Практическая работа № 7
«Изучение основ разработки приложений экспертных систем в среде CLIPS»

Теоретический материал

Язык CLIPS, который разрабатывался с 1985 г. космическим агентством NASA (США), в настоящее время находится в свободном доступе, а основное его назначение – создание интеллектуальных систем, в первую очередь на базе продукционной модели. Хотя в последнее время активного развития языка CLIPS как такового не происходит, на его основе были созданы такие средства как JESS (реализация языка разработки экспертных систем на Java) и FuzzyCLIPS (среда для разработки систем, использующих нечёткую логику), что подтверждает популярность среды CLIPS и адекватность основных подходов, заложенных в ней. CLIPS сочетает в себе 3 парадигмы программирования: логическую, процедурную и объектно-ориентированную. Также, в CLIPS предусмотрены 3 основных формата представления информации: факты, глобальные переменные и объекты. В CLIPS используется оригинальный LISP – подобный язык программирования, ориентированный на разработку экспертных систем (ЭС). CLIPS поддерживает две парадигмы программирования: объектно-ориентированную и процедурную.

Основные операции:

−команда run запускает выполнение программы в среде (с машиной логического вывода)

−reset – обновляет рабочую память

−clear – полностью очищает среду clips (рабочую память).

−опция главного меню load позволяет загрузить файл с конструкциями, а файл load batch – файл с командами (программу).

Факт представляет собой основную единицу данных, используемую правилами. Факт может описываться индексом или адресом. Всякий раз, когда факт добавляется (изменяется) ему присваивается уникальный целочисленный индекс. Индексы в fact-list начинаются с нуля.

Идентификатор факта – это короткая запись факта, которая состоит из символа факта – f и индекса факта (f-10). Например:

f-0 (today is Sunday)
f-1 (weather is warm)

Факты представимы в двух форматах: позиционные и непозиционные.

Позиционные факты – состоят из выражения символьного типа, за которым следует последовательность (возможно, пустая) из полей, разделенных пробелами. Вся запись заключается в скобки. Для того чтобы обратиться к информации, содержащейся в позиционном факте, пользователь должен знать какие данные содержаться в факте и в каком поле они хранятся.

Пример:

(altitude is 10000 feet) (today is Sunday)

(grocery_list bread milk eggs) (weather is warm)

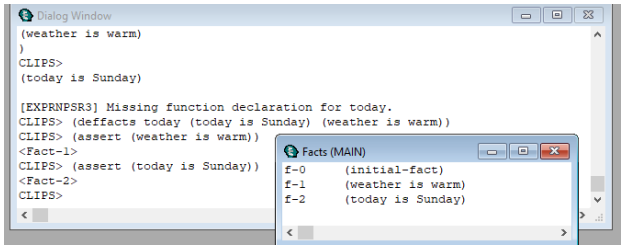
Поля в позиционных фактах могут быть любого простого типа, за исключением первого поля, которое всегда должно быть типа symbol.

Факты можно добавлять к списку фактов (assert)

В режиме интерпретатора пользователь может использовать множество команд, так можно включить в базу фактов прямо из командной строки с помощью assert, например:

CLIPS> (assert (today is Sunday))

CLIPS> (assert (weather is warm))



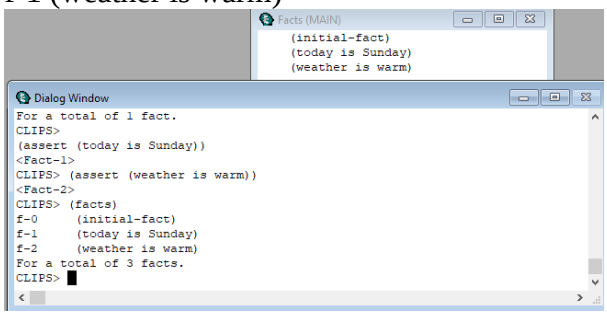
Для вывода списка фактов, имеющихся в базе, используется команда facts:

CLIPS> (facts)

В результате получим:

f-0 (today is Sunday)

f-1 (weather is warm)



Для удаления фактов из базы используется команда retract.

CLIPS> (retract 1)

Clear - очищает базу фактов (как правило, эта команда доступна в одном из выпадающих меню).

Команда reset сначала очищает базу фактов, а затем включает в нее факты из всех ранее загруженных массивов. Она также добавляет в базу единственный определенный системой факт:

Функция (printout t ”Текст” crlf) - выводит на стандартное устройство вывода (монитор) текст, заключенный в кавычки. Набор символов crlf соответствует переводу строки и возврату в начало строки.

f-0 (initial-fact)

Факты можно изменять (modify), дублировать (duplicate)

Факты можно включать в базу не по одиночке, а целым массивом. Для этого в CLIPS имеется команда deffacts:

(deffacts имя_базы_данных [необязательный комментарий] (факт_1) . . . . . (факт_N))

(deffacts today (today is Sunday) (weather is warm))

Выражение начинается с команды deffacts, затем приводится имя списка фактов, который программист собирается определить (в нашем примере – today), а за ним следуют элементы списка, причем их количество не ограничивается.

В CLIPS существуют следующие зарезервированные слова, которые не могут использоваться как первое поле любого факта: test, and, or, not, declare, logical, object, exists, forall.

Непозиционные факты (шаблонные факты) – реализуются через конструкцию, подобную структуре или записи в языках С и Паскаль. Шаблонные факты позволяют задавать имена каждому из полей факта.

Для задания шаблона, который затем может использоваться при доступе к полям по именам, используется конструкция

(deftemplate <name>)

(slot-1)

(slot-2)

. . . . . .

(slot-N)

где <name> – имя шаблона, (slot-N) – именованное поле (или слот). Слоты могут быть ограничены по типу, значению, числовому диапазону, могут содержать значение по умолчанию. Порядок следования слотов значения не имеет.

Пример:

(deftemplate student
“a student record”
(slot name (type STRING))
(slot age (type NUMBER) (default 18))
)

Каждое определение шаблона состоит из произвольного имени шаблона, необязательного комментария и некоторого количества определений слотов (начинаются с ключевого слова slot или field). Слот включает поле данных, например name, и тип данных, например STRING. Можно указать и значение по умолчанию, как в приведенном выше примере, где возраст студента по умолчанию равен 18.

Если в программу включено приведенное выше определение шаблона, то выражение

(deffacts students
(student (name “fred”))
(student (name “jack”) (age 19))
)

приведет к тому, что в базу фактов после выполнения команды reset будет добавлено

(student (name “fred”) (age 18))
(student (name “jack”) (age 19))

Работа с правилами

(defrule <имя правила>
< необязательный комментарий >
< необязательное объявление >
< предпосылка_1 >
< предпосылка_m >
=>
< действие_1 >
< действие_n >)

2. Задание. Рассмотрим предметную область, которая представляет участников некоторой конференции, приехавших из разных городов. На подобных мероприятиях все участники обычно проходят регистрацию. Пусть эта процедура представляет собой ввод сведений об участниках в базу данных, в которой на каждого участника выделяется одна запись (факт), состоящая из списка с тремя полями. Пусть первое поле имеет символьное значение rep – сокращение от representative (представитель). В общем случае это значение может быть любым, а поле может отсутствовать. Во втором поле списка хранится фамилия участника, а в третьем – город, из которого участник прибыл.

Порядок выполнения работы:

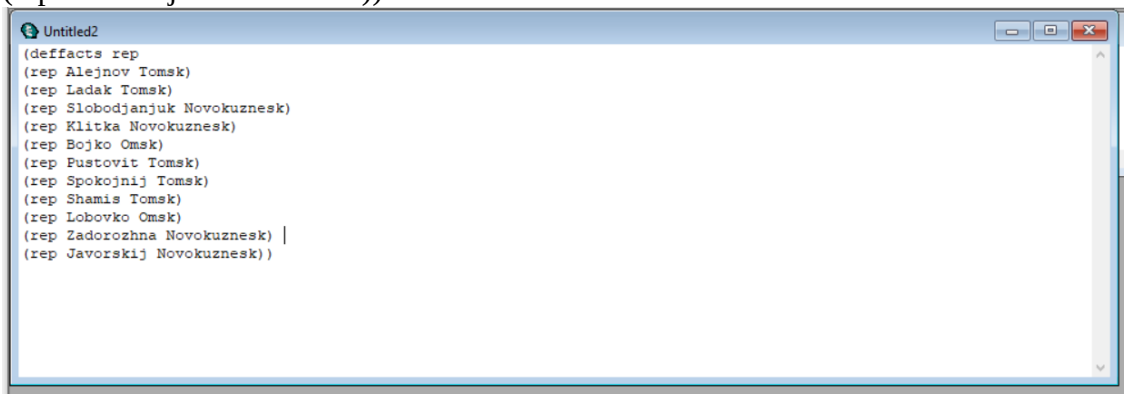
1. Добавим факты в программу Clips:

– создать новый файл File / New

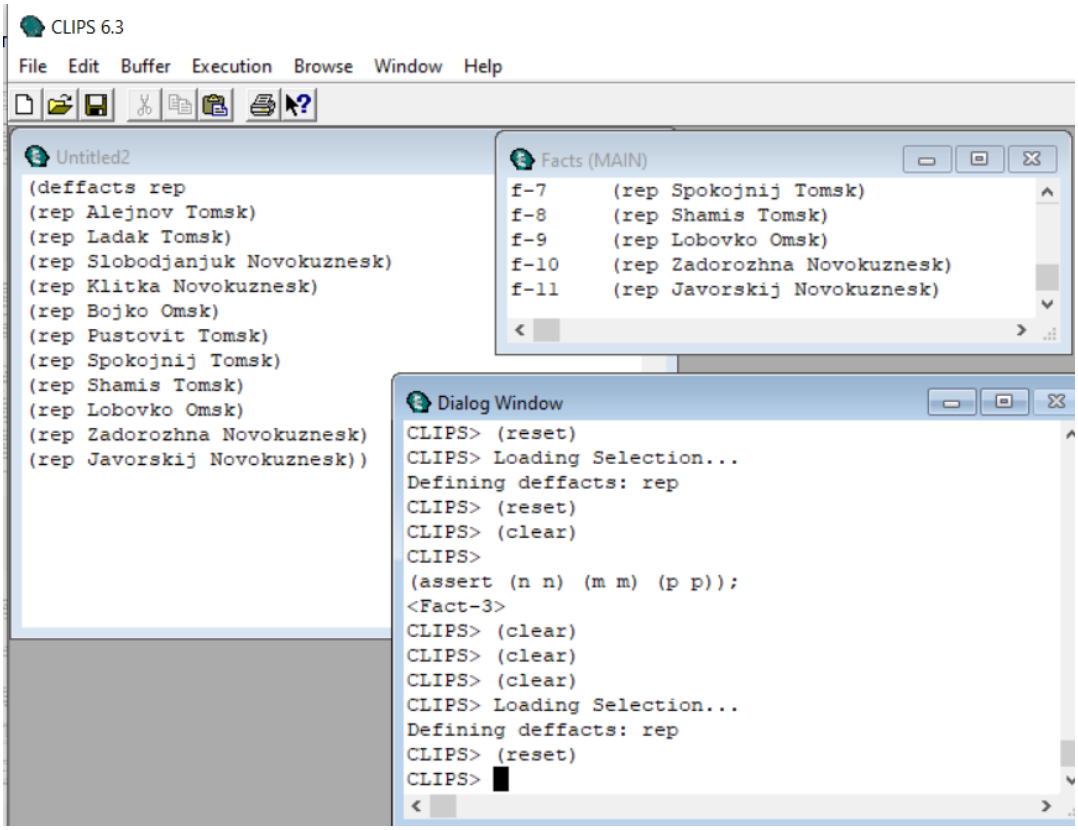
– ввести текст конструктора в открывшемся окне, приведенный ниже, для добавления массива фактов в БЗ

(deffacts rep
(rep Alejnov Tomsk)
(rep Ladak Tomsk)
(rep Slobodjanjuk Novokuznesk)
(rep Klitka Novokuznesk)
(rep Bojko Omsk)
(rep Pustovit Tomsk)

(rep Spokojnij Tomsk)
(rep Shamis Tomsk)
(rep Lobovko Omsk)
(rep Zadorozhna Novokuznesk)
(rep Javorskij Novokuznesk))



– Открыть окно текущего списка фактов (Windows / Facts).
– Очистить память (Execution / Clear CLIPS) и загрузить введенный текст (Buffer / Load Buffer).
– Выполнить (reset) и наблюдать изменение текущего списка фактов.
– Сохранить как rep.clp



2. После окончания конференции организаторы подводят итоги, определяя массу показателей. В частности, пусть требуется определить количество представителей от каждого города. Алгоритм решения такой задачи прост. Для каждого города задаем счетчик и последовательно просматриваем списки в записях файла rep. Если в записи третье поле списка имеет значение Omsk, то содержимое соответствующего счетчика увеличиваем на единицу. Для других городов – аналогично.

Напишем программу: File / New

3. Объявить переменные. При помощи конструктора defglobal объявим три глобальных переменных:

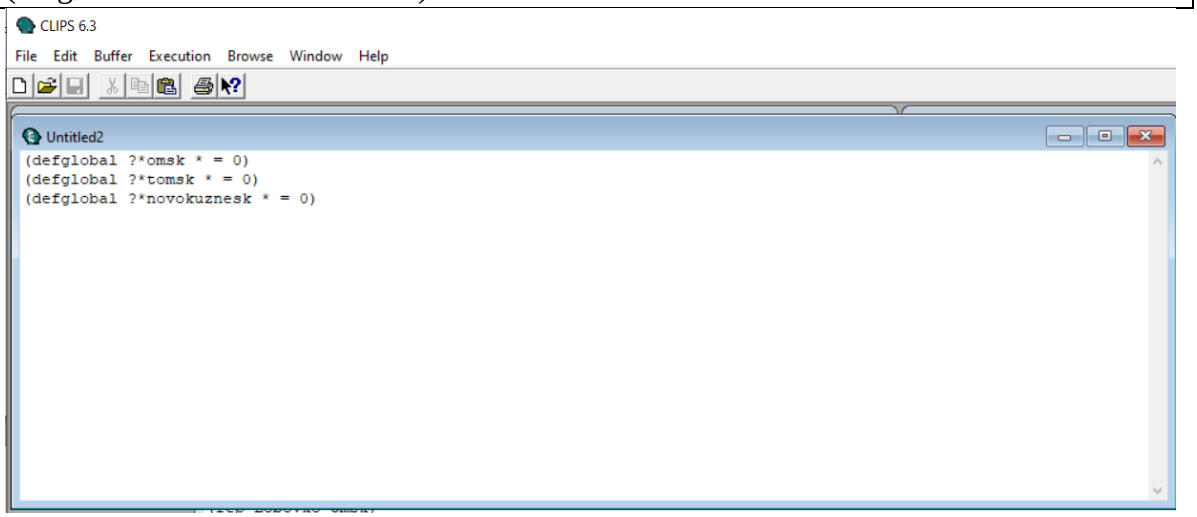
?* omsk *

?*tomsk*

?*novokuznesk*

Эти переменные являются счетчиками. В CLIPS переменная может быть и локальной – но тогда она связывается только с тем правилом, в котором объявляется.

```
(defglobal ?*omsk * = 0)
(defglobal ?*tomsk * = 0)
(defglobal ?*novokuznesk * = 0)
```



4. Задать правила.

Правила, взаимодействующие с базой данных в виде фактов, вносят в нее функциональность и образуют вместе с ней базу знаний. Для создания правила используется конструктор defrule, который имеет следующий синтаксис:

(defrule имя_правила

[необязательный комментарий]

[необязательное объявление]

(условие_1) . . . . . .

(условие_M) =>

(действие_1) . . . . . .

(действие_N))

Следует обратить внимание, что список условий отделяется от действий (*Если условие, то действие*) комбинацией символов “=>” и что количество условий и действий в правиле в общем случае не совпадает.

Правило

```
(defrule start
(initial-fact)
 =>
(printout t crlf “REPRESENTATIVES” crlf) )
; Правило representative (представитель)
```

Текст после точки с запятой «;» – это комментарий и системой не обрабатывается.

Функция (printout t crlf ” REPRESENTATIVES ” crlf) - выводит на стандартное устройство вывода (монитор) текст – REPRESENTATIVES

Следующие три правила с именами odessa, kiev и lvov можно назвать ядром программы. В них производится подсчет количества участников соответственно из Омска, Томска и Новокузнецка.

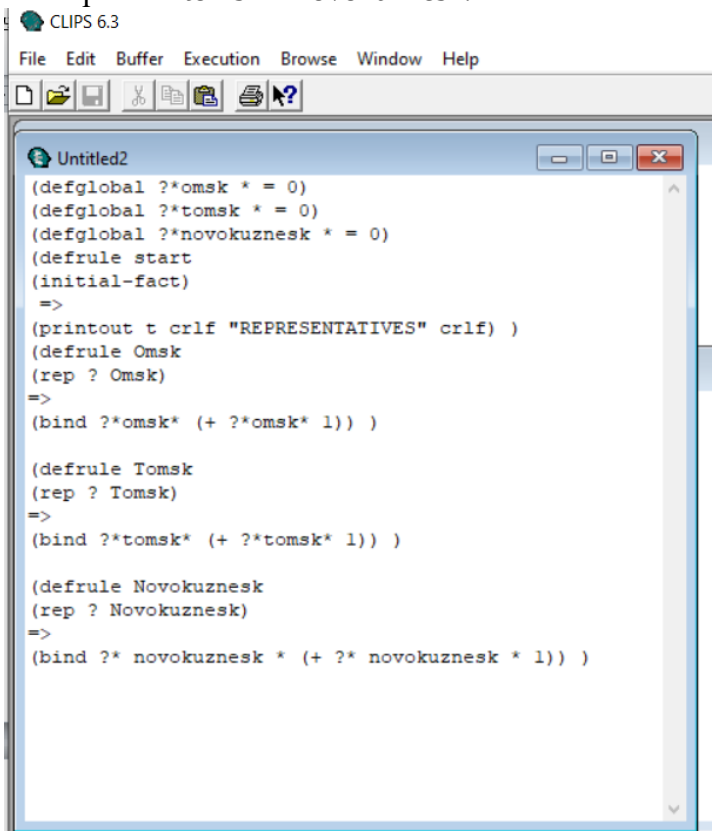
```
(defrule Omsk
```

```
(rep ? Omsk)
=>
(bind ?*omsk* (+ ?*omsk* 1)) )

(defrule Tomsk
(rep ? Tomsk)
=>
(bind ?*tomsk* (+ ?*tomsk* 1)) )

(defrule Novokuznesk
(rep ? Novokuznesk)
=>
(bind ?* novokuznesk * (+ ?* novokuznesk * 1)) )
```

Рассмотрим, например, правило Omsk. Оно активизируется в том случае, когда в базе данных находится факт (rep ? Omsk). Не трудно догадаться, что символ "?" во втором поле этого списка означает символ универсальной подстановки и заменяет собой любую фамилию. Отсюда следует, что правило Omsk активизируется столько раз, сколько раз факт (rep ? omsk) присутствует в базе данных. Встроенная функция bind — аналог оператора присваивания. Следовательно, содержимое переменной ?* omsk * увеличивается на единицу, и результат сохраняется в этой же переменной. Аналогично работают правила tomsk и novokuznesk.




```
(defrule result
(declare (salience –1))
 (initial-fact)
=>
(printout t “from Omsk ” ?*omsk* crlf)
 (printout t “from Tomsk: ” ?*tomsk* crlf)
```

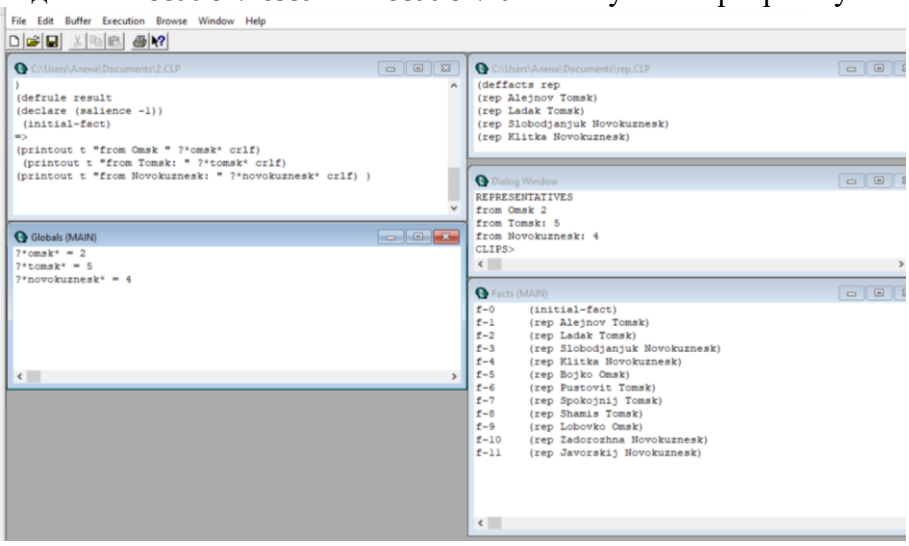
(printout t “from Novokuznesk: ” ?* novokuznesk * crlf))

Действия, которые выполняются в последнем правиле программы, отражены в его названии. В CLIPS существует несколько стратегий очередности выполнения правил, а сами правила могут иметь приоритет, который задается встроенной функцией declare с параметром salience (выпуклость). Этот параметр может принимать целочисленные значения от –10000 до +10000. По умолчанию для всех правил величина salience равна нулю. Если в правиле result не указать приоритет, то оно будет конфликтовать с правилом start за очередность выполнения. Для устранения конфликта в правиле result приоритет указан явно, и со знаком минус, в связи с чем, это правило выполнится последним.

4. Сохранить программный код с расширением .clp и с именем represent. File/Save as/ имя: represent.clp

5. Работа в среде CLIPS.
Открыть окно: Window/Dialog Window
Выполним:

– Execution/clear CLIPS
– Buffer/load Buffer
– Командами Execution/reset и Execution/run - запустим программу на выполнение.



6. Удалить участника из Омска, используя правило:

```
(defrule whithout-omsk
 ?omsk <- (rep ? Omsk
 =>
        (retract ?omsk))
```

В CLIPS удаление факта выполняется командой retract с указанием индекса удаляемого факта либо переменной, с которой факт связывается.