the early expert systems , the range and depth of applications has broadened dramatically. Application can now be found in almost all areas of business and government they include such area as

- Different types of medical diagnoses (internal medicine, pulmonary diseases, infectious blood diseases and so on)
- Diagnosis of complex electronic and electro mechanical systems

- Diagnosis of diesel electric locomotion systems
- Diagnosis of software development projects
- Planning experiments in biology, Chemistry and molecular genetics
- Forecasting crop damage
- Identification of chemical compound structure and chemical compounds
- Location of Faults in computer and communication systems
- Scheduling of customer order, job shop production operations, computer resources for operating systems and various manufacturing task.
- Evaluation of known applicants for lending institutions
- Assessment of geologic structure from dip meter logs
- Analysis of structural systems for design or as a result of earth quake damage
- The optimal configuration of components to meet given specifications for a complex system
- Estate planning for minimal taxation and other specific goals.

➢ Introduction to CLIPS

CLIPS is a public domain software tool for building expert systems. The name is an acronym for "C Language Integrated Production System." The syntax and name was inspired by Charles Forgy's OPS ("Official Production System," although there was nothing really official about it). The first versions of CLIPS were developed starting in 1985 at NASA-Johnson Space Center (as an alternative for existing system ART*Inference) until the mid 1990s when the development group's responsibilities ceased to focus on expert system technology. The original name of the project was *NASA's AI Language* (*NAIL*). CLIPS is probably the most widely used expert system tools because it is fast, efficient and free. CLIPS incorporates a complete object-oriented language *COOL* for writing expert systems. CLIPS is written in C, extensions can be written in C, and CLIPS can be called from C. Its user interface closely resembles that of the programming language LISP.

➢ CLIPS programming with syntax and semantics

Notation

```
(example)                   ;symbol
(example [1])               ;optional
(example <INTEGER>)         ;replacement
<INTEGER>*                  ;zero or more
<INTEGER>+                  ;one or more
all | none | some           ;or
```

Fields

float

```
1.5
9.3e+7
```

integer

```
5
-37
```

Entering and Exiting Clips

```
C>CLIPS
```

```
(exit)
```

Facts

```
( <relation-name> <field>+ )

(person (name "John S. Liu")
        (age 23)
        (eye-color brown)
        (hair-color black))
```

Ordered Facts

facts without a deftemplate

```
(person John S. Liu      ;name John
        23               ;age
        brown            ;eye-color
        black)           ;hair-color
```

Adding and Removing Facts

```
(assert <fact>+)

(assert (person (name John S. Liu)
                (age 23)
                (eye-color brown)
                (hair-color black)))

(retract <fact-index>+)

(facts)
(retract 0)
```

Modifying and Duplicating Facts

```
(modify <fact-index> <slot-modifier>+)

<slot-modifier>
        (<slot-name> <slot-value>)

(modify 0 (age 24) )
(facts)

(duplicate 2 (name "Jack P. Chen") )
(facts)
```

The Watch Command

```
(watch <watch-item>)

<watch-item> : facts, rules, activations, statistics,
compilations, focus, all

(watch facts)
(modify 3 (age 25) )
(facts)
```

The Components of A Rule

```
    (defrule <rule-name> [<optional comment>]
        <patterns>*            ;LHS
        =>
        <actions>*             ;RHS
        )

    (defrule fire-emergency "An example rule"       ;Rule
header
        (emergency (type fire))                     ;Patterns
        =>
        (assert (response                           ;Actions
                    (action activate-sprinkler-system))
```

➢ Features of CLIPS

- ***Knowledge Representation***: CLIPS provides a cohesive tool for handling a wide variety of knowledge with support for three different programming paradigms: rule-based, object-oriented and procedural. Rule-based programming allows knowledge to be represented as heuristics, or "rules of thumb," which specify a set of actions to be performed for a given situation. Object-oriented programming allows complex systems to be modeled as modular components (which can be easily reused to model other systems or to create new components). The procedural programming capabilities provided by CLIPS are similar to capabilities found in languages such as C, Java, Ada, and LISP.
- ***Portability*****:** CLIPS is written in C for portability and speed and has been installed on many different operating systems without code changes. Operating systems on which CLIPS has been tested include Windows XP, MacOS X, and Unix. CLIPS can be ported to any system which has an ANSI compliant C or C++ compiler. CLIPS comes with all source code which can be modified or tailored to meet a user's specific needs.
- ***Integration/Extensibility***: CLIPS can be embedded within procedural code, called as a subroutine, and integrated with languages such as C, Java, FORTRAN and ADA. CLIPS can be easily extended by a user through the use of several well-defined protocols.
- ***Interactive Development***: The standard version of CLIPS provides an interactive, text oriented development environment, including debugging aids, on-line help, and an integrated editor. Interfaces providing features such as pulldown menus, integrated editors, and multiple windows have been developed for the MacOS, Windows XP, and X Window environments.
- ***Verification/Validation***: CLIPS includes a number of features to support the verification and validation of expert systems including support for modular design and partitioning of a knowledge base, static and dynamic constraint checking of slot values and function arguments, and semantic analysis of rule patterns to determine if inconsistencies could prevent a rule from firing or generate an error.
- ***Fully Documented***: CLIPS comes with extensive documentation including a Reference Manual and a User's Guide.
- ***Low Cost***: CLIPS is maintained as public domain software.

Conclusion: Thus we have studied the expert system and CLIPS syntax,how to define rules and facts**.**

PRACTICAL NO. 2

AIM : To implement 8 puzzle A* algorithm.

Theory:

- What is A* algorithm?

A* (pronounced "A star") is a computer algorithm that is widely used in pathfinding and graph traversal, the process of plotting an efficiently traversable path between points, called nodes. Noted for its performance and accuracy, it enjoys widespread use. Peter Hart, Nils Nilsson and Bertram Raphael of Stanford Research Institute (now SRI International) first described the algorithm in 1968.[1] It is an extension ofEdsger Dijkstra's 1959 algorithm. A* achieves better performance (with respect to time) by using heuristics.

- A* algorithm:

```
function A*(start,goal)
     closedset := the empty set     // The set of nodes already
evaluated.
     openset := {start}     // The set of tentative nodes to be
evaluated, initially containing the start node
     came_from := the empty map        // The map of navigated
nodes.

     g_score[start] := 0        // Cost from start along best
known path.
     // Estimated total cost from start to goal through y.
     f_score[start]          :=          g_score[start]          +
heuristic_cost_estimate(start, goal)

     while openset is not empty
         current := the node in openset having the lowest
f_score[] value
         if current = goal
             return reconstruct_path(came_from, goal)

         remove current from openset
         add current to closedset
         for each neighbor in neighbor_nodes(current)
             if neighbor in closedset
                 continue

tentative_g_score:=g_score[current]+dist_between(current,neigh
bor)

             if neighbor not in openset or tentative_g_score
<= g_score[neighbor]
```

```
                 came_from[neighbor] := current
                 g_score[neighbor] := tentative_g_score

f_score[neighbor]:=g_score[neighbor]+heuristic_cost_estimate(n
eighbor, goal)
                 if neighbor not in openset
                     add neighbor to openset

     return failure

 function reconstruct_path(came_from, current_node)
     if came_from[current_node] in set
         p            :=            reconstruct_path(came_from,
came_from[current_node])
         return (p + current_node)
     else
         return current node
```

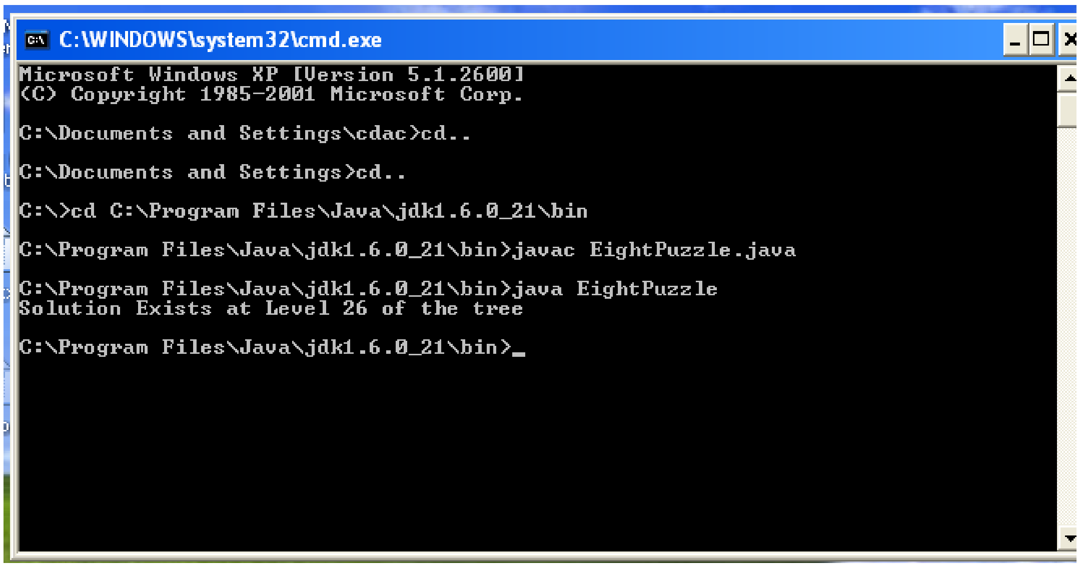
- Implementation of A* Alogorithm:

```
import java.util.*;

class EightPuzzle

 {

      Queue<String> q = new LinkedList<String>();   // Use of Queue Implemented using
      LinkedList for Storing All the Nodes in BFS.

      Map<String,Integer> map = new HashMap<String, Integer>(); // HashMap is used to
ignore repeated nodes

       public static void main(String args[])

      {

           String str="870465132";                          // Input the Board State as
           a String with 0 as the Blank Space

           EightPuzzle e = new EightPuzzle();                 // New Instance of the
      EightPuzzle

            e.add(str,0);                                    // Add the Initial State

           while(e.q.peek()!=null){
```

```
                        e.up(e.q.peek());                              // Move the blank space up
                        and add new state to queue

                         e.down(e.q.peek());                           // Move the blank space
               down

                        e.left(e.q.peek());                            // Move left

                         e.right(e.q.remove());                        // Move right and remove
               the current node from Queue

                }

                        System.out.println("Solution doesn't exist");

           }

           //Add method to add the new string to the Map and Queue

           void add(String str,int n)

       {

               if(!map.containsKey(str))

               {

               map.put(str,n);

               q.add(str);

                }

        }


           /* Each of the Methods below Takes the Current State of Board as String. Then the
       operation to move the blank space is done if possible.  After that the new string is added to
       the map and queue.If it is the Goal State then the Program Terminates. */

           void up(String str)

       {

               int a = str.indexOf("0");

               if(a>2)

               {

                        String s = str.substring(0,a-3)+"0"+str.substring(a-2,a)+str.charAt(a-
       3)+str.substring(a+1);
```

```
                    add(s,map.get(str)+1);
                    if(s.equals("123456780"))
                            {
                            System.out.println("Solution Exists at Level "+map.get(s)+" of the
tree");
                            System.exit(0);
                           }
            }
   }
   void down(String str)
{
            int a = str.indexOf("0");
            if(a<6)
            {
                    String s =
str.substring(0,a)+str.substring(a+3,a+4)+str.substring(a+1,a+3)+"0"+str.substring(a+4);
                    add(s,map.get(str)+1);
                    if(s.equals("123456780"))
                    {
                            System.out.println("Solution Exists at Level "+map.get(s)+" of the
tree");
                            System.exit(0);
                    }
            }
   }
   void left(String str)
{
            int a = str.indexOf("0");
            if(a!=0 && a!=3 && a!=6)
            {
                    String s = str.substring(0,a-1)+"0"+str.charAt(a-1)+str.substring(a+1);
                    add(s,map.get(str)+1);
                    if(s.equals("123456780"))
                     {
                            System.out.println("Solution Exists at Level "+map.get(s)+" of the
    tree");
                            System.exit(0);
                    }   }   }
       void right(String str)
    {
            int a = str.indexOf("0");
            if(a!=2 && a!=5 && a!=8)
            {
                    String s = str.substring(0,a)+str.charAt(a+1)+"0"+str.substring(a+2);
                    add(s,map.get(str)+1);
                    if(s.equals("123456780"))
                    {
                            System.out.println("Solution Exists at Level "+map.get(s)+" of the
    tree");
                            System.exit(0);
                    }  }  }   }
```

Output:



C:\WINDOWS\system32\cmd.exe

```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\cdac>cd..

C:\Documents and Settings>cd..

C:\>cd C:\Program Files\Java\jdk1.6.0_21\bin

C:\Program Files\Java\jdk1.6.0_21\bin>javac EightPuzzle.java

C:\Program Files\Java\jdk1.6.0_21\bin>java EightPuzzle
Solution Exists at Level 26 of the tree

C:\Program Files\Java\jdk1.6.0_21\bin>_
```

Our initial state is:

8 7 0

4 6 5

1 3 2

Our goal state is:

1 2 3

4 5 6

7 8 0

Conclusion: In this program for each node n that we developed , we compute the value of a function in this way the 8 puzzle problem was implemented using A *.

PRACTICAL NO.3

Aim: Rule based system for problem solving.

Theory:

- What is Rule based System

  The rule base or knowledge base :In expert system technology, the knowledge base is expressed with natural language rules IF ... THEN ... For examples :

  - "IF it is living THEN it is mortal"
  - "IF his age = known THEN his year of birth = date of today - his age in years"
  - "IF the identity of the germ is not known with certainty AND the germ is gram-positive AND the morphology of the organism is "rod" AND the germ is aerobic THEN there is a strong probability (0.8) that the germ is of type enterobacteriacae".

- Analyse the Problem.

  A simple expert system which attempts to identify an birds based on its characteristics. The knowledge base in this example is a collection of facts which represent backward chaining rules. CLIPS forward chaining rules are then used to simulate a backward chaining inference engine.

- Formulate as Rule based Model.

```
1.(rule (if phylum is soil and
          flat.bodied is yes)
         (then type.bird is vangas))
   2. (rule (if phylum is soil and
          flat.bodied is no)
         (then type.bird is grebes))
```

  In this rule we define the phylum soil and characteristics as flat bodied,if it is yes then the bird is Vangas else grebes.

- How to define rule in CLIPS.

```
(defrule propagate-goal ""
   (goal is ?goal)
   (rule (if ?variable $?)
         (then ?goal ? ?value))
   =>
 (assert (goal is ?variable)))
```

- Implement Rule based Model in CLIPS

```
;;;***************************
;;;* DEFTEMPLATE DEFINITIONS *
;;;***************************
(deftemplate rule
   (multislot if)
   (multislot then))

;;;**************************
;;;* INFERENCE ENGINE RULES *
;;;**************************
(defrule propagate-goal ""
   (goal is ?goal)
   (rule (if ?variable $?)
         (then ?goal ? ?value))
   =>
   (assert (goal is ?variable)))

(defrule goal-satified ""
   (declare (salience 30))
   ?f <- (goal is ?goal)
   (variable ?goal ?value)
   (answer ? ?text ?goal)
   =>
   (retract ?f)
   (format t "%s%s%n" ?text ?value))
;;;***************************
;;;* DEFFACTS KNOWLEDGE BASE *
;;;***************************

(deffacts knowledge-base
   (goal is type.bird)
   (legalanswers are yes no)
   (rule (if backbone is yes)
         (then superphylum is backbone))
   (rule (if backbone is no)
         (then superphylum is jellyback))
   (question backbone is "Does your bird have a backbone?")
```

```
(rule (if superphylum is backbone and
       warm.blooded is yes)
      (then phylum is warm))
(rule (if superphylum is backbone and
       warm.blooded is no)
      (then phylum is cold))
(question warm.blooded is "Is the bird warm blooded?")
(rule (if superphylum is jellyback and
       live.prime.in.soil is yes)
      (then phylum is soil))
(rule (if superphylum is jellyback and
       live.prime.in.soil is no)
      (then phylum is elsewhere))
(question live.prime.in.soil is "Does your bird live primarily in soil?")
(rule (if phylum is warm and
       has.breasts    is    yes)
      (then class is breasts))
(rule (if phylum is warm and
       has.breasts is no)
      (then type.bird is bird/penguin))
(question has.breasts is "Normally, does the female of your bird nurse its young
with milk?")
(rule (if phylum is cold and
      always.in.water  is  yes)
      (then class is water))
(rule (if phylum is cold and
      always.in.water is no)
      (then class is dry))
(question always.in.water is "Is your bird always in water?")
(rule (if phylum is soil and
       flat.bodied is yes)
      (then type.bird is vangas))
(rule (if phylum is soil and
       flat.bodied is no)
      (then type.bird is grebes))
(question flat.bodied is "Does your bird have a flat body?")
(rule (if phylum is elsewhere and
       body.in.segments is yes)
      (then class is segments))
(rule (if phylum is elsewhere and
      body.in.segments is no)
      (then class is unified))
(question body.in.segments is "Is the bird body in segments?")
(rule (if class is breasts and
       can.eat.meat is yes)
      (then order is meat))
(rule (if class is breasts and
      can.eat.meat is no)
      (then order is vegy))
(question can.eat.meat is "Does your bird  eat red meat?")
(rule (if class is water and
       boney is yes)
      (then type.bird  is Dove))
(rule (if class is water and
       boney is no)
      (then type.bird  is bat))
(question boney is "Does your bird have a boney skeleton?")
(rule (if class is dry and
       scally is yes)
      (then order is scales))
(rule (if class is dry and
       scally is no)
      (then order is soft))
(question scally is "Is your bird covered with scaled skin?")
(rule (if class is segments and
       shell is yes)
      (then order is shell))
(rule (if class is segments and
       shell is no)
      (then type.bird is antbirds/pittas/trogons))
(question shell is "Does your bird have a shell?")
(rule (if class is unified and
       digest.cells is yes)
      (then order is cells))
```

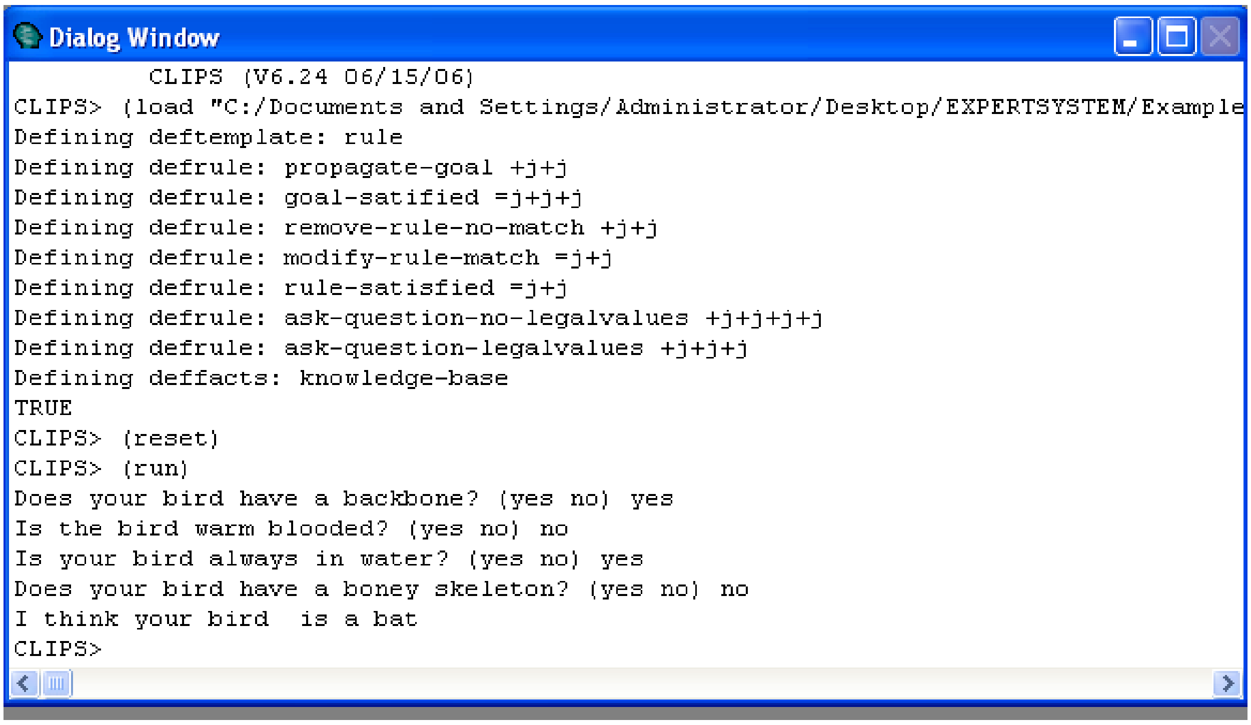
```
  (rule (if class is unified and
       digest.cells is no)
      (then order is stomach))
  (question digest.cells is "Does your bird use many cells to digest it's food instead of
a stomach?")
  (rule (if order is meat and
       fly is yes)
      (then type.bird is ostrich))
  (rule (if order is meat and
       fly is no)
      (then family is nowings))
  (question fly is "Can your bird fly?")
  (rule (if order is vegy and
       hooves is yes)
      (then family is hooves))
  (rule (if order is vegy and
       hooves is no)
      (then family is feet))
  (question hooves is "Does your bird have hooves?")
  (rule (if order is scales and
       rounded.shell is yes)
      (then type.animal is turtle))
  (rule (if order is scales and
       rounded.shell is no)
      (then family is noshell))
  (question rounded.shell is "Does the bird  have a rounded shell?")
  (rule (if order is soft and
       jump is yes)
      (then type.bird  is kiwis))
  (rule (if order is soft and
       jump is no)
      (then type.bird  is cukoo))
  (question jump is "Does your bird  jump?")
  (rule (if order is shell and
       tail is yes)
      (then type.bird  is divers))
  (rule (if order is shell and
       tail is no)
      (then type.bird  is loras))
(question tail is "Does your bird have a tail?")
(rule (if order is cells and
       stationary is yes)
      (then family is stationary))
(rule (if order is cells and
       stationary is no)
      (then type.bird  is owls))
(question stationary is "Is your bird attached permanently to an object?")
(rule (if order is stomach and
       multicelled is yes)
      (then family is multicelled))
(rule (if order is stomach and
       multicelled is no)
      (then type.bird  is sparrow))
(question multicelled is "Is your bird  made up of more than one cell?")
(rule (if family is nowings and
       opposing.thumb is yes)
      (then genus is thumb))
(rule (if family is nowings and
      opposing.thumb is no)
      (then genus is nothumb))
(question opposing.thumb is "Does your bird  have an opposing thumb?")
(rule (if family is hooves and
       two.toes is yes)
      (then genus is twotoes))
(rule (if family is hooves and
       two.toes is no)
      (then genus is onetoe))
(question two.toes is "Does your bird  stand on two toes/hooves per foot?")
(rule (if family is feet and
       live.in.water is yes)
      (then genus is water))
(rule (if family is feet and
      live.in.water is no)
      (then genus is dry))
```

```
  (question live.in.water is "Does your bird  live in water?")
  (rule (if family is noshell and
         limbs is yes)
        (then type.bird  is eagle/boatbills))
  (rule (if family is noshell and
         limbs is no)
        (then type.bird  is sitellas))
  (question limbs is "Does your bird  have limbs?")
  (rule (if family is stationary and
         spikes is yes)
        (then type.bird  is sea.anemone))
  (rule (if family is stationary and
         spikes is no)
        (then type.bird  is whistellers/palmchat))
  (question spikes is "Does your bird  normally have spikes radiating from it's
body?")
  (rule (if family is multicelled and
         spiral.shell is yes)
        (then type.bird  is bushtits))
  (rule (if family is multicelled and
         spiral.shell is no)
        (then genus is noshell))
  (question spiral.shell is "Does your bird  have a spiral-shaped shell?")
  (rule (if genus is thumb and
         prehensile.tail is yes)
        (then type.bird is larks))
  (rule (if genus is thumb and
        prehensile.tail is no)
        (then species is notail))
  (question prehensile.tail is "Does your bird  have a prehensile tail?")
  (rule (if genus is nothumb and
         over.400 is yes)
        (then species is 400))
  (rule (if genus is nothumb and
         over.400 is no)
        (then species is under400))
  (question over.400 is "Does an adult normally weigh over 400 pounds?")
```

```
(rule (if genus is twotoes and
       horns is yes)
      (then species is horns))
(rule (if genus is twotoes and
       horns is no)
      (then species is nohorns))
(question horns is "Does your bird have horns?")
(rule (if genus is onetoe and
       plating is yes)
      (then type.bird  is rhinoceros))
(rule (if genus is onetoe and
       plating is no)
      (then type.bird is thrushes/bulbuls))
(question plating is "Is your bird  covered with a protective plating?")
(rule (if genus is water and
       hunted is yes)
      (then type.bird  is leafbird))
(rule (if genus is water and
       hunted is no)
      (then type.bird is dolphin/dippers))
(question hunted is "Is your bird , unfortunately, commercially hunted?")
(rule (if genus is dry and
       front.teeth is yes)
      (then species is teeth))
(rule (if genus is dry and
       front.teeth is no)
      (then species is noteeth))
(question front.teeth is "Does your bird  have large front teeth?")
(rule (if genus is noshell and
       bivalve is yes)
      (then type.bird  is flowernaker/crippers))
(rule (if genus is noshell and
       bivalve is no)
      (then type.bird is weavers/asenters))
(question bivalve is "Is your bird  protected by two half-shells?")
(rule (if species is notail and
       nearly.hairless is yes)
```

```
    (then type.bird  is warblers))
(rule (if species is notail and
     nearly.hairless is no)
    (then subspecies is hair))
(question nearly.hairless is "Is your bird  nearly hairless?")
(rule (if species is 400 and
     land.based is yes)
    (then type.bird  is bear/tiger/lion))
(rule (if species is 400 and
     land.based is no)
    (then type.bird  is black bird))
(question land.based is "Is your bird land based?")
(rule (if species is under400 and
     thintail is yes)
    (then type.bird  is tanagers))
(rule (if species is under400 and
     thintail is no)
    (then type.bird  is coyote/pipits/whydas/snowtits))
(question thintail is "Does your bird  have a thin tail?")
(rule (if species is horns and
     one.horn is yes)
    (then type.bird  is rock sparrow))
(rule (if species is horns and
     one.horn is no)
    (then subspecies is nohorn))
(question one.horn is "Does your bird  have one horn?")
(rule (if species is nohorns and
     lives.in.desert is yes)
    (then type.bird  is tinamo))
(rule (if species is nohorns and
     lives.in.desert is no)
    (then type.bird  is giraffe))
(question lives.in.desert is "Does your bird  normally live in the desert?")
(rule (if species is teeth and
     large.ears is yes)
    (then type.bird  is peacock))
(rule (if species is teeth and
               large.ears is no the type.bird  is
ovenbird/mousebird/lyrebird/beaver/porcupine))
   (question large.ears is "Does your bird have large ears?")
   (rule (if species is noteeth and
          pouch is yes)
         (then type.bird  is "squirrel/koala bird"))
   (question domesticated is "Is your bird  domesticated?")
   (answer is "I think your bird  is a " type.bird ))
```

Output**:**



```
         CLIPS (V6.24 06/15/06)
CLIPS> (load "C:/Documents and Settings/Administrator/Desktop/EXPERTSYSTEM/Example
Defining deftemplate: rule
Defining defrule: propagate-goal +j+j
Defining defrule: goal-satified =j+j+j
Defining defrule: remove-rule-no-match +j+j
Defining defrule: modify-rule-match =j+j
Defining defrule: rule-satisfied =j+j
Defining defrule: ask-question-no-legalvalues +j+j+j+j
Defining defrule: ask-question-legalvalues +j+j+j
Defining deffacts: knowledge-base
TRUE
CLIPS> (reset)
CLIPS> (run)
Does your bird have a backbone? (yes no) yes
Is the bird warm blooded? (yes no) no
Is your bird always in water? (yes no) yes
Does your bird have a boney skeleton? (yes no) no
I think your bird  is a bat
CLIPS>
```

Conclusion: The above program in CLIPS defines rules for the bird. We apply properties as per defined in the rules and identifies the particular bird according to its defined property.

PRACTICAL NO.4

Aim: Write a program to implement Planning concept

Theory:

- What is Planning concept?

  Intelligent agents must be able to set goals and achieve them. They need a way to visualize the future (they must have a representation of the state of the world and be able to make predictions about how their actions will change it) and be able to make choices that maximize the utility (or "value") of the available choices.In classical planning problems, the agent can assume that it is the only thing acting on the world and it can be certain what the consequences of its actions may be. However, if the agent is not the only actor, it must periodically ascertain whether the world matches its predictions and it must change its plan as this becomes necessary, requiring the agent to reason under uncertainty.

- How it is considered as Expert system problem or AI problem?

  The AI based definition of planning can be described more succinctly as the systematic vis-à-vis experimental compilation,organization use of domain knowledge for the purpose of automatic solution finding to a given problem via the computer.

- Example of Expert System

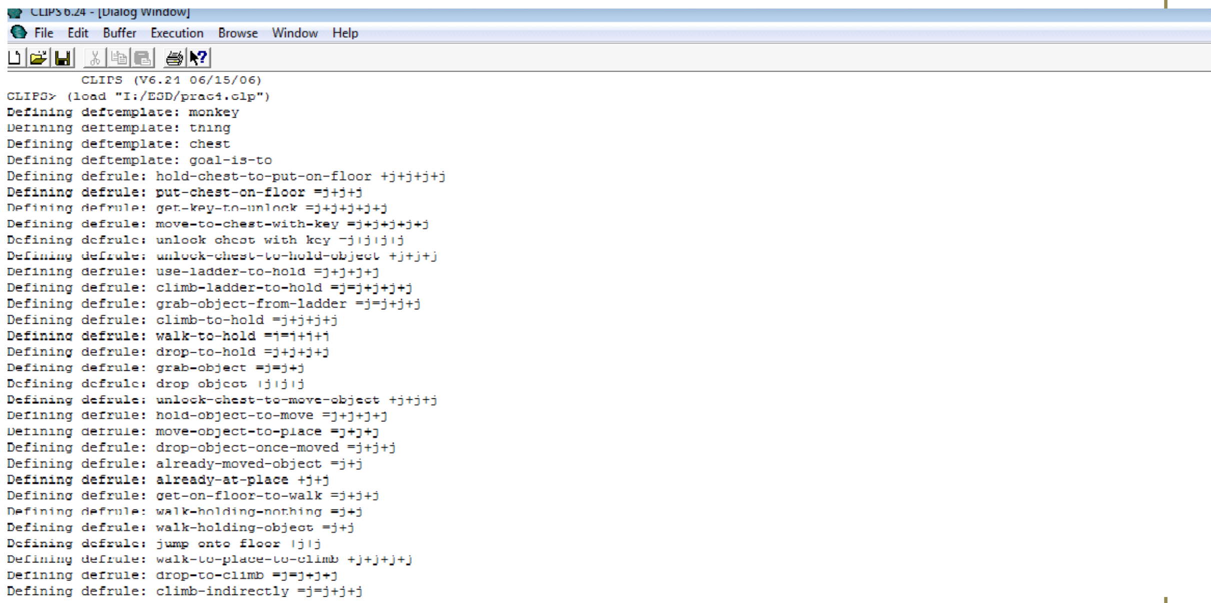
  Here we are considering example of monkeys and they are trying to eat bananas. This example is best for defining planning concept because here monkey is trying to eat banana and for that firstly he has to plan and then start to search the banana. While searching for banana he follows the planning stages and reach to the banana.After getting banana he eats.So here Monkey set the goal and then trying to achieve that goal.Here monkeys goal is to eat banana and for achieving this goal he plan some stages.

- Implementation of above defined problem with solution as planning statement for particular problem definition:

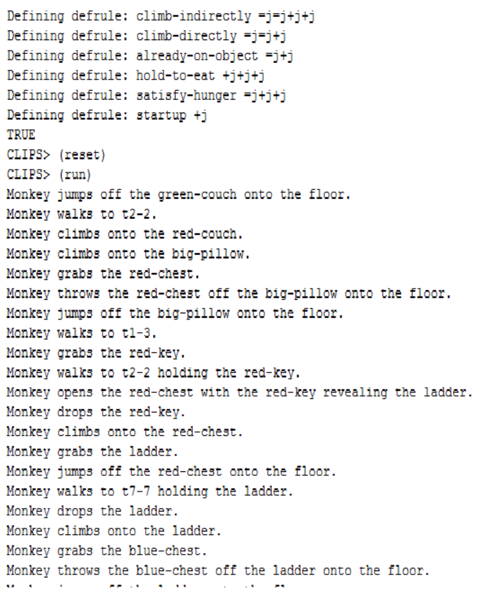
```
;;;======================================================
;;;   Monkees and Bananas Sample Problem
;;;
;;;     This is an extended version of a
;;;     rather common AI planning problem.
;;;     The point is for the monkee to find
;;;     and eat some bananas.
;;;
;;;     CLIPS Version 6.0 Example
;;;
;;;     To execute, merely load, reset and run.
;;;=========================================================

;;;*************
;;;* TEMPLATES *
;;;*************

(deftemplate monkey
   (slot location
      (type SYMBOL)
      (default green-couch))
   (slot on-top-of
      (type SYMBOL)
      (default floor))
   (slot holding
      (type SYMBOL)
      (default nothing)))

(deftemplate thing
   (slot name
      (type SYMBOL)
      (default ?NONE))
   (slot location
      (type SYMBOL)
      (default ?NONE))
   (slot on-top-of
      (type SYMBOL)
      (default floor))
   (slot weight
      (type SYMBOL)
      (allowed-symbols light heavy)
      (default light)))
```

```
(deftemplate chest
   (slot name
      (type SYMBOL)
      (default ?NONE))
   (slot contents
      (type SYMBOL)
      (default ?NONE))
   (slot unlocked-by
      (type SYMBOL)
      (default ?NONE)))

(deftemplate goal-is-to
   (slot action
      (type SYMBOL)
      (allowed-symbols hold unlock eat move on walk-to)
      (default ?NONE))
   (multislot arguments
      (type SYMBOL)
      (default ?NONE)))

;;;*************************
;;;* CHEST UNLOCKING RULES *
;;;*************************

(defrule hold-chest-to-put-on-floor ""
  (goal-is-to (action unlock) (arguments ?chest))
  (thing (name ?chest) (on-top-of ~floor) (weight light))
  (monkey (holding ~?chest))
  (not (goal-is-to (action hold) (arguments ?chest)))
  =>
  (assert (goal-is-to (action hold) (arguments ?chest))))

(defrule put-chest-on-floor ""
  (goal-is-to (action unlock) (arguments ?chest))
  ?monkey <- (monkey (location ?place) (on-top-of ?on) (holding ?chest))
  ?thing <- (thing (name ?chest))
  =>
  (printout t "Monkey throws the " ?chest " off the "
            ?on " onto the floor." crlf)
  (modify ?monkey (holding blank))
  (modify ?thing (location ?place) (on-top-of floor)))

(defrule get-key-to-unlock ""
  (goal-is-to (action unlock) (arguments ?obj))
  (thing (name ?obj) (on-top-of floor))
  (chest (name ?obj) (unlocked-by ?key))
  (monkey (holding ~?key))
  (not (goal-is-to (action hold) (arguments ?key)))
  =>
  (assert (goal-is-to (action hold) (arguments ?key))))

(defrule move-to-chest-with-key ""
  (goal-is-to (action unlock) (arguments ?chest))
  (monkey (location ?mplace) (holding ?key))
  (thing (name ?chest) (location ?cplace&~?mplace) (on-top-of floor))
  (chest (name ?chest) (unlocked-by ?key))
  (not (goal-is-to (action walk-to) (arguments ?cplace)))
  =>
  (assert (goal-is-to (action walk-to) (arguments ?cplace))))

(defrule unlock-chest-with-key ""
  ?goal <- (goal-is-to (action unlock) (arguments ?name))
  ?chest <- (chest (name ?name) (contents ?contents) (unlocked-by ?key))
  (thing (name ?name) (location ?place) (on-top-of ?on))
  (monkey (location ?place) (on-top-of ?on) (holding ?key))
  =>
  (printout t "Monkey opens the " ?name " with the " ?key
            " revealing the " ?contents "." crlf)
  (modify ?chest (contents nothing))
  (assert (thing (name ?contents) (location ?place) (on-top-of ?name)))
  (retract ?goal))

;;;*********************
;;;* HOLD OBJECT RULES *
;;;*********************

(defrule unlock-chest-to-hold-object ""
  (goal-is-to (action hold) (arguments ?obj))
  (chest (name ?chest) (contents ?obj))
  (not (goal-is-to (action unlock) (arguments ?chest)))
  =>
  (assert (goal-is-to (action unlock) (arguments ?chest))))

(defrule use-ladder-to-hold ""
  (goal-is-to (action hold) (arguments ?obj))
  (thing (name ?obj) (location ?place) (on-top-of ceiling) (weight light))
  (not (thing (name ladder) (location ?place)))
  (not (goal-is-to (action move) (arguments ladder ?place)))
  =>
  (assert (goal-is-to (action move) (arguments ladder ?place))))

(defrule climb-ladder-to-hold ""
  (goal-is-to (action hold) (arguments ?obj))
  (thing (name ?obj) (location ?place) (on-top-of ceiling) (weight light))
  (thing (name ladder) (location ?place) (on-top-of floor))
  (monkey (on-top-of ~ladder))
  (not (goal-is-to (action on) (arguments ladder)))
  =>
  (assert (goal-is-to (action on) (arguments ladder))))

(defrule grab-object-from-ladder ""
  ?goal <- (goal-is-to (action hold) (arguments ?name))
  ?thing <- (thing (name ?name) (location ?place)
                   (on-top-of ceiling) (weight light))
  (thing (name ladder) (location ?place))
  ?monkey <- (monkey (location ?place) (on-top-of ladder) (holding blank))
  =>
  (printout t "Monkey grabs the " ?name "." crlf)
  (modify ?thing (location held) (on-top-of held))
  (modify ?monkey (holding ?name))
  (retract ?goal))

(defrule climb-to-hold ""
  (goal-is-to (action hold) (arguments ?obj))
  (thing (name ?obj) (location ?place) (on-top-of ?on&~ceiling) (weight light))
  (monkey (location ?place) (on-top-of ~?on))
  (not (goal-is-to (action on) (arguments ?on)))
  =>
  (assert (goal-is-to (action on) (arguments ?on))))

(defrule walk-to-hold ""
  (goal-is-to (action hold) (arguments ?obj))
  (thing (name ?obj) (location ?place) (on-top-of ~ceiling) (weight light))
  (monkey (location ~?place))
  (not (goal-is-to (action walk-to) (arguments ?place)))
  =>
  (assert (goal-is-to (action walk-to) (arguments ?place))))

(defrule drop-to-hold ""
  (goal-is-to (action hold) (arguments ?obj))
  (thing (name ?obj) (location ?place) (on-top-of ?on) (weight light))
  (monkey (location ?place) (on-top-of ?on) (holding ~blank))
  (not (goal-is-to (action hold) (arguments blank)))
  =>
  (assert (goal-is-to (action hold) (arguments blank))))

(defrule grab-object ""
  ?goal <- (goal-is-to (action hold) (arguments ?name))
  ?thing <- (thing (name ?name) (location ?place)
                   (on-top-of ?on) (weight light))
  ?monkey <- (monkey (location ?place) (on-top-of ?on) (holding blank))
  =>
  (printout t "Monkey grabs the " ?name "." crlf)
  (modify ?thing (location held) (on-top-of held))
  (modify ?monkey (holding ?name))
  (retract ?goal))

(defrule drop-object ""
  ?goal <- (goal-is-to (action hold) (arguments blank))
  ?monkey <- (monkey (location ?place)
                   (on-top-of ?on)
                   (holding ?name&~blank))
  ?thing <- (thing (name ?name))
  =>
  (printout t "Monkey drops the " ?name "." crlf)
  (modify ?monkey (holding blank))
  (modify ?thing (location ?place) (on-top-of ?on))
  (retract ?goal))



;;;*********************
;;;* MOVE OBJECT RULES *
;;;*********************

(defrule unlock-chest-to-move-object ""
  (goal-is-to (action move) (arguments ?obj ?))
  (chest (name ?chest) (contents ?obj))
  (not (goal-is-to (action unlock) (arguments ?chest)))
  =>
  (assert (goal-is-to (action unlock) (arguments ?chest))))

(defrule hold-object-to-move ""
  (goal-is-to (action move) (arguments ?obj ?place))
  (thing (name ?obj) (location ~?place) (weight light))
  (monkey (holding ~?obj))
  (not (goal-is-to (action hold) (arguments ?obj)))
  =>
  (assert (goal-is-to (action hold) (arguments ?obj))))

(defrule move-object-to-place ""
  (goal-is-to (action move) (arguments ?obj ?place))
  (monkey (location ~?place) (holding ?obj))
  (not (goal-is-to (action walk-to) (arguments ?place)))
  =>
  (assert (goal-is-to (action walk-to) (arguments ?place))))

(defrule drop-object-once-moved ""
  ?goal <- (goal-is-to (action move) (arguments ?name ?place))
  ?monkey <- (monkey (location ?place) (holding ?obj))
  ?thing <- (thing (name ?name) (weight light))
  =>
  (printout t "Monkey drops the " ?name "." crlf)
  (modify ?monkey (holding blank))
  (modify ?thing (location ?place) (on-top-of floor))
  (retract ?goal))

(defrule already-moved-object ""
  ?goal <- (goal-is-to (action move) (arguments ?obj ?place))
  (thing (name ?obj) (location ?place))
  =>
  (retract ?goal))

;;;***********************
;;;* WALK TO PLACE RULES *
;;;***********************

(defrule already-at-place ""
  ?goal <- (goal-is-to (action walk-to) (arguments ?place))
  (monkey (location ?place))
  =>
  (retract ?goal))

(defrule get-on-floor-to-walk ""
  (goal-is-to (action walk-to) (arguments ?place))
  (monkey (location ~?place) (on-top-of ~floor))
  (not (goal-is-to (action on) (arguments floor)))
  =>
  (assert (goal-is-to (action on) (arguments floor))))

(defrule walk-holding-nothing ""
  ?goal <- (goal-is-to (action walk-to) (arguments ?place))
  ?monkey <- (monkey (location ~?place) (on-top-of floor) (holding blank))
  =>
  (printout t "Monkey walks to " ?place "." crlf)
  (modify ?monkey (location ?place))
  (retract ?goal))

(defrule walk-holding-object ""
  ?goal <- (goal-is-to (action walk-to) (arguments ?place))
  ?monkey <- (monkey (location ~?place) (on-top-of floor) (holding ?obj&~blank))
  =>
  (printout t "Monkey walks to " ?place " holding the " ?obj "." crlf)
  (modify ?monkey (location ?place))
  (retract ?goal))

;;;************************
;;;* GET ON OBJECT RULES *
;;;************************

(defrule jump-onto-floor ""
  ?goal <- (goal-is-to (action on) (arguments floor))
  ?monkey <- (monkey (on-top-of ?on&~floor))
  =>
  (printout t "Monkey jumps off the " ?on " onto the floor." crlf)
  (modify ?monkey (on-top-of floor))
  (retract ?goal))

(defrule walk-to-place-to-climb ""
  (goal-is-to (action on) (arguments ?obj))
  (thing (name ?obj) (location ?place))
  (monkey (location ~?place))
  (not (goal-is-to (action walk-to) (arguments ?place)))
  =>
  (assert (goal-is-to (action walk-to) (arguments ?place))))

(defrule drop-to-climb ""
  (goal-is-to (action on) (arguments ?obj))
  (thing (name ?obj) (location ?place))
  (monkey (location ?place) (holding ~blank))
  (not (goal-is-to (action hold) (arguments blank)))
  =>
  (assert (goal-is-to (action hold) (arguments blank))))

(defrule climb-indirectly ""
  (goal-is-to (action on) (arguments ?obj))
  (thing (name ?obj) (location ?place) (on-top-of ?on))
  (monkey (location ?place) (on-top-of ~?on&~?obj) (holding blank))
  (not (goal-is-to (action on) (arguments ?on)))
  =>
  (assert (goal-is-to (action on) (arguments ?on))))

(defrule climb-directly ""
  ?goal <- (goal-is-to (action on) (arguments ?obj))
  (thing (name ?obj) (location ?place) (on-top-of ?on))
  ?monkey <- (monkey (location ?place) (on-top-of ?on) (holding blank))
  =>
  (printout t "Monkey climbs onto the " ?obj "." crlf)
  (modify ?monkey (on-top-of ?obj))
  (retract ?goal))

(defrule already-on-object ""
  ?goal <- (goal-is-to (action on) (arguments ?obj))
  (monkey (on-top-of ?obj))
  =>
  (retract ?goal))

;;;********************
;;;* EAT OBJECT RULES *
;;;********************

(defrule hold-to-eat ""
  (goal-is-to (action eat) (arguments ?obj))
  (monkey (holding ~?obj))
  (not (goal-is-to (action hold) (arguments ?obj)))
  =>
  (assert (goal-is-to (action hold) (arguments ?obj))))

(defrule satisfy-hunger ""
  ?goal <- (goal-is-to (action eat) (arguments ?name))
  ?monkey <- (monkey (holding ?name))
  ?thing <- (thing (name ?name))
  =>
  (printout t "Monkey eats the " ?name "." crlf)
  (modify ?monkey (holding blank))
  (retract ?goal ?thing))

;;;***********************
;;;* INITIAL STATE RULE *
;;;***********************

(defrule startup ""
  =>
  (assert (monkey (location t5-7) (on-top-of green-couch) (holding blank)))
  (assert (thing (name green-couch) (location t5-7) (weight heavy)))
  (assert (thing (name red-couch) (location t2-2) (weight heavy)))
  (assert (thing (name big-pillow) (location t2-2) (on-top-of red-couch)))
  (assert (thing (name red-chest) (location t2-2) (on-top-of big-pillow)))
  (assert (chest (name red-chest) (contents ladder) (unlocked-by red-key)))
  (assert (thing (name blue-chest) (location t7-7) (on-top-of ceiling)))
  (assert (chest (name blue-chest) (contents bananas) (unlocked-by blue-key)))
  (assert (thing (name blue-couch) (location t8-8) (weight heavy)))
  (assert (thing (name green-chest) (location t8-8) (on-top-of ceiling)))
  (assert (chest (name green-chest) (contents blue-key) (unlocked-by red-key)))
  (assert (thing (name red-key) (location t1-3)))
  (assert (goal-is-to (action eat) (arguments bananas))))
```

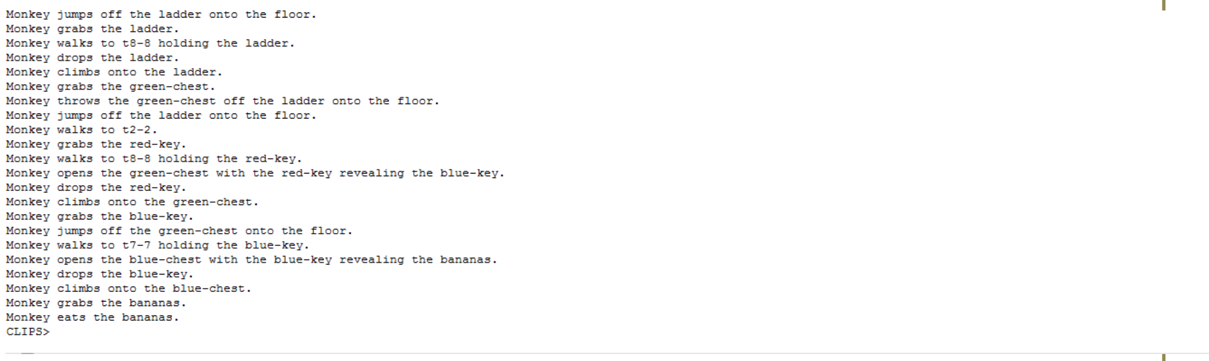
Output:


CLIPS 6.24 - [Dialog Window]
File Edit Buffer Execution Browse Window Help

```
         CLIPS (V6.24 06/15/06)
CLIPS> (load "I:/ESD/prac4.clp")
Defining deftemplate: monkey
Defining deftemplate: thing
Defining deftemplate: chest
Defining deftemplate: goal-is-to
Defining defrule: hold-chest-to-put-on-floor +j+j+j+j
Defining defrule: put-chest-on-floor =j+j+j
Defining defrule: get-key-to-unlock =j+j+j+j+j
Defining defrule: move-to-chest-with-key =j+j+j+j+j
Defining defrule: unlock-chest-with-key =j+j+j+j
Defining defrule: unlock-chest-to-hold-object +j+j+j
Defining defrule: use-ladder-to-hold =j+j+j+j
Defining defrule: climb-ladder-to-hold =j=j+j+j+j
Defining defrule: grab-object-from-ladder =j=j+j+j
Defining defrule: climb-to-hold =j+j+j+j
Defining defrule: walk-to-hold =j=j+j+j
Defining defrule: drop-to-hold =j+j+j+j
Defining defrule: grab-object =j=j+j
Defining defrule: drop-object =j+j+j
Defining defrule: unlock-chest-to-move-object +j+j+j
Defining defrule: hold-object-to-move =j+j+j+j
Defining defrule: move-object-to-place =j+j+j
Defining defrule: drop-object-once-moved =j+j+j
Defining defrule: already-moved-object =j+j
Defining defrule: already-at-place +j+j
Defining defrule: get-on-floor-to-walk =j+j+j
Defining defrule: walk-holding-nothing =j+j
Defining defrule: walk-holding-object =j+j
Defining defrule: jump-onto-floor =j+j
Defining defrule: walk-to-place-to-climb +j+j+j+j
Defining defrule: drop-to-climb =j=j+j+j
Defining defrule: climb-indirectly =j=j+j+j
```



```
Defining defrule: climb-indirectly =j=j+j+j
Defining defrule: climb-directly =j=j+j
Defining defrule: already-on-object =j+j
Defining defrule: hold-to-eat +j+j+j
Defining defrule: satisfy-hunger =j+j+j
Defining defrule: startup +j
TRUE
CLIPS> (reset)
CLIPS> (run)
Monkey jumps off the green-couch onto the floor.
Monkey walks to t2-2.
Monkey climbs onto the red-couch.
Monkey climbs onto the big-pillow.
Monkey grabs the red-chest.
Monkey throws the red-chest off the big-pillow onto the floor.
Monkey jumps off the big-pillow onto the floor.
Monkey walks to t1-3.
Monkey grabs the red-key.
Monkey walks to t2-2 holding the red-key.
Monkey opens the red-chest with the red-key revealing the ladder.
Monkey drops the red-key.
Monkey climbs onto the red-chest.
Monkey grabs the ladder.
Monkey jumps off the red-chest onto the floor.
Monkey walks to t7-7 holding the ladder.
Monkey drops the ladder.
Monkey climbs onto the ladder.
Monkey grabs the blue-chest.
Monkey throws the blue-chest off the ladder onto the floor.
```



```
Monkey jumps off the ladder onto the floor.
Monkey grabs the ladder.
Monkey walks to t8-8 holding the ladder.
Monkey drops the ladder.
Monkey climbs onto the ladder.
Monkey grabs the green-chest.
Monkey throws the green-chest off the ladder onto the floor.
Monkey jumps off the ladder onto the floor.
Monkey walks to t2-2.
Monkey grabs the red-key.
Monkey walks to t8-8 holding the red-key.
Monkey opens the green-chest with the red-key revealing the blue-key.
Monkey drops the red-key.
Monkey climbs onto the green-chest.
Monkey grabs the blue-key.
Monkey jumps off the green-chest onto the floor.
Monkey walks to t7-7 holding the blue-key.
Monkey opens the blue-chest with the blue-key revealing the bananas.
Monkey drops the blue-key.
Monkey climbs onto the blue-chest.
Monkey grabs the bananas.
Monkey eats the bananas.
CLIPS>
```

Conclusion: Thus we have solved the monkeys problem using planning concept in expert system.We have design an expert system which is helpful for monkeys who are trying to eat banana.

PRACTICAL NO. 5

Aim: Design a program / problem based on object-oriented analysis and for expert system (Representing typical objects and situations).

Theory:

- What do you mean by Object Oriented Expert System?

  In object-oriented programming a **class** is a template which describes the common characteristics or attributes of objects. The word template is used in the sense of a tool that is used to build objects having attributes. Each class is an abstraction of a real-worls system or some other logical system that we are trying to model.

- Analyse the Problem

  Object oriented program approach that models a system as a group of interacting objects.Each object is characterized by its class,its state and behavior.In our eg, it analyses class containing variable declarations,objects and functions for multiplication.

- Implementation of Object Oriented Programming

DEPTH FIRST SEARCH

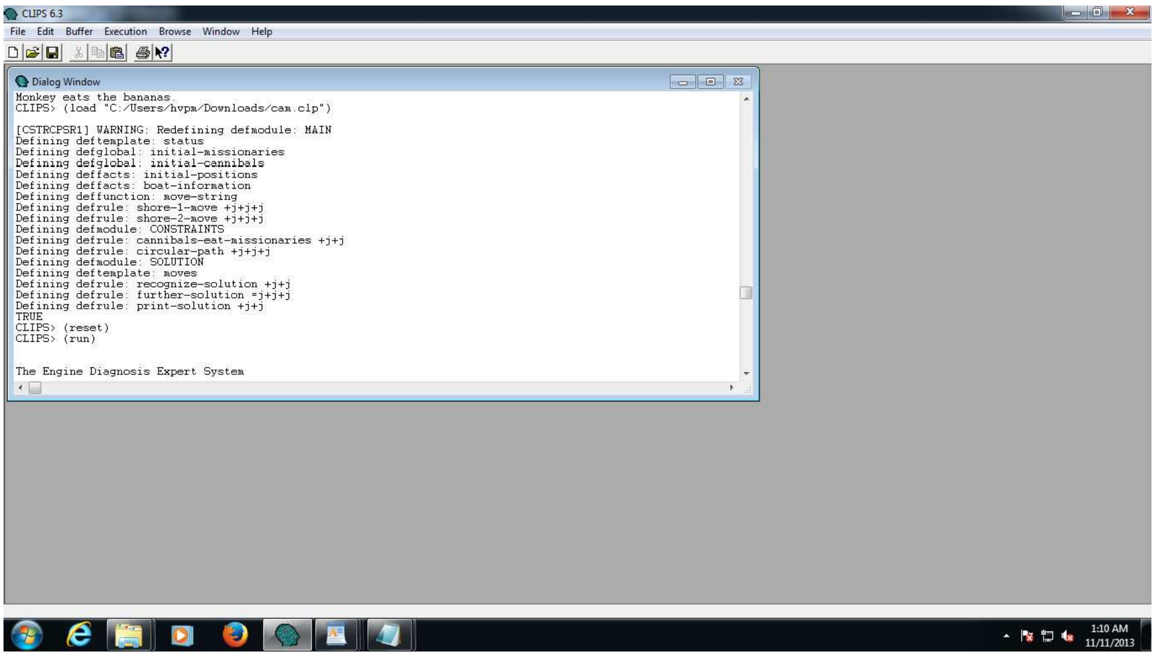
```
;;;     CLIPS Version 6.01 Example

;;;

;;;     To execute, merely load, reset and run.

;;;======================================================




(defmodule MAIN

  (export deftemplate status)

  (export defglobal initial-missionaries initial-cannibals))


;;;*************

;;;* TEMPLATES *

;;;*************
```

```
;;; The status facts hold the state
;;; information of the search tree.

(deftemplate MAIN::status
  (slot search-depth (type INTEGER) (range 1 ?VARIABLE))
  (slot parent (type FACT-ADDRESS SYMBOL) (allowed-symbols no-parent))
  (slot shore-1-missionaries (type INTEGER) (range 0 ?VARIABLE))
  (slot shore-1-cannibals (type INTEGER) (range 0 ?VARIABLE))
  (slot shore-2-missionaries (type INTEGER) (range 0 ?VARIABLE))
  (slot shore-2-cannibals (type INTEGER) (range 0 ?VARIABLE))
  (slot boat-location (type SYMBOL) (allowed-values shore-1 shore-2))
  (slot last-move (type STRING)))

;;;*****************
;;;* INITIAL STATE *
;;;*****************

(defglobal MAIN ?*initial-missionaries* = 3
                ?*initial-cannibals* = 3)

(deffacts MAIN::initial-positions
  (status (search-depth 1)
          (parent no-parent)
          (shore-1-missionaries ?*initial-missionaries*)
          (shore-2-missionaries 0)
          (shore-1-cannibals ?*initial-cannibals*)
          (shore-2-cannibals 0)
          (boat-location shore-1)
          (last-move "No move.")))

(deffacts MAIN::boat-information
  (boat-can-hold 2))

;;;****************************************
;;;* FUNCTION FOR MOVE DESCRIPTION STRING *
;;;****************************************

(deffunction MAIN::move-string (?missionaries ?cannibals ?shore)
  (switch ?missionaries
    (case 0 then
      (if (eq ?cannibals 1)
         then (format nil "Move 1 cannibal to %s.%n" ?shore)
         else (format nil "Move %d cannibals to %s.%n" ?cannibals ?shore)))
    (case 1 then
      (switch ?cannibals
        (case 0 then
          (format nil "Move 1 missionary to %s.%n" ?shore))
        (case 1 then
          (format nil "Move 1 missionary and 1 cannibal to %s.%n" ?shore))
        (default then
          (format nil "Move 1 missionary and %d cannibals to %s.%n"
                  ?cannibals ?shore))))
    (default
      (switch ?cannibals
        (case 0 then
          (format nil "Move %d missionaries to %s.%n" ?missionaries ?shore))
        (case 1 then
          (format nil "Move %d missionaries and 1 cannibal to %s.%n"
                  ?missionaries ?shore))
        (default then
          (format nil "Move %d missionary and %d cannibals to %s.%n"
                  ?missionaries  ?cannibals  ?shore)))))

;;;***********************
;;;* GENERATE PATH RULES *
;;;***********************

(defrule  MAIN::shore-1-move
  ?node <- (status (search-depth ?num)
              (boat-location shore-1)
              (shore-1-missionaries ?s1m)
              (shore-1-cannibals  ?s1c)
              (shore-2-missionaries ?s2m)
              (shore-2-cannibals  ?s2c))
  (boat-can-hold  ?limit)
  =>
  (bind ?max-missionaries (min ?s1m ?limit))
  (loop-for-count (?missionaries 0 ?max-missionaries)
    (bind ?min-cannibals (max 0 (- 1 ?missionaries)))
    (bind ?max-cannibals (min ?s1c (- ?limit ?missionaries)))
    (loop-for-count (?cannibals ?min-cannibals ?max-cannibals)
      (duplicate ?node (search-depth =(+ 1 ?num))
                (parent ?node)
                (shore-1-missionaries (- ?s1m ?missionaries))
                (shore-1-cannibals (- ?s1c ?cannibals))
                (shore-2-missionaries (+ ?s2m ?missionaries))
                (shore-2-cannibals (+ ?s2c ?cannibals))
                (boat-location shore-2)
                (last-move (move-string ?missionaries ?cannibals shore-2))))))

(defrule MAIN::shore-2-move
 ?node <- (status (search-depth ?num)
              (boat-location shore-2)
              (shore-1-missionaries ?s1m)
              (shore-1-cannibals ?s1c)
              (shore-2-missionaries ?s2m)
              (shore-2-cannibals ?s2c))
 (boat-can-hold ?limit)
 =>
 (bind ?max-missionaries (min ?s2m ?limit))
 (loop-for-count (?missionaries 0 ?max-missionaries)
  (bind ?min-cannibals (max 0 (- 1 ?missionaries)))
  (bind ?max-cannibals (min ?s2c (- ?limit ?missionaries)))
  (loop-for-count (?cannibals ?min-cannibals ?max-cannibals)
  (duplicate ?node (search-depth =(+ 1 ?num))
                (parent ?node)
                (shore-1-missionaries (+ ?s1m ?missionaries))
                (shore-1-cannibals (+ ?s1c ?cannibals))
```

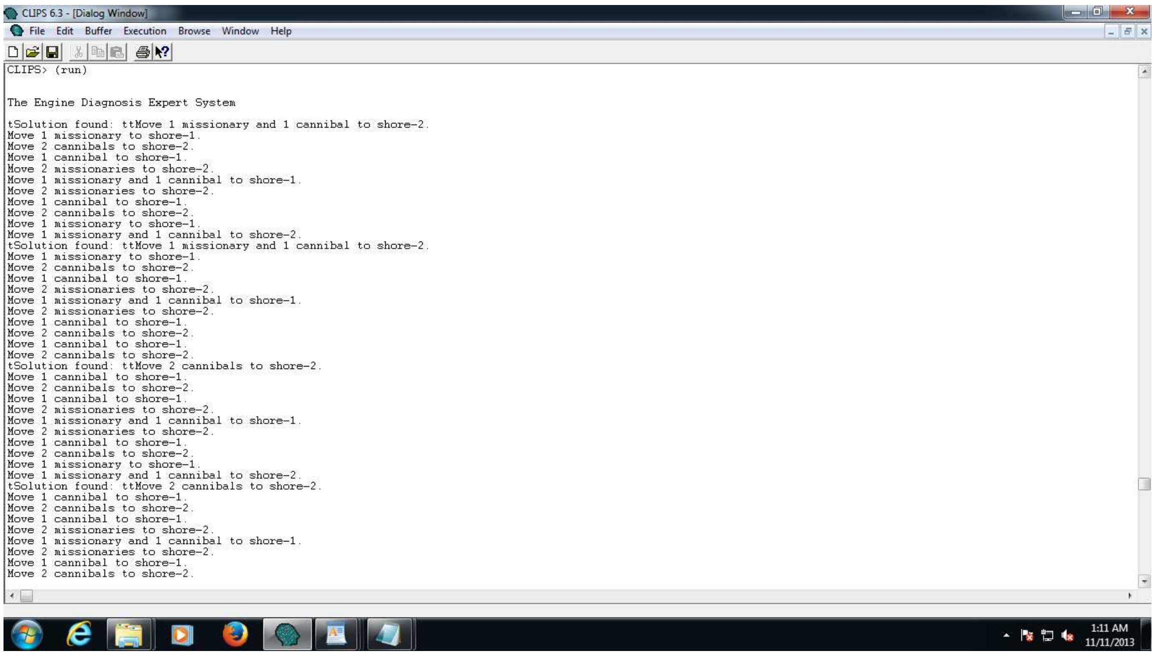
```
                    (shore-2-missionaries (- ?s2m ?missionaries))

                    (shore-2-cannibals (- ?s2c ?cannibals))

                    (boat-location shore-1)

                    (last-move (move-string ?missionaries ?cannibals shore-1)))))


;;;******************************

;;;* CONSTRAINT VIOLATION RULES *

;;;******************************


(defmodule CONSTRAINTS

  (import MAIN deftemplate status))


(defrule CONSTRAINTS::cannibals-eat-missionaries

  (declare (auto-focus TRUE))

  ?node <- (status (shore-1-missionaries ?s1m)

                  (shore-1-cannibals ?s1c)

                  (shore-2-missionaries ?s2m)

                  (shore-2-cannibals ?s2c))

  (test (or (and (> ?s2c ?s2m) (<> ?s2m 0))

            (and (> ?s1c ?s1m) (<> ?s1m 0))))

  =>

  (retract ?node))


(defrule CONSTRAINTS::circular-path

  (declare (auto-focus TRUE))

  (status (search-depth ?sd1)

          (boat-location ?bl)
```

```
        (shore-1-missionaries ?s1m)
        (shore-1-cannibals ?s1c)
        (shore-2-missionaries ?s2m)
        (shore-2-cannibals ?s2c))
  ?node <- (status (search-depth ?sd2&:(< ?sd1 ?sd2))
            (boat-location ?bl)
            (shore-1-missionaries ?s1m)
            (shore-1-cannibals ?s1c)
            (shore-2-missionaries ?s2m)
            (shore-2-cannibals ?s2c))
  =>
  (retract ?node))

;;;*********************************
;;;* FIND AND PRINT SOLUTION RULES *
;;;*********************************

(defmodule SOLUTION
  (import MAIN deftemplate status)
  (import MAIN defglobal initial-missionaries initial-cannibals))

(deftemplate SOLUTION::moves
   (slot id (type FACT-ADDRESS SYMBOL) (allowed-symbols no-parent))
   (multislot moves-list
      (type STRING)))

(defrule  SOLUTION::recognize-solution
  (declare (auto-focus TRUE))
  ?node <- (status (parent ?parent)
            (shore-2-missionaries ?m&:(= ?m ?*initial-missionaries*))
            (shore-2-cannibals ?c&:(= ?c ?*initial-cannibals*))
            (last-move ?move))
  =>
  (retract ?node)
  (assert (moves (id ?parent) (moves-list ?move))))

(defrule SOLUTION::further-solution
  ?node <- (status (parent ?parent)
            (last-move ?move))
  ?mv <- (moves (id ?node) (moves-list $?rest))
  =>
  (modify ?mv (id ?parent) (moves-list ?move ?rest)))

(defrule SOLUTION::print-solution
  ?mv <- (moves (id no-parent) (moves-list "No move." $?m))
  =>
  (retract ?mv)
  (printout t t  "Solution found: " t t)
  (progn$ (?move ?m) (printout t ?move)))
```

OUTPUT:



```
Monkey eats the bananas.
CLIPS> (load "C:/Users/hvpm/Downloads/cam.clp")

[CSTRCPSR1] WARNING: Redefining defmodule: MAIN
Defining deftemplate: status
Defining defglobal: initial-missionaries
Defining defglobal: initial-cannibals
Defining deffacts: initial-positions
Defining deffacts: boat-information
Defining deffunction: move-string
Defining defrule: shore-1-move +j+j+j
Defining defrule: shore-2-move +j+j+j
Defining defmodule: CONSTRAINTS
Defining defrule: cannibals-eat-missionaries +j+j
Defining defrule: circular-path +j+j+j
Defining defmodule: SOLUTION
Defining deftemplate: moves
Defining defrule: recognize-solution +j+j
Defining defrule: further-solution =j+j+j
Defining defrule: print-solution +j+j
TRUE
CLIPS> (reset)
CLIPS> (run)


The Engine Diagnosis Expert System
```


CLIPS 6.3 - [Dialog Window]
File Edit Buffer Execution Browse Window Help
1:11 AM
11/11/2013

```
CLIPS> (run)


The Engine Diagnosis Expert System

tSolution found: ttMove 1 missionary and 1 cannibal to shore-2.
Move 1 missionary to shore-1.
Move 2 cannibals to shore-2.
Move 1 cannibal to shore-1.
Move 2 missionaries to shore-2.
Move 1 missionary and 1 cannibal to shore-1.
Move 2 missionaries to shore-2.
Move 1 cannibal to shore-1.
Move 2 cannibals to shore-2.
Move 1 missionary to shore-1.
Move 1 missionary and 1 cannibal to shore-2.
tSolution found: ttMove 1 missionary and 1 cannibal to shore-2.
Move 1 missionary to shore-1.
Move 2 cannibals to shore-2.
Move 1 cannibal to shore-1.
Move 2 missionaries to shore-2.
Move 1 missionary and 1 cannibal to shore-1.
Move 2 missionaries to shore-2.
Move 1 cannibal to shore-1.
Move 2 cannibals to shore-2.
Move 1 cannibal to shore-1.
Move 2 cannibals to shore-2.
tSolution found: ttMove 2 cannibals to shore-2.
Move 1 cannibal to shore-1.
Move 2 cannibals to shore-2.
Move 1 cannibal to shore-1.
Move 2 missionaries to shore-2.
Move 1 missionary and 1 cannibal to shore-1.
Move 2 missionaries to shore-2.
Move 1 cannibal to shore-1.
Move 2 cannibals to shore-2.
Move 1 missionary to shore-1.
Move 1 missionary and 1 cannibal to shore-2.
tSolution found: ttMove 2 cannibals to shore-2.
Move 1 cannibal to shore-1.
Move 2 cannibals to shore-2.
Move 1 cannibal to shore-1.
Move 2 missionaries to shore-2.
Move 1 missionary and 1 cannibal to shore-1.
Move 2 missionaries to shore-2.
Move 1 cannibal to shore-1.
Move 2 cannibals to shore-2.
```

Conclusion : Thus we have design a program / problem based on object-oriented analysis and for expert system (Representing typical objects and situations).

PRACTICAL NO.6

Aim: Design a program / problem based on knowledge Acquisition with respect to expert system shell.

Theory:

- What is knowledge acquisition?

The success of knowledge based systems lies in the quality and extent of the knowledge available to the system. Acquiring and validating a large croups of consistent, correlated knowledge is not a trivial problem . This has give the acquisition process an especially important role in the design and implementation of these systems. Consequently, effective acquisition methods have become one of the principal challenges for the AI researches. The goals of this branch of AI are the discovery and development of efficient, cost effective methods of acquisition. Some important progress has recently been made in this area with the development of sophisticated editors and some general concepts related to acquisition and learning.

Knowledge acquisition is the process of adding new knowledge to a knowledge base and refining or otherwise improving knowledge that was previously acquired. Acquisition is usually associated with some purpose such as expanding the capabilities of a system or improving its performance at some specified task. It is goal oriented creation and refinement of knowledge . It may consist of facts, rules , concepts, procedures, heuristics, formulas, relationships, statistics or other useful information

- Identify and Analyze Problem:

Here we have considered Engine problem which is the best example of knowledge acquisition.In this we define some basic problem while starting with engine.For example we consider 7 different problems and by improving knowledge that was previously acquired, we provide the best solution.The success of this expert system lies in the quality and extent of the knowledge available to the system.This expert system gives the best solution because we add some new knowledge to a knowledge base and also improve the knowledge that was previously acquired.

- Implement the Problem:

```
;;####################################################
;;####################################################
;;              RX7.CLP                    ##
;;    Mazda RX-7 rotary engine troubleshooter.     ##
;;                                   ##
;;####################################################
;;####################################################
;;
;; Operating instructions:
;;   1.) Execute the CLIPS interpreter.
;;   2.) From the CLIPS prompt, type (load "rx7.clp").
;;   3.) Type (reset) and then (run) to execute the program.
;;
;;***********************************
;; Set up initial facts for testing.
;;***********************************
```

```
;;
(deffacts init
  (troubleshoot-mode engine)
  (menu-level engine main))
;;
;;*****************************************************
;; Clear the screen and present the main menu.        *
;;                                                    *
;; Variables:                                         *
;;   ?ml - used for retracting the menu level.        *
;;   ?response - used for binding in the users input.*
;;*****************************************************
;;
(defrule main-menu
  (declare (salience 500))
  (troubleshoot-mode engine)
  ?ml <- (menu-level engine main)
=>
  (retract ?ml)
;;** print 25 crlf's to clear screen **
  (printout t crlf crlf crlf)
  (printout t
  "        Choose one of the problem areas listed below" crlf
  "        by typing a letter and pressing the return key." crlf crlf
  "                1.) Difficult starting." crlf
  "                2.) Poor idling." crlf
  "                3.) Insufficient power." crlf
  "                4.) Abnormal combustion." crlf
  "                5.) Excessive oil consumption." crlf
  "                6.) Engine noise." crlf
  "                7.) Quit the program." crlf crlf
  "        Choice: " )
  (bind ?response (read))
  (assert (problem-response engine ?response))
  (printout t crlf))
;;
;;
;;****************************************
;; If the user enters a response of 7   *
;;    from the main menu,               *
;; Then print ending message and clear  *
;;      the fact base.                  *
;;****************************************
(defrule user-quits
  (troubleshoot-mode engine)
  (problem-response engine 7)
=>
  (printout t "You have QUIT the program." crlf)
  (halt))
;;
;;*******************************
;; If the user selects 1        *
;; Then assert that the problem *
;;      is DIFFICULT STARTING.  *
;;                              *
;; Variables:                  *
;;   ?pr - for retracting the   *
;;         numeric problem       *
;;         response.             *
;;*******************************
(defrule difficult-starting
  (troubleshoot-mode engine)
  ?pr <- (problem-response engine 1)
=>
  (retract ?pr)
  (assert (menu-level engine possible-causes-1-1))
  (assert (problem engine difficult-starting)))
;;
;;*******************************
;; If the user selects 2        *
;; Then assert that the problem *
;;      is POOR IDLING.         *
;;                             *
;; Variables:                  *
;;   ?pr - for retracting the   *
;;         numeric problem       *
;;         response.             *
;;*******************************
(defrule poor-idling
  (troubleshoot-mode engine)
  ?pr <- (problem-response engine 2)
=>
  (retract ?pr)
  (assert (menu-level engine possible-causes-1-1))
  (assert (problem engine poor-idling)))
;;
;;*******************************
;; If the user selects 3        *
;; Then assert that the problem *
;;      is INSUFFICIENT POWER.  *
;;                             *
;; Variables:                  *
;;   ?pr - for retracting the   *
;;         numeric problem       *
;;         response.             *
;;*******************************
(defrule insufficient-power
(troubleshoot-mode engine)
  ?pr <- (problem-response engine 3)
=>
  (retract ?pr)
  (assert (menu-level engine possible-causes-1-1))
  (assert (problem engine insufficient-power)))
;;
;;*******************************
;; If the user selects 4        *
;; Then assert that the problem *
;;      is ABNORMAL COMBUSTION. *
;;                             *
;; Variables:                  *
;;   ?pr - for retracting the   *
;;         numeric problem      *
;;         response.            *
;;*******************************
(defrule abnormal-combustion
(troubleshoot-mode engine)
  ?pr <- (problem-response engine 4)
=>
  (retract ?pr)
  (assert (menu-level engine possible-causes-1-2))
  (assert (problem engine abnormal-combustion)))
;;
;;*************************************
;; If the user selects 5              *
;; Then assert that the problem       *
;;      is EXCESSIVE OIL CONSUMPTION. *
;;                          *
;; Variables:               *
;;   ?pr - for retracting the   *
;;         numeric problem      *
;;         response.            *
;;*************************************
(defrule excessive-oil-consumption
(troubleshoot-mode engine)
  ?pr <- (problem-response engine 5)
=>
  (retract ?pr)
  (assert (menu-level engine possible-causes-1-3))
  (assert (problem engine excessive-oil-consumption)))
;;
;;*******************************
;; If the user selects 6        *
;; Then assert that the problem *
;;      is ENGINE NOISE.        *
;;                          *
;; Variables:               *
;;   ?pr - for retracting the   *
;;         numeric problem      *
;;         response.            *
;;*******************************
(defrule engine-noise
  (troubleshoot-mode engine)
  ?pr <- (problem-response engine 6)
=>
  (retract ?pr)
  (assert (menu-level engine possible-causes-1-4))
  (assert (problem engine engine-noise)))
;;
;;*******************************************
;; Clear the screen and present the possible *
;; causes for DIFFICULT STARTING, or         *
;; POOR IDLING, or INSUFFICIENT POWER        *
;; depending on the variable ?problem.       *
;;                                   *
;; Variables:                        *
;;   ?problem - the problem selected by the  *
;;             user.                        *
;;   ?response - used for binding in the      *
;;              users input.                *
;;********************************************
;;
(defrule possible-causes-1-1
  (troubleshoot-mode engine)
  (menu-level engine possible-causes-1-1)
  (problem  engine  ?problem&difficult-starting|poor-idling|insufficient-power)
=>
  (printout t crlf crlf crlf)
  (printout t
  "       You selected " ?problem " as the problem you would" crlf
  "       like to solve.  If you change your mind and would like" crlf
  "       to review the main menu, press 0 now, otherwise select one" crlf
  "       of the possible causes of " ?problem " listed below."
          crlf crlf
  "             0.) Return to main menu." crlf
  "             1.) Insufficient compression." crlf
  "             2.) Malfunction of fuel system." crlf
  "             3.) Malfunction of electrical system." crlf crlf
  "       Choice: " )
  (bind ?response (read))
  (assert (possible-cause ?problem ?response))
  (printout t crlf))
;;
;;********************************************
;; Clear the screen and present the possible *
;; causes for ABNORMAL COMBUSTION.           *
;;                                          *
;; Variables:                               *
;;   ?response - used for binding in the      *
;;              users response.             *
;;********************************************
(defrule possible-causes-1-2
  (troubleshoot-mode engine)
  (menu-level engine possible-causes-1-2)
  (problem engine abnormal-combustion)
=>
  (printout t crlf crlf crlf)
  (printout t
  "       You selected abnormal-combustion as the problem you would" crlf
  "       like to solve.  If you change your mind and would like"    crlf
  "       to review the main menu, press 0 now, otherwise select one" crlf
  "       of the possible causes of abnormal-combustion listed below."
  crlf crlf
  "              0.) Return to main menu." crlf
  "              1.) Malfunction in combustion chamber." crlf
  "              2.) Malfunction of fuel system." crlf
  "              3.) Malfunction of ignition system." crlf
  "       Choice: ")
  (bind ?response (read))
  (assert (possible-cause abnormal-combustion ?response))
  (printout t crlf))
```

```
;;
;;*******************************************
;; Clear the screen and present the possible *
;; causes for EXCESSIVE OIL CONSUMPTION.      *
;;                                      *
;; Variables:                               *
;;   ?response - used for binding in the      *
;;              users response.             *
;;*******************************************
(defrule possible-causes-1-3
  (troubleshoot-mode engine)
  (menu-level engine possible-causes-1-3)
  (problem engine excessive-oil-consumption)
=>
  (printout t crlf crlf crlf)
  (printout t
  "       You selected excessive-oil-consumption as the problem you" crlf
  "       would like to solve.  If you change your mind and would like" crlf
  "       to review the main menu, press 0 now, otherwise select one" crlf
  "       of the possible causes of excessive-oil-consumption listed below."
  crlf crlf
  "             0.) Return to main menu." crlf
  "             1.) Leakage into combustion chamber." crlf
  "             2.) Leakage into coolant passages." crlf
  "             3.) Leakage to outside of engine." crlf
  "             4.) Malfunction of lubricating system." crlf
  "       Choice: ")
  (bind ?response (read))
  (assert (possible-cause excessive-oil-consumption ?response))
  (printout t crlf))
;;
;;*******************************************
;; Clear the screen and present the possible *
;; causes for ENGINE NOISE.                 *
;;                                      *
;; Variables:                               *
;;   ?response - used for binding in the      *
;;              users response.             *
;;*******************************************
(defrule possible-causes-1-4
  (troubleshoot-mode engine)
  (menu-level engine possible-causes-1-4)
  (problem engine engine-noise)
=>
  (printout t crlf crlf crlf)
  (printout t
  "       You selected engine-noise as the problem you would like" crlf
  "       to solve.  If you change your mind and would like to" crlf
  "       review the main menu, press 0 now, otherwise select one" crlf
  "       of the possible causes of engine-noise listed below."
  crlf crlf
  "             0.) Return to main menu." crlf
  "             1.) Gas seal noise." crlf
  "             2.) Knocking noise." crlf
  "             3.) Hitting noise." crlf
  "               4.) Other." crlf
  "       Choice: ")
  (bind ?response (read))
  (assert (possible-cause engine-noise ?response))
  (printout t crlf))
;;
;;**************************************************
;; This rule replaces the users numeric input for *
;; the possible cause of insufficient compression *
;; to the textual representation.  This rule uses *
;; a variable for the problem field so it is not  *
;; tied to any particular problem.                *
;;                                                *
;; Variables:                                     *
;;   ?ml - used for retracting the menu level.    *
;;   ?pc - used for retracting the possible cause *
;;        fact.                                   *
;;   ?problem - the problem selected by the user. *
;;**************************************************
(defrule numeric-to-text-insufficient-compression-1-1
(troubleshoot-mode engine)
  ?ml <- (menu-level engine possible-causes-1-1)
  ?pc <- (possible-cause ?problem 1)
=>
  (retract ?ml)
  (assert (menu-level engine possible-causes-1-1-1))
  (retract ?pc)
  (assert (possible-cause ?problem insufficient-compression)))
;;
;;***************************************************
;; This rule replaces the users numeric input for  *
;; the possible cause of malfunction in combustion *
;; chamber to the textual representation.  This    *
;; rule is for switching from the ABNORMAL         *
;; COMBUSTION menu to the sub menu.                *
;;                                                 *
;; Variables:                                      *
;;   ?ml - used for retracting the menu level.     *
;;   ?pc - used for retracting the possible cause  *
;;        fact.                                    *
;;   ?problem - the problem selected by the user.  *
;;***************************************************
(defrule numeric-to-text-malfunction-in-combustion-chamber-1-2
(troubleshoot-mode engine)
  ?ml <- (menu-level engine possible-causes-1-2)
  ?pc <- (possible-cause ?problem 1)
=>
  (retract ?ml)
  (assert (menu-level engine possible-causes-1-2-1))
  (retract ?pc)
  (assert (possible-cause ?problem malfunction-in-combustion-chamber)))
;;
;;***************************************************
;; This rule replaces the users numeric input for  *
;; the possible cause of leakage into combustion   *
;; chamber to the textual representation.  This     *
;; rule is for switching from the EXCESSIVE OIL     *
;; CONSUMPTION menu to the sub menu.                *
;;                                                  *
;; Variables:                                       *
;;   ?ml - used for retracting the menu level.      *
;;   ?pc - used for retracting the possible cause  *
;;        fact.                                     *
;;   ?problem - the problem selected by the user.  *
;;***************************************************
(defrule  numeric-to-text-leakage-into-combustion-chamber-1-3
(troubleshoot-mode engine)
  ?ml <- (menu-level engine possible-causes-1-3)
  ?pc <- (possible-cause ?problem 1)
=>
  (retract ?ml)
  (assert (menu-level engine possible-causes-1-3-1))
  (retract ?pc)
  (assert (possible-cause ?problem leakage-into-combustion-chamber)))
;;
;;***************************************************
;; This rule replaces the users numeric input for  *
;; the possible cause of leakage into coolant       *
;; passages to the textual representation.  This   *
;; rule is for switching from the EXCESSIVE OIL     *
;; CONSUMPTION menu to the sub menu.                *
;;                                                  *
;; Variables:                                       *
;;   ?ml - used for retracting the menu level.      *
;;   ?pc - used for retracting the possible cause  *
;;        fact.                                     *
;;   ?problem - the problem selected by the user.  *
;;***************************************************
(defrule numeric-to-text-leakage-into-coolant-passages-1-3
(troubleshoot-mode engine)
  ?ml <- (menu-level engine possible-causes-1-3)
  ?pc <- (possible-cause ?problem 2)
=>
  (retract ?ml)
  (assert (menu-level engine possible-causes-1-3-2))
  (retract ?pc)
  (assert (possible-cause ?problem leakage-into-coolant-passages)))
;;
;;***************************************************
;; This rule replaces the users numeric input for  *
;; the possible cause of gas seal noise to the      *
;; textual representation.  This rule is for        *
;; switching from the EXCESSIVE OIL CONSUMPTION     *
;; menu to the sub menu.                            *
;;                                                  *
;; Variables:                                       *
;;   ?ml - used for retracting the menu level.      *
;;   ?pc - used for retracting the possible cause  *
;;        fact.                                     *
;;   ?problem - the problem selected by the user.  *
```

```
;;*****************************************************
(defrule numeric-to-text-gas-seal-noise-1-4
  (troubleshoot-mode engine)
  ?ml <- (menu-level engine possible-causes-1-4)
  ?pc <- (possible-cause ?problem 1)
=>
  (retract ?ml)
  (assert (menu-level engine possible-causes-1-4-1))
  (retract ?pc)
  (assert (possible-cause ?problem gas-seal-noise)))
;;
;;*****************************************************
;; This rule replaces the users numeric input for  *
;; the possible cause of knocking noise to the      *
;; textual representation.  This rule is for        *
;; switching from the EXCESSIVE OIL CONSUMPTION     *
;; menu to the sub menu.                            *
;;                                                  *
;; Variables:                                       *
;;   ?ml - used for retracting the menu level.      *
;;   ?pc - used for retracting the possible cause  *
;;         fact.                                    *
;;   ?problem - the problem selected by the user.  *
;;*****************************************************
(defrule numeric-to-text-knocking-noise-1-4
  (troubleshoot-mode engine)
  ?ml <- (menu-level engine possible-causes-1-4)
  ?pc <- (possible-cause ?problem 2)
=>
  (retract ?ml)
  (assert (menu-level engine possible-causes-1-4-2))
  (retract ?pc)
  (assert (possible-cause ?problem knocking-noise)))
;;
;;*****************************************************
;; This rule replaces the users numeric input for  *
;; the possible cause of hitting noise to the       *
;; textual representation.  This rule is for        *
;; switching from the EXCESSIVE OIL CONSUMPTION     *
;; menu to the sub menu.                            *
;;                                                  *
;; Variables:                                       *
;;   ?ml - used for retracting the menu level.      *
;;   ?pc - used for retracting the possible cause  *
;;         fact.                                    *
;;   ?problem - the problem selected by the user.  *
;;*****************************************************
(defrule numeric-to-text-hitting-noise-1-4
  (troubleshoot-mode engine)
  ?ml <- (menu-level engine possible-causes-1-4)
  ?pc <- (possible-cause ?problem 3)
=>
  (retract ?ml)
  (assert (menu-level engine possible-causes-1-4-3))
  (retract ?pc)
  (assert (possible-cause ?problem hitting-noise)))
;;
;;****************************************************
;; This rule replaces the users numeric input for  *
;; the possible cause of other to the textual       *
;; representation.  This rule is for switching      *
;; from the EXCESSIVE OIL CONSUMPTION menu to the  *
;; sub menu.                                  *
;;                                        *
;; Variables:                                 *
;;   ?ml - used for retracting the menu level.     *
;;   ?pc - used for retracting the possible cause  *
;;         fact.                              *
;;   ?problem - the problem selected by the user.  *
;;****************************************************
(defrule numeric-to-text-other-1-4
  (troubleshoot-mode engine)
  ?ml <- (menu-level engine possible-causes-1-4)
  ?pc <- (possible-cause ?problem 4)
=>
  (retract ?ml)
  (assert (menu-level engine possible-causes-1-4-4))
  (retract ?pc)
  (assert (possible-cause ?problem other)))
;;
;;********************************************************
;; If insufficient compression is a possible cause of  *
;;    some problem,                               *
;; Then present the possible causes of the problem.     *
;;                                            *
;; Variables:                                     *
;;   ?pc - used for retracting the possible cause.      *
;;   ?problem - the problem selected by the user.       *
;;   ?response - the users response to the menu prompt.*
;;********************************************************
(defrule possible-causes-of-insufficient-compression
  (troubleshoot-mode engine)
  (menu-level engine possible-causes-1-1-1)
  ?pc <- (possible-cause ?problem insufficient-compression)
=>
  (retract ?pc)
  (printout t crlf crlf crlf)
  (printout t
  "       You selected INSUFFICIENT COMPRESSION as a possible cause" crlf
  "       of the problem " ?problem ".  If you change your mind and" crlf
  "       would like to review the previous choices, press 0 now,"   crlf
  "       otherwise select one of the possible causes of INSUFFICIENT" crlf
  "       COMPRESSION listed below." crlf crlf
  "              0.) Return to previous menu." crlf
  "              1.) Deformation or abnormal wear of side housing." crlf
  "              2.) Deformation of abnormal wear of rotor housing." crlf
  "              3.) Wear of rotor grooves." crlf
  "              4.) Deformation or poor fastening of gas seals." crlf
  "              5.) Worn or weak spring." crlf crlf
  "       Choice: ")
```

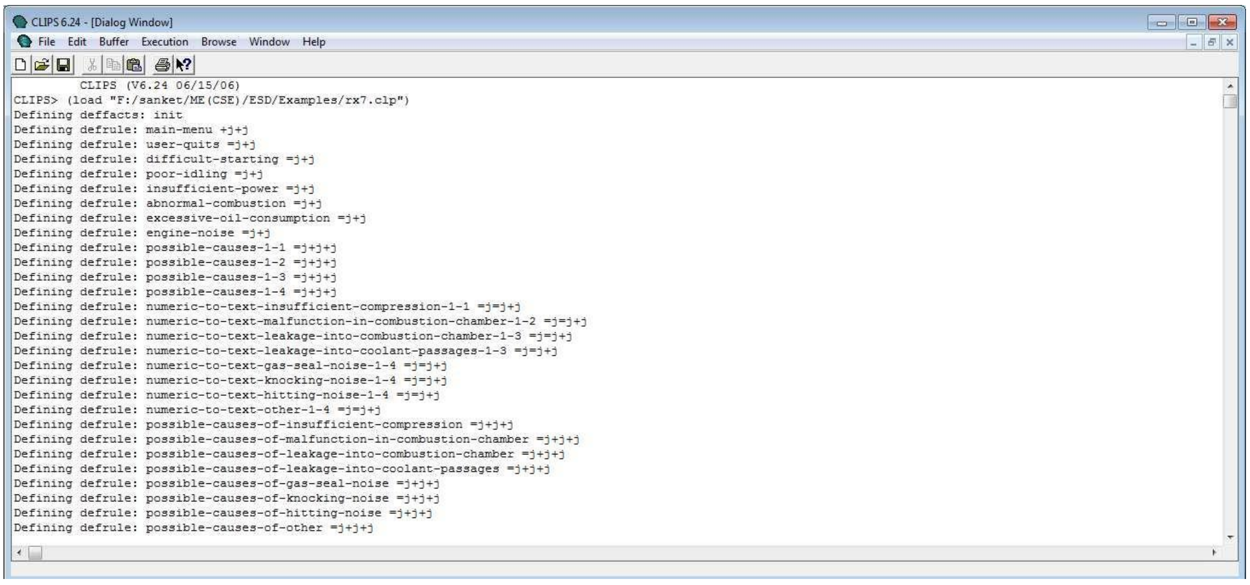
```
    (bind ?response (read))
    (assert (possible-cause ?problem ?response))
    (printout t crlf))
;;
;;*****************************************************
;; This is the sub menu for malfunction in combustion  *
;; chamber under problem of abnormal combustion.       *
;; If malfunction in combustion chamber is a possible  *
;;    cause of some problem,                           *
;; Then present the possible causes of malfunction in  *
;;      combustion chamber.                            *
;;                                                     *
;; Variables:                                          *
;;   ?pc - used for retracting the possible cause.     *
;;   ?problem - the problem selected by the user.      *
;;   ?response - the users response to the menu prompt.*
;;*****************************************************
(defrule possible-causes-of-malfunction-in-combustion-chamber
(troubleshoot-mode engine)
  (menu-level engine possible-causes-1-2-1)
  ?pc <- (possible-cause ?problem malfunction-in-combustion-chamber)
=>
  (retract ?pc)
  (printout t crlf crlf crlf)
  (printout t
  "       You selected MALFUNCTION IN COMBUSTION CHAMBER as a possible" crlf
  "       cause of the problem " ?problem ".  If you change your mind" crlf
  "       and would like to review the previous choices, press 0 now,"  crlf
  "       otherwise select the possible cause of MALFUNCTION IN " crlf
  "       COMBUSTION CHAMBER listed below." crlf crlf
  "               0.) Return to previous menu." crlf
  "               1.) Carbon accumulation." crlf crlf
  "       Choice: ")
    (bind ?response (read))
    (assert (possible-cause ?problem ?response))
    (printout t crlf))
;;
;;*****************************************************
;; This is the sub menu for leakage into combustion    *
;; chamber under problem of excessive oil consumption. *
;; If leakage into combustion chamber is a possible    *
;;    cause of some problem,                           *
;; Then present the possible causes of leakage into    *
;;      combustion chamber.                            *
;;                                                     *
;; Variables:                                          *
;;   ?pc - used for retracting the possible cause.     *
;;   ?problem - the problem selected by the user.      *
;;   ?response - the users response to the menu prompt.*
;;*****************************************************
(defrule possible-causes-of-leakage-into-combustion-chamber
(troubleshoot-mode engine)
  (menu-level engine possible-causes-1-3-1)
  ?pc <- (possible-cause ?problem leakage-into-combustion-chamber)
=>
```

```
 (retract ?pc)
 (printout t crlf crlf crlf)
 (printout t
 "       You selected LEAKAGE INTO COMBUSTION CHAMBER as a possible" crlf
 "       cause of the problem " ?problem ".  If you" crlf
 "       change your mind and would like to review the previous" crlf
 "       choices, press 0 now, otherwise select one of the possible" crlf
 "       causes of LEAKAGE INTO COMBUSTION CHAMBER listed below." crlf crlf
 "              0.) Return to previous menu." crlf
 "              1.) Deformation or abnormal wear of side housing." crlf
 "              2.) Malfunction of rotor (blowholes)." crlf
 "              3.) Scratched or burred rotor land." crlf
 "              4.) Malfunction of oil seal (incorrect angle)." crlf
 "       Choice: ")
  (bind ?response (read))
  (assert (possible-cause ?problem ?response))
  (printout t crlf))
;;
;;*******************************************************
;; This is the sub menu for leakage into coolant        *
;; passages under problem of excessive oil consumption. *
;; If leakage into coolant passages is a possible cause *
;;    of some problem,                                  *
;; Then present the possible causes of leakage into     *
;;      coolant passages.                               *
;;                                                      *
;; Variables:                                           *
;;   ?pc - used for retracting the possible cause.      *
;;   ?problem - the problem selected by the user.       *
;;   ?response - the users response to the menu prompt.*
;;*******************************************************
(defrule possible-causes-of-leakage-into-coolant-passages
(troubleshoot-mode engine)
 (menu-level engine possible-causes-1-3-2)
 ?pc <- (possible-cause ?problem leakage-into-coolant-passages)
=>
 (retract ?pc)
 (printout t crlf crlf crlf)
 (printout t
 "       You selected LEAKAGE INTO COOLANT PASSAGES as a possible" crlf
 "       cause of the problem " ?problem ".  If you" crlf
 "       change your mind and would like to review the previous" crlf
 "       choices, press 0 now, otherwise select one of the possible" crlf
 "       causes of LEAKAGE INTO COOLANT PASSAGES listed below." crlf crlf
 "              0.) Return to previous menu." crlf
 "              1.) Deformed rotor housing." crlf
 "              2.) Malfunction of sealing rubber." crlf
 "       Choice: ")
  (bind ?response (read))
  (assert (possible-cause ?problem ?response))
  (printout t crlf))
;;
;;*******************************************************
;; This is the sub menu for gas seal noise under        *
;; problem of excessive oil consumption.                *
;; If gas seal noise is a possible cause                *
;;    of some problem,                                  *
;; Then present the possible causes of gas seal noise   *
;;                                                      *
;; Variables:                                           *
;;   ?pc - used for retracting the possible cause.      *
;;   ?problem - the problem selected by the user.       *
;;   ?response - the users response to the menu prompt.*
;;*******************************************************
(defrule possible-causes-of-gas-seal-noise
  (troubleshoot-mode engine)
  (menu-level engine possible-causes-1-4-1)
  ?pc <- (possible-cause ?problem gas-seal-noise)
=>
  (retract ?pc)
  (printout t crlf crlf crlf)
  (printout t
  "       You selected GAS SEAL NOISE as a possible cause of the" crlf
  "       problem " ?problem ".  If you change your mind and" crlf
  "       would like to review the previous choices, press 0" crlf
  "       now, otherwise select one of the possible causes of" crlf
  "       GAS SEAL NOISE listed below." crlf crlf
  "             0.) Return to previous menu." crlf
  "             1.) Malfunction of gas seal." crlf
  "             2.) Malfunction of housing." crlf
  "             3.) Malfunction of seal spring." crlf
  "             4.) Malfunction of metering oil pump." crlf
  "       Choice: ")
    (bind ?response (read))
    (assert (possible-cause ?problem ?response))
    (printout t crlf))
;;
;;*******************************************************
;; This is the sub menu for knocking noise under the     *
;; problem of excessive oil consumption.                 *
;; If gas knocking noise is a possible cause             *
;;    of some problem,                                  *
;; Then present the possible causes of knocking noise.  *
;;                                                      *
;; Variables:                                           *
;;   ?pc - used for retracting the possible cause.      *
;;   ?problem - the problem selected by the user.       *
;;   ?response - the users response to the menu prompt.*
;;*******************************************************
(defrule possible-causes-of-knocking-noise
  (troubleshoot-mode engine)
  (menu-level engine possible-causes-1-4-2)
  ?pc <- (possible-cause ?problem knocking-noise)
=>
  (retract ?pc)
  (printout t crlf crlf crlf)
  (printout t
  "       You selected KNOCKING NOISE as a possible cause of the" crlf
  "       problem " ?problem ".  If you change your mind and" crlf
  "       would like to review the previous choices, press 0" crlf
  "       now, otherwise select the possible cause of" crlf
  "       KNOCKING NOISE listed below." crlf crlf
  "               0.) Return to previous menu." crlf
  "               1.) Accumulation of carbon." crlf
  "       Choice: ")
   (bind ?response (read))
   (assert (possible-cause ?problem ?response))
   (printout t crlf))
;;
;;*****************************************************
;; This is the sub menu for hitting noise under         *
;; problem of excessive oil consumption.                *
;; If hitting noise is a possible cause                 *
;;    of some problem,                                  *
;; Then present the possible causes of hitting noise    *
;;                                                      *
;; Variables:                                           *
;;   ?pc - used for retracting the possible cause.      *
;;   ?problem - the problem selected by the user.       *
;;   ?response - the users response to the menu prompt.*
;;*****************************************************
(defrule possible-causes-of-hitting-noise
  (troubleshoot-mode engine)
  (menu-level engine possible-causes-1-4-3)
  ?pc <- (possible-cause ?problem hitting-noise)
=>
  (retract ?pc)
  (printout t crlf crlf crlf)
  (printout t
  "       You selected HITTING NOISE as a possible cause of the" crlf
  "       problem " ?problem ".  If you change your mind and" crlf
  "       would like to review the previous choices, press 0" crlf
  "       now, otherwise select one of the possible causes of" crlf
  "       HITTING NOISE listed below." crlf crlf
  "               0.) Return to previous menu." crlf
  "               1.) Malfunction of main bearing or rotor bearing." crlf
  "               2.) Excessive end play." crlf
  "               3.) Foreign matter in internal gear or stationary" crlf
  "                   gear, or other malfunction." crlf
  "       Choice: ")
   (bind ?response (read))
   (assert (possible-cause ?problem ?response))
   (printout t crlf))
;;
;;*****************************************************
;; This is the sub menu for OTHER under                 *
;; problem of excessive oil consumption.                *
;; If other is a possible cause                         *
;;    of some problem,                                  *
;; Then present the possible causes of other.           *
;;                                                      *
;; Variables:                                           *
;;   ?pc - used for retracting the possible cause.      *
;;   ?problem - the problem selected by the user.       *
;;   ?response - the users response to the menu prompt.*
```

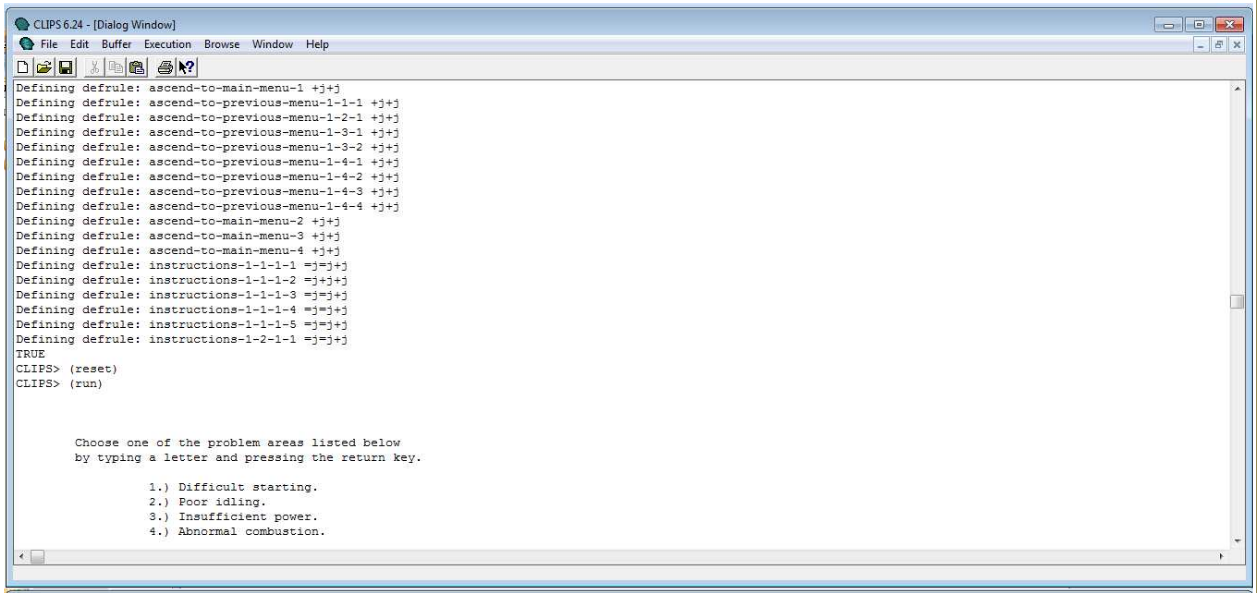
```
;;*****************************************************
(defrule possible-causes-of-other
  (troubleshoot-mode engine)
  (menu-level engine possible-causes-1-4-4)
  ?pc <- (possible-cause ?problem other)
=>
  (retract ?pc)
  (printout t crlf crlf crlf)
  (printout t
  "        You selected OTHER as a possible cause of the problem" crlf
  "        " ?problem ".  If you change your mind and" crlf
  "        would like to review the previous choices, press 0" crlf
  "        now, otherwise select one of the possible causes of" crlf
  "        OTHER listed below." crlf crlf
  "                0.) Return to previous menu." crlf
  "                1.) Malfunction of water pump bearing." crlf
  "                2.) Drive belt tension." crlf
  "                3.) Malfunction of alternator bearing." crlf
  "                4.) Exhaust gas leakage." crlf
  "                5.) Malfunction of fuel system." crlf
  "        Choice: ")
    (bind ?response (read))
    (assert (possible-cause ?problem ?response))
    (printout t crlf))
;;
;;*********************************************************
;; If the current menu level is difficult starting choices, *
;;    and the user selects choice 0,                        *
;; Then re-display the main menu for engine troublshooting. *
;;                                                  *
;; Variables:                                           *
;;   ?ml - for retracting the menu level fact.              *
;;   ?pc - for retracting the possible cause fact.          *
;;*********************************************************
;;
(defrule  ascend-to-main-menu-1
  ?ml <- (menu-level engine possible-causes-1-1)
  ?pc <- (possible-cause ?problem 0)
=>
  (retract ?ml)
  (retract ?pc)
  (assert (menu-level engine main)))
;;
;;
;;*********************************************************
;; If the current menu level is possible-causes-1-1-1,       *
;;    and the user selects choice 0,                        *
;; Then re-display the previous menu.                       *
;;                                                  *
;; Variables:                                           *
;;   ?ml - for retracting the menu level fact.              *
;;   ?pc - for retracting the possible cause fact.          *
;;*********************************************************
(defrule ascend-to-previous-menu-1-1-1
  ?ml <- (menu-level engine possible-causes-1-1-1)
  ?pc <- (possible-cause ?problem 0)
=>
  (retract ?ml)
  (retract ?pc)
  (assert (menu-level engine possible-causes-1-1)))
;;
;;************************************************************
;; If the current menu level is possible-causes-1-2-1,       *
;;    and the user selects choice 0,                         *
;; Then re-display the previous menu.                        *
;;                                                           *
;; Variables:                                                *
;;   ?ml - for retracting the menu level fact.               *
;;   ?pc - for retracting the possible cause fact.           *
;;************************************************************
(defrule ascend-to-previous-menu-1-2-1
  ?ml <- (menu-level engine possible-causes-1-2-1)
  ?pc <- (possible-cause ?problem 0)
=>
  (retract ?ml)
  (retract ?pc)
  (assert (menu-level engine possible-causes-1-2)))
;;
;;************************************************************
;; If the current menu level is possible-causes-1-3-1,       *
;;    and the user selects choice 0,                         *
;; Then re-display the previous menu.                        *
;;                                                           *
;; Variables:                                                *
;;   ?ml - for retracting the menu level fact.               *
;;   ?pc - for retracting the possible cause fact.           *
;;************************************************************
(defrule ascend-to-previous-menu-1-3-1
  ?ml <- (menu-level engine possible-causes-1-3-1)
  ?pc <- (possible-cause ?problem 0)
=>
  (retract ?ml)
  (retract ?pc)
  (assert (menu-level engine possible-causes-1-3)))
;;
;;************************************************************
;; If the current menu level is possible-causes-1-3-2,       *
;;    and the user selects choice 0,                         *
;; Then re-display the previous menu.                        *
;;                                                           *
;; Variables:                                                *
;;   ?ml - for retracting the menu level fact.               *
;;   ?pc - for retracting the possible cause fact.           *
;;************************************************************
(defrule ascend-to-previous-menu-1-3-2
  ?ml <- (menu-level engine possible-causes-1-3-2)
  ?pc <- (possible-cause ?problem 0)
=>
  (retract ?ml)
  (retract ?pc)
  (assert (menu-level engine possible-causes-1-3)))
;;
;;***********************************************************
;; If the current menu level is possible-causes-1-4-1,       *
;;    and the user selects choice 0,                         *
;; Then re-display the previous menu.                        *
;;                                                           *
;; Variables:                                                *
;;   ?ml - for retracting the menu level fact.               *
;;   ?pc - for retracting the possible cause fact.           *
;;***********************************************************
(defrule ascend-to-previous-menu-1-4-1
  ?ml <- (menu-level engine possible-causes-1-4-1)
  ?pc <- (possible-cause ?problem 0)
=>
  (retract ?ml)
  (retract ?pc)
  (assert (menu-level engine possible-causes-1-4)))
;;
;;***********************************************************
;; If the current menu level is possible-causes-1-4-2,       *
;;    and the user selects choice 0,                         *
;; Then re-display the previous menu.                        *
;;                                                           *
;; Variables:                                                *
;;   ?ml - for retracting the menu level fact.               *
;;   ?pc - for retracting the possible cause fact.           *
;;***********************************************************
(defrule ascend-to-previous-menu-1-4-2
  ?ml <- (menu-level engine possible-causes-1-4-2)
  ?pc <- (possible-cause ?problem 0)
=>
  (retract ?ml)
  (retract ?pc)
  (assert (menu-level engine possible-causes-1-4)))
;;
;;***********************************************************
;; If the current menu level is possible-causes-1-4-3,       *
;;    and the user selects choice 0,                         *
;; Then re-display the previous menu.                        *
;;                                                           *
;; Variables:                                                *
;;   ?ml - for retracting the menu level fact.               *
;;   ?pc - for retracting the possible cause fact.           *
;;***********************************************************
(defrule ascend-to-previous-menu-1-4-3
  ?ml <- (menu-level engine possible-causes-1-4-3)
  ?pc <- (possible-cause ?problem 0)
=>
  (retract ?ml)
  (retract ?pc)
  (assert (menu-level engine possible-causes-1-4)))
;;
;;***********************************************************
;; If the current menu level is possible-causes-1-4-4,       *
;;    and the user selects choice 0,                       *
;; Then re-display the previous menu.                      *
;;                                              *
;; Variables:                                       *
;;   ?ml - for retracting the menu level fact.              *
;;   ?pc - for retracting the possible cause fact.          *
;;**********************************************************
(defrule ascend-to-previous-menu-1-4-4
  ?ml <- (menu-level engine possible-causes-1-4-4)
  ?pc <- (possible-cause ?problem 0)
=>
  (retract ?ml)
  (retract ?pc)
  (assert (menu-level engine possible-causes-1-4)))
;;
;;**********************************************************
;; If the current menu level is possible-causes-1-2,        *
;;    and the user selects choice 0,                       *
;; Then re-display the previous menu.                      *
;;                                              *
;; Variables:                                       *
;;   ?ml - for retracting the menu level fact.              *
;;   ?pc - for retracting the possible cause fact.          *
;;**********************************************************
(defrule ascend-to-main-menu-2
  ?ml <- (menu-level engine possible-causes-1-2)
  ?pc <- (possible-cause ?problem 0)
=>
  (retract ?ml)
  (retract ?pc)
  (assert (menu-level engine main)))
;;
;;**********************************************************
;; If the current menu level is possible-causes-1-3,        *
;;    and the user selects choice 0,                       *
;; Then re-display the previous menu.                      *
;;                                              *
;; Variables:                                       *
;;   ?ml - for retracting the menu level fact.              *
;;   ?pc - for retracting the possible cause fact.          *
;;**********************************************************
(defrule ascend-to-main-menu-3
  ?ml <- (menu-level engine possible-causes-1-3)
  ?pc <- (possible-cause ?problem 0)
=>
  (retract ?ml)
  (retract ?pc)
  (assert (menu-level engine main)))
;;
;;**********************************************************
;; If the current menu level is possible-causes-1-4,        *
;;    and the user selects choice 0,                       *
;; Then re-display the previous menu.                      *
;;                                              *
;; Variables:                                       *
;;   ?ml - for retracting the menu level fact.              *
;;   ?pc - for retracting the possible cause fact.          *
;;**********************************************************
(defrule ascend-to-main-menu-4
  ?ml <- (menu-level engine possible-causes-1-4)
  ?pc <- (possible-cause ?problem 0)
=>
  (retract ?ml)
  (retract ?pc)
  (assert (menu-level engine main)))
;;
;;**********************************************************
;; If the current menu level is possible-causes-1-1-1,      *
;;    and the user selects choice 1,                        *
;; Then list the appropriate instructions.                  *
;;                                                          *
;; Variables:                                               *
;;   ?ml - for retracting the menu level fact.              *
;;   ?pc - for retracting the possible cause fact.          *
;;   ?response - for binding in the users response.         *
;;**********************************************************
(defrule instructions-1-1-1-1
  (troubleshoot-mode engine)
  ?ml <- (menu-level engine possible-causes-1-1-1)
  ?pc <- (possible-cause ?problem 1)
=>
  (retract ?ml)
  (retract  ?pc)
  (printout t
  "       Check the side housings (front, intermediate and rear" crlf
  "       housings) for warpage.  Replace if warpage exceeds 0.04 mm." crlf
  "       Refer to section 1-42 in service manual for further" crlf
  "       instructions or illustrations." crlf crlf
  "       Press return to view main menu." crlf)
  (bind ?response (readline))
  (assert (menu-level engine main)))
;;
;;**********************************************************
;; If the current menu level is possible-causes-1-1-1,      *
;;    and the user selects choice 2,                        *
;; Then list the appropriate instructions.                  *
;;                                                          *
;; Variables:                                               *
;;   ?ml - for retracting the menu level fact.              *
;;   ?pc - for retracting the possible cause fact.          *
;;   ?response - for binding in the users response.         *
;;**********************************************************
(defrule instructions-1-1-1-2
  (troubleshoot-mode engine)
  ?ml <- (menu-level engine ?causes)
  ?pc <- (possible-cause ?problem 2)
=>
  (retract ?ml)
  (retract  ?pc)
  (printout t
 "      1.) Check the chromium plated surface on the rotor housing for" crlf
 "          scoring, flaking or any other damage.  Replace if necessary." crlf
 "      2.) See section 1-45 in service manual for measuring rotor." crlf
 "          housing width." crlf crlf
 "      Press return to view main menu." crlf)
 (bind ?response (readline))
 (assert (menu-level engine main)))
;;
;;*************************************************************
;; If the current menu level is possible-causes-1-1-1,       *
;;    and the user selects choice 3,                         *
;; Then list the appropriate instructions.                   *
;;                                                           *
;; Variables:                                                *
;;   ?ml - for retracting the menu level fact.               *
;;   ?pc - for retracting the possible cause fact.           *
;;   ?response - for binding in the users response.          *
;;*************************************************************
(defrule instructions-1-1-1-3
 (troubleshoot-mode engine)
 ?ml <- (menu-level engine ?causes)
 ?pc <- (possible-cause ?problem 3)
=>
 (retract ?ml)
 (retract  ?pc)
 (printout t
 "      1.) Carefully inspect the rotor and replace it if it is" crlf
 "          severely worn or damaged." crlf
 "      2.) Check the internal gear for cracked, scored, worn or" crlf
 "          chipped teeth." crlf
 "      3.) Check the clearance between the side housing and rotor." crlf
 "          See section 1-45 in repair manual for illustration." crlf crlf
 "      Press return to view main menu." crlf)
 (bind ?response (readline))
 (assert (menu-level engine main)))
;;
;;*************************************************************
;; If the current menu level is possible-causes-1-1-1,       *
;;    and the user selects choice 4,                         *
;; Then list the appropriate instructions.                   *
;;                                                           *
;; Variables:                                                *
;;   ?ml - for retracting the menu level fact.               *
;;   ?pc - for retracting the possible cause fact.           *
;;   ?response - for binding in the users response.          *
;;*************************************************************
(defrule instructions-1-1-1-4
 (troubleshoot-mode engine)
 ?ml <- (menu-level engine ?causes)
 ?pc <- (possible-cause ?problem 4)
=>
 (retract ?ml)
 (retract  ?pc)
 (printout t
 "      1.) Inspect the oil seal for wear or damage." crlf
```

```
  "        Replace if necessary." crlf
  "     2.) Check the oil seal lip width." crlf
  "        Lip width should not exceed 0.5 mm." crlf
  "     3.) Check the oil seals for free vertical movement." crlf
  "        See section 1-47 in repair manual for illustration." crlf crlf
  "     Press return to view main menu." crlf)
  (bind ?response (readline))
  (assert (menu-level engine main)))
;;
;;*************************************************************
;; If the current menu level is possible-causes-1-1-1,       *
;;    and the user selects choice 5,                         *
;; Then list the appropriate instructions.                   *
;;                                                           *
;; Variables:                                                *
;;   ?ml - for retracting the menu level fact.               *
;;   ?pc - for retracting the possible cause fact.           *
;;   ?response - for binding in the users response.          *
;;*************************************************************
(defrule instructions-1-1-1-5
  (troubleshoot-mode engine)
  ?ml <- (menu-level engine ?causes)
  ?pc <- (possible-cause ?problem 5)
=>
  (retract ?ml)
  (retract  ?pc)
  (printout t
  "        Inspect the oil seal springs to see that they are properly" crlf
  "        seated in their respective grooves." crlf
  "        See section 1-47 in repair manual for illustration." crlf crlf
  "     Press return to view main menu." crlf)
  (bind ?response (readline))
  (assert (menu-level engine main)))
;;
;;*************************************************************
;; If the current menu level is possible-causes-1-2-1,       *
;;    and the user selects choice 1,                         *
;; Then list the appropriate instructions.                   *
;;                                                           *
;; Variables:                                                *
;;   ?ml - for retracting the menu level fact.               *
;;   ?pc - for retracting the possible cause fact.           *
;;   ?response - for binding in the users response.          *
;;*************************************************************
(defrule instructions-1-2-1-1
  (troubleshoot-mode engine)
  ?ml <- (menu-level engine possible-causes-1-2-1)
  ?pc <- (possible-cause ?problem 1)
=>
  (retract ?ml)
  (retract  ?pc)
  (printout t
  "        Side Housings (front, intermediate and rear housings)" crlf
  "        1.) Remove the sealing agent from the housing surface" crlf
  "           with a cloth or a brush soaked in solvent or thinner." crlf
  "        2.) Remove all carbon on the rotor chamber surface with" crlf
  "            extra-fine emery paper." crlf crlf
  "        Rotor Housing" crlf
  "        Note" crlf
  "        Before cleaning, check for traces of gas or water leakage" crlf
  "        along the inner margin of the rotor housings." crlf crlf
  "        1.) Remove all carbon from the inner surface of the rotor" crlf
  "            housing by wiping with a cloth soaked in solvent or" crlf
  "            thinner." crlf
  "        2.) Remove all deposits and rust from the cooling water" crlf
  "            passages on the housing." crlf
  "        3.) Remove the sealing agent from the housing with a cloth" crlf
  "            or brush soaked in solvent or thinner." crlf crlf
  "        Press return to continue printing instructions." crlf)
  (bind ?key (readline))
  (printout t
  "        Rotor" crlf
  "        1.) Remove the carbon from the rotor with a carbon remover" crlf
  "            or emery paper." crlf crlf
  "        Caution" crlf
  "        a) Do not use emery paper on the groove of the apex seal" crlf
  "           or the side seal." crlf
  "        b) Take care not to damage the soft material coating on the" crlf
  "           side surfaces." crlf crlf
  "        2.) Remove the carbon in each groove." crlf
  "        3.) Wash the rotor with a cleaning solution." crlf crlf
  "     Press return to view main menu." crlf)
  (bind ?response (readline))
  (assert (menu-level engine main)))
```

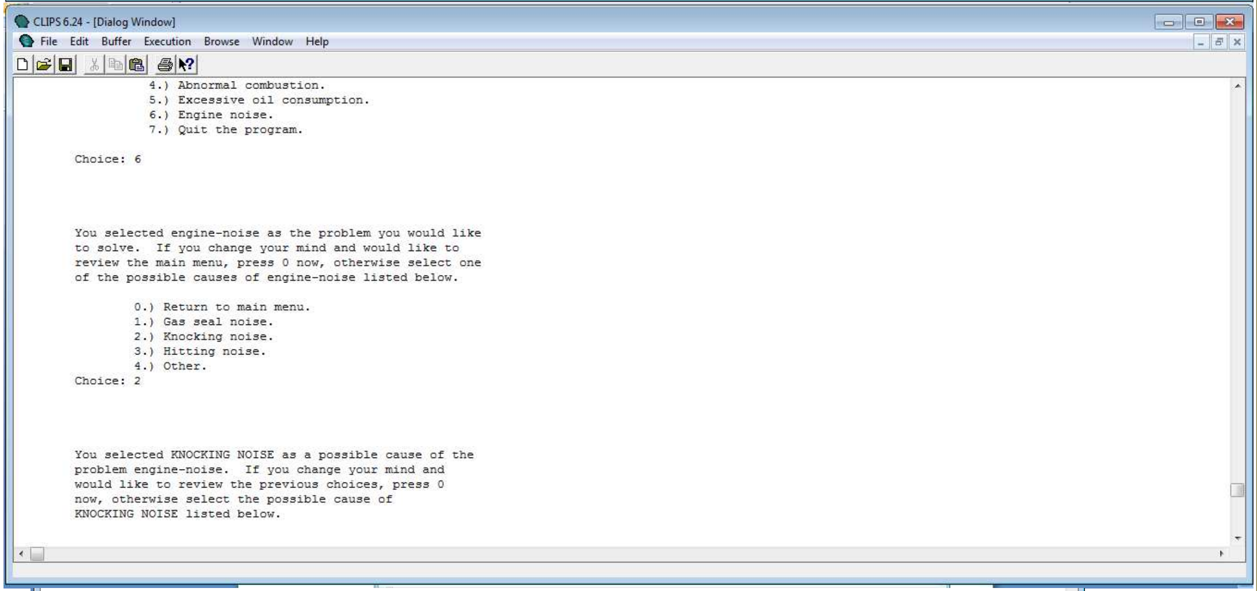
Output:


CLIPS 6.24 - [Dialog Window]
File Edit Buffer Execution Browse Window Help

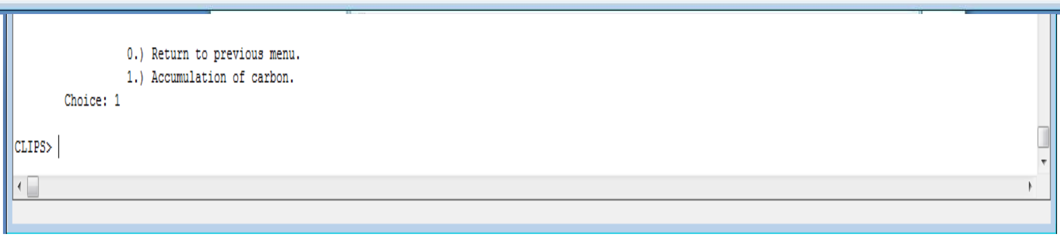
```
         CLIPS (V6.24 06/15/06)
CLIPS> (load "F:/sanket/ME(CSE)/ESD/Examples/rx7.clp")
Defining deffacts: init
Defining defrule: main-menu +j+j
Defining defrule: user-quits =j+j
Defining defrule: difficult-starting =j+j
Defining defrule: poor-idling =j+j
Defining defrule: insufficient-power =j+j
Defining defrule: abnormal-combustion =j+j
Defining defrule: excessive-oil-consumption =j+j
Defining defrule: engine-noise =j+j
Defining defrule: possible-causes-1-1 =j+j+j
Defining defrule: possible-causes-1-2 =j+j+j
Defining defrule: possible-causes-1-3 =j+j+j
Defining defrule: possible-causes-1-4 =j+j+j
Defining defrule: numeric-to-text-insufficient-compression-1-1 =j=j+j
Defining defrule: numeric-to-text-malfunction-in-combustion-chamber-1-2 =j=j+j
Defining defrule: numeric-to-text-leakage-into-combustion-chamber-1-3 =j=j+j
Defining defrule: numeric-to-text-leakage-into-coolant-passages-1-3 =j=j+j
Defining defrule: numeric-to-text-gas-seal-noise-1-4 =j=j+j
Defining defrule: numeric-to-text-knocking-noise-1-4 =j=j+j
Defining defrule: numeric-to-text-hitting-noise-1-4 =j=j+j
Defining defrule: numeric-to-text-other-1-4 =j=j+j
Defining defrule: possible-causes-of-insufficient-compression =j+j+j
Defining defrule: possible-causes-of-malfunction-in-combustion-chamber =j+j+j
Defining defrule: possible-causes-of-leakage-into-combustion-chamber =j+j+j
Defining defrule: possible-causes-of-leakage-into-coolant-passages =j+j+j
Defining defrule: possible-causes-of-gas-seal-noise =j+j+j
Defining defrule: possible-causes-of-knocking-noise =j+j+j
Defining defrule: possible-causes-of-hitting-noise =j+j+j
Defining defrule: possible-causes-of-other =j+j+j
```


CLIPS 6.24 - [Dialog Window]
File Edit Buffer Execution Browse Window Help

```
Defining defrule: ascend-to-main-menu-1 +j+j
Defining defrule: ascend-to-previous-menu-1-1-1 +j+j
Defining defrule: ascend-to-previous-menu-1-2-1 +j+j
Defining defrule: ascend-to-previous-menu-1-3-1 +j+j
Defining defrule: ascend-to-previous-menu-1-3-2 +j+j
Defining defrule: ascend-to-previous-menu-1-4-1 +j+j
Defining defrule: ascend-to-previous-menu-1-4-2 +j+j
Defining defrule: ascend-to-previous-menu-1-4-3 +j+j
Defining defrule: ascend-to-previous-menu-1-4-4 +j+j
Defining defrule: ascend-to-main-menu-2 +j+j
Defining defrule: ascend-to-main-menu-3 +j+j
Defining defrule: ascend-to-main-menu-4 +j+j
Defining defrule: instructions-1-1-1-1 =j=j+j
Defining defrule: instructions-1-1-1-2 =j+j+j
Defining defrule: instructions-1-1-1-3 =j=j+j
Defining defrule: instructions-1-1-1-4 =j=j+j
Defining defrule: instructions-1-1-1-5 =j=j+j
Defining defrule: instructions-1-2-1-1 =j=j+j
TRUE
CLIPS> (reset)
CLIPS> (run)



        Choose one of the problem areas listed below
        by typing a letter and pressing the return key.

                  1.) Difficult starting.
                  2.) Poor idling.
                  3.) Insufficient power.
                  4.) Abnormal combustion.
```


CLIPS 6.24 - [Dialog Window]
File Edit Buffer Execution Browse Window Help

```
                  4.) Abnormal combustion.
                  5.) Excessive oil consumption.
                  6.) Engine noise.
                  7.) Quit the program.

        Choice: 6



        You selected engine-noise as the problem you would like
        to solve.  If you change your mind and would like to
        review the main menu, press 0 now, otherwise select one
        of the possible causes of engine-noise listed below.

                0.) Return to main menu.
                1.) Gas seal noise.
                2.) Knocking noise.
                3.) Hitting noise.
                4.) Other.
        Choice: 2



        You selected KNOCKING NOISE as a possible cause of the
        problem engine-noise.  If you change your mind and
        would like to review the previous choices, press 0
        now, otherwise select the possible cause of
        KNOCKING NOISE listed below.
```



```
                0.) Return to previous menu.
                1.) Accumulation of carbon.
        Choice: 1

CLIPS> |
```

Conclusion: We have designed knowledge based expert system for engine which is used to solve the various problem related to engine and provide best solution by acquiring new knowledge and refining previous knowledge.

PRACTICAL NO.7

Aim: Design a program based on Heuristic Classification.

Theory:

- What do you mean by Heuristic classification?
  Heuristic classification is a non-hierarchical association between data & category requiring intermediate inferences,possibly concepts in another taxonomy.There are three basic steps in heuristic classification are data abstraction,matching data abstraction to solution abstraction,solution refinement.

- What is the need of this method for problem solving?
  Classification is a problem common to many domains foe eg,in botony of zoology ,experts are interested in placing new species in a taxonomy of existing plant or animal types.The class involved usually have a hierarchical organization in which sub classes passes the decriminating features of superclasses & classes which one siblings in the hierarchy are mutually exclusive of some set of features.

- In which areas we can use heuristic classification.
  Heuristic classification is used indiverse areas of science ,engineering ,business & medicine.
  Eg: MYCIN,SACON program,MARC

- Identify and analyse the problem.
  It is a CLIPS-based expert system leads the user through a series of questions related to wine & food preference.The result is a list of candidate wine & which associated confidence factors assigned by the program.

- Implement problem definition

```
(defmodule MAIN (export ?ALL))

;;****************
;;* DEFFUNCTIONS *
;;****************

(deffunction MAIN::ask-question (?question ?allowed-values)
   (printout t ?question)
   (bind ?answer (read))
   (if (lexemep ?answer) then (bind ?answer (lowcase ?answer)))
   (while (not (member ?answer ?allowed-values)) do
      (printout t ?question)
      (bind ?answer (read))
      (if (lexemep ?answer) then (bind ?answer (lowcase ?answer))))
   ?answer)

;;*****************
```

```
;;* INITIAL STATE *
;;*****************

(deftemplate MAIN::attribute
   (slot name)
   (slot value)
   (slot certainty (default 100.0)))

(defrule MAIN::start
  (declare (salience 10000))
  =>
  (set-fact-duplication TRUE)
  (focus QUESTIONS CHOOSE-QUALITIES WINES PRINT-RESULTS))

(defrule MAIN::combine-certainties ""
  (declare (salience 100)
           (auto-focus TRUE))
  ?rem1 <- (attribute (name ?rel) (value ?val) (certainty ?per1))
  ?rem2 <- (attribute (name ?rel) (value ?val) (certainty ?per2))
  (test (neq ?rem1 ?rem2))
  =>
  (retract ?rem1)
  (modify ?rem2 (certainty (/ (- (* 100 (+ ?per1 ?per2)) (* ?per1 ?per2)) 100))))

;;******************
;;* QUESTION RULES *
;;******************

(defmodule QUESTIONS (import MAIN ?ALL) (export ?ALL))

(deftemplate QUESTIONS::question
   (slot attribute (default ?NONE))
   (slot the-question (default ?NONE))
   (multislot valid-answers (default ?NONE))
   (slot already-asked (default FALSE))
   (multislot precursors (default ?DERIVE)))

(defrule  QUESTIONS::ask-a-question
   ?f <- (question (already-asked FALSE)
                   (precursors)
                   (the-question ?the-question)
                   (attribute ?the-attribute)
                   (valid-answers  $?valid-answers))
   =>
   (modify ?f (already-asked TRUE))
   (assert (attribute (name ?the-attribute)
                      (value (ask-question ?the-question ?valid-answers)))))
```

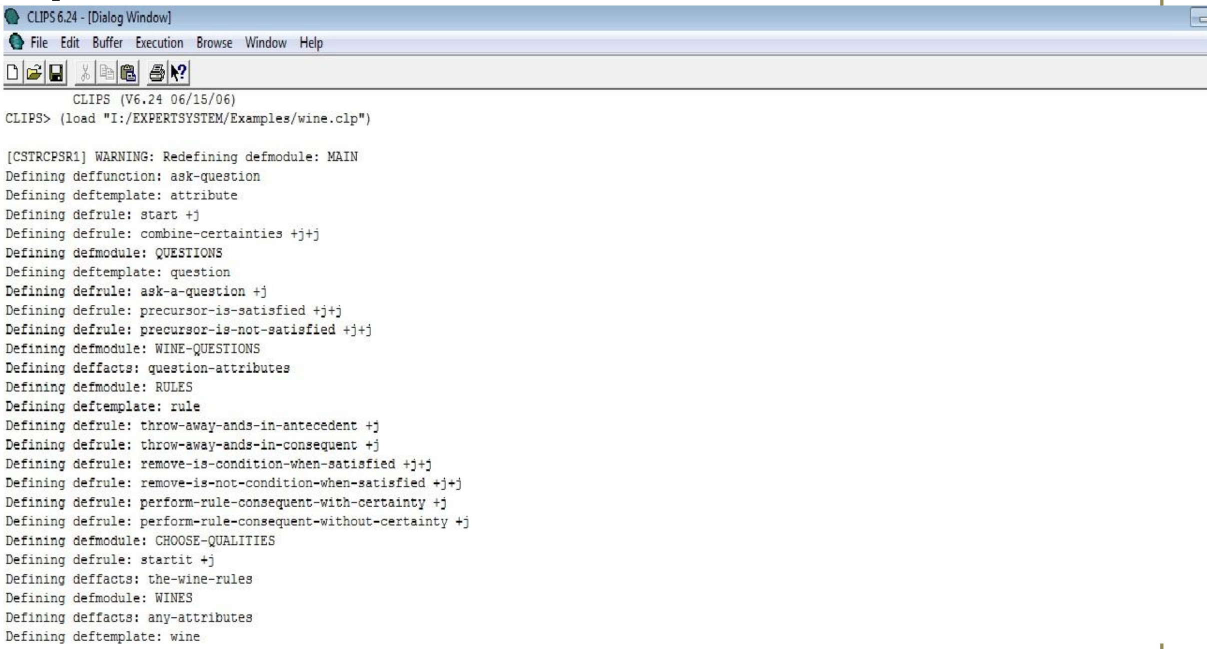
```
(defrule  QUESTIONS::precursor-is-satisfied
   ?f <- (question (already-asked FALSE)
                   (precursors ?name is ?value $?rest))
         (attribute (name ?name) (value ?value))
   =>
   (if (eq (nth 1 ?rest) and)
    then (modify ?f (precursors (rest$ ?rest)))
    else (modify ?f (precursors ?rest))))

(defrule  QUESTIONS::precursor-is-not-satisfied
   ?f <- (question (already-asked FALSE)
                   (precursors ?name is-not ?value $?rest))
         (attribute (name ?name) (value ~?value))
   =>
   (if (eq (nth 1 ?rest) and)
    then (modify ?f (precursors (rest$ ?rest)))
    else (modify ?f (precursors ?rest))))

;;*******************
;;* WINEX QUESTIONS *
;;*******************

(defmodule WINE-QUESTIONS (import QUESTIONS ?ALL))

(deffacts WINE-QUESTIONS::question-attributes
  (question (attribute main-component)
            (the-question "Is the main component of the meal meat, fish, or poultry? ")
            (valid-answers meat fish poultry unknown))
  (question (attribute has-turkey)
            (precursors main-component is turkey)
            (the-question "Does the meal have turkey in it? ")
            (valid-answers yes no unknown))
  (question (attribute has-sauce)
            (the-question "Does the meal have a sauce on it? ")
            (valid-answers yes no unknown))
  (question (attribute sauce)
            (precursors has-sauce is yes)
            (the-question "Is the sauce for the meal spicy, sweet, cream, or tomato? ")
            (valid-answers sauce spicy sweet cream tomato unknown))
  (question (attribute tastiness)
            (the-question "Is the flavor of the meal delicate, average, or strong? ")
            (valid-answers delicate average strong unknown))
  (question (attribute preferred-body)
            (the-question "Do you generally prefer light, medium, or full bodied wines? ")
            (valid-answers light medium full unknown))
  (question (attribute preferred-color)
            (the-question "Do you generally prefer red or white wines? ")
            (valid-answers red white unknown))
  (question (attribute preferred-sweetness)
         (the-question "Do you generally prefer dry, medium, or sweet wines? ")
         (valid-answers dry medium sweet unknown)))

;;******************
;; The RULES module
;;******************

(defmodule RULES (import MAIN ?ALL) (export ?ALL))

(deftemplate RULES::rule
  (slot certainty (default 100.0))
  (multislot if)
  (multislot then))

(defrule  RULES::throw-away-ands-in-antecedent
  ?f <- (rule (if and $?rest))
  =>
  (modify ?f (if ?rest)))

(defrule  RULES::throw-away-ands-in-consequent
  ?f <- (rule (then and $?rest))
  =>
  (modify ?f (then ?rest)))

(defrule  RULES::remove-is-condition-when-satisfied
  ?f <- (rule (certainty ?c1)
            (if ?attribute is ?value $?rest))
  (attribute (name ?attribute)
           (value ?value)
           (certainty ?c2))
  =>
  (modify ?f (certainty (min ?c1 ?c2)) (if ?rest)))

(defrule  RULES::remove-is-not-condition-when-satisfied
  ?f <- (rule (certainty ?c1)
            (if ?attribute is-not ?value $?rest))
  (attribute (name ?attribute) (value ~?value) (certainty ?c2))
  =>
  (modify ?f (certainty (min ?c1 ?c2)) (if ?rest)))

(defrule  RULES::perform-rule-consequent-with-certainty
  ?f <- (rule (certainty ?c1)
            (if)
            (then ?attribute is ?value with certainty ?c2 $?rest))
  =>
  (modify ?f (then ?rest))
  (assert (attribute (name ?attribute)
                    (value ?value)
                    (certainty (/ (* ?c1 ?c2) 100)))))

(defrule  RULES::perform-rule-consequent-without-certainty
  ?f <- (rule (certainty ?c1)
            (if)
            (then ?attribute is ?value $?rest))
  (test (or (eq (length$ ?rest) 0)
          (neq (nth 1 ?rest) with)))
  =>
  (modify ?f (then ?rest))
  (assert (attribute (name ?attribute) (value ?value) (certainty ?c1))))

;;*********************************
;;* CHOOSE WINE QUALITIES RULES *
;;*********************************

(defmodule CHOOSE-QUALITIES (import RULES ?ALL)
                   (import QUESTIONS ?ALL)
                   (import MAIN ?ALL))

(defrule CHOOSE-QUALITIES::startit => (focus RULES))

(deffacts the-wine-rules

  ; Rules for picking the best body

  (rule (if has-sauce is yes and
          sauce is spicy)
      (then best-body is full))

  (rule (if tastiness is delicate)
      (then best-body is light))

  (rule (if tastiness is average)
      (then best-body is light with certainty 30 and
            best-body is medium with certainty 60 and
            best-body is full with certainty 30))

  (rule (if tastiness is strong)
      (then best-body is medium with certainty 40 and
            best-body is full with certainty 80))

  (rule (if has-sauce is yes and
          sauce is cream)
      (then best-body is medium with certainty 40 and
            best-body is full with certainty 60))
```

```
(rule (if preferred-body is full)
    (then best-body is full with certainty 40))

(rule (if preferred-body is medium)
    (then best-body is medium with certainty 40))

(rule (if preferred-body is light)
    (then best-body is light with certainty 40))

(rule (if preferred-body is light and
        best-body is full)
    (then best-body is medium))

(rule (if preferred-body is full and
        best-body is light)
    (then best-body is medium))

(rule (if preferred-body is unknown)
    (then best-body is light with certainty 20 and
        best-body is medium with certainty 20 and
        best-body is full with certainty 20))

; Rules for picking the best color

(rule (if main-component is meat)
    (then best-color is red with certainty 90))

(rule (if main-component is poultry and
        has-turkey is no)
    (then best-color is white with certainty 90 and
        best-color is red with certainty 30))

(rule (if main-component is poultry and
        has-turkey is yes)
    (then best-color is red with certainty 80 and
        best-color is white with certainty 50))

(rule (if main-component is fish)
    (then best-color is white))

(rule (if main-component is-not fish and
        has-sauce is yes and
        sauce is tomato)
    (then best-color is red))

(rule (if has-sauce is yes and
        sauce is cream)
    (then best-color is white with certainty 40))
```

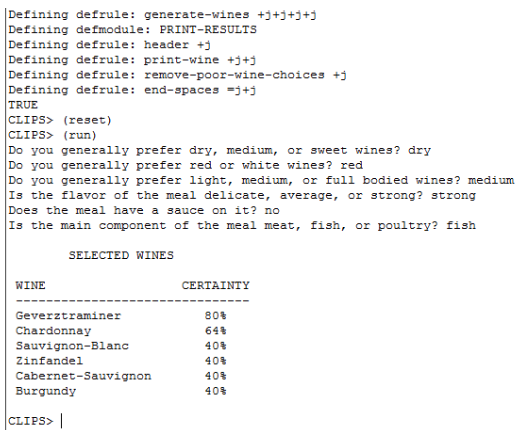
```
 (rule (if preferred-color is red)
       (then best-color is red with certainty 40))

 (rule (if preferred-color is white)
       (then best-color is white with certainty 40))

 (rule (if preferred-color is unknown)
       (then best-color is red with certainty 20 and
             best-color is white with certainty 20))

 ; Rules for picking the best sweetness

 (rule (if has-sauce is yes and
           sauce is sweet)
       (then best-sweetness is sweet with certainty 90 and
             best-sweetness is medium with certainty 40))

 (rule (if preferred-sweetness is dry)
       (then best-sweetness is dry with certainty 40))

 (rule (if preferred-sweetness is medium)
       (then best-sweetness is medium with certainty 40))

 (rule (if preferred-sweetness is sweet)
       (then best-sweetness is sweet with certainty 40))

 (rule (if best-sweetness is sweet and
           preferred-sweetness is dry)
       (then best-sweetness is medium))

 (rule (if best-sweetness is dry and
           preferred-sweetness is sweet)
       (then best-sweetness is medium))

 (rule (if preferred-sweetness is unknown)
       (then best-sweetness is dry with certainty 20 and
             best-sweetness is medium with certainty 20 and
             best-sweetness is sweet with certainty 20))

)

;;************************
;;* WINE SELECTION RULES *
;;************************

(defmodule WINES (import MAIN ?ALL))
```

```
(deffacts any-attributes
  (attribute (name best-color) (value any))
  (attribute (name best-body) (value any))
  (attribute (name best-sweetness) (value any)))

(deftemplate WINES::wine
  (slot name (default ?NONE))
  (multislot color (default any))
  (multislot body (default any))
  (multislot sweetness (default any)))

(deffacts  WINES::the-wine-list
  (wine (name Gamay) (color red) (body medium) (sweetness medium sweet))
  (wine (name Chablis) (color white) (body light) (sweetness dry))
  (wine (name Sauvignon-Blanc) (color white) (body medium) (sweetness dry))
  (wine (name Chardonnay) (color white) (body medium full) (sweetness medium
dry))
  (wine (name Soave) (color white) (body light) (sweetness medium dry))
  (wine (name Riesling) (color white) (body light medium) (sweetness medium
sweet))
  (wine (name Geverztraminer) (color white) (body full))
  (wine (name Chenin-Blanc) (color white) (body light) (sweetness medium sweet))
  (wine (name Valpolicella) (color red) (body light))
  (wine (name Cabernet-Sauvignon) (color red) (sweetness dry medium))
  (wine (name Zinfandel) (color red) (sweetness dry medium))
  (wine (name Pinot-Noir) (color red) (body medium) (sweetness medium))
  (wine (name Burgundy) (color red) (body full))
  (wine (name Zinfandel) (color red) (sweetness dry medium)))

(defrule WINES::generate-wines
  (wine (name ?name)
        (color $? ?c $?)
        (body $? ?b $?)
        (sweetness $? ?s $?))
  (attribute (name best-color) (value ?c) (certainty ?certainty-1))
  (attribute (name best-body) (value ?b) (certainty ?certainty-2))
  (attribute (name best-sweetness) (value ?s) (certainty ?certainty-3))
  =>
  (assert (attribute (name wine) (value ?name)
                     (certainty (min ?certainty-1 ?certainty-2 ?certainty-3)))))

;;*****************************
;;* PRINT SELECTED WINE RULES *
;;*****************************

(defmodule PRINT-RESULTS (import MAIN ?ALL))

(defrule PRINT-RESULTS::header ""
   (declare (salience 10))
   =>
   (printout t t)
   (printout t "        SELECTED WINES" t t)
   (printout t " WINE                  CERTAINTY" t)
   (printout t " -----------------------------------" t)
   (assert (phase print-wines)))

(defrule PRINT-RESULTS::print-wine ""
  ?rem <- (attribute (name wine) (value ?name) (certainty ?per))
  (not (attribute (name wine) (certainty ?per1&:(> ?per1 ?per))))
  =>
  (retract ?rem)
  (format t " %-24s %2d%%%n" ?name ?per))

(defrule  PRINT-RESULTS::remove-poor-wine-choices  ""
  ?rem <- (attribute (name wine) (certainty ?per&:(< ?per 20)))
  =>
  (retract ?rem))

(defrule PRINT-RESULTS::end-spaces ""
   (not (attribute (name wine)))
   =>
   (printout t t))
```

Output:


CLIPS 6.24 - [Dialog Window]
File Edit Buffer Execution Browse Window Help

```
         CLIPS (V6.24 06/15/06)
CLIPS> (load "I:/EXPERTSYSTEM/Examples/wine.clp")

[CSTRCPSR1] WARNING: Redefining defmodule: MAIN
Defining deffunction: ask-question
Defining deftemplate: attribute
Defining defrule: start +j
Defining defrule: combine-certainties +j+j
Defining defmodule: QUESTIONS
Defining deftemplate: question
Defining defrule: ask-a-question +j
Defining defrule: precursor-is-satisfied +j+j
Defining defrule: precursor-is-not-satisfied +j+j
Defining defmodule: WINE-QUESTIONS
Defining deffacts: question-attributes
Defining defmodule: RULES
Defining deftemplate: rule
Defining defrule: throw-away-ands-in-antecedent +j
Defining defrule: throw-away-ands-in-consequent +j
Defining defrule: remove-is-condition-when-satisfied +j+j
Defining defrule: remove-is-not-condition-when-satisfied +j+j
Defining defrule: perform-rule-consequent-with-certainty +j
Defining defrule: perform-rule-consequent-without-certainty +j
Defining defmodule: CHOOSE-QUALITIES
Defining defrule: startit +j
Defining deffacts: the-wine-rules
Defining defmodule: WINES
Defining deffacts: any-attributes
Defining deftemplate: wine
```



```
Defining defrule: generate-wines +j+j+j+j
Defining defmodule: PRINT-RESULTS
Defining defrule: header +j
Defining defrule: print-wine +j+j
Defining defrule: remove-poor-wine-choices +j
Defining defrule: end-spaces =j+j
TRUE
CLIPS> (reset)
CLIPS> (run)
Do you generally prefer dry, medium, or sweet wines? dry
Do you generally prefer red or white wines? red
Do you generally prefer light, medium, or full bodied wines? medium
Is the flavor of the meal delicate, average, or strong? strong
Does the meal have a sauce on it? no
Is the main component of the meal meat, fish, or poultry? fish

        SELECTED WINES

 WINE                     CERTAINTY
 -------------------------------
 Geverztraminer              80%
 Chardonnay                  64%
 Sauvignon-Blanc             40%
 Zinfandel                   40%
 Cabernet-Sauvignon          40%
 Burgundy                    40%

CLIPS> |
```

Conclusion: In this way ,we have studied the concept of heuristic classification . In heuristic classification we have designed a wine expert system in which there is a list of candidates wine & which associated confidence factors assigned by the program.

PRACTICAL NO.8

Aim: Design a program demonstrating Non-monotonic justification

Theory:

➢ What is Non-monotonic justification?

In monotonic systems there is no need to check for in consistencies between new statements and the old knowledge. When a proof is made, the basis of the proof need not be remembered, since the old statements never disappear. Non monotonic reasoning is based on default reasoning or "most probabilistic choice". S is assumed to be true as long as there is no evidence to the contrary. For example when we visit a friend's home, we buy biscuits for the children because we believe that most children like biscuits. In this case we do not have information to the contrary.

Non monotonic system are harder to deal with than monotonic systems. This is because when a statement is deleted as "no more valid", other related statements have to be backtracked and they should be either deleted or new proofs have to be found for them. Thus non – monotonic systems require more storage space as well as more processing time than monotonic systems.

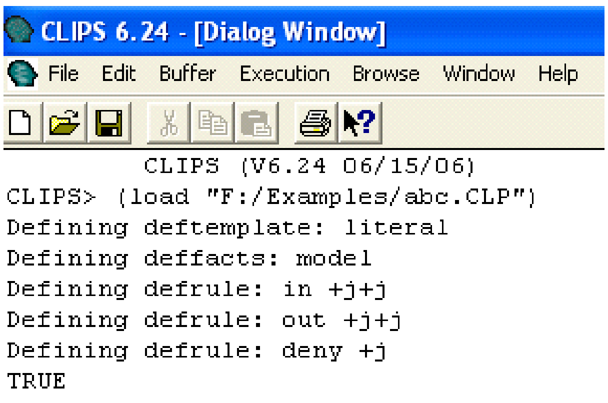
➢ Identify and Analyze the Problem

This CLIPS program implements simple non-monotonic TMS for atomic statements. A literal is an atomic proposition, such as 'P' , whose status is either in or out of the TMS. The in list says what statements have to be in for the statement to be in while the out list says what statements have to be out for the statement to in. As before, empty support lists have default value of -1.

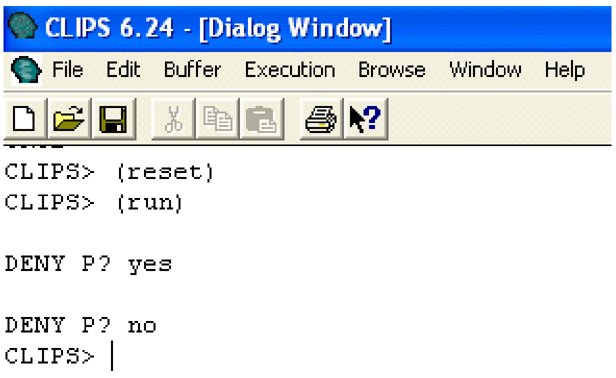
Implement the Problem

```
;;;=======================================================
;;;     Program of Nonmonotonic Justification
;;;
;;;    CLIPS Version 6.0 Example
;;;
;;;    To execute, merely load, reset and run.
;;;=======================================================

(deftemplate literal
        (field id (type INTEGER))
        (field atom (type SYMBOL))
        (field status (type SYMBOL) (default unk))
        (multifield inlist (type INTEGER) (default -1))
        (multifield outlist(type INTEGER) (default -1))
)
(deffacts model
(literal (id 0)(atom P) (status in)(outlist 1))
(literal (id 1) (atom Q) (status in) (outlist 0))
)
(defrule in
(literal (id ?A) (status out))
?F <- (literal (status out) (inlist -1) (outlist ?A))
=>
(modify ?F (status in))
)
(defrule out
(literal (id ?A) (status in))
?F <- (literal (status in) (outlist ?A))
=>
(modify ?F (status out))
)
(defrule deny
(declare (salience -10))
?L <- (literal (atom ?X) (status in) (inlist -1))
=>
(printout t crlf "DENY " ?X "? ")
(bind ?ans (read))
(if (eq ?ans yes) then (modify ?L (status out)))
)
```

Output


CLIPS 6.24 - [Dialog Window]
File Edit Buffer Execution Browse Window Help

```
         CLIPS (V6.24 06/15/06)
CLIPS> (load "F:/Examples/abc.CLP")
Defining deftemplate: literal
Defining deffacts: model
Defining defrule: in +j+j
Defining defrule: out +j+j
Defining defrule: deny +j
TRUE
```


CLIPS 6.24 - [Dialog Window]
File Edit Buffer Execution Browse Window Help

```
CLIPS> (reset)
CLIPS> (run)

DENY P? yes

DENY P? no
CLIPS> |
```

Conclusion: Thus we have studied and performed how to design a program demonstrating Non-monotonic justification.