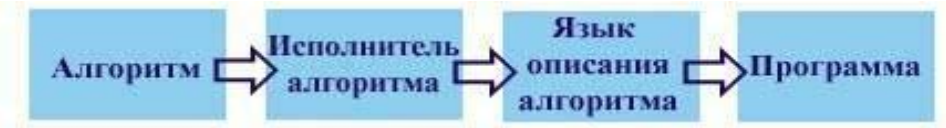
Бородатый заявил: "Лично я вижу в этом перст судьбы – шли по лесу и встретили программиста. Мне кажется, вы обречены."
— "Вам действительно так нужен программист?" — спросил я. "Нам позарез нужен программист".
(А.Стругацкий, Б.Стругацкий "Понедельник начинается в субботу")

Глава 1
В которой обсуждается, что такое программа и что означает программировать, намечается путь от алгоритма до программы и выясняется, что общение с исполнителем алгоритма на человеческом языке может привести к неожиданным и нежелательным последствиям.

Программист, программа, программируемый, запрограммированный... Все эти слова вам давно известны, но хорошо ли вы понимаете их смысл? Попробуйте объяснить самому себе, что такое программа.

Итак, программа — это...

Что-то не складывается в определении? Возникли сомнения и неясности? Это не удивительно, ведь вопрос совсем не простой, но нам необходимо в нем разобраться прежде всего. Приступая к изучению языка программирования Logo, на котором мы уже в скором времени начнем писать разнообразные программы, надо ясно представлять, чем же нам предстоит заниматься.
Для начала наметим маршрут:



1.1 Алгоритм и исполнитель

В прошлом году мы с вами говорили об алгоритмах и их исполнителях.

Алгоритм — это четко определенная последовательность действий, выполнение которых приводит к решению поставленной задачи.

Того, кто (или то, что) выполняет алгоритм, мы назвали ***исполнителем алгоритма***.

Команды, которые может выполнить конкретный исполнитель, образуют ***систему команд исполнителя (СКИ)***.

Вспомним примеры алгоритмов и исполнителей.

В дом привезли новый шкаф... То есть, шкафа как такового еще нет, на полу разложены створки, полки, шурупы и прочие детали будущего вместилища одежды и белья. Вы с отцом, следуя подробной инструкции, приступаете к сборке. Здесь *инструкция* выступает в роли *алгоритма*, а *вы с отцом* — его *исполнителей*.

На уроках математики вы выполняете разные вычисления — умножаете и делите столбиком, складываете простые дроби. В этих случаях вы являетесь исполнителями соответствующих алгоритмов.

Но исполнителем может быть не только человек. Разнообразные устройства, в том числе и компьютер, также могут выполнять заданные им алгоритмы. Например, "Луноход" — самоходный автоматический аппарат, доставленный на Луну в 1970 году, выполнял сложнейшие алгоритмы, перемещаясь по лунной поверхности и собирая необходимую людям информацию. Промышленные роботы заменяют людей на производстве, в быту на помощь хозяйкам также приходят устройства, способные действовать по заданным алгоритмам. Подумайте и приведите примеры таких устройств, используемых в вашем доме.

1.2 Формальный и неформальный исполнитель

— Вы не знаете, как пройти в библиотеку?
— Знаю.
И ушел...

В отличие от устройств, человек является неформальным исполнителем алгоритма. Что это означает?

Во-первых, человек не выполняет алгоритм бездумно и формально, соображения здравого смысла часто берут верх, какими бы строгими не были инструкции. Если ему скажут, идите прямо до второго перекрестка и никуда не сворачивайте, то вряд ли он станет наступать на встречных прохожих и опрокидывать попадающиеся на пути предметы.

Во-вторых, человек, даже если он еще мал, обладает определенным жизненным опытом и чем больше этот опыт, тем менее подробным может быть алгоритм. Взрослому человеку достаточно сказать: “Завари чай”, не объясняя до мельчайших подробностей, как это следует делать.

В-третьих, неформальному исполнителю, как правило, известна конечная цель и он к ней стремится, порою уточняя и дополняя алгоритм. Представим себе, что в алгоритме "Доберись до библиотеки" есть такая конечная инструкция:

— На всех перекрестках поворачивай направо, пока не увидишь здание библиотеки.

Вряд ли, увидев вдали здание библиотеки, исполнитель-человек остановится, как вкопанный.

Формальный исполнитель не обладает ни жизненным опытом, ни здравым смыслом, он не стремится к конечному результату и все инструкции алгоритма выполняет буквально, так, как они записаны в алгоритме. Поэтому такого исполнителя мы будем называть бездумным исполнителем или БИ. И если случится так, что наш БИ будет вести себя неразумно, выполнять что-то, не отвечающее нашим намерениям, то винить все-таки придется нам самих себя и искать неточности и ошибки в алгоритме, ведь БИ повинуется не нашим намерениям, а нашим инструкциям.

1.3 Язык описания алгоритма и программа

Исполнитель сможет выполнить алгоритм, если он ему известен, если алгоритм ему сообщили**.**

Для людей важнейшим способом общения является язык. Попадая в чужую страну и не зная национального языка, человек оказывается совершенно беспомощным. На выручку может придти язык жестов, мимика, письмо с помощью рисунков (пиктографическое письмо), но все это только частично улучшает ситуацию.

Естественный язык (украинский, русский, английский, французский, ...) — основа основ полноценного общения людей.

Естественные языки образны и многозначны. Если заглянуть в толковый словарь русского языка, то можно узнать, например, что существует более 20 значений слова "идти". Вот только несколько примеров:

человек идет по дороге;
идет дождь;
время идет;
ей идет это платье;
опята пойдут позже, в сентябре;
давай, сходим завтра на рыбалку? — идет!

В естественном языке совершенно разные понятия могут обозначаться одним и тем же словом. Как правило, человек *из контекста (общего смысла текста),* порою даже не задумываясь, из всего множества значений слова выделяет именно то, которое имел в виду отправитель сообщения. Но представьте себя на месте формального исполнителя, не вникающего в смысл всего сообщения. Как в этом случае вы будете понимать словосочетания: кислая мина; ранний побег; ели везде; знакомая среда?

Те, кто пользовался когда-нибудь электронными переводчиками, знает, какие неожиданные и далекие от смысла оригинала можно получить переводы. Так, например, наш формальный переводчик перевел с английского языка текст, рассказывающий о... Попробуйте сами догадаться, о чем идет речь.

Деревянные сделки сегодня — это умирала-вырезка от сосновых фанеры, затем опустился в жидкие химикалии, которые производят легко зажженный, погашаемый совет.

Теряетесь в догадках? А речь идет о простой деревянной спичке, но как было объяснить переводчику, что из всех значений слова "match" надо было выбрать не "сделка", а "спичка", из значений слова "tip" — "кончик", а не "совет", что "die" означает не только "умереть", но и "штамповать", не говоря уже о сложностях грамматических конструкций?

Для формального исполнителя язык общения не может быть многозначным, для таких исполнителей разрабатывают и используют специальные искусственные языки, где отдельные слова и выражения не допускают различных толкований.

Алгоритм, описанный на языке исполнителя, называется ***программой***.

Программирование — это процесс перевода алгоритма на язык конкретного исполнителя.

Так как разные исполнители владеют разными языками, а один и тот же алгоритм могут выполнять разные исполнители, то на основе одного алгоритма могут быть написаны разные программы. Мы с вами будем изучать язык программирования Logo, но вы, наверное, знаете, что существует много других языков программирования для управления компьютером - Pascal (Паскаль), C++, Basic (Бейсик), Java и другие. Чтобы научиться писать программы на том или другом языке, нужно изучить алфавит, словарь и грамматические правила, по которым строятся предложения в этом языке. При этом не допускаются никакие отклонения от правил написания слов и предложений, иначе исполнитель просто откажется выполнять ваши инструкции и не станет недоумевать и переживать за ошибки, как это делает приятель Мишки из стихотворения А.Шибаева:

Пришло письмишко мне,
Гляжу —
Из лагеря от Мишки...
Здесь чудный лук и я лижу, -
Написано в письмишке.
Лук лижет? Что за чудеса?
Наверно, шутит плут...

Читаю дальше: *Здесь лиса,*
красивый длинный прут...
На днях в лесу нашел я грусть
и очень был доволен...
Нет, нет, не шутит он! Боюсь,
Мой друг серьезно болен.
Вернется — надо подлечить:
Заставить правила учить…

Вопросы и задания:

1. Что называется алгоритмом?
2. Приведите примеры алгоритмов, которые вам приходится выполнять в вашей повседневной жизни.
3. С какими исполнителями алгоритмов вы познакомились на уроках информатики в начальной школе?
4. Придумайте систему команд для исполнителя, который может сварить картофель в “мундире”. Дайте ему имя. Запишите алгоритм варки в “мундире” пяти картофелин.

Глава 2
В которой появляется Черепашка из компьютерного микромира KTurtle и она знакомит нас со своей средой обитания и преподает первые уроки языка Logo.

2.1 Первая встреча

Первые шаги в изучении искусства программирования нам поможет сделать Черепашка KTurtle. Она многое умеет делать — рисовать и перемещаться по экрану, поворачиваться в разные стороны и делать простые и сложные вычисления, задавать вопросы и отвечать на них. Черепашка станет исполнителем наших команд и алгоритмов, и, как мы уже говорили, для того, чтобы наши замыслы стали явью, надо изучить язык Черепашки — язык Logo. Это означает, что мы должны изучить словарь, правила записи отдельных слов и выражений и правила записи программы. Но делать это мы будем постепенно, также как, изучая английский язык, мы сначала запоминаем небольшой запас новых слов, учим правила построения простых предложений и постепенно расширяем свой словарный запас и знакомимся с более сложными грамматическими конструкциями.

Из всех профессий Черепашки для начала мы веберем профессию художника-графика.

Отправляемся в мастерскую художника, для чего запускаем программу **KTurtle** и попадаем в среду обитания нашей Черепашки (Рис 1).

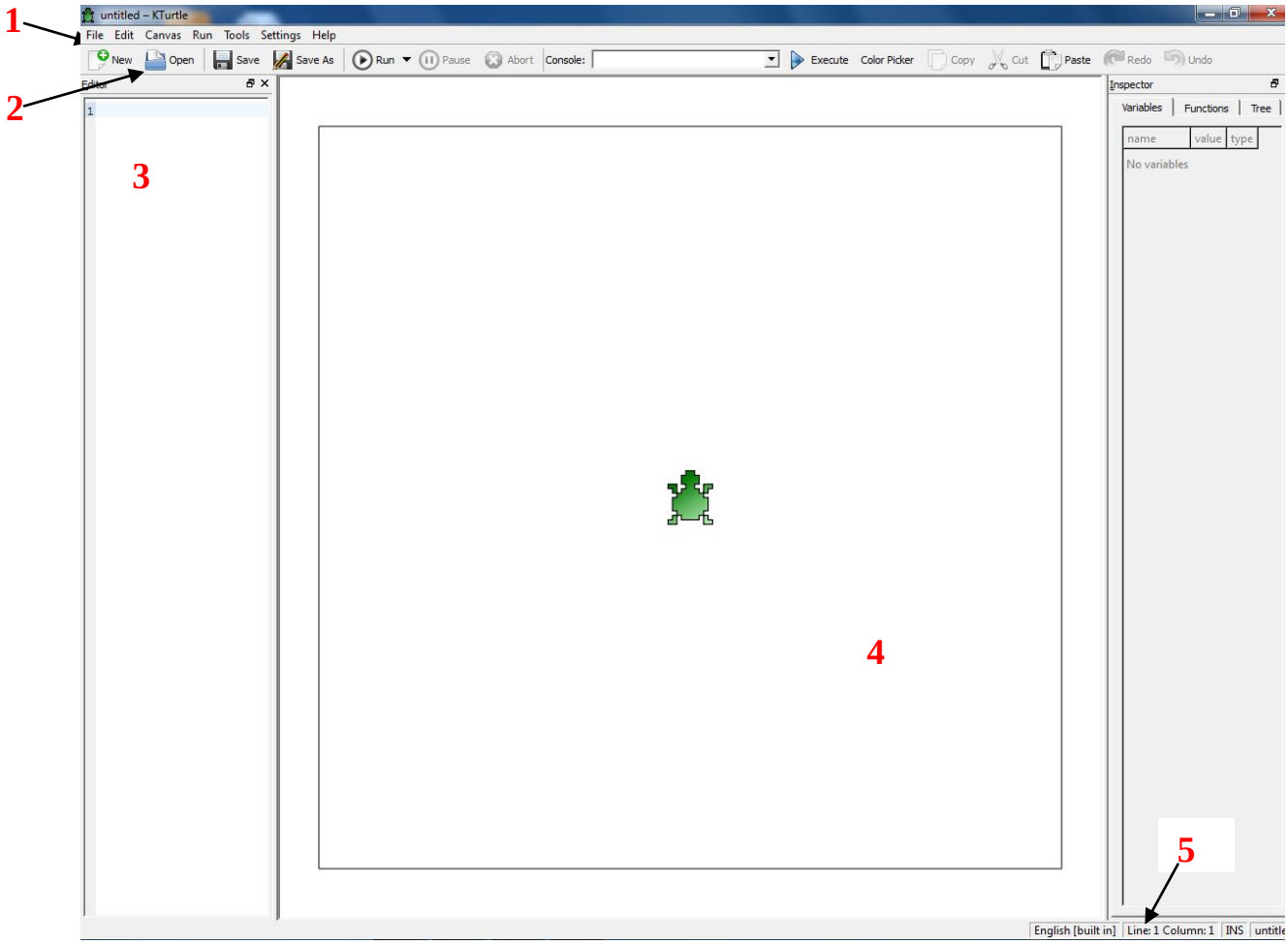


Рис. 1

Как и любое приложение, наша программа работает в окне, которое можно закрывать, перемещать по экрану монитора, сворачивать в иконку, увеличивать до размера всего экрана или придавать ему произвольный размер. Самая большая часть окна — это **рабочее поле** Черепашки или **холст** (4), здесь она и будет рисовать. Справа – **поле команд** или **программный лист** (3) предназначен для ввода команд Logo. Постепенно мы познакомимся, какие возможности нам предоставляет программа через **меню** (1), **панель инструментов** (2) и **панель состояния** (5), на которой отображается различная информация KTurtle.

***Команда* —** это одно предложение на языке Logo.

Повинуясь нашим командам, Черепашка может двигаться вперед (по направлению своей головы) и назад, поворачиваться направо и налево… Очевидно, в этих случаях нам надо указывать не только **имя** соответствующей команды, но и на какое расстояние мы хотим, чтобы Черепашка переместилась или на какой угол она должна повернуться. Мы также можем отправить Черепашку в любую точку холста, тогда нам нужно задать ей координаты этой точки. Делается это с помощью **параметров**. По нашей команде Черепашка может поднять и опустить карандаш, стереть все нарисованное… Такие команды не требуют параметра. Таким образом, **команда состоит или только из имени или из имени и параметров** (одного или нескольких).

В ответ на введенные команды Черепашка совершает соответствующие действия или выдает сообщение об ошибке. Ошибок бояться не надо, их делают и профессионалы, при решении задач на компьютере обязательно существует **этап отладки программы — поиск и устранение ошибок**. Ошибки надо уметь находить и

исправлять, а для этого надо не только научиться писать команды, но и понимать сообщения — отказ выполнить неправильно введенную команду.

2.2 Настало время испытаний

Для начала поместим в программный лист готовую программу и посмотрим, как она записана и как выполняется. Вызываем пункт меню **Файл → Открыть примеры** и выбираем необходимый пример в появившемся диалоге. Имя файла говорит о том, что данный пример демонстрирует (например flower.logo предназначен для построения изображения, похожего на цветок, на рис.1). Выбранный файл открывается в программном листе и мы запустим его на выполнение, вызвав пункт меню **File → Examples.**

Задание: загрузите и выполните каждый пример, внимательно наблюдая за тем, что происходит на холсте.

Вопросы:

1. Где находится Черепашка после запуска программы KTurtle?
2. Куда направлена ее голова?
3. Какие действия совершала Черепашка в ходе выполнения примеров?
4. Что происходило с размерами, цветом и формой холста?

В ходе наших испытаний вы правильно заметили, что после запуска программы KTurtle Черепашка находится в центре холста и голова ее направлена вверх. Еще одно важное замечание – цвет, размер и форма холста в каждом из выполненных примеров были разными. Кроме того, Черепашка умеет мастерски использовать карандаши разного цвета и толщины. Таким образом, нам предстоит научиться задавать Черепашке множество разных команд, чтобы она понимала и правильно выполняла то, что мы хотим ей поручить.

При первом знакомстве с Черепашкой мы выбрали для нее профессию художника-графика. А чем отличается художник-график от других художников? Тем, что он рисует черным карандашом на белой бумаге.

Для начала попросим теперь Черепашку пройти вперед, для чего на программном листе напечатаем команду **forward** (вы уже заметили, конечно, что язык нашего исполнителя очень близок к английскому и поэтому не так и трудно будет его учить) и выполним эту команду так, как мы выполняли готовые примеры программ. Но что такое? Вместо того, чтобы повиноваться нашим указаниям, Черепашка отвечает нам отказом и появляется окно с сообщение — **Команда forward была вызвана с 0. Требуется 1 параметр**. Ну конечно, только что поговорили о параметре и тут же про него забыли! Но как Черепашка измеряет расстояния — в сантиметрах, дюймах, милях? Нет, для измерения расстояния в яыке Logo применяется своя мера длины — черепашьи шаги. Исправляем ошибку, для чего ставим курсор в программном листе рядом со словом **forward**, делаем пробел, пишем какое-нибудь число, например, 10 и снова запускаем программу. Черепашка едва заметно продвинулась вперед — шаги у нее маленькие. Проведем еще несколько экспериментов, запустим нашу программу на выполнение несколько раз. Что получается? Черепашка продвигается вперед и оставляет за собой тонкую линию черного цвета, при этом она каждый раз продвигается дальше вперед к верхнему краю холста. Постараемся измерить высоту рабочего поля от центра до верхнего края. Сколько у вас получилось? Запомните результат.

А теперь давайте попросим ее пройти назад по направлению хвостика. Теперь мы уже не забудем про параметр и правильно запишем команду, взяв в качестве параметра то число, которое вы получили, измеряя высоту рабочего поля. У меня получилось что-то около **160**, так что я пишу **backward 160** вместо команды **forward 10** и снова выполняю

программу. Черепашка послушно перемещается на середину холста. Запустим программу еще раз. Черепашка опустилась вниз холста и снова нарисовала тонкую линию черного цвета.

Как заставить Черепашку выполнять программу заново при каждом запуске? Для этого добавим перед командой **backward 160** чистую строку (нужно поместить курсор в начале строки и нажать клавишу Enter) и напишем слово **reset.** Запустим программу. Что получилось? Черепашка очистила холст и выполнила заданную команду из его центра. Таким образом, после выполнения этой команды всё будет выглядеть так, как будто вы только что запустили KTurtle. Черепашка будет расположена в центре экрана, цвет холста будет белым, цвет пера Черепашки будет черным.

Для получения правильной картины на холсте, программу следует начинать командой **reset!**

А не захотелось ли вам узнать, что произойдет, если попросить Черепашку переместиться на **дробное** число шагов, или задать число шагов в виде выражения? Смелее, смелее с экспериментами!

Зададим теперь параметр в виде выражения, используя знаки арифметических операций. Надо только знать, что умножение обозначается звездочкой "*****", деление — косой чертой "**/**". Вводим команду

backward (2*100+100)/3

Произошло то, что вы, наверное, и ожидали – Черепашка переместилась на 100 шагов назад.

Выберем теперь такое выражение, значение которого не является целым числом, например,

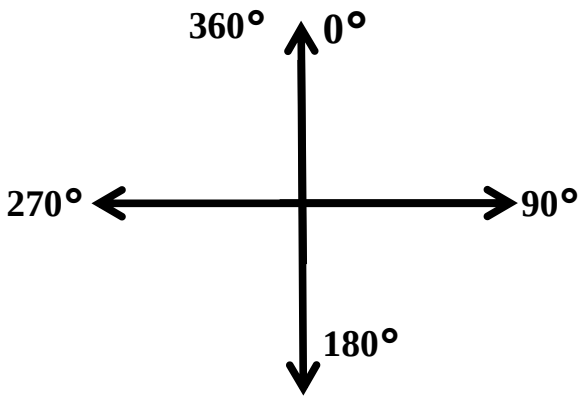
forward 100/3

Как видим, Черепашка не задумываясь шагает вперед, она понимает и доли шага.

Остался еще неисследованный вопрос — что будет, если мы попросим Черепашку сделать больше шагов, чем расстояние от нее до края рабочего поля? Ваши предположения? А теперь проверьте их экспериментальным путем.

Проведем теперь эксперименты с командами поворота. Поворот по часовой стрелке (направо) Черепашка выполняет по команде **turnright**, против часовой (налево) - по команде **turnleft**. Параметром в этих командах является градусная мера угла поворота. Вспомните, что прямой угол составляет 90 градусов, развернутый - 180, а полный угол - 360 градусов. Вы, конечно, не забудете в качестве параметра использовать дробные числа и арифметические выражения.

Команды **turnright** и **turnleft** поворачивают Черепашку на указанный угол **относительно ее текущего положения**. Но, часто в задачах возникает ситуация, когда нас интересует **абсолютное** положение Черепашки, ее "курс" относительно некоторого раз и навсегда выбранного направления независимо от ее текущего положения. В качестве такого ориентира принято направление "на север", как на обычной карте (вверх, угол 0º). Командой **direction** (direction - по-английски направление) задается поворот Черепашки по часовой стрелке относительно этого фиксированного направления.



Таким образом, попросим Черепашку нарисовать, например, квадрат со стороной 100 шагов. Для этого надо последовательно ввести команды (кстати, не обязательно каждый раз переписывать повторяющиеся строки заново, их можно один раз скопировать, а затем вставить столько раз, сколько это необходимо):

forward 100
turnright 90
forward 100
turnright 90
forward 100
turnright 90
forward 100
turnright 90

Для того, чтобы Черепашка при перемещении не оставляла следа, дают команду **penup** (поднять карандаш). Чтобы Черепашка снова могла рисовать, ей необходимо дать команду **pendown** (опустить карандаш).

Для того, чтобы Черепашка могла рисовать линии разной толщины, у нее есть команда **penwidth**, в которой нужно указать толщину карандаша. Наименьшая толщина карандаша равна 1. Чтобы Черепашка рисовала карандашом нужной толщины, например 3, ей дают команду **penwidth 3**

Для того, чтобы почистить рабочее поле, используют команды **clear и reset**. Команда **clear**, стирая графику, не меняет положение Черепашки.

Команда **reset** очищает более объёмно, чем команда **clear**. После выполнения этой команды всё будет выглядеть так, как будто вы только что запустили KTurtle. Холст будет иметь размер, установленный по умолчанию в настройках программы KTurtle. Черепашка будет расположена в центре холста, цвет холста будет белым, цвет пера Черепашки будет черным.

Глава 3

В которой Черепашка учит нас пользоваться разными кистями и красками и раскрашивать мир Лого всеми цветами радуги.

Какой же художник не имеет запас кисточек и красок? И Черепашка Лого умеет менять толщину своего пера, его цвет, цвет рабочего поля.

Для этого в ее словаре есть команды:

pencolor – устанавливает цвет пера при помощи трех параметров **R,G,B.** В качестве параметров указывается интенсивность красной (R), зеленой (G) и синей (B) составляющих цвета. Каждый параметр может быть задан числом от 0 до 255. Черный цвет устанавливается командой **pencolor 0,0,0.** Белый – командой **pencolor 255,255,255.** Красный – командой **pencolor 255,0,0.** Зеленый – командой **pencolor 0,255,0.** Синий – командой **pencolor 0,0,255.** Варьируя значения параметров цвета, можно получить более 16 миллионов цветов и оттенков.

canvascolor — устанавливает цвет фона (требуется указать те же параметры, что и в команде **pencolor**)**.**

Глава 4

В которой мы учимся анализировать допущенные ошибки и сохранять программы.

Черепашка выполняет предписанный ей алгоритм от первой команды до последней автоматически, без какого-либо внешнего вмешательства.

Но нередко, особенно на первых шагах, может показаться, что Черепашка живет своей жизнью — вы ей задаете одну программу действий, а она действует по другой. Но не надо забывать, что Черепашка — исполнитель бездумный и выполняет она не наши намерения, а наши инструкции. При выполнении программы могут встретиться разного рода ошибки. Если на экране появилось сообщение об ошибке, надо его внимательно прочитать и постараться понять. Во многих случаях оно подскажет, какие исправления надо сделать. Сложнее другая ситуация, когда сообщения об ошибке нет, но результат работы программы не тот, который мы ожидали. В таком случае надо определить место в рисунке и программе, где допущен «сбой», поставить себя на место Черепашки и выполнить на бумаге вместо нее предписанные программой инструкции.

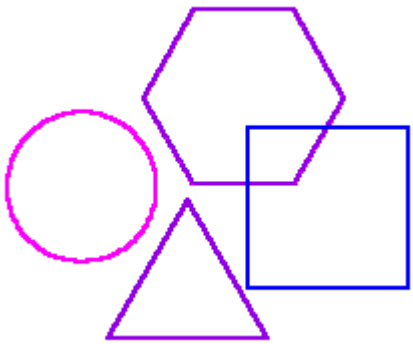
Как же исправить обнаруженные в программе ошибки?

На самом деле, поле программ — это простейший текстовый редактор, где можно выполнять обычную редакторскую правку: вставлять и удалять символы, копировать и переносить фрагменты текста.

Для того, чтобы сохранить проект, надо в горизонтальном меню программы выбрать пункт **Файл**, а в открывшемся ниспадающем меню — пункт **Сохранить как**… Выбрав нужную папку и дав имя вашему проекту, сохраните его, нажав кнопку **Сохранить**. Расширение имени файла будет приписано автоматически — logo. При повторном сохранении следует выбрать пункт ниспадающего меню **Сохранить** или щелкнуть на изображении дискеты на линейке инструментов.

Глава 5

В которой мы отправляем Черепашку в кругосветное путешествие, открываем "Закон 360-ти" и обучаем Черепашку рисовать правильные многоугольники и остроконечные звезды.



На рисунке изображены различные правильные многоугольники — треугольник, квадрат, шестиугольник, 180-угольник. В последнем случае зрение нас немного подводит, и мы не видим вершин многоугольника, для нас он превращается в круг.

Попробуйте вместе с Черепашкой обойти эти фигуры и вернуться в исходное положение.

На какой угол повернулась Черепашка, путешествуя вокруг треугольника?

А на какой угол повернется она, если ее маршрутом будет 6-угольник или 180-угольник?

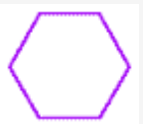
Если вы были внимательны и аккуратны, то получили один и тот же ответ во всех случаях — Черепашка сделала полный поворот, повернулась на 360 градусов!

Изменится ли это значение, если Черепашка будет обходить 5-угольник или 20-угольник? Конечно же нет! Это правило справедливо для любого правильного многоугольника.

Рассуждаем дальше.

Обходя контур многоугольника, Черепашка поворачивается в каждой его вершине. Мы знаем, что за время путешествия она совершает полный поворот. Значит, в каждой вершине она поворачивается на угол, равный 360/n, где n — число вершин многоугольника.

Составим таблицу для некоторых многоугольников.

Многоугольник	Английское название	Число вершин	Угол поворота в вершине	Рисунок
треугольник	triangle	3	360 / 3 =	
квадрат	square	4	360 / 4 =	
пятиугольник	pentagon	5	360 / 5 =	
шестиугольник	hexagon	6	360 / 6 =	
360-угольник		360	360 / 360 =	

Мы уже очень близки к цели и почти готовы написать команды для рисования любого правильного многоугольника, только есть проблема — неужели, например, для 360-угольника с длиной стороны 2, нам придется 360 раз писать одни и те же команды — **forward 2** и **turnright 1**?

Здесь на помощь нам придет команда **повтори**, которая по-английски пишется **repeat**. Записывается команда так:

```
repeat 360 {
forward 2
turnright 1
}
```

После слова **repeat** указывается, сколько раз следует повторять указанные команды, а сами команды записываются в фигурных скобках.

Таким образом, для рисования треугольника длиной стороны 90 шагов следует выполнить команду

```
repeat 3 {
forward 90
turnright 360/3
}
```

или

```
repeat 3 {
forward 90
turnright 120
}
```

Для 5-угольника и 6-угольника — соответственно:

```
repeat 5 {
forward 90
turnright
360/5
}
```

или

```
repeat 5 {
forward 90
turnright 72
}
```

```
repeat 6 {
forward
90
turnright
360/6
}
```

или

```
repeat 6 {
forward 90
turnright 60
}
```

Обратимся теперь к хорошо знакомой нам пятиконечной звездочке.
Обходя контур звездочки, Черепашка делает пять поворотов в ее вершинах и возвращается в исходное положение, так же, как и при обходе правильного пятиугольника. Но, очевидно, команда **repeat 5{forward 80 turnright 360/5}** не решит задачу рисования звездочки. Что же изменилось в этой ситуации? Ответ на этот вопрос мы найдем, если аккуратно пройдем вместе с Черепашкой вдоль контура звездочки и внимательно проследим за ее поворотами. В этом случае, в отличие от правильного пятиугольника, Черепашка совершает два полных поворота, то есть за свое путешествие она поворачивается на угол 720 градусов! Значит, команду для рисования пятиконечной звездочки следует переписать так:

```
repeat 5{
forward 80
turnright
360*2/5
}
```

или

```
repeat 5{
forward 80
turnright 720/5
}
```

или

```
repeat 5{
forward 80
turnright 144
}
```

Теперь нам не представляет труда нарисовать и восьмиконечную звездочку. Однако очевидная, казалось бы, команда **repeat 8{forward 80 turnright 720/8}** приводит к неожиданному результату — вместо ожидаемой колючей звездочки мы получаем обыкновенный квадрат. На самом деле результат очевиден, так как 720/8=90 и, совершив четыре поворота, Черепашка возвращается в исходное положение и дальше следует по уже нарисованному контуру. Догадливый читатель уже давно понял, что в этом случае Черепашка за свое путешествие

вдоль контура звездочки сделала не один и не два полных поворота. Сколько именно — оставим для самостоятельного исследования.

Теперь мы можем **подвести итог** нашим рассуждениям и сформулировать закон кругосветного путешествия или "Закон 360-ти":

Если после путешествия по рабочему полю, Черепашка возвращается в исходное положение, то она совершает один или несколько полных поворотов на 360 градусов.

Заметьте, что "несколько" может означать и ни одного. Этот закон позволяет нам легко рассчитать повороты Черепашки при рисовании любых правильных многоугольников, а также и звездчатых многоугольников.

Глава 6
В которой мы знакомимся с циклическим алгоритмом и вложенными циклами

В мире, в котором мы живем, да и внутри нашего организма многие процессы являются циклическими, повторяющимися. Смена времен года, работа двигателя автомобиля, катание на санках с горки — все это циклические процессы. Некоторые процессы повторяются бесконечное число раз, некоторые заканчиваются (или выполняются) при выполнении какого-либо условия, некоторые — заданное число раз. Также и алгоритмы могут быть циклическими.

Циклический алгоритм — алгоритм, предусматривающий многократное повторение одних и тех же команд.

Повторяющиеся команды составляют тело цикла.

С помощью команды **repeat**, с которой мы познакомились в предыдущей главе, можно описать циклические алгоритмы на языке программирования Лого. Команды, которые мы записываем в фигурных скобках, составляют тело цикла, а слово **repeat** с указанием числа повторений — это заголовок цикла. В заголовке может быть указано не только число, но и арифметическое выражение, например,

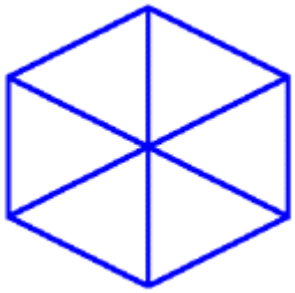
repeat 5 * 6 {forward 10 turnright 360/30}

Интересно, а что будет, если указать **нецелое** число повторений? Если, например, задать команды

repeat 3.6 { forward 70 turnright 120} или
repeat 10/3 { forward 70 turnright 120}?

Самый надежный способ получить правильные ответы на эти вопросы — провести эксперименты с самой Черепашкой.

Тело цикла могут составлять любые команды, без каких-либо ограничений, в том числе, и команды цикла. То есть, внутри одного циклического процесса могут происходить и другие самостоятельные циклические процессы.

<table>
<tr><td>  </td><td>Например, картинка слева появится из-под пера Черепашки, если она выполнит алгоритм

repeat 6 {repeat 3{forward 60 turnright 120} turnright 60}</td></tr>
<tr><td>Такая структура называется вложенные циклы и может быть изображена схемой:</td><td> 
</td></tr>
</table>

Тело внутреннего цикла опять может содержать цикл и так далее, **вложенность** может быть очень глубокой, главное, самому не запутаться и сохранить ясность и понятность алгоритма.

Глава 7

В которой мы обучаем Черепашку рисовать окружности и дуги и вспоминаем про замечательное число π, с помощью которого еще древние греки определяли длину окружности по ее радиусу.

> *"И сделал литое из меди море, — от края его до края его десять локтей, — совсем круглое… и снурок в тридцать локтей обнимал его кругом"*
>
> *(3 Цар.7:23)*

Мы уже убедились в том, что 360-угольник или 180-угольник или даже 60-угольник вполне могут служить нам моделью окружности. Наш глаз не замечает "неровностей" их контура. Эти многоугольники можно нарисовать, используя, например, команды

repeat 360 {forward 1 turnright 1}
или
repeat 180 {forward 1.5 turnright 2}
или
repeat 60{forward 2.5 turnright 6}

Но при такой записи, задавая наугад длину стороны многоугольника, трудно представить, какая именно окружность получится.

Мы знаем, что у окружности есть очень важная характеристика — ее радиус. Как рассчитать длину стороны многоугольника, заменяющего нам окружность, в зависимости от радиуса R этой окружности?

Еще в Древней Греции ученые знали, если измерить длину окружности и затем разделить полученное значение на длину диаметра (а он равен двум радиусам), то для любой окружности — маленькой, большой, крохотной или громадной — получится одно и то же число, близкое к трем, которое они обозначили π (читается: “пи”). π— замечательное число, равное приблизительно 3,1415926 и получившее свое название по первой букве греческого слова “периферия” (“окружность”). Используя π, можно записать:

длина окружности = 2πR,
π- замечательное число, равное приблизительно 3,1415926

В Лого не обязательно помнить значение числа π, Черепашка понимает его обозначение — pi, которое можно применять в формулах.

Теперь у нас достаточно идей для построения окружности. Действительно, пусть мы хотим нарисовать окружность радиуса, например, 50. Длина такой окружности составит 2 * pi * 50 (черепашьих шагов). В Лого окружность мы заменяем многоугольником и в этом случае 2 * pi * 50 есть периметр этого многоугольника. Тогда для 360-угольника длина стороны составит 2 * pi * 50 / 360, для 180-угольника — 2 * pi * 50 / 180, для 60-угольника — 2 * pi * 50 / 60. Используя "Закон 360-ти" нетрудно написать команды для рисования этих многоугольников:

repeat 360{forward 2 * pi * 50 / 360 turnright 1} — 360-ти угольник
repeat 180{forward 2 * pi * 50 / 180 turnright 2} — 180-ти угольник
repeat 60{forward 2 * pi * 50 / 60 turnright 6} — 60-ти угольник

Любой из этих многоугольников может служить моделью окружности радиуса 50.