ВВЕДЕНИЕ

Предлагаемое методическое пособие представляет собой описание лабораторных работ по курсу «Интеллектуальные информационные системы».

Основная цель пособия – помощь студентам в подготовке к выполнению лабораторных работ по изучению языка программирования CLIPS, ориентированного на разработку экспертных систем.

Целью выполнения данных лабораторных работ является приобретение и усвоение студентами навыков работы в среде CLIPS (формальная постановка задачи, преобразование в форму, пригодную для программирования на CLIPS, оценка результатов) а также закрепление теоретических знаний, полученных при прослушивании курса лекций «Интеллектуальные информационные системы».

Среда CLIPS представляет собой современный инструмент, предназначенный для создания экспертных систем. В настоящее время CLIPS находится в открытом доступе. Официальный сайт CLIPS располагается по адресу: http://www.ghg.net/clips/CLIPS.html.

В данном пособии рассматриваются основные понятия и необходимые сведения по программированию на языке искусственного интеллекта CLIPS.

Пособие содержит необходимый теоретический материал, примеры решения задач и задания для самостоятельной работы.

К каждому разделу приводятся несколько типовых задач с решениями, что позволяет студентам самостоятельно подготовиться к выполнению лабораторной работы.

В качестве результата выполнения лабораторной работы студентом должен быть подготовлен отчет о проделанной работе, который включает:

1. Постановку задачи
2. Входные/Выходные данные (при необходимости)
3. Текст программы на языке CLIPS (с необходимыми комментариями)
4. Результаты тестирования (наборы входных и выходных данных).

В настоящем пособии приводится список литературы, которая может быть использована для организации самостоятельной работы студентов.

1.ОСНОВЫ ПРОГРАММИРОВАНИЯ В CLIPS

1.1. ИНТЕРФЕЙС CLIPS. ВЫЧИСЛЕНИЕ МАТЕМАТИЧЕСКИХ ВЫРАЖЕНИЙ В РЕЖИМЕ КОМАНДНОЙ СТРОКИ

Основные сведения

Среда CLIPS (C Language Integrated Production System) предназначена для построения экспертных систем (ЭС). Язык был разработан в Центре космических исследований NASA (NASA's Johnson Space Center) в середине 1980-х годов и во многом сходен с языками, созданными на базе LISP и OPS5. Сейчас CLIPS и документация на этот инструмент свободно распространяется через интернет.

CLIPS поддерживает три основных способа представления знаний:

- продукционные правила для представления эвристических, основанных на опыте знаний;
- функции для представления процедурных знаний;
- объектно-ориентированное программирование.

Среда загружается запуском файла clipswin.exe.

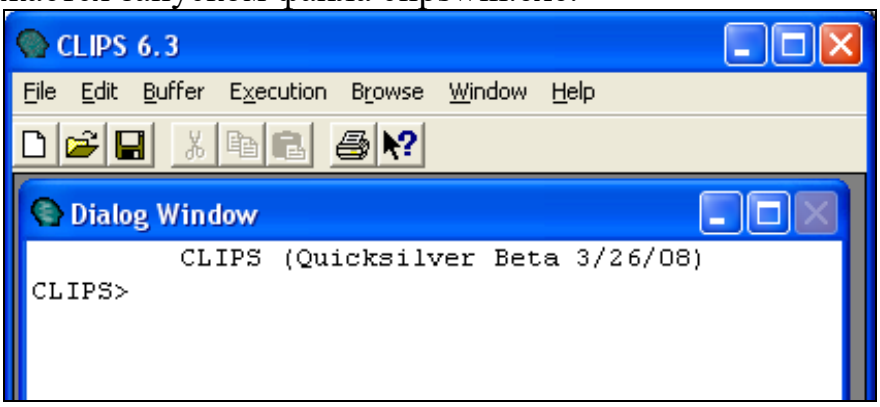

Рис.1. Окно среды CLIPS

В окне отображается стандартная строка приглашения CLIPS, куда и вводятся команды.

Назначение основных пунктов меню оконного интерфейса, используемых при выполнении данного цикла лабораторных работ представлены в табл. 1.

Таблица 1 Основные команды главного меню CLIPS

Пункт	**Подпункт**	**“Горячие” клавиши**	**Назначение команды**
File	New Open Load … Load Batch	Ctrl+N Ctrl+O Ctrl+L	Создание нового файла Открытие файла Загрузка конструкций из файла. Исполнение пакетного файла
Edit	Cut Copy Paste	Ctrl+X Ctrl+C Ctrl+V	Вырезка фрагмента Копирование фрагмента Вставка строки из буфера обмена

Execution	Reset Run Step	Ctrl+U Ctrl+R Ctrl+T	Инициализация конструкций Запуск на выполнение Выполнение одного шага вывода
Browse	Module Defrule Manager Deffacts Manager		Отображает модуль Менеджер правил Менеджер фактов
Window	Facts Window Agenda Window Clear dialog window		Активизация окна списка фактов Активизация окна агенды Очищает окно с командной строкой

CLIPS может работать в нескольких режимах:

1. интерактивно, с использованием простого текстового интерфейса командной строки;
2. интерактивно, с использованием GUI-интерфейса;
3. как ЭС, интегрированная в другие приложения.

В режиме интерпретатора пользователь может использовать множество команд. Основным методом взаимодействия пользователя с CLIPS является ввод команд с командной строки CLIPS. После появления на экране подсказки **CLIPS>** пользователь может ввести команду (рис.1).

Командами могут быть вызовы функций, конструкции, глобальные переменные или константы. Если ввести вызов функции, вычисляется значение этой функции и на экран выводится результат.

Вызовы функций в CLIPS имеют префиксную форму: аргументы функции могут стоять только после ее названия. Вызов функции начинается с открывающейся скобки, за которой следует имя функции, затем идут аргументы, каждый из которых отделен одним или несколькими пробелами. Аргументами функции могут быть данные простых типов, переменные или вызовы других функций. В конце вызова ставится закрывающаяся скобка. Например, выражение 3 + 8*9 + 4 в CLIPS записывается следующим образом:

(+ 3 (* 8 9) 4)

Синтаксис языка CLIPS можно разбить на три основных группы элементов, предназначенных для написания программ:

– типы данных;

– функции, использующиеся для обработки данных;

– конструкторы, предназначенные для создания таких структур языка, как факты, правила, классы и т. д.

В CLIPS поддерживаются восемь простейших типов данных:

integer – целые числа (237, 15, +12, -32);

float – числа с плавающей запятой (237e3, 15.09, +12.0, -32.3e-7);

symbol – символьный тип (любая последовательность символов, начинающаяся с отображаемого ASCII-символа и продолжающаяся до ограничителя. Ограничителем является любой неотображаемый символ);

external-address – внешний адрес (значение этого типа может быть создано только посредством вызова внешней функции.);

fact-address – адрес факта (оперировать с фактом можно, используя его адрес, который представлен значением данного типа.);

instance-name – имя экземпляра (данный тип предназначен для хранения значения имени объекта, который представляет собой экземпляр определенного пользователем класса);

instance-address – адрес экземпляра (данный тип предназначен для хранения значения, представляющего адрес объекта).

Наиболее часто используемыми командами в CLIPS являются:

clear –очистка рабочей памяти системы. Команда удаляет все определенные в системе на текущий момент конструкторы и ассоциированные с ними данные.

exit –завершение сеанса работы с CLIPS.

reset –перезагрузка рабочей памяти системы. Команда очищает текущий план решения задачи, удаляет все факты из списка фактов и объекты из списка объектов. При этом в систему добавляется предопределенный факт *initial-fact*, предопределенный объект *initial-object* и все факты, объекты и глобальные переменные, определенные пользователем с помощью конструкторов *deffacts*, *definstances* и *defglobals*.

В CLIPS предусмотрен ряд стандартных арифметических и математических функций

Таблица 2. Запись математических функций в CLIPS

Функция	**Обозначение функции в CLIPS**
Сложение	+
Вычитание	-
Умножение	*
Деление	/
Возведение в степень	* *
Определение абсолютного значения	abs
Вычисление квадратного корня	sqrt
Целочисленное деление	div
Остаток от деления	mod
Нахождение минимума	min
Нахождение максимума	max
Синус	sin
Косинус	cos
Тангенс	tan
Натуральный логарифм	log
Экпонента e^x	exp
Округление числа	round
Выбор целого случайного числа из интервала [n1, n2]	Random n1 n2

Пример.

В режиме командной строки вычислить значения выражений:

а) (3+5)*2 б) max(3^2, 2^3) в) $\sqrt{|\sin 3.2|} \bullet \sqrt{e^3}$

Решение

1. Запустите Clips и в командной строке окна Dialog Window запишите выражения:

а)

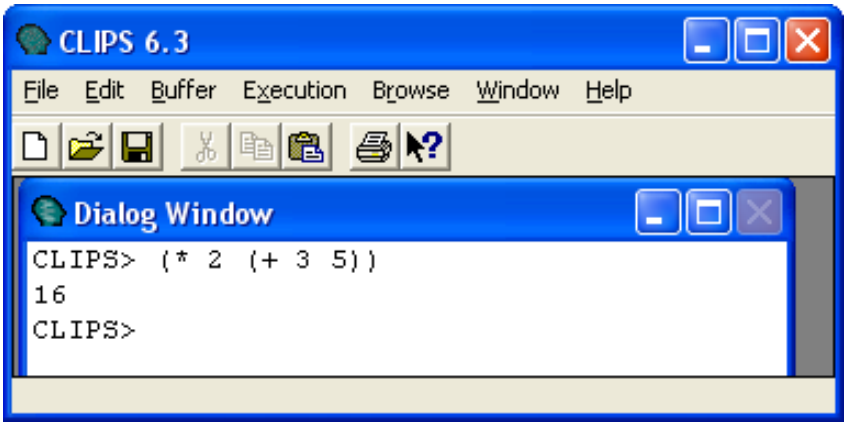

Рис.2. Вычисление выражения (3+5)*2 в режиме командной строки

б)

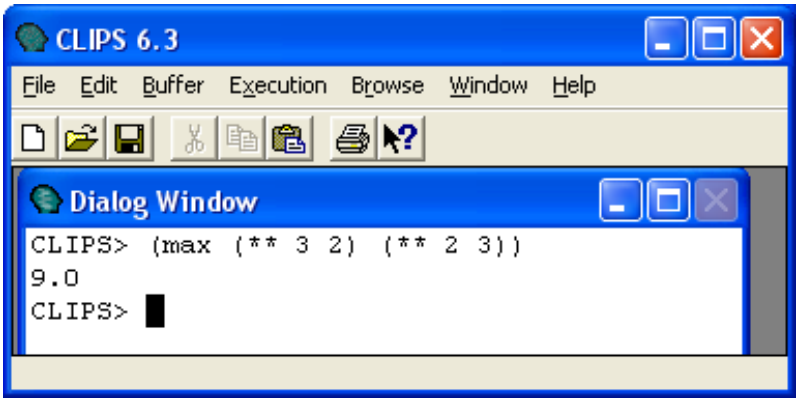

Рис.3. Вычисление выражения max(3^2, 2^3) в режиме командной строки

в)

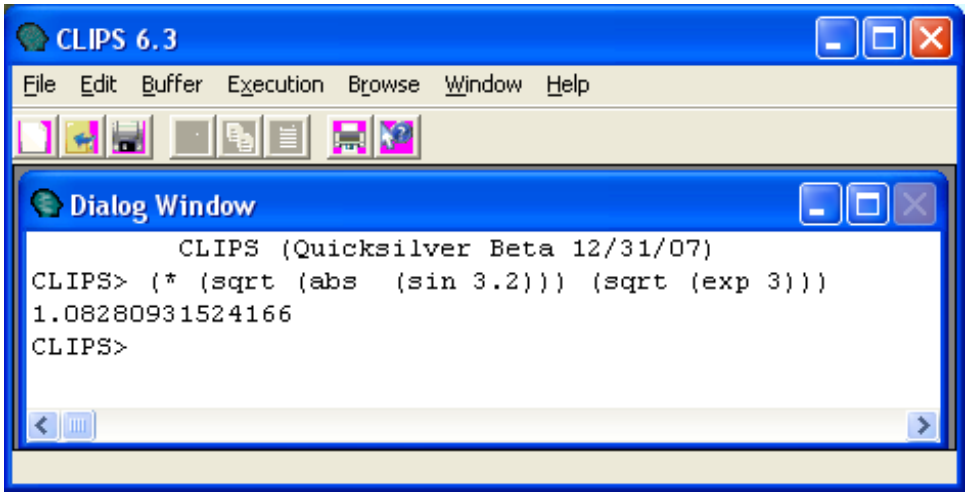

Рис.4. Вычисление выражения $\sqrt{|\sin 3.2|} \bullet \sqrt{e^3}$ в режиме командной строки

Контрольные вопросы

1. Поясните назначение инструментальной среды Clips.
2. Какие способы представления знаний поддерживает Clips?
3. Какие режимы работы допустимы в среде Clips?
4. Каково назначение основных пунктов меню оконного интерфейса Clips?
5. Какая форма записи используется в CLIPS для выражений?
6. Перечислите основные типы данных в Clips.
7. Поясните назначение команд clear, exit, reset.

Задания для самостоятельной работы

В режиме командной строки вычислить значения выражений:

a) $(4^2 -5)*(3 + 4)$ b) $4^2 + 28/(5+2)$ c) $\sqrt{5^4 + \sqrt{7^2 +1} + \ln 20.5}$

d) $|3e^3 - 2\ln 34|$ e) $\max(2^3, 3^2, 2^5)$ f) $3^3 - e^{5+\sin 2}$

g) $\sin 1 + 1/(\cos 1 - 2)$ h) $2e^4 - 4 - |\sin 6^2|$ i) $1.1e^3 + \left|\cos\sqrt{\pi}\right| - \frac{4}{9}$

1.2. ФАКТЫ В CLIPS. УПОРЯДОЧЕННЫЕ ФАКТЫ.

Основные сведения

Факты являются одной из основных форм представления информации в системе CLIPS. Каждый факт представляет фрагмент информации, который был помещен в текущий список фактов, называемый *fact-list*. Факт представляет собой основную единицу данных, используемую правилами. Количество фактов в списке и объем информации, который может быть сохранен в факте, ограничивается только размером памяти компьютера.

Факт может описываться индексом или адресом. Всякий раз, когда факт добавляется (изменяется), ему присваивается уникальный целочисленный индекс. Индексы фактов начинаются с нуля и для каждого нового или измененного факта увеличиваются на единицу. Идентификатор факта записывается в виде **f-<индекс>.** Например, запись f-10 служит для обозначения факта с индексом 10.

Для просмотра текущего списка фактов используется команда **facts.**

Каждый раз после выполнения команд *reset* и *clear* выделение индексов начинается с нуля.

Существует два формата представления фактов: упорядоченные и неупорядоченные (шаблонные). **Упорядоченные** факты состоят из выражения символьного типа, за которым следует последовательность (возможно, пустая) из полей, разделенных пробелами. Вся запись заключается в скобки. Первое поле определяет "отношение", которое применяется к оставшимся полям.

Примеры упорядоченных фактов:

(студент Смирнов_Сергей)

(однокурсники Иванов Петров Сидоров)

(цвет красный).

Для работы с фактами любого типа используются следующие функции:

1) **(assert <список фактов>)** – добавляет новые факты в текущий список. Количество фактов произвольное.

Если операция выполнена успешно, то функция возвращает адрес последнего добавленного факта, если операция по добавлению фактов была не успешной, то функция возвращает значение False.

2) **(retract <индекс>)** – удаляет из текущего списка произвольное количество фактов

3) (**fact-relation <индекс>)** – позволяет определить отношение (связь) факта с указанным индексом.

Функция возвращает значение первого поля факта, если данный факт существует, или значение false если не существует.

4) **(fact-existp <индекс факта>)** – позволяет определить, содержится ли в текущем списке фактов факт с указанным индексом. Если присутствует, то функция возвращается значение true, иначе false

Факты можно включать в базу данных не по одиночке, а целым массивом. Для этого в CLIPS имеется конструктор *deffacts*.

(deffacts <имя списка фактов> <Список фактов>)

Например,

```
(deffacts student_list
      (student Ivanov Ivan)
     (student Petrov Petr)
)
```

Выражение начинается с команды **deffacts**, затем приводится имя списка фактов (в примере — student_list), а за ним следуют элементы списка, причем их количество не ограничивается. Этот массив фактов можно тем удалить из базы командой **undeffacts**.

Пример 1.

В режиме командной строки создать список из 3-х упорядоченных фактов вида: (vedomost <name> <gruppa> <ocenka>). Просмотреть полученный список. Изменить список фактов следующим образом:

1) удалить факт f-1.

2)изменить в факте f-2 значение оценки

3)изменить в факте f-3 значение группы

Решение

Для формирования списка из 3-х упорядоченных фактов в режиме командной строки записываем команду **assert** с перечислением необходимых фактов (рис.5)

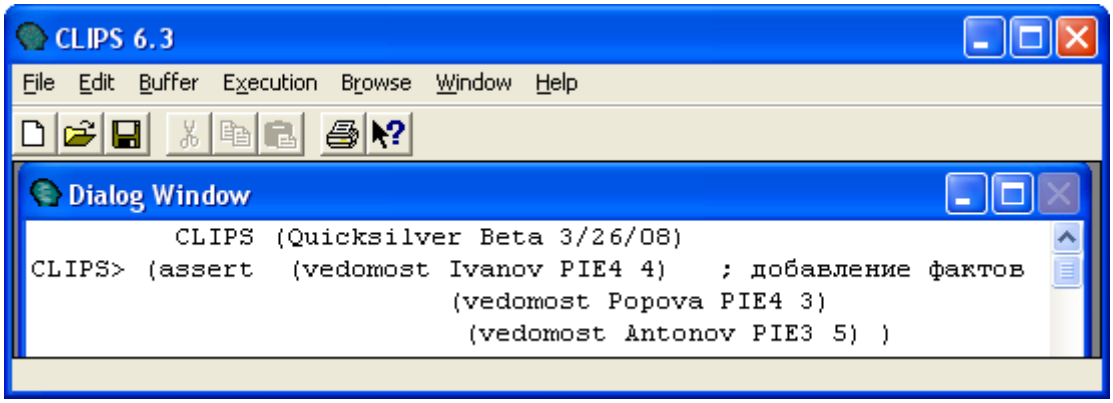

Рис.5 Добавление новых фактов в режиме командной строки

Для просмотра списка фактов записываем команду **facts** в командной строке (рис. 6).

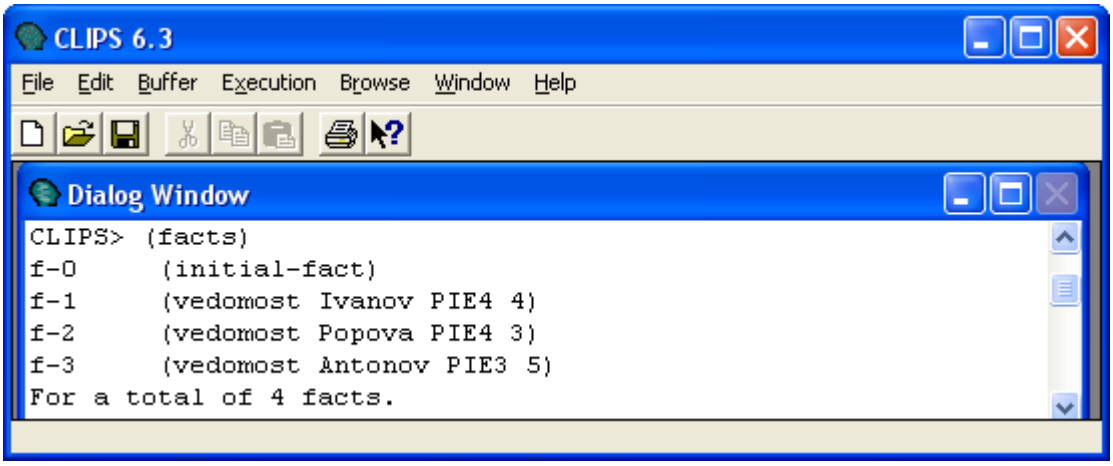

Рис. 6 Просмотр списка фактов

Чтобы из текущего списка фактов удалить факт, используем команду **retract** (рис.7)

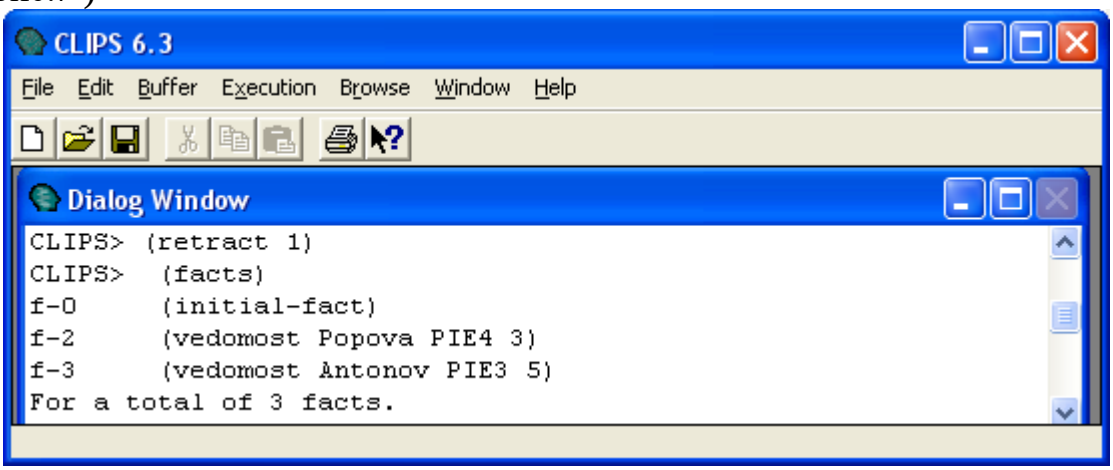

Рис. 7 Удаление факта f-1 и просмотр списка фактов.

Изменить значение поля в упорядоченном факте можно, удалив старый факт и добавив новый. Чтобы изменить в факте f-2 значение оценки, необходимо его удалить и добавить факт с новым значением оценки (рис.8)

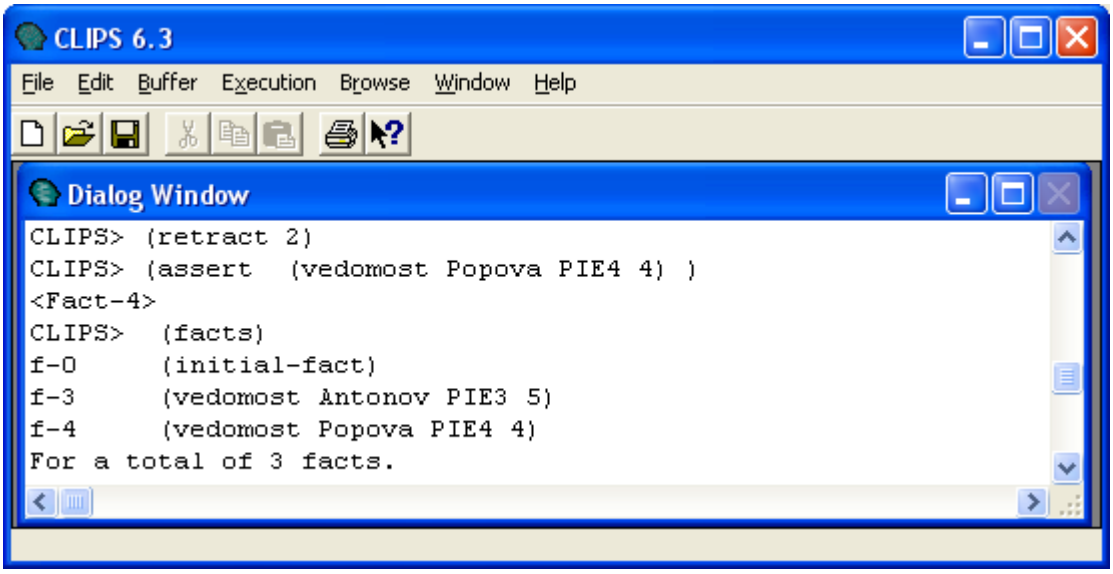

Рис. 8 Замена в факте f-2 значения оценки

Аналогично выполняется замена в факте f-3 значения группы (рис. 9)

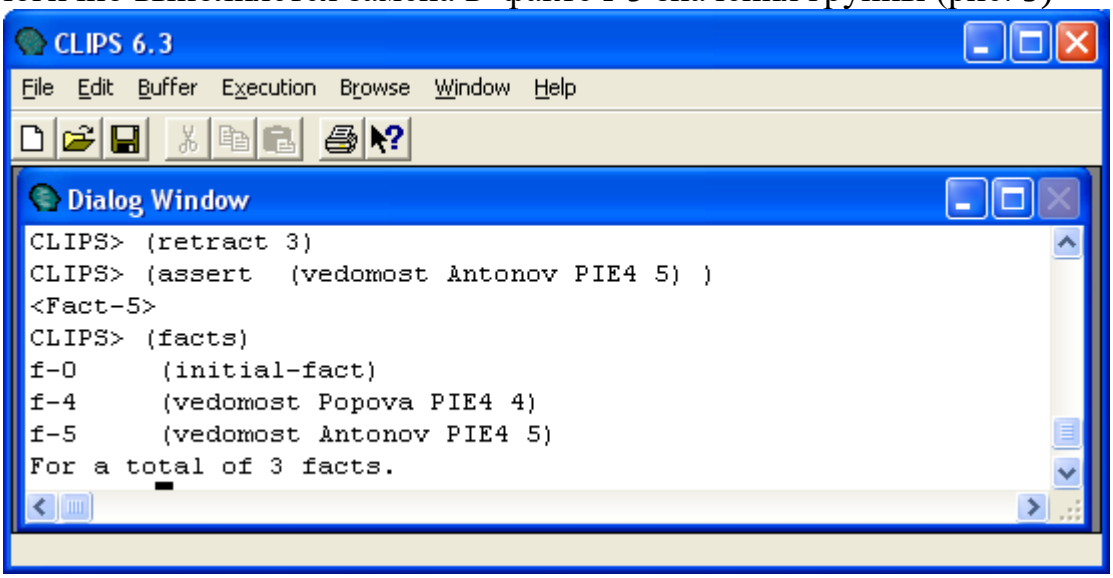

Рис. 9 Замена в факте f-3 значения группы

Пример 2.

С помощью конструктора **deffacts** создать три упорядоченных факта вида (color <название цвета>)**.** Добавить два новых факта, используя функцию **assert**. Удалить факты с индексами 2 и 4.

Решение

Создайте новый файл в CLIPS, в котором конструктор **deffacts** определяет список фактов (рис.10). Сохраните файл под именем, например color-list.

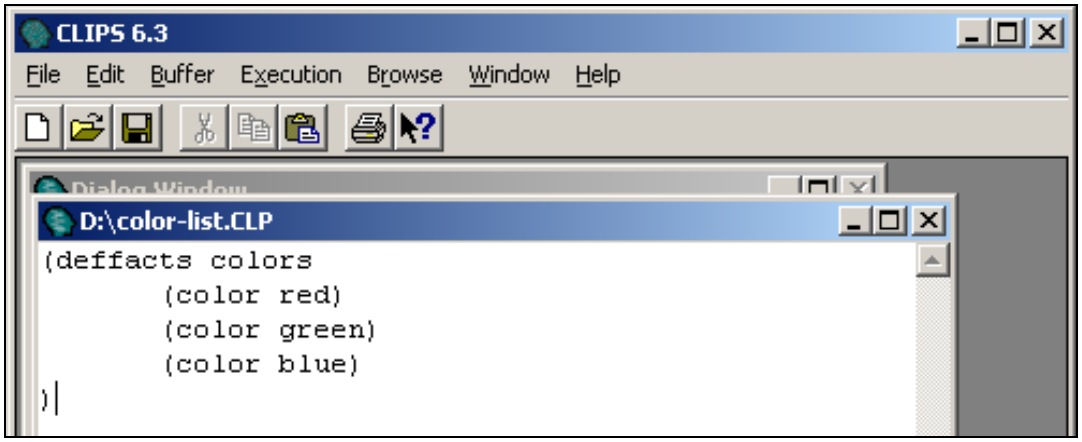

Рис.10 Содержимое файла color-list.clp

Загрузите данный файл в CLIPS с помощью команды **Load** (File -> Load). Сообщение интерпретатора TRUE означает, что в файле нет синтаксических ошибок и команда загрузки выполнена корректно (рис.11)

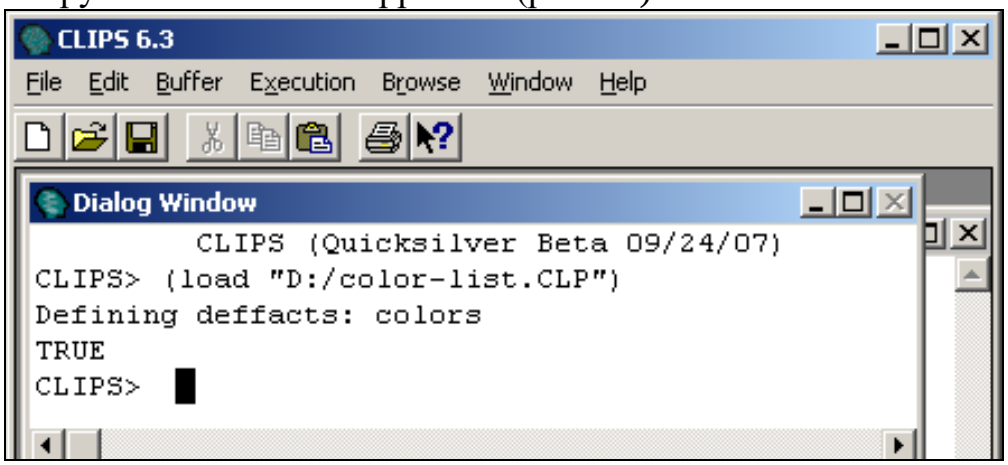

Рис. 11 Сообщение интерпретатора после загрузки файла color-list.clp

Выполните последовательно команды **reset**, а затем **facts** и просмотрите текущий список фактов:

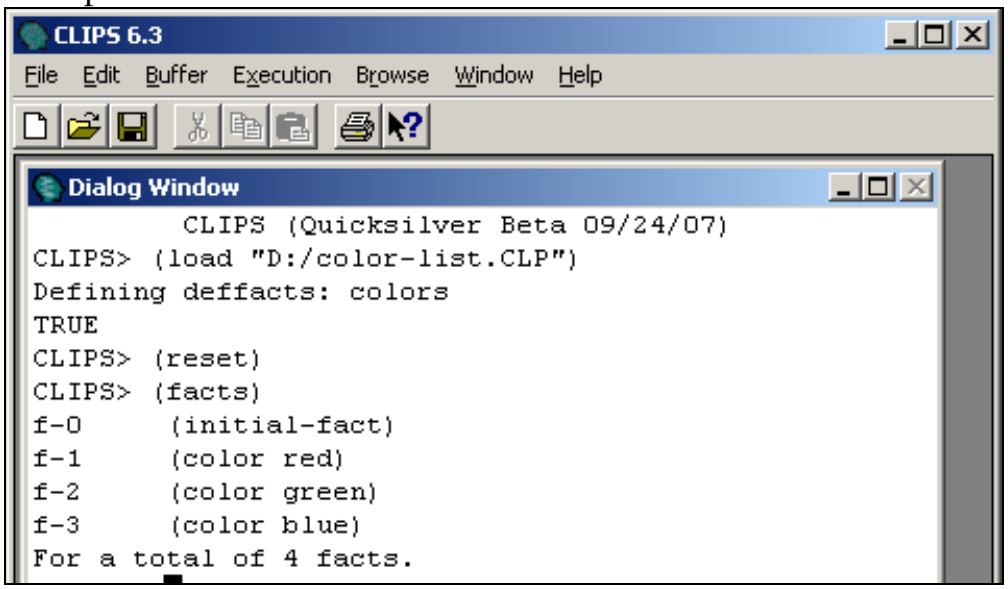

Рис.12 Текущий список фактов, полученный в результате загрузки файла color-list.clp

Добавьте еще один факт:

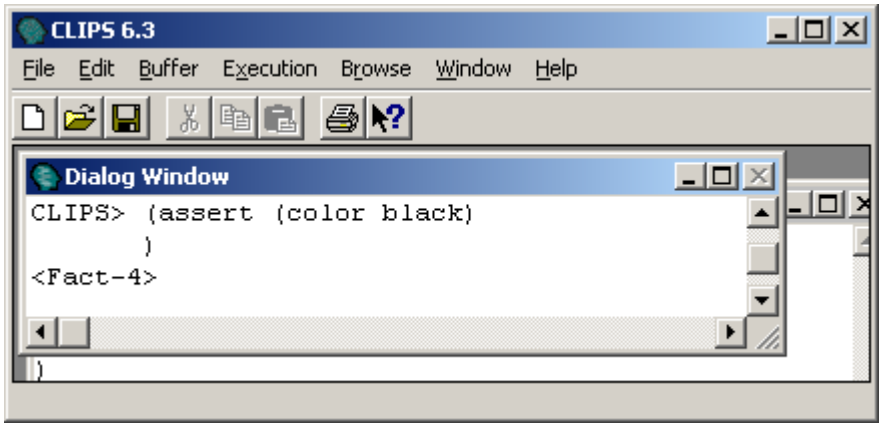

Рис.13. Добавление нового факта

Просмотрите весь список:

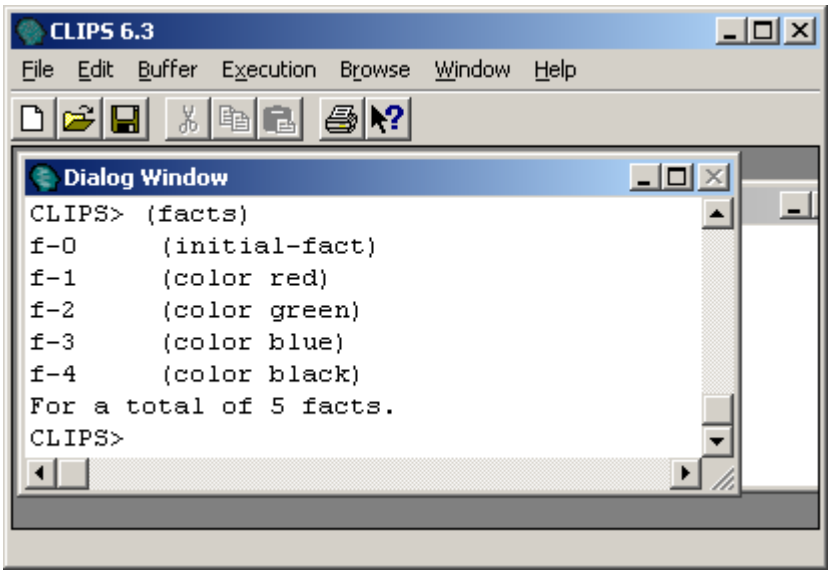

Рис.14. Текущий список фактов

5. Удалите факты 2 и 3 и просмотрите окончательный список фактов

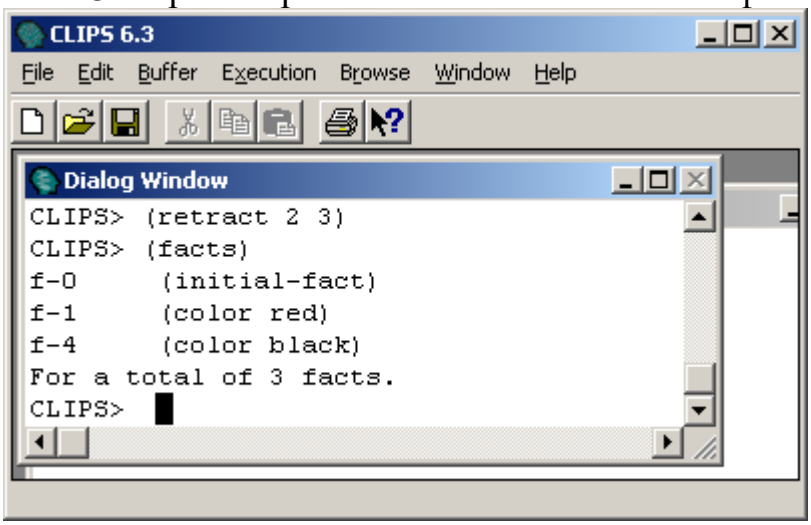

Рис.14 Текущий список фактов после удаления фактов f-2 и f-3

Контрольные вопросы

1. Поясните назначение фактов в Clips.
2. Как записывается индекс факта в Clips?
3. Какие типы фактов существуют в Clips?
4. Объясните структуру упорядоченных фактов.
5. Какая команда используется для просмотра текущего списка фактов?
6. Поясните назначение конструктора deffacts.
7. Какая функция используется для добавления новых фактов?
8. Какая функция используется для удаления фактов?
9. Поясните назначение функций fact-relation и fact-existp.

Задания для самостоятельной работы

1. В режиме командной строки создать список из 4-х упорядоченных фактов вида: (student <name> <kurs>). Просмотреть полученный список. Изменить список фактов следующим образом: а) удалить факт f-2; б) изменить в факте f-3 значение <kurs>.
2. В режиме командной строки создать три упорядоченных факта вида (poezd <номер> <пункт_назначения> <время_отправления>) с помощью конструктора deffacts. Добавить два новых факта, используя функцию assert. Просмотреть полученный список фактов. Удалить факты с индексами 1,4. В фактах с индексами 2, 3 изменить время отправления.
3. В режиме командной строки создать список из 4-х упорядоченных фактов вида: (sotrudnik <fio> <otdel>). Просмотреть полученный список. Изменить список фактов следующим образом: а) удалить факт f-1; б)изменить в факте f-3 значение <fio>
4. В режиме командной строки создать три упорядоченных факта вида (tovar <наименование> <страна-производитель> <цена> <количество>) с помощью конструктора deffacts. Добавить два новых факта, используя функцию assert. Просмотреть полученный список фактов. В фактах с индексами 2, 4 изменить цену. Удалить все факты.

1.3. НЕУПОРЯДОЧЕННЫЕ ФАКТЫ. СОХРАНЕНИЕ И ЗАГРУЗКА ФАКТОВ.

Основные сведения

Неупорядоченные (шаблонные) факты дают возможность пользователю абстрагироваться от структуры факта, задавая имена каждому из полей факта. Для задания шаблона, который затем может использоваться при доступе к полям по именам, используется конструкция **deftemplate**.

Структура: (deftemplate <имя шаблона>[«строка комментариев»])

(slot<имя слота 1>)

(slot<имя слота 2>)

……..

(slot<имя слота n>)

В шаблонном факторе значение любого слота можно задать по умолчанию с помощью служебного слова **Default**.

Для работы с шаблонными фактами используются функции

1) **(modify <индекс факта> <новое значение слота>)** - используется для модификации неупорядоченного факта

Механизм работы функции *modify* аналогичен выполнению функции удаления и добавления факта.

2) **(duplicate <индекс факта> <нов. значение слота>)** - позволяет копировать существующие факты по заданному шаблону, заменяя указанные значения слотов.

Список фактов любого типа может быть сохранен в текстовый файл, а также загружен из файла. Для этого определены функции:

3) (**save-facts** <имя файла>) - сохранение текущего списка фактов

4)(**load-facts** <имя файла>) - загрузка фактов из текстового файла

Пример.

Создать файл, содержащий три неупорядоченных факта вида **(student (name <фамилия>) (gruppa <группа>) (date <дата рождения>))**. Загрузить данные факты.

В факте с индексом 2 изменить значение слота date, факт f-1 продублировать, изменив фамилию. Сохранить полученный список фактов.

Решение.

Сначала необходимо создать файл, содержащий описание шаблона для неупорядоченных фактов вида (student (name <фамилия>) (gruppa <группа>) (date <дата рождения>)) с помощью конструктора **deftemplate**. Значения слота gruppa зададим по умолчанию – ПИЭ4-1.

Для этого создайте новый текстовый файл (File->New) и наберите текст:

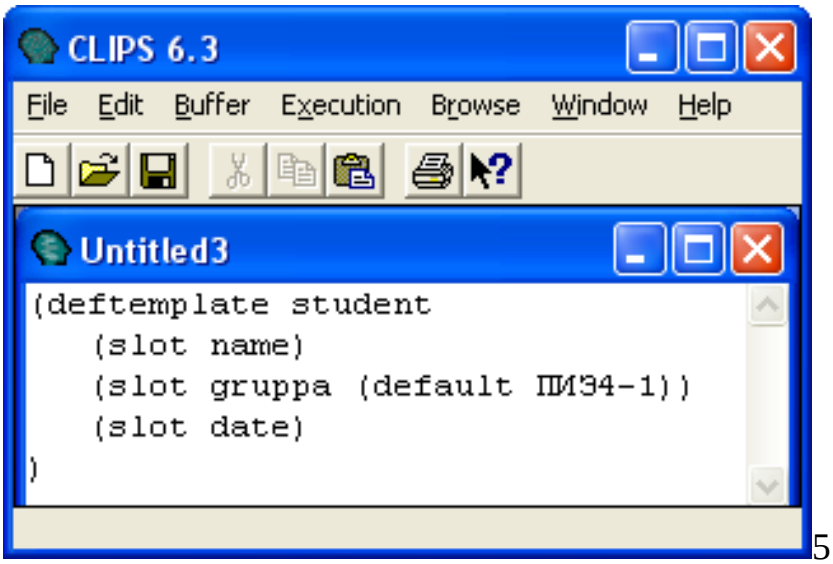

Рис. 15 Описание шаблона для неупорядоченных фактов

Сохраните файл под именем lab2-1.clp.

Для создания списка фактов создайте еще один текстовый файл и запишите 3 факта:

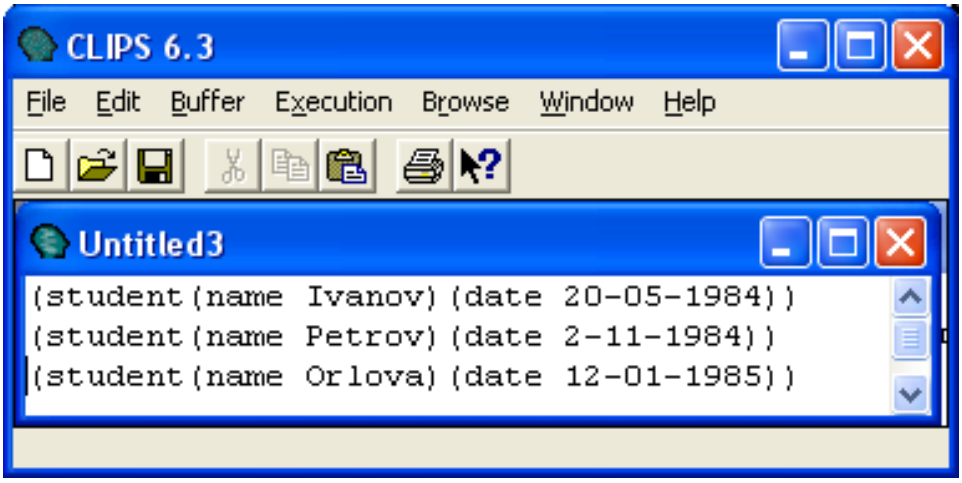

Рис. 16 Список неупорядоченных фактов

Сохраните файл под именем lab2-1-facts.clp. Исходные файлы подготовлены.

Прежде чем загрузить список фактов, необходимо загрузить в систему шаблон, описанный в файле lab2-1.clp. Для этого загрузите данный файл с помощью команд главного меню **File->Load…** Если после выполнения команды выведено сообщение TRUE, можно приступить к загрузке фактов с помощью команды **load-facts:**

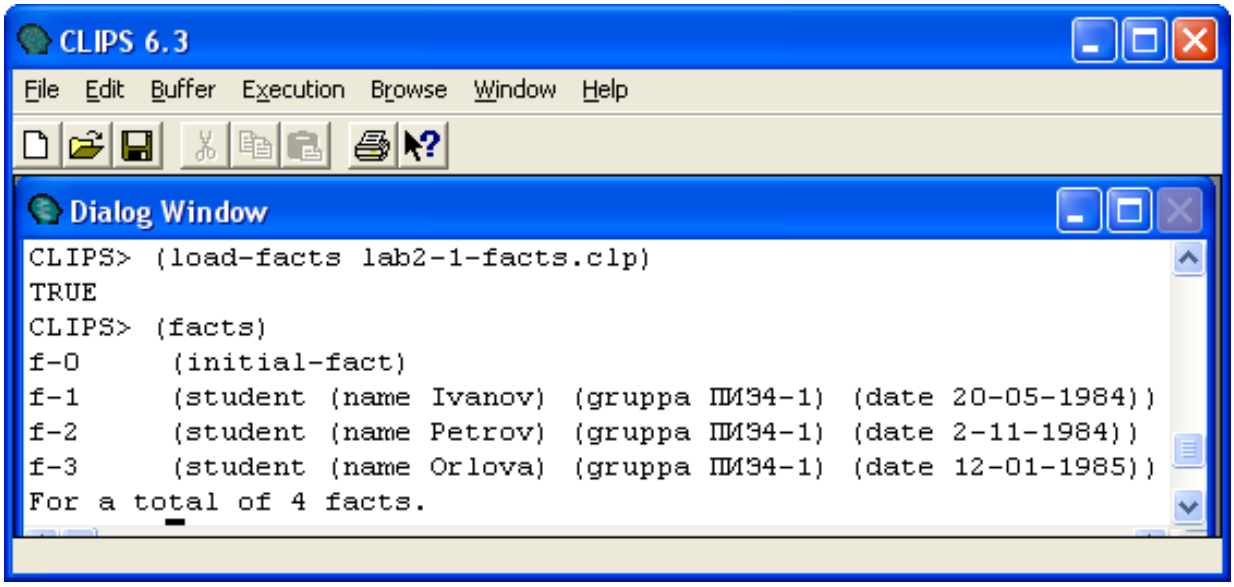

Рис. 17 Просмотр списка фактов после загрузки из текстового файла lab2-1-facts.clp

Для изменения в факте с индексом 2 значение слота date, наберите команду **(modify 2 (date 12-02-1985))** и просмотрите полученный список с помощью команды (**facts**) (рис.18):

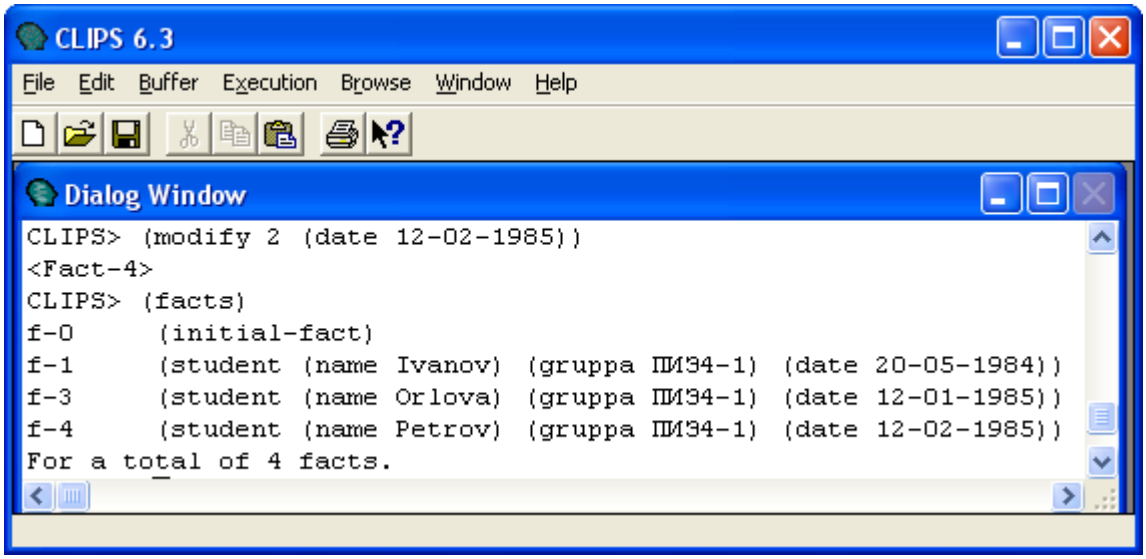

Рис. 18 Результат модификации факта f-2

Для дублирования факта с индексом 1, наберите команду **(duplicate 1 (name Sidorov))** и просмотрите полученный список с помощью команды **(facts):**

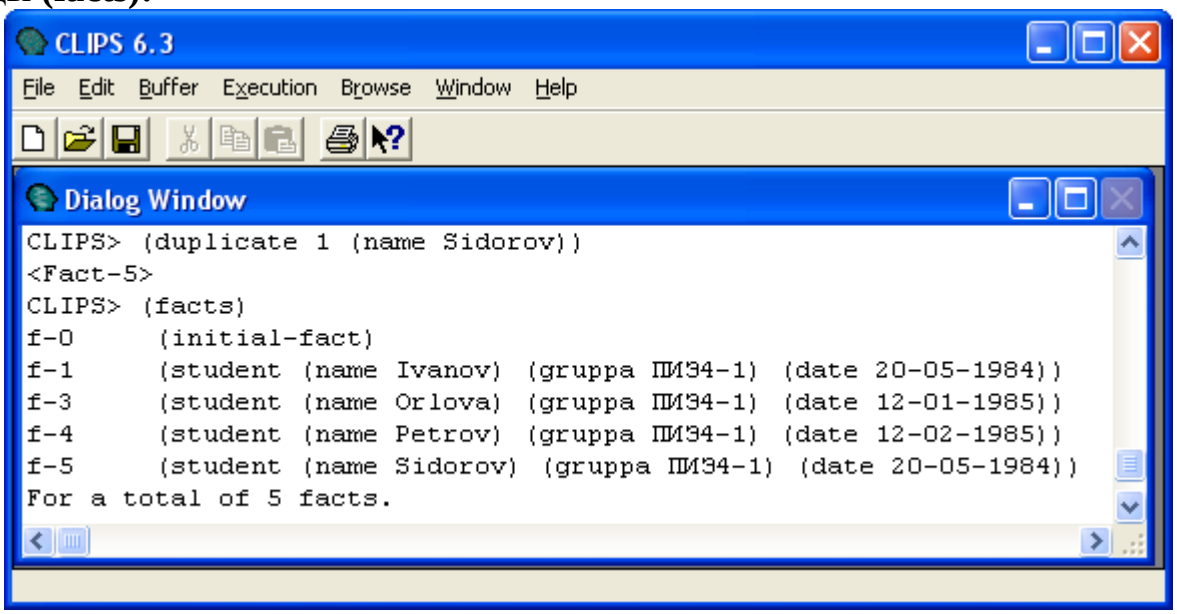

Рис. 19 Результат дублирования факта f-1

Сохраните новый список фактов:

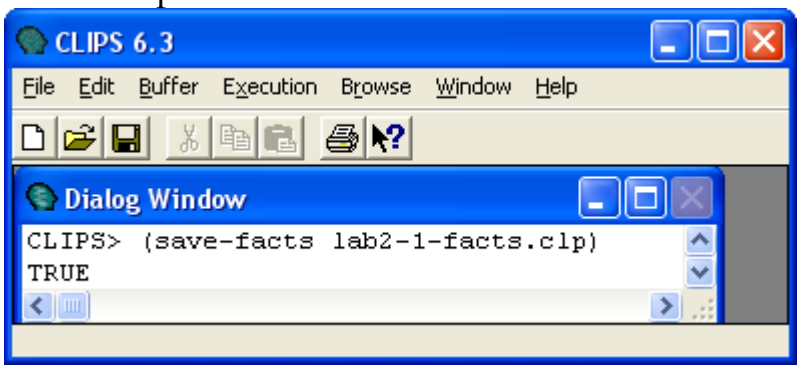

Рис. 20 Сохранение списка фактов в текстовый файл lab2-1-facts.clp

Для проверки очистите текущий список фактов, загрузите файл lab2-1.clp и сохраненные в файле lab2-1-facts.clp факты. Просмотрите список фактов:

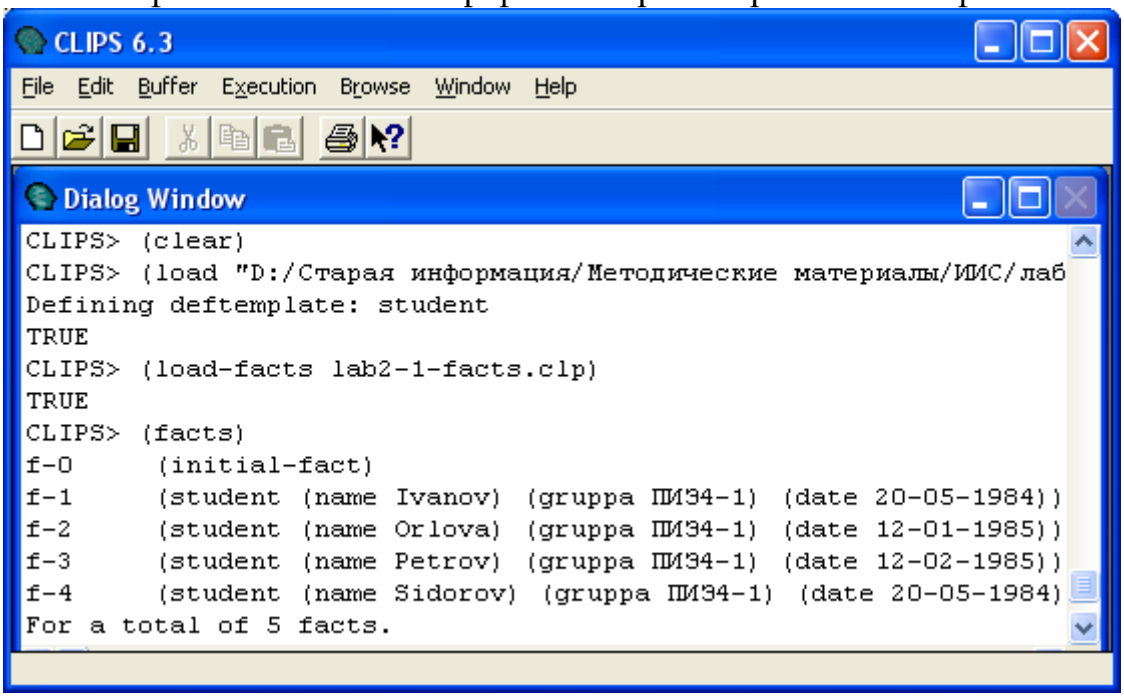

Рис. 21 Просмотр списка фактов после загрузки из текстового файла lab2-1-facts.clp

Контрольные вопросы

1. Какая конструкция используется для создания неупорядоченных (шаблонных) фактов? Поясните ее структуру.
2. Какая функция применяется для модификации неупорядоченного факта?
3. Какая функция используется для копирования неупорядоченного факта?
4. Какая функция используется для сохранения текущего списка фактов в текстовый файл?
5. С помощью какой функции можно загрузить факты из текстового файла?

Задания для самостоятельной работы

1. А) Создать файл, содержащий три неупорядоченных факта вида **(client (name <фамилия>) (in <инд.номер>) (citi <город проживания>))**. Значений слота citi задать по умолчанию. Загрузить данные факты. В факте с индексом 1 изменить значение слота citi, а в факте с индексом 2 изменить фамилию. Факт f-3 продублировать, изменив in. Сохранить полученный список фактов.

Б). Загрузить полученный список фактов. Добавить 2 новых неупорядоченных факта вида, используя функцию assert. Удалить факты с индексами 1, 3, Просмотреть полученный результат. Сохранить полученный список фактов в другой файл.

2. А) Создать файл, содержащий три неупорядоченных факта вида **(sotrudnik (name <фамилия>) (otdel <отдел>) (st <стаж работы>))** Значений слота otdel задать по умолчанию. Загрузить данные факты. В факте с индексом 1 изменить значение слота otdel, а в факте с индексом 3 изменить стаж работы. Факт f-2 продублировать, изменив фамилию. Сохранить полученный список фактов.

Б) Загрузить полученный список фактов. Добавить 2 новых неупорядоченных факта вида, используя функцию assert. Удалить факты с индексами 2, 4. Просмотреть полученный результат. Сохранить полученный список фактов в другой файл.

1.4. ПРАВИЛА. ИСПОЛЬЗОВАНИЕ УСЛОВНЫХ ЭЛЕМЕНТОВ В ЗАПИСИ ПРАВИЛ

Основные сведения

Одним из основных методов представления знаний в CLIPS являются правила. Правила используются для представления эвристик, определяющих ряд действий, которые необходимо выполнить в определенной ситуации. Разработчик программы определяет совокупность правил, которые используются совместно для решения проблемы.

Для задания правил используется конструкция *defrule* со следующим синтаксисом:

(defrule <имя_правила> ["<комментарий>"]
<условный элемент>* *; Левая часть правила (антецедент)*
=>
<действие>*) *; Правая часть правила (консеквент)*

Левая часть правила представляет собой ряд *условий (условных элементов),* которые должны выполняться, чтобы правило было применимо. В CLIPS принято считать, что условие выполняется, если соответствующий ему факт присутствует в списке фактов.

Одним из типов условных элементов может быть *образец.* Образцы состоят из набора ограничений, которые используются для описания того, какие факты удовлетворяют условию, определяемому образцом.

Процесс сопоставления фактов и образцов выполняется блоком вывода CLIPS, который автоматически сопоставляет образцы, исходя из текущего состояния списка фактов, и определяет, какие из правил являются применимыми. Если все условия правила выполняются, то оно активируется и помещается в список активированных правил, который является планом решения задачи и называется *агендой*.

Правая часть правила представляет собой совокупность действий, которые должны быть выполнены, если правило применимо. Действия, описанные в применимых правилах, выполняются тогда, когда блок вывода CLIPS получает команду начать выполнение применимых правил. Если существует множество применимых правил, то для того, чтобы выбрать одно правило, действия которого должны быть выполнены, блок вывода использует стратегию разрешения конфликтов.

Действия, описанные в выбранном правиле, выполняются (при этом список применимых правил может измениться), а затем блок вывода выбирает другое правило и т.д. Этот процесс продолжается до тех пор, пока не остается ни одного применимого правила, т.е. пока список активированных правил не окажется пуст.

Правила можно сравнить с операторами типа if-then процедурных языков программирования. Однако условие if-then оператора в процедурном языке про-

веряется только тогда, когда программа передает ему управление. С правилами ситуация иная. Блок вывода постоянно отслеживает все правила, условия которых выполняются, и, таким образом, правило может быть выполнено в любой момент, как только оно становится применимым.

В Clips используются следующие виды условных элементов:

test – предоставляет возможность наложения дополнительных ограничений на слоты факта. Внутри элемента TEST могут быть выполнены различные логические операции.

or позволяет активизировать правило любым из нескольких заданных условных элементов.

Условные элементы записи в левой части правила объединены неявно заданным условным эементом **and**. Явное задание **and** используется для группировки условных элементов вместе с OR.

Условный элемент **not** позволяет отрицать утверждения. **not** удовлетворяется, только если условный элемент, который он содержит, не удовлетворяется.

exists – позволяет определять, существует ли хотя бы один набор данных(фактов), соответствующий заданному условию.

forall – позволяет определить, что некоторое условие выполняются для всех заданных условных элементов.

logical – используется для установления логической связи между данными в левой части правила и данными в правой части, тем самым обеспечивая механизм поддержки достоверности для созданных правилом фактов. Если удаляются данные, которые поддерживают некоторые другие данные, то зависимые данные тоже автоматически удаляются.

Если левая часть правила пуста, то для его активации необходимо наличие в списке фактов исходного факта *(initial-fact).* Такие безусловные правила часто используются для того, чтобы инициировать работу программы. Поэтому перед запуском таких программ необходимо произвести сброс состояния среды CLIPS.

Как и в других языках программирования, в CLIPS для хранения значений используются *переменные*. В отличие от фактов, которые являются статическими, или неизменными, содержание переменной динамично и изменяется по мере того, как изменяется присвоенное ей значение.

Идентификатор переменной в общем случае выглядит следующим образом: **?<имя_переменной>.** Например, ?a, ?name? , ?k1.

Кроме значения самого факта, переменной может быть присвоено значение адреса факта. Это может оказаться удобным при необходимости манипулировать фактами непосредственно из правила. Для такого присвоения используется комбинация "<-".

Как правило, для запуска программы на CLIPS необходимо выполнить следующую последовательность действий:

- загрузить конструкции из файла с помощью команды *load*;
- выполнить команду *reset*;
- выполнить команду *run*.

Пример.

Имеются факты, отражающие результат прошедшей сессии, вида:

(ocenca <фамилия> <оценка1> < оценка2> < оценка3>)

Создать следующие правила:

1) начисления стипендии студентам, сдавшим сессию без троек. В результате выполнения правила должны быть добавлены факты вида: **(stip <фамилия> 1000)**
2) начисления повышенное стипендии студентам, сдавшим все экзамены на «отлично». В результате выполнения правила должны быть добавлены факты вида: **(stip <фамилия> 1500)**
3) перевода на следующий курс для студентов, сдавшим все экзамены. В результате выполнения правила должны быть добавлены факты вида: **(pereveden <фамилия>)**
4) определяющее все ли студенты, сдававшие сессию, переведены. В результате выполнения правила должна выводиться соответствующее сообщение.

Решение:

1. Запустить Clips, открыть новый файл и набрать текст программы:

```
; определяем исходные факты
(deffacts vedomost
        (ocenca Ivanov  5 5 5)
        (ocenca Petrov  4 5 4)
        (ocenca Sidorov 3 3 4)
)
; правило начисления стипендии студентам, сдавшим сессию без троек.
(defrule stipendia
    (ocenca ?name  ?o1  ?o2  ?o3)      ; ищем факт, соответствующий указанному
образцу
    (test (or (= ?o1  5) (= ?o1 4)))       ; если каждая из оценок 4 или 5
    (test (or (= ?o2  5) (= ?o2 4)))
    (test (or (= ?o3  5) (= ?o3 4)))
=>
    (assert (stip  ?name  1000)   ; то добавляем факт о начислении стипендии)
)
; правило начисления стипендии отличникам
; т.к. в результате выполнения 1-го правила отличникам была начислена обычная
;стипендия, то следует сначала удалить эти факты, а затем добавить новые –
о повышенной стипендии
(defrule stip_otl
     (ocenca ?name  ?o1 ?o2 ?o3)    ; ищем факт, соответствующий образцу
     (test (= 5 ?o1 ?o2 ?o3))          ; если все оценки 5
      ?n<- (stip  ?name  1000)        ; n – индекс факта о начислении обычной
                                      ;  стипендии  отличнику
     =>
     (retract ?n)                      ; удаляем факт
     (assert (stip ?name 1500))        ; добавляем новый
```

```
)
; правило, добавляющее факты о студентах, которые были переведены на следующий курс
(defrule perevod
    (ocenca ?name  ?o1  ?o2  ?o3) ; ищем факт, соответствующий образцу
    (test (<> 2 ?o1 ?o2  ?o3))        ;  если каждая оценка не двойка
=>
    (assert (pereveden  ?name ))          ; добавляем новый факт
)
; правило, определяющее, что все студенты переведены на следующий курс
(defrule all_student_perevod
        (forall (ocenca ?name  ?o1  ?o2  ?o3) ; если для всех студентов
                (pereveden  ?name ))              ; существуют факты о их переводе
        =>
       (printout t " vse studenty   perevedeny!" crlf))  ; то вывод сообщения
```

2. Сохранить файл, например, под именем lab4.clp, затем загрузить (**File | Load…**). Если в диалоговом окне выведено сообщение TRUE, то запустить на исполнение. Если появилось сообщение FALSE, то следует исправить ошибки

В результате выполнения программы в диалоговом окне должно быть выведено сообщение:

vse studenty perevedeny!

В окне Facts (Main) должен быть отображен список всех фактов, полученных в результате выполнения программы (рис.22).

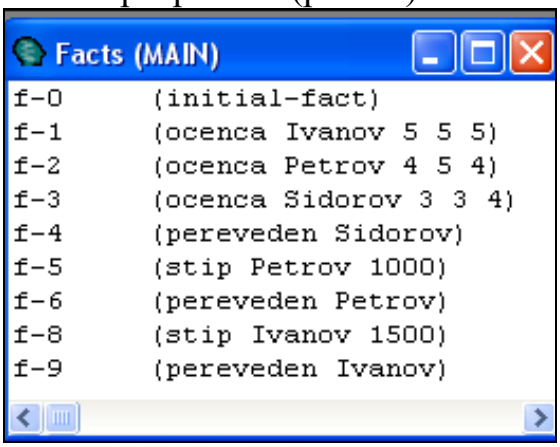

Рис.22 Текущий список фактов после выполнения программы

Контрольные вопросы

1. Какая конструкция используется для создания правил? Поясните ее структуру.
2. Какой условный элемент предоставляет возможность наложения дополнительных ограничений в левой части правила?
3. Поясните назначение условных элементов NOT, AND, OR.
4. С какой целью используются условные элементы EXISTS и FORALL?
5. Какой условный элемент используется для установления логической связи между данными в левой части правила и данными в правой части?
6. Как записываются имена переменных в CLIPS?

7. Какую последовательность действий необходимо выполнить для запуска программы на CLIPS?

Задания для самостоятельной работы

1. С помощью deffacts создайте список из 5 упорядоченных фактов вида:
(sotrudnik <фамилия> <стаж работы> <кол-во детей> <оклад>)

Определите следующие правила:

1) начисления премии тем сотрудникам, у которых стаж работы не менее 5 лет или более 2-х детей. В результате выполнения правила должны быть добавлены факты вида: **(premia < фамилия > <сумма премии>).** Премия вычисляется в размере 20% от оклада.
2) Определяющее, все ли сотрудники получили премию. В результате выполнения правила должно выводиться соответствующее сообщение.
3) Определяющее, есть ли среди сотрудников ветераны труда (ветераном труда считается сотрудник, у которого стаж работы более 20 лет). В результате выполнения правила должно выводиться соответствующее сообщение.

2. С помощью deffacts создать список из 5 упорядоченных фактов вида:
(tovar <наименование> <цена> <кол-во> <год производства> <страна-производитель>)

Определите следующие правила:

1) Уценки тех товаров, которые были выпущены до 2010 г. или количество которых менее 4 единиц. В результате выполнения правила должны быть добавлены факты вида: **(ucenka < наименование > <сумма уценки>).** Уценка вычисляется в размере 15% от цены товара.
2) Дополнительной наценки на импортные товары, выпущенные в текущем году. В результате выполнения правила должны быть добавлены факты вида: **(nacenka < наименование > <сумма доп.наценки>).** Наценка вычисляется в размере 7% от цены товара.
3) Определяющее, есть ли в базе данных товары российского производства. В результате выполнения правила должно выводиться соответствующее сообщение.

1.5. ПРОЦЕДУРНЫЕ ФУНКЦИИ

Основные сведения

В CLIPS имеются функции, которые обеспечивают возможность *процедурного* программирования. Они могут использоваться для создания отрезков процедурного кода в правилах. Наиболее часто используются следующие процедурные функции:

1. (bind <имя-переменной> <выражение>) - создание и связывание переменных

Первый аргумент функции *<имя-переменной>* является именем глобальной или локальной переменной, созданной в правиле или функции.

Если параметр *<выражение>* не определен, то выполнение функции **bind** не оказывает на локальные переменные никакого влияния, а глобальные переменные получают при этом значения по умолчанию.

Функция **bind** возвращает значение *false* при неудачном исходе операции. Во всех остальных случаях функция возвращает присвоенное переменной значение.

Если заданная переменная еще не определена, она будет создана и связана с соответствующим значением.

2. (if <выражение> then <действия1> [else<действия2>]) - ветвление

Если условие, заданное с помощью выражения, выполняется (т. е. не является ложным), выполняются действия, определенные в блоке *then*. В противном случае производятся действия из блока *else*. В каждом из таких блоков может быть задано любое количество действий. Любой блок может содержать вложенную конструкцию if.. .then.. .else. Блок *else* является необязательным.

3. (while <условие> do <действия>) - цикл с предусловием

Проверка условия выполняется перед выполнением тела цикла. Цикл выполняется, пока условие (логическое выражение) остается истинным. В теле цикла может содержаться произвольное количество действий, включая вложенные циклы или функцию if.

4. (loop-for-count <диапазон> do <действия>) - цикл с заданным числом повторений

Цикл loop-for-count производит указанные действия некоторое количество раз согласно заданному параметру *<диапазон>*. Если минимальное значение параметр не задано, ему автоматически присваивается 1. CLIPS предусматривает наличие параметра *<переменная-цикла>,* определяющего имя локальной переменной, которая может использоваться в теле цикла для определения текущего числа итераций. Использование переменной цикла после завершения работы функции loop-for-count вне тела цикла запрещено. Функции break и return могут быть использованы для экстренного прерывания работы цикла. В теле цикла может содержаться произвольное количество действий, включая вложенные циклы или функцию if.

5. (switch <выражение> (case<выражение> then <действие>) [(default <действие>)] - множественный выбор

Для эффективного применения функции switch необходимо наличие, по крайней мере, трех альтернативных групп действий, зависящих от заданного условного выражения.

Функция switch в первую очередь проводит вычисление аргумента *<выражение>*, а затем сравнивает его со всеми условиями ветвления по очереди. Если значение заданного выражения совпадает с одним из условий ветвления, выполняются соответствующие действия, и работа функции завершается. Если совпадений обнаружить не удалось, функция возвращает результат последнего сеанса выбора (если таковой имеется) или значение false.

6. (break) -прерывание

Функция break прерывает текущую итерацию циклов while и loop-for-count

7. (return [<выражение>]) - прерывает выполняющуюся функцию, цикл, правило, обработчик сообщения и т. д.

Если функция return вызывается без аргументов, функция не возвращает никакого значения. Однако если аргумент присутствует, return возвращает результат вычислений, который присваивается значению прерванной функции, цикла или обработчика сообщения.

Пример 1.

Создать правило **perevod,** которое запрашивает количество студентов, вводит данные о них и добавляет факты вида (pereveden <имя> <курс>)

Решение:

1. Запустите Clips, откройте новый файл и наберите текст программы:

```
(defrule   perevod
  =>
    (printout t "n=" )                    % выводим строку подсказки
    (bind ?n (read))                      % вводим количество студентов
    (loop-for-count ?n do                 % повторить n раз
        (printout t "FIO:" )
        (bind ?name (readline))           % вводим ФИО
        (printout t "kurs:" )
        (bind ?k (read))                  % вводим курс
        (while (or (< ?k 2) (> ?k 5))do   % если курс задан неверно: меньше 2 или
                                                                  больше 5
             (printout t "kurs:" )        % то ввод повторяется
            (bind ?k (read))
        )
        (assert (student ?name ?k))       % добавляем факт
     )
)
```

2. Сохраните файл, например, под именем lab5-1.clp, затем загрузите его (File | Load…). Если в диалоговом окне выведено сообщение TRUE, то запустите на исполнение. Если появилось сообщение FALSE, то следует исправить ошибки
3. Протестируйте программу

Пример 2.

Создать правило **prostoe,** которое определяет, является простым или составным введенное натуральное число

Решение:

```
 (defrule prostoe
  =>
   (printout t "n=")
   (bind ?n (read))                       % вводим число
   (bind ?t (div ?n 2))          % определяем возможный наибольший делитель числа-t
   (bind ?s "prostoe")                    % предполагаем, что число простое
   (loop-for-count (?i 2 ?t) do           % перебираем значения i от 2(наименьший де-
литель) до t
```

```
      ( if (= (mod ?n ?i) 0) then        % если делитель найден
                (bind ?s "sostavnoe")  % то число составное
      )
   )
  (printout t  ?s crlf)              % выводим результат
)
```

Пример 3.

Создать правило summa**,** которое находит сумму последовательных натуральных чисел от 1 до 10.

Решение:

```
(defrule summa
  =>
   (bind ?sum 0)                      % обнуляем сумму
   (loop-for-count (?n 1  10) do      % перебираем n от 1 до 10
     (bind ?sum (+ ?sum ?n))          % добавляем к сумме очередное n
   )
  (printout t "sum=" ?sum crlf)
)
```

Контрольные вопросы

1. Какая функция позволяет создать переменную и задать ее значение?
2. Какая функция реализует ветвление в CLIPS? Поясните ее структуру.
3. Какие функции используются для организации циклов? Объясните механизм работы каждой из них.
4. Какая функция обеспечивает множественный выбор? Поясните ее структуру.

Задания для самостоятельной работы

1. Создать правило **zachisl,** которое запрашивает количество студентов, вводит ФИО каждого и добавляет факты вида (student <фамилия> 1).
2. Создать правило **spisok_sotr,** которое запрашивает количество сотрудников, вводит ФИО и должность каждого и добавляет факты вида (sotrudnik <фамилия > <должность >).
3. Создать правило **factorial** для вычисления n!
4. Создать правило для вычисления суммы $S=1/2^2+1/4^2+1/6^2+\cdots+1/(2n)^2$
5. Первоначальная сумма вклада в банк составляет S0 руб., процентная ставка – p% годовых. Через сколько лет сумма вклада более чем в 2 раза превысит первоначальную? Создать соответствующее правило
6. Создать правило **nod** для нахождения наибольшего общего делителя двух натуральных чисел.

Подсказка: можно использовать алгоритм Евклида:

1) пока числа не равны, заменять большее из них разностью большего и меньшего.
2) когда числа станут равны, любое из них взять в качестве НОД

1.6. ФУНКЦИИ ПОЛЬЗОВАТЕЛЯ

Основные сведения

Под функцией в CLIPS понимается фрагмент исполняемого кода, с которым связано уникальное имя и который возвращает полезное значение или имеет полезный побочный эффект (например, вывод информации на экран).

Существует несколько типов функций. Пользовательские и системные функции - это фрагменты кода, написанные на внешних языках (например, на C) и связанные со средой CLIPS. Системными называются те функции, которые были определены изначально внутри среды CLIPS. Пользовательскими называются функции, которые были определены вне CLIPS.

Конструкция *deffunction* позволяет определять новые функции непосредственно в среде CLIPS с использованием синтаксиса CLIPS. Функции, определенные таким образом, выглядят и работают подобно остальным функциям, однако они выполняются не напрямую, а интерпретируются средой CLIPS.

Синтаксис:

(deffunction <имя_функции>
[<комментарии>]
<обязательные параметры>
[<групповой параметр>]
<действия>
)

Обязательные параметры определяют минимальное число аргументов, задаваемых при вызове функции. Если групповой параметр задан, то количество аргументов может быть больше. Определение функции может содержать только один групповой параметр, который обозначается символом $. Все необязательные аргументы группируются в одно значение составного поля.

Одна функция пользователя может вызывать другую функцию. При этом вызываемая функция должна быть определена до вызывающей.

Функции могут быть рекурсивны, т.е. вызывать сами себя. Необходимо помнить, что рекурсивная функция должна обязательно содержать условие, определяющее прекращение рекурсии.

Пример1.

Создать функцию, позволяющую пользователю задать вопрос и получить ответ типа да/нет.

Решение:

1. Создайте новый файл и наберите программу:

; *функция* ***ask-question*** *с обязательным параметром ?question (вопрос) и групповым параметром $?allowed-values (список допустимых ответов)*

```
(deffunction ask-question (?question $?allowed-values)
  (printout t ?question)                    ; вывод вопроса
  (bind ?answer (read))                     ; ввод ответа
  (if (lexemep ?answer)                     ; если введено строковое значение
```

```
        then (bind ?answer (lowcase ?answer)))  ; то символы строки преобразуем
                                                  в строчные
    ; пока пользователь не введет ответ, принадлежащий списку   допустимых
значений, повторять ввод
     (while (not (member ?answer ?allowed-values)) do
        (printout t ?question)
        (bind ?answer (read))
        (if (lexemep ?answer)
           then (bind ?answer (lowcase ?answer))))
      ?answer
    )

    ; правило, задающее вопрос пользователю и в случае утвердительного отве-
та добавляющее факт в базу данных
    (defrule yes-no
      =>
    ; вызов функции и копирование полученного ответа в переменную  ?response
    (bind ?response (ask-question "color blue?(yes/no)"  yes no y n))
     (if (or (eq ?response yes) (eq ?response y)) ; если ответ y  или yes
              then   (assert (color blue))   ; то добавляем факт
     )
    )
```

2. Протестируйте программу для разных вариантов ответов, просматривая каждый раз базу данных.

Пример 2.

Создать функцию для определения периметра прямоугольного треугольника по длинам его катетов. Для вычисления гипотенузы создать отдельную функцию и использовать ее при определении периметра. Найти периметр прямоугольного треугольника для введенных значений катетов (создать соответствующее правило)

```
    ;функция для вычисления гипотенузы треугольника с катетами  a и b
    (deffunction gipotenuza (?a ?b)
      (sqrt (+ (** ?a 2) (** ?b 2)))  ;определяем значение функции
    )
    ;функция для вычисления периметра прямоугольного треугольника с ка-
тетами  a и b
    (deffunction perimetr (?a ?b)
      (bind ?c (gipotenuza ?a ?b))  ;вычисляем гипотенузу с помощью объявленной
                                    ; функции  gipotenuza
      (+ ?a ?b ?c) )                ;определяем значение функции perimetr

    ; правило для ввода значений катетов и вычисления периметра с использова-
нием созданных функций
    (defrule perim_treug
    =>
```

```
(printout t "a=")
(bind ?a (read))                ;ввод значения катета a
(printout t "b=")
(bind ?b (read))                ;ввод значения катета b
(bind ?p (perimetr ?a ?b))      ; вычисление периметра
(printout t "p=" ?p crlf) )     ; вывод результата
```

Пример 3.

Создать рекурсивную функцию для вычисления факториала и правило, позволяющее вводить произвольное натурального числа и находить его факториал.

Факториал числа вычисляется как произведение натуральных последовательных чисел от 1 до n

n!=1*2*3*...*(n-1)*n.

Рекурсивно факториал можно определить следующим образом:

$$n!=\begin{cases}1, \text{если } n=0\\ (n-1)!*n, \text{если } n>0\end{cases}$$

Например, 4!= 3!*4=2!*3*4=1!*2*3*4*= 0!*1*2*3*4*= 1*1*2*3*4=24

Решение:

```
; функция  fact с аргументом  n
(deffunction fact (?n)
   (if (= ?n 0) then 1                    ; если n=0 то n!=1
            else (* ?n (fact (- ?n  1)))  ; иначе умножаем n на (n-1)!
   )
)
; правило для ввода числа n и вычисления n!
(defrule fact_n
 =>
  (printout t "n=")
  (bind ?n (read))                  ;ввод числа
  (bind ?fn (fact ?n))              ; вызов функции с фактическим параметром n
  (printout t ?n "!=" ?fn crlf) )   ; вывод результата
```

Контрольные вопросы

1. Какие типы функций существует в CLIPS?
2. Какая конструкция CLIPS позволяет создать новые функции? Поясните ее структуру.
3. С какой целью используется групповой параметр при описании функции пользователя?
4. Какие функции называются рекурсивными?

Задания для самостоятельной работы

1. Создать функцию для вычисления длины отрезка по заданным координатам его концов (a_1,a_2) и (b_1,b_2), $D=\sqrt{(b_1-a_1)^2+(b_2-a_2)^2}$. Используя данную функцию найти периметр треугольника по координатам его вершин.

2. Создать функцию для вычисления площади треугольника по длинам его сторон, $S = \sqrt{p\,(p-a)(p-b)(p-c)}$. Используя данную функцию сравнить площади двух треугольников, заданных длинами своих сторон.
3. Определить функцию для вычисления значения Y =f(X) с выбором формулы в соответствии с условием

$$Y = \begin{cases} 1 - x^2, x < c, \\ 0, c \le x < d; \\ 1 + x^2, x \ge d \end{cases}$$

4. Определить функцию для вычисления значения Y =f(X) с выбором формулы в соответствии с условием

$$Y = \begin{cases} \dfrac{|x-a|}{x^2}, x < 0, \\ 0, x = 0, \\ \sin(|x+a|), x > 0 \end{cases}$$

5. Создайте логическую функцию, позволяющую пользователю задать вопрос и получить ответ типа да/нет. При утвердительном ответе функция возвращает значение TRUE, а при отрицательном - FALSE.
6. Создайте рекурсивную функцию для вычисления суммы вида

$$S = \sum_{i=1}^{n} \frac{1}{i}$$

7. Создайте рекурсивную функцию для вычисления суммы вида

$$S = \sum_{i=1}^{n} \frac{1}{2i+1}$$

8. Создайте рекурсивную функцию для вычисления суммы вида

$$S = \sum_{i=1}^{n} \frac{1}{i^2}$$

1.7. ОБРАБОТКА СТРОКОВЫХ ВЕЛИЧИН

Основные сведения

Для работы со строками в CLIPS имеются функции:

1. **str-cat <список строк>** - объединяет строки и возвращает ее в качестве результата.

Аргументы этой функции должны принадлежать одному из следующих типов: symbol, string, float, integer или instance-name.

2. **sub-string <индекс первого символа> < индекс последнего символа > <строка>** - выделяет из строки подстроку

Первый аргумент функции задает индекс первого символа выделяемой подстроки, а второй аргумент — последнего символа. Сама строка определяется последним аргументом. Если первый аргумент больше второго, функция возвращает 0.

3. **str-index <подстрока> <строка>** - возвращает позицию заданной подстроки внутри строки.

Результат выполнения функции относится к целому типу и равен индексу первого символа подстроки. В случае если искомая подстрока не была найдена, функция str-index возвращает значение false.

4. **upcase <строка>** преобразует все символы строки в верхний регистр
5. **lowcase <строка>** преобразует все символы строки в нижний регистр
6. **str-compare <строка1> <строка2>** сравнивает две строки.

Сравнение выполняется посимвольно до конца строк (если строки равны), либо пока не встретятся два неравных символа. Функция возвращает целое число, представляющее результат сравнения. Если сравниваемые строки равны, результатом является 0. Если первая строка меньше второй, результат — целое число меньше 0, и, наконец, если первая строка больше второй, возвращаемый результат – целое число больше 0. Аргументы функции должны принадлежать типу string или symbol.

7. **str-length < строка >** длина строки

Результат работы этой функции возвращается в виде целого числа

Пример 1.

Имеются факты вида :

```
(ocenka ivanov 4 5 5)
(ocenka sidorov 3 5 4)
(ocenka paramonova 4 4 4)
(ocenka orlov 5 5 4)
```

Вывести данные, выравнивая длину фамилий до 15 символов за счет добавления справа пробелов.

Решение:

```
; исходные факты
(deffacts ocenki
  (ocenka ivanov 4 5 5)
  (ocenka sidorov 3 5 4)
  (ocenka paramonova 4 4 4)
  (ocenka orlov 5 5 4)
 )
; правило, формирующее список
(defrule spisok
 (ocenka ?name ?o1 ?o2 ?o3)      ; ищем факт, соответствующий образцу
=>
 (bind ?n (- 15 (str-length ?name)))   ;определяем кол-во добавляемых пробелов
```

```
  (loop-for-count (?i 1 ?n) do     ; в цикле добавляем по одному нужное число
                                                              пробелов справа
    (bind ?name (str-cat ?name " "))
   )
  (printout t ?name ?o1 "  " ?o2 "  " ?o3 crlf)                ; вывод
  )
```

Пример2.

Создать правило для подсчета заданных символов во введенной строке.

Решение:

```
(defrule kol_sym
=>
(bind ?str (readline))                ;ввод строки
(bind ?c (readline))                  ;ввод символа
(bind ?n (str-length ?str))           ;вычисляем длину строки
(bind ?k 0)                           ; обнуляем счетчик символов
(loop-for-count (?i 1 ?n) do          ;перебираем все символы строки
   (bind ?c1  (sub-string ?i ?i ?str))  ;выделяем i-й символ
   (bind  ?d   (str-compare  ?c  ?c1))  ;сравниваем  текущий символ строки с
                                                              заданным
          (if  (= ?d 0) then             ;если символы равны
      (bind ?k (+ ?k 1))                 ;то увеличиваем значение счетчика на 1
    )
)
(printout t "k=" ?k crlf)             ;вывод результата
)
```

Пример3.

Создать функцию, удаляющую из заданной строки первое вхождение указанной подстроки

Решение:

```
; функция  str-del с аргументами ?substr (подстрока ) и  ?str(строка)
(deffunction str-del (?substr ?str)
   (bind ?n1 (str-length ?substr))              ;вычисляем длину подстроки
   (bind ?n2 (str-length ?str))                 ;вычисляем длину строки
   (bind  ?k  (str-index  ?substr  ?str))       ;определяем позицию вхождения
                                                        подстроки в строку
   (if  (> ?k 0) then                           ; если подстрока содержится в строке
         (bind ?st1  (sub-string 1 (- ?k 1) ?str))   ; вырезаем часть строки слева от
                                                              подстроки
      (bind ?st2  (sub-string (+ ?k ?n1) ?n2 ?str))  ; вырезаем часть строки справа
                                                              от подстроки
   (bind ?str (str-cat ?st1 ?st2))                 ; объединяем полученные части
 )
```

```
?str )
; правило substr_delete для ввода строки и подстроки  и вызова функции str-del
(defrule substr_delete
=>
(bind ?str (read))
(bind ?substr (read))
(bind ?str (str-del ?substr ?str))
(printout t "str=" ?str crlf))
```

Контрольные вопросы

1. Какая функция объединяет строки?
2. Назовите функцию, позволяющую выделить из строки подстроку.
3. Какая функция вычисляет длину строки?
4. Назовите функцию, позволяющую определить позицию вхождения подстроки в строку.
5. Какая функция используются для преобразования символов строки в верхний регистр?
6. Какая функция используются для преобразования символов строки в нижний регистр?
7. Назовите функцию, сравнивающую две строки. Какой результат возвращает функция в результате сравнения?

Задания для самостоятельной работы

1. Создайте правило, заменяющее в строке один указанный символ другим.
2. Создайте правило, заменяющее в заданной строке все пробелы на символ "_".
3. Создайте правило, удаляющее из заданной строки **все** вхождения указанной подстроки
4. Создайте функцию, добавляющую к заданной строке справа указанное количество символов ‘*’, и правило, которое вводит две строки и выравнивает их длины, добавляя к меньшей символы ‘*’.
5. Создайте функцию, позволяющую удалить из строки лишние пробелы, оставив по одному между словами.
6. Создайте функцию, позволяющую записать заданную строку в зеркальном отображении.

.

2. ПРОЕКТИРОВАНИЕ ЭКСПЕРТНЫХ СИСТЕМ В CLIPS

2.1. Постановка задачи

Экспертные системы (ЭС) – это системы искусственного интеллекта (интеллектуальные системы), предназначенные для решения плохоформализованных и слабоструктурированных задач в определенных проблемных областях на основе заложенных в них знаний специалистов-экспертов. В настоящее время ЭС внедряются в различные виды человеческой деятельности, где использование точных математических методов и моделей затруднительно или вообще невозможно. К ним относятся: медицина, обучение, поддержка принятия решений и управление в сложных ситуациях, различные деловые приложения и т.д.

Основными компонентами ЭС являются базы данных (БД) и знаний (БЗ), блоки поиска решения, объяснения, извлечения и накопления знаний, обучения и организации взаимодействия с пользователем. БД, БЗ и блок поиска решений образуют ядро ЭС.

Разработку экспертных систем в среде CLIPS рассмотрим на примере создания прототипа экспертной системы, позволяющей выявить неисправности двигателя автомобиля по внешним признакам. Помимо этого экспертная система должна также предоставлять пользователю соответствующие рекомендации по устранению неисправности. Класс решаемой задачи - задача диагностики.

2.2.Идентификация проблемной области

Разработку любой экспертной системы следует начинать с исследования предметной области и выделения основных сущностей, имеющих значение при решении конкретной задачи. В подавляющем большинстве случаев эту информацию получают при помощи эксперта – высококлассного специалиста в этой области.

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: работать нормально, работать неудовлетворительно или не работать.

2. Если двигатель работает нормально, то это означает, что он нормально вращается, система зажигания и аккумулятор находятся в норме и ни какого 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 (правило 7).

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

2.3. Формирование базы данных ЭС

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

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 (правило 1).

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 некоторых правилах (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 правилах 6 и 8 используется понятие - состояние аккумулятора. Аккумулятор может быть в одном из двух состояний: заряженным и разряженным.

Для представления в CLIPS всех перечисленных выше данных воспользуемся упорядоченными фактами.

Факты, описывающие состояние автомобиля и его узлов

```
;  Группа фактов,  описывающая состояние автомобиля
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
```

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

Создаваемая экспертная система должна предоставлять пользователю рекомендации, позволяющие устранить найденную неисправность. Из приведенных выше правил можно выделить рекомендации и представить их в виде фактов:

repair "Добавить топливо." (правило 5);
repair "Зарядите аккумулятор." (правило 6);
repair "Замените контакты." (правило 7 или правило 12);
repair "Почистите контакты." (правило 7 или правило 12);
repair "Замените распределительные провода." (правило 8);
repair "Замените катушку зажигания." (правило 8);
repair "Прочистите систему подачи топлива." (правило 9);
repair "Отрегулируйте зазоры между контактами." (правило 10);
repair "Отрегулируйте зажигание." (правило 11).

Если ремонт не требуется в принципе, будет выдана рекомендация:
repair "Ремонт не нужен."

Если же экспертная система не смогла поставить диагноз, то:
repair "Обратитесь в сервисную службу."

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 до факта.

Факты, описывающие мощность работы двигателя

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

2.4. Реализация диалога с пользователем

Для получения необходимой информации от пользователя создадим 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 этого создадим функцию **ask-question**, задающую пользователю произвольный вопрос и получающую ответ из заданного набора корректных ответов.

```
(deffunction ask-question (?question  $?allowed-values)
    (printout t ?question)      ; вывод вопроса
    (bind ?answer (read))       ; ввод ответа
    (if (lexemep ?answer)       ; если  введено строковое значение
      then                      ;то символы строки преобразуем в строчные
      (bind ?answer (lowcase ?answer)))
; пока пользователь не введет ответ, принадлежащий списку  допустимых значений, повторять ввод
 (while  (not  (member ?answer ?allowed-values))  do
(printout  t  ?question)
(bind ?answer  (read))
(if  (lexemep ?answer)
 then
(bind ?answer  (lowcase  ?answer))))
 ?answer
)
```

Функция имеет 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 логическую функцию **yes-or-no-p**, задающую пользователю вопрос и допускающую ответ в виде **да/нет**. С учетом реализации функции ask-question эта функция примет следующий вид:

```
(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 утвердительный 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.

2.5. Диагностические правила

Для упрощения реализации 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 (правило 1).

```
(defrule determine-engine-state ""
   (not (working-state engine ?))
   (not (repair ?))
 =>
(if (yes-or-no-p " Двигатель запускается (yes/no)?")
    then    ; если  "да", то уточняющий вопрос:
          (if (yes-or-no-p " Двигатель  работает нормально? (yes/no)?
            ; в зависимости от полученных ответов добавляются факты
              then
                (assert (working-state engine normal)) ;  работа двигателя нормальная
              else
                 assert (working-state engine unsatisfactory)))     ;работа двигателя
                                                                    неудовлетворительная
   else
      (assert (working-state engine does-not-start)))        ;двигатель не заводится
)
```

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.

```
(defrule determine-rotation-state ""
        (working-state engine does-not-start)
        (not (rotation-state engine ?))
        (not (repair ?))
=>
(if (yes-or-no-p " Двигатель вращается  (yes/no)? ")
        then                            ; если ответ "да", то добавляются факты
        (assert (rotation-state engine rotates))        ;двигатель вращается
        (assert (spark-state engine irregular-spark))   ; искра не регулярна
        else                                            ; иначе добавляются факты
        (assert (rotation-state engine does-not-rotate))    ;двигатель не вращается
        (assert (spark-state engine does-not-spark)))       ; искры нет
)
```

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 (правило 4).

Далее реализуем правило**,** определяющее наличие топлива в бак.

```
(defrule determine-gas-level ""
        (working-state engine does-not-start)
        (rotation-state engine rotates)
        (not (repair ?))
=>
(if  (not (yes-or-no-p " В баке имеется топливо (yes/no)? ")) ;?
 then                                                ; если ответ "нет", то
(assert (repair " Добавить топливо.")))             ; выдается рекомендация  )
```

Чтобы проверить, заряжен ли аккумулятор, создадим правило **determine-battery-state:**

```
(defrule determine-battery-state ""
(rotation-state engine does-not-rotate)
(not (charge-state battery ?))
(not (repair ?))
 =>
(if (yes-or-no-p " Аккумулятор заряжен (yes/no)? ") ;?
Then            ; если ответ "да", то
(assert (charge-state battery charged)) ;добавляется факт "аккумулятор заряжен"
else
(assert (repair " Зарядить аккумулятор.")) ;выдается рекомендация
(assert (charge-state battery dead))) ;добавляется факт "аккумулятор разряжен"
```

)

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 следующим образом:

```
(defrule determine-low-output ""
  (working-state engine unsatisfactory)
   ;    мощность работы двигателя еще не определена
  (not (symptom engine low-output | not-low-output))
  (not (repair ?))
=>
(if (yes-or-no-p " Выходная мощность двигателя низкая (yes/no)? ")
Then   ; если ответ "да", то добавляется факт
    (assert (symptom engine low-output)) ; «низкая мощность»
Else иначе добавляется факт
  (assert (symptom engine not-low-output))) «нормальная мощность»
)
```

Правило **determine-low-output** определяет, имеет ли место низкая мощность двигателя или нет.

```
(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 "Каково состояние контактов (norm/opal/zagr)? "
          norm opal zagr))
  (if (eq ?response opal)
     then
        ; контакты опалены - замените контакты
(assert (repair "Замените контакты."))
     else (if (eq ?response zagr)
            then
                  ; контакты загрязнены - почистите контакты
 (assert (repair "Почистите контакты."))))
```

)

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 правил 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).

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 (правило 8).*

```
(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 "Катушка зажигания пропускает ток (yes/no)? ")
    then                                          ; рекомендация
      (assert (repair "Замените распределительные провода."))
    else                                          ; рекомендация
      (assert (repair "Замените катушку зажигания.")))
)
```

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 (правило 9) и выполнение диагностических правил прекращается.*

```
(defrule determine-sluggishness ""
   (working-state engine unsatisfactory)
   (not (repair ?))
 =>
   (if (yes-or-no-p "Машина ведет себя инертно (yes/no)? ")
         then        ; рекомендация
   (assert (repair "Прочистите систему подачи топлива.")))
)
```

Правило **determine-misfiring** узнает - нет ли перебоев с зажиганием. Если это так, то необходимо отрегулировать зазоры между контактами (правило 10).

```
(defrule determine-misfiring ""
  (working-state engine unsatisfactory)
  (not (repair ?))
  =>
  (if (yes-or-no-p "Перебои с зажиганием есть(yes/no)? ")
     then                ; рекомендация
    (assert (repair "Отрегулируйте зазоры между контактами."))
    (assert (spark-state engine irregular-spark)))      ; Плохая искра
)
```

Правило **determine-knocking** узнает - не стучит ли двигатель. Если это так, то необходимо отрегулировать зажигание (правило 1)

```
(defrule determine-knocking ""
  (working-state engine unsatisfactory)
  (not (repair ?))
  =>
  (if (yes-or-no-p "Двигатель стучит (yes/no)? ")
     then                ; рекомендация
    (assert (repair "Отрегулируйте зажигание.")))
)
```

В качестве реализации правила 13 используем правило **no-repairs:**

```
(defrule no-repairs ""
 (declare (salience -10))
 (not (repair ?))
 =>
 (assert (repair "Обратитесь в сервисную службу."))
)
```

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.

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

```
(defrule normal-engine-state-conclusions ""
  (declare (salience 10))
  (working-state engine normal)    ; Если двигатель работает нормально
  =>
  (assert (repair "Ремонт не нужен."))     ; 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 неудовлетворительно
  (working-state engine unsatisfactory)
  =>
  (assert (charge-state battery charged))      ; аккумулятор заряжен
  (assert (rotation-state engine rotates))     ; двигатель вращается
)
```

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.

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.

```
(defrule system-banner ""
  (declare (salience 10))
  =>
  (printout t crlf crlf)
  (printout t "ЭКСПЕРТНАЯ СИСТЕМА AUTOEXPERT")
  (printout t crlf crlf)
)
```

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 "Рекомендации по ремонту:")
   (printout t crlf crlf)
   (format t " %s%n%n%n" ?item)
)
```

2.6. ЛИСТИНГ ПРОГРАММЫ

В данном разделе приведен полный листинг программы с подробными комментариями. Внимательно изучите приведенный ниже текст программы, чтобы понять механизм работы экспертной системы.

```
;;;*****************************************************************
;;;  Пример экспертной системы на языке CLIPS, позволяющей  диагностировать
;;;    некоторые неисправности автомобиля  и предоставлять пользователю
;;;    рекомендации по их устранению
;;;
;;;                          CLIPS Version 6.3 Example
;;;
;;;*****************************************************************
;;;     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 "Двигатель запускается (yes/no)? ")
     then
     (if (yes-or-no-p "Двигатель работает нормально (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 (rotation-state engine does-not-rotate) ( правило 4).
;;; Кроме того, правило делает предположение о наличии плохой искры
;;; (spark-state engine irregular-spark) или ее отсутствии в системе
;;; зажигания (spark-state engine does-not-spark)

(defrule determine-rotation-state ""
  (working-state engine does-not-start)
  (not (rotation-state engine ?))
  (not (repair ?))
  =>
  (if (yes-or-no-p "Двигатель вращается (yes/no)? ")
     then     (assert (rotation-state engine rotates))          ; двигатель вращается
              (assert (spark-state engine irregular-spark))   ; плохая искра

     Else (assert (rotation-state engine does-not-rotate))    ; двигатель не вращается
          (assert (spark-state engine does-not-spark)))      ; нет искры
)

;;; 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 (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 "В баке имеется топливо (yes/no)? "))
     then  (assert (repair "Добавить топливо."))))
```

;;; Правило determine-battery-state по ответу пользователя определяет,
;;; заряжен ли аккумулятор. В случае если это не так, пользователю выдается
;;; рекомендация по ремонту - Зарядите аккумулятор(правило 6).
;;; Правило также добавляет факт, описывающий состояние аккумулятора.
;;; Выполнение диагностических правил прекращается.

```
(defrule determine-battery-state ""
  (rotation-state engine does-not-rotate)
  (not (charge-state battery ?))    ; состояние аккумулятора еще не определено
  (not (repair ?))
  =>
  (if (yes-or-no-p "Аккумулятор заряжен (yes/no)? ")
     then
      (assert (charge-state battery charged)) ; аккумулятор заряжен
     else
          (assert (repair "Зарядите аккумулятор."))  ; рекомендация
           (assert (charge-state battery dead))))  ; аккумулятор разряжен
```

;;; 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 | not-low-output))
  (not (repair ?))
  =>
  (if (yes-or-no-p "Выходная мощность двигателя низкая(yes/no)? ")
     then
       (assert (symptom engine low-output))  ; низкая выходная мощность двигателя
     else
       (assert (symptom engine not-low-output))) ; нормальная мощность двигателя
```

;;; Правило determine-point-surface-state определяет по ответу
;;; пользователя состояние контактов (см. правила 1, 12). Контакты могут
;;; находиться в одном из трех состояний: чистые, опаленные и загрязненные.
;;; В двух последних случаях пользователю выдаются соответствующие реко-
мендации.

```
;;; 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 "Каково состояние контактов (norm/opal/zagr)? "
            norm opal zagr))
   (if (eq ?response opal)
       then
         (assert (repair "Замените контакты."))  ; рекомендация
       else (if (eq ?response zagr)
                then
                    (assert (repair "Почистите контакты."))))) ; рекомендация

;;; 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 "Катушка зажигания пропускает ток (yes/no)? ")
       then
         (assert (repair "Замените распределительные провода.")) ; рекомендация
       else
         (assert (repair "Замените катушку зажигания.")))) ; рекомендация

;;; Правило determine-sluggishness спрашивает пользователя, не ведет ли
;;; себя машина инертно (не сразу реагирует на подачу топлива).
;;; Если такой факт обнаружен, то, 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 "Машина ведет себя инертно (yes/no)? ")
     then
       (assert (repair "Прочистите систему подачи топлива")))) ; рекомендация

;;;  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 "Перебои с зажиганием есть(yes/no)? ")
     then
       (assert (repair "Отрегулируйте зазоры между контактами.")) ; рекомендация
       (assert (spark-state engine irregular-spark)))) ; Плохая искра

;;;  Правило determine-knocking узнает - не стучит ли двигатель.
;;;  Если это так, то необходимо отрегулировать зажигание (см. правило 1)
;;;  Выполнение диагностических правил прекращается.

(defrule determine-knocking ""
  (working-state engine unsatisfactory)
  (not (repair ?))
  =>
  (if (yes-or-no-p "Двигатель стучит (yes/no)? ")
     then
       (assert (repair "Отрегулируйте зажигание")))) ; рекомендация
;;;**************************************************************
;;; 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))
  (working-state engine normal)    ; Если двигатель работает нормально
  =>
  (assert (repair "Ремонт не нужен."))     ; 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 неудовлетворительно
   (working-state engine unsatisfactory)
   =>
   (assert (charge-state battery charged))       ; аккумулятор заряжен
   (assert (rotation-state engine rotates)))     ; двигатель вращается
;;;*****************************************************************
;;;   Запуск и завершение программы
;;;*****************************************************************
;;; 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 "Обратитесь в сервисную службу.")))

;;; 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 "Рекомендации по ремонту:")
  (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 ""
  (declare (salience 10))
  =>
  (printout t crlf crlf)
  (printout t "ЭКСПЕРТНАЯ СИСТЕМА AUTOEXPERT")
  (printout t crlf crlf)
)
```

2.7. ЗАПУСК И ТЕСТИРОВАНИЕ ПРОГРАММЫ

Для запуска программы наберите приведенный листинг в каком-нибудь текстовом редакторе (лучше использовать встроенный редактор CLIPS). Сохраните набранный файл, например, с именем auto_exp.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 (**File -> Load**). Если текст програм-

мы набран без ошибок, то после загрузки файла будет выведено сообщение **TRUE** (рис.1). Если выведено **FALSE**, в синтаксисе определений функций или правил была допущена ошибка.

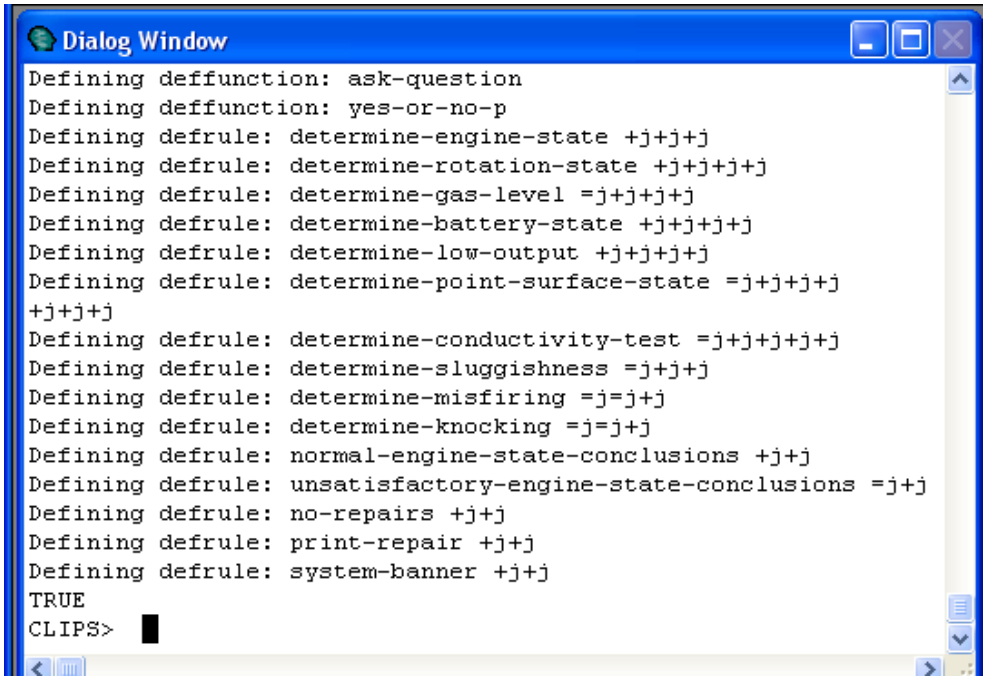

Рис. 1. Загрузка экспертной системы

После удачной загрузки файла убедитесь, что все правила присутствуют в списке правил CLIPS, а функции — в списке функций. Легче всего это выполнить с помощью менеджера правил(**Browse-> Defrule Manager**) и менеджера функций (**Browse-> Deffunction Manager**). Внешний вид этих менеджеров показан на рис. 2 и рис. 3.

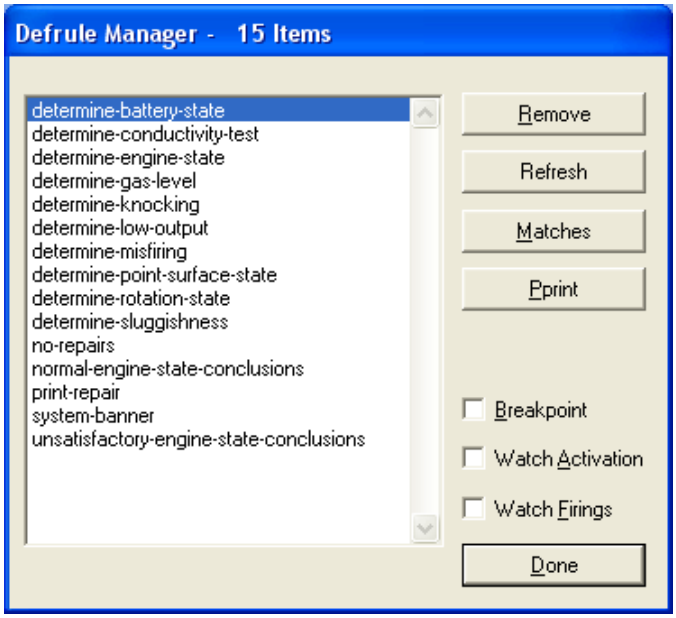

Рис2. Окно менеджера правил

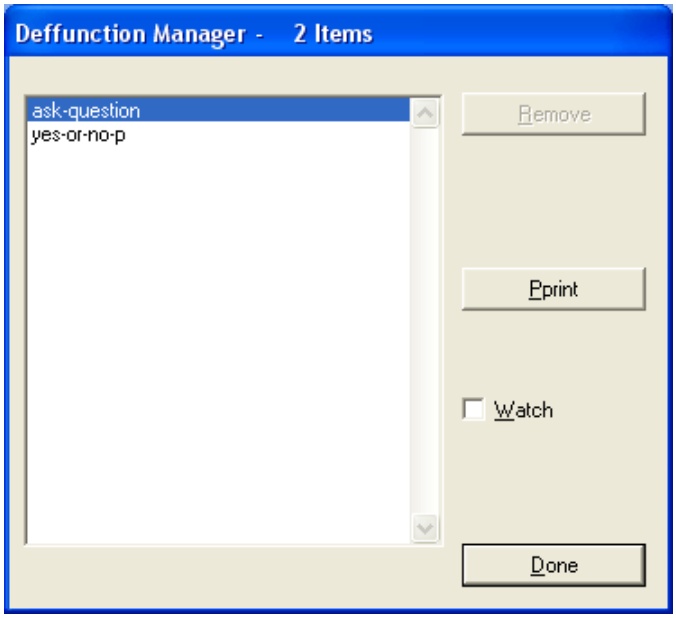

Рис3. Окно менеджера функций

Для повторного запуска экспертной системы необходимо еще раз выполнить команды **reset** и **run**.

Протестируйте экспертную систему, по-разному отвечая на ее вопросы. Один из сеансов работы экспертной системы в режиме диалога с пользователем показан на рис. 4.

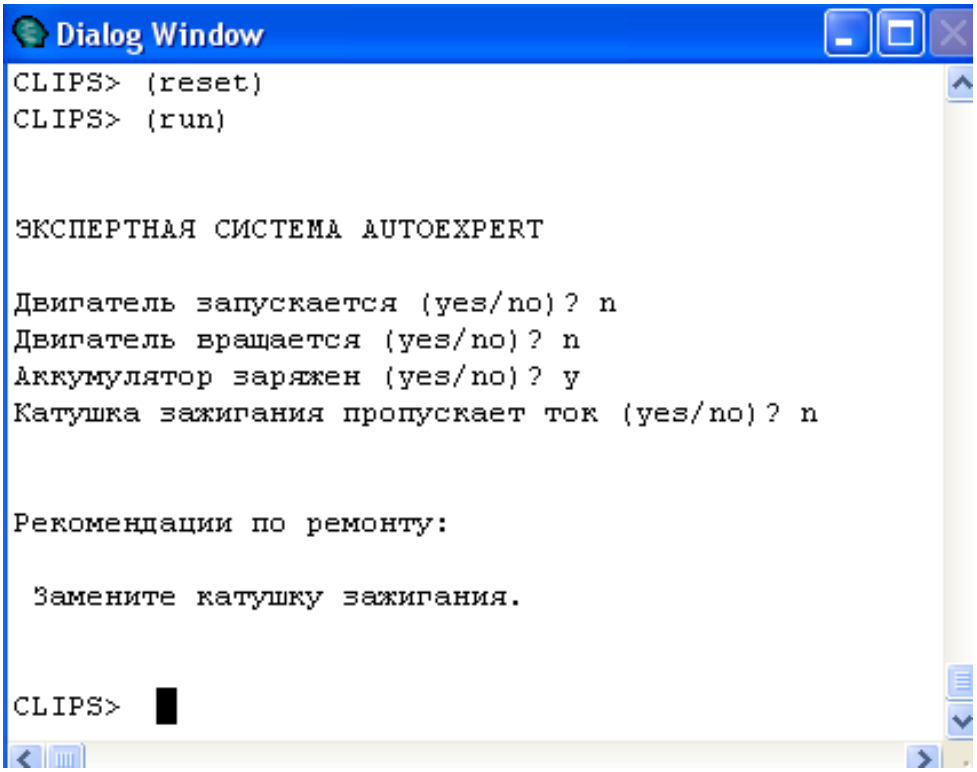

Рис. 4. Диалог с экспертной системой

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**).

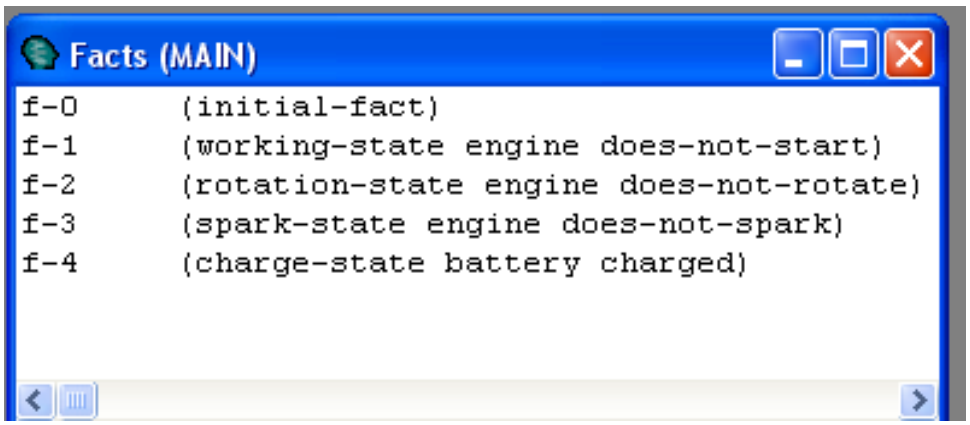

Рис. 5. Текущий список фактов в процессе работы экспертной системы

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

Задание 1. Экспертная система PrinterExpert

Постановка задачи

Необходимо разработать экспертную систему для определения неисправности принтера по внешним признакам. Экспертная система должна также предоставлять пользователю рекомендации по устранению выявленной неисправности.

Данная диагностическая ЭС должна решать задачу в режиме диалога с пользователем. При этом за один запуск система может предоставить пользователю только одну рекомендацию по исправлению неисправности. В случае если неисправностей несколько, то систему нужно будет последовательно вызывать несколько раз, удаляя обнаруженную на каждом новом шаге неисправность.

Идентификация проблемной области

Пусть в результате бесед со специалистом по ремонту принтеров были установлены следующие правила:

1. Если принтер работает нормально, то это говорит о том, что он печатает, драйвера установлены на компьютер к которому он подключен, принтер заправлен бумагой, картридж краской и на принтер подано питание.
2. Если принтер включается, но работает неправильно, то это говорит по крайней мере о том, что он включен в сеть.
3. Если принтер не включается, то нужно проверить, включен ли он в сеть.
4. Если принтер включается, но не реагирует на команду печати, то нужно проверить установлен ли на ЭВМ драйвер принтера.
5. Если принтер включается, но не реагирует на команду печати, а драйвер установлен, то это может говорить о том, что принтер не подключен к компьютеру.
6. Если принтер включается, но при печати из принтера выходят пустые листы, то нужно проверить заправлен ли картридж чернилами.
7. Если принтер включается, и реагирует на команды с ЭВМ, но не печатает, то это может говорить о том, что в нем нет бумаги.
8. Если принтер включается, и реагирует на команды с ЭВМ, но не печатает, то это может говорить также о том, что не закрыта крышка отсека с картриджем.
9. Если принтер включается, и печатает, но в процессе печать приостановилась, то возможно произошло замятие бумаги или она закончилась.
10. Если при печати на бумаге появляются черные размытые полосы, то необходимо почистить картридж.
11. Если с помощью вышеописанных правил оценить состояние принтера не представляется возможным, то его необходимо отправить в ремонт.

Из приведенных выше правил определены следующие сущности:

1. состояние принтера: нормально работает, неудовлетворительная работа или принтер не работает
2. состояние питания принтера: включается или не включается

3. состояние чернил: не заправлены или заправлены
4. состояние картриджа: в нормальном состоянии или загрязнен
5. наличие бумаги: бумага есть или бумаги нет
6. связь ЭВМ с принтером: связи нет или связь есть
7. состояние крышки отсека картриджа: отсек закрыт или отсек открыт

Для устранения неисправности выработаны рекомендации:
Из правила 1 – ремонт не требуется
Из правила 2– включить принтер в сеть.
Из правила 3 – установить драйвер принтера
Из правила 4 – подключить принтер к компьютеру
Из правила 5– заправить чернила
Из правила 6– заправить бумагу в принтер
Из правила 7 – закрыть крышку отсека с картриджем
Из правила 8 – заменить картридж
Из правила 9 – вынуть замятую бумагу из механизма
Из правила 10 – почистить картридж
Из правила 11 – неисправность не установлена, необходимо обратиться в сервисный центр

Рекомендации по разработке экспертной системы.

Для решения задачи необходимо разработать программу, которая будет включать:

А) факты для описания возможных состояний каждой из сущностей.

Например, состояние принтера можно описать с помощью фактов:

```
working-state printer normal          ; принтер нормально работает
working-state printer unsatisfactory  ; неудовлетворительная работа
working-state printer does-not-start  ; принтер не работает
```

Аналогично опишите остальные сущности.

Б) Факты, содержащие рекомендации по устранению неисправности

Например, рекомендацию «ремонт не требуется» можно представить в виде факта:

repair “ Ремонт не требуется ”.

Аналогично опишите остальные рекомендации.

В) диагностические правила для определения возможных неисправностей принтера.

Например, правило, определяющее состояние принтера можно составить следующим образом:

```
(defrule determine-working-state  ""
(not (working-state printer ?))
  (not (repair ?))
=>
  (if (yes-or-no-p "Принтер работает (yes/no)? ")
```

```
then
(if (yes-or-no-p " Принтер работает нормально (yes/no)? ")
   then (assert (repair "Ремонт не требуется"))
   else (assert (working-state printer unsatisfactory)))
else
(assert (working-state printer does-not-start))))
```

Разработайте следующие диагностические правила:
Правило, определяющее включается ли принтер
Правило, определяющее есть ли связь ПК с принтером
Правило, определяющее установлен ли драйвер принтера
Правило, определяющее заправлен ли картридж чернилами
Правило, определяющее есть ли в принтере бумага
Правило, определяющее состояние картриджа
Правило, определяющее закрыта ли крышка отсека с картриджем
Правило, действующее, если не удалось определить неисправность

Г) правила для выдачи рекомендации по устранению найденной неисправности и для вывода названия экспертной системы, а также функции для вывода вопроса пользователю и получения ответа (Подобные функции реализованы в ЭС AutoExpert).

Сохраните полученную программу, например, под именем Printer_Exp.clp. Загрузите созданный вами файл с помощью команды load (**File -> Load**). Если после загрузки файла будет выведено сообщение **TRUE,** запустите и протестируйте экспертную систему при различных ответах пользователя. Для повторного запуска экспертной системы необходимо еще раз выполнить команды **reset** и **run.**

Для просмотра текущего списка фактов сделайте видимым окно фактов (**Fact Window**). Составьте отчет о проделанной работе

Задание 2. Экспертная система PlayerExpert

Постановка задачи

Необходимо разработать экспертную систему для установления неисправностей flash-плееров по внешним признакам. Экспертная система должна также предоставлять пользователю рекомендации по устранению выявленной неисправности.

Данная диагностическая ЭС должна решать задачу в режиме диалога с пользователем. При этом за один запуск система может предоставить пользователю только одну рекомендацию по исправлению неисправности. В случае если неисправностей несколько, то систему нужно будет последовательно вызывать несколько раз, удаляя обнаруженную на каждом новом шаге неисправность.

Идентификация проблемной области

Пусть в результате бесед с экспертом были установлены следующие эмпирические правила:

1. Если плеер включается, и работает стабильно, значит микросхемы и флешдиск целы, экран плеера цел, в нем есть закаченная музыка, питание на плеер подается, аудиовыход не сломан, сбоя памяти и программного обеспечения нет, никакого ремонта не требуется.
2. Если плеер не включился, то нужно узнать, подается ли питание на плеер. Если подается, а плеер не включается, то значит, либо внутренний флешдиск, либо микросхемы сгорели или треснули, необходимо отнести его в сервис-центр. Если питание на плеер не подается, необходимо его подать, путем замены батареек.
3. Если плеер включился, но работает не стабильно, и музыка не проигрывается, то возможно в нём нет песен, необходимо убедится, что в нем есть закаченные треки.
4. Если плеер включился, но работает не стабильно, и музыка не проигрывается, а песни в нём есть, то возможно сломан аудиовыход, убедитесь что аудиовыход в порядке. Если аудиовыход сломан, расколот или видны другие механические неисправности, то необходимо отнести плеер в сервис центр.
5. Если плеер включился, но работает не стабильно, и музыка не проигрывается, а песни в нём есть, аудиовыход в порядке, то это сбой программного обеспечения плеера, необходимо подключить плеер к компьютеру, выполнить синхронизацию, и переустановить программное обеспечение плеера.
6. Если плеер включился, но работает не стабильно, есть звук, и на экране выводится не то что должно выводиться, то это сбой программного обеспечения плеера, необходимо подключить плеер к компьютеру, выполнить синхронизацию, и переустановить программное обеспечение плеера.
7. Если плеер включился, но работает не стабильно, и музыка играет некорректно, то возможно это сбой памяти, необходимо присоединить плеер к компьютеру и проверить память.
8. Если плеер включился, но работает не стабильно, и не работает экран, то это механическая ошибка, необходимо отнести плеер в сервис центр.
9. Возможны ситуации, когда состояние плеера нельзя описать приведенными выше факторами и устройству может потребоваться более детальный анализ состояния.

Из приведенных выше правил были определены следующие сущности:

1. состояние плеера: не включается, работает стабильно или работает не стабильно
2. состояние питания плеера: питание есть или питания нет
3. состояние звука: звук есть или звука нет
4. состояние экрана: экран исправен или экран неисправен
5. состояние проигрывания музыки: воспроизведение есть или воспроизведения нет

Для устранения неисправности выработаны рекомендации:
Из правила 1 – Ремонт не требуется;
Из правила 2– Необходимо подать питание на плеер, путем замены батареек;
Из правила 3 – Необходимо убедиться, что в плеере есть закаченные треки;

Из правила 3 – Необходимо убедиться, что аудиовыход не сломан;
Из правила 5 – Необходимо подключить плеер к компьютеру, выполнить синхронизацию, и переустановить программное обеспечение плеера;
Из правила 6 – Необходимо подключить плеер к компьютеру, выполнить синхронизацию, и переустановить программное обеспечение плеера;
Из правила 7 – Необходимо присоединить плеер к компьютеру и проверить память;
Из правила 8 – Необходимо отнести плеер в сервис центр;
Из правила 9 – Обращение в службу поддержки.

Рекомендации по разработке экспертной системы.

Для решения задачи необходимо разработать программу, которая будет включать:

А) факты для описания возможных состояний каждой из сущностей.

Например, состояние плеера можно описать с помощью фактов:

```
working-state player disenabled      ; не работает
working-state player stable          ; работает стабильно
working-state player trable          ; работает не стабильно
```

Аналогично опишите остальные сущности.

Б) Факты, содержащие рекомендации по устранению неисправности

Например, рекомендацию «ремонт не требуется» можно представить в виде факта:

repair “ Ремонт не требуется ”.

Аналогично опишите остальные рекомендации.

В) диагностические правила для определения возможных неисправностей плеера.

Например, правило, определяющее состояние плеера можно составить следующим образом:

```
(defrule determine-working-state  ""
(not (working-state player ?))
  (not (repair ?))
=>
  (if (yes-or-no-p "Плеер работает (yes/no)? ")
     then
     (if (yes-or-no-p " Плеер работает стабильно (yes/no)? ")
        then (assert (repair "Ремонт не требуется"))
        else (assert (working-state player trable)))
     else
     (assert (working-state player disenabled))))
```

Разработайте следующие диагностические правила:

Правило, определяющее состояние плеера.

Правило, определяющее есть ли звук в плеере
Правило, определяющее состояние экрана
Правило, определяющее подается ли питание на плеер
Правило, действующее, если не удалось определить неисправность.

Г) правила для выдачи рекомендации по устранению найденной неисправности и для вывода названия экспертной системы, а также функции для вывода вопроса пользователю и получения ответа (Подобные функции реализованы в ЭС AutoExpert).

Сохраните полученную программу, например, под именем Player_Exp.clp. Загрузите созданный вами файл с помощью команды load (**File -> Load**). Если после загрузки файла будет выведено сообщение **TRUE,** запустите и протестируйте экспертную систему при различных ответах пользователя. Для повторного запуска экспертной системы необходимо еще раз выполнить команды **reset.**

Для просмотра текущего списка фактов сделайте видимым окно фактов (**Fact Window**). Составьте отчет о проделанной работе

ЛИТЕРАТУРА

1. Джозеф Джарратано, Гари Райли Глава 7. Введение в CLIPS // «Экспертные системы: принципы разработки и программирование» : Пер. с англ. — М. : 2006. — 1152 стр. с ил., «Вильямс»

2. Трофимов В. База данных+CLIPS=База знаний // Компьютеры+программы. N 10.-2003- С. 56–61

3. Частиков, А.П. Разработка экспертных систем. Среда CLIPS. / А.П Частиков, Д.Л.Белов, Т.А.Гаврилова – СПб: БХВ-Петербург, 2003. – 393с.

4. Базы знаний интеллектуальных систем / Т.А.Гаврилова, В.Ф.Хорошевский – СПб: Питер, 2000 – 384 с.

5. Тельнов Ю.Ф. Интеллектуальные информационные системы в экономике. – Уч. пособие. – М.: Синтег, 1998. – 216 с.

6. http://ru.wikipedia.org/wiki/CLIPS

7. http://www.intuit.ru/department/human/isrob/7/

Рекомендуемая литература

8. Джозеф Джарратано, Гари Райли Глава 7. Введение в CLIPS // «Экспертные системы: принципы разработки и программирование» : Пер. с англ. — М. : 2006. — 1152 стр. с ил., «Вильямс»

9. Трофимов В. База данных+CLIPS=База знаний // Компьютеры+программы. N 10.-2003- С. 56–61

10. Частиков, А.П. Разработка экспертных систем. Среда CLIPS. / А.П Частиков, Д.Л.Белов, Т.А.Гаврилова – СПб: БХВ-Петербург, 2003. – 393с.

11. http://ru.wikipedia.org/wiki/CLIPS

12. http://www.intuit.ru/department/human/isrob/7/