Муниципальное общеобразовательное учреждение
средняя общеобразовательная школа имени А.И. Крушанова
с. Михайловка Михайловского муниципального района
Приморского края

Среда kTurtle.
Пособие для учителя.

Никитенко Павел Александрович,
учитель информатики и ИКТ

Михайловка, 2009 г.

Содержание

Введение

Первая версия языка программирования Logo была создана Сеймуром Пейпертом (Seymour Papert) в Лаборатории Искусственного Интеллекта Массачусетского Технологического Института в 1967 как ответвление языка программирования LISP. Впоследствии вышло в свет много его версий. К 1980 Logo становится очень популярным, активно используются его версии для MSX, Commodore, Atari, Apple II и IBM PC компьютеров – главным образом, в образовательных целях. В 1985 году компания LCSI выпустила среду MacLogo как профессиональный инструмент программирования, но она не получила распространения. Массачусетский Технологический Институт до сих пор поддерживает сайт, посвященный Logo, который располагается по адресу http://el.media.mit.edu/logo-foundation/. Кроме того, информацию непосредственно о kTurtle можно получить по адресу: http://edu.kde.org/kturtle/

На сегодняшний день существует большое количество версий и диалектов Logo, информацию о которых можно найти на сайте МТИ (см. выше) или при помощи любой поисковой системы.

Различие в диалектах Logo проявляется и в школьных дистрибутивах. Так, например, дистрибутивы Альт Школьный Мастер 4.0, Альт Школьный Джуниор 4.0 и 5.0, Альт Школьный Легкий 4.0 и 5.0 содержат реализацию kTurtle ver 0.6, построенную на базе оконной среды KDE 3, а Альт Школьный Мастер 5.0 — kTurtle ver 0.8 на базе KDE 4. Между этими реализациями существует различие в использовании команд, их наименовании. Кроме того, KDE 4 обладает гораздо более мощными возможностями в вычислениях — ему доступны тригонометрические и многие математические функции. Поэтому, в данном пособии параллельно описаны обе версии среды программирования.

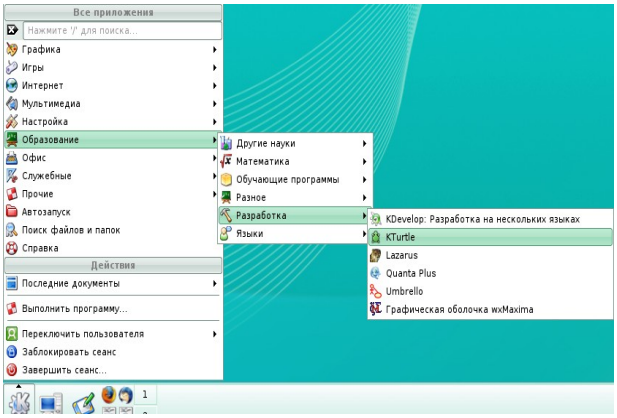
Если Вам встретилось **KDE 3**, значит данный пример будет достоверно работать в kTurtle 0.6, которая используется в следующих дистрибутивах:

- *Альт Школьный Мастер 4.0*
- *Альт Школьный Джуниор 4.0*
- *Альт Школьный Джуниор 5.0*
- *Альт Школьный Легкий 4.0*
- *Альт Школьный Легкий 5.0*

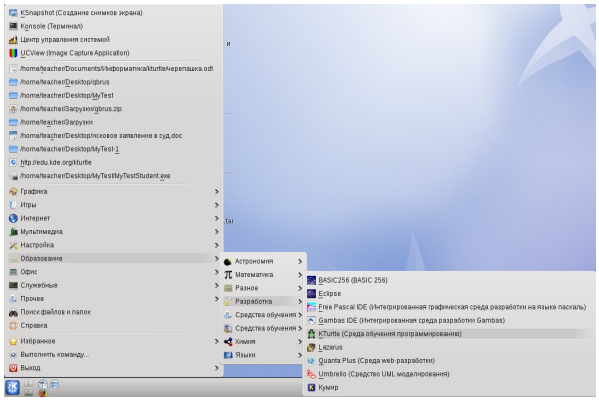
А указание **KDE 4** сообщает, что данный пример будет работать в kTurtle 0.8, которая используется в *Альт Школьный Мастер 5.0*

Интерфейс kTurtle

Для запуска среды kTurtle необходимо нажать на кнопку «К», выбрать пункт «Образование», затем «Разработка», в которой запустить KTurtle.

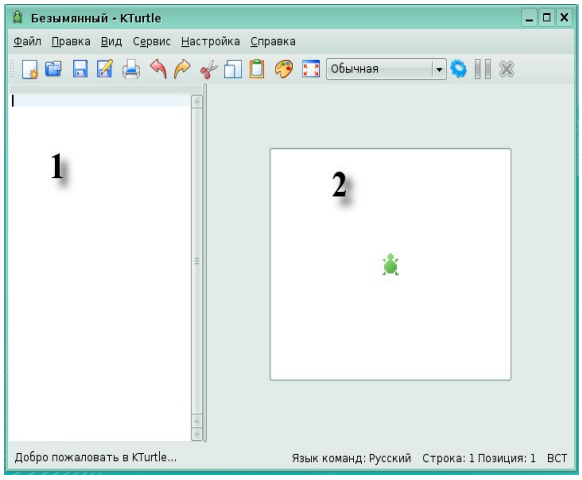


Запуск kTurtle в среде KDE 3.5

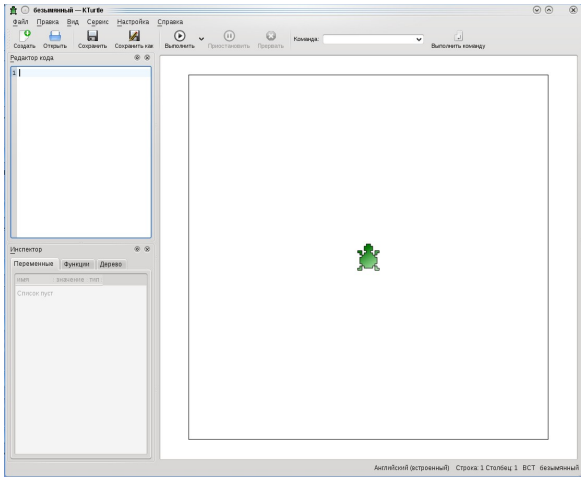


Запуск kTurtle в среде KDE 4.2

Главное окно kTurtle (в среде KDE 3) имеет панель меню, панель инструментов, а также редактор программы (1), расположенный слева, который предназначен для ввода команд Logo, и холста (2) , расположенного справа, на котором изображаются результаты выполнения команд. Под ними находится строка состояния, которая показывает язык, на котором необходимо вводить команды (допускается как английский, так и русский), местоположение курсора, режим ввода текста (вставка или замена). KDE 4 предоставляет несколько большие возможности по управлению Черепашкой: кроме редактора программы (обозначенного в виде окна «Редактор кода»), имеется окно «Инспектор» в котором можно просматривать значения объявленных в программе переменных, подпрограмм и саму программу в виде дерева команд — включить и выключить нужное окно можно при помощи соответствующей команды меню «Вид»; также существует возможность командного режима (команды вводят в строке «Команда» панели инструментов).



Общий вид kTurtle в среде KDE 3.5

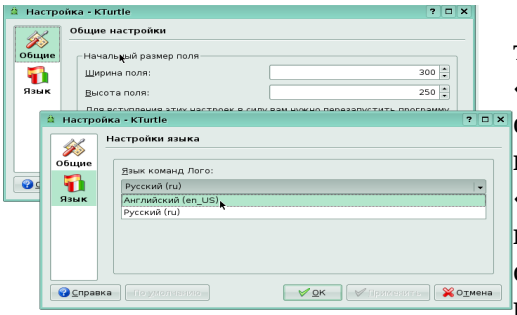


Общий вид kTurtle в среде KDE 4.2

Поскольку программы набираются в окне редактора, то над ними можно выполнять практически все стандартные процедуры обычного текстового редактора. Выделив фрагмент программы, и воспользовавшись меню «Правка», можно вырезать и скопировать выделенный фрагмент, который впоследствии можно вставить.

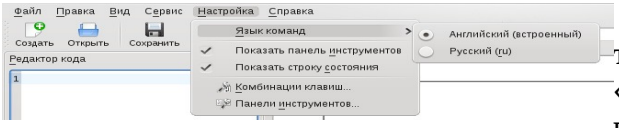
При помощи меню «Настройка» можно настроить kTurtle под свои, конкретные, условия. Так, например, можно изменить язык ввода команд. Изначально языком для ввода

команд выбран русский. Можно долго спорить о необходимости использования такого режима команд, однако, на мой взгляд, гораздо более продуктивным является использование англоязычных команд черепашки, поскольку они позволяют активно использовать межпредметные связи между английским языком и информатикой, приучают к строгости английского языка, к тому что впоследствии, при изучении программирования, будут использованы именно англоязычные команды.



Изменение настроек kTurtle в среде KDE 3.5

Для смены языка команд (в среде KDE 3) требуется выполнить следующую команду: «Настройка»/ «Настроить kTurtle...», после чего в открывшемся окне перейти во вкладку «Язык» и в выпадающем списке «Язык команд Лого» выбрать «Английский (en_US)». Подтверждаем выбор нажатием кнопок «Применить» и «ОК». Чтобы сделанные изменения вступили в силу, требуется перезапустить систему, для этого нужно закрыть окно kTurtle и запустить его вновь. После этого система kTurtle будет готова к работе с новыми настройками. Аналогично, во вкладке «Общие», можно изменить размер поля черепашки, указанный по умолчанию.



Изменение настроек kTurtle в среде KDE 4.2

Для смены языка команд (в среде KDE 4) требуется выполнить следующую команду: «Настройка»/ «Язык команд» в котором выбрать пункт «Английский»

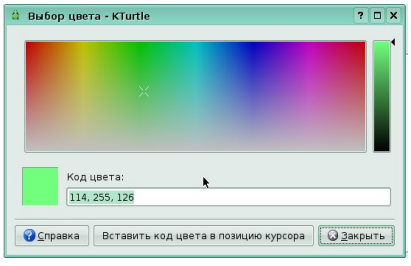
Доступ к основным, наиболее часто используемым командам осуществляется, также, при помощи панели инструментов

Панель инструментов kTurtle в среде KDE 3.5

На ней, последовательно слева направо, расположены следующие инструменты: **Создать** новую программу, **Открыть** сохраненную ранее программу, **Сохранить программу**, **Сохранить** полученное **изображение**, **Печать** программы или полученного изображения, **Отменить** действие в редакторе программ, **Вернуть** предыдущее изменение в тексте программы, **Вырезать** и **Копировать** выделенный фрагмент программы, **Вставить** ранее скопированный (вырезанный) фрагмент программы, **Определить код цвета** при помощи палитры, **Развернуть** среду в полноэкранный режим (после перевода в полноэкранный режим данный инструмент изменяется в **Вернуть первоначальное окно** среды), **Изменение скорости** движения «черепашки» по полю, **Выполнить** введенную программу, **Приостановить** выполнение программы, **Прекратить** выполнение программы.

В среде KDE 4 панель инструментов выглядит следующим образом:

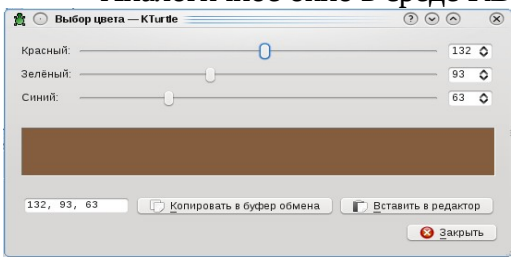
Панель инструментов kTurtle в среде KDE 4.2



Окно выбора кода цвета kTurtle в среде KDE 3.5

В том случае, если требуется узнать код цвета (для среды KDE 3) можно воспользоваться соответствующим инструментом, как было сказано выше. В этом случае на экране появится окно, перемещая курсор, в котором, по полю цвета, удерживая при этом левую кнопку мыши, можно как просто узнать код цвета, так и вставить его значение в позицию, указанную курсором в тексте программы, что очень удобно при использовании команд, изменяющих цвет пера «черепашки» или цвет холста.

Аналогичное окно в среде KDE 4 выглядит несколько иначе:



Окно выбора кода цвета kTurtle в среде KDE 4.2

Здесь нужно перемещать соответсвующие бегунки (красной, зеленой и синей составляющей цвета), при этом будет изменяться цвет прямоугольника, расположенного ниже. Там же, расположено поле в которое выводится код выбранного цвета. Его можно скопировать в буфер обмена или вставить сразу в редактор программы.

Допускается и изменение скорости движения «черепашки», которое можно выполнить при помощи соответствующего выпадающего списка (для KDE 3) или кнопки, расположенной справа от кнопки «Выполнить» (KDE 4), в котором допускаются 4 основных скорости. При отладке программ, а также демонстрации выполнения команд, полезно использовать режим **Самая медленная**. При выполнении заданий учениками, чаще всего, подойдет **Обычная**, т.к. в этом случае результат выполнения программы виден практически сразу, после запуска.

Основные команды kTurtle

В kTurtle, в отличии от большинства других реализаций языка Logo, допускается несколько способов указания команд. Их можно вводить как на национальном языке (например, на русском), так и на английском (на мой взгляд такой вариант наиболее практичен, поскольку приучает детей к строгости английского языка, а также позволяет органично связать знания, получаемые ими на уроке английского языка, и информатику), кроме того в обоих вариантах допускается использование сокращенных названий команд (хотя этой возможностью, на мой взгляд, лучше все-таки не пользоваться). В данном пособии все команды будут описаны в английском варианте, поскольку описания русскоязычных команд можно найти непосредственно в справке программы kTurtle, сокращенный вариант будет, по возможности, указан в скобках, после **сокр.**.

Для работы с переменными необходимо помнить, что хотя kTurtle и позволяет использовать в качестве имен переменных русские имена, однако использование русских имен переменных в других языках программирования недопустимо. Поэтому, крайне важно упомянуть о такой возможности, но в дальнейшем ее не использовать. В целом же для работы с переменными используются те же механизмы, что и в любом другом языке программирования. Т.е. имена переменных не могут совпадать с именами команд, не могут начинаться с цифры, могут состоять из букв (лучше латинских), цифр и символа подчёркивания «_», но первой всегда должна быть буква (в версии kTurtle, используемой в дистрибутиве Школьный Мастер 5, имя переменной обязательно должно начинаться со знака доллара - $). Присваивание значения переменной производится символом «=». Причем, символ «=» лучше читать не как «равно», а как «присвоить». В качестве значения переменной можно использовать любое число, в том числе и вещественного типа, строку, которую следует заключить между знаков кавычки, а также результат вычислений по формуле. В формулах kTurtle поддерживает все основные математические операции: сложение «+», вычитание «-», умножение «*», деление «/», для изменения порядка вычислений можно использовать скобки «(» и «)».
Значение переменной можно передавать командам, требующим дополнительный параметр. Для этого, после команды указываем имя переменной, значение которой требуется передать данной команде.

Например:

KDE 3	KDE 4
x=10 *s=”строка символов”* *y=2*x/10* *Forward x*	*$x=10* *$s=”строка символов”* *$y=2*$x/10* *forward $x*

Для перемещения «черепашки» используются следующие команды:
forward L (сокр. **fw L**) – пройти вперед на L шагов
backward L (сокр. **bw L**) – пройти назад на L шагов, «черепашка» при этом не поворачивается
go X,Y – перейти в точку с координатами (x,y), при этом переходе «черепашка» след не оставляет

Команды, используемые для изменения направления движения (черепашка при этом не перемещается):
turnright A (сокр. **tr A**) – повернуть «черепашку» направо на угол величиной A градусов
turnleft A (сокр. **tl A**) – повернуть «черепашку» налево на угол величиной A градусов
direction A (сокр. **dir A**) – изменить первоначальное направление «черепашки» направо на угол величиной A градусов

Команды, изменяющие статус «черепашки»:
reset – очищает экран и возвращает «черепашку» в начальное положение в центре холста
clear – очищает экран, но не возвращает «черепашку» в начальное положение
penup (сокр. **pu**) – «поднимает» перо «черепашки». В результате «черепашка» не оставляет при своем движении след
pendown (сокр. **pd**) – «опускает» перо «черепашки». В результате «черепашка» оставляет при своем движении след
penwidth N – изменяет ширину следа, оставляемого «черепашкой», в пикселях
pencolor R,G,B – изменяет цвет пера «черепашки». Цвет указан в виде комбинации чисел в интервале от 0 до 255 для R (красная составляющая), от 0 до 255 для G (зеленая составляющая) и от 0 до 255 для B (синяя составляющая). Для определения необходимой комбинации цветовых составляющих можно воспользоваться инструментом *Выбор цвета* на панели инструментов.
canvascolor R,G,B – изменяет цвет фона холста. Комбинации R,G,B также можно определить при помощи инструмента *Выбор цвета*
canvassize Lx,Ly – изменяет размеры холста по горизонтали Lx и вертикали Ly в пикселях

KDE 3	KDE 4	
Hide	**sh**	спрятать «черепашку»
Show	**ss**	показать «черепашку»

Команды, реализующие циклические алгоритмы:
kTurtle имеет в своем наборе команды, реализующие практически все основные типы циклов, однако при их записи крайне желательно соблюдать следующее правило: в строке программы, содержащей открывающую скобку «**[**» и закрывающую скобку «**]**», должны отсутствовать любые команды. В версии для Школьного Мастера 5 открывающая скобка выглядит как «{», а закрывающая - «}», причем они могут ставиться и в строку содержащую команды.

Цикл «повторить N раз»

KDE 3	KDE 4	
Repeat N	**Repeat N**	Повторить *N* раз команды составляющие *тело цикла*
[	**{**	
тело цикла	*тело цикла*	
]	**}**	

Цикл «для...»

KDE 3	KDE 4	
For *ПЦ=НЗ* **to** *КЗ*	**For** *ПЦ=НЗ* **to** *КЗ*	Для всех значений *переменной цикла (ПЦ)*, от *начального (НЗ)* до *конечного (КЗ) значения*, выполнять команды составляющие *тело цикла*
[	**{**	
тело цикла	*тело цикла*	
]	**}**	

Цикл «пока...»

KDE 3	KDE 4	
While *условие*	**while** *условие*	Пока *условие* верно, выполнять команды составляющие *тело цикла*
[	**{**	
тело цикла	*тело цикла*	
]	**}**	

Команды, реализующие разветвляющиеся алгоритмы:

При их записи крайне желательно соблюдать правило, аналогичное правилу записи циклов: в строке программы, содержащей открывающую скобку «**[**» и закрывающую скобку «**]**», должны отсутствовать любые команды.

Важно помнить! Для записи операций отношения (например, при записи условий в условном ветвлении, цикле «пока») используются обозначения, схожие с используемыми в языке программирования Си: **равно** – двумя знаками равно «==» (пробел между ними не ставится), **не равно** – знаками восклицательного знака и равно «**!=**», **больше** – «>», **меньше** – «<», **больше или равно** – «>=», **меньше или равно** – «<=».

Полное условное ветвление

KDE 3	KDE 4	
If *условие*	**if** *условие*	Если *условие* истинно, то выполнить команды *Действия 1*, в противном случае, если условие не истинно – выполнить команды *Действия 2*
[	**{**	
Действие 1	*Действие 1*	
]	**}**	
[	**else {**	
Действие 2	*Действие 2*	
]	**}**	

Неполное условное ветвление

KDE 3	KDE 4	
If *условие*	**if** *условие*	Если *условие* истинно, то выполнить команды *Действия*
[	**{**	
Действие	*Действие*	
]	**}**	

Команды, реализующие вспомогательные алгоритмы:

Создание подпрограммы (правило открывающей и закрывающей скобки действует и здесь)

KDE 3

Learn *ИмяПодпрограммы ИмяПеременной*
[
Тело подпрограммы
]

ИмяПеременной не является обязательным параметром!

Для работы подпрограммы необходимо вызвать ее из основной программы, указав ее имя в качестве команды с обязательным указанием значения переменной. Имя подпрограммы подчиняется правилам создания имен переменных.

KDE 4

Learn *ИмяПодпрограммы ИмяПеременной*
{
Тело подпрограммы
}

ИмяПеременной не является обязательным параметром!

Для работы подпрограммы необходимо вызвать ее из основной программы, указав ее имя в качестве команды с обязательным указанием значения переменной. Имя подпрограммы подчиняется правилам создания имен переменных.

Вызов подпрограммы:

ИмяПодпрограммы ЗначениеПеременной

Значение переменной имеет смысл передавать только в том случае, если при создании подпрограммы была указана переменная.

Возвращение значения локальной переменной подпрограммы:

Return *ИмяЛокальнойПеременной*

Пример использования подпрограмм:

KDE 3

```
Learn Квадрат_числа x
[
y=x*x
Return y
]

Learn Построить_Квадрат L
[
 Repeat 4
 [
 Forward L
 TurnRight 90
 ]
]

Reset
C=Квадрат_числа   7
Построить_Квадрат C
```

KDE 4

```
learn Квадрат_числа $x
{
$y=$x^2
return $y
}

learn Построить_Квадрат $L
{
 repeat 4
 {
 forward $L
 turnRight 90
 }
}

reset
$C=Квадрат_числа   7
Построить_Квадрат $C
```

Дополнительные команды:

Комментарий:
Символ «#» (решетка) в начале строки сообщает интерпретатору о том, что в данной строке записан комментарий к программе. Все что написано в строке после этого знака интерпретатор Logo игнорирует и не выполняет.

Вывод сообщения на холст:

Print "*сообщение*" **Print** *ИмяПеременной* **Print** *ИмяПеременной*+"*сообщение*"	Данная команда позволяет вывести текстовое сообщение и (или) значение переменной на холст в то место, где в данный момент находится «черепашка». Если требуется вывести несколько сообщений одновременно, то между ними ставится знак «+»

Пример использования команды вывода сообщения:

KDE 3	**KDE 4**
Reset C=10*20 Print "C="+C	reset $C=10*20 print "C="+$C

В центр холста будет выведено **C=200**

fontsize *Величина* – устанавливает размер символов, используемых при выводе текста на холст.

message "*сообщение*" - вызывает на экран окно, содержащее текст сообщения. Синтаксис команды схож с синтаксисом команды **print**.

Ввод значения переменной с клавиатуры:

ИмяПеременной=**InputWindow** "*вопрос*" – при выполнении на экране появляется окно содержащее *вопрос* и текстовую строку в которую необходимо ввести значение *переменной*.

В версии KDE 4, эту же функцию выполняет другая команда:
ИмяПеременной=**ask** "*вопрос*"

Пример использования команды ввода:

KDE 3	**KDE 4**
X=InputWindow "*Чему равно X?*"	$X=ask "*Чему равно X?*"

При выполнении этого фрагмента программы, на экран выводится окно с запросом «Чему равно X?», в текстовом поле которого требуется ввести значение, которое и будет присвоено переменной X.

Генерирование псевдослучайных чисел:
ИмяПеременной=**Random** *НижнееЗначение*, *ВернееЗначение* – генерирует псевдослучайное число, находящееся в интервале от *НижнегоЗначения* до *ВерхнегоЗначения*

Пример использования команды генерирования псевдослучайных чисел:

KDE 3	KDE 4
X=Random 10, 100	$X=random 10, 100

При выполнении данного фрагмента программы, переменной X будет присвоено случайное значение в интервале чисел от 10 до 100.

Математические, и другие, функции, появившиеся в KDE 4

ИмяПеременной=**round** *Значение* — округляет *Значение*
ИмяПеременной=**sqrt** *Число* — вычисляет значение квадратного корня из *Числа*
ИмяПеременной=**exp** *Число* — вычисляет значение степени числа e^x с показателем степени записанном в виде *Числа*
кроме того, в новой версии kTurtle появился оператор возведения числа в произвольную степень, записываемый, как и во многих других языках программирования, символом «^»

тригонометрические функции:

ИмяПеременной=**sin** *Угол* — вычисляет значение *синуса Угла*
ИмяПеременной=**cos** *Угол* — вычисляет значение *косинус Угла*
ИмяПеременной=**tan** *Угол* — вычисляет значение *тангенс Угла*
ИмяПеременной=**arcsin** *Значение* — вычисляет значение арксинуса
ИмяПеременной=**arccos** *Значение* — вычисляет значение арккосинуса
ИмяПеременной=**arctan** *Значение* — вычисляет значение арктангенса
***pi* —** константа, равная числу π

функции, обрабатывающие положение черепашки:

ИмяПеременной=**getx —** присваивает переменной значение текущей горизонтальной координаты (x) черепашки
ИмяПеременной=**gety —** присваивает переменной значение текущей вертикальной координаты (y) черепашки

Применение kTurtle при изучении основ программирования и алгоритмизации.

В рамках данного пособия рассматриваются лишь некоторые темы занятий, на которых применяется среда kTurtle на уроках информатики 6 и (или) 7 класса. Однако, стоит отметить, что данная среда обладает достаточными возможностями и для использования в более старших классах.

Линейные программы. Знакомство со средой kTurtle.

В ходе данного занятия учащиеся должны познакомиться с интерфейсом среды, командами Лого, выполнить простейшее задание на написание линейной программы.

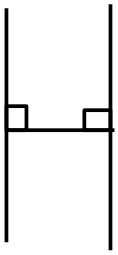
Для этого, учащиеся должны уже знать что такое: угол, в каких мерах он измеряется, что такое развернутый, прямой, острый и тупой угол, смежные углы. Поэтому, требуется тесная взаимосвязь с учителем математики. Кроме того, крайне желательно, но не обязательно, чтобы учащиеся знали что такое алгоритм, исполнитель алгоритмов, модель.

Команды, рассматриваемые на данном занятии: forward, backward, turnright, turnleft, reset.

Примерное задание: Написать программу, выполнив которую «черепашка» нарисует букву «Н».

При выполнении задания важно помнить, что любая программа пишется на основании модели. Поэтому, учитель (на доске), вместе с учащимися (в тетради) изображает, букву «Н» в достаточно большом масштабе (для учащихся высота буквы не менее 4 см, и ширина не менее 2 см) и отмечает на ней все необходимые параметры. Желательно оговорить при этом, что используется масштаб 1 см - 10 (20, 30...) шагов «черепашки».

Модель:



На основании модели пишется алгоритм действий, которые потребуется выполнить исполнителю, обязательное учитывая его начальное положение. Поскольку, «черепашка» изначально «смотрит» вверх, то алгоритм действий будет следующим:

```
вперед  на  40 шагов
назад на 20 шагов
повернуть вправо на 90 градусов
вперед на 20 шагов
повернуть влево на 90 градусов
вперед на 20 шагов
назад на 40 шагов
```

Уже на основании алгоритма, рядом с ним или ниже, записывается программа, причем лучше командами записанными на английском языке. Сокращения команд лучше всего на данном этапе не использовать, поскольку они не несут той смысловой нагрузки, которую несут в себе слова (учитель английского языка будет Вам впоследствии благодарен):

алгоритм	программа	
	KDE 3	KDE 4
вперед на 40 шагов назад на 20 шагов повернуть вправо на 90 градусов вперед на 20 шагов повернуть влево на 90 градусов вперед на 20 шагов назад на 40 шагов	Forward 40 Backward 20 TurnRight 90 Forward 20 TurnLeft 90 Forward 20 Backward 40	forward 40 backward 20 turnright 90 forward 20 turnleft 90 forward 20 backward 40

После этого программа вносится учащимися в редакторе программ среды и запускается на выполнение. Очевидно, что данный тип программ для KDE 4 отличается от программы для KDE 3 только написанием команд — напомним, что для KDE 4 написание команды регистрозависимо. В дальнейшем, следует это помнить всегда.

Дополнительные задания: Написать программу, выполнив которую «черепашка» нарисует букву «П», «М», «А» - здесь, как и при выполнении задания с буквой «М», при построении модели потребуется помощь транспортира и линейки для определения величины углов и количества шагов «черепашки». Помните, мы оговаривали масштаб?

Примерное решение дополнительных заданий для KDE 3 (для KDE4.2 отличие будет только в написании команд):

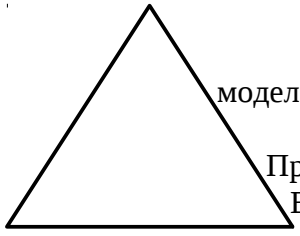
Буква «П»	Буква «М»	Буква «А»
Reset Forward 40 TurnRight 90 Forward 20 TurnRight 90 Forward 40	Reset Forward 40 TurnRight 135 Forward 20 TurnLeft 90 Forward 20 TurnRight 135 Forward 40	Reset TurnRight 45 Forward 55 TurnLeft 45 Backward 40 Forward 20 TurnLeft 90 Forward 20

Примечание. Для правильного построения букв «М» и «А» крайне желательно построить модели, с помощью которых можно измерить все требуемые углы и длину сторон букв.

Цикл.

В ходе данного занятия учащиеся должны познакомиться с командой Repeat, которая позволяет создать простейший цикл. Выполнить простейшее задание на написание программы с использованием циклической структуры.

Самый простой способ познакомить учащихся с циклическими структурами — дать им задание нарисовать равносторонний треугольник, квадрат и равносторонний шестиугольник.



1.Треугольник.

Написание программы, как и в прошлый раз, начинаем с построения модели.

При этом, указываем как внутренние углы треугольника, так и внешние. Выясняем, что для построение треугольника «черепашка» должна «смотреть» вправо, и поворачивать при переходе к новой стороне треугольника на 120 градусов, т.к. внешний угол треугольника равен разности между

величиной развернутого угла и внутреннего угла треугольника (180-60=120). А для «черепашки» именно внешний угол является главным при построении изображения.

Исходя из модели, пишем алгоритм:

```
очистить холст
повернуть вправо на 90 градусов
вперед на 40 шагов
повернуть влево на 120 градусов
вперед на 40 шагов
повернуть влево на 120 градусов
вперед на 40 шагов
```

В алгоритме видны повторяющиеся команды, которые легко заменить на цикл:

```
очистить холст
повернуть вправо на 90 градусов
повторять 3 раза
 начало цикла
   вперед на 40 шагов
   повернуть влево на 120 градусов
 конец цикла
```

На основе алгоритма пишем программу, оговаривая при этом, что знак «[» соответствует, в данном случае, указанию начала цикла, а знак «]» - указанию конца цикла (для KDE 4 это, соответственно, знаки «{» и «}»):

алгоритм	**программа**	
	KDE 3	**KDE 4**

очистить холст
повернуть вправо на 90 градусов
повторять 3 раза
начало цикла
вперед на 40 шагов
повернуть влево на 120 градусов
конец цикла
	Reset
	TurnRight 90
	Repeat 3
	[
	Forward 40
	TurnLeft 120
	]
		reset
		turnright 90
		repeat 3
		{
		forward 40
		turnleft 120
		}

2. Квадрат

Аналогично треугольнику пишем алгоритм и программу для квадрата:

алгоритм	**программа**	
	KDE 3	**KDE 4**

очистить холст
повернуть вправо на 90 градусов
повторять 4 раза
начало цикла
вперед на 40 шагов
повернуть влево на 90 градусов
конец цикла
	Reset
	TurnRight 90
	Repeat 4
	[
	Forward 40
	TurnLeft 90
	]
		reset
		turnright 90
		repeat 4
		{
		forward 40
		turnleft 90
		}

3. Шестиугольник

Для шестиугольника можно сравнить размеры программы с использованием цикла и

без его использования:

программа без использования цикла	программа с использованием цикла	
	KDE 3	KDE 4

Reset
TurnRight 90
Forward 40
TurnLeft 60
Forward 40
TurnLeft 60
Forward 40
TurnLeft 60
Forward 40
TurnLeft 60
Forward 40
TurnLeft 60
Forward 40
	Reset
	TurnRight 90
	Repeat 6
	[
	  Forward 40
	  TurnLeft 60
	]
		reset
		turnright 90
		repeat 6
		{
		  forward 40
		  turnleft 60
		}

Дополнительные задания: построить 8-ми угольник, 12-ти угольник
Примерное решение:

Восьмиугольник		Двенадцатиугольник	
KDE 3	KDE 4	KDE 3	KDE 4

Reset
TurnRight 90
Repeat 8
[
  Forward 40
  TurnLeft 45
]
	reset
	turnright 90
	repeat 8
	{
	  forward 40
	  turnleft 45
	}
		Reset
		TurnRight 90
		Repeat 12
		[
		  Forward 40
		  TurnLeft 30
		]
			reset
			turnright 90
			repeat 12
			{
			  forward 40
			  turnleft 30
			}

Переменная.

В ходе данного занятия учащиеся должны познакомиться с понятием переменной. Выполнить простейшее задание на написание программы с использованием циклической структуры и переменной.

1. Произвольные многоугольники

При выполнении заданий предыдущего занятия учащиеся должны были отметить следующую закономерность:

Количество углов в многоугольнике	3	4	6	8	12
Величина угла поворота «черепашки» (внешний угол многоугольника)	120	90	60	45	30

Исходя из нее, можно получить формулу, позволяющую рассчитывать величину внешнего угла многоугольника (математическую модель):

$$\textit{внешний угол} = \frac{360}{\textit{количество углов многоугольника}}$$

Т.е. для определения угла поворота «черепашки» можно использовать математическую формулу, исходя из известного количества углов многоугольника.

Теперь, введя понятие переменной, можно указать на возможность ее использования в данном случае:

алгоритм	**программа (KDE 3)**

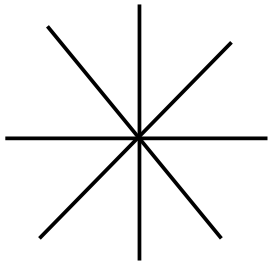
очистить холст
повернуть вправо на 90 градусов
количество углов = спросить «Сколько углов?»
угол поворота = 360 разделить на количество углов
повторять столько, сколько введено углов
 начало цикла
  вперед на 40 шагов
  повернуть влево на полученный угол поворота
 конец цикла
	Reset
	TurnRight 90
	N=InputWindow «Сколько углов?»
	A=360/N
	Repeat N
	[
	  Forward 40
	  TurnLeft A
	]

Та же программа, но в версии KDE 4 будет выглядеть следующим образом:

алгоритм	**программа (KDE 4)**

очистить холст
повернуть вправо на 90 градусов
количество углов = спросить «Сколько углов?»
угол поворота = 360 разделить на количество углов
повторять столько, сколько введено углов
 начало цикла
  вперед на 40 шагов
  повернуть влево на полученный угол поворота
 конец цикла
	reset
	turnright 90
	$N=ask «Сколько углов?»
	$A=360/$N
	repeat $N
	{
	  forward 40
	  turnleft $A
	}

2. Звезда



При построении изображения звезды, в первую очередь необходимо построить ее модель. Например для 8-ми лучевой, после чего определить углы поворота «черепашки». Для 8-ми лучевой они будут равны 45 градусов, для 12-ти лучевой - 30 градусов, для 6-ти лучевой - 60.

Таким образом становится видно, что и в данном случае можно использовать ранее полученную формулу определения угла поворота.

Получаем следующие алгоритм и программу:

алгоритм	**программа (KDE 3)**

очистить холст
количество углов = спросить «Сколько углов?»
угол поворота = 360 разделить на количество углов
повторять столько, сколько введено углов
 начало цикла
  вперед на 40 шагов
  назад на 40 шагов
  повернуть влево на полученный угол поворота
 конец цикла
	Reset
	N=InputWindow «Сколько углов?»
	A=360/N
	Repeat N
	[
	  Forward 40
	  Backward 40
	  TurnLeft A
	]

Или:

алгоритм	**программа (KDE 4)**

очистить холст
количество углов = спросить «Сколько углов?»
угол поворота = 360 разделить на количество углов
повторять столько, сколько введено углов
начало цикла
вперед на 40 шагов
назад на 40 шагов
повернуть влево на полученный угол поворота
конец цикла
	reset
	$N=ask «Сколько углов?»
	$A=360/$N
	Repeat $N
	{
	forward 40
	backward 40
	turnleft $A
	}

Дополнительные задания: построить правильный многоугольник (звезду) длина сторон (лучей) которого вводится с клавиатуры так же как и число углов.

Примерное решение:

программа (KDE 3)	**программа (KDE 4)**

Reset
N=InputWindow «Сколько углов?»
L=InputWindow «Чему равна длина лучей?»
A=360/N
Repeat N
[
Forward 40
Backward L
TurnLeft A
]
	reset
	$N=ask «Сколько углов?»
	$L=ask «Чему равна длина лучей?»
	$A=360/$N
	repeat $N
	{
	forward 40
	backward $L
	turnleft $A
	}

Условное ветвление. Подпрограммы.

На данном занятии учащиеся должны познакомиться с понятием условного ветвления, подпрограммы, псевдослучайными числами.

Задание: Написать программу, выполнив которую «черепашка» сможет построить звезду двух типов: с одинаковой длиной и разной длиной лучей. Тип звезды указывает пользователь при помощи чисел: 1 — звезда с одинаковой длиной лучей, 0 — звезда с разной длиной лучей.

Для этого можно использовать модифицированную программу «Звезда», составленную на прошлом занятии, добавив в нее генератор псевдослучайных чисел и ввод типа звезды с клавиатуры.

программа (KDE 3)	**программа (KDE 4)**

Learn Zvezda1 N,X
[
A=360/N
Repeat N
[
Forward X
Backward X
TurnLeft A
]
]
Learn Zvezda0 N
	learn Zvezda1 $N,$X
	{
	$A=360/$N
	repeat $N
	{
	forward $X
	backward $X
	turnleft $A
	}
	}
	learn Zvezda0 $N

```
[
 A=360/N
 X=Random 20,40
 Repeat N
  [
    Forward X
    Backward X
    TurnLeft A
  ]
]

Reset
C=InputWindow «Какого типа звезда - 1 или 0?»
K=InputWindow «Сколько углов?»
If  C==1
[
 L=InputWindow «Какая длина лучей?»
 Zvezda1 N,X
]
If  C==0
[
 Zvezda0 N
]
```

```
{
 $A=360/$N
 $X=random 20,40
 repeat $N
  {
    forward $X
    backward $X
    turnleft $A
  }
}

reset
$C=ask «Какого типа звезда - 1 или 0?»
$K=ask «Сколько углов?»
if  $C==1
{
 $L=ask «Какая длина лучей?»
 Zvezda1 $K, $L
}
if  $C==0
{
 Zvezda0 $K
}
```

***Дополнительные задания*:**

1. Написать программу, выполнив которую «черепашка» сможет построить один из четырех типов звезды: 0 — с разной длиной лучей, 1 — с одинаковой длиной лучей, 2 — с увеличивающейся длиной лучей (каждый следующий луч длиннее предыдущего на 5 шагов), 3 — с уменьшающейся длиной лучей (каждый следующий луч короче предыдущего на 5 шагов).

Примерное решение:

программа (KDE 3)

```
Learn Zvezda1 N,X
[
 A=360/N
 Repeat N
  [
    Forward X
    Backward X
    TurnLeft A
  ]
]
Learn Zvezda0 N
[
 A=360/N
 X=Random 20,40
 Repeat N
  [
    Forward X
```

программа (KDE 4)

```
learn Zvezda1 $N,$X
{
 $A=360/$N
 repeat $N
  {
    forward $X
    backward $X
    turnleft $A
  }
}
learn Zvezda0 $N
{
 $A=360/$N
 $X=random 20,40
 repeat $N
  {
    forward $X
```

```
    Backward X
    TurnLeft A
  ]
]

Learn Zvezda2 N,X
[
 A=360/N
 Repeat N
  [
    Forward X
    Backward X
    X=X+5
    TurnLeft A
  ]
]

Learn Zvezda3 N,X
[
 A=360/N
 Repeat N
  [
    Forward X
    Backward X
    X=X-5
    TurnLeft A
  ]
]

Reset
C=InputWindow «Какого типа звезда – 0, 1, 2, 3?»
N=InputWindow «Сколько углов?»
If  C==1
[
  X=InputWindow «Какая длина лучей?»
  Zvezda1 N,X
]


If  C==2
[
  X=InputWindow «Какая длина лучей?»
  Zvezda2 N,X
]

If  C==3
[
  X=InputWindow «Какая длина лучей?»
  Zvezda3 N,X
]
```

```
    backward $X
    turnleft $A
  }
}

learn Zvezda2 $N,$X
{
 $A=360/$N
 repeat $N
  {
    forward $X
    backward $X
    $X=$X+5
    turnleft $A
  }
}

learn Zvezda3 $N,$X
{
 $A=360/$N
 repeat $N
  {
    forward $X
    backward $X
    $X=$X-5
    turnleft $A
  }
}

reset
$C=ask «Какого типа звезда – 0, 1, 2, 3?»
$K=ask «Сколько углов?»
if  $C==1
{
  $L= ask «Какая длина лучей?»
  Zvezda1 $K, $L
}


if  $C==2
{
  $L=ask «Какая длина лучей?»
  Zvezda2 $K, $L
}

if  $C==3
{
  $L=ask «Какая длина лучей?»
  Zvezda3 $K, $L
}
```

If C==0
[
Zvezda0 N
]
	if $C==0
	{
	Zvezda0 $K
	}

2. Написать программу, выполнив которую, черепашка построит окружность радиусом R и центром, расположенным в точке с координатами (x,y). Примечание: окружностью можно считать правильный 24-х угольник

Для построения окружности радиусом R необходимо связать величину радиуса и длину стороны 24-х угольника, построив математическую модель:

$2\pi R = 24L$ - длина окружности равна периметру 24-х угольника, следовательно длину стороны многоугольника можно будет найти по формуле:

$$L = \frac{(2\pi R)}{24}$$

Значит, программа построения окружности будет выглядеть следующим образом:

программа (KDE 3)	**программа (KDE 4)**

Learn Circle x,y,R
[
L=2*3.14*R/24
Go x,y
PenUp
TurnLeft 90
Forward R
TurnRight 90
PenDown
Repeat 24
[
Forward L
TurnRight 15
]
PenUp
TurnLeft 90
Backward R
TurnRight 90
PenDown
]
 	
Reset
x1=inputwindow "Чему равно x?"
y1=inputwindow "Чему равен y?"
R1=inputwindow "Чему равен радиус?"
Circle x1,y1,R1
	learn Circle $x,$y,$R
	{
	$L=2*3.14*$R/24
	go $x,$y
	penup
	turnleft 90
	forward $R
	turnright 90
	pendown
	repeat 24
	{
	forward $L
	turnright 15
	}
	penup
	turnleft 90
	backward $R
	turnright 90
	pendown
	}
	
	reset
	$x1 = ask "Чему равно x?"
	$y1 = ask "Чему равен y?"
	$R1 = ask "Чему равен радиус?"
	Circle $x1,$y1,$R1

3. Написать программу, выполнив которую, черепашка построит половину окружности радиусом R и центром, расположенным в точке с координатами (x,y). Примечание: окружностью можно считать правильный 24-х угольник

Для построения полуокружности можно воспользоваться полученной ранее программой, модифицировав ее следующим образом:

программа (KDE 3)	**программа (KDE 4)**

Learn Circle2 x,y,R
[
  L=2*3.14*R/24
  Go x,y
  PenUp
  TurnLeft 90
  Forward R
  TurnRight 90
  PenDown
  Forward L/2
  TurnRight 15
  Repeat **11**
  [
  Forward L
  TurnRight 15
  ]
  Forward L/2
  PenUp
  TurnRight 90
  Backward R
  TurnLeft 90
  PenDown
]
 	
Reset
x1=inputwindow "Чему равно x?"
y1=inputwindow "Чему равен y?"
R1=inputwindow "Чему равен радиус?"
Circle2 x1,y1,R1
	learn Circle2 $x,$y,$R
	{
	  $L=2*3.14*$R/24
	  go $x,$y
	  penup
	  turnleft 90
	  forward $R
	  turnright 90
	  pendown
	  forward $L/2
	  turnright 15
	  repeat **11**
	  {
	  forward $L
	  turnright 15
	  }
	  forward $L/2
	  penup
	  turnright 90
	  backward $R
	  turnleft 90
	  pendown
	}
	
	reset
	$x1 = ask "Чему равно x?"
	$y1 = ask "Чему равен y?"
	$R1 = ask "Чему равен радиус?"
	Circle2 $x1,$y1,$R1

Решения дополнительных заданий 2 и 3 можно использовать при решении задач №69, 77, 78, 79, 80, 82 из [1]

Рекомендации.

В данном пособии достаточно сложно рассмотреть все возможные задания и темы, в которых можно использовать среду kTurtle, поскольку круг этих тем достаточно широк и охватывает как алгоритмизацию и программирование, так и моделирование. Поэтому можно рекомендовать воспользоваться тем богатейшим материалом, который предлагают учителю различные авторы. Максимум что потребуется сделать — провести адаптацию заданий к среде kTurtle, которая, чаще всего проходит довольно просто. Наиболее близким, к рассмотренной среде, пакетом является пакет ЛогоМиры. Отличие этих сред, пожалуй, только в том, что kTurtle это среда именно программирования, в отличии от ЛогоМиров, в которых можно выполнить несколько больше операций.

Рекомендованная литература:

1. Залогова Л.А., под ред. Семакина И.Г. Информатика. Задачник-практикум. т. 1 М.: «Бином. Лаборатория знаний», 2005 г. стр. 195-203
2. Макарова Н.В. Информатика и ИКТ {Начальный уровень]. , Спб.: ООО «Питер-пресс», 2008 г.
3. Справка программы kTurtle
4. WiKi