Романова Л. Б.,
Романов В. Ю.

Как ребята программировали игру «Аффрика» на языке Паскаль

Москва

ББК 32.973.26-018.1
Р69

Романова Л. Б., Романов В. Ю.
Р69 Как ребята программировали игру «Африка» на языке Паскаль. – М.: ДМК Пресс. – 112 с.: ил. (Юному программисту).

ISBN 5-94074-077-4

В книге даются основы программирования на языке Паскаль. Дети, вместе с родителями или самостоятельно, получат навыки структурного программирования с использованием видеопамяти компьютера. Герои придумывают собственную игру «Африка» и шаг за шагом, от простого к сложному, программируют ее.

Прочитав книгу, дети смогут придумывать и программировать собственные компьютерные игры. Издание предназначено для детей от 10 лет и старше.

ББК 32.973.26-018.1

ISBN 5-94074-077-4

СОДЕРЖАНИЕ

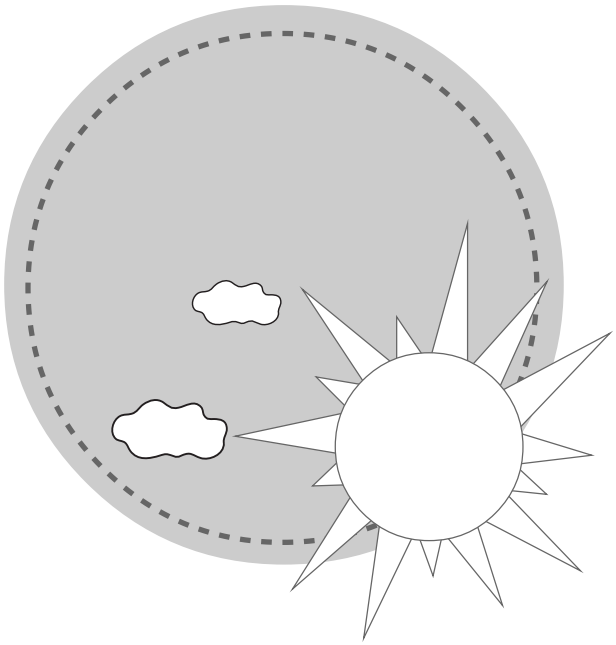
ПРЕДИСЛОВИЕ

Эта книга написана для ребят, которые хотят научиться создавать свои компьютерные игры. Возможно, вы уже знаете, что компьютер владеет многими языками программирования. Но чтобы компьютер понял вас и решил предлагаемую задачу, разговаривать с ним нужно на определенном языке, например на языке Pascal, Java, Basic, C++ и других.

В нашей книге все программы написаны на языке Pascal (Паскаль). Мы выбрали его потому, что он предназначен для решения самых разных задач: игровых, деловых, экономических, информационных и других. Швейцарский профессор Никлаус Вирт разработал этот язык и назвал его в честь французского ученого Блеза Паскаля, изобретателя первой счетной машины. Благодаря простоте, логичности и эффективности язык Паскаль получил широкое распространение во всем мире. Поэтому многие люди в разных странах учатся программировать именно на этом языке.

Ребята, мы надеемся, что, читая эту книгу, вы вместе с ее героями будете писать программу — игру «Африка». Возможно, у кого-то возникнут трудности, тогда можно попросить помочь папу или маму. А если они еще не умеют программировать, то с удовольствием начнут учиться вместе с вами.

Картинки нам помогал рисовать Сережа Романов. Вы тоже сумеете проиллюстрировать свои сказки смешными картинками, нарисованными на бумаге или на экране дисплея. Эти рисунки не пропадут, если вы и дальше будете учиться программировать. Компьютер с вашей помощью сможет сделать из них мультфильмы. Когда вы будете готовы отправиться в новое увлекательное путешествие или изучить еще один язык программирования, расскажите нам об этом. Свое письмо пришлите электронной почтой по адресу: childpascal@dmk.ru.

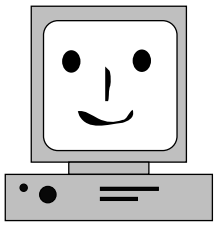


ГЛАВА 1.

ПАПА ПРИНОСИТ ДОМОЙ КОМПЬЮТЕР

Был вечер. Я решал задачи, Таня рисовала, а мама готовила ужин. Вдруг раздался звонок. Мы выбежали в коридор и открыли дверь.

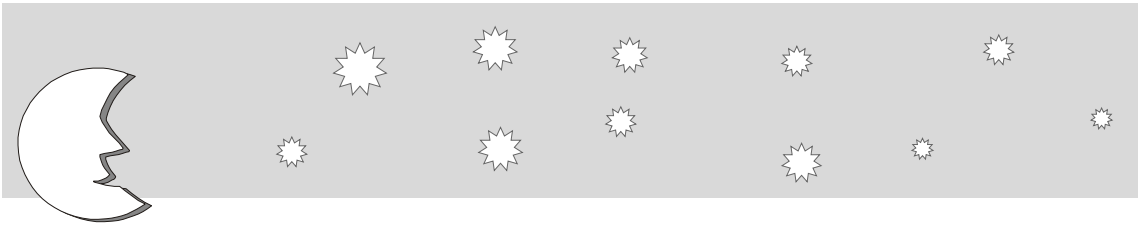
Вошел папа с большой коробкой в руках. Он улыбнулся и громко сказал:



— Ребята, к вам в гости пришел компьютер.

— Ура! — закричали мы. Папа осторожно вытащил компьютер из коробки, поставил на письменный стол, потом включил в розетку и нажал какие-то клавиши. Раздался голос:

— Здравствуйте, Таня и Ваня. Я буду рад с вами поиграть и показать на экране дисплея все, о чем попросите.



Папа рассказал нам, что компьютеры бывают большими и маленькими. Большие иногда называют электронными вычислительными машинами (сокращенно ЭВМ). Это они управляют ракетами, кораблями и другими сложными устройствами. Даже один такой компьютер не уместился бы в нашей квартире. Зато маленький, работающий от специальной батарейки, можно положить в мой или Танин портфель. Память у маленького компьютера не хуже, чем у большого. А компьютеры, похожие на нашего нового друга, чаще всего используют на работе и дома.

Компьютер опять заговорил:

— Ребята, давайте играть. Я жду ваших программ.

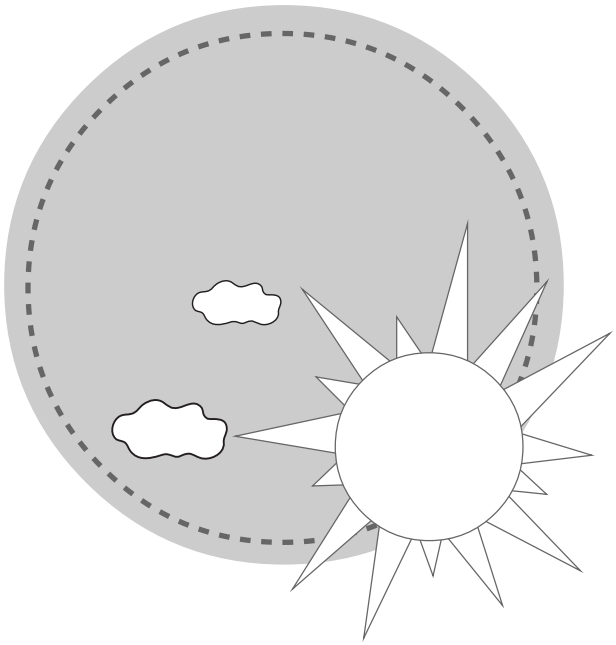
Мы сразу согласились, так как уже умели играть в некоторые компьютерные игры: «Змея», «Колобки», «Лягушонок», «Минное поле», «Самолет», «Луноходик» и другие. Но как компьютер догадался, что мы хотим писать свои программы, для нас было загадкой. Я спросил папу:

— Можно с помощью наших программ рисовать на экране дисплея? В компьютерных играх так много интересных картинок!

Папа ответил:

— Конечно, но только с помощью *дисплейного адаптера*. Вспомните, как в гостях у компьютера вы познакомились с микросхемами и микропроцессорами, похожими на игрушечных жучков. Они находились внутри компьютера. Представьте себе, что там живет еще жук-художник — дисплейный адаптер. Этот игрушечный жучок умеет рисовать.

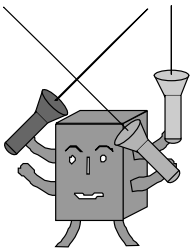
Папа нажал нужные клавиши. Компьютер подмигнул нам, и на экране появился рисунок.



ГЛАВА 2.

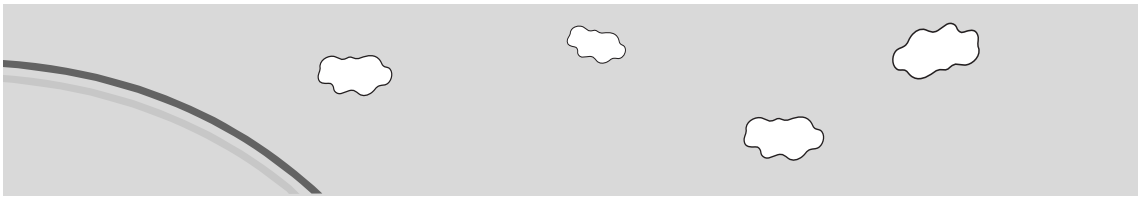
ЧТО ДЕЛАЕТ В КОМПЬЮТЕРЕ ДИСПЛЕЙНЫЙ АДАПТЕР

На рисунке был изображен очень смешной игрушечный жук-художник. В лапках он держал кисточки и был похож на микросхемы и микропроцессоры, которые находятся внутри компьютера.

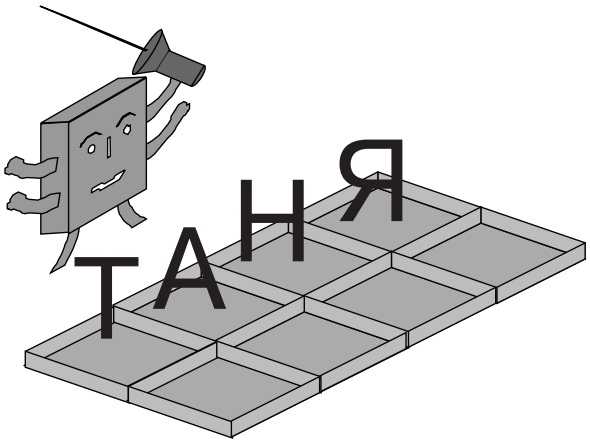


К нашему удивлению, кисточкой у жука был электронный луч, а бумагой — экран дисплея.

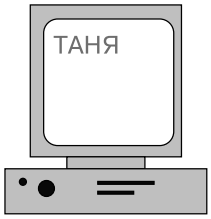
Но главное, как сказал папа, у дисплейного адаптера была своя память, похожая на ящички. Интересно было бы посмотреть, что там лежит. Компьютер, видимо, догадался о нашем желании и мгновенно откликнулся. На экране, как в мультфильме, появилась память жука. Память дисплейного адаптера состояла из маленьких ящичков, плотно прижатых



друг к другу. В них лежали буквы. Таня прочитала их и узнала свое имя.



Жук-художник все время заглядывал в ящички и кисточкой-лучом срисовывал буквы на свою «бумагу» — экран, чтобы буквы не успели погаснуть. Получилась такая картинка.



У адаптера на «листе бумаги» умещается 25 строчек, по 80 букв в каждой строчке. Но чтобы буквы не гасли на экране, адаптер должен заглядывать в ящички много-много раз в секунду.

— Здорово умеет работать адаптер. Но кто же кладет буквы в ящички? — спросил я папу.

— Буквы, Ваня, кладут программы. Они и дают задание дисплейному адаптеру написать слово на экране, — ответил он.

— А почему все буквы одного цвета? — поинтересовалась Таня.

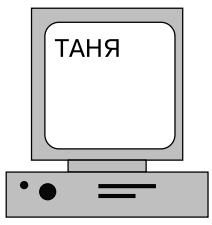
Папа объяснил, что компьютер показал нам одноцветный адаптер. Его еще называют *монохромным*. Он умеет рисовать только одним цветом. Неожиданно вмешался компьютер:

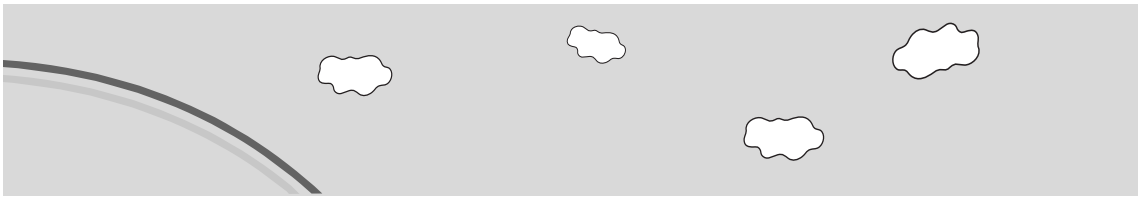
— Ребята, теперь взгляните на цветной дисплейный адаптер. Он умеет писать цветные буквы.

Через секунду на экране ожил и заговорил разноцветный жук-художник:

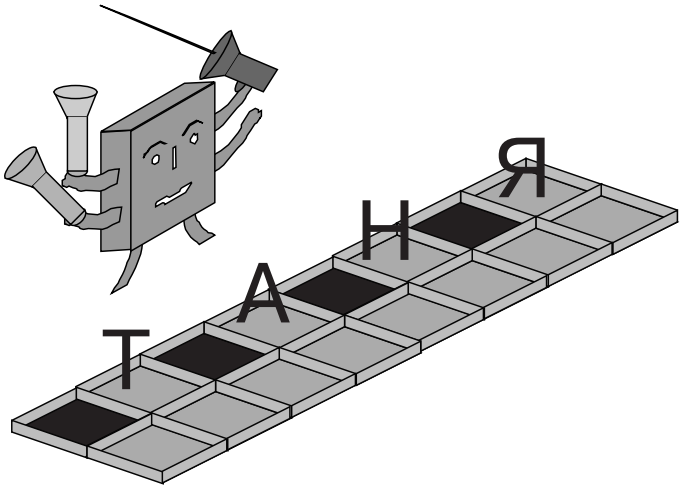
Я адаптер — жук-художник.
Я компьютерный волшебник.
В лапках кисточки держу
И с программами дружу.
Мои кисточки-лучи,
Как прожекторы в ночи,
Перемешивают краски,
Чтоб раскрасить ваши сказки.
А цвета их — РэдГринБлю.
Краски я свои люблю.
Напишу вам на экране
Красной краской слово «Таня».

Жук-художник держал в лапках три кисточки. Электронные лучи были красного, зеленого и голубого цвета. Если написать подряд названия этих трех цветов на английском языке, получится то самое непонятное слово RedGreenBlue (РэдГринБлю) из стишка жука-художника. Адаптер подбирает нужный цвет, смешивая три краски. Если смешивать красный, зеленый и голубой, можно получить много разных цветов и оттенков. В ящичках памяти цветного жука-художника буквы лежали так же, как у монохромного жука. Только перед каждой буквой стоял ящичек с краской для нее. Пока мы слушали папу и разглядывали картинку, адаптер выполнил свое обещание: написал на экране слово «Таня» красными буквами.





Потом компьютер показал, как это слово хранится в памяти жука-художника.



Очень красивые буквы умеют писать монохромный и цветной адаптеры. Интересно, могут ли они рисовать линии? Но оказалось, что некоторые адаптеры рисуют только буквы. И поэтому их еще называют *текстовыми* или *алфавитно-цифровыми*. Таня спросила:

— А какой адаптер у нашего друга?

Компьютер и папа ответили одновременно:

— Цветной графический. Он умеет делать то же самое, что и текстовый, но еще рисует линии.

— Как же он пишет буквы, ведь у него в памяти лежат только краски? — спросил я папу. Тот ответил:

— Цветной графический адаптер рисует буквы, как картинки, которые состоят из множества точек.

Тут компьютер показал на экране графический адаптер. Оказывается, в памяти этого жука-художника вообще нет букв, а есть только ящички с красками. Каждая краска соответствует маленькой точечке на экране — «бумаге» жука-художника. Этих точек — они называются *пикселами* — очень много. Каждый пиксел имеет свой

ящичек в памяти графического адаптера. Именно в этом ящичке лежит краска для точки на экране.

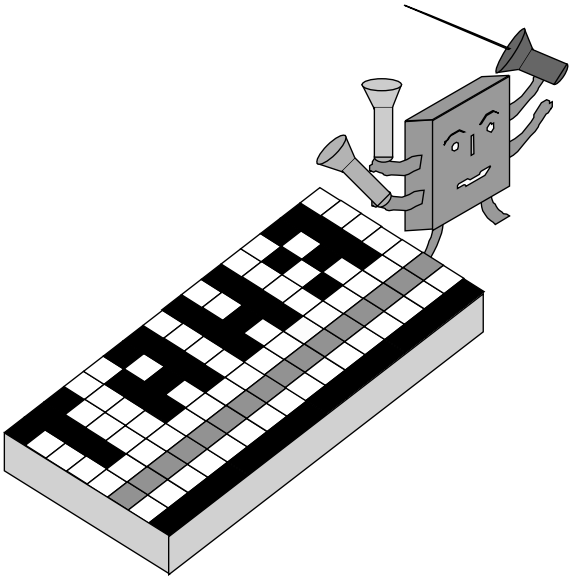
— А сколько всего пикселов умещается на «бумаге» нашего адаптера? — спросил я.

— 640 точек в ширину и 350 в высоту, — ответил папа. — Перемножьте эти числа и получите количество пикселов на экране.

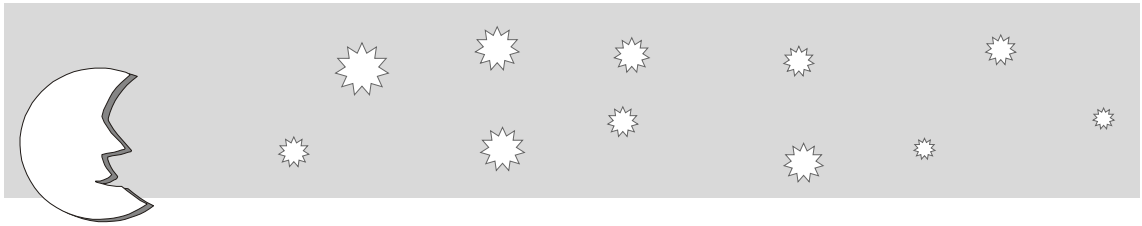
У меня получилось очень большое число — 224000. Папа объяснил, что именно столько ящичков в памяти нашего графического адаптера.

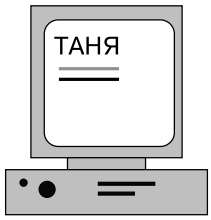
— Как интересно! — воскликнула Таня. — Я люблю рисовать. Папа, давай попросим адаптер что-нибудь нарисовать.

Графический адаптер услышал Таню, и на экране появились зеленая и красная линии.



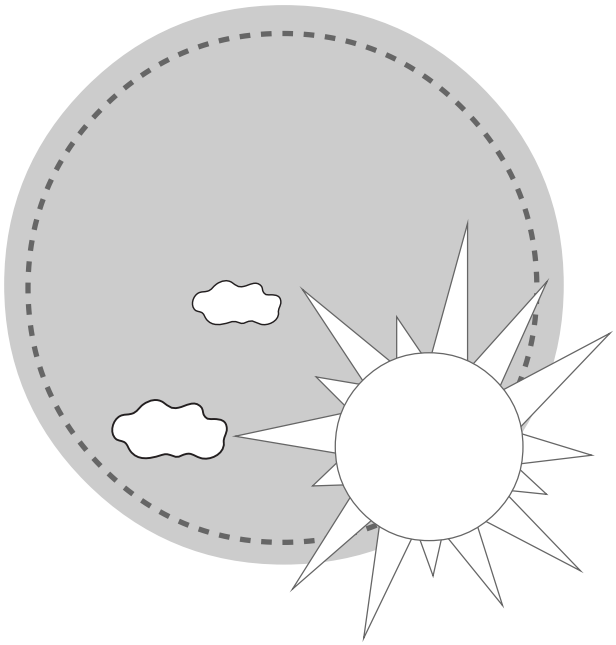
Потом компьютер показал, как зеленая и красная линии хранятся в памяти графического адаптера.





— Молодец компьютер! — закричали мы и захлопали в ладоши. Папа рассмеялся и сказал:

— Хорошо, Таня, а теперь поучимся писать программы. Для начала дадим адаптеру самое простое задание на языке программирования Паскаль: пусть адаптер напишет на экране слово «Таня».



ГЛАВА 3.

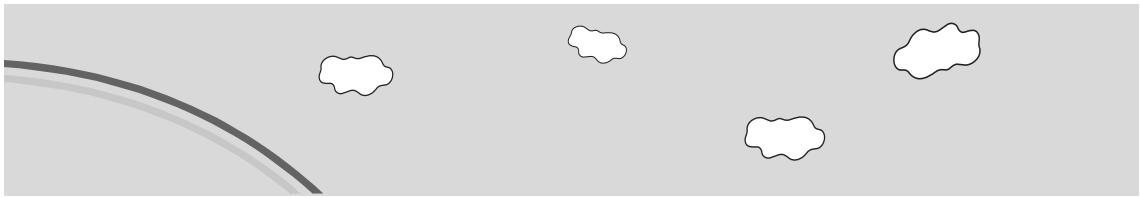
МЫ С ПАПОЙ ПИШЕМ ЗАДАНИЕ АДАПТЕРУ НА ЯЗЫКЕ ПАСКАЛЬ

Папа нажал на клавиатуре какие-то клавиши и запустил текстовый редактор. Эта программа нам уже знакома. Она помогает печатать на экране буквы, как пишущая машинка. Папа напечатал непонятные слова:

```
program Toy;
uses Crt;
begin
      Write('Таня');
end.
```

— Что же это за задание? — спросила Таня.

А я попытался его прочесть. Но смог прочитать лишь первое слово: `program` — это программа. Следующее слово я не понял и спросил папу, как оно переводится.



Оказалось, `Toy` означает «игрушка». Это и есть название нашей программы.

Таня загрустила. Она еще не изучала английский язык и испугалась, что не сможет давать задания адаптеру.

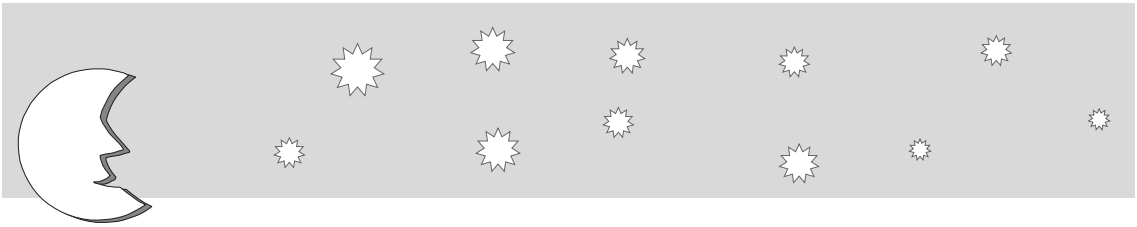
Папа стал ее успокаивать. Он сказал, что нужно знать совсем немного английских слов. Их нетрудно выучить. А некоторые слова можно даже записать латинскими буквами. Например, вместо `Toy` написать `Igra` (игра). Прочесть и понять такое название легко. Важно выучить ключевые слова: они самые главные. Например, слово `begin` — ключевое. Оно означает «начало». А ключевое слово `end` означает «конец». Папа спросил меня, как переводится слово `Write`. Я подсказал ему: «записать». И он продолжил:

— Ребята, слово `Write` — это вызов *функции*, которая помогает адаптеру писать строчку «Таня» на экране дисплея.

Таня загрустила еще больше: она не могла понять, почему нужно так много знать, чтобы написать маленькое задание жуку-художнику. Ей было трудно запомнить сразу несколько английских слов. Но папа сказал:

— Первый шаг, Таня, всегда самый трудный. Я помогу. Чтобы ты не забыла, что я рассказывал, мы будем писать в программах пояснения. Они еще называются *комментариями*. Все настоящие программисты пишут пояснения к своим программам, чтобы не забыть, как программы работают. А программу с комментариями можно даже подарить друзьям. Ваши пояснения помогут им ее понять.

Папа написал в текстовом редакторе комментарии к программе. Они располагались между скобками и звездочками: `(* … *)`.

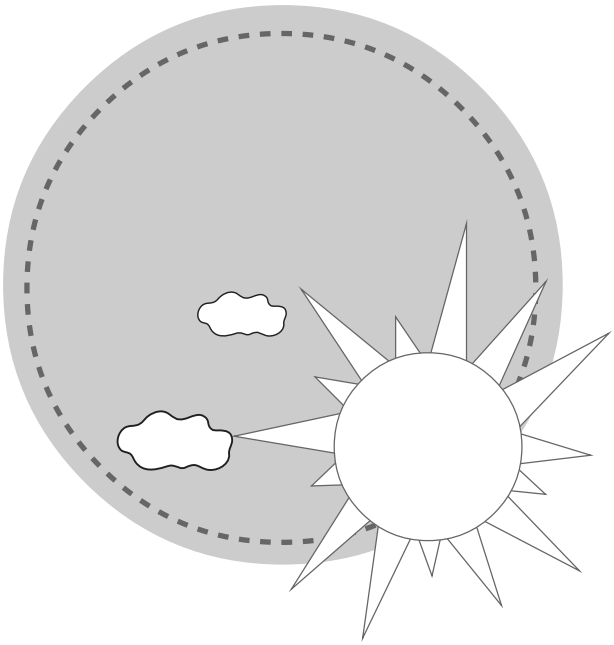


```
program Igra;       (* Это программа "Игра". *)

(* Подключение к программе функций - друзей жука-художника. *)
uses Crt;

begin               (* Это ключевое слово "начало". *)
    Write("Таня");  (* Это функция "записать" - друг жука-художника. *)
end.                (* Это ключевое слово "конец". *)
```

Мы прочитали программу и не поняли, что же такое функция, друг жука-художника. Неужели это тоже жук-художник? Папа улыбнулся:

— Функция — это маленькая программа, которая была написана раньше другими людьми, но вы можете использовать ее в своей программе. Адаптеры изготавливают на заводе из металла, пластмассы и других материалов. А вот функции и программы делаются намного проще. И вы, ребята, немного изучив язык программирования, начнете писать свои функции. А сейчас давайте представим, что функция `Write` — это маленький волшебный человечек, который приносит в память адаптера разные буквы.

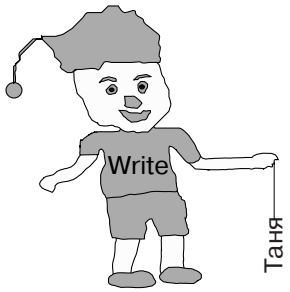
В наш разговор вмешался компьютер. Он весело подмигнул и показал на экране смешного игрушечного человечка — функцию `Write` (Записать).



ГЛАВА 4.

ФУНКЦИЯ WRITE ПОМОГАЕТ ДИСПЛЕЙНОМУ АДАПТЕРУ ПИСАТЬ НА ЭКРАНЕ

На рисунке функция `Write` держала в руках строчку «Таня».



В нашей программе эта строчка была заключена в скобки и еще в кавычки:

```
Write("Таня").
```

— Что же они означают? — спросил я у папы.

Он объяснил, что между открывающей и закрывающей скобками (…) находится *параметр* — задание для функции. Его и дают в руки человечку, прежде чем он должен что-то сделать. Расположенные между кавычками буквы и есть та строчка, которую нужно отнести в память адаптера. Функция `Write` отнесла параметр «Таня» в память адаптера, и он сразу же написал на экране строчку «Таня». Таня очень обрадовалась, увидев свое имя, и предложила:

— Папа, давай сделаем вот как: «Таня пишет программу».

Папа согласился и сказал, что для этого необходимо дать адаптеру другое задание. И дописал к нашей программе еще одну строчку:

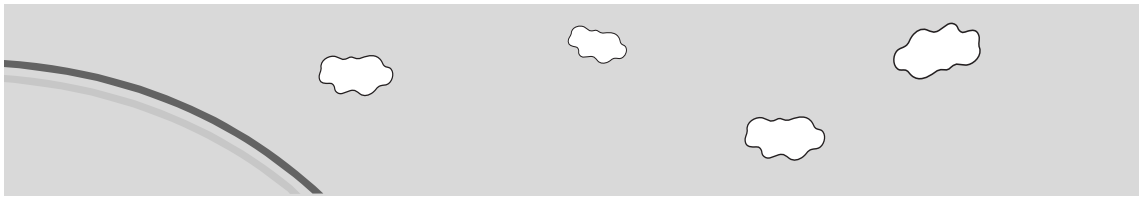
```
program Igra;

(* Подключение к программе функций - друзей жука-художника. *)
uses Crt;
begin
    Write("Таня");
    Write(" пишет программу.");
end.
```

Компьютер запыхтел, и мы прочли на экране:

```
Таня пишет программу.
```

Мне тоже захотелось, чтобы адаптер написал что-нибудь обо мне. Я сел за компьютер и дописал в программу свою строчку:

```
program Igra;

(* Подключение функций - друзей жука-художника. *)
uses Crt;
```



```
begin
    Write("Таня");
    Write(" пишет программу.");
    Write("Ваня тоже пишет программу.");
end.
```

На экране мы увидели:

```
Таня пишет программу.Ваня тоже пишет программу.
```

— Почему адаптер написал все слитно и не отделил мою строчку от Таниной? — спросил я.

— Но ты же не поставил пустой символ, который называется *пробелом*, перед словом «Ваня». Поэтому адаптер написал два слова слитно. Обрати внимание на то, как написана предыдущая строчка, и ты поймешь свою ошибку.

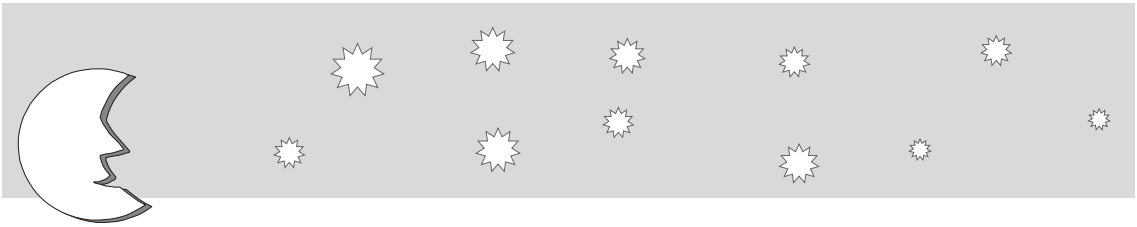
— А почему адаптер сам не отделил наши строчки? — поинтересовался я.

Папа объяснил:

— Функция `Write` принесла в память адаптера строчку без пробела. Какую строчку принесла, такую он и срисовал на экран дисплея. Адаптер пишет все подряд, пока не дойдет до границы экрана, и только после этого начинает писать текст с новой строчки. Если хочешь, чтобы текст начинался с новой строки, нужно вызвать другую функцию. Ее зовут `WriteLn`, что означает «записать строку и перейти на новую строку экрана».

Папа исправил нашу программу:

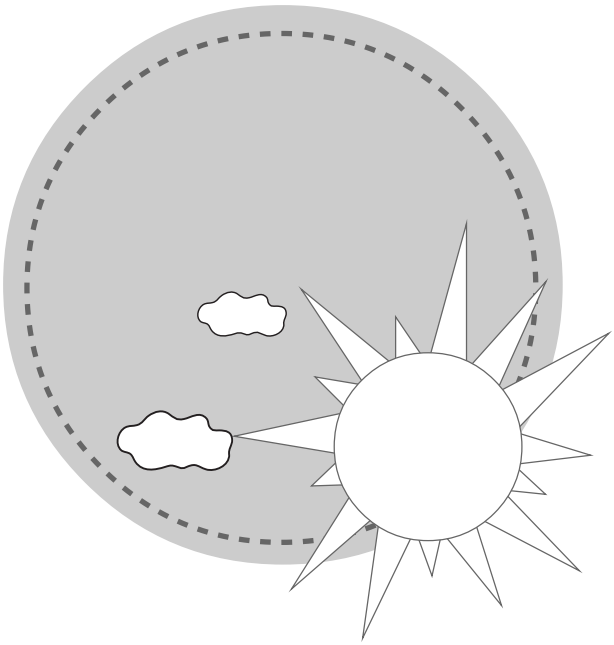
```
program Igra;
(* Подключение функций - друзей жука-художника. *)
uses Crt;
begin
    Write("Таня");
    WriteLn(" пишет программу.");
    WriteLn("Ваня тоже пишет программу.");
end.
```



И мы увидели свои строчки, написанные раздельно:

```
Таня пишет программу.
Ваня тоже пишет программу.
```

Я подумал: у нашего друга цветной адаптер, вот бы сделать наши строчки разноцветными. А как?



ГЛАВА 5.

НАШИ СТРОЧКИ НА ЭКРАНЕ СТАНОВЯТСЯ РАЗНОЦВЕТНЫМИ

На экране появился смешной человечек — функция `TextColor`.

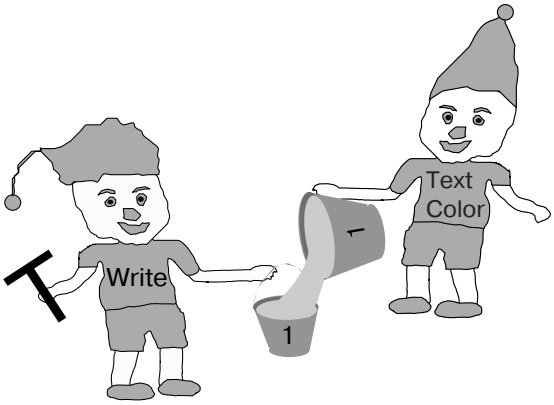
В переводе с английского TextColor означает «установить цвет текста». В руках у человечка было ведерко. Таня громко рассмеялась и спросила:

— А зачем этой функции ведерко?

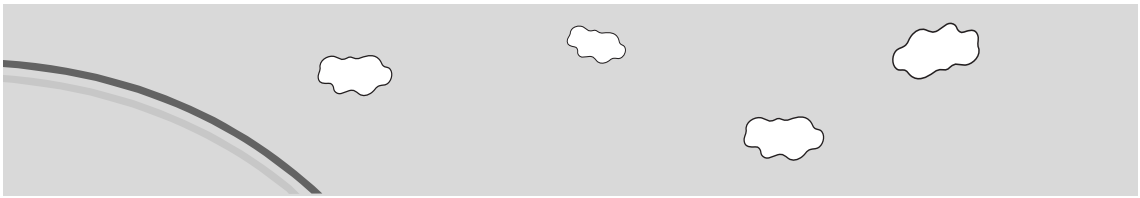
Папа объяснил, что все цвета у компьютера пронумерованы. И в ведерке у TextColor лежит номер цвета, который дают функции перед заданием. Например, у черного цвета — 0, у синего — 1, у зеленого — 2, а у красного — 4. Чтобы сделать цвета ярче, необходимо добавить к нужному номеру цвета число 8. А чтобы выбранный цвет мигал на экране, как лампочка на новогодней елке, надо добавить число 128.

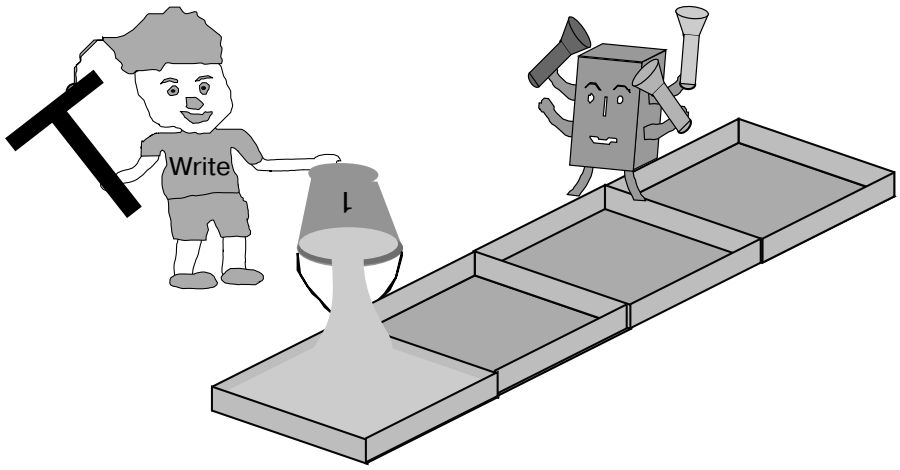
— Какое же задание у функции TextColor? — спросил я. — Она ходит и раскрашивает буквы?

— Нет, функция TextColor дает свою краску функции Write каждый раз, когда та должна что-нибудь написать на экране, — ответил папа.

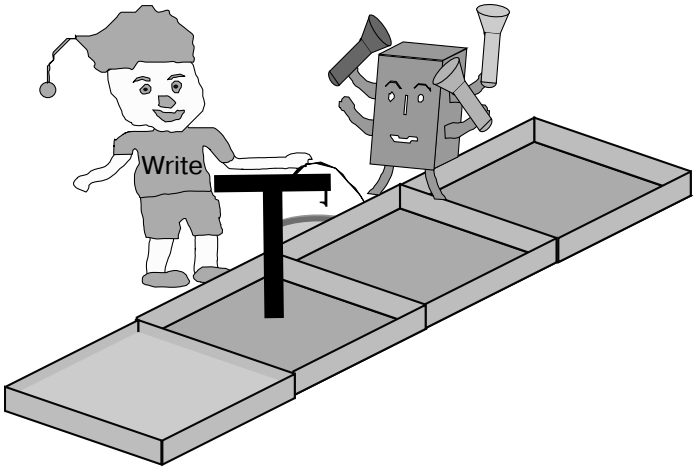


— После этого функция Write несет свои буквы и краску в ведре в память жука-художника. Сначала она наливает краску в ящичек для цвета буквы.





А затем кладет в следующий ящичек букву. Теперь жук-художник будет знать, каким цветом надо нарисовать каждую из букв на экране.



Таня воскликнула:

— Пусть человечек покрасит мою строчку синей краской!

А я захотел, чтобы моя строчка была ярко-красной и мигала на экране. Мы дописали программу:

```
program Game;

uses Crt;

begin
 TextColor(1); (* Синяя краска с номером 1. *)
 Write("Таня");
 WriteLn(" пишет программу.");
 TextColor(4+8+128); (* Красная краска с номером 4. Краска яркая и мигает. *)
 WriteLn("Ваня тоже пишет программу.");
end.
```

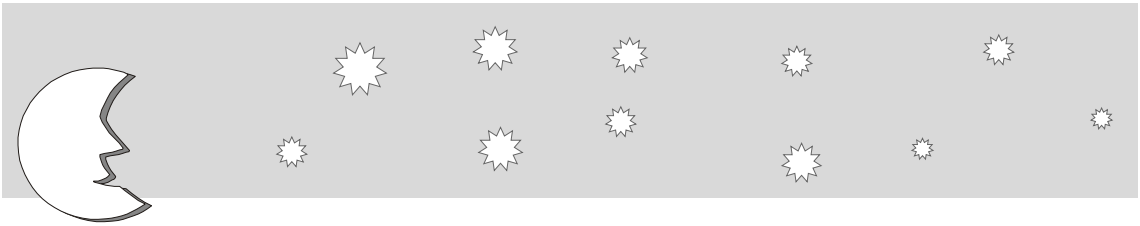
Потом запустили ее. И компьютер показал на экране разноцветные строчки. Они были очень красивые.

Как работают функции, было понятно, но почему они используют только четыре цвета? Ведь их гораздо больше. Например, желтый, фиолетовый, коричневый. Папа рассказал, что черный, синий, зеленый и красный — это основные цвета. А остальные можно получить, смешивая их. В программе цвета складывают, вот так: `TextColor(1+4)`. И если функцию `TextColor` вызвали с параметром `(1+4)`, то адаптер смешивает краски: первую (синюю) с четвертой (красной). А получается фиолетовая.

— Здорово! — закричали мы. — Как много цветов и оттенков можно сделать!

— Ваня, давай попробуем написать наши строчки разными красками, — предложила Таня.

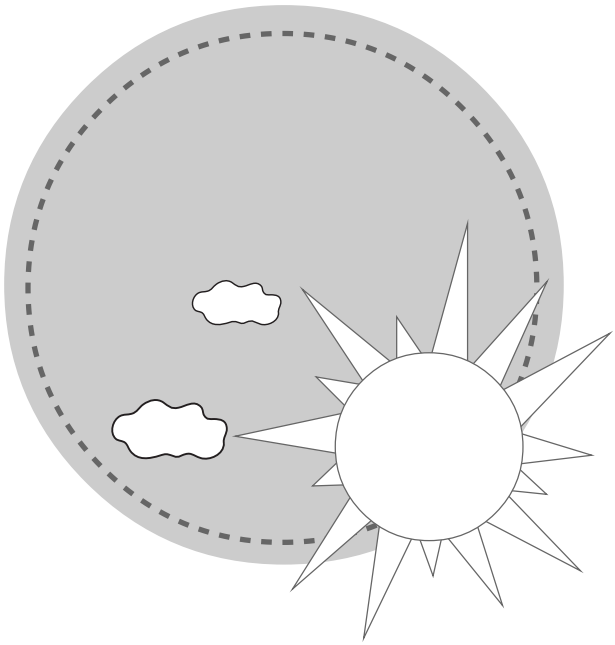
Я согласился, и мы стали складывать разные цвета. А компьютер на экране показывал строчки, разрисованные очень красивыми красками. Таких у нас еще не было — ведь обычные краски не могут мигать, как лампочки на елке. Папа был доволен компьютером. И рассказал, что наш друг с помощью дисплейного адаптера умеет писать буквы латинского, русского и других алфавитов, а также разные значки. А еще он может



рисовать прямоугольные рамки и смешные символы. Мы спросили у папы о смешных символах и о том, как узнать, какие символы есть у компьютера.

Папа ответил:

— Все буквы и значки у компьютера пронумерованы от 1 до 256. Представьте восемь очень маленьких ящичков, похожих на ящички в памяти адаптера. Они плотно прижаты друг к другу. Так вот, каждый такой ящичек называется *битом*. Он самый маленький. А *байт* — это большой ящик, в котором лежат восемь ящичков-бит. То есть один байт равен восьми битам. Для написания одной буквы выделяется восемь бит, или один байт. А для целого числа потребуется шестнадцать бит, или два байта. А сейчас посмотрим все буквы и значки, какие есть у компьютера.



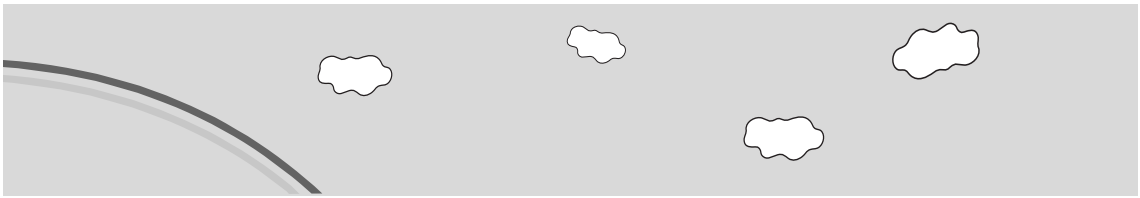
ГЛАВА 6.

КОМПЬЮТЕР ПОКАЗЫВАЕТ НАМ ВСЕ СВОИ БУКВЫ И ЗНАЧКИ

Папа дописал в конце нашей программы еще одну строчку:

```
program Game;
uses Crt;
begin
 TextColor(1); (* Синяя краска с номером 1. *)
 Write("Таня");
 WriteLn(" пишет программу.");
 TextColor(4+8+128); (* Красная краска с номером 4. Краска яркая и мигает. *)
 WriteLn("Ваня тоже пишет программу.");
 Write(Chr(1));
end.
```

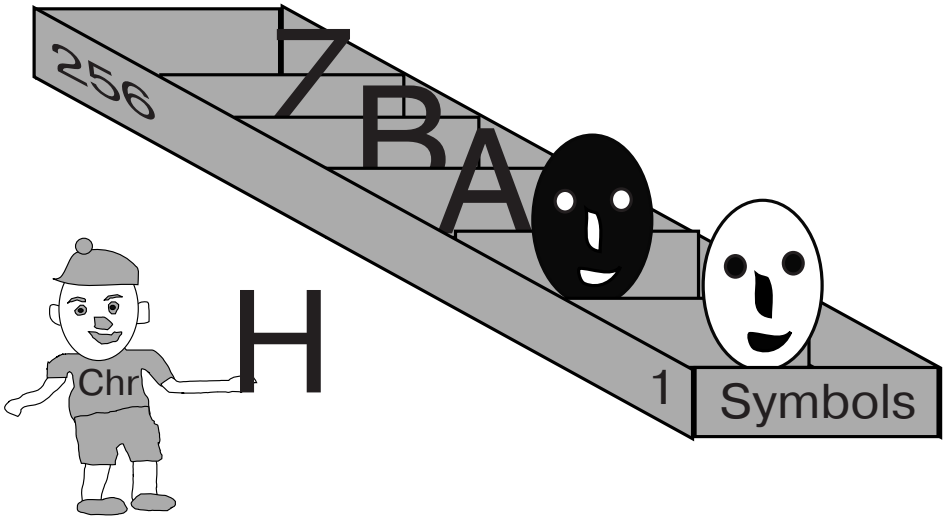
— В программе вы видите еще одну функцию — Chr. Это сокращение слова Character, что в переводе



означает «символ». Функции Chr дают параметр — число 1, а она приносит символ (букву или значок) с таким номером.

— Но как же функция Chr узнает, какую букву нужно принести? — спросил я папу.

— Ваня, я же рассказывал, что все буквы в памяти компьютера пронумерованы и каждая лежит в своем ящичке, как в шкафу-таблице. Функция Chr легко находит нужную букву, берет ее и приносит.



— Куда же она ее приносит? — поинтересовались мы.

— Функция Chr может сразу передать букву другой функции, например Write, или положить ее в память компьютера на какое-то время. А кто-нибудь потом возьмет ее оттуда. Об этом вы узнаете, когда познакомитесь с переменными. В программе строчка Write(Chr(1)) означает, что функция Chr нашла в таблице у компьютера букву с номером 1 и отдала ее функции Write.

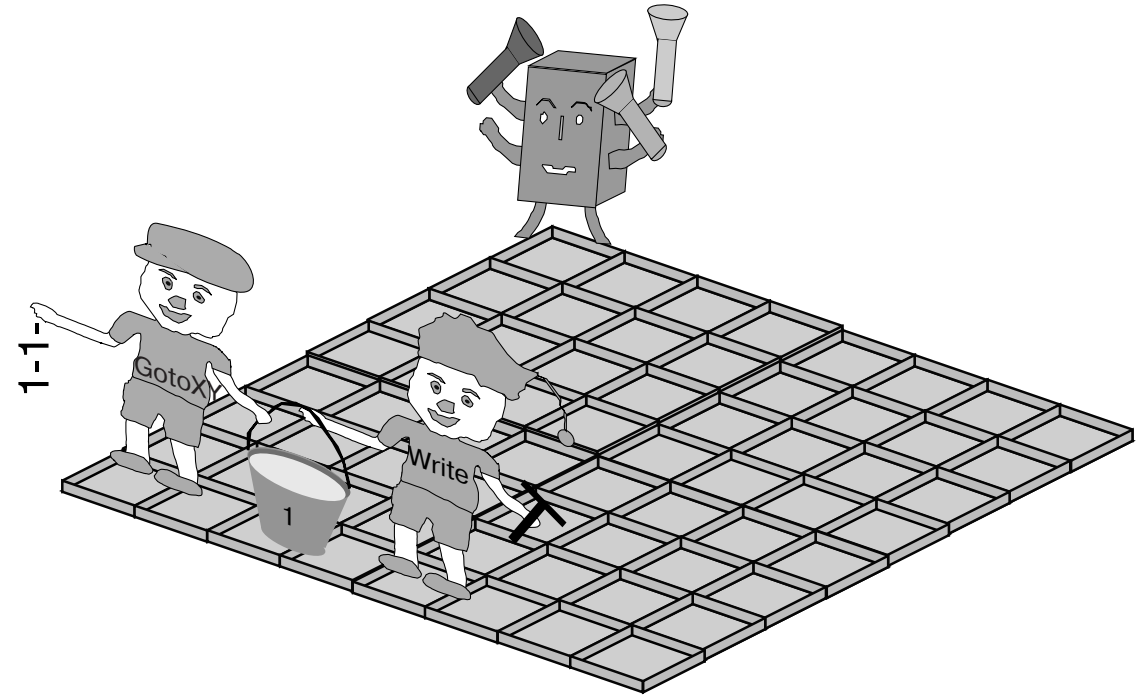
Таня предложила посмотреть на букву с номером 2. В программе вместо единицы мы поставили два, вот так: `Write(Chr(2))`.

Как же мы удивились, когда компьютер показал вместо буквы смешную рожицу! Оказывается, под номерами хранятся не только буквы, но и забавные значки. Мы задавали в программе функции Chr разные номера и с любопытством ждали, что появится на экране.

Я подумал, как ловко функции `Write` и `Chr` помогают адаптеру писать на экране. Но интересно узнать, умеет ли `Write` раскладывать буквы в памяти адаптера не с первой строчки, а с последней. Я спросил об этом папу.

— Функция `Write` раскладывает буквы строго по порядку, — ответил он, — начиная с последней буквы, которую раньше клала в память адаптера. Сама она ходить не умеет, поэтому ей помогает еще одна функция. Ее зовут `GotoXY` — «отвести». Чтобы функция `GotoXY` отвела `Write` в нужное место, ей дают два параметра — два числа. Их называют *координатами*. Первое число — координата `X` — показывает, на сколько ящичков вправо относительно верхнего левого угла экрана отойдет `GotoXY` и отведет туда `Write`. А второе число — координата `Y` — показывает, на сколько ящичков вниз спустится `GotoXY` вместе с `Write`. Например, самый первый ящичек памяти дисплейного адаптера расположен в верхнем левом углу экрана. Он имеет координаты `(1, 1)`. Если функции `GotoXY` дать координаты `(2, 2)`: `GotoXY(2, 2)`, то она отведет `Write` сначала на одну клетку вправо, а затем на одну клетку вниз.

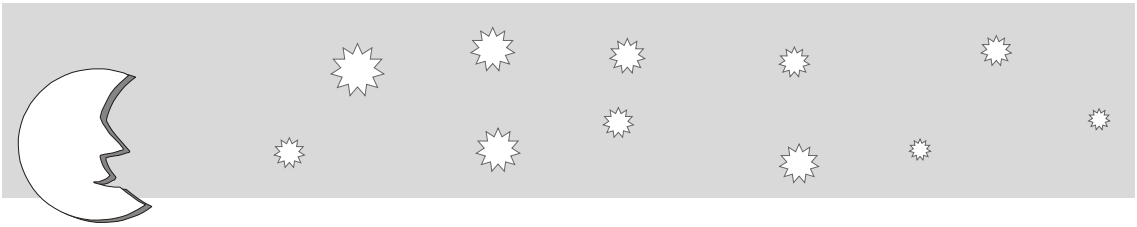
Тане захотелось поскорее поводить человечка `Write`, и она попросила об этом папу. Мне тоже было интересно, как `Write` будет относить буквы в ящички памяти адаптера, которые соответствуют разным клеткам на



экране. Папа предложил каждому из нас выбрать место для своей строчки на экране дисплея. Таня выбрала середину экрана, а я решил написать строчку внизу.

Наша программа стала вот такой:

```
program Game;

(* Подключение функций - друзей жука-художника. *)
uses Crt;

begin
    (* Эти строчки программы написала Таня. *)
    TextColor(1);        (* Синий цвет. *)
    GotoXY(12,12);       (* Середина экрана. *)
    Write("Таня");
    WriteLn(" пишет программу.");

    (* Эти строчки программы написал Ваня. *)
    TextColor(4+8+128);  (* Красный цвет. *)
    GotoXY(1,24);        (* Нижняя часть экрана. *)
    WriteLn("Ваня тоже пишет программу.");
    Write(Chr(1));
end.
```



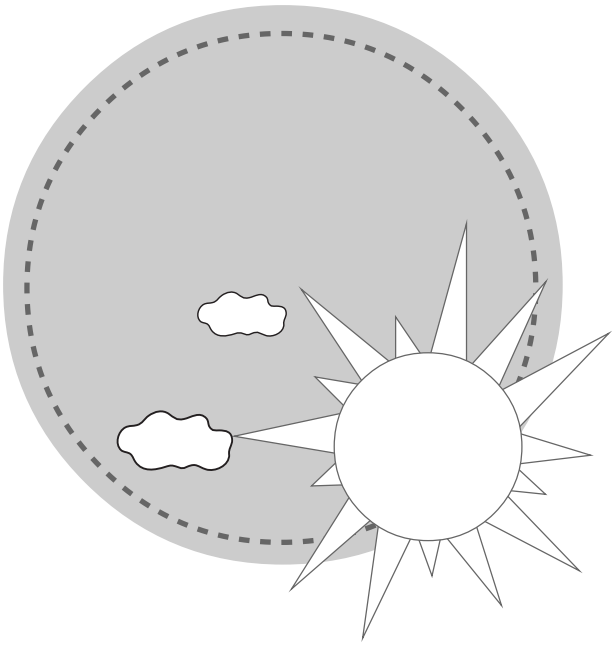
Тут заговорил компьютер:

— Ребята, предлагаю игру «Вопросы и ответы». Я спрашиваю, а вы отвечаете. Посмотрите на экран: там есть подсказки.

На экране мы увидели своих знакомых. Человечки весело подмигивали. Каждый что-то держал в руках. Мы стали присматриваться к человечкам, а компьютер начал задавать вопросы:

Отгадай, кто на рисунке
носит числа, буквы в сумке? (`Chr`, `Write`, `WriteLn`)
Кто же краску разливает
и цвета соединяет? (`TextColor`)
Кто умеет отводить,
вправо, влево, вниз ходить? (`GotoXY`)
Кто рисует? Кто читает? (адаптер) (`Chr`)
В моей памяти считает? (`Chr`)

Мы отвечали без труда, потому что там действительно были подсказки. Как интересно играть с компьютером! А что если придумать сказку и запрограммировать ее?



ГЛАВА 7.

МЫ СОЧИНЯЕМ СКАЗКУ ПРО АФРИКУ

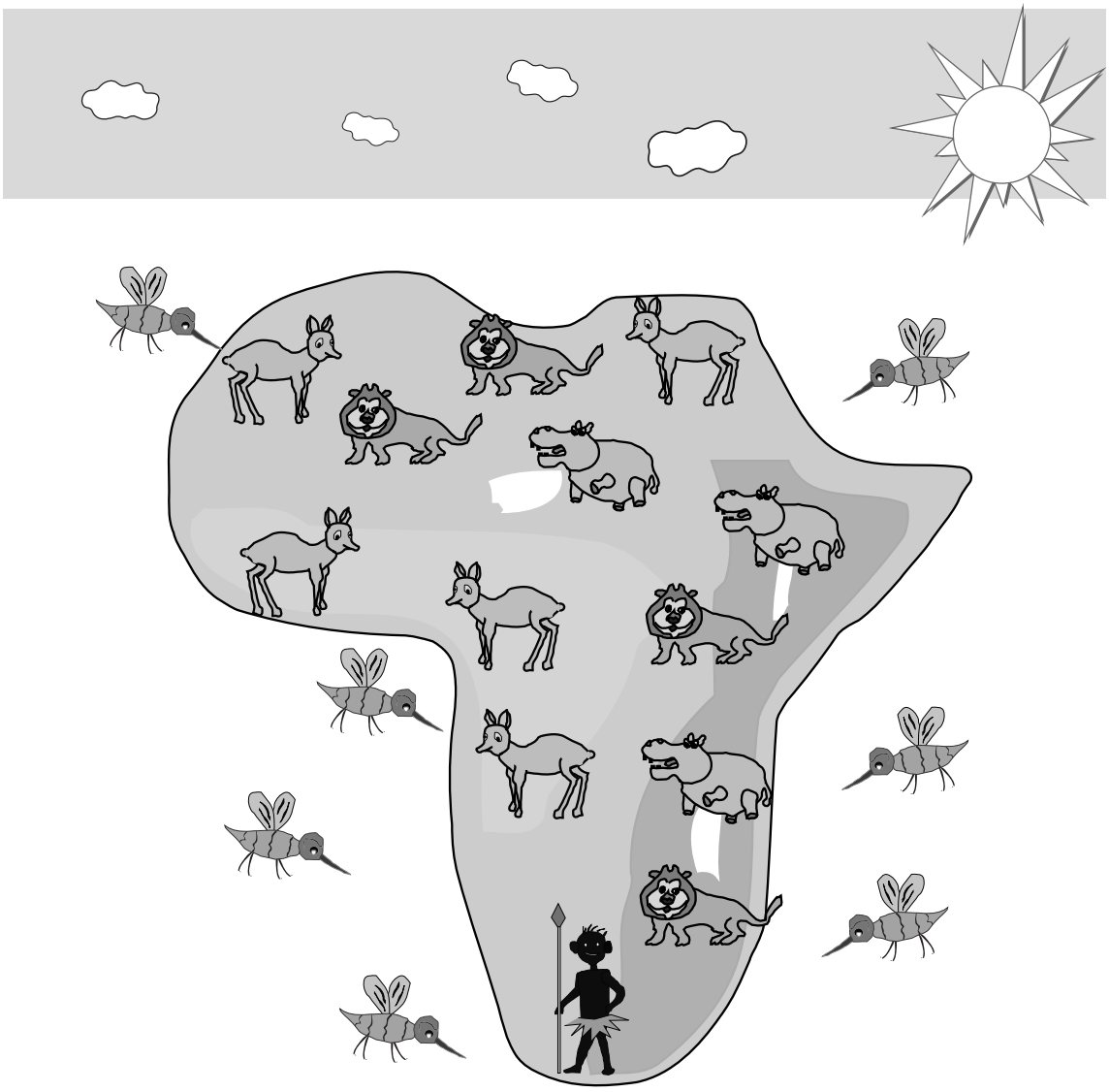
Таня обрадовалась:

— Ура! Давайте придумаем сказку про Африку!

Таня знала много сказок про нее. Смотрела мультфильмы и разные передачи по телевизору. Мы с папой даже не успели ответить: нас опередил компьютер. Он включил веселую музыку и сказал:

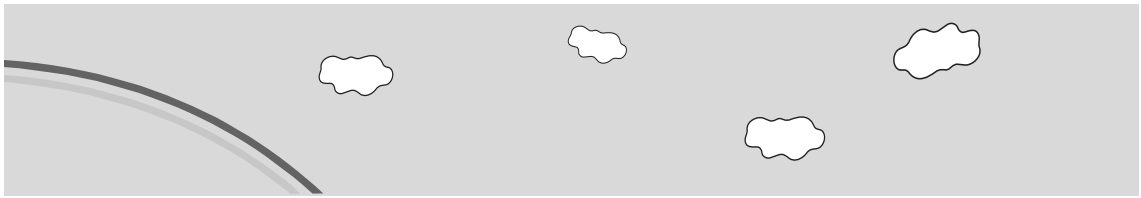
— Ребята, я покажу вам Африку и расскажу, что о ней написано в энциклопедиях.

Мы увидели в Африке жирафов, гиен, гепардов, антилоп, бегемотов, буйволов. Узнали, как живут слоны. Компьютер рассказал, что саванны немного похожи на наши поля. Только там много незнакомых растений, например слоновая трава, баобабы. Мы с Таней засмеялись,



когда узнали, что в Африке живет муха Цеце. Компьютер предупредил, что укус этой мухи смертелен. Когда Африка исчезла с экрана, мы захлопали в ладоши. Как же здорово компьютер умеет показывать мультфильмы! Папа сказал:

— Ребята, я знаю интересную сказку про Африку. Жил-был в Африке негр. Он охотился в саванне на львов, но никогда не брал с собой ружья, потому что очень любил животных, даже самых злых. Ходить по саванне всегда опасно. Но добрый охотник умел быстро строить заграждения. И ни один лев не мог его съесть. Львы попадали за изгородь, а выбраться оттуда не могли. Потом охотник раздавал пойманных львов зоопаркам разных стран.



Нам сразу вспомнился наш зоопарк. Там тоже были львы. А еще мы их видели в цирке. Но как же запрограммировать сказку?

— Папа, а мы будем участвовать в этой сказке? — спросила Таня.

— Конечно.

Я поинтересовался:

— Кем мы будем?

— Если хотите, охотником, — ответил папа. — Пока вы только учитесь программировать и, конечно, не сможете сразу нарисовать на экране настоящего льва. Для начала обозначим всех участников сказки символами, ведь у компьютера их много.

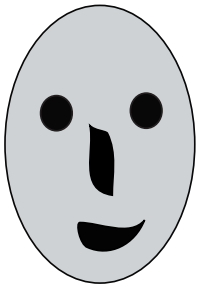
Я вспомнил, что у компьютера среди символов есть смешная черная рожица. Ею можно было бы обозначить негра-охотника. Папа согласился со мной.

А Тане очень хотелось увидеть на экране настоящего льва. И она расстроилась, что не сможет этого сделать.

Папа объяснил:

— Таня, ты пока учишься. И если ты хорошо выучишь язык программирования Паскаль, то сумеешь потом сама нарисовать настоящего льва.

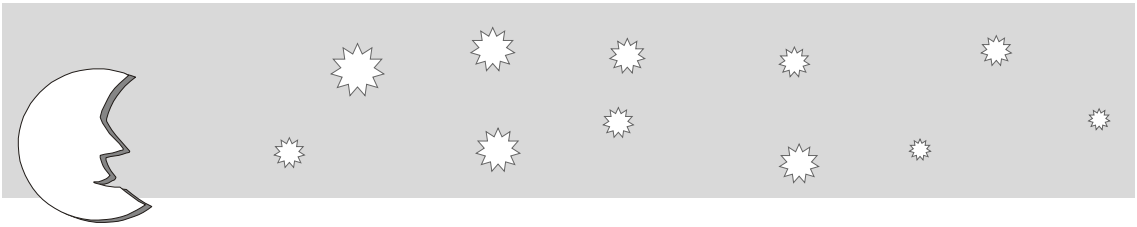
Вдруг компьютер показал на экране смешную желтую рожицу.



Таня улыбнулась и сказала:

— Ой, как она похожа на льва! Пусть будет львом.

— Вот и отлично, ребята, — заметил папа. — Главные герои сказки уже обозначены.



Он предложил одну часть экрана дисплея покрасить в зеленый цвет и назвать саванной, а другую сделать коричневой, это будет пустыня.

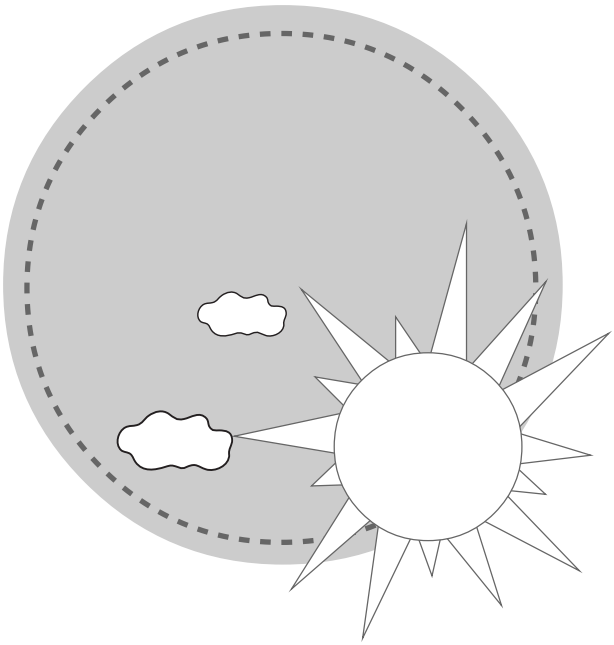
— Очень важно поставить перед собой цель. Как вы думаете, что для нашего охотника самое главное?

— Не убить льва, а поймать его.

— Правильно. А как вы назовете свою игру?

Мы хором закричали:

— «Африка»!



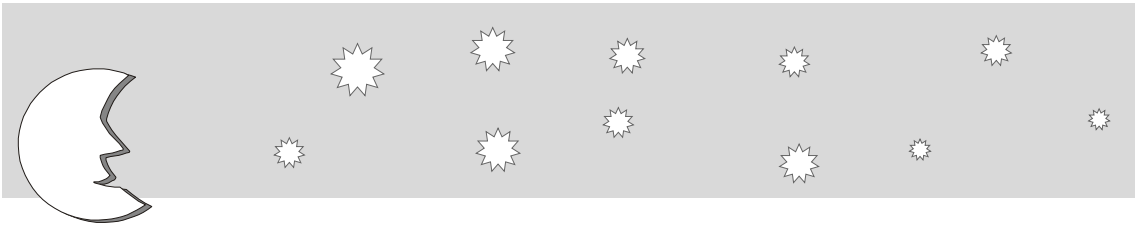
ГЛАВА 8.

НАЧИНАЕМ ПРОГРАММИРОВАТЬ ИГРУ «АФРИКА» НА ЯЗЫКЕ ПАСКАЛЬ

Папа напомнил, как обозначен лев. Это желтая рожица. Льву соответствует символ с номером 2.

А желтый цвет получится от смешения двух ярких красок — красной и зеленой. Сначала мы решили научить льва бегать по экрану дисплея. Папа написал новую программу под названием Africa1:

```
program Africa1;

(* Подключение функций - друзей жука-художника. *)
uses Crt;
```



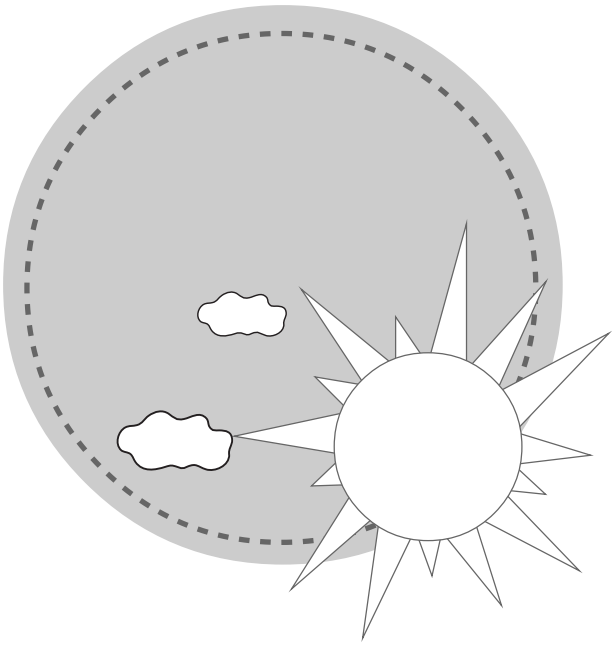
```
begin
    ClrScr;              (* Очистим экран. *)
    TextColor(2+4+8);    (* Нальем для функции Write *)
                         (* яркой желтой краски. *)
    GotoXY(1, 1);        (* Отведем функцию Write *)
                         (* в левый верхний угол экрана. *)
    Write(Chr(2));       (* Нарисуем символ с номером 2 - *)
                         (* желтого льва. *)
end.
```

Потом он передал программу компьютеру. Наш друг тут же показал на экране символ желтого льва.

— Когда же он побежит? — спросила Таня.

— Чтобы лев побежал, нужно стереть его изображение на старом месте, а на новом нарисовать, — ответил папа.

Это, наверное, так же, как оживают картинки в мультфильме. Их нужно очень быстро показывать друг за другом. А как запомнить старое место, где сидел лев, я не знал. Но мы же забыли про память компьютера! Ее-то и нужно использовать! И папа предложил нам попросить у компьютера два ящичка в памяти для хранения координат льва на экране.



ГЛАВА 9.

ПАПА РАССКАЗЫВАЕТ О ПЕРЕМЕННЫХ И КОНСТАНТАХ

Папа дописал программу:

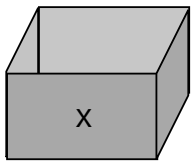
```
program Africa2;

(* В программе будут использоваться функции-помощники. *)
uses Crt;

var x: integer;     (* Ящички-переменные в памяти компьютера, *)
var y: integer;     (* где мы будем хранить координаты льва на экране. *)

const Yellow = 14; (* Положим в ящичек-константу Yellow число 14. *)
begin
   (* Функция ClrScr очистит экран. *)
   ClrScr;
   (* Функция TextColor положит желтую краску в ведерко функции Write. *)
```

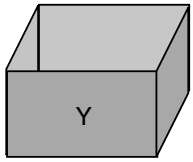
```
  TextColor(Yellow);

  (* Функция GotoXY отведет функцию Write в верхний левый угол экрана. *)
  GotoXY(1, 1);

  (* Функция Write нарисует символ льва в верхнем углу экрана. *)
  Write(Chr(2));
end.
```

Мы сразу заметили, что в программе Africa2 появились непонятные слова. Что они означают? И где лежат два ящичка из памяти компьютера?

— Ребята, не пугайтесь новых слов, — сказал папа. — Давайте по порядку. Нам встретилось незнакомое ключевое слово var. Так сокращенно записывается слово variable, что в переводе означает «переменная». Ящички из памяти компьютера мы также будем называть *переменными*. Представьте, что на ящичках есть названия.

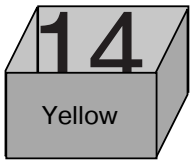
```
var x: integer; (* Ящички-переменные в памяти компьютера, *)
var y: integer; (* где мы будем хранить координаты льва на экране. *)
```



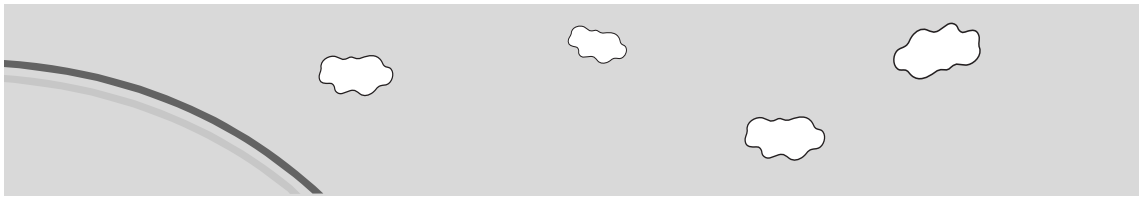


На одном ящичке надпись x, а на другом — y. В них можно класть числа и забирать их оттуда. Есть ящички, из которых разрешается только брать числа, а класть в них ничего нельзя. Такие ящички называются *константами* (постоянными). В программе ящички-константы обозначаются словом const. К примеру, в ящичке-константе с названием Yellow (Желтый) должно лежать число 14. А записывается это вот так:

```
(* В ящичке-константе Yellow число будет лежать всегда. *)
const Yellow = 14;
```

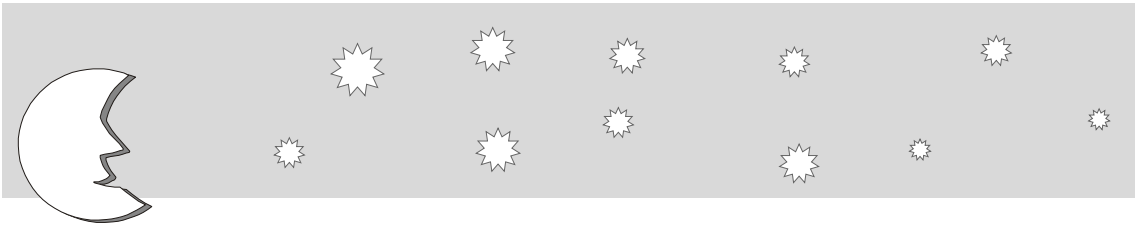


Если попробовать положить в такой ящичек что-нибудь другое, компьютер предупредит о том, что мы ошиблись.



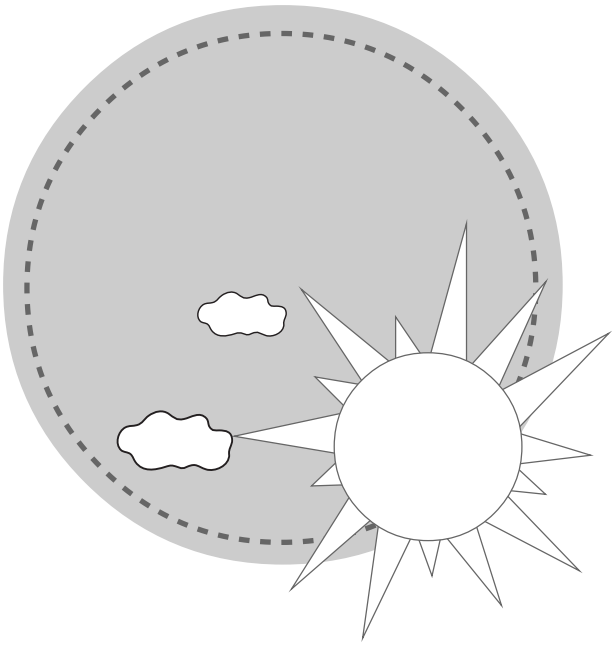
Потом мы узнали еще одно ключевое слово: integer (целое). Оно означает, что в ящички-переменные можно класть только целые числа. При попытке положить букву или не целое число (например, 1/2) компьютер сообщит об ошибке. Папа сказал, что совсем необязательно описывать переменные x и y в программе отдельно друг от друга. Целые числа разрешается описать вместе, разделяя их запятыми. И слово const тоже не нужно каждый раз повторять. Можно отделять константы точкой с запятой, вот так:

```
program Africa3;

(* Используем функции - друзей жука-художника. *)
uses Crt;


(*********************************)
(* Константы для главных цветов. *)
(*********************************)
const
Black     = 0;      (* Черный. *)
Blue      = 1;      (* Синий. *)
Green     = 2;      (* Зеленый. *)
Red       = 4;      (* Красный. *)
Intensity = 8;      (* Яркий. *)
Blink     = 128;    (* Мигающий. *)

(***********************************)
(* Константы для смешанных цветов. *)
(***********************************)
const
Yellow  = Red + Green + Intensity;
Brown   = Red + Green;
Gray    = Red + Green + Blue;
White   = Red + Green + Blue + Intensity;
Magenta = Red + Blue;
```



```
(***************************)
(* Константы для символов. *)
(***************************)
const
HunterSym = Chr (1); (* Символ для негра-охотника. *)
LionSym = Chr (2);   (* Символ для льва. *)

(* Опишем ящички-переменные x и y для координат льва. *)
var
x, y: integer;

begin
    (* Функция ClrScr очистит экран. *)
    ClrScr;

    (* Функция TextColor нальет желтую краску в ведерко функции Write. *)
    TextColor(Yellow);

    (* Функция GotoXY отведет функцию Write в левый верхний угол экрана. *)
    GotoXY(1, 1);

    (* Функция Write нарисует символ льва в левом верхнем углу экрана. *)
    Write(LionSym);
end.
```

Таня обрадовалась, что сумеет теперь что-нибудь положить в ящички. А компьютер поможет! Она выбрала переменную x, а мне досталась переменная y.



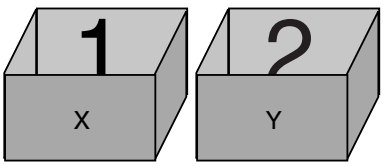
ГЛАВА 10.

МЫ ЗНАКОМИМСЯ С ОПЕРАТОРАМИ ПРИСВАИВАНИЯ И СЛОЖЕНИЯ

Папа объяснил:

— В ящички числа кладут с помощью *оператора присваивания*. Он обозначается двумя символами: двоеточием и равно, которые нужно обязательно писать рядом, вот так :=. В программе его легко узнать:

```
x := 1; (* В переменную x положим число 1. *)
y := 2; (* В переменную y положим число 2. *)
```

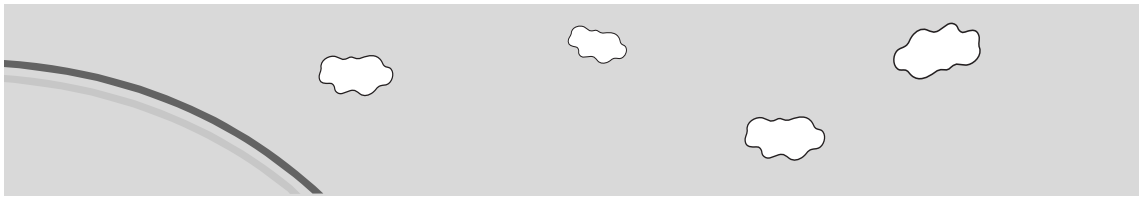


— А каким оператором достают числа из ящичков? — поинтересовался я.

— Сделать это намного проще. Если в программе встречаются

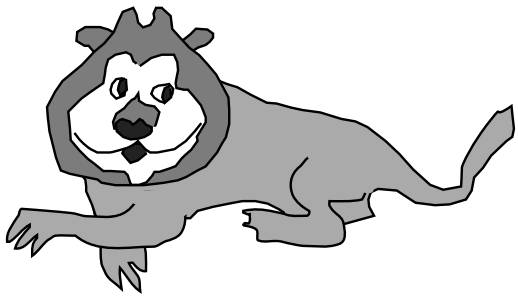
названия переменных, значит, из ящичка в этом месте программы достают буквы или цифры. Посмотрите:

```
program Africa4;

(* Используем функции - друзей жука-художника. *)
uses Crt;

(*********************************)
(* Константы для главных цветов. *)
(*********************************)
const
Black     = 0;      (* Черный. *)
Blue      = 1;      (* Синий. *)
Green     = 2;      (* Зеленый. *)
Red       = 4;      (* Красный. *)
Intensity = 8;      (* Яркий. *)
Blink     = 128;    (* Мигающий. *)

(***********************************)
(* Константы для смешанных цветов. *)
(***********************************)
const
Yellow  = Red + Green + Intensity;
Brown   = Red + Green;
Gray    = Red + Green + Blue;
White   = Red + Green + Blue + Intensity;
Magenta = Red + Blue;

(*****************************)
(* Константы для символов. *)
(*****************************)
const
HunterSym = Chr (1); (* Символ для негра-охотника. *)
LionSym = Chr (2);   (* Символ для льва. *)

(* Опишем ящички-переменные x и y для координат льва. *)
var
x, y: integer;

begin
    ClrScr; (* Очистим экран. *)

    (* Положим в переменные x и y числа 1 и 2. *)
    x := 1;
    y := 2;
```



```
    (* Льва нарисуем желтым цветом. *)
    TextColor(Yellow);

    (* Достанем числа из ящичков-переменных x и y *)
    (* и передадим их функции GotoXY. *)
    GotoXY (x, y);

    (* Нарисуем на экране льва в координатах (1, 2). *)
    Write(LionSym);
end.
```

Понятно, что хранить в ящичках-переменных числа и буквы удобно: не запутаешься, где что лежит, у ящичков-то есть названия. И мы уже положили в них координаты льва на экране. Лев сидел смирно.



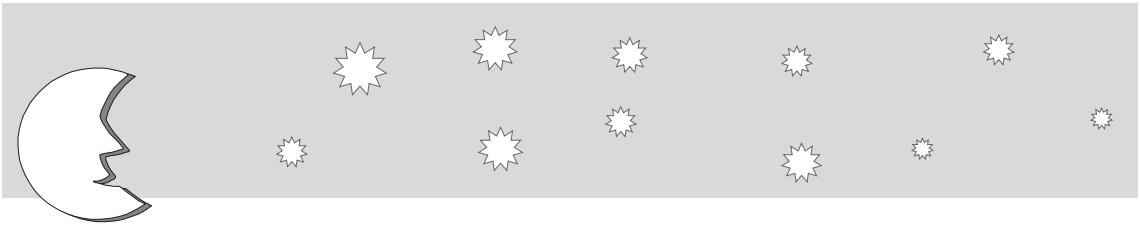
— Теперь давайте поучим льва ходить, — предложил папа. — Нам поможет *оператор сложения*. В программе он обозначается плюсом (+).

В школе мы давно научились складывать разные числа, поэтому быстро догадались, как лев сможет сделать шаг: его координату нужно увеличить на единицу. Например, к числу в ящичке-переменной `x` прибавить единицу, вот так:

```
(* Достанем из ящичка-переменной x число, прибавим к нему 1 *)
(* и положим результат обратно в ящичек-переменную x. *)
x := x + 1;
```

— Но чтобы передвинуть льва на новое место, нужно стереть его на старом. Если этого не сделать, то на экране будет два льва, — напомнил папа и дописал программу:

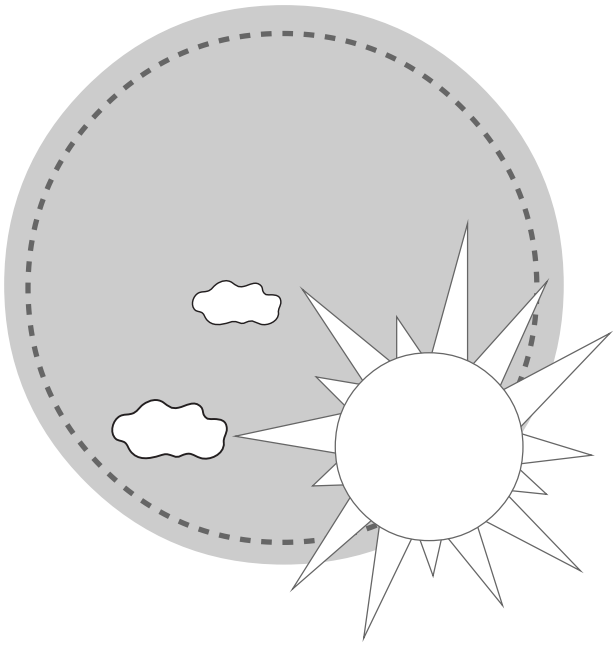
```
program Africa5;

(* Используем функции - друзей жука-художника. *)
uses Crt;

(*********************************)
(* Константы для главных цветов. *)
(*********************************)
const
Black     = 0;      (* Черный. *)
Blue      = 1;      (* Синий. *)
Green     = 2;      (* Зеленый. *)
Red       = 4;      (* Красный. *)
Intensity = 8;      (* Яркий. *)
Blink     = 128;    (* Мигающий. *)

(***********************************)
(* Константы для смешанных цветов. *)
(***********************************)
const
Yellow  = Red + Green + Intensity;
Brown   = Red + Green;
Gray    = Red + Green + Blue;
White   = Red + Green + Blue + Intensity;
Magenta = Red + Blue;

(*****************************)
(* Константы для символов. *)
(*****************************)
const
HunterSym = Chr (1); (* Символ для негра-охотника. *)
LionSym = Chr (2);   (* Символ для льва. *)

(* Опишем ящички-переменные x и y для координат льва. *)
var
x, y: integer;
```



```
begin
    ClrScr; (* Очистим экран. *)

    (* Положим в переменные x и y числа 1 и 2. *)
    x := 1;
    y := 2;

    (* Льва нарисуем желтым цветом. *)
    TextColor(Yellow);

    (* Достанем числа из ящичков-переменных x и y *)
    (* и передадим их функции GotoXY. *)
    GotoXY (x, y);

    (* Нарисуем на экране льва в координатах (1, 2). *)
    Write(LionSym);

    (* Отведем функцию Write на старое место льва. *)
    (* Сотрем изображение льва пустым символом - пробелом. *)
    GotoXY(x, y);
    Write(' ');

    (* Увеличим координаты в ящичках x и y на 1. *)
    x := x + 1;
    y := y + 1;

    (* Отведем функцию Write на новое место льва. *)
    GotoXY (x, y);

    (* Нарисуем льва на новом месте. *)
    Write(LionSym);
end.
```

В программе был оператор сложения. Он помог льву сделать один шаг. Мы заметили, как лев передвинулся.

Я подумал: «А вдруг льву захочется сделать сто или тысячу шагов? Тогда в нашей программе будет несколько тысяч строк. И нам придется писать их целую неделю!»



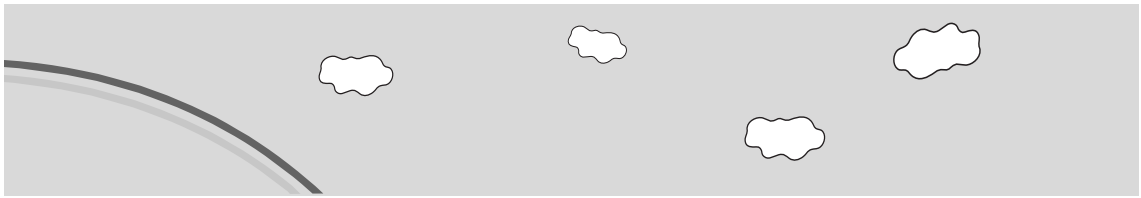
ГЛАВА 11.

ОПЕРАТОР ЦИКЛА ПОМОГАЕТ ЛЬВУ БЕГАТЬ ПО САВАННЕ

Папа успокоил нас:

— Ребята, программисты придумали оператор цикла. Его используют, когда нужно много-много раз повторить какие-нибудь действия в программе. У нас этот оператор научит льва ходить. — И дописал в программу новые строчки:

```
program Africa6;

(* Используем функции - друзей жука-художника. *)
uses Crt;

(*********************************)
(* Константы для главных цветов. *)
(*********************************)
const
Black     = 0; (* Черный *)
```



```
Blue      = 1;      (* Синий *)
Green     = 2;      (* Зеленый *)
Red       = 4;      (* Красный *)
Intensity = 8;      (* Яркий *)
Blink     = 128;    (* Мигающий *)

(***********************************)
(* Константы для смешанных цветов. *)
(***********************************)
const
Yellow  = Red + Green + Intensity;
Brown   = Red + Green;
Gray    = Red + Green + Blue;
White   = Red + Green + Blue + Intensity;
Magenta = Red + Blue;

(*****************************)
(* Константы для символов. *)
(*****************************)
const
HunterSym = Chr (1); (* Символ для негра-охотника. *)
LionSym = Chr (2);   (* Символ для льва. *)

(* Опишем ящички-переменные x и y для координат льва. *)
var
x, y: integer;

(* Ящичек-переменная для подсчета шагов льва. *)
var k : integer;
begin

    (* Очистим экран. *)
    ClrScr;

    (* Положим в переменные x и y числа 1 и 2. *)
    (* Это начальное положение льва. *)
    x := 1; y := 2;

    (* Достанем из ящичков-констант числа 2 и 4, *)
    (* сложим их, а результат передадим функции TextColor. *)
    (* Это цвет льва. *)
    TextColor(Yellow);
```

```
    (* Лев сделает тридцать тысяч шагов. *)
    for k := 1 to 30000 do
    begin
        (* Пробелом сотрем льва на старом месте. *)
        GotoXY(x, y);
        Write(' ');

        (* Вычислим новые координаты льва. *)
        x := x + 1;
        y := y + 1;

        (* Нарисуем льва на новом месте. *)
        GotoXY(x, y);
        Write(LionSym);
    end
end.
```

В программе слова `begin` и `end` использовались по два раза. Мы знали, что они обозначают начало и конец программы.

— Неужели мы написали еще одну программу? — спросил я папу.

— Нет, Ваня, это небольшая часть программы. Ключевые слова `begin` и `end` показывают, где начинается и кончается эта часть.

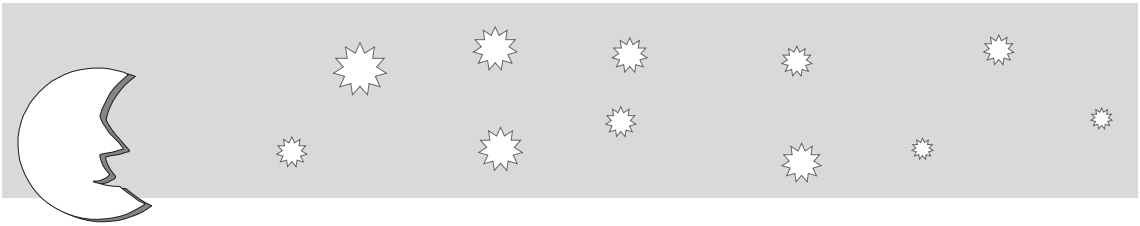
— А зачем нужны такие кусочки программы? — поинтересовалась Таня.

Папа улыбнулся:

— Для того чтобы повторять эти кусочки много-много раз. А сколько раз повторять, записано в строчке, которая начинается со слова `for` и стоит перед словом `begin`. Вот эта строчка:

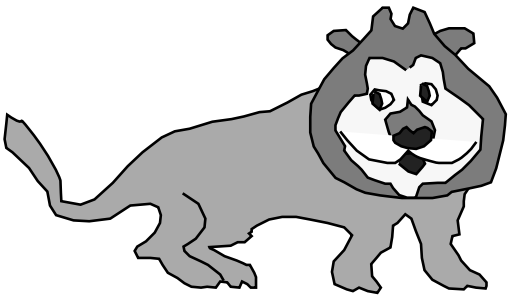
```
for k := 1 to 30000 do (* Для k от 1 до 30000 выполнить. *)
```

Здесь `k` — переменная. В ней будет храниться номер шага льва. Сначала в ящичек-переменную `k` кладут



единицу, а затем добавляют еще по одной единице, пока в нем не окажется число 30000. Тогда оператор цикла выполнит это действие в последний раз, и наш лев остановится. — Папа запустил программу.

На экране появился маленький лев.

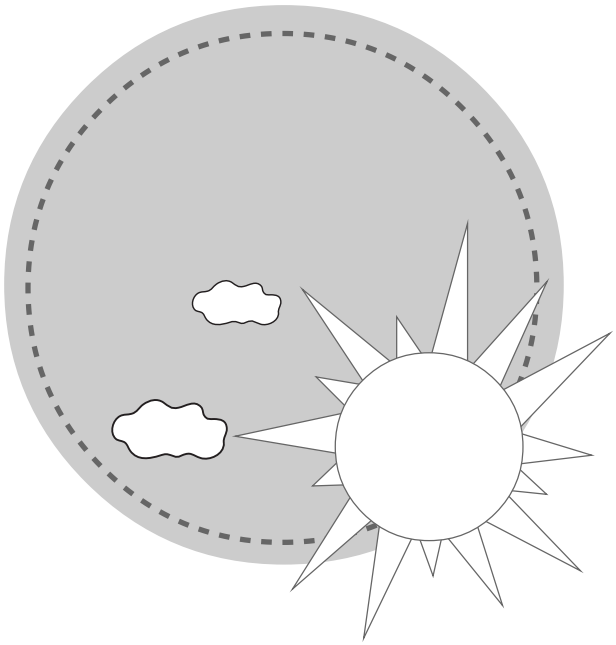


Лев так быстро бегал, что мы не успевали его разглядеть. Он добегал до нижней границы экрана, а потом начинал искать выход, бегая вдоль нее. Нам хотелось помочь льву вернуться назад. Но как это сделать, мы не знали.

Таня попросила папу:

— Пусть лев бегает помедленнее, чтобы мы успели его рассмотреть.

— Хорошо, — сказал папа.

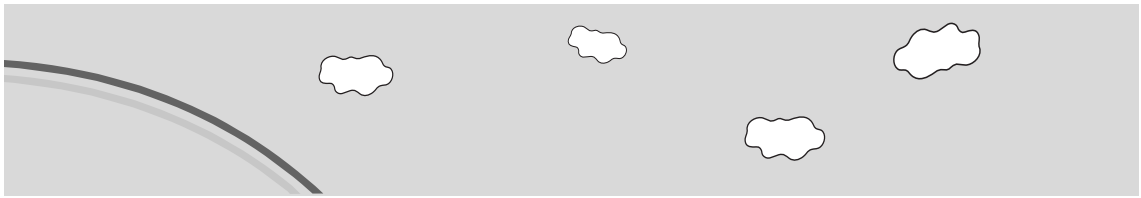


ГЛАВА 12.

УСЛОВНЫЙ ОПЕРАТОР НЕ ДАЕТ ЛЬВУ ЗАБЛУДИТЬСЯ В АФРИКЕ

— Наш лев еще не умеет определять границы экрана и не знает, как поворачивать назад в саванну. Но его можно научить. Для этого познакомимся с *операторами сравнения*. Они позволяют сравнивать числа. Операторы сравнения могут обозначаться вот такими знаками:

больше	>
меньше	<
равно	=
неравно	<>
меньше или равно	<=
больше или равно	>=



— Давайте сравним два числа, например 3 и 2.

Мы уже познакомились с оператором сложения. Если сложить 3 и 2, будет 5.

— А что может получиться, если сравнить 3 и 2?

Папа объяснил:

— Сравнение даст одно из двух значений: `true` или `false`, что в переводе означает «да» и «нет». Например, если написать вот так: 3 > 2 (3 больше, чем 2), то получится `true` (да), а если наоборот: 3 < 2 (3 меньше, чем 2), будет `false` (нет).

— А можно положить слова `true` или `false` в какие-нибудь ящички в памяти компьютера как целые числа? Ведь там есть много ящичков-переменных, — спросил я папу.

— Конечно, можно. Такие переменные названы *булевыми*, в честь английского математика Буля. А описать эти ящички-переменные для хранения `true` и `false` в программе можно следующим образом:

```
var b : boolean;
```

В таких переменных мы будем хранить результаты сравнения. А еще результаты сравнения можно использовать в *условном операторе*. С его помощью лев, прежде чем сделать очередной шаг, остановится и посмотрит по сторонам. Условный оператор имеет ключевые слова:

```
if ... then ... else ... (* если ... то ... иначе ... *)
```

— Давайте что-нибудь сравним в нашей программе, — предложила Таня.

— Но сначала нам нужно разобраться с условным оператором. Ответьте, что сделает вот такая программа:

```
if (3>2) then Write("3>2") else Write("3<2");
```

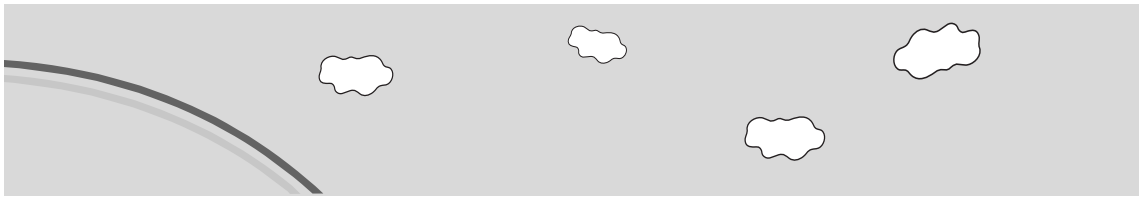
— Если число 3 больше, чем 2, то программа напишет на экране 3>2, в противном случае она напишет 3<2, — ответил я.

— Молодец, Ваня. Теперь наш лев не заблудится в саванне. Нам осталось только узнать, что такое *отрицательные числа*. Для обучения льва нам нужно попросить у компьютера память еще для двух переменных dx и dy. Числа, которые мы положим в ящичек, будут указывать льву, куда он должен бежать. При единице лев побежит вправо или вниз, а при минус единице — влево или вверх.

— Что же означает минус единица? — поинтересовались мы.

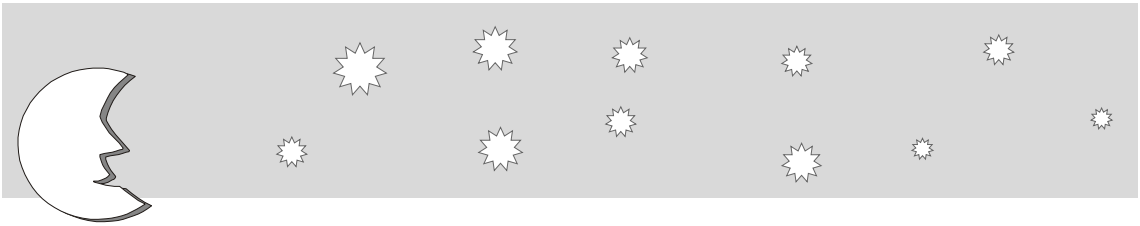
— Прибавлять к числу единицу со знаком минус то же самое, что вычитать единицу из этого числа. А вычитать из числа единицу со знаком минус — это все равно, что прибавлять к нему единицу. — И папа дописал в программу новые строчки:

```
program Africa7;

(* Используем функции - друзей жука-художника. *)
uses Crt;

(*********************************)
(* Константы для главных цветов. *)
(*********************************)
const
Black     = 0;      (* Черный. *)
Blue      = 1;      (* Синий. *)
Green     = 2;      (* Зеленый. *)
Red       = 4;      (* Красный. *)
Intensity = 8;      (* Яркий. *)
Blink     = 128;    (* Мигающий. *)
```



```
(***********************************)
(* Константы для смешанных цветов. *)
(***********************************)
const
Yellow  = Red + Green + Intensity;
Brown   = Red + Green;
Gray    = Red + Green + Blue;
White   = Red + Green + Blue + Intensity;
Magenta = Red + Blue;

(*****************************)
(* Константы для символов. *)
(*****************************)
const
HunterSym = Chr (1); (* Символ для негра-охотника. *)
LionSym = Chr (2);   (* Символ для льва. *)

(********************************)
(* Константы для границ экрана. *)
(********************************)
const
minScreenX     = 1; (* Левая. *)
minScreenY     = 1; (* Верхняя. *)
maxScreenX     = 80;(* Правая. *)
maxScreenY     = 24;(* Нижняя. *)

(***********************************)
(* Константы для направления льва. *)
(***********************************)
const
right   =  1; (* Вправо. *)
down    =  1; (* Вниз. *)
up      = -1; (* Вверх. *)
left    = -1; (* Влево. *)

(* Опишем ящички-переменные x и y *)
(* для хранения координат льва. *)
var x, y: integer;

(* Переменная для подсчета шагов льва. *)
var k : integer;
(* Переменная для замедления бега льва. *)
var i : integer;
```

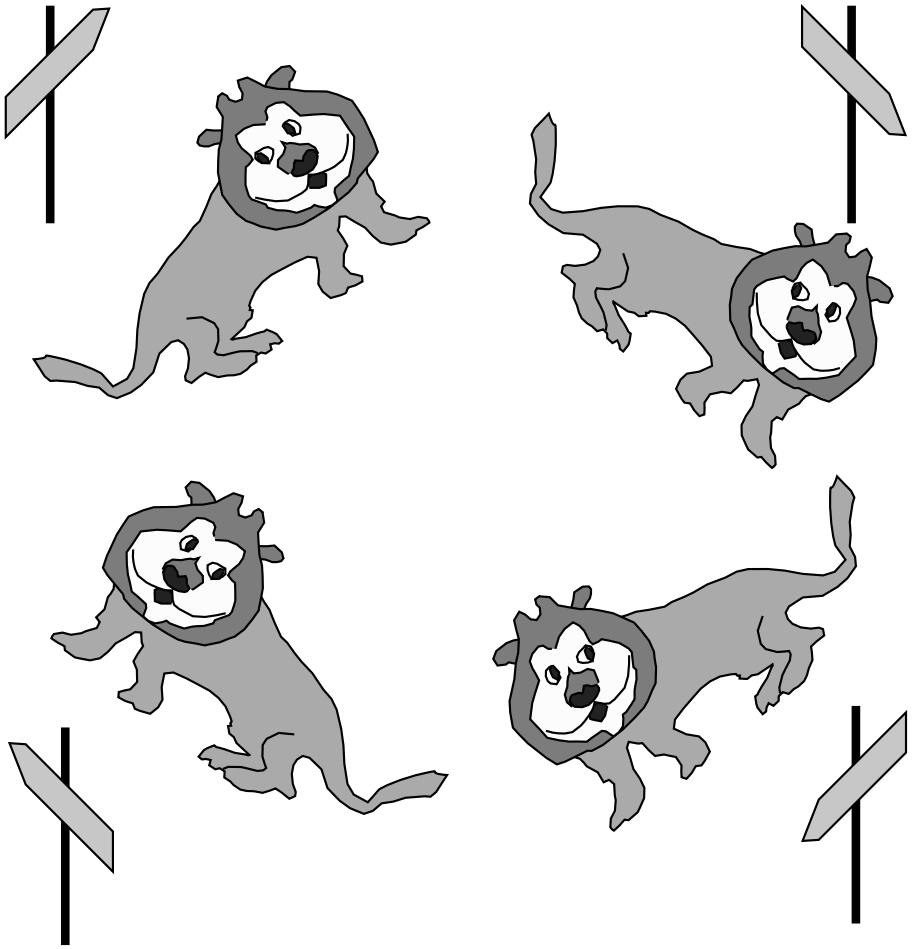
```
(* Переменные для хранения информации о направлении движения льва. *)
var dx, dy : integer;

begin
    (* Очистим экран. *)
    ClrScr;

    (* Положим в переменные x и y числа 1 и 2. *)
    (* Это будет начальное положение льва. *)
    x := 1;
    y := 2;

    (* Достанем из ящичков-констант числа 2 и 4, *)
    (* сложим их, а результат передадим функции TextColor. *)
    (* Это будет цвет льва. *)
    TextColor(Yellow);

    (* Сначала лев бежит направо и вниз. *)
    dx := right;
    dy := down;

    for k := 1 to 10000 do
    begin
        (* Чтобы лев бегал помедленнее, складываем числа в i. *)
        (* Ребята в это время успеют разглядеть льва. *)
        for i := 1 to 32000 do; (* Цикл от 1 до почти *)
        for i := 1 to 32000 do; (* самого большого *)
        for i := 1 to 32000 do; (* целого числа *)
        for i := 1 to 32000 do; (* компьютера. *)

        if x >= maxScreenX   (* Если лев дошел до правого края, *)
        then dx := left;     (* то ему нужно повернуть налево. *)

        if x <= minScreenX   (* Если лев дошел до левого края, *)
        then dx := right;    (* то ему нужно повернуть направо. *)

        if y >= maxScreenY   (* Если лев дошел до нижней строчки, *)
        then dy := up;       (* то ему нужно повернуть вверх. *)

        if y <= minScreenY   (* Если лев дошел до верхней строчки, *)
        then dy := down;     (* то ему нужно повернуть вниз. *)

        (* Сотрем льва на старом месте. *)
        GotoXY(x, y);
```

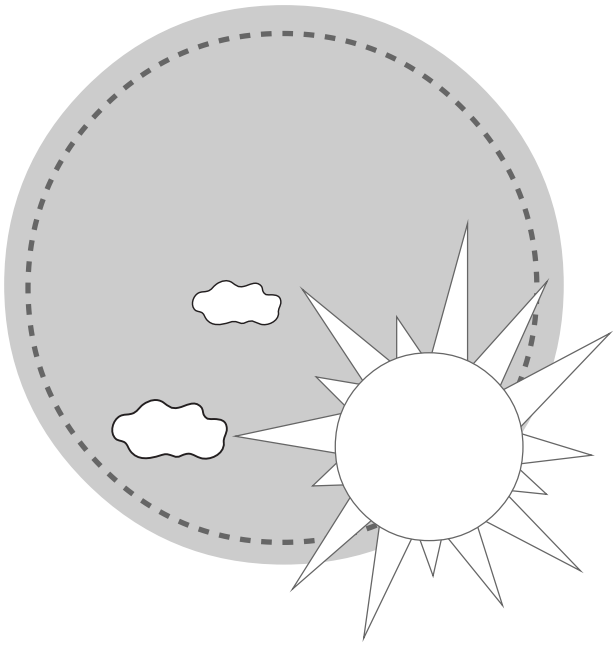


```
        Write(' ');

        (* Передвинем льва в нужном направлении. *)
        x := x + dx;
        y := y + dy;

        (* Нарисуем льва на новом месте. *)
        GotoXY(x, y);
        Write(LionSym);
    end
end.
```

Лев научился бегать по экрану.

Но где же саванна и пустыня? В нашей сказке лев ходит только по саванне, а охотник — еще и по пустыне.





ГЛАВА 13.

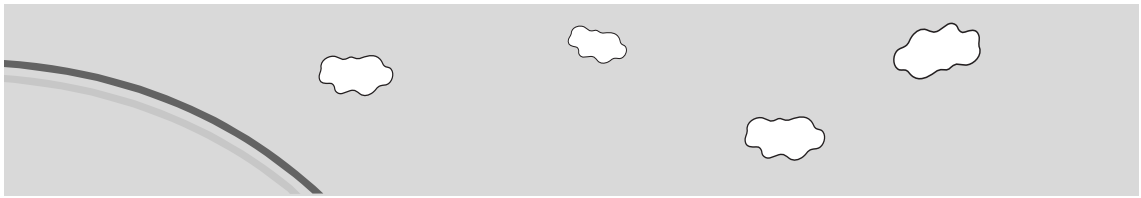
ВМЕСТЕ С ПАПОЙ МЫ ПРИДУМЫВАЕМ СВОЮ ФУНКЦИЮ

Папа ответил:

— Нарисовать прямоугольник — саванну и пустыню — нам помогут жук-художник и волшебные человечки. Программисты не пишут одну непрерывную программу, а разделяют ее на множество функций. Профессионалы используют также функции, написанные другими людьми, например функции `Write`, `GotoXY` и `TextColor`. А сейчас, ребята, мы придумаем своего волшебного человечка и дадим ему задание.

Таня спросила:

— Как мы назовем этого человечка?



Папа рассмеялся и сказал:

— Нашу функцию мы назовем DrawRect, что в переводе с английского означает «рисовать прямоугольник». Функции DrawRect дадим пять параметров (пять чисел). Это координаты верхнего левого и правого нижнего углов прямоугольника, а также цвет, которым будет закрашен прямоугольник. Посмотрите, как мы опишем нашу функцию в программе:

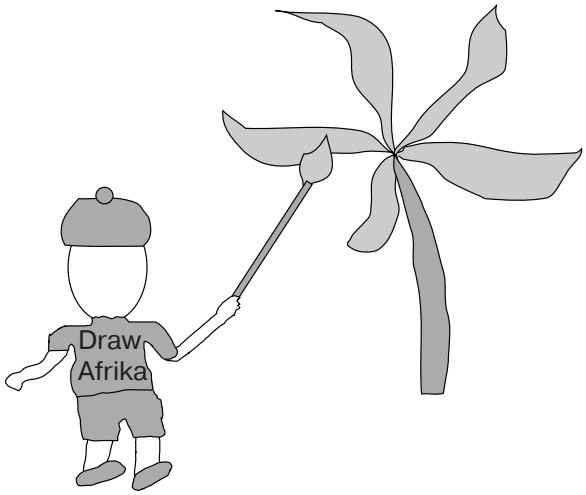
```
(*****************************)
(* Нарисовать прямоугольник. *)
(*****************************)
procedure DrawRect(
    leftG      : integer,     (* Левая граница. *)
    rightG     : integer,     (* Правая граница. *)
    topG       : integer,     (* Верхняя граница. *)
    bottomG    : integer,     (* Нижняя граница. *)
    color      : integer);    (* Цвет прямоугольника. *)
begin
(* Пока наша функция ничего не делает. *)
end;
```

Я внимательно прочел программу и увидел новое слово procedure.

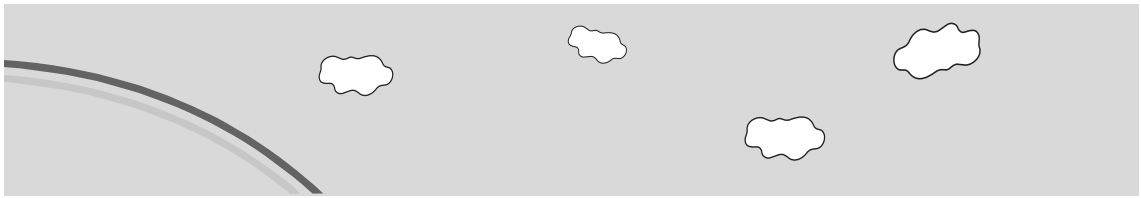
— Переводится оно так же, как и читается, — «процедура» и означает, что функция DrawRect не будет передавать кому-нибудь результаты своей работы. Сравните DrawRect с функцией Chr, которая вам уже знакома.

В программе в круглых скобках стояли переменные. Мы с Таней знали, как кладут в ящички буквы и числа. Понятно, как попали в переменные пять целых чисел: верхний левый угол прямоугольника, нижний правый угол прямоугольника и номер цвета. В эти ящички складывают числа, когда дают задание функции DrawRect. А она делает то, что написано между словами begin и end. Потом папа предложил нам попросить функцию TextColor налить для Write нужную краску, чтобы с помощью дисплейного адаптера и оператора цикла закрасить на экране

прямоугольник и пустые строчки. Затем написал функцию, с помощью которой мы будем рисовать саванну, вот она:




```
(* В этом массиве мы будем хранить цвета, *)
(* которые используются в нашей игре. *)
var Africa : array[1..80, 1..25] of integer;

(*****************************)
(* Нарисовать прямоугольник. *)
(*****************************)
procedure DrawRect(
  leftG        : integer,     (* Левая граница. *)
  rightG       : integer,     (* Правая граница. *)
  topG         : integer,     (* Верхняя граница. *)
  bottomG      : integer,     (* Нижняя граница. *)
  color        : integer);    (* Цвет прямоугольника. *)
var X, Y       : integer;
begin
  TextBackGround(color);           (* Установим цвет фона бумаги. *)
  TextColor(color);                (* Установим цвет символа. *)

  for Y := topG to bottomG do      (* Для строчек от верхней до нижней *)
  begin                            (* делаем следующее. *)
    GotoXY(leftG, Y);              (* Идем в начало строчки. *)
    for X := leftG to rightG do    (* На экране все символы *)
    begin                          (* от левого до правого *)
     Write(" ");                   (* сотрем цветным пробелом. *)
```



```
      Africa[X][Y] := color;   (* В массиве Africa запомним *)
                               (* цвет пробела на экране. *)
    end;
  end;
end;
```

В программе мы увидели оператор цикла `for`. Но почему в ней оказался еще один оператор цикла? И что за слово `array`? Папа объяснил:

— Оператор цикла

```
for Y := topG to bottomG do
begin
end;
```

означает, что координата `Y` будет изменяться от минимальной (верхней) до максимальной (нижней) границы прямоугольника. А главное, этот оператор выполнит задание — то, что стоит между словами `begin` и `end`, причем столько раз, сколько строк в прямоугольнике.

Я поинтересовался:

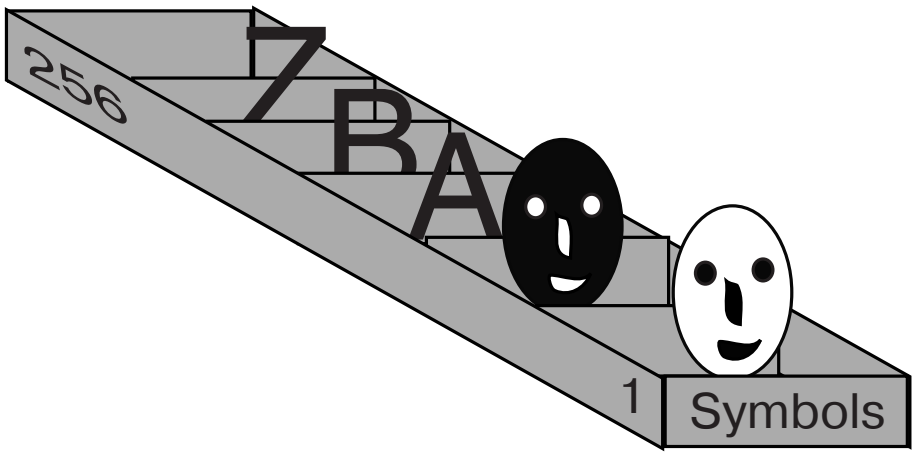
— Но если всю работу будет делать оператор `for Y`, для чего нужен следующий оператор цикла

```
for X := leftG to rightG do
begin
    Write(" ");
    Africa[X][Y] := color;
end;
```

— А этот оператор цикла, Ваня, будет закрашивать саванну в текущей строке с номером `Y` от левой (`leftG`) до правой (`rightG`) границы прямоугольника. Перед этим оператором цикла стоит вызов функции `GotoXY(leftG, Y)`, которая отводит функцию `Write` в начало текущей строки прямоугольника. А теперь поговорим о массивах. Так переводится слово `array`. Помните, как функция `Chr` приносила нам символы? Когда мы давали ей номер нужного

нам символа, она брала его из большого ящика, в который были вложены нумерованные ящички. Если мы захотим в своей программе сделать такой же ящик, то должны написать:

```
var Symbols: array[1..256] of integer;
```



Это значит, что в массиве Symbols с ящичками, пронумерованными от 1 до 256, мы будем хранить целые числа. Чтобы положить целое число, например 10, в третий ящичек, нам нужно написать такую строку:

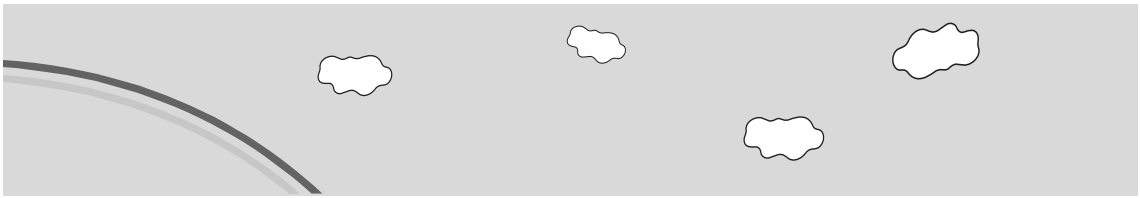
```
Symbols[3] := 10;
```

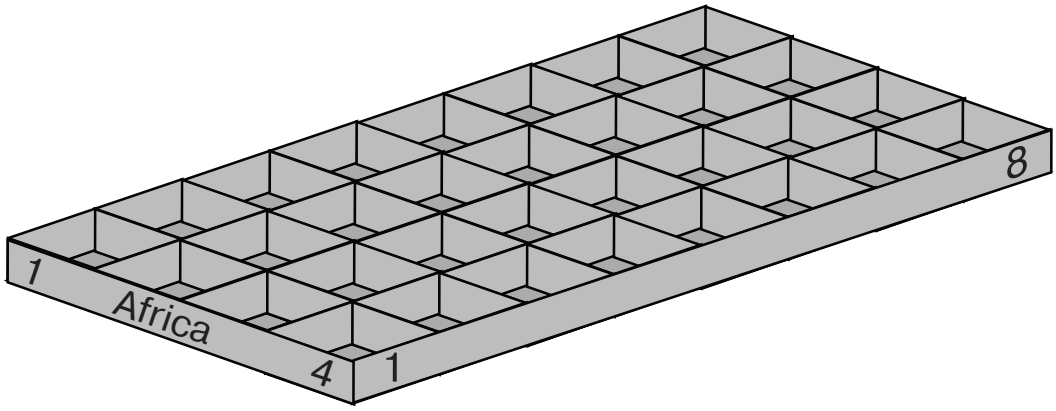
А чтобы взять число из двадцатого ящичка и положить его, предположим, в переменную X, мы должны написать так:

```
X := Symbols[20];
```

Массив Symbols у нас был одномерный. А массив Africa — двумерный, например такой:

```
var Africa : array[1..4, 1..8] of integer;
```





В массиве Africa мы будем записывать, в какие цвета покрасили каждую клеточку на экране. Изменить цвет на экране мы уже не сможем.

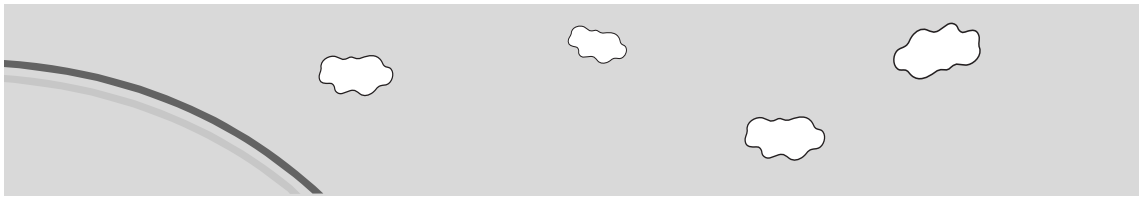
Таня спросила:

— Когда же мы увидим саванну и пустыню, как в сказке?

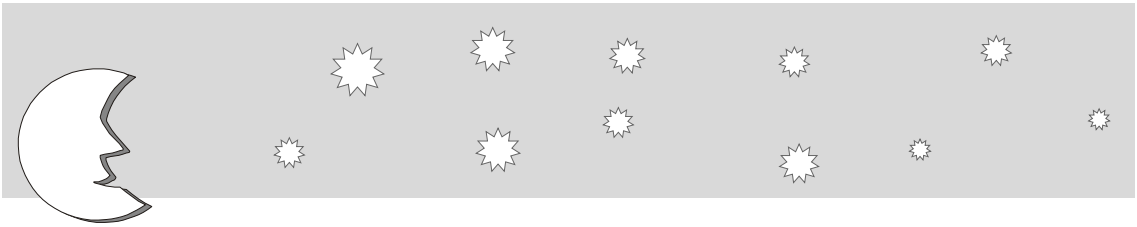
— Сейчас, — ответил папа и запустил программу:

```
program Africa8;

(* Используем функции - друзей жука-художника. *)
uses Crt;

(*********************************)
(* Константы для главных цветов. *)
(*********************************)
const
Black     = 0;      (* Черный. *)
Blue      = 1;      (* Синий. *)
Green     = 2;      (* Зеленый. *)
Red       = 4;      (* Красный. *)
Intensity = 8;      (* Яркий. *)
Blink     = 128;    (* Мигающий. *)

(***********************************)
(* Константы для смешанных цветов. *)
(***********************************)
const
Yellow  = Red + Green + Intensity;
```

```
Brown   = Red + Green;
Gray    = Red + Green + Blue;
White   = Red + Green + Blue + Intensity;
Magenta = Red + Blue;

(***************************)
(* Константы для символов. *)
(***************************)
const
HunterSym = Chr (1);     (* Символ для негра-охотника. *)
LionSym = Chr (2);       (* Символ для льва. *)

(*********************************)
(* Константы для границ экрана. *)
(*********************************)
const
minScreenX     = 1;      (* Левая. *)
minScreenY     = 1;      (* Верхняя. *)
maxScreenX     = 80;     (* Правая. *)
maxScreenY     = 24;     (* Нижняя. *)

(* Границы саванны: *)
const
minSavanaX     =  3;     (* Левая. *)
minSavanaY     =  3;     (* Верхняя. *)
maxSavanaX     = 78;     (* Правая. *)
maxSavanaY     = 23;     (* Нижняя. *)

const
savanaColor    = Green;  (* Цвет саванны - зеленый. *)
groundColor    = Brown;  (* Цвет пустыни - коричневый. *)

(* Описываем ящички-переменные x и y *)
(* для хранения координат льва. *)
var x, y: integer;

(* Переменная для подсчета шагов льва. *)
var k : integer;

(* Переменная для замедления движения льва. *)
var i : integer;

(* Переменные для хранения информации *)
(* о направлении движения льва. *)
```



```
var dx, dy : integer;

(* В этом массиве мы будем хранить цвета, *)
(* которые используются в нашей игре. *)
var Africa : array[1..80, 1..25] of integer;


(*****************************)
(* Нарисовать прямоугольник. *)
(*****************************)
procedure DrawRect(
    leftG      : integer,     (* Левая граница. *)
    rightG     : integer,     (* Правая граница. *)
    topG       : integer,     (* Верхняя граница. *)
    bottomG    : integer,     (* Нижняя граница. *)
    color      : integer);    (* Цвет прямоугольника. *)
var X, Y : integer;
begin
    TextBackground(color);    (* Установим цвет экрана. *)
    TextColor(color);         (* Установим цвет символа. *)

    for Y := topG to bottomG do     (* Для строчек от верхней до нижней *)
    begin                                (* делаем следующее. *)
        GotoXY(leftG, Y);                (* Идем в начало строки. *)
        for X := leftG to rightG do      (* На экране все символы *)
        begin                            (* от левого до правого *)
          Write(" ");                    (* сотрем цветным пробелом. *)
          Africa[X][Y] := color;         (* В массиве Africa запомним *)
                                         (* цвет пробела на экране. *)
        end;
    end;
end;


(**********************)
(* Нарисовать Африку. *)
(**********************)
procedure DrawAfrica;
begin
(* Рисуем пустыню. *)
DrawRect(
    minScreenX, maxScreenX,
    minScreenY, maxScreenY,
    groundColor);
```

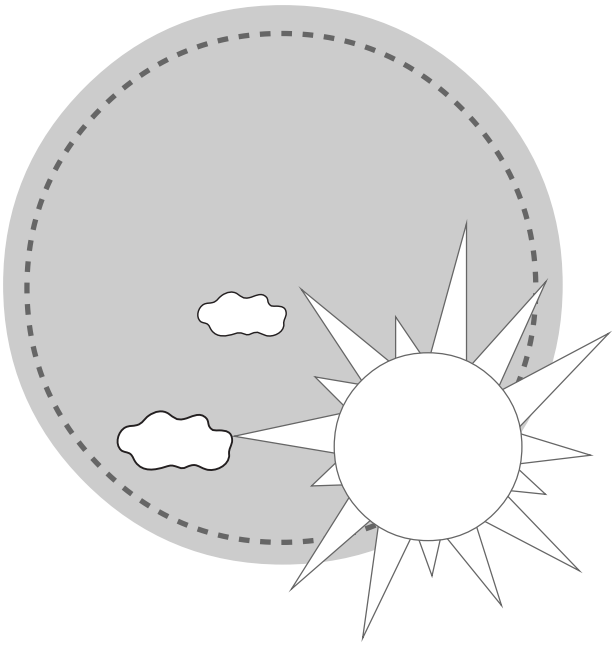


```
(* Рисуем саванну. *)
DrawRect(
    minSavanaX, maxSavanaX,
    minSavanaY, maxSavanaY,
    savanaColor);
end;

begin
    ClrScr;         (* Очистим экран. *)
    DrawAfrica;     (* Нарисуем Африку. *)
end.
```

Я прочитал программу и удивился:

— А куда же делась часть программы, которая рисовала льва?

— Теперь вы, ребята, можете, как настоящие программисты, выделить часть программы, рисующую льва. И дать задание специальной функции выполнить ту часть программы, которая передвигала льва по экрану.



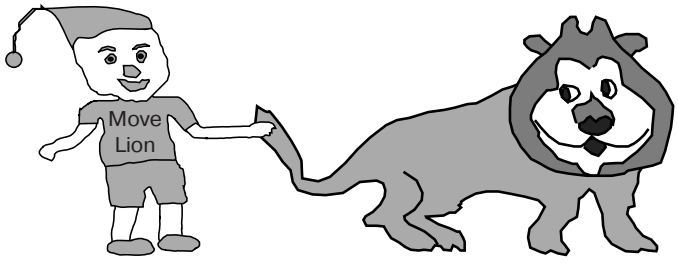
ГЛАВА 14.

НАША ФУНКЦИЯ ВОДИТ ЛЬВА ПО САВАННЕ

Я подумал, что сейчас мы начнем давать функциям разные параметры. Но папа сказал, что в этом нет необходимости, потому что в ящичках-переменных уже лежат координаты для льва. А Таня поинтересовалась:

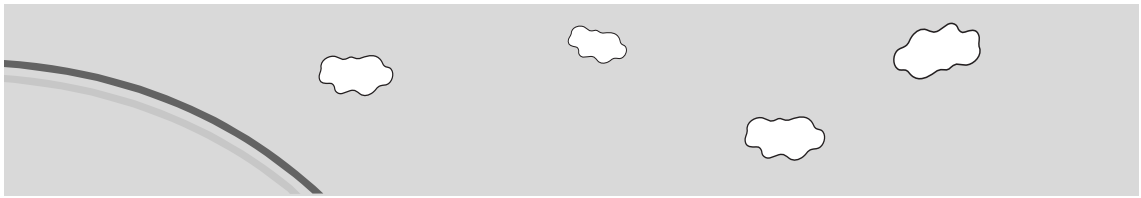
— Папа, как мы назовем функцию, которая будет водить льва на экране?

— Эту функцию, ребята, мы назовем `MoveLion`, что в переводе означает «отвести льва».



Точно так же мы напишем функции, которые будут водить по экрану муху Цеце и охотника, — ответил папа и дописал переменные и функции к программе:

```
const
savanaColor    = Green;  (* Цвет саванны - зеленый. *)
groundColor    = Brown;  (* Цвет пустыни - коричневый. *)
lionColor      = Yellow; (* Цвет льва - желтый. *)

(* Описываем ящички-переменные x и y *)
(* для хранения координат льва. *)
var LionX, LionY: integer;

(* Направление движения. *)
const
up      = -1;
down    =  1;
left    = -1;
right   =  1;

(* Переменные для хранения информации *)
(* о направлении движения льва. *)
var LionDX, LionDY : integer;
(*************************************)
(* Эта функция водит льва по саванне. *)
(*************************************)
procedure MoveLion;
begin
    (*************************************************)
    (* Проверим, не пытается ли лев выйти из саванны. *)
    (* Посмотрим, какого цвета следующая клетка       *)
    (* и соседние с ней клетки на экране.             *)
    (*************************************************)
    nextColor  := Africa[LionX + LionDX][LionY + LionDY];
    nextColorY := Africa[LionX ][LionY + LionDY];
    nextColorX := Africa[LionX + LionDX][LionY ];

    if nextColor <> savanaColor then
    (* Лев пытается выйти из саванны. *)
    (* Попробуем повернуть в сторону. *)
    begin
        if nextColorX = nextColorY then
        begin
          (* Повернем льва назад. *)
```

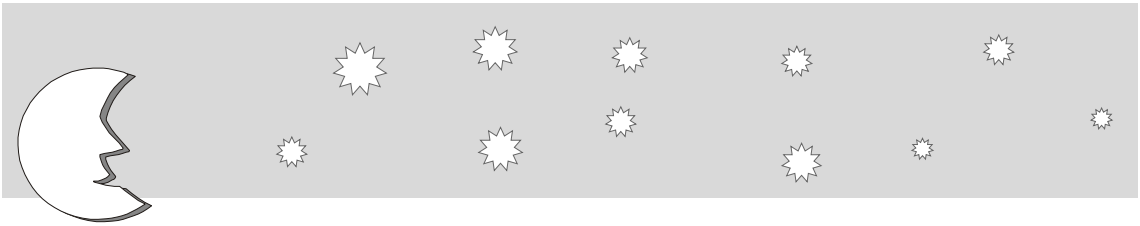


```
          LionDX := -LionDX;
          LionDY := -LionDY;
        end
        else
        if nextColorY = savanaColor then
          (* Попробуем свернуть в другую сторону по горизонтали. *)
          LionDX := -LionDX
        else
        if nextColorX = savanaColor then
          (* Попробуем свернуть в другую сторону по вертикали. *)
          LionDY := -LionDY
    end;

    (*************************)
    (* Лев делает один шаг. *)
    (*************************)
    (* Сначала сотрем льва на старом месте. *)
    GotoXY(LionX, LionY);
    WriteLn(" ");

    (* Вычислим новое место льва. *)
    LionX := LionX + LionDX;
    LionY := LionY + LionDY;

    (* Нарисуем льва на новом месте *)
    (* тем же цветом на прежнем фоне. *)
    TextColor(LionColor);
    GotoXY(LionX, LionY);
    WriteLn(LionSym);
end;
(*****************************************)
(* Функция, которая поселит льва в саванне. *)
(*****************************************)
procedure SetLion;
begin
    (* Это начальное положение льва на экране. *)
    LionX := 5;
    LionY := 5;

    (* Сначала лев побежит направо и вниз. *)
    LionDX := right;
    LionDY := down;
end;
```

```
(*******************************************************)
(* Чтобы ребята успели рассмотреть Африку,             *)
(* не станем торопиться показывать следующий           *)
(* кадр мультфильма. Будем складывать числа в цикле.  *)
(* Это делает функция Wait.                            *)
(*******************************************************)
procedure Wait(time : integer);
var j,i: integer;
begin
    for j := 1 to time do
    begin
        for i := 1 to 32000 do;
        for i := 1 to 32000 do;
        for i := 1 to 32000 do;
        for i := 1 to 32000 do;
    end;
end;

begin
    ClrScr;         (* Очистим экран. *)
    DrawAfrica;     (* Нарисуем Африку. *)
    SetLion;        (* Поселим льва в Африке. *)

    for k := 1 to 10000 do
    begin
        Wait(20);   (* Подождем. *)
        MoveLion;   (* Потом отведем льва на новое место. *)
    end
end.
```

Папа запустил программу. На экране дисплея мы увидели льва. Он бегал от одного края саванны до другого. Казалось, он высматривает охотника. Но охотника там не могло быть, мы же еще не написали в программе функцию, которая должна водить его по экрану. Таня с нетерпением ждала появления охотника. Но папа не спешил писать программу. Он сказал:

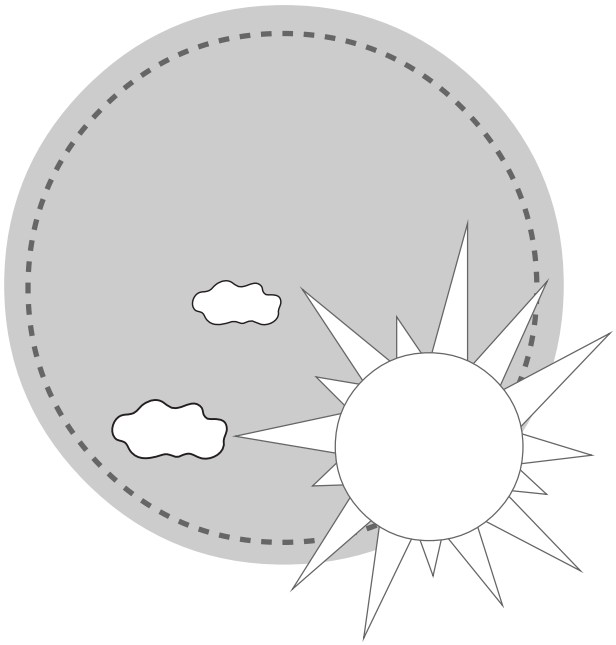
— Не торопись, Таня. Функция для охотника гораздо сложнее, чем для льва. Наш охотник должен слушаться клавиш, которые ты будешь нажимать. К тому



же в Африке пока не хватает мухи Цеце. Попробуйте-ка сами написать функцию, которая будет передвигать муху по экрану. Эта функция похожа на `MoveLion`. Но учтите, что муха Цеце летает над пустыней, а не над саванной.

Я понял, как мы писали функцию `MoveLion`, и сказал папе:

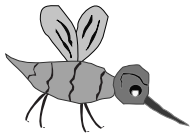
— Для мухи Цеце можно написать такую же функцию, как и для льва. Только в программе вместо `MoveLion` нужно писать `MoveMyxaCeCe`, что означает «отвести муху Цеце».



ГЛАВА 15.

В АФРИКЕ ЛЕТАЕТ МУХА ЦЕЦЕ

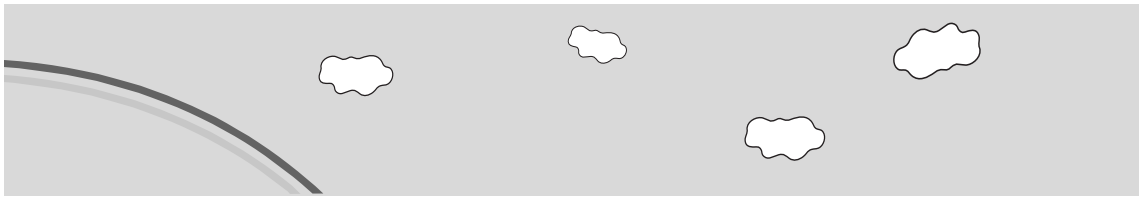
— Правильно, Ваня, но этого еще мало. В сказке льву нельзя выходить из саванны, а мухе Цеце — залетать туда. Вспомните, в нашей программе лев не сделал ни одного шага по пустыне. Муха, как и лев, должна быть внимательна. Ее задача — вовремя повернуть назад и укусить охотника.



— А на какую букву похожа муха?

Мы решили, что она похожа на букву «ж», потому что умеет жужжать, и придумали мухе такой символ, добавив в программу описание ее констант и переменных:

```
const
MухаCeCeSym    = "Ж";          (* Символ мухи. *)
MухаCeCeColor  = Black;        (* Цвет мухи. *)
```

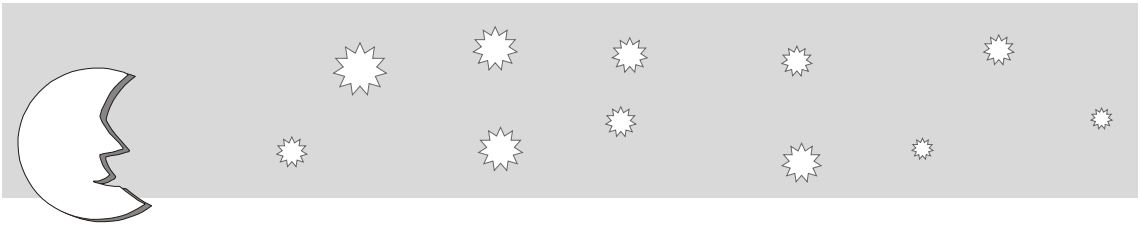


```
var
MyxaCeCeX,                  (* Координаты *)
MyxaCeCeY,                  (* мухи. *)
MyxaCeCeDX,                 (* Направление *)
MyxaCeCeDY : integer;       (* полета мухи. *)
```

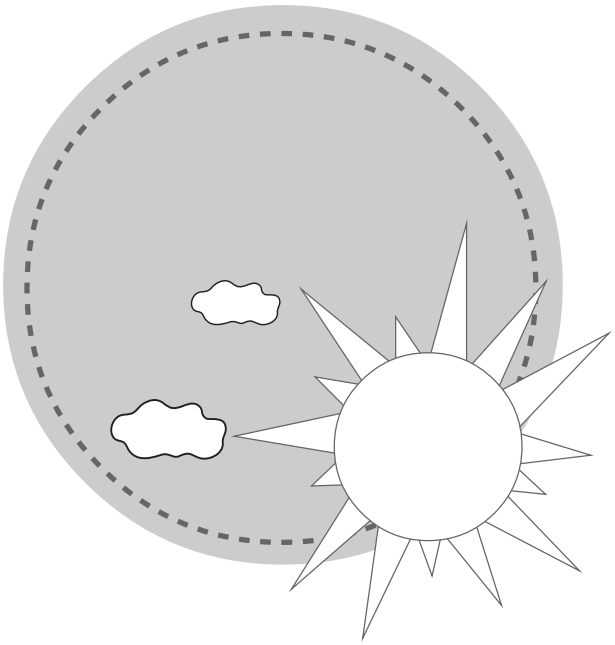
Чтобы поселить муху Цеце в Африке и научить ее летать, папа написал функции SetMyxaCeCe и MoveMyxaCeCe:

```
(*********************************************)
(* Функция, которая поселит муху Цеце в Африке. *)
(*********************************************)
procedure SetMyxaCeCe;
begin
    (* Сначала муха находится в правом нижнем углу экрана. *)
    MyxaCeCeX := maxScreenX -1;
    MyxaCeCeY := maxScreenY -1;

    (* Потом муха полетит вправо и вверх. *)
    MyxaCeCeDX := right;
    MyxaCeCeDY := up;
end;

(*****************************************)
(* Эта функция водит муху Цеце по Африке. *)
(*****************************************)
procedure MoveMyxaCeCe;
var
nextColor,              (* Цвет следующей клетки. *)
nextColorX,             (* Цвет клеток *)
nextColorY : integer;   (* слева и справа от следующей. *)
begin
    (*************************************************************)
    (* Проверим, не пытается ли муха Цеце залететь в саванну.  *)
    (* Посмотрим, какого цвета следующая клетка                *)
    (* и клетки для возможного поворота мухи по осям x и y.    *)
    (*************************************************************)
    nextColor  := Africa[MyxaCeCeX + MyxaCeCeDX][MyxaCeCeY + MyxaCeCeDY];
    nextColorY := Africa[MyxaCeCeX][MyxaCeCeY + MyxaCeCeDY];
    nextColorX := Africa[MyxaCeCeX + MyxaCeCeDX][MyxaCeCeY];

    (* Сотрем муху Цеце на старом месте пробелом цвета пустыни. *)
    TextBackground(groundColor);
```

```
    GotoXY(MyxaCeCeX, MyxaCeCeY);
    Write(" ");

    if nextColor <> groundColor (* Впереди не цвет пустыни. *)
    then (* Муха пытается залететь в саванну! *)
    begin
        if nextColorX = nextColorY then
        begin  (* Повернем муху назад. *)
          MyxaCeCeDX := -MyxaCeCeDX;
          MyxaCeCeDY := -MyxaCeCeDY;
        end
        else
        if nextColorY = groundColor then
          (* Попробуем повернуть муху в другую сторону по горизонтали. *)
          MyxaCeCeDX := -MyxaCeCeDX
        else
        if nextColorX = groundColor then
          (* Попробуем повернуть муху в другую сторону по вертикали. *)
          MyxaCeCeDY:= -MyxaCeCeDY;
    end;

    (*********************************************************)
    (* Проверим, не пытается ли муха Цеце улететь с экрана. *)
    (*********************************************************)
    if (MyxaCeCeX + MyxaCeCeDX < minScreenX) or
        (MyxaCeCeX + MyxaCeCeDX > maxScreenX) then
        MyxaCeCeDX := -MyxaCeCeDX;

    if (MyxaCeCeY + MyxaCeCeDY < minScreenY) or
        (MyxaCeCeY + MyxaCeCeDY > maxScreenY) then
        MyxaCeCeDY := -MyxaCeCeDY;

    (***********************************)
    (* Муха Цеце летит на новое место. *)
    (***********************************)
    (* Вычислим новое место мухи. *)
    MyxaCeCeX := MyxaCeCeX + MyxaCeCeDX;
    MyxaCeCeY := MyxaCeCeY + MyxaCeCeDY;

    (* Нарисуем муху на новом месте. *)
    TextColor(MyxaCeCeColor);
    GotoXY(MyxaCeCeX, MyxaCeCeY);
    Write(MyxaCeCeSym);
end;
```



К программе папа добавил вызов этих функций — там, где был вызов функций, двигающих льва. Потом мы запустили программу. Компьютер с удовольствием показывал на экране льва и муху Цеце. Льву было просторно в саванне. Мухе Цеце досталась лишь узкая полоска пустыни. Нам с Таней казалось, что муха очень злится, летая над таким маленьким пространством. Не хватало лишь ее сердитого жужжания. Тане не терпелось увидеть в Африке охотника. Она просила папу скорее впустить его в сказку.



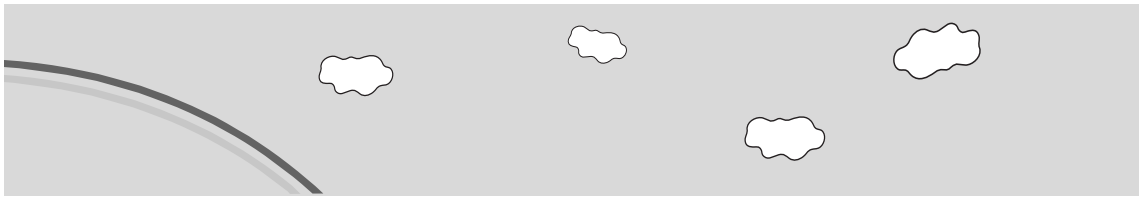
ГЛАВА 16.

ПО САВАННЕ ХОДИТ ОХОТНИК

— Вы правы, ребята, — сказал папа. — Сказка без охотника неинтересна. Но пока охотник будет ходить по саванне по тем же правилам, что и лев. А слушаться клавиш мы научим его чуть позже.

Так как в нашей сказке охотник шагает по саванне и по пустыне, подумали мы, то функция для него будет похожа на функцию, которая водит льва. Папа согласился с нами, напомнив:

— В саванне охотник строит заграждения, на экране это будут клетки малинового цвета. Но он должен построить забор до того, как подойдет лев. Если охотник не успеет спрятаться, лев сразу его съест. Функцию для охотника мы назовем `MoveHunter` — «отвести охотника».



Я предложил заменить в программе слово Lion на Hunter.

— Согласен, — сказал папа. — Многие так обычно и делают. Копируют готовые программы, а затем исправляют их так, чтобы решить задачу. С помощью текстового редактора можно копировать большие части программы. Но перед этим мы должны четко уяснить, чем отличается поведение охотника от поведения льва и что у них общего.

Таня воскликнула:

— Я знаю: охотник в отличие от льва и мухи может передвигаться и по пустыне, и по саванне! Ему разрешается ходить по всему экрану.

Папа улыбнулся и сказал:

— Молодец, Таня. Теперь нам ясно, что убрать из скопированной программы, а что оставить. Убрать нужно условные операторы, которые будут поворачивать охотника назад, если впереди прямоугольник не того цвета. У нас охотник строит забор малиновыми прямоугольниками. Поэтому прежде чем наступить на зеленое поле (саванну) он должен как бы покрасить это поле в малиновый цвет. Забор-то малиновый. А когда охотник его построит и выйдет из саванны в пустыню, мы перекрасим забор под цвет пустыни — коричневый. Если лев наткнется на малиновый прямоугольник, значит, охотник еще в саванне, и лев его съест. А если забор коричневый, лев уже не страшен.

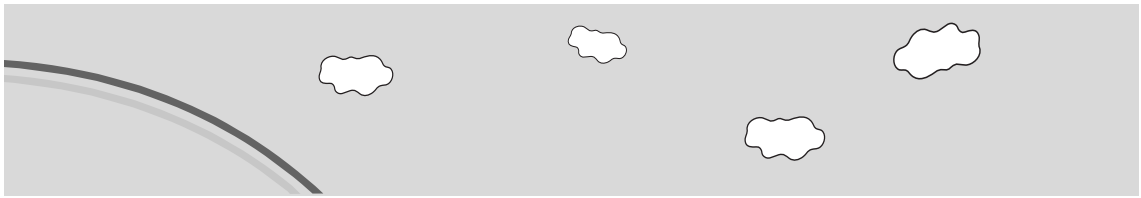
Папа добавил в программу описание констант для охотника:

```
(************)
(* Охотник. *)
(************)
const
HunterSym      = Chr(1); (* Символ охотника. *)
HunterSound    = Chr(7); (* Звук охотника. *)
```

```
HunterColor    = Black;  (* Цвет охотника. *)
ZaborColor     = Malina; (* Цвет забора. *)

var
HunterX,            (* Координаты *)
HunterY,            (* охотника. *)
HunterDX,           (* Направление *)
HunterDY : integer; (* движения охотника. *)
```

Чтобы поселить охотника в Африке и водить его по экрану, папа добавил в программу функции:

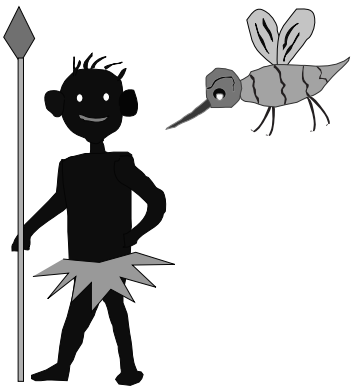
```
(*****************************************)
(* Эта функция поселит охотника в Африке. *)
(*****************************************)
procedure SetHunter;
begin
    HunterY    := minScreenY;          (* Вверху. *)
    HunterX    := maxScreenX div 2;    (* Посередине. *)
    HunterDX   := 0;                   (* Пойдет *)
    HunterDY   := down;                (* вниз. *)
end;

(***********************************)
(* Функция красит построенный забор *)
(* в цвет пустыни.                  *)
(***********************************)
procedure paintZabor;
var x, y, color : integer;
begin
    for y := minScreenY to maxScreenY do
        for x := minScreenX to maxScreenX do
        begin
          (* Посмотрим цвет клетки. *)
          color := Africa[x, y];
          (* Если это цвет забора, *)
          (* перекрасим его в цвет пустыни. *)
          if color = zaborColor then
          begin
               TextBackground(groundColor);
               GotoXY(x, y);
               Write(" ");
          end
        end;
end;
```



```
(*****************************************)
(* Эта функция водит охотника по саванне. *)
(*****************************************)
procedure MoveHunter;
var
newPosColor : integer;   (* Цвет клетки, на которую *)
                         (* наступил охотник. *)
begin
    (*****************************************************)
    (* Проверим, не пытается ли охотник уйти с экрана. *)
    (* Если пытается, вернем его назад.                *)
    (*****************************************************)
    if (HunterX + HunterDX < minScreenX) or
        (HunterX + HunterDX > maxScreenX) then
        HunterDX := -HunterDX;

    if (HunterY + HunterDY < minScreenY) or
        (HunterY + HunterDY > maxScreenY) then
        HunterDY := -HunterDY;

    (****************************)
    (* Охотник делает один шаг. *)
    (****************************)
    (* Сначала сотрем охотника на старом месте. *)
    TextBackground( Africa[HunterX][HunterY] );
    GotoXY(HunterX, HunterY);
    Write(" ");

    (* Вычислим новое место охотника. *)
    HunterX := HunterX + HunterDX;
    HunterY := HunterY + HunterDY;

    (* Нарисуем охотника на новом месте. *)
    newPosColor := Africa[HunterX][HunterY];

    if (newPosColor <> groundColor)
    then  (* Охотник не в пустыне. *)
        begin  (* Рисуем цветом забора. *)
          TextBackGround(zaborColor);
          Africa[HunterX][HunterY] := zaborColor;
        end
```

```
    else begin (* Иначе - старым цветом. *)
        TextBackGround (newPosColor);
        Africa[HunterX][HunterY] := newPosColor;
    end;
    TextColor(HunterColor);
    GotoXY(HunterX, HunterY);
    Write(HunterSym);

    (* Если охотник вернулся из саванны *)
    (* в пустыню, то забор построен. *)
    (* Покрасим забор в цвет пустыни. *)
    if (newPosColor = groundColor)
    then paintZabor;
end;
```

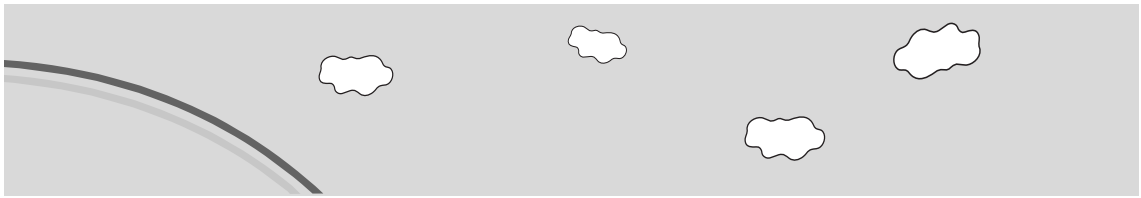
Когда запустили программу, на экране показались лев, муха Цеце и охотник. Каждый двигался сам по себе, не замечая других. Охотник строил забор. Лев через него перепрыгивал, но не съедал охотника. Муха мирно летала над пустыней и не замечала охотника.



— Эта сказка неправильная, — неожиданно загрустила Таня.

— Пока да, — сказал папа. — Но вспомни: когда мы писали функции для мухи Цеце и льва, охотника на экране еще не было. Для того чтобы муха его укусила, нужно в функцию MoveMyxaCeCe вставить такие строчки:

```
(* Посмотрим, нет ли на новом месте охотника. *)
if (MyxaCeCeX = HunterX) and (* Координаты X и Y *)
    (MyxaCeCeY = HunterY)    (* мухи и охотника совпали. *)
then begin
    (* Муха укусит охотника. *)
    Write(HunterSound);  (* Охотник кричит. *)
    Halt;                (* Программа заканчивается. *)
end
```



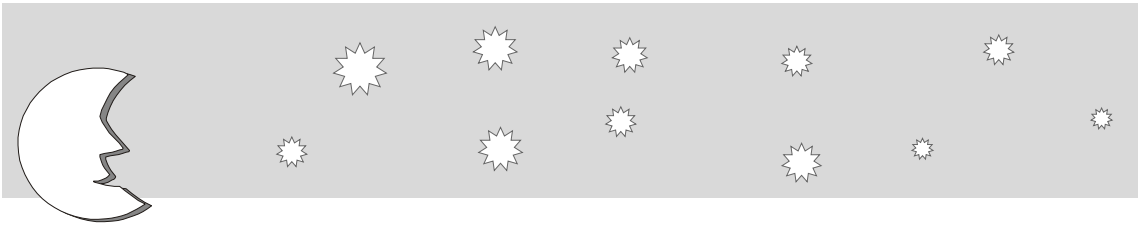
А чтобы лев съедал охотника и не перепрыгивал через забор, нужно изменить функцию MoveLion:

```
(*************************************)
(* Эта функция водит льва по саванне. *)
(*************************************)
procedure MoveLion;
var
nextColor,              (* Цвет следующей клетки. *)
nextColorX,             (* Цвет клеток *)
nextColorY : integer;   (* слева и справа от следующей. *)

begin
(*****************************************************)
(* Проверим, не пытается ли лев выйти из саванны. *)
(* Посмотрим, какого цвета следующая клетка       *)
(* и соседние с ней клетки на экране.             *)
(*****************************************************)
nextColor := Africa[LionX + LionDX][LionY + LionDY];
nextColorY     := Africa[LionX][LionY + LionDY];
nextColorX     := Africa[LionX + LionDX][LionY];

(************************)
(* Лев делает один шаг. *)
(************************)
(* Если лев наткнулся на забор, *)
(* а охотник еще не в пустыне, *)
(* то лев съест охотника. *)
if nextColor = zaborColor then
begin
    Write(HunterSound); (* Охотник кричит. *)
    Write(HunterSound); (* Охотник кричит. *)
    Write(HunterSound); (* Охотник кричит. *)
    Halt; (* Программа заканчивается. *)
end;

if nextColor <> savanaColor then
(* Лев пытается выйти из саванны. *)
(* Попробуем повернуть его в сторону. *)
begin
    if nextColorX = nextColorY then
    begin
        (* Повернем льва назад. *)
```

```
        LionDX := -LionDX;
        LionDY := -LionDY;
    end
    else
    if nextColorY = savanaColor then
        (* Попробуем свернуть в другую сторону по горизонтали. *)
        LionDX := -LionDX;
    else
    if nextColorX = savanaColor then
        (* Попробуем свернуть в другую сторону по вертикали. *)
        LionDY := -LionDY;
    end;

    (* Сначала сотрем льва на старом месте. *)
    TextBackground( Africa[LionX][LionY] );
    GotoXY(LionX, LionY);
    WriteLn(" ");

    (* Вычислим новое место льва. *)
    LionX := LionX + LionDX;
    LionY := LionY + LionDY;

    (* Нарисуем льва на новом месте *)
    (* тем же цветом на прежнем фоне. *)
    TextBackground( Africa[LionX][LionY] );
    TextColor(LionColor);
    GotoXY(LionX, LionY);
    WriteLn(LionSym);
end;
```

Мы исправили программу и вновь запустили ее. На экране опять были лев, муха Цеце и охотник. Теперь лев уже не мог перепрыгнуть через забор. Он добегал до него и поворачивал назад. Но мы сильно огорчались, когда лев съедал охотника, а муха Цеце кусала его. Несчастный охотник не успевал спрятаться или убежать от них. Как же ему было страшно! Нам казалось, что охотник просит о помощи. Я крикнул ему:

Поверь мне, охотник!
Тебе помогу!



Забор я, как плотник,
Построить смогу.
Мы вместе с тобою
Пойдем по саванне,
И лев в западню
Попадет на экране.

— Папа! — воскликнула Таня. — Давай быстрей впишем в программу такие строчки, которые помогут охотнику вовремя замечать льва и муху Цеце и убегать.

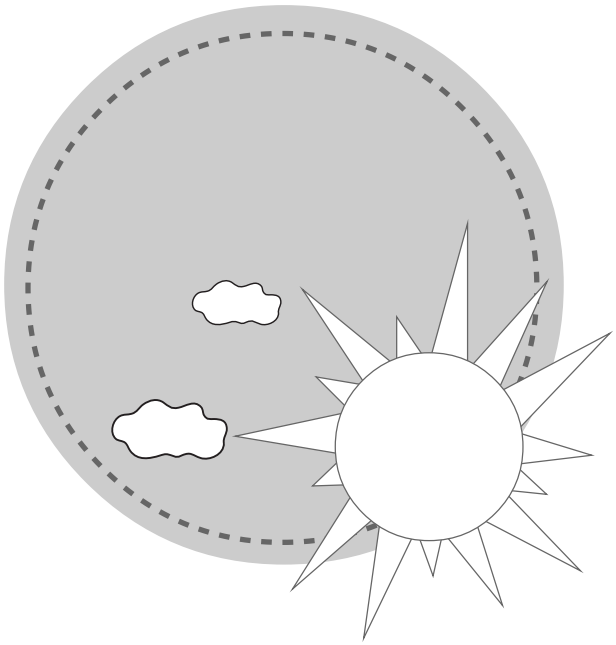
— Конечно, мы можем это сделать, но тогда охотник будет всегда убегать от врагов. Лев никогда не съест его, а муха не укусит. И наша сказка не будет такой интересной. Давайте лучше поможем охотнику. Но сначала нужно научиться управлять охотником с помощью клавиш. Вы же играли в компьютерные игры!

Папа прав. Мы хорошо умеем играть, используя клавиатуру и даже джойстик.

— Но как же функция `MoveHunter` узнает, какую клавишу мы нажали? — спросил я папу.

Он ответил:

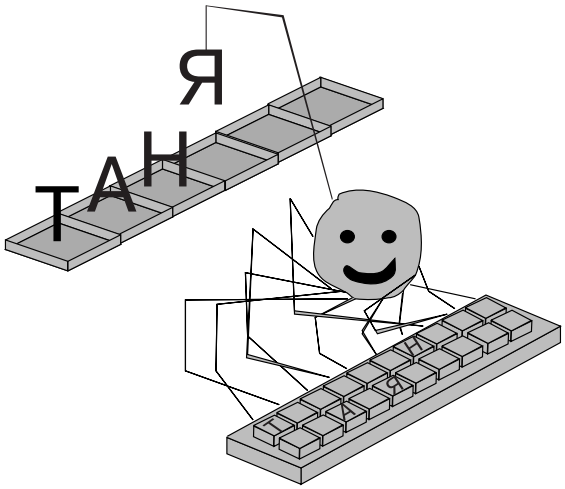
— Чтобы понять это, нужно познакомиться с *жуком-контроллером клавиатуры* и его функциями. Этот жук тоже дружит с волшебными человечками.

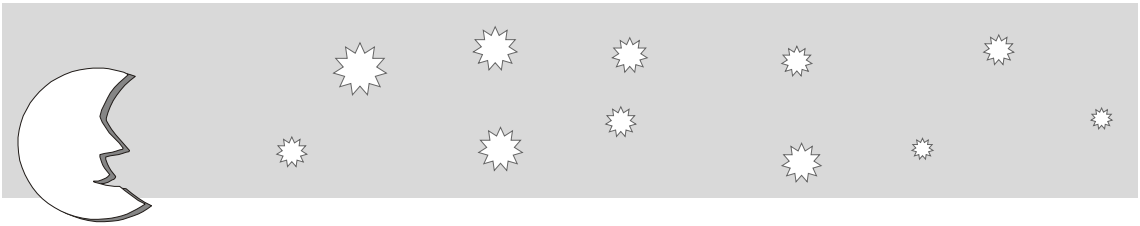


ГЛАВА 17.

МЫ ЗНАКОМИМСЯ С КОНТРОЛЛЕРОМ КЛАВИАТУРЫ

Компьютер показал на экране очень странного игрушечного жука.





Вместо лапок у него были провода, которые тянулись к клавишам. И еще у жука была память с ящичками-переменными. Папа объяснил, что он не рисует содержимое своей памяти на экране, как жук-художник. Главная задача контроллера клавиатуры — следить за клавиатурой и относить нажатые буквы и символы в свою память. Жук чувствует, когда мы нажимаем клавишей на его лапку, и знает, какую именно клавишу мы нажали. Жук кладет в свою память ее номер, букву и номер значка, нарисованные на ней.

— А интересно, успеет ли контроллер положить нужные буквы в свою память, если очень быстро нажимать на клавиши?

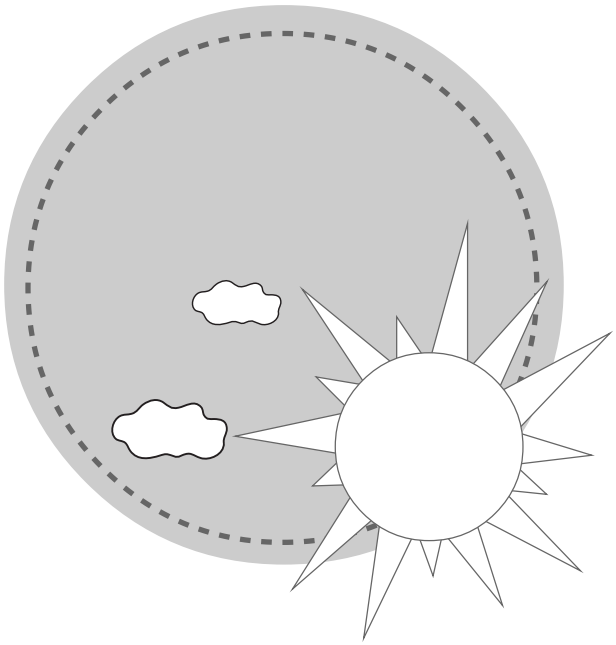
Папа ответил:

— Ребята, контроллер клавиатуры постепенно кладет буквы, написанные на клавишах, в ящички. А начинает с первого. Если держать нажатой клавишу или быстро нажимать на них, то жук так же быстро будет укладывать в свою память одну или разные буквы. Например, курсор бегает на экране до тех пор, пока не отпустишь клавишу.

Нам было интересно, что будет делать жук, когда все его ящички-переменные заполнятся.

— Он будет громко жужжать в динамик. Так контроллер клавиатуры сообщает о переполнении памяти. Он просит подождать, пока программы не разберут все буквы из его ящичков.

— Папа, а мы напишем функции, которые будут брать буквы из памяти контроллера клавиатуры? — спросил я.



ГЛАВА 18.

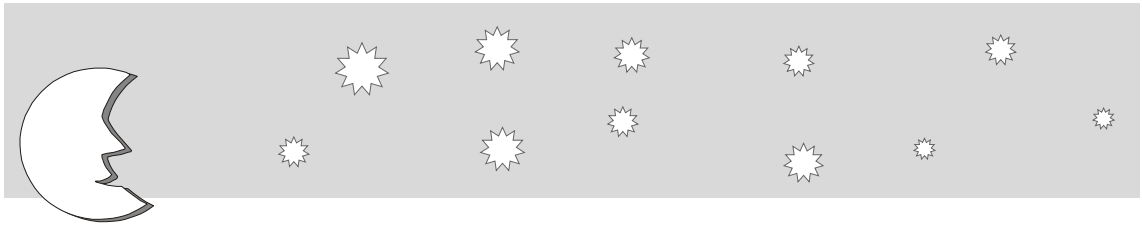
ФУНКЦИИ KEYPRESSED И READKEY ПОМОГАЮТ КОНТРОЛЛЕРУ КЛАВИАТУРЫ

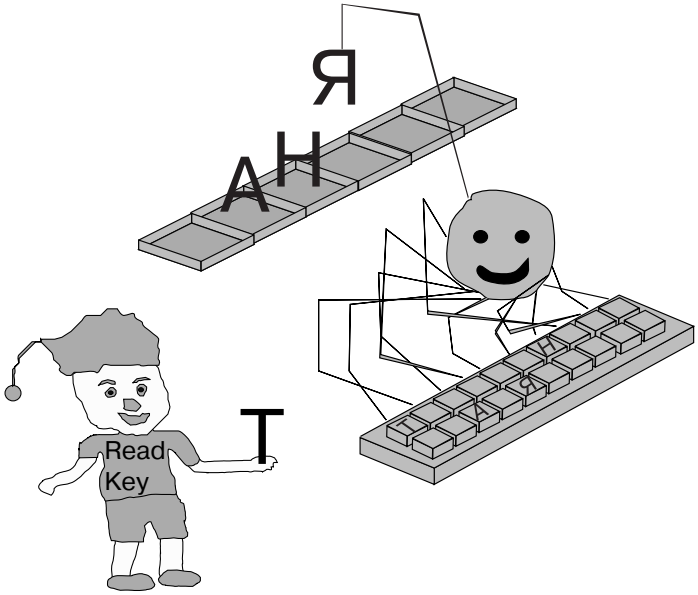
– Такие функции, Ваня, уже написаны программистами. Их две. Одну зовут `KeyPressed`, или «клавиша нажата». Волшебный человечек — функция `KeyPressed` — смотрит, есть ли в памяти контроллера клавиатуры буквы. Если есть, то `KeyPressed` приносит в результате своей работы `true` (Да), а если нет — `false` (Нет). Узнав с помощью функции `KeyPressed`, были ли нажаты клавиши, мы сможем считать их. В нашу программу их принесет другая функция — `ReadKey`, или «читай клавишу».

Тут неожиданно заговорил компьютер:

— Ребята, посмотрите на экран дисплея.

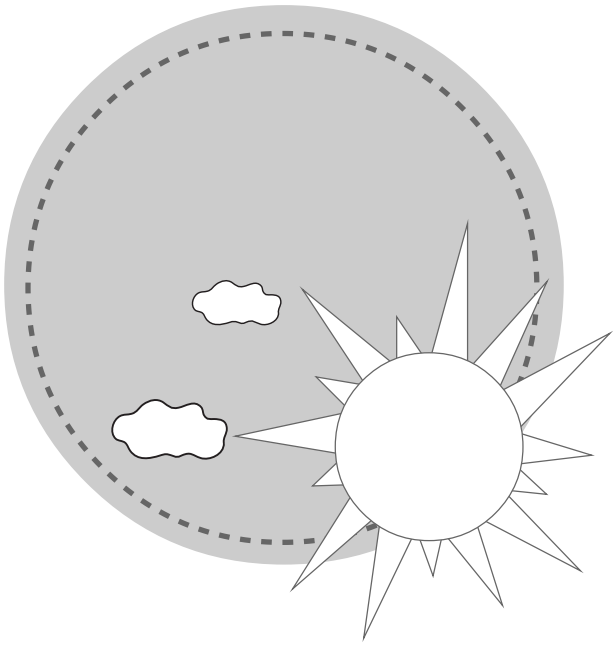
Мы увидели улыбающегося человечка.





Папа предложил:

— Давайте сообщать охотнику, куда ему пойти, с помощью клавиш. Вы их хорошо знаете.

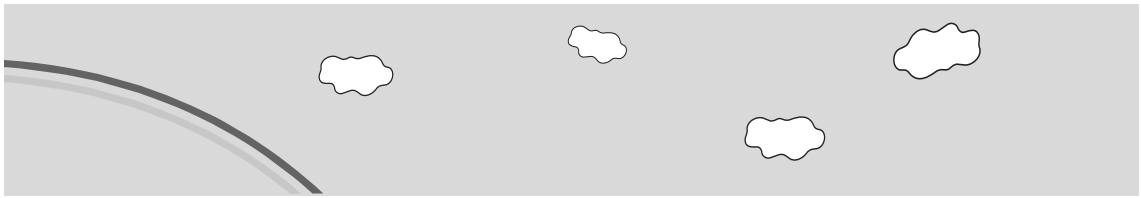


ГЛАВА 19.

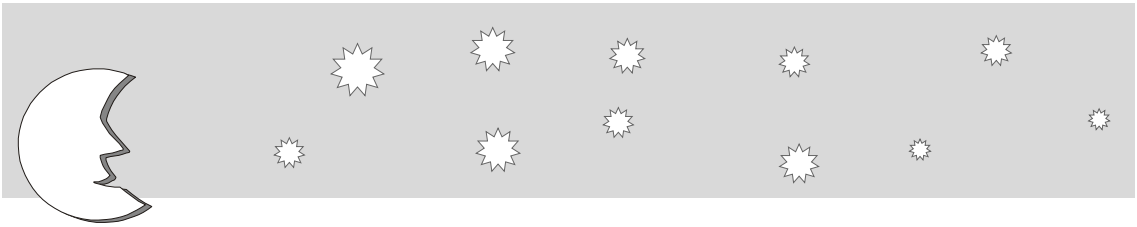
ОХОТНИК УБЕГАЕТ ОТ ЛЬВА

Действительно, клавиши со стрелками нам знакомы. И папа изменил конец нашей программы:

```
var key : char;

begin
    ClrScr;         (* Очистим экран. *)
    DrawAfrica;     (* Нарисуем Африку. *)
    SetLion;        (* Поселим льва в Африке. *)
    SetMyxaCeCe;    (* Поселим муху в Африке. *)
    SetHunter;

    for k := 1 to 10000 do
    begin
        GotoXY(80,25);

        Wait(20);         (* Подождем. *)
        MoveLion;         (* Потом отведем льва на новое место. *)
        MoveMyxaCeCe;     (* Потом отведем муху на новое место. *)
```

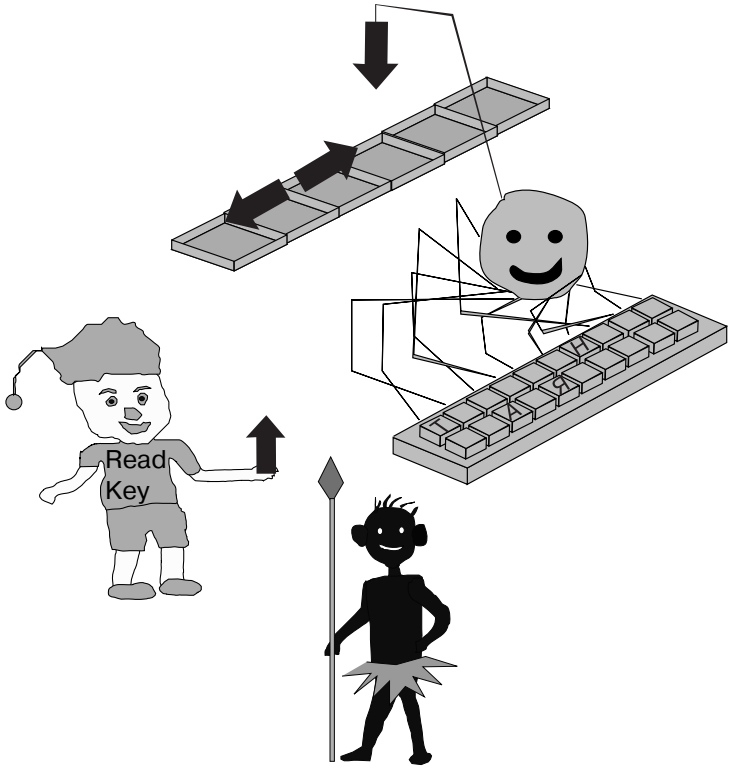


```
      (* Пока нужные клавиши не нажаты, охотник стоит! *)
      HunterDX := 0;
      HunterDY := 0;
      MoveHunter;

      (* Нарисуем охотника. *)
      (* Если клавиши нажаты, *)
      (* то отведем охотника на новое место. *)

      if KeyPressed then
      begin
        key := ReadKey;

        if (key = #27)          (* Нажмем клавишу Esc, *)
        then exit;              (* чтобы прекратить игру. *)

        if (key = #0)           (* Это клавиша без буквы. *)
        then key := ReadKey;    (* Прочитаем номер клавиши. *)

        (* В зависимости от нажатой клавиши *)
        (* укажем, куда охотнику идти: *)
        (* в - вверх, *)
        (* н - вниз, *)
        (* п - вправо, *)
        (* л - влево. *)
        if key = "в" then HunterDY :=   -1;
        if key = "н" then HunterDY :=    1;
        if key = "п" then HunterDX :=    1;
        if key = "л" then HunterDX :=   -1;

        (* То же самое, но для клавиш со стрелками. *)
        if key = "4" then HunterDX :=   -1;
        if key = "6" then HunterDX :=    1;
        if key = "8" then HunterDY :=   -1;
        if key = "2" then HunterDY :=    1;

        (* Отвести охотника на один шаг *)
        (* в указанном клавишами направлении. *)
        MoveHunter;
      end;
    end;
end.
```

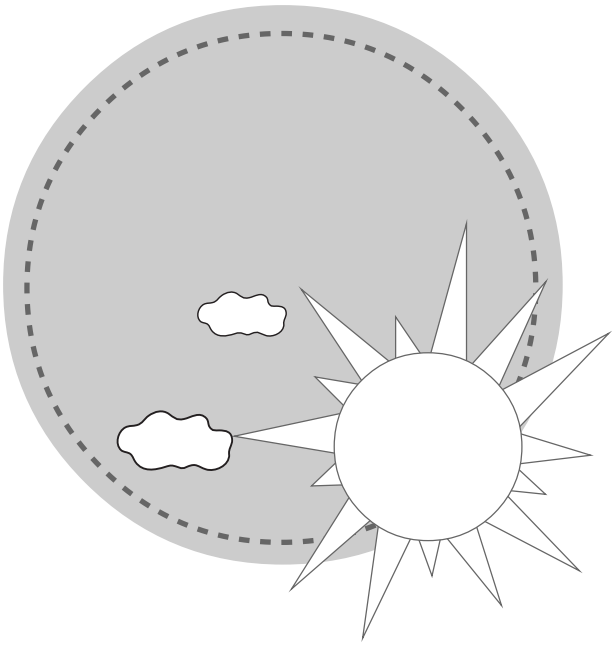


Затем папа объяснил:

— Охотник может направиться туда, куда мы его попросим. Но может и не пойти. Например, он не перешагнет через границы экрана.



Лишь после такой проверки функция `MoveHunter` поместит охотника в указанное место. А без проверки он будет стоять на месте.



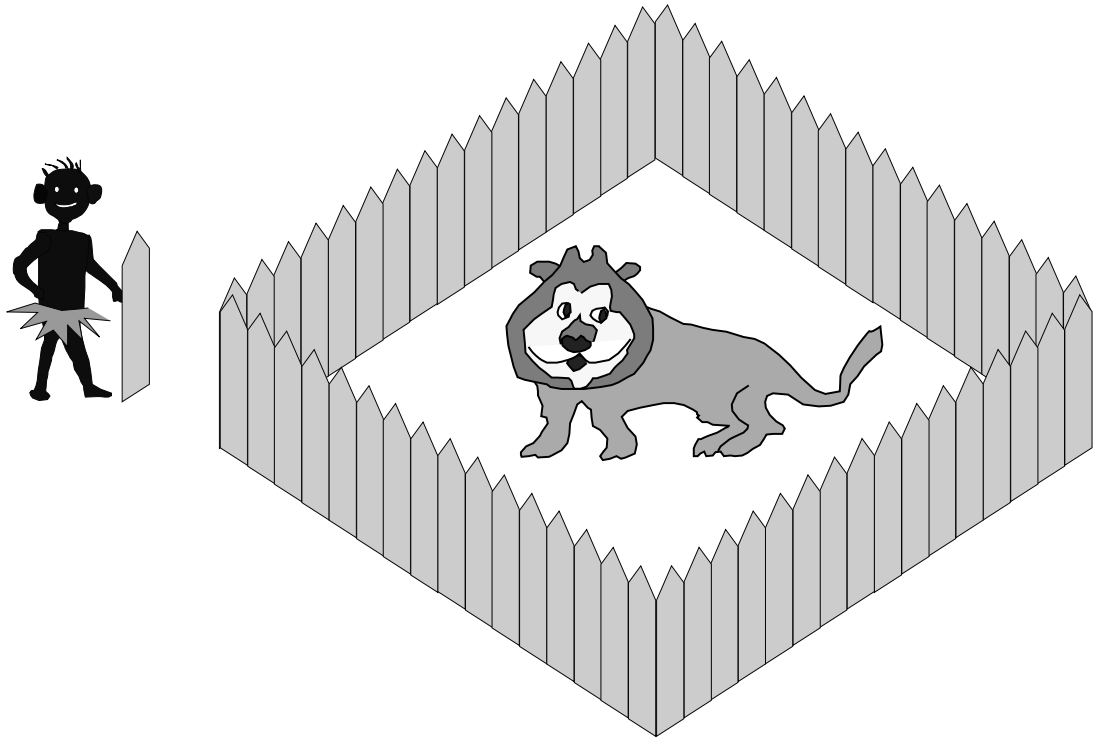
ГЛАВА 20.

МЫ ПОМОГАЕМ ОХОТНИКУ ЛОВИТЬ ЛЬВОВ

Мы снова запустили программу. На экране появились охотник, лев и муха Цеце. Мы с Таней договорились играть по очереди. Когда я нажимал клавиши, охотник ловко строил заборы в саванне. Он очень осторожно заходил в саванну: несколько шагов вверх и тут же назад, в пустыню. Охотник рискует, оставаясь надолго в саванне: лев может его съесть. Но и в пустыне нельзя долго находиться. Здесь летает муха Цеце, она тоже опасна.

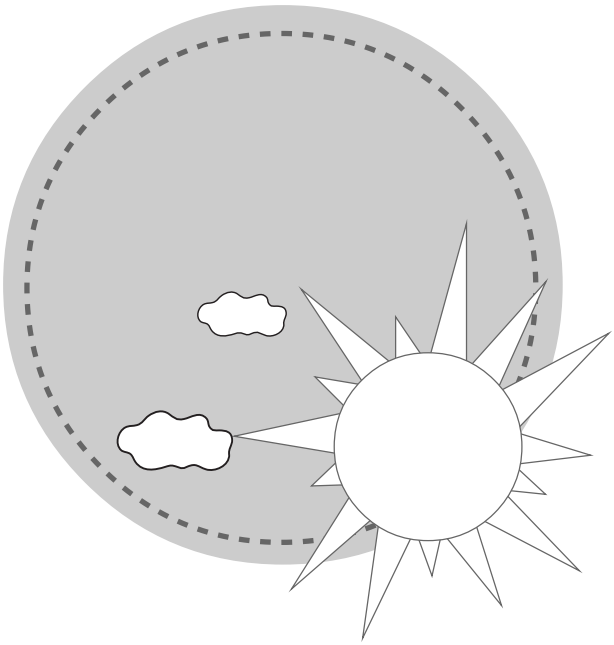
Однако охотник с каждым шагом становился смелее и смелее. Он научился строить западни для льва. Узнал много мест в пустыне, куда не могла залететь муха, и отлично прятался в углах экрана. Для него важно захватить

как можно больше саванны. Ему даже удавалось занять сразу половину этой площади, а потом построить заграждение и загнать туда льва. С одним львом мой охотник теперь справлялся легко. А вот Таниному охотнику часто не везло. Лев его съедал, муха кусала. Компьютер грустно сообщал об этом. Но чем дольше Таня играла, тем проворней становился охотник. Он научился строить забор так быстро, что лев сразу оказывался в ловушке. И тогда охотник в саванне мог пойти в любое место.



Папа молча смотрел, пока мы играли. Потом весело сказал:

— Ребята, вы стали настоящими охотниками! По-моему, пора сделать нашу сказку еще интереснее. Хотите, в саванне будет много львов?



ГЛАВА 21.

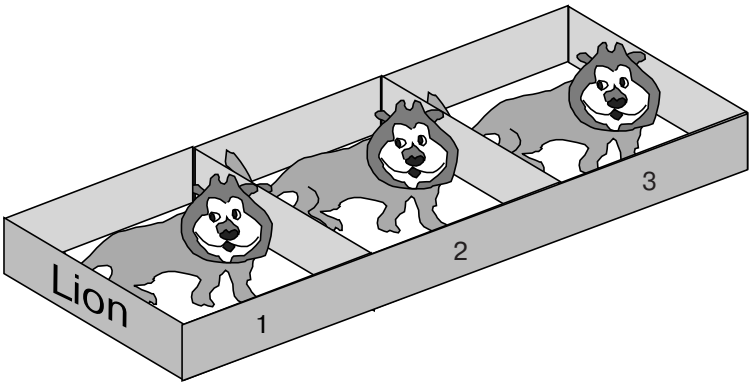
ЧТО ТАКОЕ МАССИВ

Мы решили поселить в Африке трех львов. А если справимся с ними, то пустим и других зверей. Папа сказал:

— В нашей программе можно описать переменные для трех львов. Например, поставить в конце имени переменной номер льва. А для ста и даже больше львов переменные займут много листов бумаги. Написать такую программу без ошибок сложно. Но профессионалы придумали способ, чтобы программа не занимала много места (об этом написано в главе 14). Там переменная `LionX` была представлена в памяти компьютера одним ящичком для хранения целого числа. Давайте нарисуем эту переменную в виде большого ящика, в который можно вложить одинаковые пронумерованные маленькие

ящички. Такой большой ящик называется *массивом*, а на его боковой стенке есть номера ящичков.

На рисунке мы увидели массив.



Но нам было непонятно, как можно использовать массив в программе. Папа предложил попросить у компьютера память для массива и сказал:

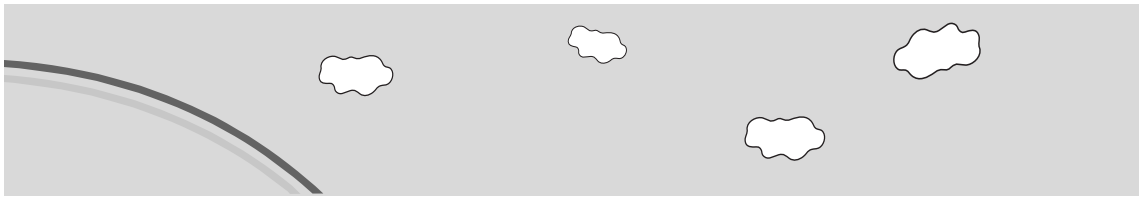
— Смотрите, как можно описать массив в программе. — И напечатал на экране дисплея следующие строчки:

```
(* Так мы писали в программе раньше. *)
var LionX : integer;

(* Теперь мы напишем по-другому. *)
const nLions = 3; (* Столько у нас львов. *)
var LionsX : array[1..nLions] of integer;
```

— Что это за ключевое слово `array`? — спросила Таня.

— Оно означает «массив», а слово `of` переводится как «из». Всю строчку можно прочитать так: переменная `LionsX` — это массив из трех целых чисел. Здесь мы будем хранить координату `x` для трех львов.

— Как класть в ящички массива числа и доставать их оттуда? — поинтересовался я.



— Чтобы, например, положить целое число в первый ящичек массива LionsX, нужно записать такую строчку:

```
LionsX[1] := 2;
```

А чтобы достать целое число из него, можно сделать так:

```
x := LionsX[1];
```

Потом папа спросил нас:

— Ребята, что означает вот эта строчка:

```
LionsX[k]:= 1;
```

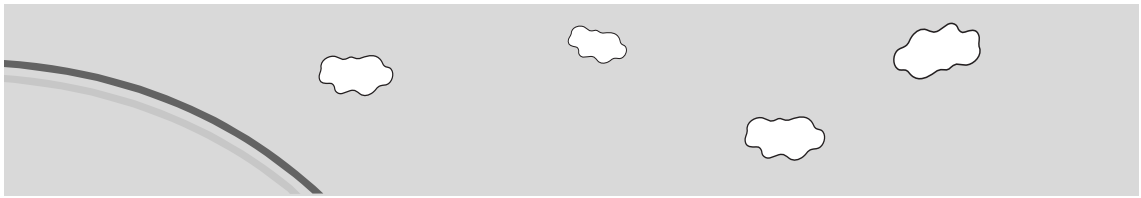
Я ответил:

— Эта строчка показывает, что ящичек в массиве имеет номер, который лежит в переменной k. И нам нужно положить в этот ящичек целое число — единицу.

— Молодец, Ваня, теперь вы поймете, как с помощью оператора цикла for можно пройти по всем ящичкам массива (об этом написано в главе 11). А теперь посмотрите, как будут выглядеть функции SetLions, MoveLions и MoveLionK. Функция SetLions расселяет всех львов в Африке. С помощью функции MoveLions они перемещаются, а функция MoveLionK водит льва с номером k. Смотрите, как они выглядят:

```
(*********************)
(* Львы с номерами. *)
(*********************)
const
nLions = 3; (* Столько всего у нас львов. *)

(* Массивы для координат x и y львов. *)
var LionsX : array[1..nLions] of integer;
var LionsY : array[1..nLions] of integer;

(* Массивы для данных о направлении движения львов. *)
var LionsDX : array[1..nLions] of integer;
var LionsDY : array[1..nLions] of integer;
```

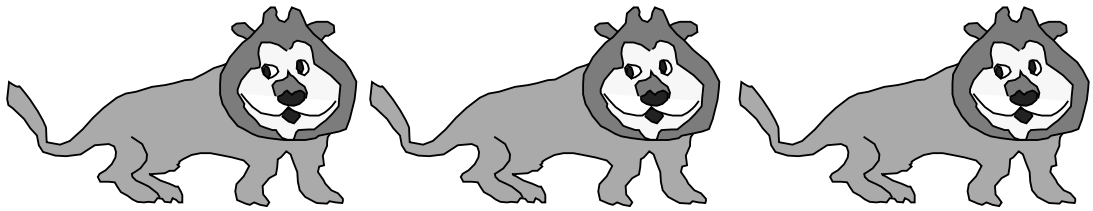
```
(*************************************************)
(* Эта функция водит льва с номером k по саванне. *)
(*************************************************)
procedure MoveLionK(k : integer);
var
nextColor,              (* Цвет следующей клетки. *)
nextColorX,             (* Цвет клеток *)
nextColorY : integer;   (* слева и справа от следующей. *)

begin
(*************************************************)
(* Проверим, не пытается ли лев выйти из саванны. *)
(* Посмотрим, какого цвета следующая клетка       *)
(* и соседние с ней клетки на экране.             *)
(*************************************************)
nextColor := Africa[LionsX[k] + LionsDX[k]][LionsY[k] + LionsDY[k]];
nextColorY     := Africa[LionsX[k]][LionsY[k] + LionsDY[k]];
nextColorX     := Africa[LionsX[k] + LionsDX[k]][LionsY[k]];
(************************)
(* Лев делает один шаг. *)
(************************)
(* Если лев наткнулся на забор, *)
(* а охотник не в пустыне, *)
(* то лев съест охотника. *)
if nextColor = zaborColor then
begin
    Write(HunterSound); (* Охотник кричит. *)
    Write(HunterSound); (* Охотник кричит. *)
    Write(HunterSound); (* Охотник кричит. *)
    Halt; (* Программа заканчивается. *)
end;

if nextColor <> savanaColor then
(* Лев пытается выйти из саванны. *)
(* Попробуем повернуть в сторону. *)
begin
    if nextColorX = nextColorY then
    begin
        (* Повернем льва назад. *)
        LionsDX[k] := -LionsDX[k];
        LionsDY[k] := -LionsDY[k];
    end
```



```
    else
    if nextColorY = savanaColor then
        (* Попробуем свернуть в другую сторону по горизонтали. *)
        LionsDX[k] := -LionsDX[k]
    else
    if nextColorX = savanaColor then
        (* Попробуем свернуть в другую сторону по вертикали. *)
        LionsDY[k] := -LionsDY[k]
    end;

    (* Сначала сотрем льва на старом месте. *)
    TextBackground( Africa[LionsX[k]][LionsY[k]] );
    GotoXY(LionsX[k], LionsY[k]);
    WriteLn(" ");

    (* Вычислим новое место льва. *)
    LionsX[k] := LionsX[k] + LionsDX[k];
    LionsY[k] := LionsY[k] + LionsDY[k];

    (* Нарисуем льва на новом месте *)
    (* тем же цветом на прежнем фоне. *)
    TextBackground( Africa[LionsX[k]][LionsY[k]] );
    TextColor(LionColor);
    GotoXY(LionsX[k], LionsY[k]);
    WriteLn(LionSym);
end;

(*****************************)
(* Поселим львов в саванне. *)
(*****************************)
procedure SetLions;
var k : integer;
begin
    for k:=1 to nLions do
    begin
        (* Это начальное положение львов. *)
        (* Они будут бегать в одной группе. *)
        LionsX[k] := minScreenX + k;
        LionsY[k] := minScreenY + 5;

        (* Сначала львы бегут вправо и вниз. *)
        LionsDX[k] := 1;
        LionsDY[k] := 1;
    end;
end;
```

```
(*****************************)
(* Отведем львов на один шаг. *)
(*****************************)
procedure MoveLions;
var k : integer;
begin
    for k:=1 to nLions do MoveLionK(k);
end;
```

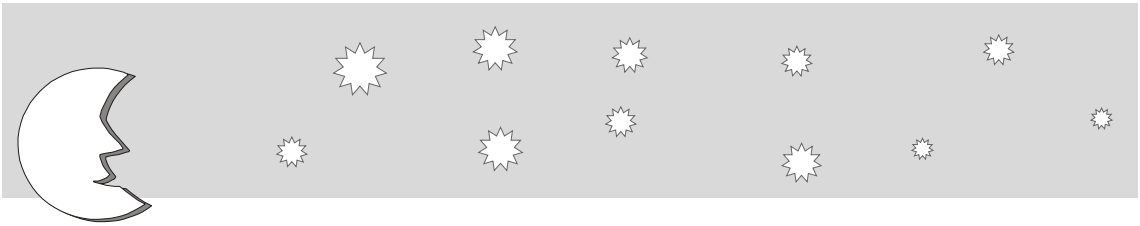
Запустив программу, мы увидели на экране свою сказку. Теперь в Африке жили три льва.



Они быстро бегали. Охотнику стало тяжелее строить заборы в саванне. Я придумал новые ловушки для львов. Мой охотник огораживал длинные и узкие участки. Получались как бы коридоры, внутри которых он находился. Здесь львы были ему не страшны. Когда один лев забегал в такую ловушку, охотник успевал перегородить ему выход. И лев оказывался в западне! Его можно было отправлять в зоопарк.

Потом у нас стало 70 львов. Они важно ходили рядом, пока не натыкались на забор. Тогда некоторые принимались бродить поодиночке. Танин охотник был менее удачлив: его часто съедали. Я научил ее строить коридоры. Но все же Таня устала быть охотником и сказала:

— Папа, я не хочу, чтобы львы съедали охотника. Пусть в саванне поселятся антилопы.

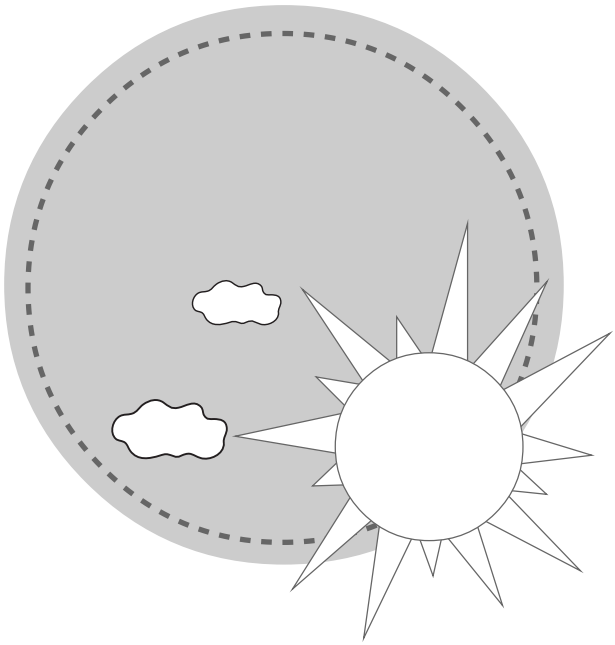


Я стал возражать:

— Не хочу, чтобы в Африке бегали антилопы. Пусть лучше здесь будут другие хищные звери. Я помогу охотнику их поймать.

— Не ссорьтесь, ребята, — сказал папа. — Тот, кто научится писать программы, сможет поместить в сказку любых животных.

Мы с Таней помирились и решили, что чем больше у нас зверей, тем интересней игра. Например, бегемот не так опасен для Таниного охотника. Этот зверь большой и неповоротливый. Папа сказал, что бегемот будет передвигаться в два раза медленнее, чем лев.

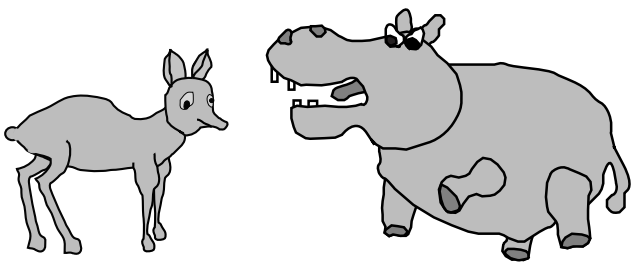


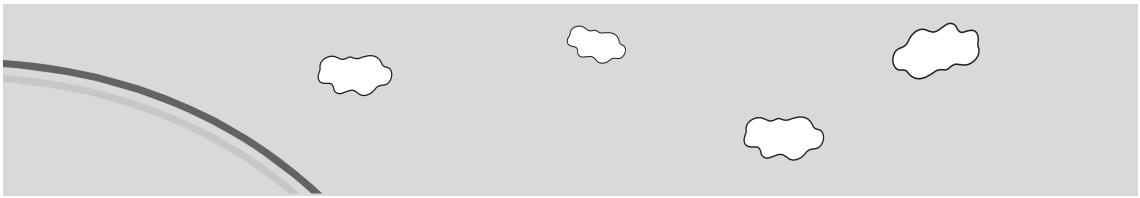
ГЛАВА 22.

ПАПА РАССКАЗЫВАЕТ О СТРУКТУРАХ ЯЗЫКА ПАСКАЛЬ

Я думал, что в программе мы напишем две функции — `MoveAntilopa` и `MoveBegemot`. Но папа возразил:

— Конечно, мы можем написать и так. Однако программисты придумали более удобный способ. Он

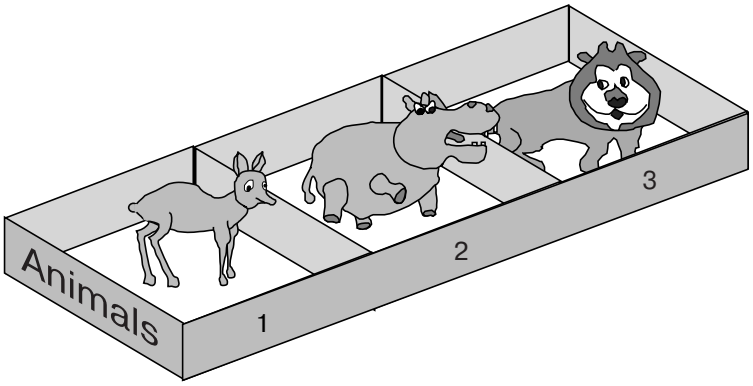




позволяет поселить в Африке самых разных животных. Для этого нам нужно научиться описывать в программе *структуры*.

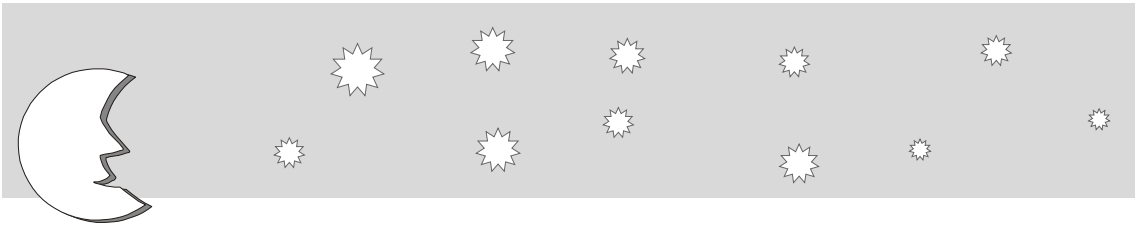
— Что же такое структура? — хором спросили мы.

— Давайте вспомним, — сказал папа, — сначала в нашей программе были описаны ящички-переменные. Затем вы научились соединять и складывать такие одинаковые ящички в массив (большой ящик). А структура — это большой ящик, в который можно вложить разные маленькие ящички-переменные. Например, в этот большой ящик поместим координаты `x` и `y` зверя, направление его движения и имя. Посмотрите...



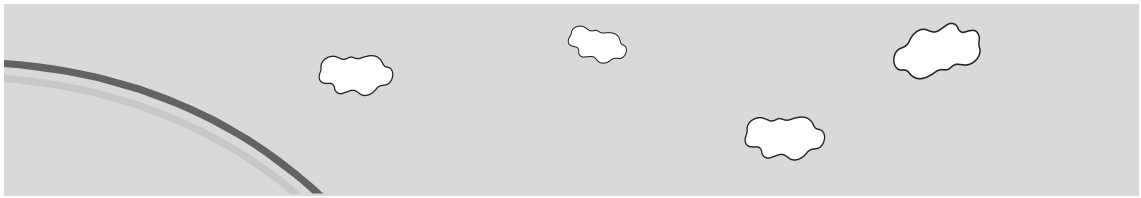
На рисунке был большой ящик с маленькими ящичками-переменными. На ящичках надписи: первая буква названия зверя — `Name`, его координаты — `x` и `y`, направления движения — `DX` и `DY`, символ, которым обозначается животное, — `Symbol` и его цвет — `Color`. Папа объяснил:

— Одной такой структуры хватит для описания множества разных животных. Чтобы поселить в Африке



одного из них, достаточно в первый ящичек-переменную положить его букву-имя, например А (антилопа) или Б (бегемот). А в последний ящичек структуры нужно поместить его символ на экране. В программе структура записывается так:

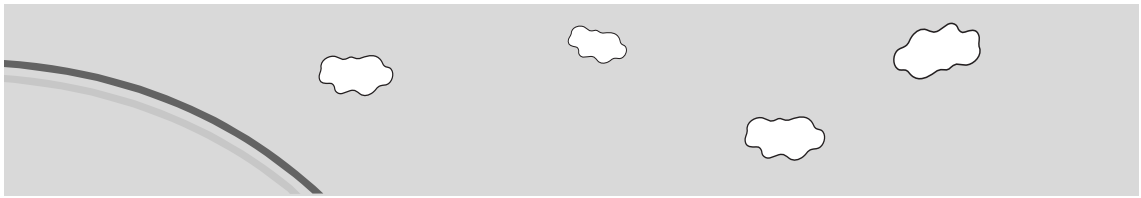
```
(***************************)
(* Животные с их номерами. *)
(***************************)

(* Это описание животного. *)
type Animal =
record
    Name : char;          (* Имя животного. *)
    X : integer;          (* Координаты животного. *)
    Y : integer;
    DX : integer;         (* Направление движения животного. *)
    DY : integer;
    Color : integer;      (* Цвет животного. *)
    Symbol : char;        (* Символ животного. *)
end;
const
(* У нас есть три синих бегемота. *)
nHipopotamus        = 3;
HipopotamusColor    = Blue;
HipopotamusSym      = Chr(5);

(* У нас есть четыре белых антилопы. *)
nAntilops = 4;
AntilopaColor  = White;
AntilopaSym    = Chr(6);

nAnimals = nLions + nHipopotamus + nAntilops;

(* Есть массив для животных. *)
var Animals : array[1..nAnimals] of Animal;

(* Что должен делать бегемот, лежать или идти, *)
(* определяется этой логической переменной. *)
var standUpHipopotamus : boolean;
```



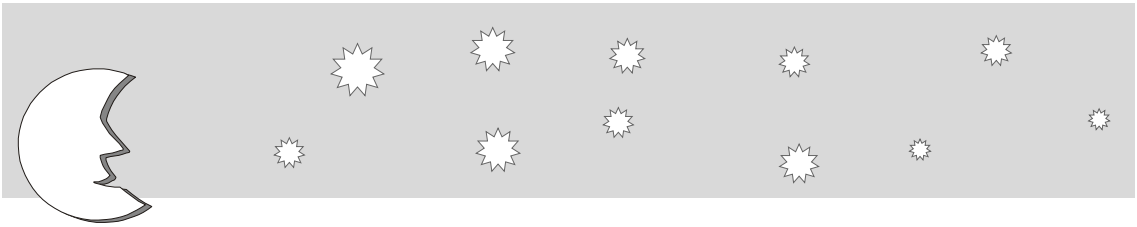
Функции, которые будут селить и водить зверей, выглядят так:

```
(*********************************)
(* Пусть звери живут в саванне. *)
(*********************************)
procedure SetAnimals;
var k : integer;
begin
    for k := 1 to nLions do
    begin
        (* Это начальное положение львов. *)
        (* Они будут ходить группой. *)
        Animals[k].X := minScreenX + 2 * k;
        Animals[k].Y := minScreenY + 5;

        (* Сначала львы идут направо и вниз. *)
        Animals[k].DX         := 1;
        Animals[k].DY         := 1;
        Animals[k].Symbol     := LionSym;
        Animals[k].Color      := LionColor;
        Animals[k].Name       := "L";
    end;

    (* Сначала бегемоты бегут. *)
    standUpHipopotamus := true;

    for k := nLions to nLions + nHipopotamus do
    begin
        (* Начальное положение бегемотов. *)
        (* Они бегают стадом. *)
        Animals[k].X := minScreenX + 5 + 3 * k;
        Animals[k].Y := maxScreenY - 5;

        (* Сначала бегемоты бегут вверх и направо. *)
        Animals[k].DX         :=  1;
        Animals[k].DY         := -1;
        Animals[k].Symbol     := HipopotamusSym;
        Animals[k].Color      := HipopotamusColor;
        Animals[k].Name       := "H";
    end;

    for k := nLions + nHipopotamus to nLions + nHipopotamus + nAntilops do
    begin
        (* Начальное положение антилоп. *)
```

```
        (* Они бегают стадом. *)
        Animals[k].X := maxScreenX - 5 - 4 * k;
        Animals[k].Y := maxScreenY - 5;

        (* Сначала антилопы бегут вверх и направо. *)
        Animals[k].DX        :=  1;
        Animals[k].DY        := -1;
        Animals[k].Symbol    := AntilopaSym;
        Animals[k].Color     := AntilopaColor;
        Animals[k].Name      := "A";
    end;
end;

(*********************************************)
(* Эти функции управляют животными в саванне. *)
(*********************************************)
procedure MoveAnimalK(k : integer);
var
nextColor,              (* Цвет следующего поля. *)
nextColorX,             (* Цвета следующих полей *)
nextColorY : Byte;      (* слева и справа. *)

begin
    (* Если это бегемот и если раньше он ходил, *)
    (* то пусть теперь полежит. *)
    if (Animals[k].Name = "H") and standUpBegemot
    then exit; (* Уйдем из функции. *)

    (*******************************************************)
    (* Проверим, не пытается ли животное уйти из саванны.  *)
    (* Посмотрим на цвет следующего поля и соседних полей. *)
    (*******************************************************)
    nextColor  := Africa[Animals[k].X + Animals[k].DX]
                  [Animals[k].Y + Animals[k].DY];
    nextColorY := Africa[Animals[k].X][Animals[k].Y + Animals[k].DY];
    nextColorX := Africa[Animals[k].X + Animals[k].DX][Animals[k].Y];

    (*****************************)
    (* Животное делает один шаг. *)
    (*****************************)
    (* Если животное двигается навстречу охотнику, а охотник в саванне, *)
    (* животное (не антилопа) может съесть охотника. *)
    if (nextColor = zaborColor) (* Животное столкнулось с охотником. *)
        and
        (Animals[k].Name <> "A") (* Антилопа не ест охотника. *) then
    begin
```

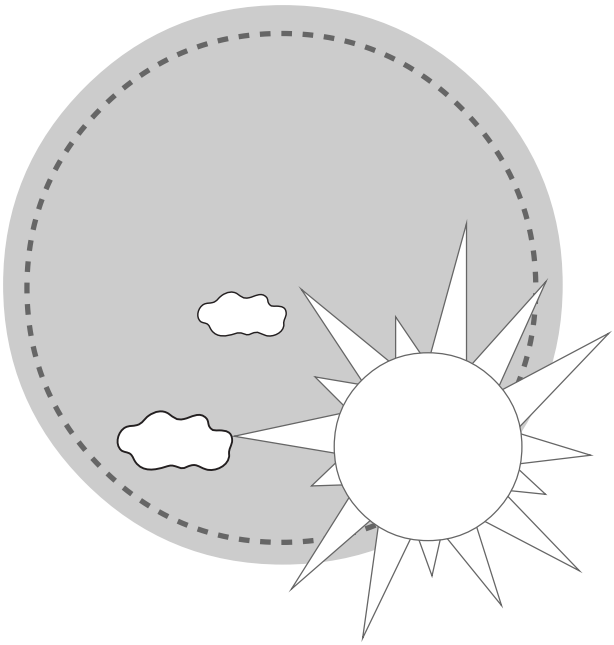


```
        Write(HunterSound); (* Охотник кричит. *)
        Write(HunterSound); (* Охотник кричит. *)
        Write(HunterSound); (* Охотник кричит. *)
        Halt; (* Игра заканчивается. *)
    end;

    if nextColor <> woodColor then
    (* Животное пробует выйти из саванны. *)
    (* Вернем его обратно. *)
    begin
        if nextColorX = nextColorY then
        begin
          (* Повернем животное назад. *)
          Animals[k].DX := -Animals[k].DX;
          Animals[k].DY := -Animals[k].DY;
        end
        else
        if nextColorY = savanaColor then
          (* Повернем по горизонтали. *)
          Animals[k].DX := -Animals[k].DX
        else
        if nextColorX = savanaColor then
          (* Повернем по вертикали. *)
          Animals[k].DY := -Animals[k].DY
    end;

    (* Сначала сотрем животное на старом месте. *)
    TextBackground( Africa[Animals[k].X][Animals[k].Y] );
    GotoXY(Animals[k].X, Animals[k].Y);
    WriteLn(" ");

    (* Вычислим новое положение животного. *)
    Animals[k].X := Animals[k].X + Animals[k].DX;
    Animals[k].Y := Animals[k].Y + Animals[k].DY;

    (* Нарисуем животное на новом месте. *)
    TextBackground( Africa[Animals[k].X][Animals[k].Y] );
    TextColor(Animals[k].Color);
    GotoXY(Animals[k].X, Animals[k].Y);
    WriteLn(Animals[k].Symbol);
end;
```



```
(*****************************************)
(* Пусть все животные сделают один шаг. *)
(*****************************************)
procedure MoveAnimals;
var k : integer;
begin
    (* Если бегемоты лежали, пусть походят. *)
    (* Если бегемоты ходили, пусть полежат. *)
    standupHipopotamus := not standupHipopotamus;
    for k := 1 to nAnimals do
        MoveAnimalK(k);
end;
```

Игра «Африка» получилась очень интересной. Даже не верится, что мы с Таней смогли запрограммировать придуманную сказку. Но это еще не все.



ГЛАВА 23.

МЫ ГОТОВИМСЯ К КОСМИЧЕСКОМУ ПУТЕШЕСТВИЮ

Компьютер был доволен. Он сказал:

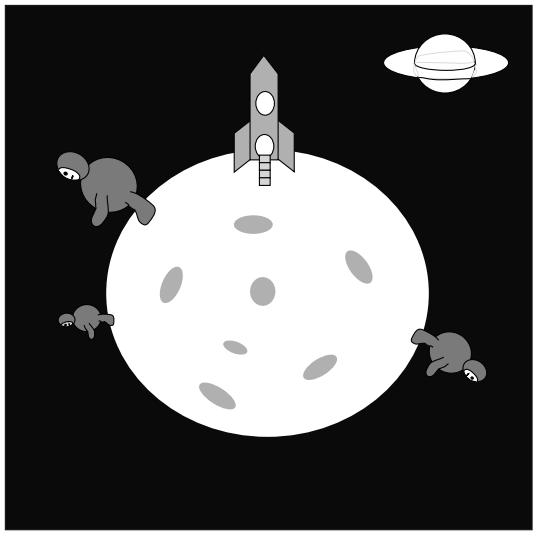
Я ваши жду фантазии
В Америке и в Азии.
Не хватит частей света,
Мала будет планета,
Попробуйте до космоса
Себе построить мост.

На экране вдруг появилось звездное небо.

Мы поняли, что компьютер ждет от нас новых программ. Ведь не зря же он подарил нам это чудо.

— Посмотри, сколько звезд! — радостно воскликнула Таня.

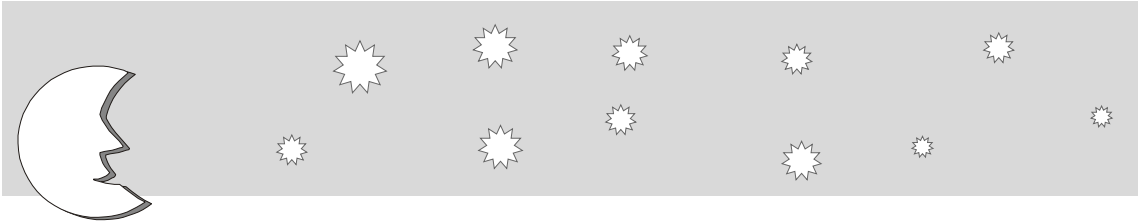
Звезды мерцали. Они были как настоящие. Компьютер не зря сказал о космосе. Вот куда в следующий раз нужно отправиться в путешествие! Только трудно сразу определить маршрут. Потому что кроме планет (мы уже знали Марс, Венеру, Юпитер и Сатурн) есть еще и созвездия. Компьютер подсказал их названия: Змея, Голубь, Рысь, Жираф, Рыбы, Дракон, Дельфин, Лев, Конь, Ворон и многие другие.



— Давайте придумаем путешествие по Луне, — неожиданно предложила Таня. — Мы будем управлять космонавтом, как охотником в Африке.

— Нет, лучше отправимся на космическом корабле к Сатурну. Я читал, что около этой планеты кружится много метеоритов. Подлетать к Сатурну нужно осторожно, чтобы не повредить корабль.

— Ребята, космические путешествия требуют хорошей подготовки. Чтобы побывать на Луне, нужно почитать учебники по физике и астрономии. После этого вы лучше поймете, как космонавт будет ходить по другой планете. Такие игры станут очень увлекательными, когда вы научитесь рисовать картинки в *графическом режиме*.



И тогда космонавт, космический корабль и кратеры вулканов на Луне будут на экране дисплея совсем как настоящие.

— Вот было бы здорово! — закричали мы.

Папа пообещал нам помочь. Наш друг-компьютер очень устал. Мы попрощались с ним. Потом папа нажал какие-то клавиши и выключил его.

ПОСЛЕСЛОВИЕ

Для создания предлагаемой игры использовался компилятор Borland Pascal версии 7.0. Хотя игра предназначена для DOS, она будет работать и с другими операционными системами: Windows 95, Windows 98, Windows NT.

Если вы хотите запускать свои игры с операционными системами Windows, транслируйте их в защищенном режиме. Для этого в компиляторе Borland Pascal выберите меню:

Compile ⇒ Target ⇒ Protected mode application

Тексты программ, приведенные в этой книге, можно переписать с сайта издательства: http://www.dmk.ru.

ВНИМАНИЕ!

Вы можете стать активным участником процесса книгоиздания в России!

Заполните предлагаемую анкету и укажите, книги на какие темы Вас интересуют.

При полном заполнении анкеты Вы получаете возможность в течение 6 месяцев пользоваться **10% скидкой** при приобретении компьютерной и радиотехнической литературы, имеющейся в нашем Internet-магазине, выпускаемой как «ДМК Пресс», так и другими издательствами (всего 700 наименований). Для этого достаточно получить логин и пароль, которые Вы будете использовать при входе в Internet-магазин на сайте www.dmk.ru. Узнать логин и пароль Вы можете, позвонив по тел. 369-33-60 или прислав письмо по электронной почте (info@dmk.ru) после отправки анкеты по адресу: 105023, Москва, пл. Журавлева, д. 2/8, оф. 400.

1. Где Вы приобрели эту книгу? _____
(в магазине (адрес), на рынке, у знакомых)

2. Вы приобрели эту книгу за ___ руб. **Это** очень дорого ☐ приемлемо ☐ дешево ☐

3. Оцените по 5-балльной системе:	**1**	**2**	**3**	**4**	**5**
а) качество выполнения иллюстраций	☐	☐	☐	☐	☐
б) качество изложения материала	☐	☐	☐	☐	☐
в) актуальность рассмотренных тем	☐	☐	☐	☐	☐
г) общее впечатление от книги	☐	☐	☐	☐	☐

4. С какой целью Вы приобрели эту книгу?

а) книга необходима Вам по работе ☐
б) для самостоятельного изучения предмета ☐
в) почитать для общего развития ☐

5. Если Вы обнаружили какие-либо ошибки или неточности в книге, пожалуйста, перечислите их ниже (с указанием номеров страниц)_____
_____
_____

077-4 Как ребята программировали игру «Африка»

Предлагаем Вам принять участие в подготовке наших будущих изданий. Для этого ответьте на нижеследующие вопросы:

Наиболее актуальные, по Вашему мнению, темы компьютерной и радиотехнической литературы
_____
_____
_____

Основные аспекты, которые, с Вашей точки зрения, должны быть подробно рассмотрены в книге
_____
_____

Примерный объем (число страниц)_____
Уровень читателя (начинающий, опытный пользователь, ...) _____
Приемлемая для Вас цена _____ **руб.**

Мы приглашаем авторов к сотрудничеству. Пишите по адресу: books@dmk.ru. Факс: 369-78-74

Темы книг по компьютерам и электронике

Укажите в анкете любую из нижеперечисленных тем или предложите свой вариант

Учебник (в помощь пользователю)	Самоучитель IBM PC Шифрование (алгоритмы) Алгоритмы сжатия (компрессии) данных Спецификации AGP, PCI и т.п. Локальные сети Материнские платы Пакеты компьютерной верстки (Adobe FrameMaker 5.5, Corel Ventura 8.0, MS Publisher) WinFaxPro и другие «маленькие помощники» Совместная работа MacIntosh+Windows Системы и протоколы безопасных транзакций через Internet, ID-systems DVD E-commerce, E-marketing, E-business Novell Netware 5.0
Для программиста	C++, Pascal, Delphi, FoxPro, ... Отладка программ (дебаггеры) Программирование игр для PC Программирование для приложений распознавания речи OpenGL, DirectX – программирование приложений VBA for Office 2000 Oracle 8i, Solaris, Clarion 5.0 Программы компании «1С» ASP, HTML, XML, Java, JavaScript SQL, SQL Server 2000
Операционные системы	Windows, UNIX, Linux, EPOC, Palm OS, Mac OS, ...
Компьютерная графика и анимация	Adobe inDesign, Adobe Photoshop 5.0, 3D MAX, Fractal Design Painter 5.0, LightWave, Corel Draw 9.0, Maya 2.5, SoftImage 3D, Macromedia Flash, Bryce ...
В помощь проектировщику	Пакеты схемотехнического моделирования и проектирования печатных плат (Workbench, Accel Eda 14.0, ORCAD, ...) Различные пакеты САПР (Mechanical Desktop, Real Architect 3.8, ...) Системы архитектурного проектирования (ArchiCAD, ...) CASE-средства (ErWin, BpWin, Rational Rose)
В помощь радиолюбителю	Электронные охранные устройства Справочник по радиолампам Автомобильная электроника Акустика и акустические системы Книги для начинающих радиолюбителей Микроконтроллеры и ОЭВМ (программирование) Радиолюбительские конструкции Спутниковое телевидение
Ремонт бытовой и радиоаппаратуры	Модернизация и ремонт ПК Ремонт устройств бытовой техники Мини-АТС Охранные системы для автомобилей Справочник по бытовой радиотехнике и радиодеталям СВ радиосвязь Миноискатели Радиопередатчики СВЧ схемотехника

ИЗДАТЕЛЬСТВО DMKPress ПРЕДОСТАВЛЯЕТ ВАМ

возможность приобрести интересующие Вас книги, посвященные компьютерным технологиям и радиоэлектронике, самым быстрым и удобным способом. Для этого Вам достаточно всего лишь посетить Internet-магазин «ДМК Пресс» по адресу **www.dmk.ru**. Вашему вниманию будет представлен самый полный перечень книг по программированию, компьютерному дизайну, проектированию, ремонту радиоаппаратуры, выпущенных в нашем и других издательствах. В Internet-магазине Вы сможете приобрести любые издания, не отходя от домашнего компьютера: оформите заказ, воспользовавшись готовым бланком, и мы доставим Вам книги в самый короткий срок по почте или с курьером.

Internet-магазин на www.dmk.ru

- экономит Ваше время, позволяя заказать любые книги в любом количестве, не выходя из дома;
- избавляет Вас от лишних расходов: мы предлагаем компьютерную и радиотехническую литературу по ценам значительно ниже, чем в магазинах;
- дает возможность легко и быстро оформить заказ на книги – как новинки, так и издания прошлых лет, пользующиеся постоянным спросом.

Если Вы живете в Москве, то доставка с курьером позволит Вам увидеть книгу перед покупкой. При этом Вам не придется пользоваться кредитными картами или оплачивать почтовые услуги.

www.dmk.ru

Романова Людмила Борисовна,
Романов Владимир Юрьевич

Как ребята программировали игру «Африка» на языке Паскаль

Главный редактор *Захаров И. М.*
Выпускающий редактор *Левицкая Т. В.*
Технический редактор *Прока С. В.*
Верстка *Булыгина А. А.*
Графика *Бахарев А. А.*
Дизайн обложки *Панкусова Е. Н.*

Гарнитура «Балтика». Печать офсетная.
Усл. печ. л. 14. Тираж 5000. Зак. №

Издательство «ДМК Пресс», 105023, Москва, пл. Журавлева, д. 2/8.

Отпечатано в типографии № 9,
Волочаевская, 40.