ЛАБОРАТОРНАЯ РАБОТА №4

ТЕМА: Разработка экспертной системы AutoExpert.

ЦЕЛЬ: Построить экспертную систему AutoExpert при помощи фактов, функций и правил, используя управляющие команды (*if-then-else* и *while*) в среде CLIPS.

ПЛАН ЗАНЯТИЯ:

1. Исходные данные.
2. Выделение сущностей.
3. Сбор информации.
4. Определение диагностических правил.
5. Метод вывода итоговой информации.
6. Запуск и тестирование программы.
7. Контрольные вопросы.

В данной лабораторной работе рассмотрим, как с помощью CLIPS можно создавать экспертные системы. Для начала попробуем создать полноценную, хотя и не очень сложную экспертную систему, проводящую диагностику неисправности мотора автомобиля по внешним признакам, причем созданная нами диагностическая экспертная система должна также предоставлять пользователю соответствующие рекомендации по устранению неисправности.

Для реализации данной экспертной системы будем использовать управляющие команды CLIPS, такие как *if-then-else* и *whil*e. Знания какого-нибудь процедурного языка программирования (C, C++), позволит Вам без особого труда выполнить поставленную задачу.

1. ИСХОДНЫЕ ДАННЫЕ

Разработку любой экспертной системы следует начинать с выделения основных сущностей, имеющих значение при решении конкретной задачи и законов, скорее всего эмпирических, действующих над этими сущностями. В подавляющем большинстве случаев эту информацию получают при помощи эксперта, человека хорошо знающего и давно работающего в этой области. Методы получения информации от эксперта и ее обработка были рассмотрены в лекционной части курса. Для решения нашей конкретной задачи предположим, что в результате бесед с

экспертом в области установления неисправностей и ремонта автомобилей были установлены следующие **эмпирические правила**:

1. Двигатель обычно находится в одном из 3-х состояний: он может работать нормально, работать неудовлетворительно или не заводиться.
2. Если двигатель работает нормально, то это означает, что он нормально вращается, система зажигания и аккумулятор находятся в норме и никакого ремонта не требуется.
3. Если двигатель запускается, но работает ненормально, то это говорит, по крайней мере, о том, что аккумулятор в порядке.
4. Если двигатель не запускается, то нужно узнать, пытается ли он вращаться. Если двигатель вращается, но при этом не заводится, то это может говорить о наличии плохой искры в системе зажигания. Если двигатель даже не пытается заводиться, то это говорит о том, что искры нет в принципе.
5. Если двигатель не заводится, но вращается, нужно проверить наличие топлива. Если топлива нет – то, скорей всего, для ремонта машины нужно просто заправиться.
6. Если двигатель не заводится, нужно также проверить, заряжен ли аккумулятор, если нет, то его следует зарядить.
7. Если двигатель не заводится, и существует вероятность плохой искры в системе зажигания, то необходимо проверить контакты. Контакты могут быть в одном из трех состояний — чистые, опаленные и грязные, в случае опаленных контактов их необходимо заменить, в случае если контакты грязные, их достаточно просто почистить.
8. Если двигатель не заводится, искры нет и аккумулятор заряжен, то нужно проверить катушку зажигания на электрическую проводимость. В случае если ток не проходит через катушку, то ее необходимо заменить. Если катушка зажигания в порядке, значит необходимо заменить распределительные провода.
9. Если двигатель запускается, но при этом ведет себя инертно, не сразу реагирует на подачу топлива, то необходимо прочистить топливную систему.
10. Если двигатель запускается, но происходят перебои с зажиганием, то это говорит о наличии плохой искры в системе зажигания, для устранения данной неисправности необходимо отрегулировать зазоры между контактами.

11. Если двигатель запускается и стучит, то необходимо отрегулировать зажигание.

12. Если двигатель запускается, но не развивает нормальной мощности, то это может говорить об опаленных или загрязненных контактах (см. правило 7).

13. Возможны ситуации, когда состояние двигателя нельзя описать приведенными выше факторами и машине может потребоваться более детальный анализ состояния.

Имея эти данные, приступим к решению поставленной задачи.

2. ВЫДЕЛЕНИЕ СУЩНОСТЕЙ

Из приведенных выше правил можно выделить следующие **сущности**, имеющие значение при решении задачи.

- Во-первых, для решения задачи экспертной системе необходимо знать, в каком состоянии находится машина, диагностика которой производится. Эксперт выделил три возможных состояния: нормальная работа двигателя, двигатель работает неудовлетворительно, не заводится (см. правило 1).
- Во-вторых, большинство приведенных правил помимо состояния двигателя в целом используют понятие состояния вращения двигателя. Согласно этим правилам двигатель может находиться в одном из двух состояний, которые определяются в зависимости от того, способен он вращаться (работать) или нет.
- В-третьих, в некоторых правилах (см. правила 4, 7, 8, 10) используется понятие состояния системы зажигания. Система зажигания может быть в одном из трех состояний: нормальное состояние, не регулярная работа и нерабочее состояние.
- В-четвертых, в правилах 6 и 8 используется понятие — состояние аккумулятора. Аккумулятор может быть в одном из двух состояний: заряженным и разряженным.

Для того чтобы решения данной задачи было более наглядным, мы не будем использовать шаблоны. Для представления в CLIPS всех перечисленных выше данных воспользуемся упорядоченными фактами CLIPS. Исходя из приведенного выше списка, нам могут понадобиться факты, приведенные в примере 1

Пример 1. Факты, описывающие состояние автомобиля и его узлов

```
;Группа фактов, описывающая состояние машины
```

```
working-state engine normal                ;нормальная работа
working-state engine unsatisfactory        ;неудовлетворительная
работа
working-state engine does-not-start        ;не заводится

;Группа фактов, описывающая состояние двигателя
rotation-state engine rotates              ;двигатель вращается
rotation-state engine does-not-rotate      ;двигатель не
вращается

;Группа фактов, описывающая состояние системы зажигания
spark-state engine normal                  ;зажигание в порядке
spark-state engine irregular-spark         ;искра не регулярна
spark-state engine does-not-spark          ;искры нет

;Группа фактов, описывающая состояние системы питания
charge-state battery charged               ;аккумулятор заряжен
charge-state battery dead                  ;аккумулятор разряжен
```

Обратите внимание, что факты, входящие в одну группу (содержат одинаковое первое поле), являются взаимоисключающими, т.е. наличие в системе сразу двух фактов из одной группы лишено смысла.

Из постановки задачи следует, что наша экспертная система должна предоставлять пользователю рекомендации, позволяющие устранить найденную неисправность. Из приведенных выше правил можно выделить следующие рекомендации:

- ❒ добавить топливо (правило 5);
- ❒ зарядить аккумулятор (правило 6);
- ❒ заменить или почистить контакты (правило 7 или правило 12);
- ❒ заменить катушку зажигания или распределительные провода (правило 8);
- ❒ прочистить топливную систему (правило 9);
- ❒ отрегулировать зазоры между контактами (правило 10);
- ❒ отрегулировать зажигание (правило 11).

Необходимо помнить также о двух крайних случаях:

- ❒ ремонт не требуется в принципе;
- ❒ экспертная система не смогла поставить диагноз.

Для представления всех этих рекомендаций будем использовать факты, представленные в примере 2.

Пример 2. Факты, описывающие рекомендации по ремонту автомобиля

```
repair "Add gas."
repair "Charge the battery."
repair "Replace the points."
repair "Clean the points."
repair "Replace the ignition coil."
repair "Repair the distributor lead wire."
repair "Clean the fuel line."
repair "Point gap adjustment."
repair "Timing adjustment."
repair "No repair needed."
repair "Take your car to a mechanic."
```

Все приведенные факты, использующиеся для предоставления пользователю рекомендаций по ремонту, во втором поле содержат текстовое значение с рекомендацией по ремонту.

Обратите внимание, что одни и те же рекомендации могут выводиться как правилом 7, так и правилом 12. Однако состояние машины при этой поломке отличается. Для того чтобы иметь возможность обрабатывать эту ситуацию с помощью одного правила CLIPS, введем еще два дополнительных факта.

Пример 3. Факты, описывающие мощность работы двигателя

```
symptom engine low-output                    ;низкая мощность
symptom engine not-low-output                ;нормальная
мощность
```

Кроме описанных выше фактов системе могут понадобиться факты, описывающие проявления неисправности. Однако в нашей версии экспертной системы таких фактов не будет. Приведенный выше список фактов вполне достаточен для решения поставленной задачи. Приступим к следующему этапу — сбору исходной информации для диагностики.

3. СБОР ИНФОРМАЦИИ

Как упоминалось выше, для работы нашей системы можно заставить пользователя вручную вводить факты, описывающие проявление возникшей неисправности. Однако такой метод имеет ряд серьезных недостатков: пользователь может забыть о каких-нибудь существенных деталях или, наоборот, указать слишком много информации, что может помешать нормальной работе системы. Кроме того, факты, описывающие

проявление неисправности, должны были бы иметь строго определенный формат, и система не смогла бы их обработать в случае ошибки со стороны пользователя.

В нашей экспертной системе мы реализуем правила диагностики, которые в зависимости от той или иной ситуации будут задавать пользователю необходимые вопросы и получать ответ в строго заданной форме. Дальнейшая диагностика будет производиться с учетом предыдущих ответов на вопросы, заданные пользователю. Эти ответы будут формировать описание текущей ситуации с помощью фактов, приведенных выше.

Для реализации подобной архитектуры будет необходимо реализовать функцию, задающую пользователю произвольный вопрос и получающую ответ из заданного набора корректных ответов. В примере 4 приведена одна из возможных реализаций такой функции.

Пример 4. Функция *ask-question*

```
(deffunction 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
)
```

Функция принимает два аргумента: простую переменную *question*, которая содержит текст вопроса, и составную переменную *allowed-values* с набором допустимых ответов. Сразу после своего вызова функция выводит на экран соответствующий вопрос и читает ответ пользователя в переменную *answer*. Если переменная *answer* содержит текст, то она будет принудительно приведена к прописному алфавиту. После этого функция проверяет, является ли полученный ответ одним из заданных корректных

ответов. Если нет, то процесс повторится до получения корректного ответа, иначе функция вернет ответ, введенный пользователем.

Будет также очень полезно определить функцию, задающую пользователю вопрос и допускающий ответ в виде да/нет, т. к. это один из самых распространенных типов вопросов. С учетом реализации функции ***ask-question*** эта функция примет вид, представленный в примере 5.

Пример 5. Функция *yes-or-no-p*

```
(deffunction yes-or-no-p (?question)
              (bind ?response (ask-question ?question yes no
y n))
              (if (or (eq ?response yes) (eq ?response y))
                   then
                        TRUE
                   else
                        FALSE)
)
```

Функция ***yes-or-no-p*** вызывает функцию ***ask-question*** с постоянным набором допустимых ответов: *yes, no, y* и *n*. В случае если пользователь ввел ответ *yes* или *y*, функция возвращает значение TRUE, иначе — FALSE. Обратите внимание, что поскольку функция ***yes-or-no-p*** использует функцию ***ask-question***, то она должна быть определена после нее.

4. ОПРЕДЕЛЕНИЕ ДИАГНОСТИЧЕСКИХ ПРАВИЛ

Для упрощения реализации нашей экспертной системы введем следующее ограничение: за один запуск система может предоставить пользователю только одну рекомендацию по исправлению неисправности. В случае если в машине несколько неисправностей, то систему нужно будет последовательно вызывать несколько раз, удаляя обнаруженную на каждом новом шаге неисправность. Таким образом, одним из образцов всех диагностических правил будет *(not (repair ?))*, гарантирующий, что диагноз еще не поставлен.

Первым реализуем правило, определяющее общее состояние двигателя (см. правило 1).

Пример 6. Правило *determine-engine-state*

```
(defrule determine-engine-state ""
     (not (working-state engine ?))
     (not (repair ?))
```

```
     =>
     (if (yes-or-no-p "Does the engine start (yes/no)? ")
          then
               (if (yes-or-no-p "Does the engine run
normally(yes/no)? ")
               then
                    (assert (working-state engine normal))
               else
                    (assert (working-state engine
unsatisfactory)))
          else
               (assert (working-state engine does-not-
start)))
 )
```

Условный элемент (*not* (*working-state engine ?*)) гарантирует, что общее состояние двигателя еще не определено. Если это так, то пользователю задаются соответствующие вопросы и в систему добавляется факт, описывающий текущее общее состояние двигателя.

Теперь реализуем правило, определяющее, пытается ли двигатель вращаться, в случае если он не заводится.

Пример 7. Правило *determine-rotation-state*

```
 (defrule determine-rotation-state " "
          (working-state engine does-not-start)
          (not (rotation-state engine ?) )
          (not (repair ?))
          =>
          (if (yes-or-no-p "Does the engine rotate (yes/no)?
")
               then
                    (assert (rotation-state engine rotates))
                    (assert (spark-state engine irregular-
spark))
               else
                    (assert (rotation-state engine does-not-
rotate))
                    (assert (spark-state engine does-not-
spark)))
 )
```

Это правило выполняется, в случае если общее состояние двигателя определено и известно, что он не заводится. Кроме того, условный элемент (*not* (*rotation-state engine ?*)) гарантирует, что это правило еще не вызывалось. В зависимости от того или иного ответа пользователя правило добавляет соответствующий набор фактов (см. правило 4).

Далее реализуем довольно простые правила 5 и 6. Выполняемые ими действия вы поймете без дополнительных комментариев.

Пример 8. Правила *determine-gas-level* и *determine-battery-state*

```
(defrule determine-gas-level " "
    (working-state engine does-not-start)
    (rotation-state engine rotates)
    (not (repair ?))
    =>
    (if (not (yes-or-no-p "Does the tank have any gas in it
                                              (yes/no)? "))
        then
            (assert (repair "Add gas.")))
)
(defrule determine-battery-state " "
    (rotation-state engine does-not-rotate)
    (not (charge-state battery ?))
    (not (repair ?))
    =>
    (if (yes-or-no-p "Is the battery charged (yes/no)? ")
        then
            (assert (charge-state battery charged))
        else
            (assert (repair "Charge the battery."))
            (assert (charge-state battery dead)))
)
```

Обратите внимание, что правило *determine-battery-state*, помимо определения возможной неисправности, также применяется для добавления в систему факта, описывающего текущее состояние аккумулятора, который может быть использован другими правилами.

При реализации правила 7 необходимо обратить внимание на то, что рекомендации, предоставляемые этим правилом, подходят для двух в корне отличающихся ситуаций. Во-первых, в случае если двигатель не заводится, и существует вероятность плохой искры в системе зажигания (правило 7). Во-вторых, в случае если двигатель запускается, но не

развивает нормальной мощности (правило 12). Поэтому выполним реализацию этих правил так, как представлено в примере 9.

Пример 9. Правила *determine-low-output* и *determine-point-surface-state*

```
 (defrule determine-low-output " "
     (working-state engine unsatisfactory)
     (not (symptom engine low-output I not-low-output))
     (not (repair ?))
     =>
     (if (yes-or-no-p "Is the output of the engine low
(yes/no)? ")
          then
               (assert (symptom engine low-output))
          else
               (assert (symptom engine not-low-output)))
 )
 (defrule determine-point-surface-state " "
     (or (and (working-state engine does-not-start)
               (spark-state engine irregular-spark))
          (symptom engine low-output))
     (not (repair ?))
     =>
          (bind ?response (ask-question "What is the surface
state
               of the points (normal /burned /contaminated)?"
               normal burned contaminated))
          (if (eq ?response burned)
               then
                    (assert (repair "Replace the points."))
               else
                    (if (eq ?response contaminated)
                    then
                         (assert (repair "Clean the
points."))))
 )
```

Правило *determine-low-output* определяет, имеет ли место низкая мощность двигателя или нет. Правило *determine-point-surface-state* адекватно реагирует на условия, заданные в правилах 7 и 12. Обратите внимание на использование условных элементов ***or*** и ***and***, которые

обеспечивают одинаковое поведение правила в двух абсолютно разных ситуациях. Кроме того, правило *determine-point-surface-state* отличается от приведенных ранее тем, что непосредственно использует функцию ***ask-question***, вместо ***yes-or-no-p***, т.к. в данный момент пользователю задается вопрос, подразумевающий три варианта ответа.

Реализация оставшихся диагностических правил (8—11) также не должна вызвать затруднений.

Пример 10. Оставшиеся диагностические правила

```
 (defrule determine-conductivity-test " "
      (working-state engine does-not-start)
      (spark-state engine does-not-spark)
      (charge-state battery charged)
      (not (repair ?))
      =>
      (if (yes-or-no-p "Is the conductivity test for the
ignition coil
                                              positive
(yes/no)? ")
           then
                (assert (repair "Repair the distributor lead
wire."))
           else
                (assert (repair "Replace the ignition
coil.")))
 )
 (defrule determine-sluggishness " "
      (working-state engine unsatisfactory)
      (not (repair ?))
      =>
      (if (yes-or-no-p "Is the engine sluggish (yes/no)? ")
           then
                (assert (repair "Clean the fuel line.")))
 )
 (defrule determine-misfiring " "
      (working-state engine unsatisfactory)
      (not (repair ?))
      =>
      (if (yes-or-no-p "Does the engine misfire (yes/no)? ")
           then
```

```
                (assert (repair "Point gap adjustment."))
                (assert (spark-state engine irregular-spark)))
)
(defrule determine-knocking  " "
     (working-state engine unsatisfactory)
     (not (repair ?))
     =>
     (if (yes-or-no-p "Does the engine knock (yes/no)? ")
          then
                (assert (repair "Timing adjustment.")))
)
```

ЗАДАНИЕ

Внимательно взглянув на список правил, мы увидим, что некоторые правила (2, 3 и 13) остались до сих пор не реализованными. Попробуйте их реализовать самостоятельно.

В качестве реализации правила 13 мы будем использовать правило *no-repairs*, приведенное в примере 11.

Пример 11. Правило *no-repairs*

```
(defrule no-repairs  " "
     (declare (salience -10))
     (not (repair ?))
     =>
     (assert (repair "Take your car to a mechanic."))
)
```

Обратите внимание на использование приоритета при определении этого правила. Все правила, приведенные в предыдущем разделе, определялись с приоритетом, по умолчанию равным нулю. Использование для правила *no-repairs* приоритета, равного —10, гарантирует, что правило не будет выполнено, пока в плане решения задачи находится, по крайней мере, одно из диагностических правил. Если все активированные диагностические правила отработали и ни одно из них не смогло подобрать подходящую рекомендацию по устранению неисправности, то CLIPS запустит правило *no-repairs*, которое просто порекомендует пользователю обратиться к более опытному механику.

Реализация правил 2 и 3 приведена ниже.

Пример 12. Правила *normal-engine-state-conclusions* и *unsatisfactory-engine-state-conclusions*

```
(defrule normal-engine-state-conclusions  " "
    (declare   (salience 10))
    (working-state engine normal)
    =>
    (assert (repair "No repair needed."))
    (assert (spark-state engine normal))
    (assert (charge-state battery charged))
    (assert (rotation-state engine rotates))
)
(defrule unsatisfactory-engine-state-conclusions " "
    (declare (salience 10))
    (working-state engine unsatisfactory)
    =>
    (assert (charge-state battery charged))
    (assert (rotation-state engine rotates))
)
```

В этих правилах, наоборот, используется более высокий приоритет, что гарантирует их выполнение до выполнения любого диагностического правила (естественно, только в случае удовлетворения условий, заданных в левой части правил). Это избавит нашу систему от лишних проверок, а пользователя от лишних вопросов.

5. МЕТОД ВЫВОДА ИТОГОВОЙ ИНФОРМАЦИИ

Наша экспертная система фактически готова к работе. Единственное, чего ей не хватает, — это метода вывода итоговой информации и правила, сообщающего пользователю о начале работы. Ниже приведена реализация этих правил.

Пример 13. Правила *system-banner* и *print-repair*

```
(defrule system-banner " "
    (declare (salience 10))
    =>
    (printout t crlf crlf)
    (printout t "****************************************"
crlf)
    (printout t "* The Engine Diagnosis Expert System *"
crlf)
    (printout t "****************************************"
crlf)
```

```
    (printout t crlf crlf) )
(defrule print-repair " "
    (declare (salience 10))
    (repair ?item)
    =>
    (printout t crlf crlf)
    printout t "Suggested Repair:")
    (printout t crlf crlf)
    (format t " %s%n%n%n" ?item)
)
```

6. ЗАПУСК И ТЕСТИРОВАНИЕ ПРОГРАММЫ

Для запуска программы, наберите приведенный в приложении 1 листинг в каком-нибудь текстовом редакторе (лучше использовать встроенный редактор CLIPS). Сохраните набранный файл, например, с именем *auto.CLP*.

После этого запустите CLIPS или, если он уже был у вас запущен, очистите его командой ***clear***. Загрузите созданный вами файл с помощью команды ***load*** *"auto.CLP"*. Если файл был набран без ошибок, то вы должны увидеть сообщения, представленные на рис. 1.

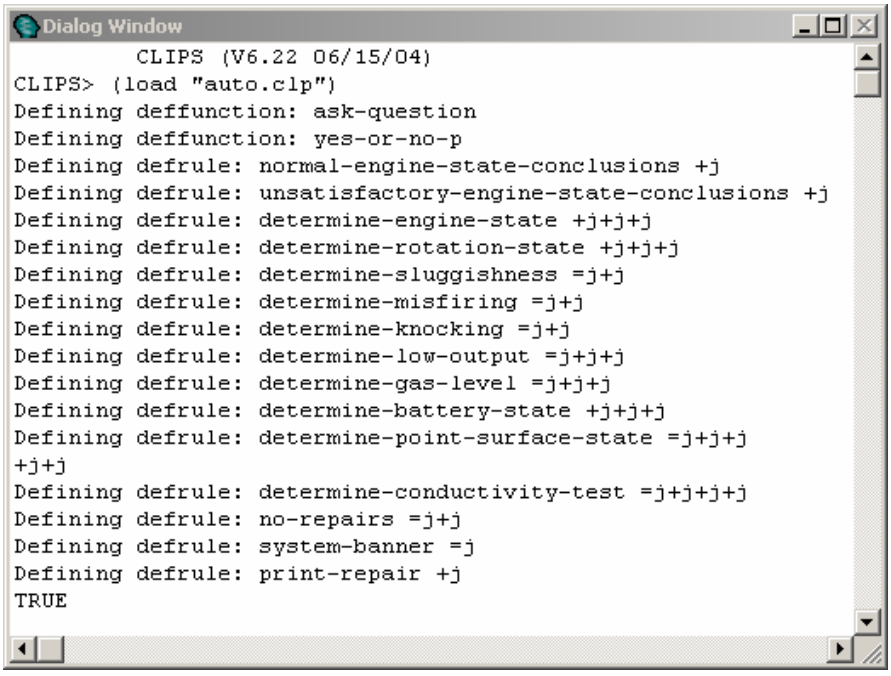

Рис 1. Загрузка экспертной системы

Рисунок 1 демонстрирует успешную попытку загрузки файла конструкторов. Обратите внимание, что функция ***load*** вернула значение TRUE. Если это не так, значит, в синтаксисе определений функций или правил была допущена ошибка. Для загрузки вы также могли бы воспользоваться функцией ***load****. В этом случае на экран не выводилась бы информация, отражающая процесс загрузки.

После удачной загрузки файла убедитесь, что все правила присутствуют в списке правил CLIPS, а функции — в списке функций. Легче всего это выполнить с помощью менеджеров правил и функций соответственно. Внешний вид этих менеджеров показан на рис. 2 и 3.

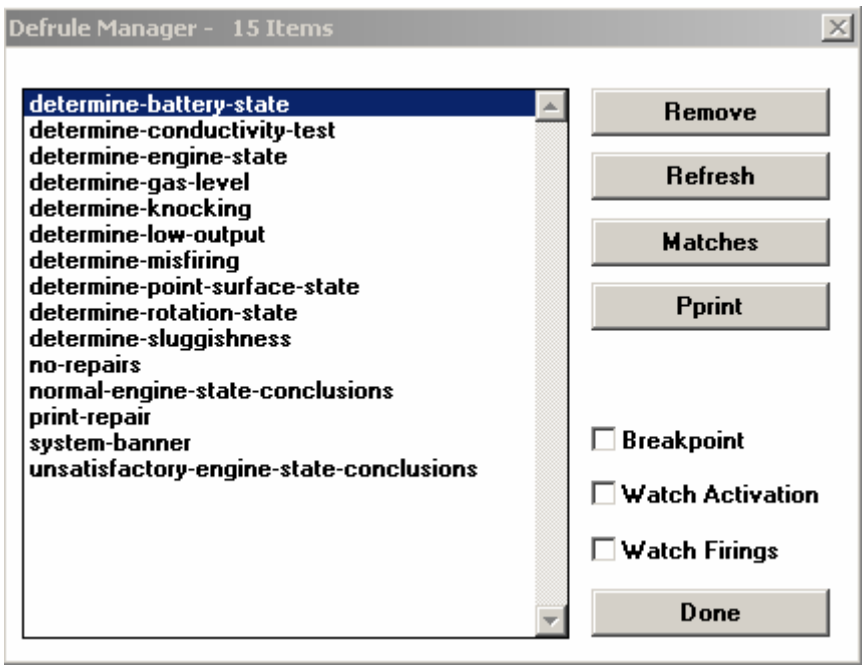

Рис 2. Правила экспертной системы

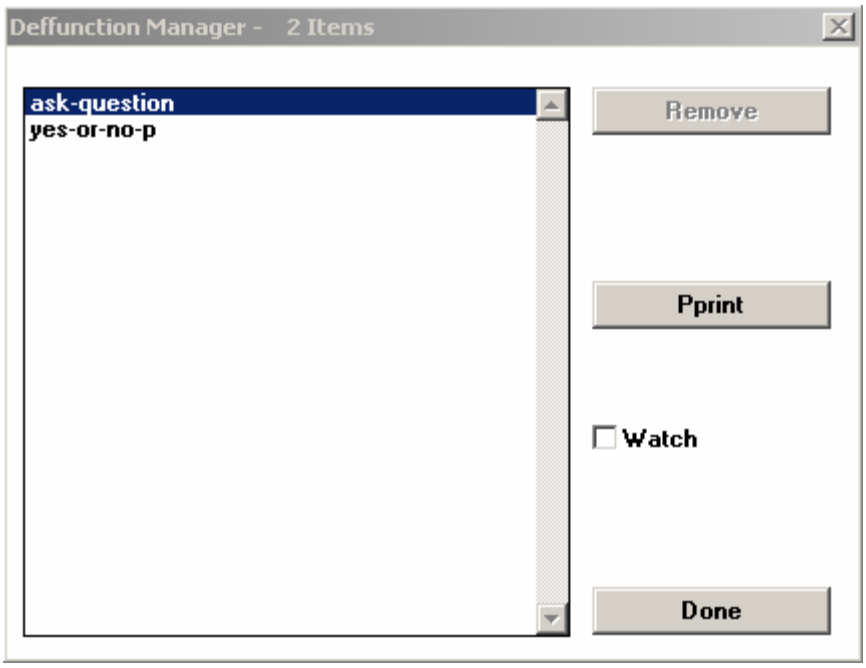

Рис 3. Функции экспертной системы

Для того чтобы запустить нашу экспертную систему, достаточно выполнить команду ***reset***, которая добавит факт ***initial-fact***, необходимый для правила *system-banner*, и команду ***run***. После этого вы сразу увидите сообщение *"The Engine Diagnosis Expert System"*, которое означает, что система начала работать, и получите серию вопросов, ответы на которые помогут экспертной системе оценить текущее состояние вашей машины и подобрать соответствующую рекомендацию по ремонту. Пример работы системы показан на рис. 4.

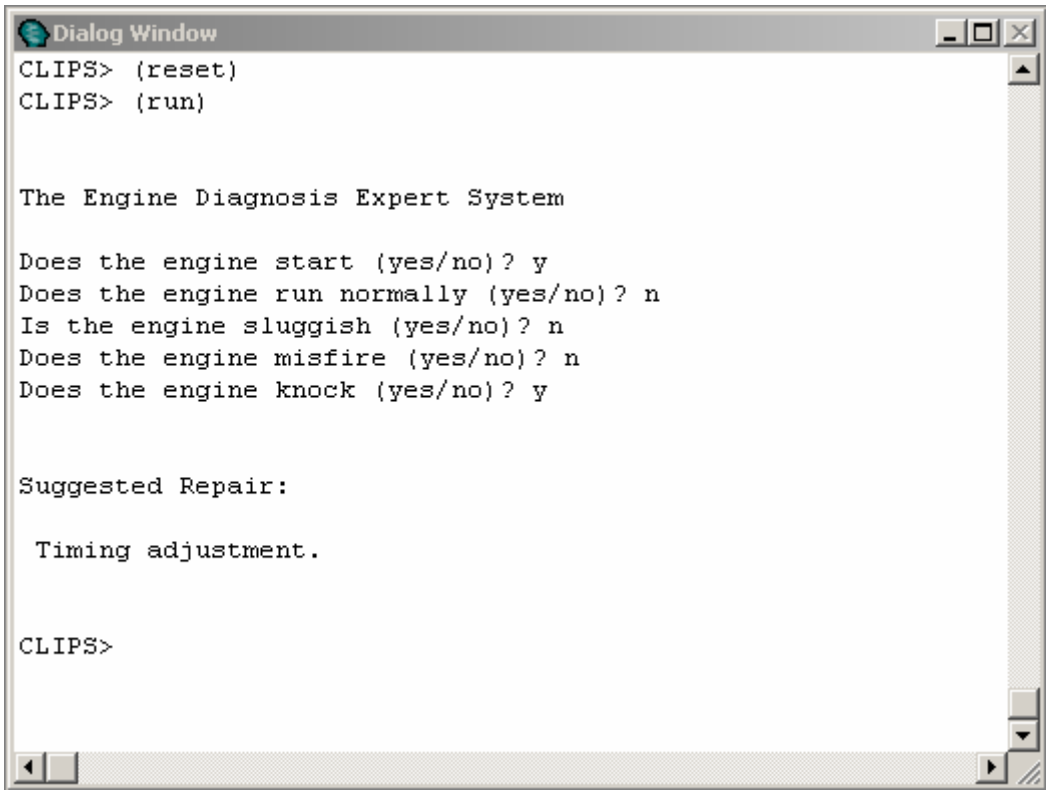

Рис 4. Диалог с экспертной системой

Обратите внимание, что если после завершения работы нашей экспертной системы в списке фактов CLIPS остаются факты, описывающие состояние автомобиля, их легко просмотреть с помощью команды **Fact Window** из меню **Window**. Факты для нашего примера изображены на рис. 5. Для повторного запуска экспертной системы необходимо еще раз выполнить команды ***reset*** и ***run***.

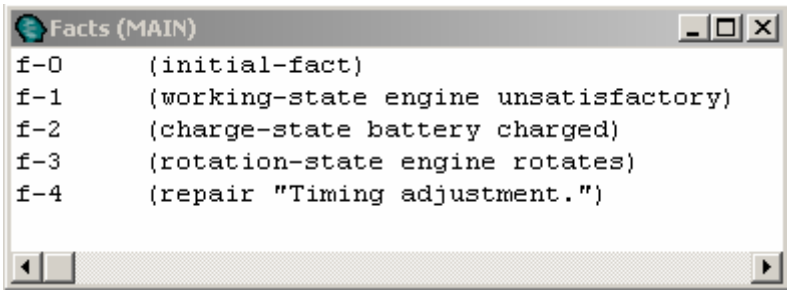

Рис 5. Результат работы экспертной системы

Протестируйте экспертную систему, по-разному отвечая на ее вопросы. Чтобы лучше понять механизмы ее работы и логический

механизм вывода CLIPS, перед запуском системы сделайте видимым окно фактов **Fact Window** и окно плана решения задачи **Agenda Window**.

В данной лабораторной работе мы рассмотрели достаточно реальный пример построения экспертной системы в CLIPS. Он является наглядным подтверждением того, что с помощью фактов, функций и правил CLIPS можно построить вполне работоспособную систему. Однако CLIPS содержит и более сложные конструкции, такие как родовые функции, объекты и модули. С их помощью вы сможете строить еще более гибкие и мощные экспертные системы.

7. КОНТРОЛЬНЫЕ ВОПРОСЫ:

1. Для чего служат правила в CLIPS? Из каких частей строятся правила? Как в CLIPS формируются эмпирические правила?
2. Что является в системе CLIPS фактом? Какие два типа фактов поддерживает CLIPS?
3. Какие функции в CLIPS являются внешними, а какие внутренними?
4. Как наложить ограничения на выполнение действий?
5. Как запустить экспертную систему, созданную в CLIPS?
6. Как протестировать созданную в CLIPS экспертную систему?

ПРИЛОЖЕНИЕ
Полный листинг программы

Приведем полный листинг программы с подробными комментариями. Помните, что среда CLIPS (по крайней мере, последняя версия 6.20) воспринимает только символы английского алфавита. Все комментарии в приведенном листинге даны на русском языке для наглядности, однако при вводе программы в таком виде CLIPS выведет сообщение об ошибке. Если у вас еще не сложилась целостная картина о том, как работает созданная экспертная система, из каких частей она состоит, внимательно изучите приведенный ниже код.

Пример 14. Полный листинг программы

```
; Пример экспертной системы на языке CLIPS

; Приведенная ниже экспертная система способна диагностировать некоторые неисправности автомобиля и предоставлять пользователю рекомендации по устранению неисправности.
```

```
; Вспомогательные функции.
```

; Функция ***ask-question*** задает пользователю вопрос, полученный в переменной *?question*, и получает от пользователя ответ, принадлежащий списку допустимых ответов, заданному в *$?allowed-values*.

```
(deffunction 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
 )
```

; Функция ***yes-or-no-p*** задает пользователю вопрос, полученный в переменной *?question*, и получает от пользователя ответ yes(y) или; no(n). В случае положительного ответа функция возвращает значение TRUE,

иначе — FALSE

```
(deffunction yes-or-no-p (?question)
(bind ?response (ask-question ?question yes no y n))
(if (or (eq ?response yes) (eq ?response y))
then
TRUE
else
FALSE)
 )
```

; Диагностические правила

; Правило ***determine-engine-state*** определяет текущее состояние двигателя машины по ответам, получаемым от пользователя. Двигатель может находиться в одном из трех состояний:

- ❒ работать нормально (working-state engine normal),
- ❒ работать неудовлетворительно (working-state engine unsatisfactory)
- ❒ и не заводиться (working-state engine; does-not-start) (см. правило 1).

```
(defrule determine-engine-state ""
(not (working-state engine ?))
(not (repair ?))
=>
(if (yes-or-no-p "Does the engine start (yes/no)? ")
then
(if (yes-or-no-p "Does the engine run normally
(yes/no)? ")
then
(assert (working-state engine normal))
else
(assert (working-state engine unsatisfactory)))
else
(assert (working-state engine does-not-start)))
 )
```

; Правило ***determine-rotation-state*** определяет состояние вращения двигателя по ответу, получаемому от пользователя. Двигатель может вращаться (rotation-state engine rotates) или не вращаться (spark-state engine does-not-spark) (см. правило 4). Кроме того, правило делает предположение о наличии плохой искры или ее отсутствии в системе зажигания.

```
(defrule determine-rotation-state ""
(working-state engine does-not-start)
```

```
(not (rotation-state engine ?))
(not (repair ?))
(if (yes-or-no-p "Does the engine rotate (yes/no)? ")
then
; Двигатель вращается
(assert (rotation-state engine rotates))
; Плохая искра
(assert (spark-state engine irregular-spark))
else
; Двигатель не вращается
(assert (Dotation-state engine does-not-rotate))
; Нет искры
(assert (spark-state engine does-not-spark)))
 )
```

; Правило ***determine-gas-level*** по ответу пользователя определяет наличие топлива в баке. В случае если топлива нет, пользователю выдается рекомендация по ремонту — машину необходимо заправить (repair "Add gas.") (см. правило 5). При появлении соответствующей рекомендации выполнение диагностических правил прекращается.

```
(defrule determine-gas-level ""
(working-state engine does-not-start)
(rotation-state engine rotates)
(not (repair ?))
(if (not (yes-or-no-p "Does the tank have any gas in it
(yes/no)? "))
then
; Машину необходимо заправить
(assert (repair "Add gas.")))
 )
```

; Правило ***determine-battery-state*** по ответу пользователя определяет, заряжен ли аккумулятор. В случае если это не так, пользователю выдается рекомендация по ремонту — Зарядите аккумулятор (repair "Charge the battery.") (см. правило 6). Кроме того, правило добавляет факт, описывающий состояние аккумулятора. Выполнение диагностических правил прекращается.

```
(defrule determine-battery-state ""
(rotation-state engine does-not-rotate)
; Состояние аккумулятора еще не определено
```

```
(not (charge-state battery ?))
(not (repair ?))
=>
(if (yes-or-no-p "Is the battery charged (yes/no)? ")
then
```
; Аккумулятор заряжен
```
(assert (charge-state battery charged))
else
```
; Зарядите аккумулятор
```
(assert (repair "Charge the battery."))
```
; Аккумулятор разряжен
```
(assert (charge-state battery dead))) )
```

; Правило ***determine-low-output*** определяет, развивает ли двигатель нормальную выходную мощность или нет и добавляет в систему факт, описывающий эту характеристику (см. правило 12).

```
(defrule determine-low-output ""
(working-state engine unsatisfactory)
```
; Мощность работы двигателя еще не определена
```
(not (symptom engine low-output I not-low-output))
(not (repair ?))
=>
(if (yes-or-no-p "Is the output of the engine low (yes/no)?
")
then
```
; Низкая выходная мощность двигателя
```
(assert (symptom engine low-output))
else
```
; Нормальная выходная мощность двигателя
```
(assert (symptom engine not-low-output)))
 )
```

; Правило ***determine-point-surface-state*** определяет по ответу пользователя состояние контактов (см. правила 7, 12). Контакты могут находиться в одном из трех состояний: чистые, опаленные и загрязненные. В двух последних случаях пользователю выдаются соответствующие рекомендации. Выполнение диагностических правил прекращается.

```
(defrule determine-point-surface-state  ""
(or (and (working-state engine does-not-start)
```
; не заводится

```
(spark-state engine irregular-spark))
; и плохая искра
(symptom engine low-output))
; или низкая мощность
(not (repair ?))
 =>
(bind ?response (ask-question "What is the surface state of
the points (normal /burned /contaminated)?" normal burned
contaminated))
(if (eq ?response burned)
then
; Контакты опалены — замените контакты
(assert (repair "Replace the points."))
else
(if (eq ?response contaminated)
then
; Контакты загрязнены - почистите их
(assert (repair "Clean the points.")))
 )
```

; Правило ***determine-conductivity-test*** по ответу пользователя определяет, пропускает ли ток катушка зажигания. Если нет, то ее следует заменить. Если пропускает, то причина неисправности — распределительные провода. Для нормальной работы правила необходимо убедиться, что аккумулятор заряжен и искры нет (см. правило 8). Выполнение диагностических правил прекращается.

```
(defrule determine-conductivity-test ""
(working-state engine does-not-start)
(spark-state engine does-not-spark)
;нет искры
(charge-state battery charged)
;аккумулятор заряжен
(not (repair ?))
 =>
(if (yes-or-no-p "Is the conductivity test for the ignition
coil
positive(yes/no)? ")
then
; Замените распределительные провода
(assert (repair "Repair the distributor lead wire."))
```

```
else
; Замените катушку зажигания
(assert (repair "Replace the ignition coil."))) )
```

; Правило ***determine-sluggishness*** спрашивает пользователя, не ведет ли себя машина инертно (не сразу реагирует на подачу топлива). Если такой факт обнаружен, то необходимо прочистить топливную систему (см. правило 9) и выполнение диагностических правил прекращается.

```
(defrule determine-sluggishness ""
(working-state engine unsatisfactory)
(not (repair ?))
 =>
(if (yes-or-no-p "Is the engine sluggish (yes/no)? ")
then
; Прочистите систему подачи топлива
(assert (repair "Clean the fuel line.")))
 )
```

; Правило ***determine-misfiring*** узнает — нет ли перебоев с зажиганием. Если это так, то необходимо отрегулировать зазоры между контактами (см. правило 10). Выполнение диагностических правил прекращается.

```
(defrule determine-misfiring ""
(working-state engine unsatisfactory)
(not (repair ?))
=>
(if (yes-or-no-p "Does the engine misfire (yes/no)? ")
then
; Отрегулируйте зазоры между контактами
(assert (repair "Point gap adjustment."))
; Плохая искра
(assert (spark-state engine irregular-spark)))
 )
```

; Правило ***determine-knocking*** узнает — не стучит ли двигатель. Если это так, то необходимо отрегулировать зажигание (см. правило 11). Выполнение диагностических правил прекращается.

```
(defrule determine-knocking ""
(working-state engine unsatisfactory)
(not (repair ?))
 =>
(if (yes-or-no-p "Does the engine knock (yes/no)? ")
```

```
then
; Отрегулируйте положение зажигания
(assert (repair "Timing adjustment.")))
 )
```

; Правила, определяющие состояние некоторых подсистем автомобиля по характерным состояниям двигателя

; Правило ***normal-engine-state-conclusions*** реализует правило 2

```
(defrule normal-engine-state-conclusions ""
(declare (salience 10))
; Если двигатель работает неудовлетворительно
(working-state engine normal)
=>
; то
(assert (repair "No repair needed."))
; ремонт не нужен
(assert (spark-state engine normal))
; зажигание в норме
(assert (charge-state battery charged))
; аккумулятор заряжен
(assert (rotation-state engine rotates))
; двигатель вращается
 )
```

; Правило ***unsatisfactory-engine-state-conclusions*** реализует правило 3.

```
(defrule unsatisfactory-engine-state-conclusions ""
(declare (salience 10))
; Если двигатель работает нормально
(working-state engine unsatisfactory)
=>
; то
(assert (charge-state battery charged))
; аккумулятор заряжен
(assert (rotation-state engine rotates))
; двигатель вращается
 )
```

Запуск и завершение

Правило ***no-repairs*** запускается в случае, если ни одно из диагностических правил не способно определить неисправность. Правило корректно прерывает выполнение экспертной системы и предлагает пройти более

тщательную проверку (см. правило 13).

```
(defrule no-repairs ""
(declare (salience -10))
(not (repair ?))
=>
(assert (repair "Take your car to a mechanic."))
 )
```

; Правило ***print-repair*** выводит на экран диагностическое сообщение по устранению найденной неисправности,

```
(defrule print-repair ""
(declare (salience 10))
(repair ?item)
=>
(printout t crlf crlf)
(printout t "Suggested Repair:")
(printout t crlf crlf)
(format t " %s%n%n%n" ?item)
 )
```

; Правило ***system-banner*** выводит на экран название экспертной системы при каждом новом запуске.

```
(defrule system-banner " каждом новом запуске."
(declare (salience 10))
=>
```

; каждом новом запуске.

```
(printout t crlf crlf)
(printout t ч**************************************" crlf)
(printout t "* The Engine Diagnosis Expert System *" crlf)
(printout t "**************************************" crlf)
(printout t crlf crlf)
 )
```