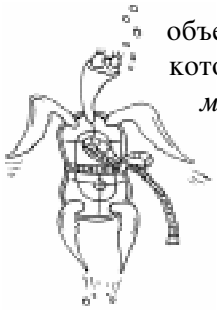
Кузнецова Ирина Николаевна

МОДЕЛИРОВАНИЕ В СРЕДЕ ЛОГО
ЗАНЯТИЕ 2. АЛГОРИТМЫ ОБРАБОТКИ

Мы познакомились со встроенными объектами среды Лого. Изучение любого языка программирования подразумевает знакомство со встроенными объектами этого языка.



Мы узнали, что любой объект обладает параметрами, которые можно *измерять* и *изменять*. Для каждого объекта существуют действия (в программировании они называются – *методы*), которые как раз и меняют значения параметров.

Некоторые действия в среде Лого выполняются с помощью визуальных инструментов (кнопок). Все возможные действия задаются командами языка программирования. Язык программирования позволяет описывать и новые действия, и новые объекты.

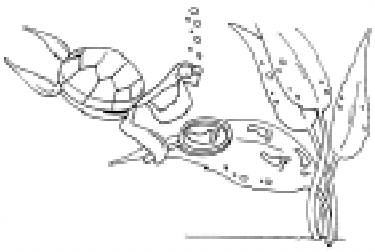
Однако прежде чем мы перейдем к таким высотам использования языка, следует разобраться с алгоритмами.

Зачем нам нужны программы? Обычно, работая на компьютере, мы ставим себе какую-то цель. Достичь этой цели можно, выполняя команды в определенной последовательности. Человек, естественно, ошибается и хотел бы повторить некоторые действия сначала. Писать их или даже просто выполнять из командного центра в нужной последовательности – довольно утомительно, что приводит к новым ошибкам.

Программа – это последовательность команд (действий), которая имеет уникальное имя и которую можно запускать, указывая это имя.

Оказывается, можно читать текст программы, как математическую формулу, и понимать, что будет происходить в компьютере при ее выполнении.

Анализ текста программы, поиск ошибок в программе при неправильной ее работе напоминает работу сыщика.

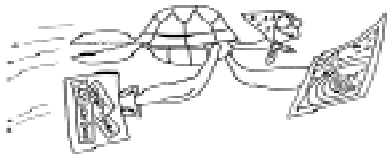


Мы воспользуемся достижениями науки и не будем набивать себе шишки бессистемным программированием. Мы познакомимся с основными типами алгоритмов, научимся их записывать на языке программирования. Затем более сложные алгоритмы уже будем конструировать из этих базовых. Научно говоря, мы будем осуществлять структурное и процедурное программирование.

Итак, основных типов алгоритмов всего три: линейный, циклический и разветвляющийся. Не забудем и про вспомогательный алгоритм.

ЗАДАНИЕ 1.
Линейный алгоритм.
Вспомогательный алгоритм

Рассмотрим задачу про черепашку-путешественницу (см. рисунок 1).



Черепашка `t1` живет в Санкт-Петербурге. Она хочет посетить все столицы Европы, рисуя свой путь линией случайного цвета. Столицы отмечены на карте. Выберите для черепашки маршрут путешествия. В каждой столице находится туроператор (расположите там черепашку с именем `tour`),

Рисунок 1.

который помогает определить расстояние и направление до этого города. Для измерения расстояния между городами используется датчик *distance*. Для поворота к нужному городу имеется команда **towards**.

Вы можете сами выбрать подходящий маршрут, а черепашке поручить нарисовать его.

Допустим, маршрут такой: Санкт-Петербург – Лондон – Париж – Берн.

Напишем процедуру «конструктор» для размещения на карте черепашек **tour** и **t1**. Назначим **tour** действия для измерения и вывода в командный центр координат выбранных городов.

Для удобства напишем процедуру следования черепашки за курсором мыши и назначим ее кнопке *bmouse*. Не забудем написать и процедуру «деструктор».

Получились такие программы:

```
to constr
newturtle "t1 st pu setc 95 home
setsh 3 stamp setsh 0
t1, setinstruction [eurotour]
newturtle "tour st pu fd 20 setc 14
tour, setinstruction [show pos]
newbutton "bmouse [-183 -170] [за_мышью]
end

to за_мышью
tour, forever [setpos mousepos]
end

to destr
cg
remove "t1
remove "tour
remove "bmouse
end
```

Все действия в этих программах выполняются один раз строго в том порядке, в каком они написаны.

Линейный (последовательный) алгоритм – описание действий, которые выполняются однократно в заданном порядке.

Заданный порядок определяется нашими правилами чтения текста – слева направо и сверху вниз.

Как вызывать написанные программы? Имя программы стало новой командой. Можно написать имя программы в командном центре и ввести его как команду. Можно назначить для выполнения при щелчке на кнопку (см. кнопку *bmouse*).

Щелкните на кнопку *за_мышью*.

Черепашка **tour** «приклеется» к указателю мыши. Наведите указатель на столицу и щелкните. После щелка на черепашку в командном центре будут появляться координаты местоположения выбранных городов.

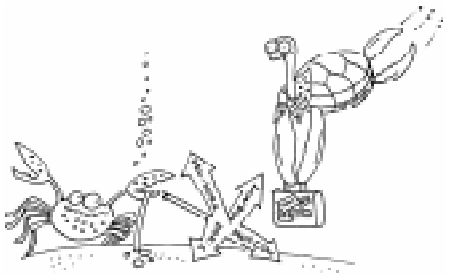
Допустим, вы получили такие данные:

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

Осталось написать программу для черепашки **t1**, используя выведенные данные.

```
to eurotour
setpos [62 94]
setpensize 5 pd
setpos [-53 -32]
setpos [-90 -36]
setpos [-98 31]
setpos [62 94]
pu
end
```

Конечно, вы могли перепутать последовательность городов, захотели изменить



маршрут, а на карте уже нарисованы линии, которые будут мешать при следующем запуске программы **eurotour**.

Для возобновления состояния карты и других объектов и служит программа **constr**. Правда, перед ее запуском надо удалить созданные ранее объекты. Для этого служит программа **destr**. Для удобства пользования этими двумя программами напишем новую программу:

```
to again
destr
constr
end
```

Мы видим, что из этой программы вызываются две программы, написанные ранее. В этом случае они помогают быстро восстановить карту.

Вспомогательный алгоритм – алгоритм, который можно использовать в других алгоритмах, указав только его имя.

Вспомогательная программа (процедура) – программа, которая вызывается из других программ.

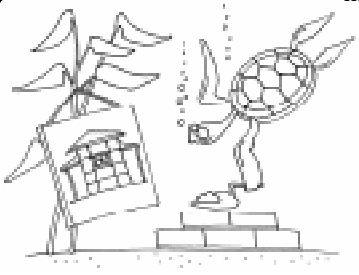
Очевидно, что для проверки и выполнения программы необходимо, чтобы текст ее был понятен не только компьютеру, но и человеку.

Правила:

– один алгоритм должен помещаться на одной странице;

– у каждого алгоритма только один вход и только один выход;

– сложный алгоритм составляется из более простых вспомогательных алгоритмов.



Процедуры – это кирпичики, из которых можно собирать более сложные программы.

Программный модуль – программа, из которой вызываются процедуры.

Составьте процедуры для рисования цифр почтового кода (задание 7.15 из [2]).

Напишите программу-модуль для рисования своего почтового кода, используя эти процедуры.

ЗАДАНИЕ 2.
Разветвление

В задании 1 вы сами назначали маршрут путешествия, выбирая его из разных возможных вариантов. Человек часто стоит перед выбором – выбором действия, предмета и т. д. В русском языке выбор записывается с помощью *условного* предложения, которое имеет следующую структуру:

если <условие> то <действие1 > иначе <действие2>

Что такое условие?

Условие – такой вопрос, на который можно ответить только «Да» или «Нет»

Выбор можно поручить и компьютеру, если суметь записать условное предложение на языке программирования.

Для этого служит алгоритмическая конструкция, которая обеспечивает выбор одного из двух возможных списков действий в зависимости от значения условия.

Алгоритм выбора – алгоритм, в котором выбирается одно или другое действие в зависимости от условия.

Условие задается операцией сравнения или более сложным логическим выражением и может быть истинным (значение ‘да’) или ложным (значение ‘нет’).

Разветвление может быть *полным* и *неполным*. Соответственно, в языке Лого имеются две команды:

для полного разветвления

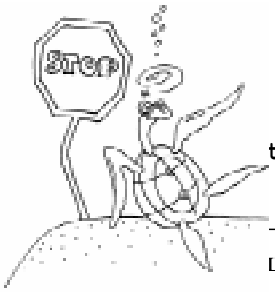
```
ifelse <условие> [<список действий 1>]
[<список действий 2>]
```

и
для неполного разветвления, в котором отсутствует список действий для ложного значения условия

```
if <условие> [<список действий 1>]
```

Выбор из нескольких вариантов

Команду **if** можно использовать для организации выбора одного списка действий из нескольких возможных – так называемой конструкции *case*. Если все условия ложны, выполнение процедуры может заканчиваться, даже если этот выбор включен в цикл. Останов текущей процедуры – команда **stop**.




```
if <условие> [<список действий 1>]
if <условие> [<список действий 2>]
if <условие> [<список действий 3>]
...
if <условие> [stop]
```

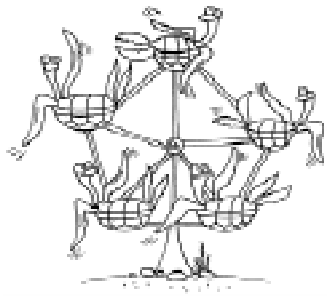
В качестве примера разобрем задачу управления объектом с помощью клавиатуры. Допустим, мы хотим управлять черепашкой **tour** через клавиши управления курсором. Код нажатой клавиши считывается и вычисляется датчиками *readchar* и *ascii*:

Выбор действия, соответствующего нажатой клавише, осуществляется по следующему алгоритму:

```
to actcase
tour,
if (ascii readchar) = 38 [seth 0 fd 10]
                  ;нажата клавиша «вверх»
if ( ascii readchar) = 40 [seth 180 fd 10]
                  ;нажата клавиша «вниз»
if ( ascii readchar) = 39 [seth 90 fd 10 ]
                  ;нажата клавиша «вправо»
if ( ascii readchar) = 37 [seth 270 fd 10 ]
                  ;нажата клавиша «влево»
if ( ascii readchar) = 27 [stop ]
                  ;нажата клавиша «ESC»
end
```

Заметим, что датчик *readchar* начинает работать после щелчка мышью в любом месте Рабочего поля.

ЗАДАНИЕ 3.
Циклический алгоритм



Рассмотрим текст программы **за_мышью**.

В теле программы всего одна строчка:

```
tour, forever [setpos mousepos]
```

Первое слово называет имя объекта – **tour**,, который будет выполнять следующие команды.

В квадратных скобках – команда установки объекта **tour** на место положения указателя мыши: **setpos mousepos**.

Что означает слово **forever**?

В переводе на русский язык это значит «всегда».

На языке Лого команда **forever** или **всегда** показывает, что список команд, заключенный в квадратные скобки, надо выполнять много раз.

Циклический алгоритм – описание действий, которые должны повторяться в указанном порядке (как записано) указанное число раз или пока не выполнится заданное условие.

Известны три вида циклов:

– перечисляемый (с заранее известным числом повторений),

– с предусловием,

– с постусловием.

Перечисляемый цикл

В Лого такой цикл можно организовать разными способами:

1. Команда **repeat** обеспечивает повторение указанное число раз списка команд.

```
repeat <число повторений>
                    [<повторяемые действия>]
```

Вот алгоритм рисования цветного квадрата:

```
to sqc
pd
repeat 4[fd 30 rt 90]
rt 45 pu fd 30 / 2 pd fill bk 30 / 2 lt 45
end
```

2. Команда **dolist** организует повторение указанных действий для каждого значения заданной переменной.

```
dolist [<имя переменной>
        [<список значений этой переменной>]]
                              [<список повторяемых
                                    действий>]
```

Предположим, надо нарисовать 5 квадратов, каждый из которых имеет определенный цвет (см рисунок 2). Цвета квадратов не связаны никакой зависимостью. Команда **dolist** будет содержать список значений этих цветов.

Ряд из квадратов:

```
to row
dolist [col [15 45 65 75 95]]
                          [setc :col sqc fd 30]
end
```

3. Команда **dotimes** обеспечивает повторение указанных действий для каждого значения из диапазона указанной числовой переменной (минимальное значение этой переменной – 0).

```
dotimes [<имя переменной >
                  <верхнее значение диапазона>]
              [<список повторяемых действий >]
```

Если запустить программу **clock**, вы увидите на Рабочем поле секундную стрелку (см. рисунок 3):

```
to clock
seth 0
dotimes [k 360] [setc 9 arrow wait 1 setc 0
arrow seth :k]
setc 9 arrow
st
end
```

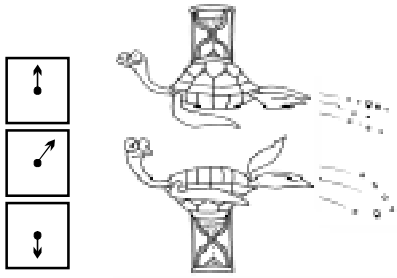


Рисунок 3.

Рисунок 2.

Вспомогательный алгоритм рисования стрелки:

```
to arrow
ht
fd 100
lt 45 bk 20 fd 20 rt 45
rt 45 bk 20 fd 20 lt 45
bk 100
end
```

Цикл с постусловием

На языке Лого для цикла с постусловием нет специальной команды, поэтому приходится организовывать его с помощью других команд:

```
repeat <max число повторений>
[ddd1 if <условие окончания цикла> [stopme]]
to ddd1
<повторяемые действия>
end
```

Рассмотрим пример.

Как установить защиту своего проекта? Можно защитить его паролем. Напишем программу, которая вызывается сразу после загрузки проекта и запрашивает пароль. Дается три попытки для ввода пароля. Если все-таки он оказался неверным, программа запрещает работу с проектом (см. рисунок 4).

```
to startup
repeat 3 [;запрашиваем и вводим пароль
 question [Введи пароль]
 if answer = 123456
 [announce [Привет, хозяин!] stopme]
 ]
;если пароль неверный, экран чернеет,
появляется сообщение о взломе и звучит колокол
if not answer = 123456
```

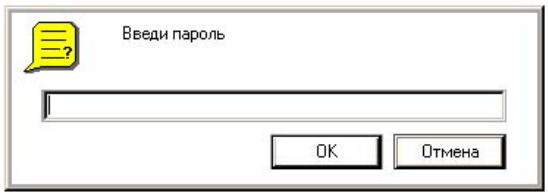


Рисунок 4.

```
 [presentationmode setbg 9
 announce [Это взломщик!!! Проект отключен]
 forever [note 60 3]]
end
```

Цикл с предусловием

На языке Лого для цикла с предусловием также нет специальной команды, поэтому приходится организовывать его с помощью других команд:

```
repeat <max число повторений>
            [ifelse <условие> [ddd][stop]]
to ddd
<повторяемые действия>
end
```

Рассмотрим в качестве примера классическую задачу поиска чисел с заданным признаком в заданном интервале.

Нужно вывести в текстовое окно *N* чисел натурального ряда, которые делятся на 3 (см. рисунок 5).

```
to startnum
newtext mtxt1 [-100 100] [50 100]]
;подготовка данных в переменных
local [N k i]
make "k 1 ;текущее число
make "i 0 ;счетчик чисел
question [введи число N]
make "N answer ;количество чисел
repeat 1000 [ifelse :i < :N
 [if (remainder :k 3) = 0
                    [pr :k make «i :i + 1]
 make "k :k + 1]
 [stop]
 ]
end
```

Вы видите, что для организации циклов в большинстве случаев требуется некоторые данные хранить и изменять в памяти. В следующей статье будет рассказано, как это делать.

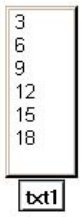


Рисунок 5.

Приведены процедуры для рисования фрагментов на основе правильных многоугольников:

Половина окружности

```
1    to arc1rt180
repeat 180[fd 1 rt 360 / 360]   end
2    to arc1lt180
repeat 180[fd 1 lt 360 / 360]   end
3    to arc2rt180
repeat 180[fd 2 rt 360 / 360]   end
4    to arc2lt180
repeat 180[fd 2 lt 360 / 360]   end
```

Четверть окружности

```
1    to arc1rt90
repeat 90[fd 1 rt 360 / 360]    end
2    to arc1lt90
repeat 90[fd 1 lt 360 / 360]    end
3    to arc2rt90
repeat 90[fd 2 rt 360 / 360]    end
4    to arc2lt90
repeat 90[fd 2 lt 360 / 360]    end
```

Исследуйте, какая фигура получится при выполнении следующих программ:

```
1    to l1
arc1rt90 rt 90 arc1rt90 rt 90   end

2    to fl1
repeat 10[l1 rt 360 / 10]       end

3    to l2
arc1lt90 arc1rt90    rt 180 arc2lt90
lt 90                           end

4    to fl2
repeat 10[l2 rt 360 / 10]       end
```


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