

ЛАБОРАТОРНАЯ РАБОТА № 1

- ТЕМА:** Знакомство со средой CLIPS 6.2. Работа с фактами в среде CLIPS.
- ЦЕЛЬ:** Научиться общим приемам работы в среде CLIPS. Научиться использовать имеющиеся возможности CLIPS для работы с фактами с помощью соответствующих конструкторов, операций и функций: создавать шаблоны с помощью конструктора *deftemplate*; осуществлять создание, изменение, удаление, поиск фактов, просмотр, сохранение и загрузку списка фактов, определение списка предопределенных фактов с помощью конструктора *deffacts*.

ПЛАН ЗАНЯТИЯ:

1. Краткие теоретические сведения.
2. Знакомство со средой CLIPS 6.2.
3. Синтаксис определений.
4. Использование конструкторов *deftemplate* и *deffacts* для работы с фактами в CLIPS.
5. Использование функций *assert*, *retract* и *modify*.
6. Использование функций *duplicate*, *assert-string*, *fact-existp*.
7. Функции для работы с неупорядоченными фактами: *fact-relation*, *fact-slot-names* и *fact-slot-value*.
8. Функции сохранения и загрузки списка фактов: *save-facts* и *load-facts*.
9. Контрольные вопросы.

1. КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Первоначально аббревиатура CLIPS была названием языка — C Language Integrated Production System (язык C, интегрированный с продукционными системами), удобного для разработки баз знаний и макетов экспертных систем. Теперь CLIPS представляет собой современный инструмент, предназначенный для создания экспертных систем (expert system tool). CLIPS состоит из интерактивной среды — экспертной оболочки со своим способом представления знаний, гибкого и мощного языка и нескольких вспомогательных инструментов.

Появление языка CLIPS можно датировать 1984 г., место рождения CLIPS — космический центр Джонсона NASA. Именно в это время отдел искусственного интеллекта (теперь Software Technology Branch) разработал множество прототипов экспертных систем, использующих современное программное и техническое обеспечение.

Последняя, доступная сейчас версия CLIPS 6.2, вышла в свет 31 марта 2002 г. Основными отличиями новой версии CLIPS являются поддержка разработки встроенных приложений, использующих CLIPS, и улучшенный интерфейс для Windows-версии, оптимизированный для использования на платформах Windows 95/98/NT.

2. ЗНАКОМСТВО СО СРЕДОЙ CLIPS 6.2

Для обучения будем использовать Windows-версия CLIPS 6.2. Внешний вид главного окна CLIPS показан см. рис. 1.

Windows-версия среды CLIPS полностью совместима с базовой спецификацией языка. Ввод команд осуществляется непосредственно в главное окно CLIPS. Однако по сравнению с базовой Windows-версия предоставляет множество дополнительных визуальных инструментов (например, менеджеры фактов или правил) значительно облегчающих жизнь разработчика экспертных систем.

Экспертные системы созданные с помощью CLIPS, могут быть запущены тремя способами:

- ☐ вводом соответствующих команд и конструкторов языка непосредственно в среду CLIPS;
- ☐ использованием интерактивного оконного интерфейса CLIPS (например, для версий Windows или Macintosh);
- ☐ с помощью программ-оболочек, реализующих свой интерфейс общения с пользователем и использующих механизмы представления знаний и логического вывода CLIPS.

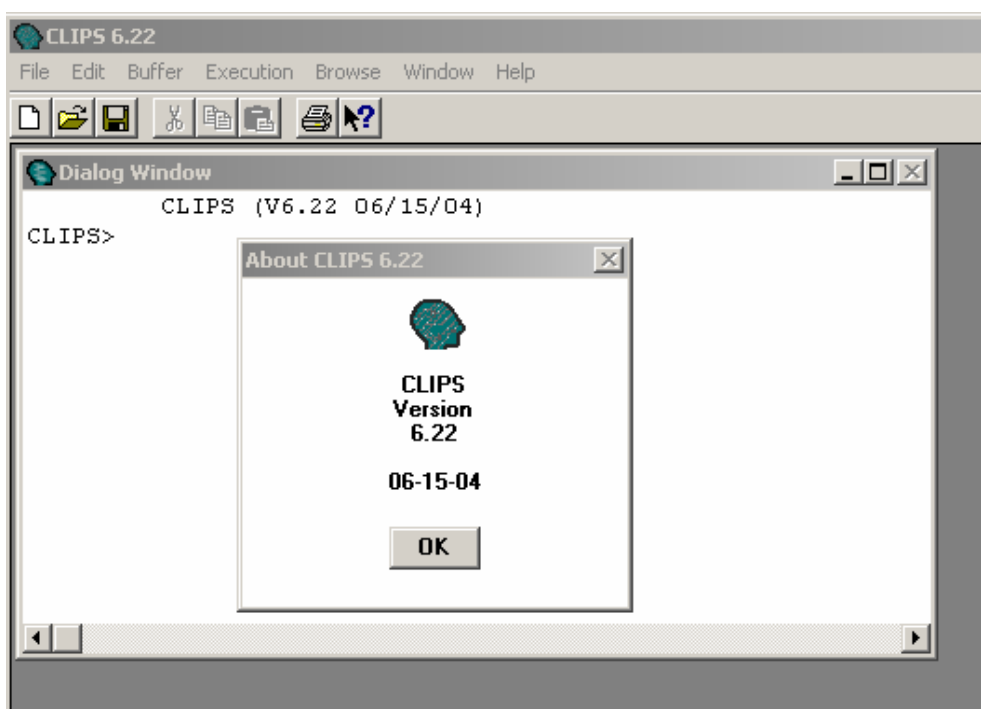


Рис 1. Главное окно CLIPS

Кроме того, CLIPS при запуске позволяет выполнять командные файлы собственного формата (эта возможность также доступна при помощи команды **batch**). Для реализации этой возможности необходимо запустить CLIPS (в нашем случае это файл CLIPSWin.exe) с одним из трех следующих аргументов:

- ☐ – *f* <имя-файла>
- ☐ – *f2* <имя-файла>
- ☐ – *l* <имя-файла>

Текстовый файл, заданный с помощью опции *-f*, должен содержать команды CLIPS. Если заданный файл содержит команду **exit**, то CLIPS завершит свою работу и пользователь вернется в операционную систему. В случае если команда **exit** отсутствует, то после выполнения всех команд из заданного файла пользователь попадет в главное окно CLIPS. Команды в текстовом файле должны быть набраны так же, как если бы они вводились непосредственно в командную строку CLIPS (т. е. все команды должны быть заключены в круглые скобки и разделены символом перехода на новую строку). Опция *-f* фактически эквивалентна запуску команды **batch** сразу после запуска CLIPS.

Опция *-f2* идентична *-f*, но, в отличие от опции *-f*, она использует команду **batch***. Файл, заданный этой опцией, также выполняется после

запуска CLIPS, но результаты выполнения команд не отображаются на экране.

Опция *-l* задает текстовый файл, содержащий конструкторы CLIPS, которые тут же запускаются на выполнение. Использование этой опции эквивалентно использованию команды ***load*** сразу после запуска CLIPS.

Основным методом общения с CLIPS, является применение командной строки. После появления в главном окне CLIPS приглашения — *clips>* — команды пользователя могут вводиться в среду непосредственно с клавиатуры. Команды могут быть вызовами системных или пользовательских функций, конструкторами различных данных CLIPS и т. д. В случае вызова пользователем некоторой функции, она немедленно выполняется, и результат ее работы отображается пользователю. Для вызова функций или операций CLIPS использует префиксную нотацию — аргументы всегда следуют после имени функции или операции. При вызове конструкторов CLIPS создает новый объект соответствующего типа, так или иначе представляющий некоторые знания в системе. При вводе в среду имени созданной ранее глобальной переменной CLIPS отобразит ее текущее значение. Ввод в среду некоторой константы просто приведет к ее немедленному отображению в главном окне CLIPS. В качестве примера введите в среду следующие команды (листинг 1).

Листинг 1. Пример команд CLIPS

```
(+ 3 4)
(defglobal ?*x*=3)
?*x*
red
```

Результат выполнения этих команд приведен на рис. 2.

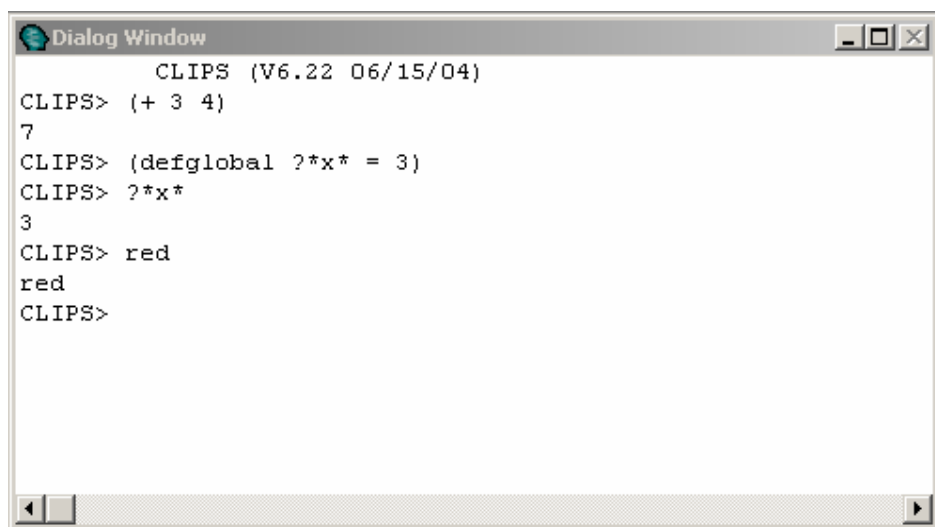


Рис 2. Результат выполнения команд листинга 1

Первой командой данного примера вызывается функция арифметического сложения двух чисел: 3 и 4. Результат работы этой функции — 7 сразу же отображается на экран. После этого, с помощью конструктора *defglobal*, создается глобальная переменная *?*x**, которая инициализируется значением 3. При вводе в командную строку имени глобальной переменной *?*x** CLIPS отображает ее текущее значение, равное 3. Последней командой была введена некая константа *red*, значение которой см. рис. 2.

3. СИНТАКСИС ОПРЕДЕЛЕНИЙ

Рассмотрим синтаксис, используемый для определения тех или иных конструкций языка CLIPS. В качестве базового синтаксиса для определения конструкций языка используется стандартная БНФ-нотация. Приведем правила, используемые для построения определений.

Слово или выражение, заключенное в угловые скобки, называется **нетерминальным символом** (например, *<string>*). Нетерминальный символ требует дальнейшего определения.

Слова или выражения, не заключенные в угловые скобки, называются **терминальными символами**, и представляют синтаксис описываемой конструкции языка CLIPS. Терминальные символы (особенно круглые скобки) должны вводиться в командную строку именно так, как показано в определении.

- ❑ Если за нетерминальным символом следует символ ***, то это означает, что в данном месте может находиться список из нуля или более элементов этого типа.
- ❑ Если же за нетерминальным символом следует *+*, то в данном месте может находиться список из одного или более элементов этого типа.
- ❑ Символы *** и *+*, встречающиеся сами по себе (не следующие после нетерминальных символов), являются терминальными.
- ❑ Многоточие, как горизонтальное, так и вертикальное, также используется для отображения списка из одного или более элементов.
- ❑ Элементы, заключенные в квадратные скобки (например, [*<комментарии>*]), являются необязательными элементами, которые могут входить в определение.

- Вертикальная черта, разделяющая два или более элемента определения, указывает на то, что в конструкции необходимо использовать один из перечисленных элементов.
- Символ ::= используется для обозначения необходимости замены некоторого нетерминального символа.

Например, определение:

`<lexeme> ::= <symbol> | <string>`

обозначает, что нетерминальный символ `<lexeme>`, встречающийся в некотором определении, должен быть заменен либо на символ `<symbol>`, либо на символ `<string>`.

- Пробелы, символы табуляции, переходы на другую строку используются только для логического разделения элементов определения и игнорируются CLIPS (кроме строк, заключенных в двойные кавычки).
- Символ ; применяется при необходимости использования комментария, который полностью игнорируется системой CLIPS. (Если же Вам необходимо сохранить комментарий в базе знаний системы, его необходимо заключить в кавычки).

Для функционирования любой экспертной системы критически важным является наличие базы знаний. Об этом говорит даже тот факт, что в последнее время все чаще термин "система, основанная на знаниях" (knowledge-base system) употребляется в качестве синонима термина "экспертная система". Как правило, в любой экспертной системе знания представляются фактами и правилами, заданными на некотором языке описания знаний. CLIPS не является исключением и предоставляет возможности для приобретения, хранения и обработки фактов и правил.

4. ИСПОЛЬЗОВАНИЕ КОНСТРУКТОРОВ *DEFTEMPLATE* И *DEFFACTS* ДЛЯ РАБОТЫ С ФАКТАМИ В CLIPS

Одной из основных форм представления данных в CLIPS являются **факты** — список неделимых (или атомарных) значений примитивных типов данных. CLIPS поддерживает два типа фактов — **упорядоченные факты** (*ordered facts*) и **неупорядоченные факты** или шаблоны (*non-ordered facts* или *template facts*).

Ссылаться на данные, содержащиеся в факте, можно либо используя строго заданную позицию значения в списке данных для упорядоченных фактов, либо указывая имя значения для шаблонов.

Факты можно добавлять, удалять, изменять и дублировать, вводя соответствующие команды с клавиатуры, либо из программы. После добавления факта в список фактов ему присваивается целый уникальный идентификатор, называемый **индексом факта** (*fact-index*). Индекс первого факта равен нулю, в дальнейшем индекс увеличивается на единицу при добавлении каждого нового факта. CLIPS предоставляет команды, очищающие текущий список фактов или всю базу знаний, эти команды присваивают текущему значению индекса 0.

Некоторые команды, например изменения, удаления или дублирования фактов, требуют указания определенного факта. Факт можно задать либо индексом факта, либо его адресом. Адрес факта представляет собой переменную-указатель, хранящую индекс факта. Процесс создания адресов фактов будет описан ниже.

Упорядоченные факты состоят из поля, обязательно являющимся данным типа *symbol* и следующей за ним, возможно пустой, последовательности полей, разделенных пробелами. Ограничением факта служат круглые скобки.

Определение 1. Упорядоченный факт

(данное_типа_symbol [поле]*)

Первое поле факта определяет так называемое отношение, или связь факта (*relation*). Термин "связь" означает, что данный факт принадлежит некоторому определенному конструктору или неявно объявленному шаблону.

Пример 1. Упорядоченные факты

(duck is bird)	;утка это птица
(Bob Mike is schoolboy)	;Боб Майкл – ученик
(Nuke did report)	;Нюк сделал доклад
(altitude is 1000 feet)	;высота 1000 футов

Количество полей в факте не ограничено. Поля в факте могут хранить данные любого примитивного типа CLIPS, за исключением первого поля, которое обязательно должно быть типа *symbol*. Следующие слова зарезервированы и не могут быть использованы в качестве первого поля: *test*, *and*, *or*, *not*, *declare*, *logical*, *object*, *exist* и *forall*. Эти слова могут использоваться в качестве имен слотов шаблонов, хотя это не рекомендуется.

CLIPS различает неупорядоченные факты от упорядоченных по первому полю факта. Первое поле фактов любого типа является значением типа *symbol*. Если это значение соответствует имени некоторого шаблона,

то факт — упорядоченный. Определение неупорядоченного факта, как и упорядоченного, ограничивается круглыми скобками.

Пример 2. Неупорядоченные факты

```
(client (name "Joe Brown") (id X9345A))  
(point-mass (x-velocity 100) (y-velocity -200))  
(class (teacher "Martha Jones") (#-students 30) (Room  
"37A"))',  
(grocery-list (#-of-items 3) (items bread milk eggs))
```

Порядок слотов в неупорядоченном факте не важен. Например, все приведенные ниже факты считаются идентичными:

```
(class (teacher "Martha Jones") (#-students 30) (Room "37A"))  
(class (#-students 30) (teacher "Martha Jones") (Room "37A"))  
(class (Room "37A") (#-students 30) (teacher "Martha Jones"))
```

В отличие от фактов, приведенных выше, упорядоченные факты из следующего примера не являются идентичными:

```
(class "Martha Jones" 30 "37A")  
(class 30 "Martha Jones" "37A")  
(class "37A" 30 "Martha Jones")
```

4.1. Конструктор *deftemplate*

Для создания неупорядоченных фактов в CLIPS предусмотрен специальный конструктор *deftemplate*. Его использование приводит к появлению в текущей базе знаний системы информации о шаблоне факта, с помощью которого в систему в дальнейшем можно будет добавлять факты, соответствующие данному шаблону. Таким образом, конструктор *deftemplate* аналогичен операторам *record* и *struct* таких процедурных языков программирования как Pascal или C.

Пример 3. Применение конструктора *deftemplate*

```
(deftemplate MyObject  
  (slot name)  
  (slot location)  
  (slot weight)  
  (multislot contents))
```

Как и все конструкторы CLIPS, конструктор *deftemplate* не возвращает никакого значения. При вводе данной команды в CLIPS вы должны увидеть результат, приведенный на рис. 3.

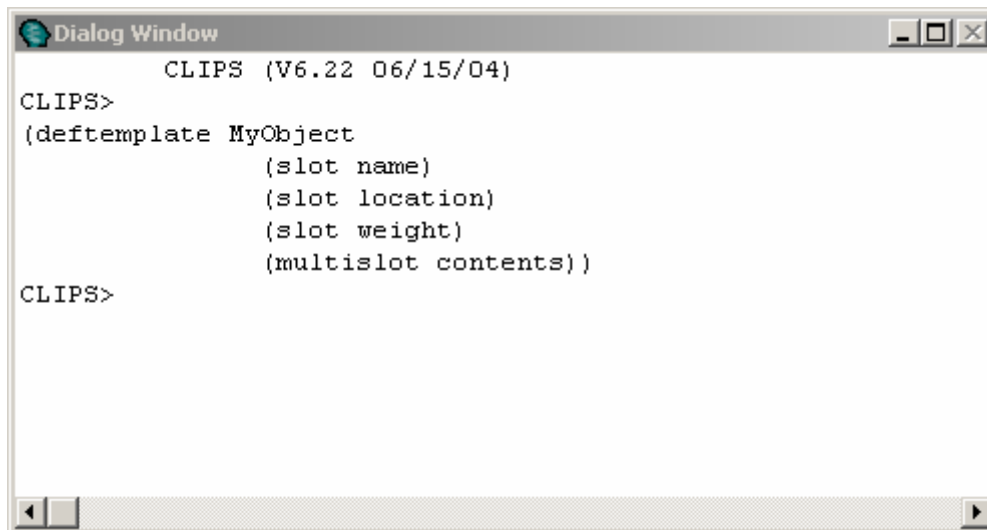


Рис 3. Использование конструктора **deftemplate**

Для просмотра всех определенных в текущей базе знаний шаблонов можно воспользоваться специальным инструментом **Deftemplate Manager** (Менеджер шаблонов), доступным в Windows-версии среды CLIPS. Для запуска менеджера шаблонов воспользуйтесь меню **Browse** и выберите пункт **Deftemplate Manager**. Менеджер шаблонов позволяет в отдельном окне просматривать список всех шаблонов, доступных в текущей базе знаний, удалять выбранный шаблон и отображать все его свойства (например, такие как имена и типы слотов). Внешний вид менеджера шаблонов представлен на рис. 4.

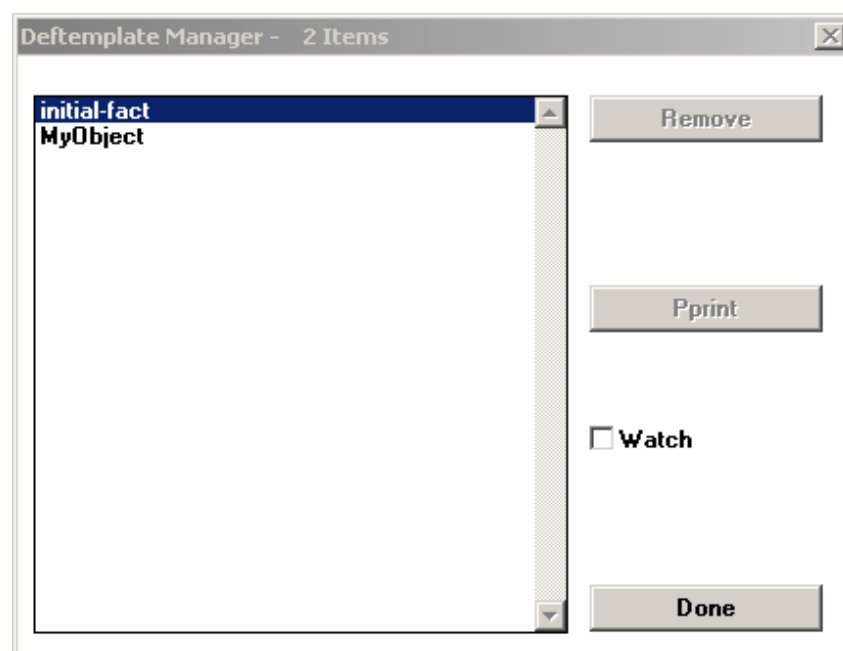


Рис 4. Окно менеджера шаблонов

После выполненной нами операции в текущей базе знаний находится два шаблона, о чем сообщается в заголовке окна менеджера (**Deftemplate Manager — 2 Items**). Первый шаблон является предопределенным шаблоном *initial-fact*. Он не имеет слотов и всегда добавляется при запуске среды. Его нельзя удалить с помощью менеджера, или просмотреть его определение. Вторым шаблоном является только что добавленный шаблон *MyObject*. Менеджер шаблонов позволяет вывести в главное окно среды его определение с помощью кнопки **Pprint** или удалить его из среды посредством кнопки **Remove**. На рис. 5 приведен результат последовательных операций вывода информации об определении шаблона и удалении его из текущей базы знаний.

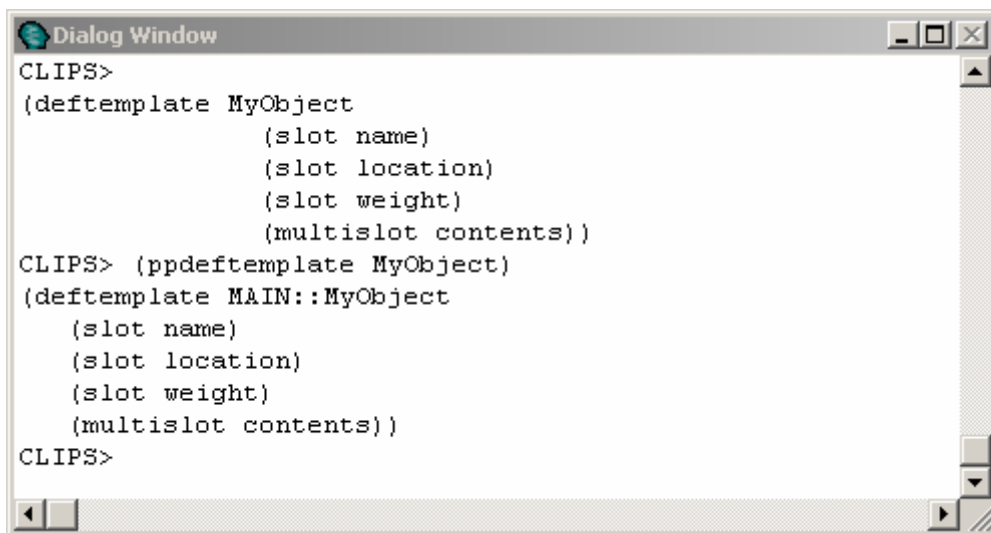


Рис 5. Получение информации и удаление шаблона

Флажок **Watch** позволяет включать/выключать режим отображения сообщений об использовании шаблонов для каждого присутствующего в системе шаблона в главном окне среды CLIPS. Если этот режим включен, пользователь будет получать сообщения при добавлении и удалении неупорядоченных фактов, использующих данный шаблон.

Рассмотрим полный синтаксис конструктора *deftemplate*:

Определение 2. Синтаксис конструктора *deftemplate*

(deftemplate <имя-шаблона>	[<необязательные-комментарии>] [<определение-слота>*])
<определение-слота>	::= <определение-простого-слота> <определение-составного-слота>

<определение-простого-слота>	::= (slot<имя-поля> <атрибуты-шаблона>)
<определение-составного-слота>	::= (multislot <имя-поля> <атрибуты-шаблона>)
<атрибуты-шаблона>	::= <атрибут-значение-по-умолчанию> <атрибут-ограничения>
<атрибут-значение-по-умолчанию>	::= (default ?DERIVE ?NONE <Выражение>) (default-dynamic <Выражение>)

Помимо ключевого слова *slot*, определяющего простой слот, допустимо также применение ключевого слова *multislot*, для определения составного слота. Простой слот, или слот, предназначен для хранения единицы информации одного из примитивных типов данных CLIPS. Составной слот способен хранить список подобных единиц информации неограниченного объема. Для доступа к конкретным данным, хранящимся в составном слоте, используются специальные групповые символы и функции.

При создании шаблона с помощью конструктора *deftemplate* каждому полю можно назначать определенные атрибуты, задающие значения по умолчанию или ограничения на значение слота. Рассмотрим эти атрибуты подробнее.

<Атрибут-значение-по-умолчанию> определяет значение, которое будет использовано в случае, если при создании факта не задано конкретное значение слота. В CLIPS существует два способа определения значения по умолчанию, поэтому в конструкторе *deftemplate* предусмотрено два различных атрибута, задающих значения по умолчанию: *default* и *default-dynamic*.

Атрибут *default* определяет статическое значение по умолчанию. С его помощью задается выражение, которое вычисляется один раз при конструировании шаблона. Результат вычислений сохраняется вместе с шаблоном. Этот результат присваивается соответствующему слоту в момент объявления нового факта. В случае если в качестве значения по умолчанию используется ключевое слово ?DERIVE, то это значение будет извлекаться из ограничений, заданных для данного слота. По умолчанию для всех слотов установлен атрибут *default* ?DERIVE.

В случае если в место выражения для значения по умолчанию используется ключевое слово ?NONE, то значение поля обязательно должно быть явно задано в момент выполнения операции добавления

факта. Добавление факта без определения значений полей с атрибутом *default ?NONE* вызовет ошибку.

Атрибут *default-dynamic* предназначен для установки динамического значения по умолчанию. Этот атрибут определяет выражение, которое вычисляется всякий раз при добавлении факта по данному шаблону. Результат вычислений присваивается соответствующему слоту.

Простой слот может иметь только одно значение по умолчанию. У составного слота может быть определено любое количество значений по умолчанию (количество значений по умолчанию должно соответствовать количеству данных, сохраняемых в составном слоте).

Пример 5. Использование атрибутов значения по умолчанию

(deftemplate foo

```
(slot w (default ?NONE))  
(slot x (default ?DERIVE))  
(slot y (default (gensym*)))  
(slot z (default-dynamic (gensym*))))
```

Ссылка на индекс факта в командах на изменение значения факта или его дублирование не связывает факт с соответствующим шаблоном явно. Это делает статическую проверку неоднозначной. Поэтому в командах, использующих индекс факта, статическая проверка не выполняется. Статическая проверка ограничений включена по умолчанию. Эту установку среды CLIPS можно изменить с помощью функции *set-static-constraint-checking*.

Помимо статической, CLIPS также поддерживает динамическую проверку ограничений. Если режим динамической проверки ограничений включен, то все новые факты, созданные с использованием некоторого шаблона и имеющие определенные значения, проверяются в момент их добавления в список фактов.

В случае если нарушение заданных ограничений произойдет в момент выполнения динамической проверки в процессе выполнения программы, то выполнение программы прекращается и пользователю будет выдано соответствующее сообщение.

По умолчанию в CLIPS отключен режим динамической проверки ограничений. Эту среду установки можно изменить с помощью функции *set-dynamic-constraint-checking*.

Помимо описанных выше функций для изменения состояний режимов статической и динамической проверки ограничений, пользователям Windows-версии среды CLIPS доступен визуальный способ

настройки этих установок. Для этого необходимо открыть диалоговое окно **Execution Options**, выбрав пункт **Options** из меню **Execution**. Внешний вид этого диалогового окна приведен на рис. 6.

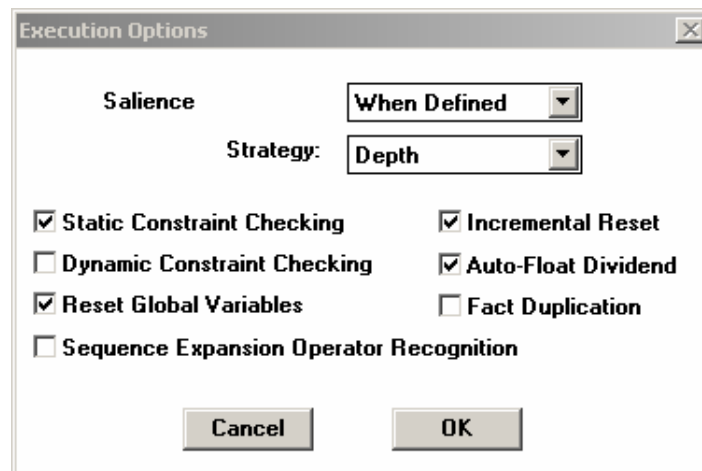


Рис 6. Диалоговое окно **Execution Options**

Для включения или выключения необходимых режимов проверки ограничений атрибутов выставите в соответствующее положение флажки **Static Constraint Checking** и **Dynamic Constraint Checking** и нажмите кнопку **OK**

Пример 6. Использование атрибутов ограничения

```
(deftemplate object
  (slot name
    (type SYMBOL)
    (default ?DERIVE))
  (slot location
    (type SYMBOL)
    (default ?DERIVE)))
```

4.2. Конструктор *deffacts*

Помимо конструктора *deftemplates*, CLIPS предоставляет конструктор *deffacts*, также предназначенный для работы с фактами. Данный конструктор позволяет определять список фактов, которые будут автоматически добавляться всякий раз после выполнения команды *reset*, очищающей текущий список фактов. Факты, добавленные с помощью конструктора *deffacts*, могут использоваться и удаляться так же, как и

любые другие факты, добавленные в базу знаний пользователем или программой, с помощью команды *assert*.

Определение 3. Синтаксис конструктора *deffacts*

```
(deffacts <имя-списка-фактов> [<необязательные-комментарии>]  
    [<факт>*])
```

Добавление конструктора *deffacts* с именем уже существующего конструктора приведет к удалению предыдущего конструктора, даже если новый конструктор содержит ошибки. В среде CLIPS возможно наличие нескольких конструкций *deffacts* одновременно и любое число фактов в них (как упорядоченных, так и неупорядоченных). Факты всех созданных пользователем конструкторов *deffacts* будут добавлены при инициализации системы.

В поля факта могут быть включены динамические выражения, значения которых будут вычисляться при добавлении этих фактов в текущую базу знаний CLIPS.

Пример 7. Использование конструктора *deffacts*

```
(deffacts startup "Refrigerator Status"  
    (refrigerator light on)  
    (refrigerator door open)  
    (refrigerator temp (+ 5 10 15)))
```

Обратите внимание, что третий факт содержит выражение, в данном примере сумму трех констант, но в качестве выражения, инициализирующего значение факта, могут использоваться и более сложные выражения, например, вызовы функций CLIPS или функций, определенных пользователем.

Проверить работу конструктора *deffacts* можно воспользовавшись диалогом **Watch Options**. Для этого выберите пункт **Watch** меню **Execution** или используйте комбинацию клавиш <Ctrl>+<W>. В диалоговом окне **Watch Options** включите режим просмотра изменения списка фактов, поставив галочку в поле **Facts**, как показано на рис. 7.



Рис 7. Диалоговое окно Watch Options

После этого нажмите кнопку **OK** и введите в CLIPS приведенный выше конструктор *deffacts*. Затем в меню **Execution** выберите пункт **Reset** (комбинация клавиш <Ctrl>+<E>). Если пример был набран правильно, то на экране должны появиться сообщения, аналогичные приведенным на рис. 8.

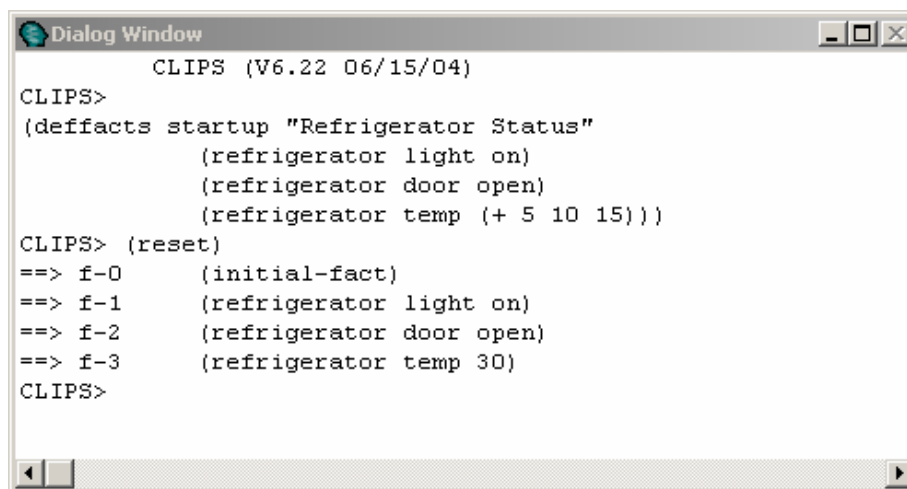


Рис 8. Просмотр процесса добавления файлов

Так же, как и для конструкторов *deftemplate*, CLIPS предоставляет визуальный инструмент для манипуляции с определенными в данный момент в системе конструкторами *deffacts* — **Deffacts Manager** (Менеджер предопределенных фактов). Для запуска **Deffacts Manager** в меню **Browse** выберите пункт **Deffacts Manager**. Внешний вид менеджера приведен на рис. 9.

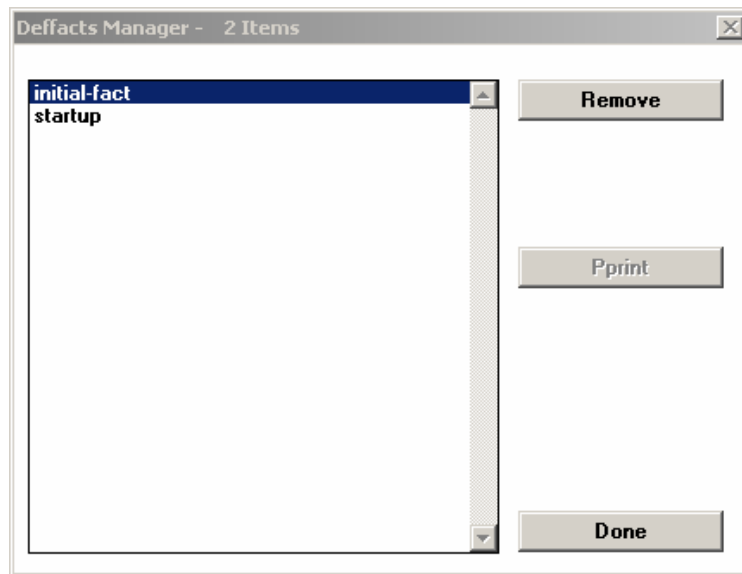


Рис 9. Окно менеджера предопределенных фактов

Менеджер отображает все введенные на текущий момент в систему конструкторы *deffacts*. В нашем случае это *initial-fact*, речь о котором пойдет ниже, и только что добавленный нами *startup*. Менеджер позволяет выводить в основное окно CLIPS информацию об определениях существующих в данный момент в системе конструкторов *deffacts* с помощью кнопки **Pprint** (кроме *deffacts initial-fact*) и удалять любой существующий конструктор. Пример вывода информации об определении конструктора *deffacts startup* приведен на рис.10. Обратите внимание, что комментарии, введенные после имени конструктора, сохраняются и выводятся на экран так же, как и конструкторе *deftemplate*.

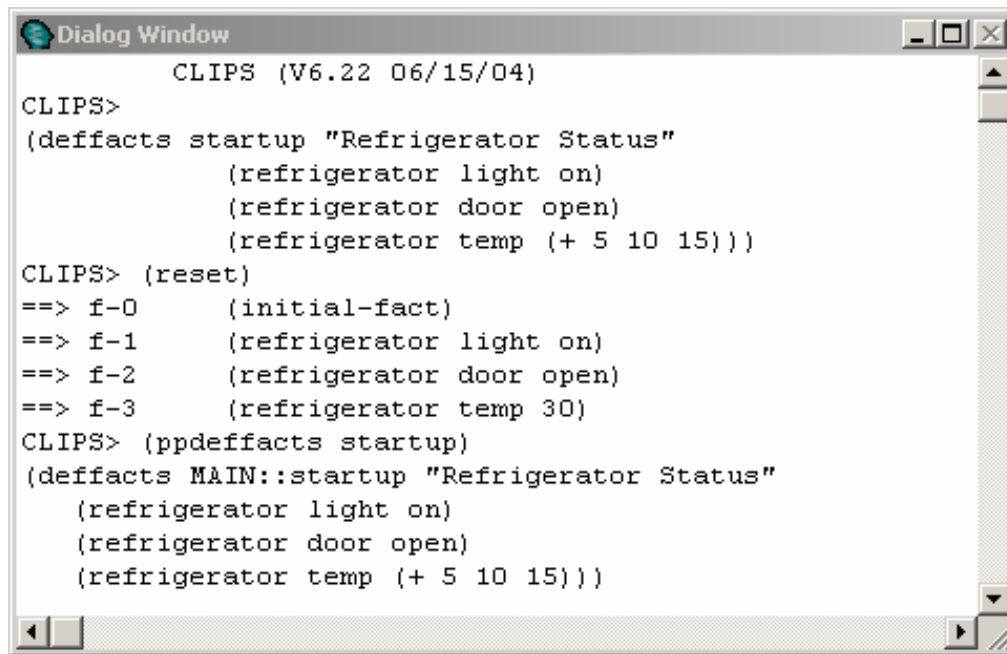


Рис 10. Получение информации об определенном конструкторе

Во время запуска и после выполнения команды *clear* CLIPS автоматически конструирует следующие предопределенные шаблоны и факты:

Определение 4. Предопределенные шаблоны и факты

```
(deftemplate initial-fact)
(defeffacts initial-fact
  (initial-fact))
```

Предопределенный факт *initial-fact* шаблона *initial-fact* предоставляет удобный способ для запуска программ на языке CLIPS — правила, не имеющие условных элементов, автоматически преобразуются в правила с условием, проверяющим наличие факта *initial-fact*. Факт *initial-fact* можно обрабатывать так же, как и все остальные факты CLIPS, добавленные пользователем или программой с помощью команды *assert*.

5. ИСПОЛЬЗОВАНИЕ ФУНКЦИЙ *ASSERT*, *RETRACT* И *MODIFY*

5.1. Функция *assert*

Функция *assert* — одна из наиболее часто применимых команд в системе CLIPS. Без использования этой команды нельзя написать даже самую простую экспертную систему и запустить ее на выполнение в среде

CLIPS. Функция ***assert*** позволяет добавлять факты в список фактов текущей базы знаний. Каждым вызовом этой функции можно добавить произвольное число фактов. В случае если был включен режим просмотра изменения списка фактов, то соответствующее информационное сообщение будет отображаться в окне CLIPS при добавлении каждого факта.

Определение 5. Синтаксис команды ***assert***

(assert <факт>+)

При использовании команды ***assert*** необходимо помнить, что первое поле факта обязательно должно быть значением типа *symbol*. В случае удачного добавления фактов в базу знаний, функция возвращает адрес последнего добавленного факта. Если во время добавления некоторого факта произошла ошибка, команда прекращает свою работу и возвращает значение FALSE.

Слотам неупорядоченного факта, значения которых не заданы, будут присвоены значения по умолчанию.

Пример 8. Использование функции ***assert***

```
(clear)  
(assert (color red))  
(assert (color blue)  
      (value (+ 3 4)))  
(deftemplate status  
      (slot temp)  
      (slot pressure  
      (default low)))  
(assert (status (temp high)))
```

Команда ***clear*** очищает текущий список фактов (а также все определенные конструкторы). В отличие от *reset*, команда ***clear*** не добавляет в список фактов *initial-fact*. Эту команду также можно выполнить, выбрав пункт **Clear** CLIPS в меню **Execution**. При выборе данной команды на экране появляется диалоговое окно, представленное на рис. 11. Это окно запрашивает подтверждение пользователя на очистку текущей базы знаний.

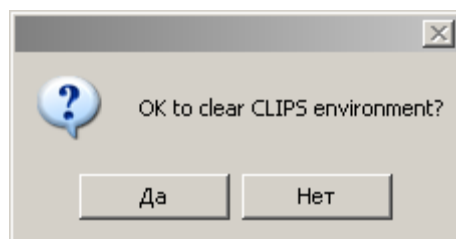


Рис 11. Подтверждение очистки среды CLIPS

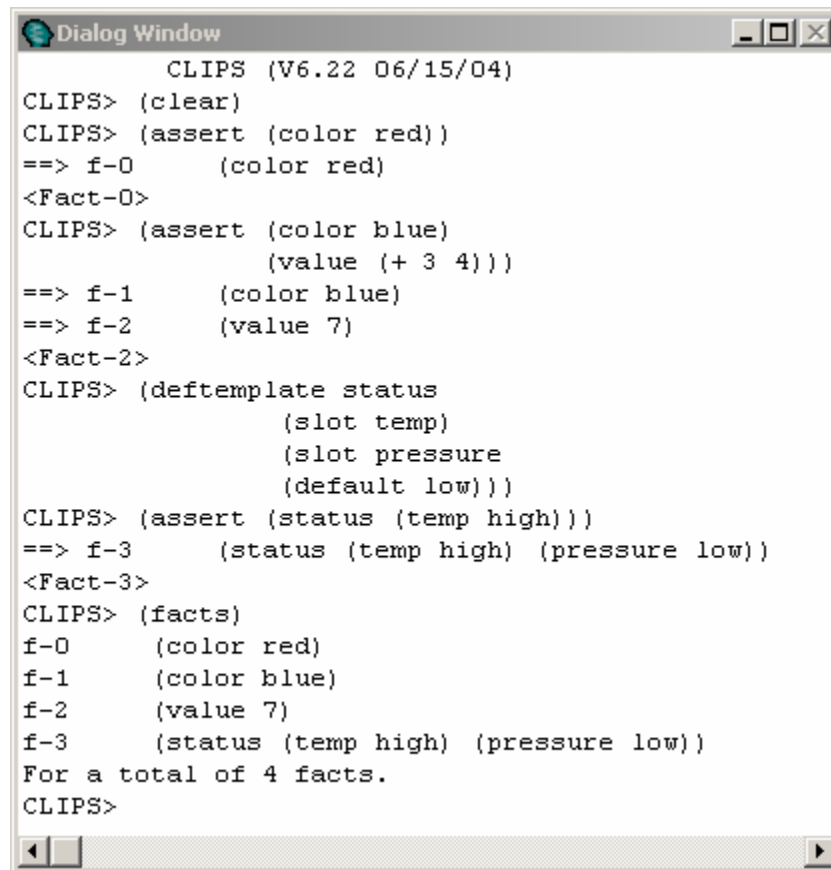
В случае, если команда была набрана с клавиатуры, никакого подтверждения на выполнение этой операции система не запрашивает. Если вы недавно начали работать в среде CLIPS, то для очистки системы лучше использовать меню.

Задание 1 Включите режим просмотра изменения списка фактов и наберите приведенный выше пример. После этого выполните команду (*facts*).

Если при выполнении этих действий не было допущено ошибок, то вы должны получить результат, идентичный изображенному на рис. 12.

Обратите внимание, что при инициализации факта *value* использовалось выражение, а слот *pressure* неупорядоченного факта *status* получил значение по умолчанию *low*.

По умолчанию CLIPS не позволяет добавлять в список фактов два одинаковых факта. Например, попытка добавить два факта *color red* приведет к ошибке и функция ***assert*** вернет значение FALSE. Данную установку системы можно изменить с помощью функции ***set-fact-duplication***. Кроме того, пользователям Windows-версии CLIPS доступен еще один способ настройки. Для этого необходимо открыть диалоговое окно **Execution Options**, выбрав пункт **Options** из меню **Execution**, установить флажок **Fact Duplication**. Внешний вид этого диалогового окна приведен на рис. 6.



```
CLIPS (V6.22 06/15/04)
CLIPS> (clear)
CLIPS> (assert (color red))
==> f-0      (color red)
<Fact-0>
CLIPS> (assert (color blue)
          (value (+ 3 4)))
==> f-1      (color blue)
==> f-2      (value 7)
<Fact-2>
CLIPS> (deftemplate status
          (slot temp)
          (slot pressure
            (default low)))
CLIPS> (assert (status (temp high)))
==> f-3      (status (temp high) (pressure low))
<Fact-3>
CLIPS> (facts)
f-0      (color red)
f-1      (color blue)
f-2      (value 7)
f-3      (status (temp high) (pressure low))
For a total of 4 facts.
CLIPS>
```

Рис 12. Добавление фактов

5.2. Функция *retract*

После добавления факта в базу знаний рано или поздно встанет вопрос о том, как его оттуда удалить. Для удаления фактов из текущего списка фактов в системе CLIPS предусмотрена функция ***retract***. Каждым вызовом этой функции можно удалить произвольное число фактов. Удаление некоторого факта может стать причиной удаления других фактов, которые логически связаны с удаляемым. Кроме того, удаление факта вызывает удаления правил из плана решения текущей задачи, активированных удаляемым фактом. В случае если был включен режим просмотра изменения списка фактов, то соответствующее информационное сообщение будет отображаться в окне CLIPS при удалении каждого факта.

Определение 6. Синтаксис команды *retract*

(retract <определение-факта>+ | *)

Аргумент *<определение-факта>* может являться либо переменной, связанной с адресом факта с помощью правила, либо индексом факта без префикса (например, 3 для факта с индексом $f-3$), либо выражением, вычисляющим этот индекс (например, $(+ 1 2)$ для факта с индексом $f-3$). Если в качестве аргумента функции **retract** использовался символ *, то из текущей базы знаний системы будут удалены все факты. Функция **retract** не имеет возвращаемого значения.

Для демонстрации работы функции **retract** воспользуемся еще одним визуальным инструментом, не описанным ранее. Он предназначен для просмотра содержимого списка фактов в реальном времени. Для того чтобы активизировать просмотр списка фактов, поставьте флажок рядом с пунктом **Facts Window** меню **Windows**, как показано на рис. 13. Внешний вид инструмента просмотра списка фактов показан на том же рисунке. Сразу после запуска CLIPS этот список пуст.

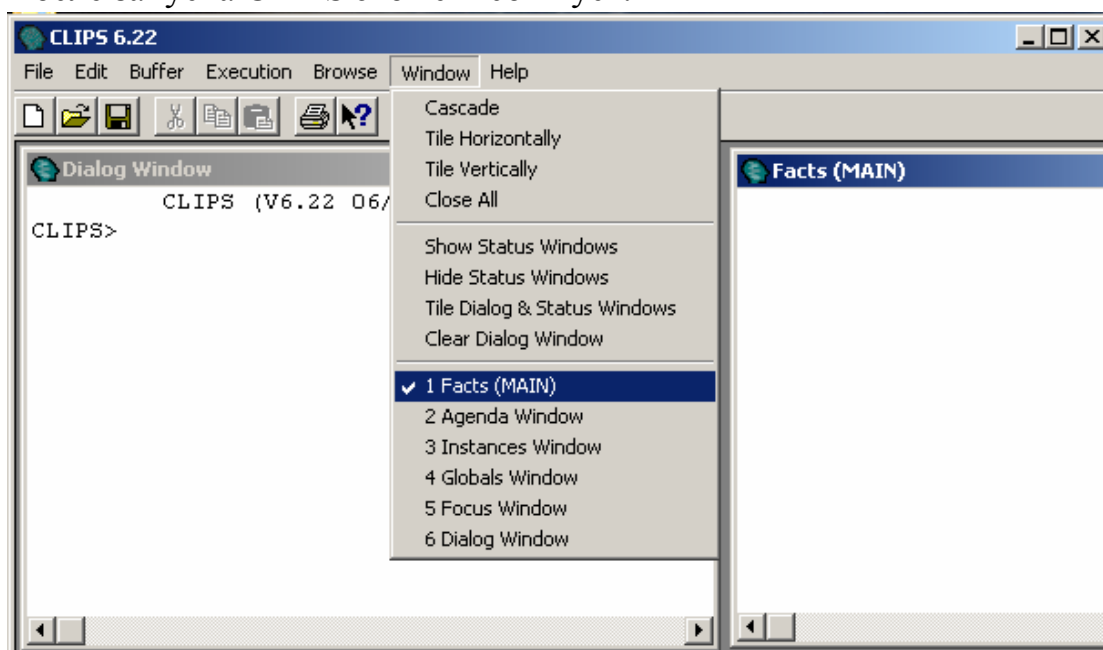


Рис 13. Список фактов

Включите режим просмотра изменения списка фактов с помощью диалогового окна **Watch Options** и добавьте в список фактов следующие факты:

Пример 9. Добавление фактов

(assert (a) (b) (c) (d) (e) (f))

Обратите внимание, что в окне просмотра фактов теперь отображаются все 6 добавленных фактов.

Пример 10. Удаление фактов

(retract 0 (+ 0 2) (+022))

Эта команда удалит все факты с четными индексами, используя индекс факта непосредственно (первый аргумент) и выражение, которое вычисляет индекс факта (второй и третий аргумент). Если перечисленные команды были выполнены правильно, то результат должен соответствовать рис. 14.

В случае, если факт с указанным индексом не будет найден, CLIPS выдаст соответствующее сообщение об ошибке

Выполните команду:

Пример 11. Удаление всех фактов

(retract *)

После выполнения данной команды список фактов будет очищен полностью и окно отображения текущего состояния списка фактов станет идентично изображенному на рис. 13.

Необходимо заметить, что функция *retract* не оказывает никакого воздействия на индекс следующих добавленных фактов, т.е. этот индекс не обнуляется. Если после удаления всех введенных фактов добавить в систему какой-нибудь факт, то он получит индекс *f-6*, несмотря на то, что список фактов в данный момент пуст.

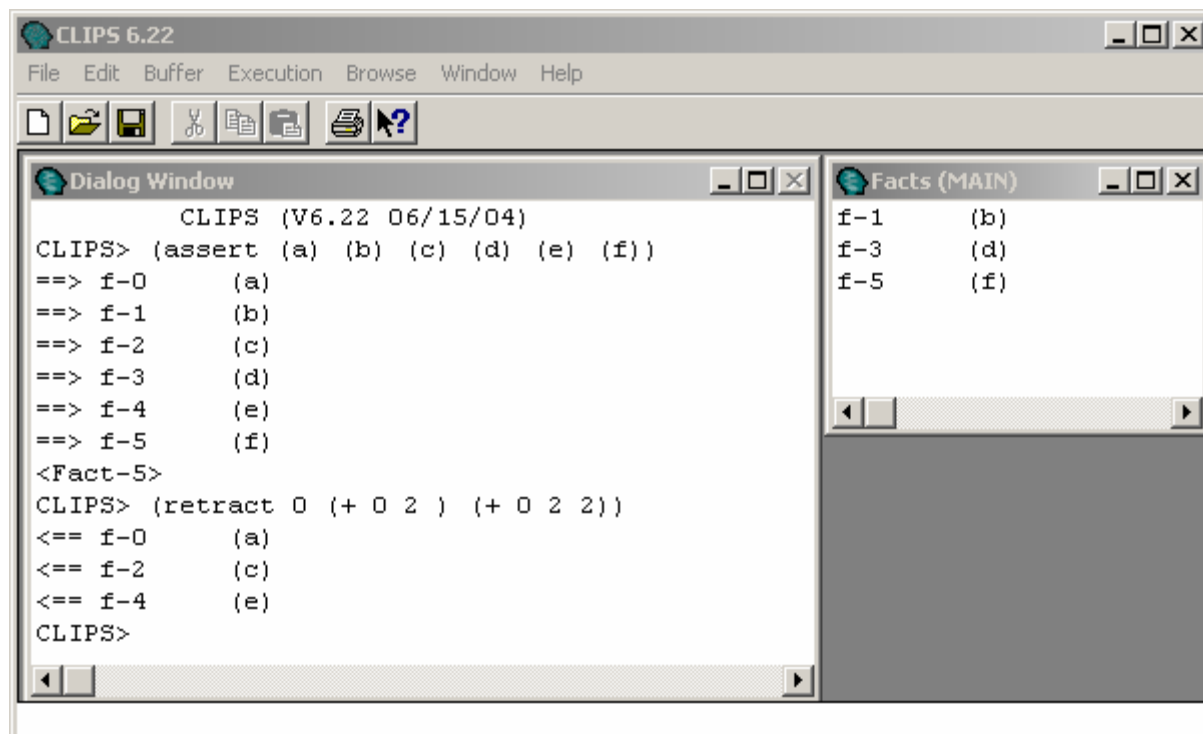


Рис 14. Результат добавления и удаления фактов

5.3. Функция *modify*

Используя функции *assert* и *retract*, можно выполнять большинство необходимых для функционирования правил действий. В том числе и изменения существующего факта. Например, если в список фактов ранее был добавлен факт (*temperature is low*), который получил индекс 0, то изменить его значение можно, например, следующим образом:

Пример 12. Изменение существующего факта

```
(clear)
(assert (temperature is low))
(retract 0)
(assert (temperature is high))
```

Для изменения упорядоченных фактов доступен только этот способ. Для упрощения операции изменения неупорядоченных фактов CLIPS предоставляет функцию *modify*, которая позволяет изменять значения слотов таких фактов. *Modify* просто упрощает процесс изменения факта, но ее внутренняя реализация эквивалентна вызовам пар функций *retract* и *assert*. За один вызов *modify* позволяет изменять только один факт. В случае удачного выполнения функция возвращает новый индекс модифицированного факта. Если в процессе выполнения произошла какая-либо ошибка, то пользователю выводится соответствующее предупреждение и функция возвращает значение FALSE.

Определение 7. Синтаксис команды *modify*

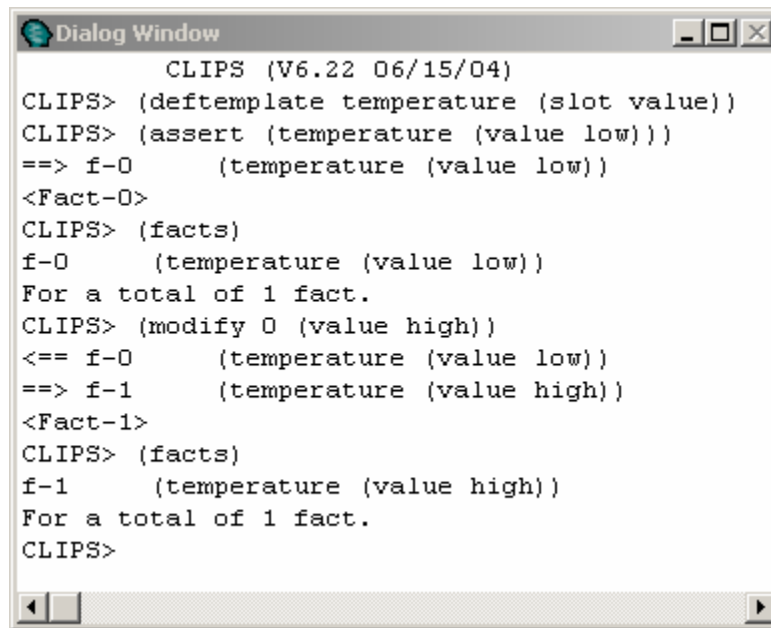
```
(modify <определение-факта>
      <новое-значение-слота>+)
```

Аргументом <определение-факта> может быть либо переменная, связанная с адресом факта с помощью правила, либо индекс факта без префикса (например, 3 для факта с индексом *f-3*). После определения факта следует список из одного или более новых значений слотов указанного шаблона. Для использования приведенного выше примера его необходимо переделать следующим образом

Пример 13. Изменение существующего неупорядоченного факта

```
(deftemplate temperature (slot value))
(assert (temperature (value low)))
(modify 0 (value high))
```

Если включить режим просмотра изменения списка фактов и выполнить приведенные выше команды, то полученный результат должен соответствовать рис. 15.



```
CLIPS (V6.22 06/15/04)
CLIPS> (deftemplate temperature (slot value))
CLIPS> (assert (temperature (value low)))
==> f-0      (temperature (value low))
<Fact-0>
CLIPS> (facts)
f-0      (temperature (value low))
For a total of 1 fact.
CLIPS> (modify 0 (value high))
<== f-0      (temperature (value low))
==> f-1      (temperature (value high))
<Fact-1>
CLIPS> (facts)
f-1      (temperature (value high))
For a total of 1 fact.
CLIPS>
```

Рис 15. Результат изменения существующего неупорядоченного факта

Обратите внимание на движение фактов в базе знаний CLIPS при выполнении функции *modify* — сначала удаляется старый факт с индексом *f-0*, а затем добавляется новый факт с индексом *f-1*, идентичный предыдущему, но с новым значением заданного слота.

Если в шаблоне заданного факта отсутствует слот, значение которого требуется изменить, CLIPS выведет соответствующее сообщение об ошибке. Если заданный факт отсутствует в списке фактов, пользователь также получит соответствующее предупреждение.

6. ИСПОЛЬЗОВАНИЕ ФУНКЦИЙ *DUPLICATE*, *ASSERT-STRING*, *FACT-EXISTP*

6.1. Функция *duplicate*

Помимо функции *modify*, в CLIPS существует еще одна очень полезная функция, упрощающая работу с фактами, — функция *duplicate*. Эта функция создает новый неупорядоченный факт заданного шаблона и копирует в него определенную пользователем группу полей уже существующего факта того же шаблона. По действиям, которые выполняет, функция *duplicate* аналогична *modify*, за исключением того, что она не удаляет старый факт из списка фактов. Одним вызовом функции

duplicate можно создать одну копию некоторого заданного факта. Как и функция *modify*, *duplicate*, в случае удачного выполнения, возвращает индекс нового факта, а в случае неудачи — значение FALSE.

Определение 8. Синтаксис команды *duplicate*

```
(duplicate      <определение-факта>
                <новое-значение-слота>+)
```

Аргумент *<определение-факта>* может быть либо переменной, связанной с адресом факта с помощью правила, либо индексом факта без префикса. После определения факта следует список из одного или более новых значений слотов указанного шаблона. Продемонстрируем работу данной функции на следующем примере:

Пример 14. Создание копии существующего неупорядоченного факта

```
(deftemplate car
  (slot name)
  (slot producer)
  (slot type)
  (slot max-speed))

(assert ( car
  (name scorio)
  (producer ford)
  (type sedan)
  (max-speed 180)))

(duplicate 0
  (type off-road)
  (max-speed 130))
```

В приведенном примере определяется шаблон, описывающий свойства автомобиля, и добавляется факт — автомобиль Ford Scorio с типом кузова седан и максимальной скоростью 180 (км/ч). После этого с помощью функции *duplicate* добавляется факт с информацией об еще одном автомобиле с похожими характеристиками — это внедорожник Ford Scorio с максимальной скоростью 130 (км/ч). *Duplicate* просто облегчает нам жизнь, избавляя от излишнего ввода значений данных совпадающих слотов (см. рис. 16).

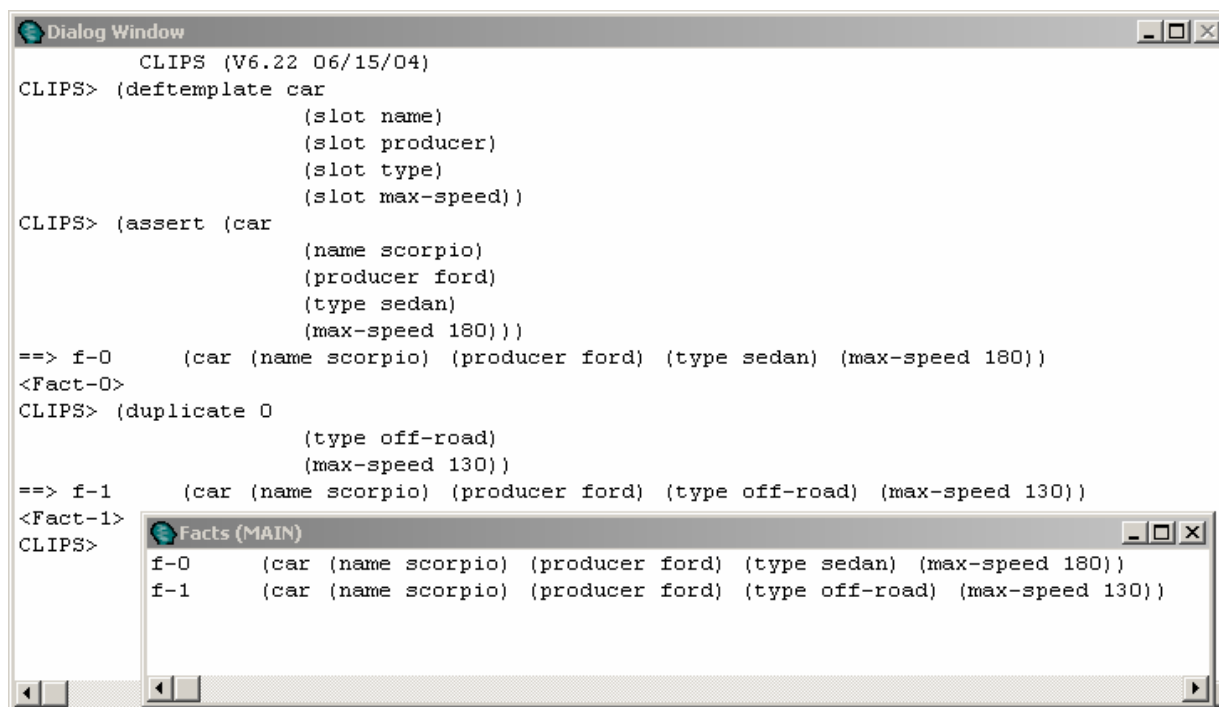


Рис 16. Добавление факта с похожими характеристиками

В случае, если добавляемый с помощью *duplicate* факт уже присутствует в списке фактов, будет выдана соответствующая информация об ошибке и возвращено значение FALSE. Факт при этом добавлен не будет. Это поведение можно изменить, разрешив существование одинаковых фактов в базе знаний.

6.2. Функция *assert-string*

Кроме функции *assert*, CLIPS предоставляет еще одну функцию, полезную при добавлении фактов, — *assert-string*. Эта функция принимает в качестве единственного аргумента символьную строку, являющуюся текстовым представлением факта (в том виде, в котором вы набираете его, например, в функции *assert*), и добавляет его в список фактов. Функция *assert-string* может работать как с упорядоченными, так и с неупорядоченными фактами. Одним вызовом функции *assert-string* можно добавить только один факт.

Определение 9. Синтаксис команды *assert-string*

(assert-string <строковое-выражение>)

Строковое выражение должно быть заключено в кавычки. Функция преобразует заданное строковое выражение в факт CLIPS, разделяя отдельные слова на поля, с учетом определенных в системе на текущий

момент шаблонов. Если в строке необходимо записать внутреннее строковое выражение, представляющее, скажем, некоторое поле, то для включения в строковое выражение символа кавычек используется обратная косая черта (*backslash*). Например, факт (*book-name "CLIPS User Guide"*) можно добавить следующим образом:

Пример 15. Использование кавычек внутри строки

```
(assert-string "(book-name \"CLIPS User Guide\")")
```

Для добавления содержащегося в поле символа обратной косой черты используйте ее дважды. Если обратная косая должна содержаться внутри подстроки, ее необходимо использовать четыре раза. Например, для помещения в текущий список факта (*a\b "c\d"*) необходимо вызвать функцию *assert-string* со следующим строковым аргументом:

Пример 16. Использование обратной косой черты

```
(assert-string "(a\\b \"c\\\\d\")")
```

Если добавление факта прошло удачно, функция возвращает индекс только что добавленного факта, в противном случае функция возвращает сообщение об ошибке и значение FALSE. Функция *assert-string* не позволяет добавлять факт в случае, если такой факт уже присутствует в базе знаний (если вы еще не включили возможность присутствия одинаковых фактов).

6.3. Функция *fact-existp*

Рассмотрим очень простую, но чрезвычайно важную функцию *fact-existp*. Эта функция определяет, присутствует ли в данный момент факт, заданный индексом или переменной указателем, в базе знаний системы. В случае если факт присутствует в списке фактов, функция возвращает значение TRUE, иначе — FALSE.

Определение 10. Синтаксис команды *fact-existp*

```
(fact-existp <определение-факта>)
```

Приведем простой пример использования данной функции:

Пример 17. Использование функции *fact-existp*

```
(clear)
(assert-string "(a\\b \"c\\\\d\")")
(fact-existp 0)
(retract 0)
(fact-existp 0)
```

Не забудьте выполнить команду *clear*, чтобы добавленный факт имел нулевой индекс. После первого вызова функция *fact-existp* вернет значение TRUE, а после удаления факта с индексом 0 — FALSE.

7. ФУНКЦИИ ДЛЯ РАБОТЫ С НЕУПОРЯДОЧЕННЫМИ ФАКТАМИ: *FACT-RELATION*, *FACT-SLOT-NAMES* И *FACT-SLOT-VALUE*

7.1. Функция *fact-relation*

Функция *fact-relation* позволяет получить связь (relation) существующего факта с шаблоном. Связь факта с шаблоном, определенным с помощью конструктора *deftemplate* или неявно созданным шаблоном, определяется по первому полю факта. Это поле всегда является простым полем и используется CLIPS в качестве имени шаблона, с которым связан факт. Таким образом, функция *fact-relation* просто возвращает первое поле факта, или значение FALSE, если указанный факт не найден.

Определение 11. Синтаксис команды *fact-relation*

(fact-relation <определение-факта>)

В качестве определения факта, как и в описанных выше функциях, нужно использовать или переменную указатель, содержащую адрес факта, или индекс факта.

Пример 18. Использование функции *fact-relation*

```
(clear)
(assert (car Ford))
(fact-relation 0)
(retract 0)
(fact-relation 0)
```

В первом случае функция *fact-relation* вернет значение *car*, а во втором — FALSE.

7.2. Функция *fact-slot-names*

Для получения имен всех слотов заданного факта в CLIPS предназначена функция *fact-slot-names*.

Определение 12. Синтаксис команды *fact-slot-names*

(fact-slot-names <определение-факта>)

Данная функция возвращает список имен слотов в составном поле. Для упорядоченных фактов функция возвращает значение *implied* (подразумеваемый), т. к., если вы помните, CLIPS представляет упорядоченные факты как неявно заданные неупорядоченные с одним составным слотом. В случае если заданный факт не найден, функция возвращает значение FALSE.

Пример 19. Использование функции *fact-slot-names*

```
(clear) (deftemplate car
          (slot name)
          (slot producer)
          (slot type)
          (slot max-speed))
(assert ( car
          (name scorio)
          (producer ford)
          (type sedan)
          (max-speed 180)))
(fact-slot-names 0)
```

Если приведенный пример был набран без ошибок, то функция *fact-slot-names* вернет значение *(name producer type max-speed)*.

7.3. Функция *fact-slot-value*

Для работы с неупорядоченными фактами используют функцию *fact-slot-value*.

Определение 13. Синтаксис команды *fact-slot-value*

```
(fact-slot-value <определение-факта> <имя-слота >)
```

Данная функция позволяет получать значения слота некоторого заданного факта. Если факт является упорядоченным, то для получения значения неявно определенного составного слота используется значение *implied*. В случае если указанный факт не существует, или имя слота указано не верно, функция возвращает значение FALSE.

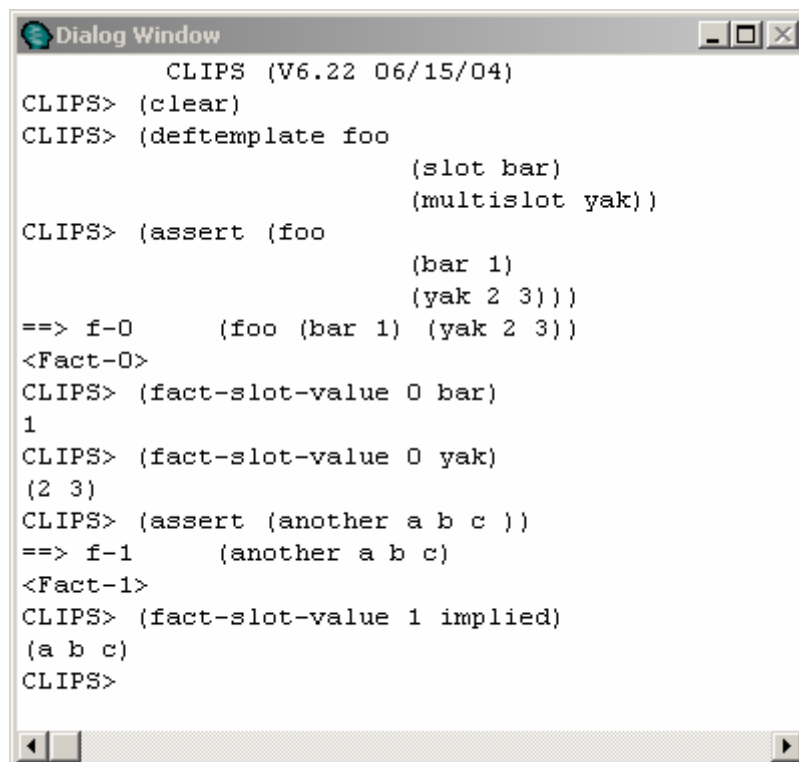
Выполните в среде CLIPS следующий пример:

Пример 20. Использование функции *fact-slot-value*

```
(clear)
(deftemplate foo
          (slot bar)
          (multislot yak))
(assert (foo (bar 1) (yak 23)))
```

```
(fact-slot-value 0 bar)
(fact-slot-value 0 yak)
(assert (another a b c))
(fact-slot-value 1 implied)
```

Если предыдущий пример был выполнен без ошибок, то полученный результат должен соответствовать приведенному на рис. 17.



```
Dialog Window
CLIPS (V6.22 06/15/04)
CLIPS> (clear)
CLIPS> (deftemplate foo
      (slot bar)
      (multislot yak))
CLIPS> (assert (foo
      (bar 1)
      (yak 2 3)))
==> f-0      (foo (bar 1) (yak 2 3))
<Fact-0>
CLIPS> (fact-slot-value 0 bar)
1
CLIPS> (fact-slot-value 0 yak)
(2 3)
CLIPS> (assert (another a b c))
==> f-1      (another a b c)
<Fact-1>
CLIPS> (fact-slot-value 1 implied)
(a b c)
CLIPS>
```

Рис 17. Результат использования функции **fact-slot-value**

8. ФУНКЦИИ СОХРАНЕНИЯ И ЗАГРУЗКИ СПИСКА ФАКТОВ *SAVE-FACTS И LOAD-FACTS*

Как можно заметить, наполнение списка фактов в CLIPS довольно кропотливое и длительное занятие. Если фактов достаточно много, этот процесс может растянуться на несколько часов, или даже дней. Так как список фактов хранится в оперативной памяти компьютера, теоретически, из-за сбоя компьютера или, например, неожиданного отключения питания, список фактов можно безвозвратно потерять. Чтобы этого не произошло, а так же для того чтобы сделать работу по наполнению базы знаний фактами более удобной, CLIPS предоставляет команды сохранения и загрузки списка фактов в файл — *save-facts* и *load-facts* соответственно.

8.1. Функция *save-facts*

Команда *save-facts* сохраняет факты из текущего списка фактов в текстовый файл. На каждый факт отводится одна строка. Неупорядоченные факты сохраняются вместе с именами слотов.

Определение 14. Синтаксис команды *save-facts*

(save-facts <имя-файла> [<границы-видимости> <список-шаблонов>])

<границы-видимости> ::= visible|local

В функции существует возможность ограничить область видимости сохраняемых фактов. Для этого используется аргумент *<граница-видимости>*. Он может принимать значение *local* или *visible*. В случае если этот аргумент принимает значение *visible*, то сохраняются все факты, присутствующие в данный момент в системе. Если в качестве аргумента используется ключевое слово *local*, то сохраняются только факты из текущего модуля. По умолчанию аргумент *<границы-видимости>* принимает значение *local*. После аргумента *<границы-видимости>* может следовать список определенных в системе шаблонов. В этом случае будут сохранены только те факты, которые связаны с указанными шаблонами.

Пример 21. Использование функции *save-facts*

```
(clear)
(deftemplate template
  (slot a)
  (slot b))
(assert (template (a 1) (b 2)))
(assert (simple-fact1) (simple-fact2))
(save-facts f1 local template simple-fact1)
```

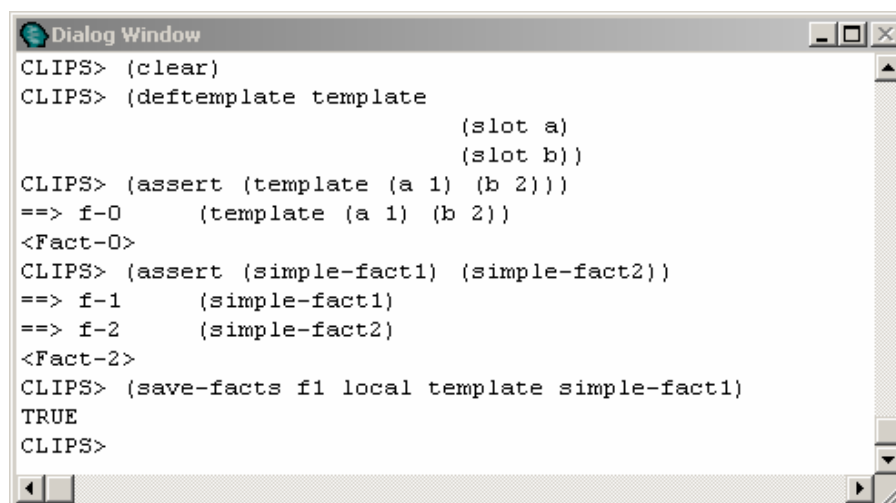


Рис 18. Использование функции **save-facts**

Последовательность действий, приведенная в данном примере, сохраняет файл *f1*, находящийся в текущем каталоге, все факты, видимые в текущем модуле и связанные с шаблонами *template* и *simple-fact1* (как вы помните, после добавления факта *simple-fact1* CLIPS определяет неявно созданный шаблон *simple-fact1*). В результате будет получен текстовый файл со следующим содержанием:

Пример 22. Содержание файла *f1*

```
(template (a 1) (b 2))  
(simple-fact1)
```

В случае успешного выполнения, команда возвращает значение TRUE, а в случае неудачи — соответствующее сообщение об ошибке. Если указанный файл уже существует, он будет перезаписан.

8.2. Функция *load-facts*

Для загрузки сохраненных ранее файлов используется функция *load-facts*. Функция имеет следующий формат:

Определение 15. Синтаксис команды *load-facts*

```
(load-facts <имя-файла>)
```

Здесь <имя-файла> — имя текстового файла, сохраненного ранее с помощью команды *save-facts*, содержащего список фактов. Файл со списком фактов можно также создать в любом текстовом редакторе, если вы хорошо разобрались с представлением фактов в CLIPS. Для загрузки сохраненного в предыдущем примере файла выполните:

Пример 23. Использование функции *load-facts*

```
(load-facts f1)
```

В случае успешного выполнения команда возвращает значение TRUE, а в случае неудачи — FALSE и соответствующее сообщение об ошибке. Обратите внимание, что если в файле содержатся факты, связанные с явно созданными с помощью конструктора *deftemplate* шаблонами, то в момент загрузки все необходимые шаблоны должны быть уже определены в системе. Если это условие не будет выполнено, то загрузка фактов закончится неудачно. (К счастью, CLIPS также позволяет и загрузку конструкторов из текстового файла см. в других лабораторных работах).

ЗАДАНИЕ

Используя конструкторы *deftemplate* и *deffacts* создайте 10 упорядоченных и 10 неупорядоченных фактов, описывающих предметную область, заданную преподавателем.

9. КОНТРОЛЬНЫЕ ВОПРОСЫ:

1. Для каких основных целей используют язык программирования CLIPS?
2. Кто является разработчиком CLIPS?
3. Как запустить экспертную систему, созданную в CLIPS?
4. Какие правила синтаксиса в CLIPS, используемые для построения определений, Вы знаете?
5. Какие два типа фактов поддерживает система CLIPS? Охарактеризуйте их.
6. Какой конструктор используется в CLIPS для создания неупорядоченных фактов? Приведите примеры его синтаксиса.
7. Какой конструктор используется в CLIPS для создания списка предопределенных фактов? Приведите примеры его синтаксиса.
8. Для каких целей используются функций *assert*, *retract* и *modify* в CLIPS?
9. Для каких целей используются функций *fact-relation*, *fact-slot-names* и *fact-slot-value* в CLIPS?
10. Как осуществляется сохранение списка фактов при использовании функции *save-facts*?
11. Как осуществляется загрузка списка фактов при использовании функции *load-facts*?