Министерство образования и науки Российской Федерации

Калужский филиал
федерального государственного бюджетного образовательного
учреждения высшего образования
**«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»**
(КФ МГТУ им. Н.Э. Баумана)

Ю.С. Белов, С.С. Гришунов

ОБЩИЕ ФУНКЦИИ И СИСТЕМА ВВОДА-ВЫВОДА В СРЕДЕ CLIPS

Методические указания по выполнению лабораторной работы
по курсу «Экспертные системы»

Калуга - 2017

УДК 004.891
ББК 32.813
Б435

Б435 Белов Ю.С., Гришунов С.С. Общие функции и система ввода-вывода в среде CLIPS. Методические указания по выполнению лабораторной работы по курсу «Экспертные системы». — М.: Издательство МГТУ им. Н.Э. Баумана, 2017. — 53 с.

Методические указания к выполнению лабораторной работы по курсу «Экспертные системы» содержат описание синтаксиса общих встроенных функций и функций ввода-вывода языка CLIPS.

Предназначены для студентов 4-го курса бакалавриата КФ МГТУ им. Н.Э. Баумана, обучающихся по направлению подготовки 09.03.04 «Программная инженерия».

УДК 004.891
ББК 32.813

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Настоящие методические указания составлены в соответствии с программой проведения лабораторных работ по курсу «Экспертные системы» на кафедре «Программное обеспечение ЭВМ, информационные технологии и прикладная математика» факультета фундаментальных наук Калужского филиала МГТУ им. Н.Э. Баумана.

Указания, ориентированные на студентов 4-го курса направления подготовки 09.03.04 «Программная инженерия», содержат краткое описание для работы со встроенными функциями и системой ввода-вывода в среде *CLIPS,* учебные примеры с комментариями и пояснениями, а также задание на лабораторную работу.

Методические указания составлены для ознакомления студентов со средой NASA CLIPS v.6.2 и овладения навыками работы с функциями. Для выполнения лабораторной работы студенту необходимы минимальные знания по программированию на высокоуровневом языке программирования.

ЦЕЛЬ И ЗАДАЧИ РАБОТЫ, ТРЕБОВАНИЯ К РЕЗУЛЬТАТАМ ЕЕ ВЫПОЛНЕНИЯ

Целью выполнения лабораторной работы является формирование практических навыков работы с функциями в среде CLIPS.

Основными задачами выполнения лабораторной работы являются:

1. изучить основные математические функции,
2. получить навыки работы с функциями системы ввода-вывода
3. научиться работать со списками и строками
4. получить навыки работы по созданию собственных функций.

Результатами работы являются:

- Созданные в среде CLIPS функции
- Сохраненные в файлах скрипты, тестовые входные данные и полученные выходные данные
- Подготовленный отчет

КРАТКАЯ ХАРАКТЕРИСТИКА ОБЪЕКТА ИЗУЧЕНИЯ, ИССЛЕДОВАНИЯ

CLIPS поддерживает не только эвристическую парадигму представления знаний (в виде правил), но и процедурную парадигму, используемую в большинстве языков программирования, таких, например, как *Pascal* или *C*. Функции в *CLIPS* являются последовательностью действий с заданным именем, возвращающей некоторое значение или выполняющей различные полезные действия (например, вывод информации). В *CLIPS* существуют внутренние и внешние функции. Внутренние функции реализованы средой *CLIPS,* поэтому их можно использовать в любой момент. Внешние функции — это функции, написанные пользователем. Внешние функции можно создавать как с помощью среды *CLIPS,* так и на любых других языках программирования, а затем подключать готовые, откомпилированные исполняемые модули к *CLIPS*. Для создания новых функций в *CLIPS* используется конструктор `deffunction`, описанный далее.

Создание функций

Конструктор `deffunction` позволяет пользователю создавать новые функции собственно в среде *CLIPS*. Способ вызова функций, определенных пользователем, эквивалентен способу вызова внутренних функций *CLIPS*.

Вызов функции осуществляется по имени, заданному пользователем. За именем функции следует список необходимых аргументов, отделенный одним или большим числом пробелов. Вызов функции вместе со списком аргументов должен заключаться в скобки. Последовательность действий определенной с помощью конструктора `deffunction` функции исполняется интерпретатором *CLIPS* (в отличие от функций, созданных на других языка программирования, которые должны иметь уже готовый исполняемый код).

Синтаксис конструктора `deffunction` включает в себя 5 элементов:

- имя функции;

- необязательные комментарии;
- список из нуля или более параметров;
- необязательный символ групповых параметров для указания того, что функция может иметь переменное число аргументов;
- последовательность действий или выражений, которые будут выполнены (вычислены) по порядку в момент вызова функции.

Синтаксис команды `deffunction`:

```
(deffunction <имя-функции> [<комментарии>]
   <обязательные-параметры> [<групповой-
         параметр>]
         <действия>
)
```

Функция, создаваемая с помощью конструктора `deffunction`, должна иметь уникальное имя, не совпадающее с именами других внешних и внутренних функций. Функция, созданная с помощью `deffunction`, не может быть перегружена. Конструктор `deffunction` должен быть объявлен до первого использования создаваемой им функции. Исключения составляют только рекурсивные функции. Приведем пример создания функции, вычисляющей длину гипотенузы прямоугольника, а также правила, содержащего в себе вызов данной функции.

Исходный код:

```
(deffunction hypotenuse (?a ?b)
  (sqrt (+ (* ?a ?a) (* ?b ?b)))
)
(defrule calculate-hypotenuse
  (dimensions ?base ?height)
   =>
  (printout t "Hypotenuse  = " (hypotenuse ?base
    ?height) crlf)
 )
```

Результат работы данной программы показан на рис. 1.

В зависимости от того, задан ли групповой параметр, функция, созданная конструктором, может принимать точное число параметров

или число параметров не меньшее, чем некоторое заданное. Обязательные параметры определяют минимальное число аргументов, которые должны быть переданы функции при её вызове. В действиях функции можно ссылаться на какие-то из этих параметров как на обычные переменные, содержащие простые значения. Если был задан групповой параметр, то функция может принимать любое количество аргументов, большее или равное минимальному. Если групповой параметр не задан, то функция может принимать число аргументов точно равное числу обязательных параметров. Все аргументы функции, которые не соответствуют обязательным параметрам, группируются в одно значение составного поля. Ссылаться на это значение можно, используя символ группового параметра. Для работы с групповым параметром могут использоваться стандартные функции *CLIPS,* предназначенные для работы с составными полями, такие как `length и nth`. Определение функции может содержать только один групповой параметр.

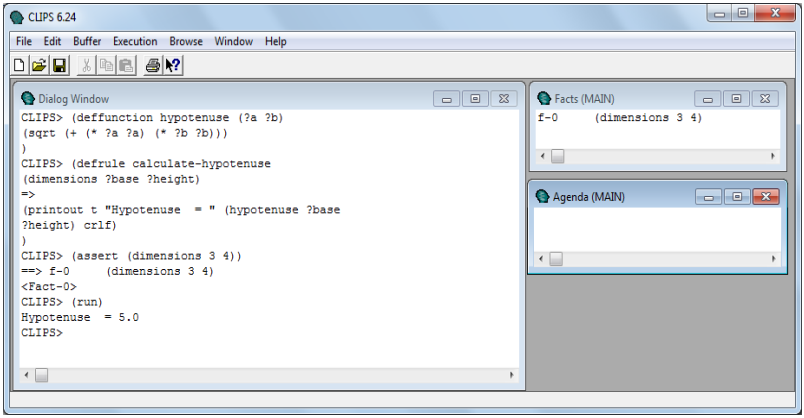

Рис.1 - Пример создания и работы функции

При вызове функции интерпретатор *CLIPS* последовательно выполняет действия в порядке, заданном конструктором. Функция возвращает значение, равное значению, которое вернуло последнее действие или вычисленное выражение. Если последнее действие не вернуло никакого результата, то выполняемая функция также не

вернет результата (как в приведенном выше примере). Если функция не выполняет никаких действий, то возвращенное значение равно FALSE. В случае возникновения ошибки при выполнении очередного действия выполнение функции будет прервано и возвращенным значением также будет FALSE.

Функции могут быть само- и взаимно рекурсивными. Саморекурсивная функция просто вызывает сама себя из списка своих собственных действий, например функция, вычисляющая факториал (В примере используется конструкция *if-then-else*).

Исходный код:

```
(deffunction factorial  (?a)
  (if
    (or
      (not (integerp ?a))
      (< ?a 0)
    )
  then
    (printout t "Factorial Error!" crlf)
  else
    (if (« ?a 0)
    then 1
    else (* ?a (factorial (- ?a 1)))
    )
  )
)
```

Взаимная рекурсия между двумя функциями требует предварительного объявления одной из этих функций. Для предварительного объявления функции в *CLIPS* используется конструктор `deffunction` с пустым списком действий. Для демонстрации взаимной рекурсии приведем следующий пример.

Исходный код:

```
(deffunction a())
(deffunction b()
  (a)
)
```

```
(deffunction a()
   (printout t "nothing")
)
```

CLIPS предоставляет специальный инструмент для работы с функциями — *«Deffunction Manager»* (Менеджер функций). Для запуска этого инструмента воспользуйтесь пунктом «*Deffunction Manager»* из меню *«Browse».* В случае, если в *CLIPS* не определена ни одна функция, данный пункт меню недоступен. Менеджер функций, отображающий функции, созданные нами, изображен на рис. 2.

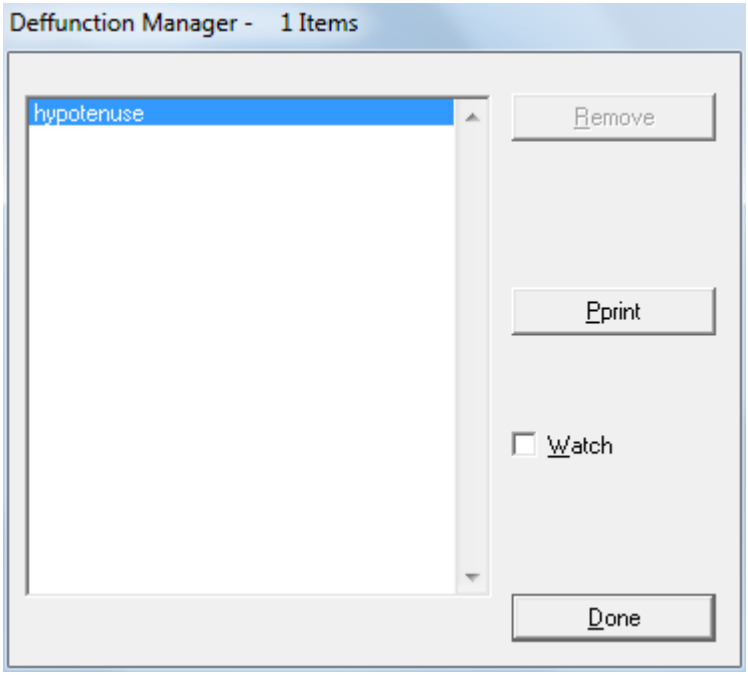

Рис.2 - Менеджер функций

Общее количество внешних функций отображается в заголовке окна менеджера — *«Deffunction Manager»* — 1 `Items`. С его помощью можно распечатать определение функции — кнопка *«Pprint»,* удалить её — кнопка *«Remove»,* а также установить режим просмотра вызова для отдельно выбранной функции — флаг *«Watch».*

Работа с функциями

Для удаления функции из памяти используют команду `undeffunction`, имеющую следующий синтаксис:

`(undeffunction` *<идентификатор_функции>)*

Если вместо параметра *<идентификатор_функции>* использовать символ «*», то из памяти удалятся все функции, находившиеся там к моменту удаления (конечно, если в рабочей памяти нет функции с именем «*»). Данная команда не возвращает никакого значения, и ей могут быть удалены любые функции, находящиеся в данный момент в рабочей памяти.

Для просмотра списка функций, находящихся в памяти, используют команду `list-deffunctions`, которая имеет следующий синтаксис:

`(list-deffunctions` *[<идентификатор_модуля>])*

Данная команда не возвращает никакого значения. Она выводит на экран имена всех функций, находящихся в памяти.

Для просмотра (тела) функции, находящейся в памяти, используют команду `ppdeffunction`, которая имеет следующий синтаксис:

`(ppdeffunction` *<идентификатор_функции>)*

Данная команда не возвращает никакого значения, а лишь выводит текст функции на экран.

Методы

Для определения функций с большими возможностями используют `defgeneric` и `defmethod`. Функции, определенные таким образом, называют *методами*. `Defgeneric и defmethod` имеют ряд преимуществ по сравнению с `deffunction`:

- возможность перегрузки функций (в том числе стандартных);
- явное задание типов аргументов и возможность наложения на них дополнительных ограничений.

Но помимо преимуществ имеют место и следующие недостатки:

- снижение производительности при перегрузке функций на 15-20%;
- сложный синтаксис.

При определении прототипа используется следующий синтаксис:

```
(defgeneric <идентификатор_функции> ["комментарии"])
```

Для определения метода используется следующий синтаксис:

```
(defmethod <идентификатор_функции> [<индекс>]
["комментарии"]
    (<arg1>  ...  <argN>  [<list_arg>])   ;
    Аргументы
    [<action1>  ...  <actionK>        ; Действия
    <actionk+1>]           ; Функция возвращает
    результат
                    ; последнего действия
)
```

Название прототипа функции должно соответствовать названию методов. Прототип функции необходимо вводить только при опережающем описании и перегрузке. В других случаях в этом нет необходимости (неявно прототип будет введен в систему автоматически). При прямом рекурсивном вызове в явном введении прототипа также нет необходимости.

Индекс нового метода будет автоматически инкрементироваться при перегрузке (по умолчанию равен 1).

Приведем пример работы с методами.

Исходный код:

```
(defmethod >  ((?a STRING) (?b STRING))
   (> (str-compare ?a ?b) 0)
)
```

Если два и более перегруженных метода претендуют на обработку функционального вызова, предпочтение отдается методу, предоставляющему большие ограничения на аргументы в порядке их перечисления. Продемонстрируем данную ситуацию на примере.

Исходный код:

```
; Системный оператор '+' является начальным
методом
;Его системное описание таково:
```

```
;#1
(defmethod + ((?a NUMBER)(?b NUMBER)
  ($?rest NUMBER)))
;#2
(defmethod + ((?a NUMBER) (?b INTEGER)))
;#3
(defmethod + ((?a INTEGER) (?b INTEGER)))
;#4
(defmethod + ((?a INTEGER) (?b NUMBER)))
;#5
(defmethod + ((?a NUMBER) (?b NUMBER)
  ($?rest NUMBER SYMBOL)))
;#6
(defmethod + ((?a NUMBER) (?b INTEGER (> ?b
2))))
;#7
(defmethod + ((?a INTEGER (> ?a 2))
  (?b INTEGER (> ?b 3))))
;#8
(defmethod + ((?a INTEGER (> ?a 2)) (?b
NUMBER)))
```

Приоритет методов при вызове функции «+» будет следующим (в порядке убывания): #7; #8; #3; #4; #6; #2; #1; #5.

Для вызова другого перегруженного метода можно пользоваться командой `call-next-method`, которая имеет следующий синтаксис:

```
(call-next-method)
```

Для удаления прототипа из памяти используют команду `undefgeneric`, которая имеет следующий синтаксис:

```
(undefgeneric
<идентификатор_функции>)
(undefgeneric *)
```

Для просмотра списка прототипов, находящихся в памяти, используют команду `list-defgenerics`, которая имеет следующий синтаксис:

(list-defgenerics *[<идентификатор_модуля>])*

Для просмотра прототипа, находящегося в памяти, используют команду ppdefgeneric, которая имеет следующий синтаксис:

(ppdefgeneric *<идентификатор_функции>)*

Для просмотра списка методов, подходящих для выполнения данного функционального вызова, можно использовать функцию preview-generic, которая имеет следующий синтаксис:

(preview-generic *<функциональное_выражение>)*

Приведем пример работы с данной командой:

```
(preview-generic  (> "duck1" "duck2") )
```

Для удаления метода из памяти можно использовать команду undefmethod:

(undefmethod *<идентификатор_функции> <индекс_метода>)*

(undefmethod *<идентификатор_функции> *)*

(undefmethod * *)

Для просмотра списка методов, находящихся в памяти, используют команду list-defmethods:

(list-defmethods *<идентификатор_функции>)*

Для просмотра (тела) функции, находящейся в памяти, используют команду ppdefmethod:

(ppdefmethod *<идентификатор_функции> <индекс_метода>)*

В остальном команда defmethod сходна с deffunction.

СИСТЕМА ВВОДА-ВЫВОДА ИНФОРМАЦИИ

Система ввода-вывода *CLIPS* базируется на концепции логических имен (сходной с потоками), присваиваемых устройствам (физическим и логическим). Логические имена могут быть символьного, строкового или числового типа. Ряд имен зарезервирован (табл. 1).

Таблица 1

Список зарезервированных имен

Имя	Описание
`stdin`	Имя, используемое по умолчанию в операциях ввода *(read, readln)*. Аналогом является указание символа «t»
`stdout`	Имя, используемое в операциях вывода. Аналогом является указание символа «t»
`wclips`	Имя, используемое командной строкой *CLIPS*
`wdialog`	Имя для работы с сообщениями
`wdisplay`	Имя, используемое при отображении информации *CLIPS* (факты, правила)
`werror`	Имя, используемое при обработке ошибок
`wwarning`	Имя, используемое при обработке предупреждений
`wtrace`	Имя, используемое при просмотре отладочной информации (watch и др.)

Функция OPEN

Эта функция позволяет открывать файл (связывать файловую переменную с файлом):

`(open` *<имя_файла> <логическое_имя> [<флаг>])*

В табл. 2 приведен список флагов и их значений.

Таблица 2

Список значений флагов

Флаг	Значение (тип доступа к файлу)
"r"	Только для чтения (значение по умолчанию)
"w"	Только для записи
"r+"	Для чтения и записи
"a"	Только для добавления
"wb"	Бинарный доступ на запись

<имя_файла> должно быть строковым или символьным литералом, содержащим относительный или абсолютный путь к файлу.

Функция возвращает TRUE в случае успешного открытия файла, иначе FALSE.

Функция open, используемая в правилах, может находиться только в правой части. Приведем пример работы с функцией open.

Исходный код:

```
CLIPS> (open "myfile.clp" writeFile "w")
TRUE
CLIPS>
```

Функция CLOSE

Функция close закрывает файл (поток), открытый при помощи open, и имеет следующий синтаксис:

(close *[<логическое_имя>])*

В случае вызова функции без параметров будут закрыты все файлы. Функция возвращает TRUE в случае успешного закрытия файла, иначе FALSE. Приведем пример использования данной функции.

Исходный код:

```
CLIPS>(open "myfile.clp" writeFile "w")
TRUE
CLIPS>(open "MS-DOS\\directory\\file.dp" readFile)
TRUE
CLIPS>(close writeFile)
TRUE
CLIPS>(close writeFile)
FALSE
CLIPS>(close)
TRUE
CLIPS>(close)
FALSE
CLIPS>
```

Функция PRINTOUT

Функция, позволяющая выводить информацию на устройство, ассоциированное с логическим именем (например, может производиться вывод информации на экран или запись в файл). Если в качестве логического имени выбрано `nil`, информация никуда не будет выведена.

`(printout` <*логическое_имя*> <*строковое_выражение*>*)*

В `<строковом_выражении>` могут быть использованы, помимо строк, символов, чисел и адресов (а также переменных, содержащих их), специальные символы, приведенные в табл. 3.

Таблица 3

Специальные символьные константы

Символьная константа	Действие
crlf	Перенос каретки на начало новой строки
tab	Табуляция горизонтальная
vtab	Табуляция вертикальная

Приведем пример использования функции printout.

Исходный код:

```
CLIPS>(printout t "Hello there!" crlf)
Hello There!
CLIPS>(open "data.txt" raydata "w")
TRUE
CLIPS>(printout mydata "red green")
CLIPS>(close)
TRUE
CLIPS>
```

Функция READ

Функция, позволяющая считывать информацию по одному полю (признаком, разделяющим поля, является пробел, табуляция или перенос строки; в строковых полях должны быть двойные кавычки), имеет следующий синтаксис:

(read *[<логическое_имя>]*)

Считывание информации из файла происходит последовательно до получения символа EOF (включительно). В случае ошибки чтения будет выведено строковое сообщение

"*** READ ERROR ***".

Приведем пример работы с функцией read.

Исходный код:

```
CLIPS>(clear)
CLIPS>(open "data.txt" mydata "w")
TRUE
CLIPS>(printout mydata "red green")
CLIPS>(close)
TRUE
CLIPS>(open "data.txt" mydata)
TRUE
CLIPS>(read mydata)
red
```

```
CLIPS>(read mydata)
green
CLIPS>(read mydata)
EOF
CLIPS>(close)
TRUE
CLIPS>
```

Функция READLINE

Функция, позволяющая считывать информацию построчно (признаком окончания строки является перенос строки или символ EOF).

(readline *[<логическое_имя>])*

Считывание информации из файла происходит последовательно до получения символа EOF (включительно). В случае ошибки чтения будет выведено строковое сообщение
"*** READ ERROR ***",в противном случае — считанная строка.

Приведем пример работы с функцией readline.

Исходный код:

```
CLIPS>(open "data.txt" mydata "w")
TRUE
CLIPS>(printout mydata "red green")
CLIPS>(close)
TRUE
CLIPS>(open "data.txt" mydata)
TRUE
CLIPS>(readline mydata)
"red green"
CLIPS>(readline mydata)
EOF
CLIPS>(close)
TRUE
CLIPS>
```

Функция FORMAT

Функция, позволяющая выводить форматированную информацию на устройство, ассоциированное с логическим именем. Функция всегда возвращает строку форматированного текста. Если в качестве логического имени выбрано `nil`, форматированный текст не будет записан ни на какое устройство.

Функция является аналогом `printf` языка *C* и дополняет рассмотренную ранее простую функцию `printout`.

(`format` <*логическое_имя*> <*строковое выражение*> <*флаги*>)

Флаги (табл. 4) определяют стиль вывода каждого параметра в <*строковом_выражении*>. Флаги имеют следующий общий формат: `|%-M.Nx|`

`M.N` — опциональный параметр, определяющий длину целой и дробной частей (соответственно) числа, а знак «-» — тип заполнения.

Таблица 4

Формат вывода параметра x

x	Значение (формат вывода параметра)
c	Вывод одного символа
d	Вывод длинного целого числа
f	Вывод вещественного числа
e	Вывод вещественного числа в экспоненциальной форме
g	Вывод в кратчайшей форме, доступной для данного типа
o	Вывод беззнакового восьмеричного числа
x	Вывод беззнакового шестнадцатеричного числа
s	Вывод строки
n	Переход на новую строку
r	Перевод каретки на начало строки
%	Выводит символ «%»

В настоящий момент функция format не поддерживает списковые структуры (переменные).

Приведем несколько примеров работы с функцией format.

Исходный код:

```
CLIPS>(clear)
CLIPS>(format t "Hello World!%n")
"Hello World!"
CLIPS>(format nil "Integer: |%ld|" 12)
"Integer: |12|"
CLIPS>(format nil "Integer: |%4id|" 12)
"Integer: |12|"
CLIPS>(format nil "Integer: |%-04ld|" 12)
"Integer: |12|"
CLIPS>(format nil "Float: |%f|" 12.01)
"Float: |112.010000|"
CLIPS>(format nil "Float: |%7.2f| "12.01)
"Float: |12.01|"
CLIPS> format nil "Test: |%e|" 12.01)
"Test: |1.201000e+01|"
CLIPS>(format nil "Test: |%7.2e|" 12.01)
"Test: |1.20e+01|"
CLIPS>(format nil "General: |%g|" 1234567890)
"General: |1.23457e+09|"
CLIPS>(format nil "Hexadecimal: |%x|" 12)
"Hexadecimal: |c|"
CLIPS>(format nil "Octal: |%o|" 12)
"Octal: |14|"
CLIPS>(format nil "Symbols: |%s| |%s|" value-a1
capacity)
"Symbols: |value-a1| |capacity|"
CLIPS>
```

Функция RENAME

Функция, переименовывающая файл:

(rename <старое_имя_файла> <новое_имя_файла>)

Возвращает TRUE в случае удачного переименования и FALSE — в противном случае. Приведем пример работы с функцией rename.

Исходный код:

```
CLIPS>(rename "1.txt" "new_1.txt")
TRUE
CLIPS>
```

Функция REMOVE

Функция, удаляющая файл:

(remove *<имя_файла>)*

Возвращает TRUE в случае удачного удаления и FALSE — в противном случае. Приведем пример работы с функцией remove.

Исходный код:

```
CLIPS>(remove "new_1.txt")
TRUE
CLIPS>
```

ЗАДАЧИ И ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

Общие функции. Операции над списками и строками

Функция, создающая список, — `create`:

`(create$ <элементы_списка>)`

Элементы списка разделяются пробелом и могут быть произвольного типа. Приведем пример работы с функцией `create`.

Исходный код:

```
CLIPS>(create$ hammer drill saw screw pliers
wrench)
(hammer drill saw screw pliers wrench)
CLIPS>(create$ (+ 3 4) (* 2 3) (/ 8 4))
(7 6 2)
CLIPS>
```

Функция, возвращающая элемент списка по номеру, — `nth`:

`(nth$ <номер_элемента> <список>)`

<*Номер элемента*> — целое число, большее или равное 1. Если <*Номер_элемента*> превышает длину списка, функция возвращает `nil`. Приведем пример.

Исходный код:

```
CLIPS>(nth$ 3 (create$ a b c d e f
g))
c
CLIPS>
```

Функция, проверяющая вхождение элемента в список, —`member`:

`(member$ <элемент> <список>)`

В случае вхождения элемента функция возвратит его номер в списке. Если элемент является списком и является подмножеством искомого списка, функция возвратит номера первого и последнего элементов исходного списка, соответствующие заданному списку. Иначе возвратит `FALSE`. Приведем пример, демонстрирующий работу с данной функцией.

Исходный код:

```
CLIPS>(member$ blue (create$ red 3 "text" 8.7
blue))
5
CLIPS>(member$ 4 (create$ red 3 "text" 8.7
blue))
FALSE
CLIPS>(member$ (create$ b c) (create$ a b e d ) )
(2 3)
CLIPS>
```

Функция (предикат), проверяющая вхождение одного списка в другой, — `subsetp`:

`(subsetp` *<список1> <список2>)*

В случае вхождения всех элементов *< списка 1>* в *<список2>* функция возвратит `TRUE`, иначе — `FALSE`. Приведем пример, демонстрирующий работу с данной функцией.

Исходный код:

```
CLIPS>(subsetp (create$ hamraer saw drill)
(create$ hammer drill wrench pliers saw))
TRUE
CLIPS>(subsetp (create$ wrench crowbar) (create$
hammer drill wrench pliers saw) )
FALSE
CLIPS>
```

Функция, удаляющая элементы из списка (заключенные между индексированными начальным и конечным элементами), — `delete`:

`(delete$` *<список> <начальный_индекс> <конечный индекс)*

Данная функция возвращает измененный список. Для того чтобы удалить один элемент из списка, необходимо чтобы начальный и конечный индексы совпадали. Приведем пример, демонстрирующий работу с данной функцией.

Исходный код:

```
CLIPS>(delete$ (create$ hammer drill saw pliers
wrench) 3 4)
(hammer drill wrench)
```

```
CLIPS>(delete$ (create$ computer printer hard-
disk) 1 1)
(printer hard-disk)
CLIPS>
```

Функция, создающая список из строки, — explode:

(explode$ <*строковое_выражение*>)

Если <*строковое_выражение*> не содержит ни одного значения, то будет создан список нулевой длины. Числовые значения в строковом выражении будут преобразованы в строку. Приведем пример, демонстрирующий работу с данной функцией.

Исходный код:

```
CLIPS>(explode$ "hammer drill saw
screw")
(hammer drill saw screw)
CLIPS>(explode$ "1 2 abc 3 4 \"abc\" V'defV")
(1 2 abc 3 4 "abc"
"def")
CLIPS>(explode$ "?x
~ )")
("?x" "~" ")")
CLIPS>
```

Функция, обратная функции explode, создающая строку из списка:

(implode$ <*список*>)

Приведем пример, демонстрирующий работу с данной функцией.

Исходный код:

```
CLIPS>(implode$ (create$ hammer drill
screwdriver))
"hammer drill screwdriver wrench pliers saw"
CLIPS>(implode$ (create$ 1 "abc" def "ghi" 2))
"1 "abc" def "ghi" 2"
CLIPS>(implode$ (create$ "abc def ghi"))
""abc def ghi""
CLIPS>
```

Функция, возвращающая подсписок списка (заключенный между индексированными начальным и конечным элементами), — subseq:

(subseq$ <*список*> <*начальный_индекс*> <*конечный_индекс*>)

Приведем пример, демонстрирующий работу с данной функцией.

Исходный код:

```
CLIPS>(subseq$ (create$ hammer drill wrench
pliers) 3 4) (wrench pliers)
CLIPS>(subseq$ (create$ 1 "abc" def "ghi" 2) 1
1)
(1)
CLIPS>
```

Функция, заменяющая подсписок списка (заключенный между индексированными начальным и конечным элементами) на список (или некоторое поле), — `replace`:

`(replace$` *<список> <начальный_индекс> <конечный_индекс> <список_или_поле>)*

Приведем пример работы с функцией `replace`.

Исходный код:

```
CLIPS>(clear)
CLIPS>(replace$ (create$ drill wrench pliers) 3
3 machete)
(drill wrench machete)
CLIPS>(replace$ (create$ a b e d) 2 3 x y
(create$ q r s))
(a x y q r s d)
CLIPS>
```

Функция, вставляющая поле или список в заданное место списка (определенное индексом) и возвращающая результирующий список, – `insert`:

`(insert$` *<список> <индекс> <список_или_поле>)*

Приведем пример, демонстрирующий работу с данной функцией.

Исходный код:

```
CLIPS>(insert$ (create$ a b e d) 1 x)
(x a b c d)
CLIPS>(insert$ (create$ a b e d) 4 y z)
(a Ь c y z d)
CLIPS>(insert$ (create$ a b e d) 5 (create$ q
r))
(a b c d q r)
```

```
CLIPS>
```

Функция, возвращающая первый элемент (голову) списка, — `first:`

`(first$ <список>)`

Приведем пример, демонстрирующий работу с данной функцией.

Исходный код:

```
CLIPS>(first$ (create$ a b c))
(a)
```

Функция, возвращающая все элементы списка, кроме первого (хвост), — `rest:`

`(rest$ <список>)`

Приведем пример, демонстрирующий работу с данной функцией.

Исходный код:

```
CLIPS>(rest? (create$ a b c))
(b c)
```

Функция, возвращающая длину (количество элементов, полей) списка, — `length:`

`(length$ <список>)`

Приведем пример, демонстрирующий работу с данной функцией.

Исходный код:

```
CLIPS>(lengths (createS a b c d e f
g))
7
CLIPS>
```

Функция, возвращающая длину строки или количество подполей в мультиполе:

`(length <строка> | <мультиполе>)`

Пример функции `length`.

Исходный код:

```
CLIPS>(length "abed")
4
CLIPS>(clear)
CLIPS>(deffunction count ($?arg)
   (length $?arg)
```

)

```
CLIPS>(count 1 2  3  four "five")
5
CLIPS>
```

Функция, удаляющая все элементы одного списка (второго) из другого (первого), — `delete-member`:

`(delete-member$ <`*список1*`> <`*список2*`>)`

Возвращает результирующий список. Приведем пример, демонстрирующий работу с данной функцией.

Исходный код:

```
CLIPS>(delete-member$ (create$ a b a c )   b a)
(c)
```

Функция, заменяющая выбранные элементы (третий список) одного списка (первого) элементами другого (второго), — `repiace-member`:

`(replace-member$ <список-1> <список-2> <список-3> [<список-4> ...])`

Приведем пример, демонстрирующий работу с данной функцией.

Исходный код:

```
CLIPS>(replace-member$ (create$  a b a b)
(create$ a b a) a b)
( a b a a b a a b a a b a )
```

Функция, осуществляющая конкатенацию строк, — `str-cat`:

`(str-cat <`*строковое_выражение*`>)`

Пример работы функции `str-cat`.

Исходный код:

```
CLIPS>
(clear)
CLIPS> (str-cat "super" "star")
«superstar»
CLIPS>
```

Функция, осуществляющая конкатенацию символов, — `sym-cat`:

`(sym-cat <`*строковое_выражение*`>)`

Пример работы функции `sym-cat`.

Исходный код:

```
CLIPS>(clear)
CLIPS>(sym-cat super star)
superstar
CLIPS>
```

Функция, возвращающая часть строки (заключенной между начальным и конечными индексами), — `sub-string`:

`(sub-string` *<начальный_индекс>* *<конечный_индекс>* *<строка>)*

Приведем пример, демонстрирующий работу с данной функцией.

Исходный код:

```
CLIPS>(sub-string 3 8 "abcdefghijkl")
"cdefgh"
CLIPS>
```

Функция, возвращающая индекс начала первой строки во второй, — `str-index`:

`(str-index` *<строка1>* *<строка2>)*

Если вторая строка не содержит первую, функция возвращает `FALSE`. Приведем пример, демонстрирующий работу с данной функцией.

Исходный код:

```
CLIPS>(str-index "def" "abcdefghi")
4
CLIPS>
```

Функция, осуществляющая сравнение строк, — `str-compare`:

`(str-compare` *<строка1>* *<строка2>)*

Функция возвращает 1, -1 или 0 в случае, если первая строка соответственно больше, меньше или равна второй строке. Приведем пример, демонстрирующий работу с данной функцией.

Исходный код:

```
CLIPS>(str-compare "def" "abcdefghi")
-1
```

```
CLIPS>
```

Функция, возвращающая длину строки или символьного литерала, — `str-length:`

`(str-length` *<строка_или_символьный_литерал>)*

Приведем пример, демонстрирующий работу с данной функцией.

Исходный код:

```
CLIPS>(str- length
"def")
3
CLIPS>
```

Функция, преобразующая первое поле строки или символьного литерала в соответствующий тип данных, — `string-to-field:`

`(string-to-field` *<строка_или_символьный_литерал>)*

Пример работы функции `string-to-field.`

Исходный код:

```
CLIPS>(clear)
CLIPS>(string-to-field "3.4")
3.4
CLIPS>(string-to-field "a b")
a
CLIPS>
```

Функция, преобразующая строку в последовательность команд и выполняющая эту последовательность, — `eval:`

`(eval` *<строка>)*

Возможности функции `eval` ограничены: так, запрещено использовать в выражениях переменные и объявлять структуры данных (функции, объединения, правила, шаблоны и др.). Приведем пример.

Исходный код:

```
CLIPS>(eval "(+ 3 4)")
7
CLIPS>
```

Аналог `eval,` позволяющий использовать переменные и создавать структуры, — функция `build:`

`(build <`*строка*`>)`

Пример работы функции `build`.

Исходный код:

```
CLIPS>(clear)
CLIPS>(build "(defrule foo (a)  => (assert
(b)))")
TRUE
CLIPS>(rules)
foo
For a total of 1 rule.
CLIPS>
```

Функции, позволяющие преобразовывать регистр строки (в верхний и нижний соответственно), — `upcase` и `lowcase`:

`(upcase` *<строка>)*

`(lowcase` *<строка>)*

Приведем пример работы с данными функциями.

Исходный код:

```
CLIPS>(upcase hello)
HELLO
CLIPS>(lowcase HELLO)
hello
CLIPS>
```

МАТЕМАТИЧЕСКИЕ ФУНКЦИИ

Сумма (аргументы — числа):

```
(+ арг1 арг2  [арг3 ...])
```

Разность (аргументы — числа):

```
(-арг1 арг2  [арг3 ...])
```

Умножение (аргументы — числа):

```
(* арг1 арг2  [арг3 ...])
```

Деление (аргументы — числа):

```
(/арг1 арг2  [арг3 ...])
```

Приведем пример работы с данными математическими функциями.

Исходный код:

```
CLIPS>( + 2  3 4)
9
CLIPS>(+ 2  3.0 5)
10.0
CLIPS>(+ 3.1  4.7)
7.8
CLIPS>(/ ( + 2 3 4 5 )  (- (* 8 9) 12))
0.233333333333
CLIPS>
```

Целочисленное деление (аргументы — целые числа):

```
(div арг1 арг2  [арг3 ...])
```

Продемонстрируем работу данной функции на предыдущем примере.

Исходный код:

```
CLIPS>(div (+ 2 3 4 5) (- (* 8 9) 12))
0
CLIPS>
```

Максимум (аргументы — числа):

```
(max aprj арг2 [арг3 ...])
```

Минимум (аргументы — числа):

```
(min арг1 арг2  [арг3 ...])
```

Приведем пример работы с данными математическими функциями.

Исходный код:

```
CLIPS>(max 2 3 4)
4
CLIPS>(min 2 5)
2
CLIPS>(max (min 2 3) (min 3 4))
3
CLIPS>
```

Абсолютное значение, т.е. модуль (аргументы — числа):

```
(abs арг1 арг2 [арг3 ...])
```

Продемонстрируем работу данной функции на следующем примере.

Исходный код:

```
CLIPS>(abs 3)
3
CLIPS>(abs -3)
3
CLIPS>
```

Перевод числа в вещественный формат (аргумент — число):

```
(float арг)
```

Перевод числа в целочисленный формат (аргумент — число):

```
(integer арг)
```

Тригонометрические функции (аргументы — числа; исчисление в радианах) приведены в табл. 5.

Преобразование градусов в различные форматы (аргумент — число):

```
(deg-grad арг)
(deg-rad арг)
(grad-deg арг)
(rad-deg арг)
```

Число π (пи):

```
(pi)
```

Приведем пример работы с тригонометрическими функциями и числом π.

Исходный код:

```
CLIPS>(deffunction square (?r)
   (* 2 (pi) ?r)
)
CLIPS>(square 2)
12.5663706143592
CLIPS>(tan (/ (pi) 4))
1.0
CLIPS>(cos (/ (pi) 3))
0.5
CLIPS>(deg-grad (pi))
3.49065850398866
CLIPS>(deg-rad (pi))
0.0548311355616075
CLIPS>(grad-deg (pi))
2.82743338823081
CLIPS>(rad-deg (pi))
180.0
CLIPS>
```

Таблица 5

Тригонометрические функции CLIPS

Обозначение *(CLIPS)*	Определение
acos	арккосинус
acosh	гиперболический арккосинус
acot	арккотангенс
acoth	гиперболический арккотангенс
acsc	арккосеканс
acsch	гиперболический арккосеканс
asec	арксеканс
asech	гиперболический арксеканс
asin	арксинус
asinh	гиперболический арксинус
atan	арктангенс
atanh	гиперболический арктангенс

Таблица 5 - продолжение

cos	косинус
cosh	гиперболический косинус
cot	котангенс
coth	гиперболический котангенс
CSC	косеканс
csch	гиперболический косеканс
sec	секанс
sech	гиперболический секанс
sin	синус
sinh	гиперболический синус
tan	тангенс
tanh	гиперболический тангенс

Квадратный корень из числа:

```
(sqrt арг)
```

Возведение числа (первый аргумент) в степень (второй аргумент) (аргументы — числа):

```
(** арг1 арг2)
```

Приведем пример работы с данными математическими функциями.

Исходный код:

```
CLIPS>(sqrt 4)
2.0
CLIPS>(** 2 2)
4.0
CLIPS>
```

Возведение экспоненты (эйлерова константа) в степень (аргумент — число):

```
(exp арг)
```

Натуральный логарифм (аргумент — число):

```
(log арг)
```

Десятичный логарифм (аргумент — число):

```
(log10 арг)
```

Округление до целого (аргумент — число):

```
(round арг)
```

Остаток от деления одного числа (первый аргумент) на другое (второй аргумент) (аргументы — числа):

(mod арг$_1$ арг$_2$)

Приведем пример работы с данными математическими функциями.

Исходный код:

```
CLIPS>(exp 4)
54.5981500331442
CLIPS>(log 2)
0.693147180559945
CLIPS>(log10 4)
0.602059991327962
CLIPS>(round 2.204)
2
CLIPS>(mod 23 2)
1
CLIPS>
```

ДОПОЛНИТЕЛЬНЫЕ ФУНКЦИИ

Функция, создающая новую или модифицирующая существующую переменную, — `bind`:

`(bind` *<переменная>* *<выражение>)*

Если параметр *<выражение>* отсутствует, то переменная сбрасывает свое текущее значение, в противном случае переменной присваивается результат вычисления выражения. На месте данного параметра может стоять несколько выражений одновременно. В данном случае они вычисляются последовательно, и в переменную заносится результат последнего действия. Пример работы функции `bind`.

Исходный код:

```
(bind ?*x* (+ 8 9))
```

Условная конструкция `if then else` имеет следующий синтаксис:

```
(if <условие>
  then
  <действия> [
  else
  <действия>
  ]
)
```

Все действия в данной конструкции выполняются последовательно и конструкция возвращает результат последнего действия.

Продемонстрируем работу с конструкцией `if then else` на конкретном примере.

Исходный код:

```
(defrule closed-valves
  (temp high)
  (valve ?v closed)
=>
```

```
  (if (= ?v 6)
    then
      (printout t "The special valve" ?v "is
        closed!" crlf)
      (assert (perform special operation))
    else
      (printout t "Valve " ?v " is normally
      closed"
        crlf)
  )
)
```

Конструкция для организации циклов с условием `while`:

`(while` *<условие>* `[do]` *<действия>)*

Продемонстрируем работу с конструкцией для организации циклов с условием `while` на конкретном примере.

Исходный код:

```
(defrule open-valves
  (valves-open-through ?v)
=>
  (while (> ?v 0)
    (printout t "Valve " ?v " is open" crlf)
    (bind ?v (- ?v 1))
  )
)
```

Оператор, прерывающий выполнение циклов `break`:

`(break)`

Приведем пример использования оператора `break`.

Исходный код:

```
CLIPS>(clear)
CLIPS>(deffunction iterate (?num)
  (bind ?i 0)
  (while TRUE do
    (if (>= ?i ?num) then (break))
    (printout t ?i " ")
    (bind ?i (+ ?i 1))
  )
  (printout t crlf)
```

```
  )
  CLIPS> (iterate 10)
  0 1 2 3 4 5 6 7 8 9
  CLIPS>
```

Конструкция для организации циклов со счетчиком `loop-for-count`:

(loop-for-count *<количество_итераций>* |
(<переменная_цикла> <начальное_значение_переменной>
<максимальное_значение_переменной>) |
(<переменная_цикла> <максимальное_значение_переменной>)
[do]
<действия>)

Приведем пример работы с конструкцией `loop-for-count`.

Исходный код:

```
CLIPS>(clear)
CLIPS>(loop-for-count (?cntl 2 4) do
   (loop-for-count (?cnt2 1 3) do
     (printout t ?cntl " " ?cnt2 crlf)
   )
)
2 1
2 2
2 3
3 1
3 2
3 3
4 1
4 2
4 3
FALSE
CLIPS>
```

Конструкция множественного выбора `switch`:

```
(switch <тестовое_выражение>
  <выражение-1>
  <выражение-2>
```

```
   ...
   <выражение-n>
   [<выражение_по-умолчанию> ]
)
<выражение-m>  :: =  (case <значение-m> then
<действия>) <выражение_по-умолчанию> ::=
(default <действия>)
```

Пример работы со `switch`.

Исходный код:

```
CLIPS>(defglobal ?*x* = 0)
CLIPS>(defglobal ?*y* = 1)
CLIPS>(deffunction foo (?val)
   (switch ?val
     (case ?*x* then *x*)
     (case ?*y* then *y*)
     (default none)
   )
)
CLIPS>(foo 0)
*x*
CLIPS>(foo 1)
*y*
CLIPS>
```

Оператор `return` позволяет прервать выполнение правила (правой части), функции или метода и вернуть при этом (если необходимо) какое-либо значение:

```
(return [<выражение>])
```

Пример работы оператора `return`.

Исходный код:

```
CLIPS>(clear)
CLIPS>(deffunction sign (?num)
   (if (> ?num 0) then (return 1))
   (if (< ?num 0) then (return -1))
   0
)
```

```
CLIPS>(sign 5)
1
CLIPS> (sign 0)
0
CLIPS>
```

Оператор `progn`, вычисляющий последовательность операндов (в списке) и возвращающий результат последнего:

`(progn <`*список_операндов*`>)`

Пример работы оператора `progn`.

Исходный код:

```
CLIPS>(progn (+ 5 0.7) (- 1 89) (neq "a" "b"))
TRUE
CLIPS>
```

Оператор `gensym`, генерирующий уникальный идентификатор (при каждом вызове возвращается новый идентификатор):

`(gensym)`

Пример работы оператора `gensym`.

Исходный код:

```
CLIPS>(assert (new-id (gensym) flagl 7))
==> f-2 (new-id gen5 flagl 7)
<Fact-2>
CLIPS>
```

Оператор `setgen`, присваивающий номер, начиная с которого будут генерироваться уникальные идентификаторы посредством `gensym`:

`(setgen <число>)`

Приведем пример работы с функцией `setgen`.

Исходный код:

```
CLIPS>(setgen 32)
32
CLIPS>(gensym)
gen32
CLIPS>
```

Функция random, возвращающая число из заданного интервала (интервал указывается опционально, значения могут быть только целочисленными):

```
(random [<нач.-значение> <кон.-значение>])
```

Приведем пример работы с функцией random.

Исходный код:

```
CLIPS>(random 1 100)
48
CLIPS>
```

Функция seed, инициализирующая генератор случайных чисел:

```
(seed <число>)
```

Параметр <число> должен быть целого типа. Он определяет значение, с которого будут генерироваться случайные числа. Приведем пример работы с данной функцией.

Исходный код:

```
CLIPS>(seed 100)
CLIPS>(random)
348
CLIPS>
```

Функция time, возвращающая вещественное представление системного времени:

```
(time)
```

Приведем пример работы с функцией time. Предположим, что на часах 00:49.

Исходный код:

```
CLIPS>(time)
2250.25
CLIPS>
```

Функция timer, возвращающая количество секунд, затраченное интерпретатором на обработку введенного выражения:

```
(timer <выражение>)
```

Пример работы функции timer.

Исходный код:

```
CLIPS>(timer (loop-for-count 10000 (+ 3 4)))
0.0416709999999512
CLIPS>
```

Функция `sort`, возвращающая отсортированный по ключу список:

`(sort <ключ> <список>)`

Пример работы функции `sort`.

Исходный код:

```
CLIPS>(sort > 4 3 5 7 2 7)
(2 3 4 5 7 7)
CLIPS>
```

ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

Решить задачу, указанную в варианте, использую функциональный стиль программирования в среде CLIPS.

Разработать алгоритм для решения поставленной задачи в соответствии с вариантом.

Реализовать разработанный алгоритм в среде CLIPS, реализовать ввод данных из файла и вывод результата в файл.

Протестировать работу алгоритма на всех возможных вариантах наборов входных данных.

ТРЕБОВАНИЯ К РЕАЛИЗАЦИИ

Вариант задания назначается преподавателем.

Все функции, входные и выходные данные алгоритма должны быть сохранены в файлы.

ВАРИАНТЫ ЗАДАНИЙ

1. Написать функцию, получающую в качестве параметра имя файла, в котором в трёхмерных координатах заданы прямая (двумя точками) и некоторый замкнутый пространственный контур (множеством точек, соединённых отрезками, перечисленных в определённом порядке обхода). Определить положение прямой относительно контура (проходит внутри контура, пересекает контур или лежит вне контура). Результат вывести на экран.
2. Написать функцию, получающую в качестве параметра имя файла, содержащего координаты выпуклого n-угольника в порядке обхода(n>3). Разбить его на треугольники диагоналями, так чтобы сумма длин диагоналей была минимальной. Координаты получившихся треугольников записать в файл.

3. Написать функцию, получающую в качестве параметра имя файла, содержащего число n, координаты вершин выпуклого n-угольника в порядке обхода (в двумерных декартовых координатах), число m, координаты выпуклого m-угольника. Написать функцию, находящую k-угольник, являющийся пересечением n- и m-угольников.

4. Написать функцию, получающую в качестве параметра имя файла. В котором содержатся размерность и элементы матрицы в виде списка. Написать функцию, вычисляющую обратную матрицу. Результат записать в файл.

Пример входного файла

3 3

2 11 4 5 7 3 2 1 1

5. Написать функцию, получающую в качестве параметра имя файла, содержащего два числа – n и m. Найти на промежутке от n до m числа с наибольшим количеством простых делителей.

Выходных данные – число простых делителей и для каждого числа само число и список его простых делителей. Результат записать в файл.

Пример входного файла:

2 13

Результат:

2

6 2 3

10 2 5

12 2 3

6. Написать функцию, получающую в качестве параметра имя файла, содержащего число в десятичной системе счисления. Перевести число в двоичную и восьмеричную системы. Результат записать в файл.

7. Написать функцию, получающую в качестве параметра имя файла, содержащего карту кораблей и размерность карты. Каждый корабль представляет собой вертикальный или горизонтальный набор подряд идущих закрашенных клеток(1), разные корабли не соприкасаются по сторонам или углам и не накладываются друг на друга. Корабли могут быть более, чем из четырех клеток. Необходимо найти корабль наибольшей площади. Результат записать в файл.

Пример входного файла:

12 12
0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 1 1 1 0 0 0 0 0 1
0 0 0 0 0 0 0 1 0 0 0 1
0 1 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0
0 1 0 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0
0 1 1 0 0 0 0 0 0 0 0 0

Результат:

6

8. Написать функцию, получающую в качестве параметра имя файла, в котором содержатся: количество дорог, информация о каждой дороге в формате первый город, второй город, стоимость проезда (все дороги двусторонние). Так же в файле указаны города начала и конца пути. Требуется проложить наиболее дешёвую дорогу. Результат записать в файл.

9. Написать функцию, получающую в качестве параметра имя файла, в первой строке которого указана размерность первой матрицы(m n), во второй - размерность второй матрицы (n p). В третьей строке - матрица A в виде списка элементов, в четвертой – матрица B. Необходимо вычислить произведение матриц. Результат вычислений записать в файл.

Пример входного файла:

3 3

3 2

3 2 1 4 2 1 2 2 1

4 2 5 1 2 3

10. Написать функцию, получающую в качестве параметра имя файла, содержащего число n и ещё n^2 чисел. Необходимо расположить их в матрице n*n, так что бы определитель был минимален. Результат записать в файл.

Пример входного файла:

2

0 1 2 3

Результат:

0 3

2 1

11. Написать функцию, получающую в качестве параметра имя файла, в котором содержится множество координат точек (в двумерных декартовых координатах) и число n. Написать функцию, находящую n-угольник с наибольшей площадью с вершинами в данных точках.

12. Написать функцию, получающую в качестве параметра имя файла, содержащего элементы двух матриц размерности 3*3 в виде списков. Написать функцию, вычисляющую произведение матриц. Результат записать в файл.

Пример входного файла:

2 11 4 5 7 3 2 1 1

4 5 2 7 1 1 2 1 3

13. Написать функцию, получающую в качестве параметра имя файла, в котором содержится левая часть уравнения, заданного относительно переменной X. В уравнении могут быть использованы скобки и 4 арифметических операции над переменной. Необходимо написать функцию, считывающую уравнение из файла и решающую его относительно X. Результат решения следует записать в файл. Результатами могут быть: указание на отсутствие решения (NO SOLUTIONS); указание на любое значение переменной (ANY X); указание на бесконечное множество решений, за исключением некоторого или некоторых (ANY X EXCEPT y_1, y_2, …, y_n); найденный корень уравнения (ROOT IS y). Все решения должны быть представлены в целочисленной форме, либо в виде несократимых дробей.

Примеры входного файла (для уравнения $2+\frac{0}{x-1}-2=0$ и $8x-20=0$):
2+0/(X-1)-2
8*X-20
Примеры выходного файла:
ANY X EXCEPT 1
ROOT IS 5/2

14. Написать функцию, получающую в качестве параметра имя файла, содержащего коэффициенты системы линейных уравнений, заданные в виде прямоугольной матрицы. С помощью допустимых преобразований привести систему к треугольному виду. Результат записать в файл.

15. Написать функцию, получающую в качестве параметра имя файла, содержащего множество координат точек (в двумерных декартовых координатах) и число n. Найти n-звенную незамкнутую ломанную наибольшей длины с узлами в данных точках. Результат записать в файл.

16. Написать функцию, получающую в качестве параметра имя файла, содержащего два числа – n и m. Найти все пары дружественных чисел в диапазоне от n до m.

Два натуральных числа называются дружественными, если каждое из них равно сумме всех делителей другого, кроме самого этого числа. Результат записать в файл.

17. Написать функцию, получающую в качестве параметра имя файла, содержащего лабиринт и его размерность. Подсчитать количество изолированных областей в лабиринте. Передвижение возможно только по вертикали и горизонтали на незанятые стенами участки. Результат записать в файл.

Пример входного файла:

```
15
4
###############
#   # #         ##
#   #    ## ### #
###############
```

Результат:

```
2
```

18. Написать функцию, получающую в качестве параметра имя файла, содержащего карту кораблей и размерность карты. Каждый корабль представляет собой вертикальный или горизонтальный набор подряд идущих закрашенных клеток(1), разные корабли не соприкасаются по сторонам или углам и не накладываются друг на друга. Корабли могут быть более, чем из четырех клеток. Необходимо найти число кораблей. Результат записать в файл.

Пример входного файла:

```
12 12
0 0 0 0 0 0 0 0 0 0 0 1
0 1 1 1 1 1 0 0 0 0 0 1
0 0 0 0 0 0 0 1 0 0 0 1
0 1 0 0 0 0 0 0 0 0 0 0
```

0 1 0 0 0 0 0 0 0 0 0 0
0 1 0 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0
0 1 1 0 0 0 0 0 0 0 0 0

Результат:
7

19.

КОНТРОЛЬНЫЕ ВОПРОСЫ И ЗАДАНИЯ

1. Укажите синтаксис для объявления функции в CLIPS.
2. Приведите пример удаления функции.
3. Чем отличаются команды *DEFFUNCTION*, *DEFGENERIC* и *DEFMETHOD*?
4. Если несколько перегруженных методов удовлетворяет набору входных параметров, то которые из них будут выполнены?
5. Какие функции используются для ввода и вывода информации? Укажите их синтаксис.
6. С помощью какой функции можно создать список? Приведите ее синтаксис.
7. Приведите пример функции для работы со списками.
8. Приведите пример функции для работы со строками.
9. Что выполняют функции *EXPLODE* и *IMPLODE*?
10. Для чего используются функции *EVAL* и *BUILD*? Приведите пример их работы.
11. Приведите пример работы 5-7 математических функций.
12. Приведите синтаксис присвоения переменной значения.
13. Сформулируйте правила создания условных конструкций.
14. Приведите синтаксис создания циклических конструкций.
15. Приведите пример генерирования случайного числа?

ФОРМА ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ

На выполнение лабораторной работы отводится 3 занятия (6 академических часа: 5 часов на выполнение и сдачу лабораторной работы и 1 часа на подготовку отчета).

Номер варианта студенту выдается преподавателем.

Отчет на защиту предоставляется в печатном виде.

Структура отчета (на отдельном листе(-ах)): титульный лист, формулировка задания (вариант), этапы выполнения работы (со

скриншотами), результаты выполнения работы (скриншоты и содержимое файлов), выводы.

ОСНОВНАЯ ЛИТЕРАТУРА

1. Малышева Е.Н. Экспертные системы [Электронный ресурс]: учебное пособие по специальности 080801 «Прикладная информатика (в информационной сфере)»/ Малышева Е.Н.— Электрон. текстовые данные.— Кемерово: Кемеровский государственный институт культуры, 2010.— 86 с.— Режим доступа: http://www.iprbookshop.ru/22126.— ЭБС «IPRbooks»
2. Павлов С.Н. Системы искусственного интеллекта. Часть 1 [Электронный ресурс]: учебное пособие/ Павлов С.Н.— Электрон. текстовые данные.— Томск: Томский государственный университет систем управления и радиоэлектроники, Эль Контент, 2011.— 176 с.— Режим доступа: http://www.iprbookshop.ru/13974.— ЭБС «IPRbooks»
3. Павлов С.Н. Системы искусственного интеллекта. Часть 2 [Электронный ресурс]: учебное пособие/ Павлов С.Н.— Электрон. текстовые данные.— Томск: Томский государственный университет систем управления и радиоэлектроники, Эль Контент, 2011.— 194 с.— Режим доступа: http://www.iprbookshop.ru/13975.— ЭБС «IPRbooks»
4. Чернышов, В.Н. Системный анализ и моделирование при разработке экспертных систем : учебное пособие / В.Н. Чернышов, А.В. Чернышов ; Министерство образования и науки Российской Федерации, Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Тамбовский государственный технический университет». - Тамбов : Издательство ФГБОУ ВПО «ТГТУ», 2012. - 128 с. : ил. - Библиогр. в кн. ; То же [Электронный ресурс]. - URL: //biblioclub.ru/index.php?page=book&id=277638 (22.02.2017).

ДОПОЛНИТЕЛЬНАЯ ЛИТЕРАТУРА

5. Воронов, А.Е. Технология использования экспертных систем / А.Е. Воронов. - М. : Лаборатория книги, 2011. - 109 с. : ил. - ISBN

978-5-504-00525-6 ; То же [Электронный ресурс]. - URL: //biblioclub.ru/index.php?page=book&id=142527 (22.02.2017).
6. Трофимов, В.Б. Интеллектуальные автоматизированные системы управления технологическими объектами : учебно-практическое пособие / В.Б. Трофимов, С.М. Кулаков. - Москва-Вологда : Инфра-Инженерия, 2016. - 232 с. : ил., табл., схем. - Библиогр. в кн.. - ISBN 978-5-9729-0135-7 ; То же [Электронный ресурс]. - URL: //biblioclub.ru/index.php?page=book&id=444175 (22.02.2017).
7. Интеллектуальные и информационные системы в медицине: мониторинг и поддержка принятия решений : сборник статей / . - М. ; Берлин : Директ-Медиа, 2016. - 529 с. : ил., схем., табл. - Библиогр. в кн. - ISBN 978-5-4475-7150-4 ; То же [Электронный ресурс]. - URL: //biblioclub.ru/index.php?page=book&id=434736 (22.02.2017).
8. Джарратано Дж., Райли Г. Экспертные системы. Принципы разработки и программирование, 4-е издание.: Пер. с англ. – М.: ООО «И. Д. Вильямс», 2007. – 1152 с.: ил. – Парал. тит. англ.

Электронные ресурсы:

9. https://ru.wikipedia.org/wiki/CLIPS - CLIPS — Википедия
10. http://clipsrules.sourceforge.net/ - A Tool for Building Expert Systems (англ.)
11. http://clipsrules.sourceforge.net/WhatIsCLIPS.html - What is CLIPS? (англ.)