Кузнецова Ирина Николаевна

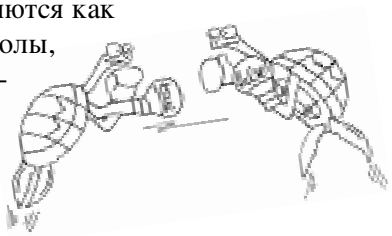
МОДЕЛИРОВАНИЕ В СРЕДЕ ЛОГО
ЗАНЯТИЕ 3. РАБОТА С ПЕРЕМЕННЫМИ

На прошлом занятии для создания алгоритмов нам потребовалось использовать память для хранения и изменения некоторых данных.

Информацию, которая записана в памяти компьютера, называют *данными*.

Процессор читает и записывает эти данные, выполняя команды программы.

В среде программирования Лого данные представляются как числа, символы, слова или логические постоянные (слова *да (true)* и *нет (false)*).



Цвет, звук – это числа или слова.

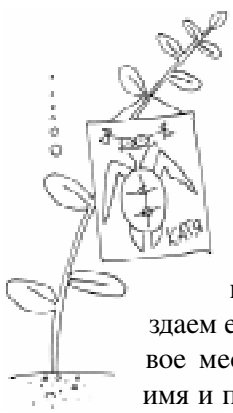
Числа бывают целыми (положительными или отрицательными) и рациональными.

Рациональное число состоит из цифр и запятой (точки), разделяющей целую часть и дробную.

Слово состоит из символов.

Логические постоянные получаются при выполнении сравнений и логических операций *и (and)*, *или (or)*, *не (not)*.

Программы во время выполнения располагаются в оперативной памяти. Например, при загрузке системы Лого Миры в оперативной памяти располагаются программы этой системы, а также данные текущего проекта. Когда вы создаете новую черепашку, она рисуется на экране. А в память записывается ее имя и все ее параметры – цвет, направление головы, координаты и т. д. Создаем еще одну черепашку, и в новое место в памяти записывается имя и параметры этой новой черепашки. Как узнать, где находится параметр определенной черепашки?



Для того, чтобы выделять и использовать определенное место в памяти, в программировании введено понятие переменной.

Переменная – область внутренней памяти, имеющая имя, в которую можно записать некоторое значение и из которой можно прочитать (взять) имеющееся там значение.

В алгоритме и программе переменные используются для хранения и изменения данных во время выполнения.

Некоторые переменные создаются системой, и их имена включаются в язык программирования. Такие переменные называются *системными*.

Все датчики, измеряющие параметры Лого-объектов, и есть имена переменных. Например, *color, heading, xcor, pos* и т.п.

Имеются команды, которые меняют значения этих переменных. Например, для изменения цвета черепашки `t1` используется команда `setc 103`, для изменения положения на рабочем поле – несколько команд: `fd 30`, `setx 100`...

Для хранения и изменения данных во время выполнения алгоритма можно создавать собственные, *пользовательские* переменные.

Например, бегунки и текстовые окна представляют собой переменные. При создании бегунка или текстового окна в памяти появляется область с указанным именем. Если с помощью мыши или клавиатуры менять значение бегунка или текстово-

Таблица 1.

Лого Миры 2		MicroWorldPro
Русская нотация	Английская нотация	
установи <объект> "значение <значение>	**set <объект> "значение <значение>**	**a) set <объект> "value <значение>** **b) set<объект> <значение>**

Пример

Пусть имеется бегунок d1

Лого Миры 2		MicroWorldPro
Русская нотация	Английская нотация	
установи "d1 "значение 25	**set "d1 "значение 25**	**a) set "d1 "value 25** **b) setd1 25**

го окна, то меняется значение в соответствующей области памяти.

Объект *Бегунок* служит для хранения

только целых чисел (положительных и отрицательных) в заданном интервале.

Объект *Текстовое окно* служит для хранения одного числа или строки. Если над содержимым окна производится математическая операция, то значение окна воспринимается как число. При этом символы, отличные от цифр, считаются ошибочными.

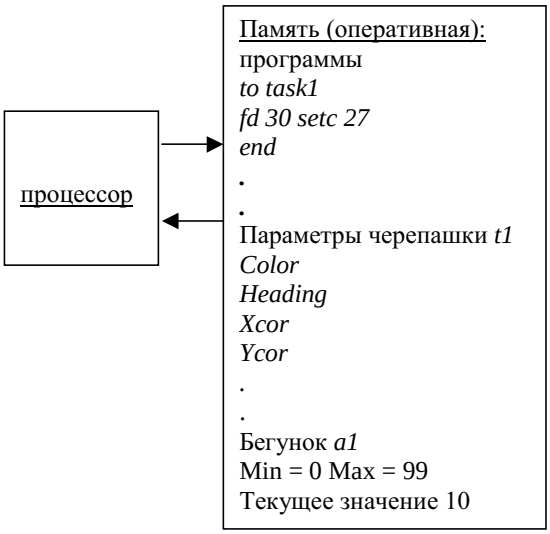


Рисунок 1.

Запись в переменные этого типа может производиться вручную с помощью мыши и клавиатуры. Имеются также специальные команды для записи значения в такие переменные.

К сожалению, форматы этих команд в системах Лого Миры 2 и MicroWorldPro различаются (см. таблицу 1).

Для чтения значения бегунка и текстового окна используется датчик <*объект*>.

Примеры

Пусть имеется бегунок с именем d1.

```
setd1 d1 + 4;в переменную d1 записывается
          ;значение на 4 больше имеющегося
setd1 d1 - 4;в переменную d1 записывается
          ;значение на 4 меньше имеющегося
setd1 d1 * 10;в переменную d1 записывается
          ;произведение значения d1 на 10.
```

Пусть имеется текстовое окно с именем t1.

```
sett1 t1 - 1 ; в окно t1 записывается значение
           ; на 1 меньше
sett1 t1 + 10 ; в окно t1 записывается значение
           ; на 10 больше
sett1 d1 + 4 ; в переменную t1 записывается
           ; значение на 4 больше,
           ; чем в бегунке d1
```

Задание 1. Работа с текстовым окном как с переменной

Вернемся к задаче, которую рассматривали на прошлом занятии (см. рисунок 2).

Вы узнали координаты городов, которые будет посещать черепашка-путешественница:

```
Санкт-Петербург: 62 94
Берн -53 -32
Париж: -90 -36
Лондон: -98 31
```

Как вычислить расстояние, которое пройдет черепашка?

Нам просто необходима переменная – область памяти, в которой будем накапливать сумму расстояний между городами.

Для измерения расстояния используем датчик *distance*. Обычно расстояние не бывает целым числом. Для вычисления суммы расстояний создадим на рабочем поле текстовое окно с именем **s**.

```
to sum_dist
sets 0 ; записать в окно s число 0
t1, setpos [62 94] ; СПб
tour, setpos [-53 -32]
; Добавить расстояние до Берна
sets s + distance "t1
t1, setpos [-53 -32]
tour, setpos [-90 -36]
; Добавить расстояние от Берна до Парижа
sets s + distance "t1
t1, setpos [-90 -36]
tour, setpos [-98 31]
; Добавить расстояние от Парижа до Лондона
sets s + distance "t1
t1, setpos [-98 31]
tour, setpos [62 94]
; Добавить расстояние от Лондона до СПб
sets s + distance "t1
end
```

После выполнения программы **sum_dist** в окне **s** окажется сумма расстояний.

Задание 2. Работа с внутренней переменной

В среде Лого Миры можно также работать с внутренними переменными, создаваемыми только в оперативной памяти.

Такая переменная создается командами **make** или **let**.

Рисунок 2.

Значение внутренней переменной изменяется тоже командой **make** или **let**, например:

```
make "work 100
```

Здесь – "**work** – имя переменной, то есть имя области памяти,

100 – то значение, которое записано в этой области.

Для чтения значения внутренней переменной используется датчик *thing*.

Например, показать в командном центре значение переменной *n* – **show thing "n**

Обычно слово **thing "** заменяется сокращенной записью – символом «**:**» (двоеточие), например, показать в командном центре значение переменной *n*:


```
show :n
```

Во внутреннюю переменную можно записывать данные любого типа – числа как целые, так и рациональные, слова, логические выражения, предложения.

В рассмотренном выше алгоритме подсчета суммы расстояний используем внутреннюю переменную **sum**:

```
to sum_dist
make "sum 0 ; записать в окно s число 0
t1, setpos [62 94] ; СПб
```

```
tour, setpos [-53 -32]
; Добавить расстояние до Берна
make "sum :s + distance «t1
t1, setpos [-53 -32]
tour, setpos [-90 -36]
; Добавить расстояние от Берна до Парижа
make "sum :s + distance «t1
t1, setpos [-90 -36]
tour, setpos [-98 31]
; Добавить расстояние от Парижа до Лондона
make "sum :s + distance «t1
t1, setpos [-98 31]
tour, setpos [62 94]
; Добавить расстояние от Лондона до СПб
make "sum :s + distance «t1
end
```

Задание 3. Процедура с параметрами

На рисунке 3 изображено дерево, состоящее из веток. Надо написать процедуру рисования веток разной длины.

Можно ли значение переменной задавать не внутри процедуры, а при вызове этой процедуры?

Да, для этого переменные, необходимые для работы, перечисляются в заголовке процедуры. Это тоже внутренние переменные, они называются также *параметрами процедуры*. Для изменения значения параметров внутри алгоритма также используется команда **make**.

Процедура рисования одной ветки:

```
; после имени процедуры – имя переменной :h
(формального параметра)
; в памяти создается область с именем h. Мы
не видим ее на рабочем поле
; в командах используется значение, записанное
в этой области памяти
to brush :h
fd :h bk :h
lt 15 fd :h bk :h rt 30
fd :h bk :h
lt 15 bk :h
end
```

Для вызова такой процедуры используется команда:

```
brush 20
```

После имени процедуры указывается значение переменной **h** (*фактический параметр*). Это значение (в данном случае - число 20) записывается в область памяти *h*.

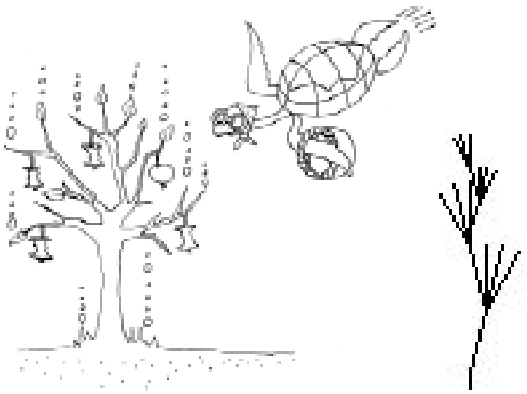


Рисунок 3.

Получается ветка – см. рисунок 4.

Используя процедуру **brush**, напишем программу **tree** для рисования дерева из 4-х веток разной длины. Опишем в этой программе два параметра **:col** – цвет и **:ht** – высота дерева.

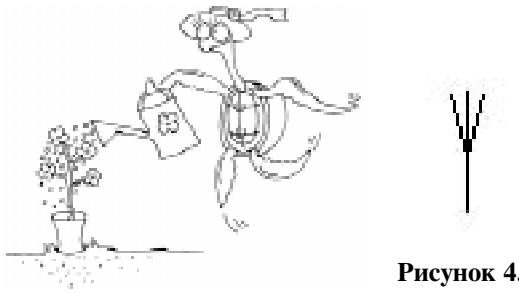


Рисунок 4.

```
to tree :col :ht
setc :col seth 0 pd
lt 15 brush :ht / 2
rt 30 brush :ht / 2
lt 15 bk :ht / 4
lt 15 brush :ht
rt 30 brush :ht
lt 15 bk :ht / 4 pu
end
```

Как запустить такую программу? Введем команду:

```
tree 75 2 0
```

Независимо от имени параметра, первое значение всегда попадает в переменную, описанную в заголовке первой, а второе значение – в переменную, описанную в заголовке второй. После выполнения команды на рабочем поле нарисовано дерево зеленого цвета (см. рисунок 3).

Задание 4. Переменные в рекурсивном алгоритме

Переменные открывают новые возможности для разработки красивых алгоритмов.

Рассмотрим такой простой алгоритм:

```
to circle
fd 1 rt 1
circle
end
```

Черепашка проходит вперед на 1 шаг, поворачивается на 1 градус и... снова выполняется этот же алгоритм. Из процедуры вызывается эта же самая процедура. Такой алгоритм называется *рекурсивным*. В предложенном варианте выполнение процедуры `circle` никогда не закончится, а черепашка будет бегать по окружности.

Чтобы остановить бесконечный процесс, нужно вставить в алгоритм проверку какого-то условия. Откуда взять данные для проверки? Конечно, из памяти. Например, можно вычислять текущий угол поворота и сравнивать его с числом 360. Для этого зададим параметр процедуры `:a` и будем использовать его для вычисления общего значения угла поворота:

```
to circle :a
if :a < 360 [fd 1 rt 1
 make "a :a + 1
 circle :a]
end
```

После выполнения команды `circle 0` на рабочем поле будет нарисована одна окружность (см. рисунок 5).

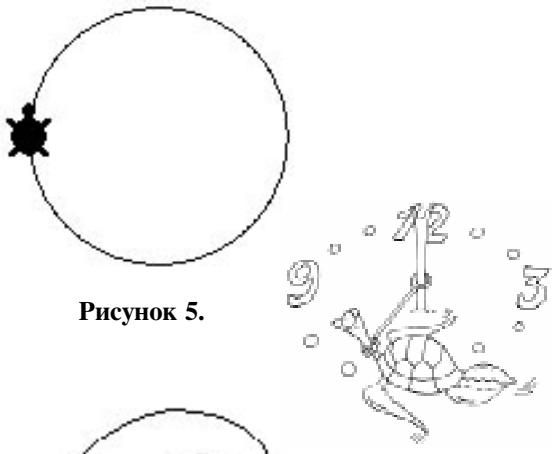


Рисунок 5.

После выполнения команды `circle 180` на рабочем поле будет нарисована половина окружности.

А что получится, если черепашка будет поворачиваться на угол `:a`?

```
 to circled :a
if :a < 180 [fd 10 rt :a
 make "a :a + 1
 circled :a]
end
```

После выполнения команды `circle 0` на рабочем поле появится спираль (см. рисунок 6).

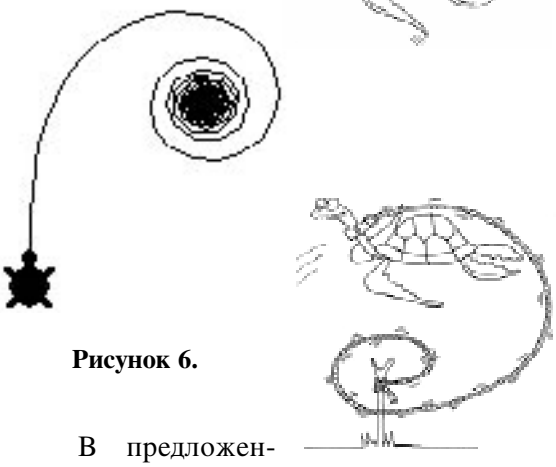


Рисунок 6.

В предложенном алгоритме `circle` команда `make "a :a + 1` лишняя. Можно просто фактический параметр процедуры записать в виде суммы `:a + 1`:

```
to circled :a
if :a < 180 [fd 10 rt :a
 circled :a + 1]
end
```

Алгоритм `square` предназначен для рисования спирали с прямыми углами (рисунок 7).

```
to square :s
if :s < 90 [fd :s rt 90
 square :s + 2]
end
```

Какая команда выполнена для рисования спирали, изображенной на рисунке 7?

Рекурсивные алгоритмы с параметрами широко используются в математике.

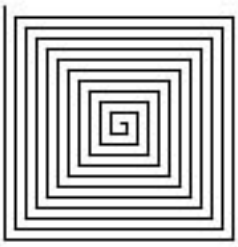


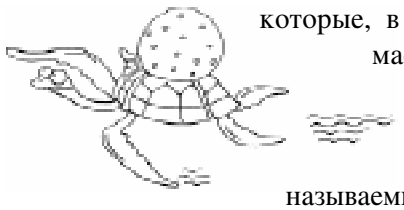
Рисунок 7.

Задание 5. Обработка данных разного объема

При составлении алгоритма учитывается объем обрабатываемых данных и, в зависимости от этого, выбирается как способ размещения их в памяти, так и метод получения или поиска данных для преобразования.

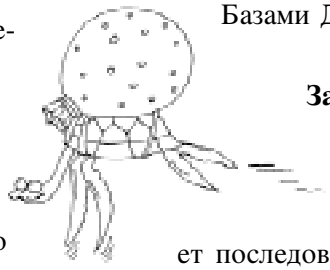
Будем использовать три категории для обозначения объема данных – малый, средний и большой.

В алгоритме для хранения и совершения действия с каждым таким объемом данных создается одна переменная, которая имеет одно значение.



Пример из Лого:

а) цвет, форма объекта «черепашка» указывается числами, которые хранятся в отдельных переменных *color* и *shape*.



б) одно число можно записать в одном бегунке.

Примерами данных среднего объема являются очереди, списки.

Для хранения данных среднего объема в памяти создается переменная типа *массив* или *список*, которая имеет одно имя и много значений, расположенных друг за другом. Каждое такое значение называют *элементом*, каждый элемент имеет порядковый номер.

Пример из Лого:

а) место на рабочем поле указывается с помощью двух чисел, которые хранятся в переменной *pos*. Первое число – всегда координата по оси *X*, второе – координата по оси *У*;

б) целое число состоит из цифр, которые пронумерованы по разрядам.

Алгоритм, в котором используются данные среднего объема, должен содержать действия по поиску элемента в списке или массиве.

Пример из Лого:

Существуют специальные функции для определения первого, последнего элемента или элемента с указанным номером в последовательности цифр.

В современном мире создана глобальная сеть компьютеров – Интернет, в которой хранится огромное количество информации, обновляемой ежедневно. Для поиска конкретных сведений в таких хранилищах созданы и создаются специальные алгоритмы.

Данные большого объема хранятся во внешней памяти компьютера – создаются базы данных, которые, в свою очередь, состоят из массивов или списков.

Алгоритмы поиска данных в базах объединяются в отдельные системы, называемые Системами Управления Базами Данных (СУБД).

Задание 6. Обработка данных среднего объема

Использование массивов или списков позволяет последовательный алгоритм свести к циклическому.

Вернемся к задаче о черепашке-путешественнице. Мы задали координаты четырех городов постоянными числами. Для подсчета расстояния между каждыми двумя городами в алгоритм включаются три команды, поэтому программа состоит из 12 команд.

Если городов будет 10, программа уже будет включать 30 команд.

Зададим координаты городов в виде списка. Каждый элемент списка содержит пару координат очередного города. Количество элементов в списке равно количеству городов, посещаемых черепашкой. Этот список можно формировать отдельно.

Тогда программа подсчета расстояния, пройденного черепашкой `t1`, будет выглядеть так:

```
to sum_distl :lst_cap
make "s 0
make "k count :lst_cap ;количество городов
make "i 1 ;номер текущего города
repeat :k - 1 [make "w item :i :lst_cap
 t1, setpos :w
 make "w item :i + 1 :lst_cap
 tour, setpos :w
; Добавить очередное расстояние
 make "s :s + distance "t1
 make "i :i + 1
 ]
show :s
end
```

Вызываем эту программу командой:

```
sum_distl [[62 94][-53 -32][-90 -36][-98 31]
[-100 30][62 94]]
```

Алгоритм программы **sum_distl** сложнее, чем алгоритм **sum_dist**. В него включены действия для выбора элементов из списка. Но зато этот алгоритм НЕ ЗАВИСИТ от количества городов (элементов в списке). Мы применили его к списку из 5 городов. Для списка из 100 городов алгоритм подсчета расстояния будет таким же.

Применение массивов или списков позволяет создать алгоритмы, не зависящие от количества обрабатываемых элементов. К сожалению, при изучении языка Лого в школе редко доходят до обработки списков. А ведь именно на списках можно познакомиться со многими классическими алгоритмами поиска, сортировки, исследования графов и т. д.

Приведем встроенные функции для работы со списками Лого (таблица 2).

Задание 7. Обработка данных большого объема

В туристских фирмах имеются каталоги (базы данных), содержащие сведения о разных странах, городах и т. п. В базе данных информация об одном городе представлена в виде списка, в котором в определенном порядке записаны сведения.

Эти каталоги подготавливаются специалистами, которые изучают новые места, создают новые маршруты. Операторы туристских фирм пользуются такими базами данных для выбора маршрута туристу.

Например, будем записывать в текстовое окно *DB_cap* данные о городах в таком порядке:

название города – координаты его на карте.

Сведения о каждом городе располагаются на отдельной строке.

Специалист исследует карту и записывает все данные в окно (рисунок 8).

Текст из этого окна можно записать в файл командой:

```
savetext "euro1
```

Сведения о городах Азии, записанные в текстовое окно, также сохраним в файле:

```
savetext "africa.
```

Так получится несколько файлов со списками городов – маленькая база данных.

Для просмотра и выбора информации из этой базы нужно написать специальные программы. Все необходимые для этого средства имеются в языке программирования Лого. Пример Базы данных, созданной в среде Лого, представлен в проекте База.mw2, который поставляется на CD с системой Лого Миры 2 (см. папку Проекты\Пособия).

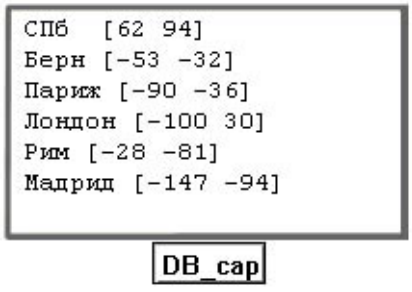


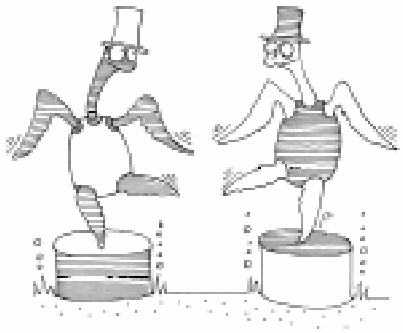
Рисунок 8.

Таблица 2.

Название		Пример	Значение
Русская нотация	Английская нотация		
Подсчет и выбор элементов списка			
сколько	**count**	**count [12 13 24 35 45]**	Число элементов в списке – 5
первый	**first**	**first "2002**	Первый элемент списка – 2
последний	**last**	**last [ф б в г д]**	Последний элемент списка – д
элемент	**item**	**item 4 [12 13 14 15 16]**	Элемент с номером 4 – 15
Изменение содержимого списка			
кпрв	**butfirst**	**butfirst [12 13 14 15 16]**	Список без первого элемента: *[13 14 15 16]*
кпсл	**butlast**	**butlast [12 13 14 15 16]**	Список без последнего элемента: *[12 13 14 15]*
вксп	**lput**	**lput 43 [12 13 14 15 16]**	Список с добавленным в конец элементом: *[12 13 14 15 16 **43**]*
внсп	**fput**	**fput 43 [12 13 14 15 16]**	Список с добавленным в начало элементом: *[**43** 12 13 14 15 16]*
Формирование (склеивание) частей в список (слово)			
список	**list**	**list [12 13] [14 15 16]**	Список из двух списков: *[[12 13] [14 15 16]]*
предложение	**sentence**	**sentence [12 13] [14 15 16]**	Список из слов двух списков: *[12 13 14 15 16]*
слово	**word**	**word "Ло "го**	Склеивание слова: "Лого

Задание 9. Глобальные и локальные переменные

Каждая переменная занимает определенное место в памяти. Однако программы имеют неодинаковый доступ (чтение и запись) к этим переменным. Различают *глобальные* и *локальные* переменные.



Глобальные переменные доступны из любой выполняемой программы. Глобальными являются *системные* переменные (параметры Лого-объектов), переменные, созданные как *бегунки*, *текстовые окна,* и переменные, созданные командой **make**. Имя глобальной переменной обязательно уникально. Системные переменные, бегунки, текстовые окна должны иметь уникальные имена на одном листе проекта. Если разным объектам на разных листах дать одинаковые имена, то команды будут видеть переменную, соответствующую объекту с указанным именем на текущем листе.

Проверим это утверждение:

1. Создайте на листе1 черепашку **t1**, на листе2 также черепашку **t1**. Установите черепашке на листе1 цвет с номером 15, а черепашке на листе2 – цвет с номером 105.
2. Сделайте текущим лист1. Ведите команду **show color**. В командном центре появится число 15.
3. Сделайте текущим лист2. Ведите команду **show color**. В командном центре появится число 105.

Локальные переменные образуются при вызове программы с параметрами, а также с помощью команд **local** и **let**.

Локальные переменные существуют только до тех пор, пока выполняется создавшая их программа. После завершения этой программы все ее локальные перемен-

ные удаляются из памяти, память освобождается. Локальные переменные доступны также процедурам, которые вызываются из программы, создавшей эти локальные переменные.

Создайте текстовое окно для вывода результатов работы следующих программ:

```
to localvar :v1
pr (se "localvar "begin :v1)
lv2
pr (se "localvar "end :v1)
end

to lv2
pr (se "lv2 "begin "v1: :v1)
let [v1 [v1 внутри lv2]]
pr (se "lv2 "end :v1)
end
```

Введите команду: `localvar 1`

В текстовом окне (рисунок 9) видно, что значение параметра `v1` не меняется к концу выполнения программы `localvar`. В процедуре `lv2` сначала была прочитана переменная `v1`. Затем была создана локальная переменная с таким же именем, ее значение выведено во время работы этой процедуры.

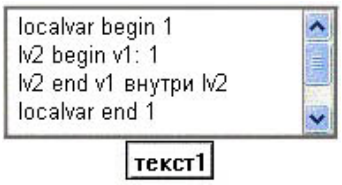


Рисунок 9.

Задание 8. Объектный подход

Существование локальных переменных позволяет не перегружать память лишними данными. Глобальные переменные необходимы, если такие данные читаются или меняются разными программами или должны быть видны на рабочем поле.

Однако изменение одной переменной разными программами может привести к серьезным ошибкам, которые трудно обнаружить. При создании моделей нужны определенные технологические приемы для избежания подобных ошибок.

В современном программировании принят объектный подход в разработке программных моделей – игр, имитационных сред, обучающих программ.

Сначала описываются объекты, которые будут действовать в новой среде, то есть определяются параметры этих объектов, создаются программы, которые меняют или читают эти параметры. Программы и данные (параметры), обрабатываемые этими процедурами, объединены в одну структуру, называемую *объектом*. Данные определенного объекта можно обработать только процедурами этого объекта, а процедуры объекта можно использовать только с данными этого объекта. Процедуры (и функции) объекта называются *методами*, а данные называются *свойствами* или *полями.*

С объектного подхода мы начали изучение среды Лого. Его будем применять при разработке различных программных моделей.

Кузнецова Ирина Николаевна, учитель информатики школы № 640, координатор секции Лого-Лего Международной конференции «Школьная информатика и проблемы устойчивого развития».