Сергей Мельников

DELPHI и TURBO PASCAL на занимательных примерах

Санкт-Петербург
«БХВ-Петербург»
2006

УДК 681.3.06
ББК 32.973.26-018.1
М48

Мельников С. В.

М48 Delphi и Turbo Pascal на занимательных примерах. — СПб.: БХВ-Петербург, 2006. — 448 с.: ил.

ISBN 5-94157-886-5

Рассмотрены примеры решения нестандартных задач в Delphi и Turbo Pascal: словесные игры и головоломки, решение олимпиадных задач, разбор удивительного парадокса У. Пенни и головоломки известного американского автора Э. Фридмана, создание оригинальных игр в Turbo Pascal и Delphi с использованием Flash, Windows API и DirectX в полноэкранном режиме. Показано применение библиотеки регулярных выражений PCRE — Perl Compatible Regular Expressions — в Delphi для синтаксического разбора текста и арифметических выражений. Компакт-диск содержит исходные тексты всех программ, а также полный словарь существительных русского языка и другие вспомогательные файлы.

Для учащихся, студентов и программистов

УДК 681.3.06
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Анна Кузьмина*
Компьютерная верстка	*Натальи Смирновой*
Корректор	*Наталия Першакова*
Дизайн обложки	*Инны Тачиной*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 30.08.06.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 36,12.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Санитарно-эпидемиологическое заключение на продукцию № 77.99.02.953.Д.006421.11.04
от 11.11.2004 г. выдано Федеральной службой по надзору
в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 5-94157-886-5

Оглавление

Введение

- О чем эта книга
- Структура и особенности книги

Существует немало книг для изучающих язык Pascal и среду программирования Delphi. Видимо, многие из них написаны преподавателями (возможно, вчерашними студентами по своим конспектам), потому что эти книги содержат лишь учебные примеры на закрепление пройденного материала. В этой книге вы увидите решение интересных задач, которые возникали в моей практике. Это не работа с базами данных и бухгалтерией — это создание игр и головоломок, решение задач из области занимательной математики и другая творческая работа. Эта книга поможет вам найти увлекательное занятие в изучении программирования. Вы сможете натренировать свою сообразительность и накачать мышцы, которые шевелят извилинами, а если вы — смелый и творческий человек, то научитесь воплощать свои оригинальные идеи в законченных программах.

Когда человек, решая головоломку, проявляя находчивость и целеустремленность, преодолевает хитроумные препятствия и находит выход из тупиков мысли, то он испытывает творческое удовлетворение и чувствует себя интеллектуальной личностью. Кто же тогда тот человек, который придумал эту головоломку и создал интересную программу для ее решения? Он — творец, находящийся, как олимпийский бог, на вершине этой области человеческой деятельности. Вы сможете создавать свои шедевры творческой мысли и доносить их до потребителей интеллекта во всем мире с помощью технологий Интернета (ну и, конечно, что-то на этом зарабатывать).

Надеюсь, что каждый читатель найдет в этой книге и на прилагаемом к ней компакт-диске что-то интересное и новое для себя.

Обзор книги

- *Глава 1.*

  Эта глава посвящена программированию в среде Turbo Pascal. В ней мы рассмотрим алгоритмы и программы вывода всех перестановок, размещений и сочетаний из N предметов. Для перестановок рассмотрим олимпиадный и более продвинутый алгоритм их получения. Напишем программу для моделирования удивительного нетранзитивного парадокса, описанного в книге Мартина Гарднера "Путешествие во времени". Займемся поиском словесных рекордов, для чего на компакт-диске имеется проверенный словарь существительных русского языка более чем из 42 000 слов. Это рекорд из области анаграмм и квадратов слов. Изучим рекурсивный и нерекурсивный алгоритмы перебора с возвратами на

примере поиска гамильтоновых циклов в графе и решения оригинальной олимпиадной задачи о блохе. Просчитаем головоломку Full Board известного американского мастера составления головоломок Эриха Фридмана и получим на зависть автору много позиций с единственным решением. Напишем свою разновидность игры "Ним", свободную от простого алгоритма нахождения выигрывающего хода.

- *Глава 2.*

  В этой главе мы перейдем к Delphi 7 и с помощью ее стандартных компонентов напишем оригинальную логическую игру Calculator. В ее маленькой, но мощной процедуре поиска выигрывающего хода также используются рекурсия и перебор с возвратами.

- *Глава 3.*

  В ней мы рассмотрим приемы программирования в Windows API без библиотеки визуальных компонентов VCL. Начав с разбора шаблона минимальной программы Delphi, рассмотрим работу под DirectX и закончим написанием полноэкранной игры Domino Dilemma, которую вы можете увидеть на моем сайте **www.gameintellect.com**. (Не забудьте перед этим отключить поддержку русского языка в браузере или сделать его вторым. Для Internet Explorer это можно сделать через меню **Сервис | Свойства обозревателя**, вкладка **Общие**, кнопка **Языки...**).

- *Глава 4.*

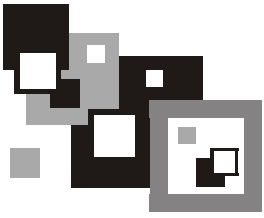
  Здесь мы займемся использованием технологии Flash в Delphi и напишем Flash-версию игры Calculator. Процедура нахождения выигрывающего хода будет находиться в оболочке Delphi, потому что ActionScript работает для этой цели недостаточно быстро, а вся игра будет сделана во Flash, программный код которой будет обмениваться информацией с Delphi-программой. Это все будет работать в одном exe-файле и выглядеть как обычное приложение Windows, но с Flash-графикой и звуком.

- *Глава 5.*

  Поиск и замена текста — задача, которая часто встречается при обработке данных. В *главе 5* мы рассмотрим применение для этой цели Perl-совместимых регулярных выражений (PCRE), и вы будете это делать мастерски. Эта библиотека дает при минимальных затратах времени огромные возможности для поиска и замены текста, заданного шаблонами. Аналогичную технологию вы можете видеть в редакторе Delphi при поиске и замене текста. Мы изучим правила составления таких шаблонов и опять столкнемся с рекурсией и перебором с возвратами. И в заключение с помощью PCRE напишем программу для синтаксического разбора и вычисления арифметических выражений по составленной нами LL(1)-грамматике для этих выражений. По аналогии вы сможете написать программу для трансляции или интерпретации программ с таких языков программирования, как Basic или Pascal, или разработать свой язык.

Компакт-диск, прикладываемый к книге, содержит все исходные тексты и исполняемые программы, разработанные в этой книге.

Глава 1



Практика в Turbo Pascal

- Исправление файла библиотеки turbo.tpl (turbo.tph) и задержка времени
- Перестановки, размещения и сочетания
- Моделирование удивительного парадокса
- Гамильтоновы циклы, рекурсия, просчет головоломки известного автора
- Анаграммы и квадраты слов
- Оригинальный вариант игры "Ним"

1.1. Преодоление ошибки "Runtime error 200"

Прежде чем начать программировать в Turbo Pascal, надо разобраться с одной известной ошибкой, которая встречается в некоторых версиях библиотеки turbo.tpl (turbo.tph для защищенного режима) Turbo Pascal и Borland Pascal. Если программа пытается использовать модуль `Crt` для работы с экраном и клавиатурой, то, начиная с процессора с тактовой частотой 200 МГц, она еще до начала исполнения вашего кода выдает ошибку переполнения при делении (Runtime error 200) и завершает свою работу. Виной тому ошибка разработчиков. Транслятор перед вашим кодом вставляет вызовы подпрограмм инициализации всех используемых модулей, и в секции инициализации модуля `Crt` происходит переполнение при делении.[1] Если у вас возникает эта ошибка, то проще всего взять шестнадцатеричный редактор файлов и заменить эту команду деления в указанном файле на нейтральные команды.

[1] Читайте об этом более подробно в статье Федора Меньшикова "Исправление ошибок в Паскале 7_00/7_01" в двух htm-файлах на прилагаемом к книге компакт-диске.

Найдите последовательность шестнадцатеричных байтов

```
b9 37 00 f7 f1
```

которые соответствуют командам

```
B93700 mov  cx,0037
F7F1   div  cx
```

и замените байты `F7`, `F1` на байты `90`, `90`. Тем самым вы заменяете команду, вызывающую ошибку, парой нейтральных команд, которые обменивают содержимое регистра `AX` с самим собой. При этом будьте внимательны и сначала поищите все такие последовательности байтов, чтобы случайно не заменить другой код. После этой замены уже не следует использовать процедуру `Delay`. Если вам нужна задержка на миллисекунды, то мы напишем более современную функцию для этой цели.

1.2. Процедуры задержки времени

Для задержки на время, кратное примерно 55 мс, можно воспользоваться ячейками по адресу `$40:$6C` из области данных BIOS (Basic Input/Output System, базовая система ввода/вывода), где расположен 4-байтовый счетчик времени, который увеличивается на единицу каждые 55 мс. Сама процедура задержки времени на число, кратное 55 мс, приведена в листинге 1.1.

Листинг 1.1. Процедура `Delay55`

```
procedure Delay55(ms55 : longint);
VAR
 biosct : longint absolute $40:$6C;
 ct : longint;
begin
      ct:=biosct+ms55;
      repeat until ct <= biosct;
end; {Delay55}
```

Замечание

Все программы для Turbo Pascal, представленные на прилагаемом к книге компакт-диске, работают в реальном режиме, транслируются 6-й версией этой системы, для этого я не использую операторы `break` и `continue`, которые появились в 7-й версии. На компакт-диске в папке chapter1 находится тестовая программа test55.pas (test55.exe) для проверки работы процедуры `Delay55`.

Она выводит 20 чисел с интервалом в 1 с. И еще: если вы хотите запускать программы для DOS в Windows XP, то это надо делать в полноэкранном режиме работы таких программ, как Far или DOS Navigator, иначе не будет видно, что наши примеры выводят на экран. То есть в свойствах окна программы на вкладке **Параметры** надо задать **Во весь экран**. После этого можно по нажатию комбинации клавиш <Alt>+<Tab> переключиться в оконный режим, и примеры сохранят работоспособность (хотя это не гарантируется; для Far надо в полноэкранном режиме что-то вывести на экран, а затем можно переключиться по <Alt>+<Tab> в оконный режим). Добраться до свойств окна программы можно из контекстного меню, нажав правую кнопку мыши на ярлыке программы или на заголовке ее окна. Можно также воспользоваться меню **Пуск | Все программы | Стандартные | Командная строка** и перевести окно нажатием комбинации клавиш <Alt>+<Tab> в полноэкранный режим. Для закрытия этого окна надо ввести команду `exit`.

А как быть, если нужна задержка меньше, чем на 55 мс? Для игр это часто бывает необходимо. Ведь BIOS откуда-то берет данные для счетчика времени, расположенного по адресу `$40:$6C`? Да, и берет она эти данные из микросхемы часов реального времени. Эта микросхема имеет несколько каналов, которые могут считать импульсы независимо от процессора. BIOS и Windows программируют эту микросхему для своих нужд. Канал 0 используется для счета времени суток. Он имеет шестнадцатиразрядный счетчик, который вначале содержит число 0 и с частотой примерно 1,19 МГц уменьшается на 2, обнуляясь дважды за 55-миллисекундный интервал. (Это рассуждение касается и систем Windows 9*x*. В DOS и Windows XP этот счетчик уменьшается на единицу и обнуляется один раз за 55-миллисекундный интервал.) При достижении нуля происходит аппаратное прерывание таймера `INT 8`. Процессор на миг отвлекается от выполнения программ и обновляет значение счетчика времени суток BIOS по адресу `$40:$6C`. Число 0 в этом счетчике соответствует полночи. Когда счетчик достигает значения, эквивалентного 24-м часам, он сбрасывается в ноль. В этот момент наша процедура `Delay55` может работать некорректно.

Для чтения регистра счетчика 0-го канала таймера надо послать в порт `43H` число `4`. (Мы здесь не будем разбирать, какие биты что означают в этом числе.) При этом значение счетчика копируется в буферный регистр, который и считывается из порта `40H`, ведь счетчик все время считает и очень быстро! Поэтому для чтения обоих байтов этого счетчика надо на время его заморозить.

Текст этой процедуры приведен в листинге 1.2.

Листинг 1.2. Процедура `DelayMs`

```
{ Задержка на заданное число миллисекунд по 0-му каналу таймера (для
  работы под Windows) }
```

```
procedure  DelayMs(ms : word); assembler;
        asm
        mov     cx,ms
        jcxz    @3
        { Берем в DX 1-й отсчет таймера }
@1:     mov     al,4
        out     43h,al
        in      al,40h          { читаем младший байт счетчика }
        mov     dl,al
        nop
        in      al,40h          { читаем старший байт }
        mov     dh,al
        { Берем в AX 2-й отсчет таймера }
@2:     mov     al,4
        out     43h,al
        in      al,40h
        mov     ah,al
        nop
        in      al,40h
        xchg    al,ah
        mov     bx,dx
        sub     bx,ax
        cmp     bx,1190*2       { прошла 1 мс? }
        jb      @2              { нет }
        loop    @1
@3:     end; {DelayMs}
```

Константа `1190*2` связана с частотой счета 1,19 МГц и тем, что счетчик считает по два. Для DOS и Windows 9*x* счетчик считает по одному, и поэтому эта константа должна равняться `1190`. Но продолжим наши занимательные исследования.

Замечание

На компакт-диске в папке chapter1 находится тестовая программа testms.pas (testms.exe) для проверки работы процедуры `DelayMs`. Она выводит 20 чисел с интервалом в 1 с. Это касается DOS и Windows 9*x*. Для Windows XP в той же папке находится аналогичная программа testmsxp.pas (testmsxp.exe).

В конце скажу еще об одной возможности отсчетов времени. В процессорах класса Pentium Pro появилась команда `rdtsc`, которая в регистрах `EDX:EAX` возвращает 8-байтное значение счетчика тактов центрального процессора со времени начала его работы. В `EDX` возвращается старшее двойное слово, а в `EAX` — младшее. Этот счетчик считает с нуля при начале работы процессора, и каждый такт его работы прибавляет к своему значению единицу. Но чтобы измерять им время, надо знать частоту процессора. В DOS это можно сделать довольно косвенно, засекая, сколько этот счетчик успеет насчитать за данное время, которое можно отмерять с помощью двух вышеприведенных процедур. Поэтому я даю только пример чтения этого счетчика в массив из четырех слов и вывода этого массива.

Как же в 16-разрядной программе работать с 32-разрядными регистрами? Оказывается, очень просто. Если процессор работает в шестнадцатиразрядном режиме, а мы хотим использовать в какой-то команде 32-разрядные данные, то перед данной командой надо вставить байт со значением `66H` (66 в шестнадцатеричной системе). Это префикс смены разрядности данных. А сама команда будет такой же, как и для шестнадцатиразрядных данных. Например, мы имеем команду

```
mov ax,mem16
```

Предполагаем, что `mem16` — это переменная типа `word` или `integer` из области данных программы. Эта команда загружает в регистр `AX` значение из `mem16`. Тогда команда

```
db 66H; mov ax,word ptr mem32
```

загрузит в регистр `EAX` 4-байтовое значение из адреса `mem32`. Это может быть переменная типа `longint`, `pointer` или какая-то переменная с индексом. Если длина типа этой переменной отлична от двух байтов, то перед ней необходимо написать `word ptr`, чтобы транслятор не выдал сообщение об ошибке. Ведь он не понимает, что мы хотим сделать.

Но не все 32-разрядные команды отличаются лишь этим префиксом `66H` от своих 16-разрядных двойников, иногда встречается отличие и в коде операции. Для выяснения этого нужно посмотреть листинг ассемблера для 32-разрядного и 16-разрядного вариантов команд.

Скажу еще больше: в 16-разрядном режиме DOS можно применять также 32-разрядную адресацию памяти и даже получить полный доступ ко всему 4-гигабайтному адресному пространству ОЗУ (оперативное запоминающее устройство). Но для этого нужно загрузить компьютер в DOS real mode (реальном режиме DOS) без программ типа emm386.exe, которые переводят процессор в виртуальный реальный режим работы. При этом, естественно, не будет механизма защиты памяти и каждая программа сможет делать все, что захочет. Этот финт, который получил название big real mode (большой реальный режим), может быть актуален лишь в очень узкой области приме-

нения, где нужны специальные операционные системы, поэтому мы его не обсуждаем. Ведь защищенный режим работы процессора и Windows дают почти то же самое, и вдобавок мультизадачность и механизм защиты программ друг от друга.

Мы рассмотрим пример программы, которая считывает значение счетчика тактов процессора командой `rdtsc` и выводит это длинное целое число в виде четырех слов, из которых оно состоит. Команда `rdtsc` состоит из двух байтов и имеет шестнадцатеричный код `0FH, 31H`. Мы будем 20 раз опрашивать счетчик тактов с задержкой, которую организуем с помощью рассмотренной процедуры `DelayMs`. Значение из регистров `EDX:EAX`, которые возвращает команда `rdtsc`, будем для простоты записывать в глобальный массив `rdcount`, состоящий из четырех слов. В листинге 1.3 приведен текст этой процедуры.

Листинг 1.3. Чтение счетчика тактов командой `rdtsc`

```
{ Возврат значения счетчика частоты ЦП }
procedure GetRDCount; assembler;
      asm
      db 0FH,31H { rdtsc }
      db 66h; mov word ptr rdcount,ax
      db 66h; mov word ptr rdcount+4,dx
      end; {GetRDCount}
```

Сам пример вы найдете на компакт-диске. Это файлы rdtsc.exe и rdtsc.pas. Программа 20 раз выводит значение счетчика тактов процессора с небольшой задержкой. Из вывода программы видно, как растет значение этого счетчика.

1.3. Перестановки, размещения и сочетания

В практике программирования иногда нужно организовать перебор вариантов, который не укладывается в схему циклов языков программирования. В одной программистской конференции я несколько раз встречал такой вопрос: как вывести все перестановки из N элементов (например, N первых натуральных чисел)? Если число элементов известно, то это можно сделать с помощью вложенных циклов. Пусть, к примеру, имеем четыре элемента 1, 2, 3, 4. Надо вывести все 24 их перестановки. Перестановка — это размеще-

ние друг за другом различных предметов. (Например, мы рассаживаем N программистов за N различными компьютерами.) То есть предметы должны отличаться друг от друга, и если это, к примеру, одинаковые шары, то они должны быть пронумерованы разными числами или значками. Всего существует $N!$ (Эн факториал) различных перестановок N элементов.

$$N! = N \times (N - 1) \times (N - 2) \times ... \times 2 \times 1.$$

Это легко доказывается: на первое место мы можем поставить любой из N предметов, это можно сделать N способами. Для каждой такой расстановки на второе место мы можем поставить любой из оставшихся $N - 1$ предметов. Получается $N \times (N - 1)$ способов разместить два предмета из N. Рассуждая дальше по аналогии, мы придем к представленной формуле. Число перестановок обозначается буквой P от французского слова *permutation* (перестановка, перемещение). Итак, пэ из эн элементов равно

$$P_n = n!$$

Еще заметим, что математики для удобства формально считают, что $0! = 1$.

1.3.1. Вывод перестановок

Для получения всех перестановок любых элементов (а в программе это будут элементы какого-то массива) достаточно научиться получать все перестановки из N первых натуральных или целых неотрицательных чисел. Действительно, ведь эти числа можно использовать как индексы для заданного массива элементов и, переставляя индексы, мы будем получать перестановки самих элементов массива.

Чтобы вывести все перестановки из четырех чисел, напишем четыре вложенные цикла как в листинге 1.4.

Листинг 1.4. Первая попытка вывода перестановок

```
VAR
 i1,i2,i3,i4: word;

BEGIN
for i1:=1 to 4 do
 for i2:=1 to 4 do
  for i3:=1 to 4 do
   for i4:=1 to 4 do
    Writeln(i1:2,i2:2,i3:2,i4:2);
END.
```

Запустив эту программку, мы увидим, что она выводит не только все перестановки, но и много чего еще, всего четыре в четвертой степени строк. Надо как-то отсеять строки с повторяющимися числами. Для этого в листинге 1.5 применим условный оператор.

Листинг 1.5. Удачная, но некрасивая попытка вывода перестановок

```
VAR
 i1,i2,i3,i4: word;

BEGIN
for i1:=1 to 4 do
 for i2:=1 to 4 do
  for i3:=1 to 4 do
   for i4:=1 to 4 do
    begin
    if (i1 <> i2) and
       (i1 <> i3) and
       (i1 <> i4) and
       (i2 <> i3) and
       (i2 <> i4) and
       (i3 <> i4) then
     Writeln(i1:2,i2:2,i3:2,i4:2);
    end;
END.
```

Эта программка выводит то, что нам нужно:

```
 1 2 3 4
 1 2 4 3
 1 3 2 4
 1 3 4 2
 1 4 2 3
 1 4 3 2
 2 1 3 4
 2 1 4 3
 2 3 1 4
 2 3 4 1
```

```
2 4 1 3
2 4 3 1
3 1 2 4
3 1 4 2
3 2 1 4
3 2 4 1
3 4 1 2
3 4 2 1
4 1 2 3
4 1 3 2
4 2 1 3
4 2 3 1
4 3 1 2
4 3 2 1
```

Но не слишком ли это громоздко? И к тому же данный подход явно не годится для общего случая с N элементами.

Можно было бы поступить, используя рекуррентные соотношения между перестановкой N и $N + 1$ элементов: если у нас есть все перестановки N элементов, то нетрудно получить из них все перестановки $N + 1$ элементов, повторяя каждую перестановку N элементов $N + 1$ раз и помещая $(N + 1)$-й предмет всеми возможными способами. Например, мы имеем все перестановки двух чисел:

```
1 2
2 1
```

Повторяем каждую строку три раза:

```
1 2
1 2
1 2
2 1
2 1
2 1
```

Теперь вставляем число 3 всеми способами с помощью "лесенки":

```
1 2 3
1 3 2
3 1 2
3 2 1
```

```
2 3 1
2 1 3
```

Мы получили все перестановки из трех элементов.

Но это тоже некрасиво, ведь нам надо запоминать в массиве все предыдущие перестановки. Неужели не существует метода прямо так сразу на блюдечке получить результат, не используя массивов? Такие методы есть, и такая задача была в 1980 году на 1-й Московской олимпиаде по программированию для школьников. Просто надо научиться по данной перестановке из N элементов строить "следующую" перестановку. Чтобы внести смысл в понятие "следующая перестановка", их надо как-то упорядочить. Естественно начать с перестановки 1, 2, ..., N, а закончить перестановкой N, (N − 1), ..., 2, 1. Первая перестановка упорядочена в словарном порядке, т. е. числа расположены по возрастанию. Если бы мы имели не числа, а буквы a, b, c, ..., то это было бы более наглядно.

Чтобы в массиве `m` по данной перестановке получить следующую, проделаем такие шаги:

1. Просмотрим числа справа налево, пока не встретим число `m[i]`, меньшее своего правого соседа. Если такого числа нет, то мы имеем конечную перестановку и наша работа закончена.
2. Числа, стоящие правее `m[i]`, образуют убывающую последовательность. Рассмотрим эти числа с правого края и, двигаясь влево, найдем среди них первое число, большее, чем `m[i]`, и поменяем его местами с числом `m[i]`.
3. Затем числа в хвосте перестановки, который идет за числом, занявшим место числа `m[i]`, расположим в порядке возрастания. Для этого надо просто изменить порядок следования этих чисел на обратный. Следующая перестановка получена.

Вот собственно и все программирование для данной задачи. Написание текста программы является более приземленной задачей и называется *кодированием алгоритма*. В листинге 1.6 приведена вся программа с функцией получения следующей перестановки из `n` различных чисел в заданном массиве `m`. Этот алгоритм выводит отсортированные по возрастанию строки с перестановками.

Замечание

В языке Pascal в отличие от языка C массивы могут начинаться с любого индекса и обычно их начинают с индекса 1. Мы в этой книге будем начинать массивы с нуля по практическому соображению: чтобы эти примеры проще было переносить на язык C. По этой же причине перестановки будут иметь диапазон от 0 до N – 1.

Листинг 1.6. Олимпиадный вариант генерации перестановок

```
{$A-,B-,G+,I+,R-,S-}
program Per1;
LABEL EX;
CONST
 { Максимальное число элементов перестановки }
 MAXN=10;
TYPE
 { Тип массива для хранения перестановок }
 tmw=array[0..MAXN-1] of word;
VAR
 i,n: integer;
 m: tmw;
 numall: longint;

{ Получение следующей перестановки n различных чисел в массиве m.
  По алгоритму решения задачи 80.1.2 из сборника московских
  олимпиад по программированию для школьников.
  Если следующая перестановка получена, возвращается TRUE,
  если данная перестановка была последняя, возвращается FALSE. }
function NextPer(var m: tmw; n: word): boolean;
VAR
 i,j,i1,i2,temp: word;

begin
      for i:=n-2 downto 0 do
       { Ищем первое справа число m[i], которое меньше своего правого
         соседа m[i+1] }
       if m[i] < m[i+1] then
        begin
        for j:=n-1 downto i+1 do
         if m[j] > m[i] then
          { Меняем местами m[i] и первое справа число m[j], которое
            больше m[i] }
```

```
          begin
          temp:=m[i];
          m[i]:=m[j];
          m[j]:=temp;
          { Переставляем в обратном порядке хвост перестановки, начиная
            с числа m[i+1] }
          i1:=i+1;
          i2:=n-1;
          while i1 < i2 do
           begin
           temp:=m[i1];
           m[i1]:=m[i2];
           m[i2]:=temp;
           Inc(i1);
           Dec(i2);
           end;
          { Перестановка создана. Выходим и возвращаем TRUE }
          NextPer:=TRUE;
          Exit;
          end;
        end;
      { В массиве m данная перестановка последняя }
      NextPer:=FALSE;
end; {NextPer}

BEGIN
        Write('Введите число элементов перестановки: (1-',MAXN,'): ');
        { Получаем число элементов перестановки }
        Readln(n);
        Writeln;
        if (n < 1) or (n > MAXN) then
         begin
         Writeln('Ошибка ввода.');
         goto EX;
         end;
```

```
        { Обнуляем счетчик числа перестановок }
        numall:=0;
        { Создаем начальную перестановку 0, 1, ..., n-1 }
        for i:=0 to n-1 do m[i]:=i;
        repeat
         { Выводим очередную перестановку }
         for i:=0 to n-1 do Write(m[i]:2);
         Writeln;
         Inc(numall);
        until not NextPer(m,n);
        Writeln('Всего ',numall);
EX:     Writeln('Press Enter to exit...');
        Readln;
END.
```

На компакт-диске вы найдете программу per1.exe (per1.pas), которая спрашивает число элементов и выводит все перестановки для этого числа.

Замечание

Так как вывод может не поместиться на экране, то вы можете перенаправлять его в файл. Для этого в конце командной строки надо добавить имя этого файла со знаком "больше" перед ним. Например: `per1 > out.txt`. Если файл out.txt существует, то он будет уничтожен и воссоздан заново. Если вы хотите дописать к файлу, то используйте формат `per1 >> out.txt`. Аналогично можно перенаправлять ввод программы с клавиатуры к файлу, используя знак <.

Еще немного по поводу вывода на экран: если в программу включить модуль `Crt` строкой

```
USES Crt;
```

то вывод на экран будет происходить намного быстрее, т. к. процедура `Write` (`Writeln`) будет записывать текст напрямую в видеопамять вместо вывода через функцию BIOS. В случае перенаправления вывода в файл мы не увидим запроса на ввод числа элементов перестановки, и нам надо будет ввести это число и нажать клавишу <Enter>, а потом еще раз <Enter>, чтобы завершить программу. Я пробовал менять значение у переменной `directvideo` модуля `Crt` (`FALSE` для вывода через BIOS и `TRUE` для вывода в видеопамять), но это не помогает решить данную проблему. Turbo Pascal как-будто не реагирует на присвоение `directvideo` значения `FALSE`, и перенаправление вывода в файл не происходит. Впрочем, я избегаю использова-

ния в примерах к данной книге модуля `Crt` по соображениям, высказанным в начале этой главы.

А сейчас мы рассмотрим другой вариант процедуры вывода всех перестановок ввиду того, что он обладает замечательными особенностями. Этот вариант был найден в 1958 г., а рассматриваемый далее алгоритм и программу составил я. Сначала ознакомимся с понятием *беспорядка*, но не того, при котором на лоток привода CD-ROM ставят чашку с кофе, а с математическим беспорядком.

Последовательность чисел 1, 2, 3, 4 с математической точки зрения обладает *полным порядком*, т. к. все ее элементы упорядочены по возрастанию. Если мы переставим местами числа 2 и 1, то в этой последовательности возникнет один беспорядок, т. к. одно большее число стоит левее меньшего. А если переставить местами числа 1 и 4: 4, 2, 3, 1, как здесь подсчитать число беспорядков? В общем случае надо для каждого числа подсчитать, сколько чисел, больших него, стоят левее, чем оно. Для двойки это одно число, для тройки тоже одно, а для единицы целых три. Итого, в этой последовательности имеется пять беспорядков.

Подсчет числа беспорядков зависит от того, что мы определяем как порядок. Мы могли бы считать порядком размещение чисел от больших к меньшим и для каждого числа суммировать число чисел, меньших его, которые стоят левее его. Вот, собственно, и все о теории беспорядков.

Если присмотреться к только что рассмотренному алгоритму получения очередной перестановки, то мы увидим, что в нем неявно присутствовало понятие беспорядка: на шаге 2 мы для числа с `i`-го места увеличиваем на минимум число беспорядков, обменивая `m[i]` с более правым числом, большим его, и на шаге 3 приводим хвост последовательности в порядок. Это аналогично тому, как наращивается на единицу счетчик с 0999 до 1000: старший разряд увеличивается на единицу, а все разряды, младшие него, сбрасываются в ноль.

Вернемся к уже промелькнувшей идее "лесенки" и будем с ее помощью получать перестановки рекуррентно, пока нас не осенит какая-нибудь гениальная идея насчет получения очередной перестановки по предыдущей. Начнем с перестановки из одного числа, это просто:

```
0
```

Теперь получим из нее все перестановки из двух чисел. Для этого выпишем перестановку 2 раза, оставляя слева место для числа 1:

```
  0
  0
```

и впишем в каждую строку число 1, двигая его лесенкой справа налево:

```
  0 1
1 0
```

Мы получили все перестановки из двух чисел. Теперь аналогично выпишем их, повторив каждую строку 3 раза, оставляя между числами место для числа 2:

```
   0   1
   0   1
   0   1
   1   0
   1   0
   1   0
```

Применяя нашу любимую "лесенку", получаем все перестановки из трех чисел:

```
   0   1 2
   0 2 1
2  0   1
2  1   0
   1 2 0
   1   0 2
```

Вы еще не заметили общего правила? Тогда повторим этот прием для получения перестановок из четырех чисел, выписав каждую перестановку из трех чисел четыре раза:

```
   0   1   2 3
   0   1 3 2
   0 3 1   2
3  0   1   2
3  0   2   1
   0 3 2   1
   0   2 3 1
   0   2   1 3
   2   0   1 3
   2   0 3 1
   2 3 0   1
3  2   0   1
3  2   1   0
   2 3 1   0
   2   1 3 0
   2   1   0 3
   1   2   0 3
```

```
   1   2 3 0
   1 3 2   0
3  1   2   0
3  1   0   2
   1 3 0   2
   1   0 3 2
   1   0   2 3
```

Мы видим следующее:

1. Чем больше число, тем чаще оно двигается.
2. Каждое число кроме единицы двигается по лесенке справа налево и обратно *среди чисел, меньших его*. При этом для чисел, меньших N (здесь у нас $N = 3$), каждая строка повторяется в зависимости от величины этого числа.
3. Когда число N доходит до левого или правого края, в следующей строке оно остается там же, т. к. в этой строке двигается какое-то число, меньшее его. Это с уточнением из предыдущего пункта верно и для чисел, меньших N.

Для нахождения алгоритма получения следующей перестановки по данной надо ответить на вопрос: какие соседние числа на данном шаге следует обменять местами, т. к. мы видим, что у нас постоянно происходит обмен какой-то пары соседних чисел.

А что если в каждой строке вывести числа беспорядков для всей последовательности, а также для всех более младших последовательностей? В листинге 1.7 за перестановками вы в каждой строке видите числа беспорядков для всей перестановки и для этой перестановки, из которой последовательно вычеркнуты числа 3 и 2.

Листинг 1.7. Перестановки с беспорядками

```
0 1 2 3   0  0  0
0 1 3 2   1  0  0
0 3 1 2   2  0  0
3 0 1 2   3  0  0
3 0 2 1   4  1  0
0 3 2 1   3  1  0
0 2 3 1   2  1  0
0 2 1 3   1  1  0
2 0 1 3   2  2  0
```

```
2 0 3 1   3  2  0
2 3 0 1   4  2  0
3 2 0 1   5  2  0
3 2 1 0   6  3  1
2 3 1 0   5  3  1
2 1 3 0   4  3  1
2 1 0 3   3  3  1
1 2 0 3   2  2  1
1 2 3 0   3  2  1
1 3 2 0   4  2  1
3 1 2 0   5  2  1
3 1 0 2   4  1  1
1 3 0 2   3  1  1
1 0 3 2   2  1  1
1 0 2 3   1  1  1
```

На самом деле нам первый столбец беспорядков не нужен, он приведен здесь для полноты.

Из листинга 1.7 видно, что если в какой-то перестановке число беспорядков без максимального числа 3 (это число беспорядков стоит во втором столбце беспорядков) четно и максимальное число не самое левое, то в следующей перестановке максимальное число меняется местами со своим левым соседом. Если это число беспорядков нечетно и максимальное число не самое правое, то оно меняется местами со своим правым соседом.

Остается разобрать случаи, когда тройка является самым левым элементом в перестановке и число беспорядков без нее четно, и когда тройка является самым правым элементом в перестановке и число беспорядков без нее нечетно. В этих случаях обменивается местами со своим соседом какое-то меньшее число.

Но ведь это аналогично тому, что мы рассмотрели двумя абзацами выше: в этих случаях мы вычеркиваем тройку и рассматриваем эту перестановку без нее. Старшее число теперь у нас двойка. Если число беспорядков без двойки (это число стоит в третьем столбце беспорядков) четно и двойка не самая левая, то двойка меняется местами со своим левым соседом. Если число беспорядков без двойки нечетно и двойка не самая правая, то она меняется местами со своим правым соседом.

Для остальных случаев надо вычеркнуть двойку и рассуждать аналогично. Теперь максимальное число — единица (см. 12-ю строку предыдущего листинга), она стоит справа, и число беспорядков без нее нулевое, т. е. четное.

И тут верно открытое нами правило: мы меняем единицу с ее левым соседом нулем.

Вот, в общем, и весь алгоритм получения следующей перестановки, составить который нам помогло понятие числа беспорядков. Опишем теперь этот алгоритм более формально.

Как и в случае с программой per1.pas, у нас будет та же функция `NextPer`, которая будет на входе получать массив `m` как `var`-параметр, и число `n` — количество элементов в перестановке. Если `NextPer` создаст следующую перестановку, то она возвратит `TRUE` (и конечно, новую перестановку в массиве `m`), если на входе окажется последняя возможная перестановка, то функция ничего не сделает и возвратит `FALSE`.

Вычеркивать числа мы не будем, а просто заведем две переменные `i1` и `i2` — индексы начального и конечного числа в текущей перестановке, которую мы рассматриваем. Вначале `i1` присвоим `0`, а `i2` — `n-1`. Когда нам надо будет вычеркнуть очередное максимальное число, мы увеличим `i1` или уменьшим `i2` на единицу, ведь число для вычеркивания у нас всегда стоит с краю.

Вот сам формальный алгоритм.

***

Задаем начальные границы текущей рассматриваемой части перестановки:

```
i1:=0;
i2:=n-1;
```

Задаем максимальный элемент в рассматриваемой перестановке: `max:=n-1`.

1. Начало основного цикла. Если `i1 >= i2`, то переходим за цикл к пункту 3.
2. Рассматривая часть перестановки с индекса `i1` до индекса `i2` включительно, подсчитаем в `numbesp` число беспорядков для чисел, меньших `max`.

   Найдем индекс `indmax` числа `max` в этой же части перестановки от индекса `i1` до `i2`.

   Если число беспорядков `numbesp` нечетно, то обмениваем число `max` с его правым соседом и выходим, возвратив `TRUE`.

   Переходим к перестановке без `max`: уменьшаем `max` и `i2` на единицу и переходим к пункту (метке) 2.

   Если `indmax` не равен `i1`, то обмениваем число `max` с его левым соседом и выходим, возвращая `TRUE`.

   Переходим к перестановке без `max`, уменьшив `max` на единицу, увеличив `i1` на единицу и совершив переход к пункту (метке) 1.

   Конец основного цикла.
3. Выходим, возвращая `FALSE`.

***

Вы наверно заметили, что этому алгоритму получения следующей перестановки в отличие от предыдущего требуется, чтобы диапазон переставляемых чисел был строго от `0` до `n-1`. Для работы с другими диапазонами чисел нужно исправлять алгоритм и текст функции `NextPer`. В таком случае условие основного цикла можно было записать так: вместо `i1 >= i2` написать `max = 0`.

Вдумчивый читатель может заметить, что в алгоритме мы один раз переходим на метку 1, где проверяем выполнение условия цикла, а другой раз переходим к пункту 2, где условие `i1 >= i2` не проверяется. Не возникнет ли из-за этого ошибка? Можно, конечно, оба раза переходить к метке 1, но ошибка здесь не возникает, потому что мы приходим к равенству границ `i1` и `i2`, только увеличивая `i1`. Этот момент соответствует последней строке (у нас в примере это `1 0 2 3`). Здесь мы увеличиваем `i1` на единицу и получаем `i1` и `i2` равными одному. Текущая перестановка здесь состоит из одного нуля.

В листинге 1.8 вы видите полный текст программы с моим алгоритмом получения следующей перестановки.

Листинг 1.8. Программа с авторским алгоритмом получения перестановок

```
{$A-,B-,G+,I+,R-,S-}
program Per2;
LABEL EX;
CONST
  MAXN=10;
TYPE
 tmw=array[0..MAXN-1] of word;
VAR
 i,n: word;
 m: tmw;
 numall: longint;

{ Процедура выдачи в массиве m следующей перестановки чисел
  0, 1, 2, ..., n-1 по предыдущей. Начиная с перестановки
  0, 1, ..., n-1, можно получить все n! перестановок, причем каждая
  следующая будет отличаться от предыдущей ровно одной транспозицией
  соседних элементов.
}
```

```
function NextPer(var m: tmw; n: word): boolean;
LABEL 1,2;
VAR
  i,j,i1,i2,max,numbesp,indmax: word;

begin
      { Зададим начальные границы рассматриваемой перестановки }
      i1:=0;
      i2:=n-1;
      { Зададим максимальный элемент в рассматриваемой перестановке }
      max:=n-1;
      while i1 < i2 do
       begin
       { Подсчитаем число беспорядков для чисел, меньших max }
1:     numbesp:=0;
       for i:=i1 to i2-1 do
        begin
        if m[i] < max then
          for j:=i+1 to i2 do
           if m[j] < m[i] then Inc(numbesp);
        end;
       { Найдем индекс indmax числа max в перестановке m }
       for i:=i1 to i2 do
        if m[i] = max then
          begin
          indmax:=i;
          goto 2;
          end;
2:     if Odd(numbesp) then
        begin
        if indmax <> i2 then
          begin
          m[indmax]:=m[indmax+1];
          m[indmax+1]:=max;
          NextPer:=TRUE;
          Exit;
          end;
```

```
        { Переходим к  перестановке без max }
        Dec(max);
        Dec(i2);
        goto 1;
        end;
       if indmax <> i1 then
        begin
        m[indmax]:=m[indmax-1];
        m[indmax-1]:=max;
        NextPer:=TRUE;
        Exit;
        end;
       { Переходим к перестановке без max }
       Dec(max);
       Inc(i1);
      end;
      NextPer:=FALSE;
end; {NextPer}

BEGIN
      Write('Введите число элементов перестановки: (1-',MAXN,'): ');
      { Получаем число элементов перестановки }
      Readln(n);
      Writeln;
      if (n < 1) or (n > MAXN) then
       begin
       Writeln('Ошибка ввода.');
       goto EX;
       end;
      for i:=0 to n-1 do m[i]:=i;
      numall:=0;
      repeat
       for i:=0 to n-1 do Write(m[i]:2);
       Writeln;
       Inc(numall);
```

```
      until not NextPer(m,n);
      Writeln('Всего ',numall);
EX:   Writeln('Press Enter to exit...');
      Readln;
END.
```

Взглянув на функцию `NextPer`, читатель может сказать: а не красивее было бы цикл

```
for i:=i1 to i2 do
 if m[i] = max then
  begin
   indmax:=i;
   goto 2;
  end;
```

записать вот так:

```
for indmax:=i1 to i2 do
 if m[indmax] = max then goto 2;
```

Да, красивее, но этот код при попытке перенести его в другой язык или компилятор может преподнести сюрприз: ведь счетчики циклов некоторые оптимизирующие компиляторы держат в регистрах процессора, поэтому при выходе из цикла значение этого счетчика не определено, т. к. регистры имеют свойство быстро затираться новыми данными. В Turbo Pascal с этим делом все в порядке: оптимизации кода у него нет.

И напоследок — два замечательных свойства этого алгоритма генерации перестановок. Во-первых, мы попутно конструктивно доказали интересную теорему о том, что все перестановки из N элементов можно разместить по кругу, и при этом каждая перестановка будет отличаться от своих соседей ровно одной транспозицией соседних элементов. Действительно, взгляните на листинг 1.7: если в последней перестановке поменять местами два первых числа, то мы получим первую перестановку. И во-вторых: строки 13—24 получаются, соответственно, из строк 1—12 путем их реверса, т. е. рассмотрения от конца к началу. Получается, что нам достаточно сгенерировать только первую половину из $N!$ перестановок, а вторая половина получается из первой путем реверса строк!

Позже мы получим еще один способ генерации перестановок как побочный результат при составлении другого алгоритма.

1.3.2. Размещения

А теперь, как говорится, ударим интеллектом по размещениям и сочетаниям M предметов по N местам. Пусть мы имеем N различных (например, перенумерованных) урн и M различных (тоже перенумерованных) шаров, $1 \le M \le N$. Сколькими способами можно разместить эти M шаров по N урнам? Первый шар можно разместить N способами, положив его в любую урну. Для второго шара уже будет иметься $N - 1$ способов, для третьего — $N - 2$ способа и т. д., а для последнего, M-го шара останется $N - M + 1$ урн или способов. Количество размещений обозначается буквой A от французского слова *arrangement* (размещение, приведение в порядок). Число мест записывают в виде нижнего индекса, а количество шаров — в виде верхнего. Формулу размещений можно записать более компактно в виде

$$A_n^m = \frac{n!}{(n-m)!}.$$

Читается "а из эм по эн". При $M = N$ размещения вырождаются в перестановки:

$$A_n^n = P_n = n!.$$

Мы не будем использовать размещения в наших примерах, поэтому сразу перейдем к сочетаниям из M элементов по N местам. Заинтересованный читатель сможет написать функцию, выводящую все размещения, используя функцию для вывода всех сочетаний, которую мы напишем в следующем разделе, и уже готовую функцию вывода всех перестановок.

1.3.3. Сочетания и их вывод

Это проще генерации перестановок, потому что сочетаемые предметы неотличимы друг от друга. Мы, как программисты, заведем массив байтов, в котором в пустые ячейки будем заносить нули, а в занятые — единицы. При выводе на экран пустые места будем для удобства обозначать точками, а занятые — звездочками.

Итак, пусть имеется N различных урн и M одинаковых шаров, $1 \le M \le N$. Сколькими способами можно положить эти M шаров в N урн? Для ответа на этот вопрос заметим, что каждое размещение M шаров по N урнам может быть получено из такого же сочетания путем всех перестановок его элементов, т. е. шаров. Каждое сочетание порождает $M!$ размещений, поэтому сочетаний будет меньше в $M!$ раз. В математике сочетания обозначают буквой C от французского слова *combinaison* (сочетание, комбинация).

А формула числа сочетаний из M элементов по N выглядит так:

$$C_n^m = \frac{n!}{m!(n-m)!}.$$

Для поиска алгоритма вывода всех сочетаний примем $N = 5$ и $M = 3$. Выпишем все десять сочетаний вручную, двигая звездочки слева направо:

```
***..
**.*.
**..*
*.**.
*.*.*
*..**
.***.
.**.*
.*.**
..***
```

Мы видим, что чем правее звездочка, тем чаще она перемещается. Это аналогично тому, как в счетчике младшие разряды меняются чаще старших. Когда младшие звездочки уперлись вправо и не могут больше двигаться, начинает движение следующая по старшинству звездочка, и после ее шага вправо все более младшие звездочки сдвигаются к ней в самые левые позиции.

Принцип ясен: идем справа налево до первого пустого места, непосредственно перед которым стоит звездочка. Если мы не находим такого места, то это у нас последнее сочетание. Иначе двигаем звездочку на шаг вправо, и все звездочки, что стоят правее ее (если они есть), сдвигаем до упора влево. В программе мы при движении по массиву вправо одновременно будем считать встретившиеся единицы, чтобы потом легче было восстанавливать хвост массива. Нам незачем сдвигать единицы в хвосте влево, мы просто выведем их после сдвинутой единицы в количестве, которое было найдено при движении вправо, а за этими единицами запишем нули. На компакт-диске, прилагаемом к книге, вы найдете программу comb.pas (comb.exe), которая выводит все сочетания, запрашивая перед этим N и M. В листинге 1.9 приведен ее текст.

Листинг 1.9. Программа вывода всех сочетаний

```
{$A-,B-,G+,I+,R-,S-}
program Comb;
LABEL EX;
```

```
CONST
 MAXN=10;
TYPE
 tmb= array[0..MAXN-1] of byte;
VAR
 i,n,k: word;
 m: tmb;
 numall: longint;

{  Выдача следующего сочетания в массиве m[0], ..., m[n-1].
   Если оно найдено, то возвращается TRUE, если заданное сочетание
   было последним, массив m не меняется и возвращается FALSE.
   Пустые места в массиве m содержат 0, занятые места содержат 1 }
function NextComb(var m: tmb; n: word): boolean;
LABEL 10;
VAR i,j,ct1: word;

begin
       { Ищем от конца m пару m[i] = 0 и m[i-1] > 0,
         считая попутно единицы }
        ct1:=0;
        for i:=n-1 downto 1 do
          begin
            if m[i] = 1 then
            begin
              Inc(ct1);
              goto 10;
            end;
            if m[i-1] = 1 then
            { Продвигаем m[i-1] на шаг вправо }
            begin
              m[i]:=1;
              m[i-1]:=0;
              { Переписать правые элементы? }
              if ct1 > 0 then
```

```
              begin
                Inc(i);
                for j:=i to i+ct1-1 do m[j]:=1;
                  for j:=i+ct1 to n-1 do m[j]:=0;
              end;
              NextComb:=TRUE;
              Exit;
            end;
10:       end;
        { Последнее сочетание }
        NextComb:=FALSE;
end;  {NextComb}

BEGIN
      Write('Введите через пробел число мест и элементов сочетания: '+
          '(1-',MAXN,') (1-',MAXN,'):');
      { Получаем параметры сочетания }
      Readln(n,k);
      Writeln;
      if (n < 1) or (n > MAXN) or (k < 1) or (k > n) then
        begin
          Writeln('Ошибка ввода.');
          goto EX;
        end;
      { Очищаем массив m и создаем начальное сочетание }
      for i:=0 to n-1 do m[i]:=0;
        for i:=0 to k-1 do m[i]:=1;
      numall:=0;
      repeat
        for i:=0 to n-1 do
          if (m[i] = 0) then Write('.')
          else Write('*');
        Writeln;
        Inc(numall);
      until not NextComb(m,n);
      Writeln('Всего ',numall);
```

```
EX:   Writeln('Press Enter to exit...');
      Readln;
END.
```

1.4. Моделируем один удивительный парадокс

Теперь немного отвлечемся от комбинаторики, чтобы поговорить об одном удивительном нетранзитивном парадоксе, и промоделируем его в программе. Вспомним, что такое транзитивность: это такое свойство, которое транзитом переходит через второй элемент к третьему. Например, рассмотрим множество целых чисел и рассмотрим свойство "быть эквивалентным". Пусть $a = b$ и $b = c$. Тогда $a = c$. Еще со школы мы знаем, что эквивалентность транзитивна. Аналогично транзитивны свойства "быть меньше", "быть больше", "быть меньше или равно", "быть больше или равно". А как насчет реальных вещей и человеческих отношений? Свойство "быть родным братом" транзитивно: брат моего брата — мой брат. С другими родственными связями не так все гладко, например, сын сына — уже не сын, а внук.

Оказывается (и мы убедимся в этом из следующего парадокса), что понятие транзитивности заложено в человеке на уровне интуиции. Но в сложных случаях происходит сбой: человек какое-то отношение считает транзитивным, хотя реально это не так. В этом случае возникают удивительные парадоксы, в которые не могут поверить даже математики, пока сами в этом не убедятся. Этот вид парадоксов был открыт сравнительно недавно. За первый из открытых нетранзитивных парадоксов, точнее, за доказательство теоремы "о невозможности идеальной избирательной системы" (и не только за это) в 1972 г. была присуждена Нобелевская премия по экономике. Мы же рассмотрим нетранзитивное пари, которое было обнаружено математиком У. Пенни в 1969 г. Предположим, что мы равновероятным образом выбираем ноль или единицу (например, бросаем монету). При трех выборах получается 8 равновероятных комбинаций:

```
000
001
010
011
100
101
110
111
```

Пусть игрок *A* выбирает из этих комбинаций любую и делает на нее ставку. Здесь важно то, что выбор игрока *A* известен игроку *B* и игрок *B* делает свою ставку после игрока *A*. Затем бросается монета и строится случайная последовательность бросаний, состоящая из 0 и 1. Тот игрок, чья тройка встретится в этой последовательности раньше, выигрывает. Например, пусть *A* поставил на тройку 001, *B* — на тройку 101, и выпала последовательность 10001. После пятого бросания игра заканчивается победой *A*, т. к. его тройка имеется в выпавшей последовательности.

На первый взгляд, появление данной тройки первой имеет одну и ту же вероятность для всех троек, но это не так. Рассмотрим тройки 000 и 100. Если в случайной последовательности нулей и единиц тройка 000 встретилась раньше тройки 100, то перед тройкой 000 не может стоять 1, иначе 100 встретилась бы раньше. Но это может быть лишь тогда, когда случайная последовательность начинается с 000, а это один шанс из восьми. Мы видим, что тройка 100 выигрывает у тройки 000 с вероятностью 7/8! Но и это еще не все: оказывается, какую бы тройку ни выбрал игрок *A*, игрок *B* может выбрать тройку, которая выигрывает у тройки игрока *A* более, чем в 50% случаев! Вот она, нетранзитивность: не существует "самой сильной" тройки исходов, для любой находится более сильная, которая у нее выигрывает! Даже математически подготовленный человек, который не сталкивался с этим, в это не поверит. А если от троек исходов перейти к четверкам, то вероятность выигрыша становится еще выше!

Для проверки этого парадокса мы напишем программу и научимся для каждой тройки выбирать лучшую тройку. Программа будет выдавать таблицу 8×8, сверху будут идти все тройки игрока *A*, а слева — те же тройки, выбранные игроком *B*. В самой таблице на пересечении `i`-ой строки и `j`-го столбца будет записана вероятность, с которой тройка игрока *B* выигрывает у тройки, выбранной игроком *A*.

Последовательность бросаний будем моделировать битами в переменной `t`. Для нас достаточно хранить результат последних трех бросаний, это будут три младших бита, самый младший отвечает за последнее бросание. Затем в программе идет цикл по `i` по строкам таблицы, и в нем цикл по `j` по ее столбцам. В цикле по `j` мы будем проводить 20 000 игр и накапливать выигрыши игроков в переменных `i1` и `j1`. Такое большое число выбрано, чтобы вероятность исхода игры была точнее. После этой серии игр программа подсчитает вероятность выигрыша 1-го игрока у 2-го и выведет его в `i`-ой строке и `j`-ом столбце таблицы.

В листинге 1.10 представлен текст этой программы, которая также присутствует на компакт-диске, приложенном к книге.

Листинг 1.10. Моделирования результата парадоксальной игры

```
{$G+,E+,S-,B-}
program Goat;
LABEL  1,2;
VAR
  i,j,k,i1,j1,t : word;

{ Возврат в строке 3-х младших битов числа t }
function ToBits(t: word): string;
VAR
 s: string;
 i: word;

begin
      s:='';
      for i:=2 downto 0 do s:=s+char(t shr i and 1+ord('0'));
      ToBits:=s;
end; {ToBits}

BEGIN
      Writeln;
      Write('   ');
      for i:=0 to 7 do Write(ToBits(i):7);
      Writeln;
      Randomize;
      { Создаем случайную битовую последовательность xxx }
      t:=Random(8);
      Writeln;
      { Цикл по строкам таблицы }
      for i:=0 to 7 do
       begin
       Write(ToBits(i)+' ');
       { Цикл по столбцам таблицы }
       for j:=0 to 7 do
```

```
        begin
        { На главной диагонали таблицы (при i = j) выводим * }
        if i <> j then
         begin
         { Число побед 1-го игрока }
         i1:=0;
         { Число побед 2-го игрока }
         j1:=0;
         { Число бросков монеты }
         k:=1;
         repeat
          { Последовательность в t совпала со ставкой 1-го игрока? }
1:        if t = i then
           begin
           { Да, 1-й игрок выиграл. Увеличим число его выигрышей }
           Inc (i1);
           { Бросаем 3 раза монету для начала
             следующей игры }
           t:=Random(8);
           { Увеличиваем число игр для данного выбора
             (ставки) i,j игроков }
           Inc (k);
           goto 2;
           end;
          { Последовательность в t совпала со ставкой 2-го игрока? }
          if t = j then
           begin
           { Да, 2-й игрок выиграл. Увеличим число его выигрышей }
           Inc (j1);
           { Бросаем 3 раза монету для начала следующей игры }
           t:=Random (8);
           { Увеличиваем число игр для данного выбора
             (ставки) i,j игроков }
           Inc (k);
           goto 2;
           end;
```

```
          { При этом бросании монеты никто не выиграл.
            Бросаем монету еще раз. }
          t:=t shl 1 and 7+Random(2);
          goto 1;
2:       until k > 20000;
         { Выводим вероятность выигрыша 1-го игрока у 2-го
           по результатам 20 000 игр. }
         Write (i1/(j1+i1):7:3);
         end
          else Write ('    *  ');
        end;
       Writeln;
       end;
     Writeln;
     Writeln('Press Enter to exit...');
     Readln;
END.
```

А в листинге 1.11 вы видите результат работы этой программы.

Листинг 1.11. Таблица выбора наилучшего хода

```
       000    001    010    011    100    101    110    111

000     *    0.501  0.401  0.400  0.123  0.412  0.302  0.501
001   0.497    *    0.664  0.668  0.251  0.626  0.500  0.703
010   0.599  0.336    *    0.501  0.498  0.502  0.378  0.582
011   0.600  0.331  0.502    *    0.496  0.496  0.750  0.875
100   0.873  0.750  0.503  0.501    *    0.501  0.330  0.600
101   0.581  0.378  0.500  0.506  0.497    *    0.331  0.596
110   0.698  0.504  0.627  0.250  0.661  0.669    *    0.495
111   0.500  0.300  0.421  0.127  0.403  0.403  0.500    *
```

Теперь о методе выигрыша. После того как 1-й игрок выберет свою комбинацию, мы в ее столбце находим максимальное число и в его строке читаем

комбинацию, которая выигрывает против нее с вероятностью, не меньшей двух третей.

1. Против 000 выигрывает 100.
2. Против 001 выигрывает 100.
3. Против 010 выигрывает 001.
4. Против 011 выигрывает 001.
5. Против 100 выигрывает 110.
6. Против 101 выигрывает 110.
7. Против 110 выигрывает 011.
8. Против 111 выигрывает 011.

Эту таблицу можно было бы составить с помощью логических рассуждений, подсчитывая вероятности, и получить точные результаты в виде натуральных дробей, но это сложнее моделирования игры на компьютере.

На компакт-диске вы также найдете мой научно-фантастический рассказик "Прыжок через козла", который был опубликован в журнале "Наука и жизнь".[2] В нем рассказывается, как двое студентов остроумно и психологически тонко использовали этот парадокс для получения зачета по физкультуре.

Предприимчивый читатель, вероятно, уже представил, как можно организовать малый бизнес на этой игре и потом настрочить свою *success story* (историю успеха) под заголовком "Как я заработал свой первый миллион". Ну что же, в добрый путь, а мы продолжаем наше занимательное программирование.

1.5. Словесные игры и рекорды

Словесные игры продолжают оставаться популярными в Интернете, и это касается также англоязычной его части. Как тут не пожалеть некоторые азиатские народы, которые пользуются иероглифическим письмом? Ведь у них в принципе не может быть таких развлечений, как кроссворды. Но изобретательные японцы компенсировали этот недостаток своего языка такими известными головоломками, как японское рисование и судоку. Вообще в Японии головоломки популярны как нигде в мире, и, возможно, в этом один из секретов таких крупных научно-технических достижений этой страны. А что же нам с нашим относительно неплохим алфавитом и языком мешает проявить свою изобретательность и получить интересные, а может быть, и

[2] Мельников С. В. Прыжок через козла // Наука и жизнь. — 1997. — № 5.

выдающиеся результаты? Я это говорю еще и потому, что читатель с этой книгой получает на компакт-диске проверенный словарь существительных русского языка (больше 42 000 слов!). Скажем за это спасибо любителям головоломок, делавшим набор (их имена приведены в файле с лицензией (capter1\Словесные игры и рекорды\Словарь существительных\License.txt).

1.5.1. Анаграммы

Анаграммами называют слова, состоящие из одних и тех же букв, но записанных в другом порядке. Например: АВАНС — САВАН, АДРЕС — СРЕДА, АНГАР — НАГАР, ШТУКА — ШУТКА. Есть и более длинные ряды, например: АРАБ — АРБА — РАБА. Задания найти анаграммы к данному слову или словосочетанию популярны в газетах и журналах для досуга. Раньше энтузиасты находили эти слова вручную, а мы напишем программку и вытянем все анаграммы из имеющегося у нас словаря существительных. После этого и узнаем, какой ряд анаграмм самый длинный, и какие слова содержат максимум букв.

Для наших целей я записал слова из словаря существительных в файлы 2.txt, 3.txt, ..., 29.txt, распределив их по длине слова, отсортировав по возрастанию кодов букв (буква Е — последняя), выкинув слова с дефисом и преобразовав строчные буквы в прописные. Кроме того, в этих файлах слова идут подряд, переводы строк отсутствуют. Эти 29 файлов и будут служить исходной информацией для наших программ. Самый большой файл 10.txt имеет размер 44 060 байтов, т. е. мы можем держать его целиком в памяти. Вы найдете исходные тексты программ, которые я для этого использовал, в файлах makelex.pas и sortlex.pas.

Чтобы вывести все анаграммы, применим полный перебор: внешний цикл будет по этим файлам, а внутри него будет цикл по всем словам в текущем файле. В слова, которые уже использовались как анаграммы, будем на место первой буквы записывать ноль, чтобы находить только длиннейшие цепочки анаграмм без повторов их частей.

Для чтения очередного файла со словами выделим динамическую память в массиве символов с указателем `p`. По размеру файла и длине слова узнаем число слов, которые он содержит, и запомним его в переменной `numw`. Для поиска цепочки анаграмм нам надо завести два индекса в массиве `p^`: индекс слова, для которого мы ищем анаграммы (`indj`), и индекс `indk` слова для внутреннего цикла, которое мы будем сравнивать с текущим словом из внешнего цикла. Сами эти слова скопируем, соответственно, в массивы `wordj` и `wordk`. Внешний цикл по `j` будет от `0` до `numw-2`, т. е. от самого первого слова в массиве `p^` до предпоследнего слова. Внутренний цикл по `k` будет начинаться от следующего слова, которое идет вслед за словом `wordj`, и до последнего слова в `p^`. Эти переменные циклов будут использоваться

только как счетчики, а следить за позициями текущих слов будут их индексы `indj` и `indk`, которые мы будем наращивать на длину слова и для перехода к следующему слову.

Записывая цикл сравнения слова `wordj` со словом `wordk` и выяснения, все ли их буквы совпадают, надо быть внимательным: во-первых, если какая-то буква из слова `wordk` совпала с буквой слова `wordj`, то эту букву следует вычеркнуть из `wordk`. Во-вторых, при этом совпадении надо не только не забыть увеличить счетчик совпавших букв, но и прекратить сравнение для текущей буквы слова `wordj` и перейти к поиску совпадений для следующей его буквы. В листинге 1.12 дана эта программа.

Листинг 1.12. Программа выдачи всех анаграмм

```
{$A-,B-,G+,I+,R-,S-}
{$M 16384,0,655360}
program Anagr;
LABEL 1,2,10,EX;
CONST
 MIN=2;
 MAX=29;
TYPE
 tmc=array[0..1] of char;
 tpmc=^tmc;
 tw=array[0..MAX-1] of char;
VAR
 i,fsize,numw,j,k,indj,indk,l1,l2,ct: word;
 iores: integer;
 f: file;
 fname: string;
 p: tpmc;
 wordj,wordk,wordk1: tw;
 notout: boolean;

BEGIN
      Writeln;
      { Работаем с файлами в режиме только чтения }
      System.filemode:=0;
```

```
      { Цикл по файлам 2.txt, 3.txt, ..., 29.txt }
      for i:=MIN to MAX do
       begin
       { Получаем в fname имя следующего файла }
       Str(i,fname);
       fname:=fname+'.txt';
       Assign(f,fname);
{$I-}
       { Открываем файл на чтение }
       Reset(f,1);
{$I+}
       iores:=IOResult;
       if iores <> 0 then
        begin
        { Если ошибка 2, то такого файла нет }
        if iores = 2 then goto 10;
        Writeln('Ошибка открытия файла ',iores);
        goto EX;
        end;
       fsize:=FileSize(f);
       { Выделяем память под файл и читаем его }
       GetMem(p,fsize);
       if p = NIL then
        begin
        Close(f);
        Writeln('Ошибка при выделении памяти.');
        goto EX;
        end;
       BlockRead(f,p^,fsize);
       Close(f);
       { Подсчитаем число слов в p^ }
       numw:=fsize div i;
       { Вывод анаграмм из массива слов в p^. Длина слова i }
       { Индекс очередного слова wordj }
       indj:=0;
       for j:=0 to numw-2 do
```

```
        begin
        { Следующее слово уже использовано? }
        if p^[indj] = #0 then goto 2;
        { Для слова wordj не было вывода }
        notout:=TRUE;
        { Запомним слово в wordj }
        Move(p^[indj],wordj,i);
        { Индекс слова wordk для сравнения со словом wordj }
        indk:=(j+1)*i;
        { Цикл поиска анаграмм для слова wordj }
        for k:=j+1 to numw-1 do
         begin
         if p^[indk] > #0 then
          begin
          { Запомним слово в wordk и wordk1 }
          Move(p^[indk],wordk,i);
          wordk1:=wordk;
          { Проверяем, все ли буквы слова wordj содержатся
            среди букв слова wordk }
          { Счетчик совпавших букв }
          ct:=0;
          for l1:=0 to i-1 do
           begin
           for l2:=0 to i-1 do
            if wordj[l1] = wordk[l2] then
             begin
             { Буква wordj[l1] совпала с буквой wordk[l2] }
             wordk[l2]:=#0;
             Inc(ct);
             goto 1;
             end;
1:         end;
          if ct = i then
           { Нашли анаграмму }
           begin
           { Вычеркиваем слово wordk из массива p^ }
```

```
            p^[indk]:=#0;
            if notout then
             { Выводим начальное слово цепочки wordj }
             begin
             for l1:=0 to i-1 do Write(wordj[l1]);
             notout:=FALSE;
             end;
            { Выводим слово wordk1 (wordk мы затерли) }
            Write('-');
            for l1:=0 to i-1 do Write(wordk1[l1]);
            end;
          end;
         { Переходим к проверке следующего слова для слова wordj }
         Inc(indk,i);
         end;
        { Если был вывод цепочки для слова wordj, переводим строку }
        if not notout then Writeln;
        { Переходим к следующему слову wordj }
2:      Inc(indj,i);
        end;
       { Освобождаем память для файла fname }
       FreeMem(p,fsize);
10:    end;
EX:    Writeln('Press Enter to exit...');
       Readln;
END.
```

Взгляните на результат работы программы в файле anagr.txt. Программа нашла 1091 цепочку анаграмм. Рекордная цепочка состоит из шести слов: АВТОР — ВТОРА — ОТВАР — РВОТА — ТАВРО — ТОВАР. Вот эти цепочки тоже очень интересны: ВОСПРЕЩЕНИЕ — ВСЕПРОЩЕНИЕ — ПРОСВЕЩЕНИЕ, БАРЫНЯ — РАБЫНЯ, а анаграмма для самого длинного слова вызывает улыбку: ВОДОГРЯЗЕЛЕЧЕБНИЦА — ГРЯЗЕВОДОЛЕЧЕБНИЦА. Жаль, в словаре не оказалось слова ДИЛЕР, получилась бы цепочка ДИЛЕР — ЛИДЕР. Попробуйте вручную находить анаграммы к словам. Если число букв в слове больше пяти, то это, как правило, непростая задача!

1.5.2. Квадраты слов

Напишем друг под другом пять слов, отобранных по правилу кроссворда, например:

ДОБРО
ОМЛЕТ
БЛАНК
РЕНТА
ОТКАЗ

На первый взгляд, в них нет ничего особенного. Но попробуйте теперь читать не слева направо, а сверху вниз. Ага, сверху вниз читаются те же пять слов! Об этих словесных наборах говорят, что они образуют *квадрат*. В русском языке таких квадратов тысячи, но найти их без помощи компьютера, ох, как нелегко. Пока я не взял в руки клавиатуру, были известны единицы таких наборов, найденных вручную. Когда же за дело принялся компьютер, они посыпались как из рога изобилия. В вышеприведенном квадрате содержится пять различных слов. Но есть квадраты, в которых все десять слов различны. Например:

КОКОН
АТАКА
НАБОБ
ОРАВА
НАКАТ

Как правило, эти слова содержат часто встречающиеся буквы, такие как А, которая является чуть ли не каждой десятой в текстах на русском языке.

А как насчет большей длины слова? Есть и шестибуквенные рекорды, но их намного меньше. Без помощи компьютера найти их, по-моему, никому не удавалось. Квадратов из слов большей длины в русском языке, видимо, не существует.

Мне иногда жалко изобретателей игр и головоломок, не знающих программирования. Они напоминают сборщиков грибов и ягод, которые ходят с лукошком по лесу. И тут подъезжает программист на тракторе, снимает растительность с верхним слоем земли, просеивает его и выгребает из него все. После этого в лесу уже ничего не растет. Но к счастью, не все игры и головоломки можно расколоть при помощи компьютера. Попробуйте, к примеру, создать программу, находящую *фразы-палиндромы*, т. е. такие фразы, которые одинаково читаются слева направо и справа налево:

Морда казаха за кадром.

Закопан, а попа напоказ.

Лом о смокинги гни, комсомол!

```
Утречко летело к черту.
Лилипут сома на мосту пилил.
Мыло — голым!
Я, Светка, в акте вся!
Летя, дремал как ламер дятел.
```

Красиво, но для программирования вряд ли доступно. А квадраты слов не представляют для нас принципиальной трудности. Здесь мы применяем лобовую атаку с полным перебором. Но эту программу надо будет оптимизировать из-за очень большого и длительного перебора. Сначала набросаем общий алгоритм программы на примере поиска квадратов из пяти слов.

1. Читаем файл 5.txt в массив символов `p`. Длину файла запоминаем в переменной `fsize`. Подсчитываем число слов в файле: `numw:=fsize div 5`. Для хранения квадрата из пяти слов заводим массив `square` из пяти массивов по пять символов в каждом: `array[0..4,0..4] of char`.
2. Далее идут пять вложенных циклов от `0` до `numw-1` по числу слов в `p`. В каждом из них мы записываем очередное слово из `p` в следующий подмассив массива `square`. Во внешнем цикле — в `square[0]`, в следующем цикле — в `square[1]` и т. д. И после каждой такой записи, начиная со второй, проверяем для всех пяти столбцов, есть ли в `p` слова, начинающиеся буквами из пяти столбцов массива `square`. Иначе программа будет делать огромный лишний перебор вариантов. Если внутри какого-либо цикла после записи в `square` очередного слова выяснится, что в каком-то столбце `square` буквы не образуют начала никакого слова из `p`, то мы продолжим этот цикл со следующего слова, иначе перейдем к следующему внутреннему циклу. В самом внутреннем цикле при записи в `square` пятого слова эта же проверка покажет, получили ли мы квадрат. Если да, то выводим его и продолжаем самый внутренний цикл со следующего слова, иначе вывод квадрата пропускаем.

В листинге 1.13 вы видите текст этой нехитрой программы. На компакт-диске эта программа имеет имя dummy5.pas.

Листинг 1.13. Прямолинейная программа выдачи всех квадратов 5 × 5

```
{$A-,B-,G+,I+,R-,S-}
{$M 16384,0,655360}
program Dummy5;
LABEL 2,3,4,5,EX;
CONST
 FMAX=50000;
 MAX=5;
```

```
TYPE
 tmc=array[0..FMAX-1] of char;
 tsquare=array[0..MAX-1,0..MAX-1] of char;
VAR
 fsize,numw,i0,i1,i2,i3,i4,j: word;
 f: file;
 fname: string;
 square: tsquare;
 p: tmc;
 iores: integer;

{ Проверка существования пяти слов в p для пяти столбцов в square.
  Число букв в столбце равно n. При успехе возвращаем TRUE }
function CompN(n: word): boolean;
LABEL 10,20;
VAR
 column: array[0..MAX-1] of char;
 i,j,k: word;

begin
      CompN:=FALSE;
      { Цикл по числу столбцов в square }
      for i:=0 to MAX-1 do
       begin
       { Переписываем j-й столбец из square в массив column }
       for j:=0 to n-1 do column[j]:=square[j,i];
       { Цикл по числу слов в массиве p }
       for j:=0 to numw-1 do
        begin
        for k:=0 to n-1 do
         begin
         if column[k] < p[j*MAX+k] then
          { Дальше проверять бесполезно, т. к. слова отсортированы
            по возрастанию буквенных кодов }
          Exit;
         if column[k] > p[j*MAX+k] then
```

```
          { Продолжаем проверку со следующего слова }
          goto 10;
         end;
        { Для j-го столбца square слово в p найдено. Переходим к проверке
          следующего столбца }
        goto 20;
10:     end;
       { Перебрали все слова, а подходящего для j-го столбца не нашли }
       Exit;
20:    end;
      { Нашли все пять слов }
      CompN:=TRUE;
end; {CompN}


BEGIN
      { Работаем с файлами в режиме только чтения }
      System.filemode:=0;
      fname:='5.txt';
      Assign(f,fname);
{$I-}
       { Открываем файл на чтение }
       Reset(f,1);
{$I+}
       iores:=IOResult;
       if iores <> 0 then
        begin
        Writeln('Ошибка открытия файла ',iores);
        goto EX;
        end;
       fsize:=FileSize(f);
       { Проверяем размер файла }
       if fsize > FMAX then
        begin
        Close(f);
```

```
        Writeln('Слишком большой файл: ',fname);
        goto EX;
        end;
       { Читаем файл }
       BlockRead(f,p,fsize);
       Close(f);
       { Подсчитаем число слов в p }
       numw:=fsize div 5;
       for i0:=0 to numw-1 do
        begin
        Move(p[MAX*i0],square[0],MAX);
        for i1:=0 to numw-1 do
         begin
         Move(p[MAX*i1],square[1],MAX);
         if not CompN(2) then goto 2;
         for i2:=0 to numw-1 do
          begin
          Move(p[MAX*i2],square[2],MAX);
          if not CompN(3) then goto 3;
          for i3:=0 to numw-1 do
           begin
           Move(p[MAX*i3],square[3],MAX);
           if not CompN(4) then goto 4;
           for i4:=0 to numw-1 do
            begin
            Move(p[MAX*i4],square[4],MAX);
            if not CompN(5) then goto 5;
            { Выводим квадрат }
            Writeln;
            for j:=0 to MAX-1 do
             Writeln(square[j]);
5:          end;
4:         end;
3:        end;
2:       end;
        end;
```

```
EX:    Writeln('Press Enter to exit...');
       Readln;
END.
```

Как видите, вычисления в этом примере не оптимизированы, многих умножений можно было избежать. Но здесь эта оптимизация только затруднила бы понимание алгоритма. Для такого большого перебора (пять циклов в цикле, и каждый перебирает 3256 слов — это очень много итераций, несмотря на отсечение ненужных вариантов.) Вместо этого мы напишем оптимизированную программу, в которой длина слова будет параметром. И таким образом эта программа будет искать квадраты любого заданного ей размера.

Прежде чем составлять алгоритм, подумаем о возможных оптимизациях скорости работы нашей будущей программы.

1. Кандидаты на первое слово в квадрате не могут содержать буквы Ы, Ъ, Ь, т. к. с этих букв не могут начинаться слова. В этом случае внешний цикл перебора слов переходит к следующему слову в словаре (массиве `p`). Далее пойдут менее очевидные оптимизации.
2. Заводим массив `offs2` на 33×33 элемента типа `word` и записываем в него смещения первых слов в `p` для всех пар букв АА, АБ, АВ, ..., АЯ, АЕ, БА, ББ, ..., БЯ, БЕ, ..., ЕА, ..., ЕЯ, ЕЕ, с которых эти слова начинаются. Если для какой-либо пары букв нет слова, которое начинается с этой пары, то в соответствующем элементе массива `offs2` записываем число `65535`, у нас это будет как бы бесконечность, поэтому в разделе констант определим константу `INFINITE=65535`. Например, в файле 5.txt первое слово, начинающееся на АБ, — это АБАКА, это слово — первое в словаре. Поэтому `offs[1]` будет равно 0. (`offs[0]` присваиваем `INFINITE`, поскольку слова на АА в словаре нет.) Первое слово на АВ — это АВАЛЬ, оно расположено по смещению `45` в файле и массиве `p`, поэтому `offs[2]` у нас будет равно `45`, и т. д. Этот массив здорово поможет нам в сокращении перебора при выяснении наличия слова, начинающегося с заданной пары букв. Во-первых, мы будем знать, есть ли такое слово (если его нет, то соответствующий элемент массива `offs2` равен `INFINITE`), а если такое слово есть, то мы начнем перебор этих слов не с начала словаря, а с первого такого слова. Для этого по паре букв надо уметь определять индекс элемента массива `offs2`, содержащего смещение в массиве `p` первого слова, начинающегося с данной пары букв. Создаем вспомогательный массив

   ```
   ALPH: array[0..32] of char='АБВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯЕ'
   ```

   Формула будет такой: индекс в массиве `ALPH` первого символа пары умножаем на `33` и прибавляем к нему индекс в `ALPH` второго символа пары. Например, для пары АВ получаем $0 \times 33 + 2 = 2$, значит, `offs2[2]` со-

держит смещение в массиве `p` первого слова, начинающегося на АБ, а если такого слова нет, то там будет число `INFINITE`. В принципе, можно было бы завести аналогичный массив `offs3` для каждой тройки букв, но он имел бы размер больше 64 Кбайт, что не подходит для 16-разрядной программы.

3. Объявляем массив `is2: array[0..33*33-1] of boolean`. Он аналогичен массиву `offs2`. Каждый элемент массива `is2` будет говорить нам о том, есть ли в каком-то слове массива `p` соответствующее сочетание букв, но в отличие от массива `offs2` это сочетание букв может находиться в любом месте слова, а не только в начале. Например, если ни в одном слове нет пары ЕЕ, то последний элемент `is2` будет `FALSE`, иначе `TRUE`. А для сочетания ББ имеется слово АББАТ, поэтому `is2[1*33+1]` будет содержать `TRUE`. (Слова из пяти букв мы рассматриваем только как пример, программа будет универсальной.) А теперь пример того, как этот логический массив будет ускорять перебор слов: предположим, первое слово в квадрате у нас — АБАКА, а второе — БОБОК, и мы ищем кандидата на третье слово. Цикл стартует с начала словаря, мы будем опять рассматривать слова АБАКА, АББАТ, ..., КОРЬЕ, ... Но прежде чем проверять в словаре наличие всех пяти слов, начинающихся с трех букв из столбцов, мы проверим по массиву `is2`, встречаются ли в словах пять пар букв. Пусть мы в переборе кандидатов на третье слово дошли до слова КОРЬЕ. Теперь мы по массиву `is2` проверяем наличие слов, содержащих пять пар: АК, БО, БР, АЬ и ТЕ. Эти пары букв образуют слова АББАТ и КОРЬЕ. Массив `is2` скажет нам, что сочетания АЬ ни в одном слове нет, поэтому мы отбросим слово КОРЬЕ и сразу перейдем к проверке следующего слова.

4. Создаем аналогичный массив `is3: array[0..33*33*33-1] of boolean`. Он будет отвечать за всевозможные сочетания из трех букв, которые встречаются *в начале слова*. К примеру, тройке БАГ будет соответствовать элемент с индексом $1 \times 33 \times 33 + 0 \times 33 + 3$. Если какое-то слово начинается с БАГ, то на этом месте будет стоять `TRUE`, иначе `FALSE`. Этот массив уже не вместится в сегмент данных, поэтому выделим ему динамическую память, хотя с указателями работа идет медленнее, чем напрямую с переменными.

5. Чтобы избежать лишних умножений, перебор слов в массиве `p` выполним по индексу, который будет равен смещению первой буквы слова в `p` и будет увеличиваться на длину слова. Проверять конец массива слов в `p` будем по наличию нулевого байта на месте букв. С этой целью в конец `p` допишем `i` нулевых байтов, где `i` — длина слова.

Не много ли оптимизаций? В конце окажется, что нет. Нахождение всех квадратов слов из шести букв будет занимать несколько часов.

А теперь — описание данных:

- константа `MAX=29` означает наибольшую возможную длину слова;
- массив символов `p` будет содержать все слова, которые будут идти подряд. Слова должны быть отсортированы по возрастанию кодов символов (слова, начинающиеся на Е, будут в самом конце);
- массив `square` типа `array[0..MAX-1] of word` содержит смещения в `p` для первых букв слов, являющихся кандидатами в квадрат слов. К примеру, если `p[0]` равно `0`, то это означает, что первое слово в квадрате — АБАКА. Индексом в массиве `square` будет `j`. В цикле по `j` мы будем подбирать `j`-е слово для будущего квадрата.

... И алгоритм работы программы:

1. Получаем от пользователя минимальную и максимальную длину слов в файле и присваиваем это переменным `min1` и `max1`.
2. Цикл по `i` от `min1` до `max1` по файлам словарей. Если файл с именем <*i*>.txt отсутствует, то переходим на повтор цикла по `i`.
3. Читаем файл <*i*>.txt в массив `p`. Дописываем в конец `p` `i` нулевых байтов. Заполняем массивы `is2`, `offs2` и `is3`, переменной `j` присваиваем `0`.
4. **Метка `20`.** `square[j]` присваиваем `0`. Это означает, что на роль `j`-го слова в квадрате пока претендует слово из `p`, имеющее смещение `0`. (Смещение слова — это смещение его первой буквы.)
5. **Метка `30`.** Если `j` равно `0`, то проверяем, есть ли в слове со смещением `j` буквы Ы, Ъ, Ь. Если есть, то переходим к метке `60` для продвижения по словарю слова `square[j]`, иначе увеличиваем `j` на `1` и переходим к метке `20`.
6. Если `j` равно `1`, то по массиву `offs2` проверяем, что в словаре есть все `i` слов, которые начинаются с соответствующих букв слов из `square[0]` и `square[1]`. К примеру, если `square[0]` указывает на слово АБАКА и `square[1]` указывает на это же слово (так будет в начале цикла по `j`), то проверяем наличие в `p` слов, начинающихся с пар АА, ББ, АА, КК и АА. Если для какой-либо пары мы получим значение `INFINITE`, то переходим к метке `60`. Если все пять таких слов имеются, то обходим перебор и сразу переходим к метке `50` для увеличения `j` и переходу к поиску следующего слова для квадрата.
7. Если `j` равно `2`, то по массиву `is3` проверяем наличие в `p` `i` таких слов, которые начинаются с соответствующих сочетаний из трех букв слов `square[0]`, `square[1]` и `square[2]`. Здесь также в случае неудачи переходим к метке `60`, а в случае удачи — к метке `50`.

8. Если j больше 2, то теперь будем проверять наличие в p i слов, которые имеют сочетания букв из двух последних слов квадрата square[j-1] и square[j]. Если такое слово не найдено для пары букв, с которых начинаются слова со смещениями square[j-1] и square[j], будем продолжать перебирать кандидатов для последнего слова square[j], переходя на метку 60.
9. Далее проверяем, что в массиве p существуют слова, начинающиеся с первых букв слов, на которые указывают square[0], square[1], ..., square[j], со вторых букв этих слов, и т. д., с i-1 букв этих слов. В случае неудачи переходим на метку 60 для продолжения перебора последнего слова square[j]. Если все i проверок окончились успешно, то в массиве square мы имеем смещения для j-1 таких слов, что по всем i столбцам нашего кандидата в квадраты читаются начала каких-то слов из p. И тут мы выходим на строку с меткой 50.
10. **Метка 50.** Если j+1 меньше i, то увеличиваем j на 1 и переходим к метке 20 для начала подбора следующего слова квадрата. Иначе программа нашла квадрат слов! Выводим этот квадрат и выходим на строку с меткой 60.
11. **Метка 60.** Увеличиваем square[j] на i и, таким образом, переходим на следующее слово в массиве p для j-ой строки квадрата.
12. **Метка 70.** Если p[square[j]] равно #0, то мы вышли за последнее слово в словаре p. Если кроме того, j равно 0, то весь перебор и работа программы закончены. Иначе мы увеличиваем square[j] на i и переходим на метку 70. Если в процессе выполнения этого маленького цикла мы получим, что p[square[j]] не равно #0, то переходим к метке 30 для проверки, подходит ли слово со смещением square[j] для будущего квадрата...

Как видим, за универсальность и ускорение программы надо платить усложнением ее алгоритма. Особенно это касается общей проверки, подходит ли данное слово для j-ой строки квадрата. В листинге 1.14 представлен полный текст данной программы.

Листинг 1.14. Универсальная программа выдачи всех квадратов слов *n* × *n*

```
{$A-,B-,G+,I+,R-,S-}
{$M 16384,0,655360}
program WordSq;
LABEL 1,2,20,30,40,50,60,70,1000,EX;
CONST
 MIN=2;
```

```
 MAX=29;
 FMAX=50000;
 INFINITE=65535;
 ALPH: array[0..32] of char='АБВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯЕ';
TYPE
 tmc=array[0..FMAX-1] of char;
 tsquare=array[0..MAX-1] of word;
 tis3=array[0..33*33*33-1] of boolean;
VAR
 i,fsize,j,j1,j2,k,k1,k2,k3,k1i,min1,max1: word;
 iores: integer;
 f: file;
 fname: string;
 square: tsquare;
 column: array[0..MAX-1] of char;
 offs2: array[0..33*33-1] of word;
 is2: array[0..33*33-1] of boolean;
 is3: ^tis3;
 p: tmc;
 char0: char;

BEGIN
      GetMem(is3,33*33*33);
      if is3 = NIL then
       begin
       Writeln('Не хватает памяти.');
       goto EX;
       end;
      { Работаем с файлами в режиме только чтения }
      System.filemode:=0;
      { Получаем диапазон длин слов min1-max1 для
        поиска в файлах }
      min1:=0;
      max1:=0;
      Writeln('Имеются слова с длинами от ',MIN,' до ',MAX,' букв.');
      Writeln('Введите через пробел min и max длины слов');
```

```
      Write('для вывода всех квадратов из этих слов: ');
      Readln(min1,max1);
      if (min1 < MIN) or (max1 > MAX) or (min1 > max1) then
       begin
       Writeln('Ошибка во вводе.');
       goto EX;
       end;

      { Это может быть цикл по всем файлам 2.txt, 3.txt, ..., 29.txt }
      for i:=min1 to max1 do
       begin
       { Получаем в fname имя следующего файла }
       Str(i,fname);
       fname:=fname+'.txt';
       Assign(f,fname);
{$I-}
       { Открываем файл на чтение }
       Reset(f,1);
{$I+}
       iores:=IOResult;
       if iores <> 0 then
        begin
        { Если ошибка 2, то такого файла нет }
        if iores = 2 then goto 1000;
        Writeln('Ошибка открытия файла ',iores);
        goto EX;
        end;
       fsize:=FileSize(f);

       { Проверяем размер файла }
       if fsize+i > FMAX then
        begin
        Close(f);
        Writeln('Слишком большой файл: ',fname);
        goto EX;
        end;
```

```
{ Читаем файл }
BlockRead(f,p,fsize);
Close(f);
{ Дописываем нули }
FillChar(p[fsize],i,#0);

{ Заполняем массив is2: если есть в словаре пары
  АА, АБ, ..., АЕ, БА, ББ, ..., ЕЕ, то соответствующий
  элемент массива is2 будет TRUE, иначе FALSE }
for j:=0 to 33*33-1 do is2[j]:=FALSE;
k1i:=0;
repeat
 { Обходим границы слов }
 if k1i mod i < i-1 then
  begin
  if p[k1i] < 'E' then k:=(word(p[k1i])-word('A'))*33 else
   k:=32*33;
  if p[k1i+1] < 'E' then Inc(k,word(p[k1i+1])-word('A')) else
   Inc(k,32);
  is2[k]:=TRUE;
  end;
 Inc(k1i);
until p[k1i] = #0;

{ Записываем в offs2 смещения в p для первых слов,
  начинающихся на АА, АБ, ..., АЯ, АЕ, ..., ЕЕ.
  Если для какой-то пары букв нет начинающегося с
  них слова, то соответствующее смещение равно INFINITE }
for j:=0 to 32 do
 for j1:=0 to 32 do
  begin
  k:=0;
  repeat
   if (p[k] = ALPH[j]) and (p[k+1] = ALPH[j1]) then
    begin
    offs2[j*33+j1]:=k;
```

```
           goto 1;
           end;
          Inc(k,i);
         until (p[k] > ALPH[j]) or (p[k] = #0);
         offs2[j*33+j1]:=INFINITE;
1:       end;

       { Записываем в is3 существование в p для первых слов,
         начинающихся на AAA, AAБ, ..., AAЯ, AAE, ..., EEE.
         Если для какой-то тройки букв нет начинающегося с
         них слова, то соответствующее значение равно FALSE }
       for j:=0 to 32 do
        for j1:=0 to 32 do
         for j2:=0 to 32 do
          begin
          k:=0;
          repeat
           if (p[k] = ALPH[j]) and (p[k+1] = ALPH[j1])
            and (p[k+2] = ALPH[j2])then
            begin
            is3^[(j*33+j1)*33+j2]:=TRUE;
            goto 2;
            end;
           Inc(k,i);
          until (p[k] > ALPH[j]) or (p[k] = #0);
          is3^[(j*33+j1)*33+j2]:=FALSE;
2:        end;

       j:=0;
20:    square[j]:=0;
30:    if j = 0 then
        begin
        for k:=0 to i-1 do
         if p[square[0]+k] in ['Ы','Ъ','Ь'] then goto 60;
        Inc(j);
```

```
 goto 20;
 end;
if j = 1 then
 begin
 for j1:=0 to i-1 do
  begin
  char0:=p[square[0]+j1];
  if char0 < 'E' then k1i:=(word(char0)-word('A'))*33 else
   k1i:=32*33;
  char0:=p[square[1]+j1];
  if char0 < 'E' then Inc(k1i,word(char0)-word('A')) else
   Inc(k1i,32);
  k1i:=offs2[k1i];
  if k1i = INFINITE then goto 60;
  end;
 { Нашли все i таких слов }
 goto 50;
 end;

if j = 2 then
 { Проверяем наличие в p i слов, начинающихся на первые 3 буквы
   из столбцов square }
 begin
 for j1:=0 to i-1 do
  begin
  char0:=p[square[0]+j1];
  if char0 < 'E' then k1i:=(word(char0)-word('A'))*33*33 else
   k1i:=32*33*33;
  char0:=p[square[1]+j1];
  if char0 < 'E' then Inc(k1i,(word(char0)-word('A'))*33) else
   Inc(k1i,32*33);
  char0:=p[square[2]+j1];
  if char0 < 'E' then Inc(k1i,word(char0)-word('A')) else
   Inc(k1i,32);
  if not is3^[k1i] then goto 60;
  end;
```

```
        { Нашли все i таких слов }
        goto 50;
        end;

       if j > 2 then
        { Проверяем наличие в p i слов, которые имеют сочетания
          букв из двух последних слов квадрата k1 и k2. Если надо,
          продолжаем перебирать кандидатов для последнего слова
          square[j] }
        begin
        k1:=square[j-1];
40:     k2:=square[j];
        for j1:=0 to i-1 do
         begin
         char0:=p[k1+j1];
         if char0 < 'E' then k:=(word(char0)-word('A'))*33 else
          k:=32*33;
         char0:=p[k2+j1];
         if char0 < 'E' then Inc(k,word(char0)-word('A')) else
          Inc(k,32);
         if not is2[k] then
          begin
          if j1 = 0 then
           { Нет пары для первых букв слов p[square[j-1]] и
             p[square[j]]. Для ускорения работы тут же выводим square[j]
             на первое слово с другой начальной буквой }
           begin
           repeat
            Inc(square[j],i);
            { Вышли за последнее слово в p? }
            if p[square[j]] = #0 then goto 70;
           until p[square[j]] > p[k2];
           { Нашли для square[j] слово, что первые буквы из слов
             square[j-1] и square[j] образуют пару, встречающуюся
             в каком-то слове }
           goto 40;
           end;
```

```
    { Отвергаем слово square[j], т. к. не нашли какой-то пары букв
      по массиву is2 }
    goto 60;
    end;
   end;
  end;

{ Находим смещение в p для первых двух букв вертикального слова }
for j1:=0 to i-1 do
 begin
 { Запоминаем начало j1-го вертикального слова в column }
 for j2:=0 to j do column[j2]:=p[square[j2]+j1];
 { Выходим на 1-е слово в p, которое начинается на
   column[0]column[1] }
 if column[0] < 'E' then k1i:=(word(column[0])-word('A'))*33 else
  k1i:=32*33;
 if column[1] < 'E' then Inc(k1i,word(column[1])-word('A')) else
  Inc(k1i,32);
 k1i:=offs2[k1i];

 { Выходим на 2-ю (считая с 0) букву в p, которая равна
   column[2]. Слово с этой буквой обязательно
   существует ввиду значения TRUE для
   этой начальной тройки букв в массиве is3 }
 while (column[2] <> p[k1i+2]) do Inc(k1i,i);

 { Проверяем остальные буквы в column,
   начиная с 3-й (считая с 0) }
 for k2:=3 to j do
  begin
  while column[k2] > p[k1i+k2] do
   begin
   Inc(k1i,i);
   for k3:=0 to k2-1 do
    if column[k3] <> p[k1i+k3] then goto 60;
   end;
```

```
          if column[k2] < p[k1i+k2] then goto 60;
          end;
         { Найдено такое слово в p }
         end;
        { Найдены все i таких слов в p }

50:     if j+1 < i then
         begin
         Inc(j);
         goto 20;
         end;
        { Квадрат найден! }
        Writeln;
        for k1:=0 to i-1 do
         begin
         for k2:=0 to i-1 do Write(p[square[k1]+k2]);
         Writeln;
         end;
60:     Inc(square[j],i);
70:     while p[square[j]] = #0 do
         begin
         if j = 0 then goto 1000;
         Dec(j);
         Inc(square[j],i);
         end;
        goto 30;

1000:   end;
EX:     Writeln('Press Enter to exit...');
        Readln;
END.
```

Домашнее задание

1. Дополните эту программу, чтобы она выводила квадраты не просто один под другим, а более экономно используя ширину страницы. И для каждого квадрата отпечатайте число различных слов, которые он содержит.

2. Попробуйте найти такие квадраты слов 5 × 5, внутренний квадрат 3 × 3 которых также является квадратом слов.
3. Напишите программу, выполняющую вывод по заданному ей слову всех слов, которые можно сложить из его букв. Каждая буква может использоваться столько раз, сколько раз она встречается в исходном слове. Найдите слово, порождающее наибольшее количество слов. Эта игра называется "Наборщик". Например, задано слово ОКНО. Получаем слова КОН, ОКО, ...

1.6. Перебор с возвратами

Перебор с возвратами — сердцевина программ для искусственного интеллекта (ИИ). Пытаемся ли мы поставить на шахматную доску $n \times n$ n ферзей, никакая пара из которых не угрожает друг другу, решаем ли мы шахматную задачу, ищем ли выигрывающий ход в логической игре, ищем выход из лабиринта, обходим ли граф, чтобы побывать в каждой его вершине ровно по одному разу, — всюду мы делаем перебор с возвратами. В предыдущей рассмотренной программе мы его уже делали. При таком переборе, если мы попадаем в тупик (например, не можем поставить на шахматную доску i-го ферзя, чтобы он никому не угрожал), то делаем шаг назад, перемещая $(i - 1)$-го ферзя на следующую его позицию, и начинаем перебор позиций для i-го ферзя с начала. Перечисленные выше задачи математики называют *NP-полными*, для них не найдены формулы, позволяющие решить их без большого перебора вариантов. Великий немецкий математик Карл Фридрих Гаусс пытался получить такую формулу, которая выдавала бы все 12 различных позиций для размещения восьми ферзей на шахматной доске, но тщетно. Пришлось делать перебор вручную, т. к. компьютера Гаусс не имел.

Алгоритмы перебора с возвратами могут быть нерекурсивными и рекурсивными. Мы рассмотрим оба варианта таких алгоритмов. В качестве первого примера решим задачу поиска в графе *гамильтоновых циклов*. Для нас в этой книге *граф* — это просто несколько точек, некоторые из них соединены отрезками. Мы не будем рассматривать несвязные графы и графы с петлями (вершина соединена с собой), заметим только, что отрезок может иметь направление движения, т. е. он проходит только в одну сторону. Такой отрезок рисуется со стрелкой на конце и называется *ориентированным ребром* или *дугой*. Неориентированный отрезок называется просто *ребром*. Для нашего алгоритма не важно, имеет ли граф дуги или нет, граф может быть любым — неориентированным, ориентированным и смешанным. Вершина B называется смежной для вершины A, если из A в B можно попасть за один ход. Обратите внимание, что в случае дуги A уже не будет смежной верши-

ной для B, если, конечно, A и B не связывают другие отрезки. Случаи кратных дуг и ребер мы тоже не рассматриваем.

1.6.1. Поиск гамильтоновых циклов

Если мы можем обойти все вершины графа, побывав в каждой из них ровно по разу, и вернуться в исходную вершину, то мы прошли по гамильтонову циклу этого графа. Задача состоит в том, чтобы найти все такие циклы или доказать, что их нет.

В процессе поиска таких циклов возникают сугубо практические задачи: например, надо минимизировать какую-либо величину. Представьте, что вы хотите посетить N городов и вернуться обратно в свой город, при этом вы хотите минимизировать пройденное расстояние или цену билетов. Эта задача в математике называется *задачей о бродячем торговце* или *задачей коммивояжера*. В 50-е годы прошлого века группа американских математиков и программистов решила задачу облета столиц всех 50-ти штатов США наиболее коротким путем. Для компьютеров той поры это была серьезная задача, а сейчас ее можно решить на своем домашнем супер (по тем временам) компьютере.

Для решения задачи поиска гамильтоновых циклов надо договориться, в какой форме программа будет представлять в памяти этот граф. Во-первых, все вершины занумеруем от 0 до $N-1$. Далее создадим матрицу $N \times N$, в i-ой строке которой будем ставить нолики в столбцах, с номерами вершин, не связанных отрезками с i-ой вершиной, и единички в тех столбцах, которые непосредственно связаны с i-ой вершиной. Такую матрицу называют *матрицей смежности вершин графа*. Если мы имеем не орграф, то его матрица смежности вершин симметрична относительно главной диагонали. Если граф имеет петлю, то на главной диагонали стоит единица, иначе вся главная диагональ состоит из одних нулей. Главная диагональ проходит от левого верха к правому низу.

Теперь опишем смысл и назначение данных будущей программы, которую назовем hamilton.pas.

Так как мы заранее не определяем число вершин графа, то введем константу `MAXN` для максимального числа этих вершин. Ее значение может быть любым, но не меньше трех. Величина этой константы определяет размер используемых массивов.

Сама матрица смежности вершин графа представляется двумерным массивом `graph: array[0..MAXN-1,0..MAXN-1] of byte`. `i`-я строка матрицы будет представлять `i`-ю вершину графа: если в этой строке на `j`-ом месте стоит `1`, то это означает, что из `i`-ой вершины можно попасть в `j`-ю вершину за один шаг. В противном случае `graph[i,j]` будет содержать `0`.

Перебирая для `i`-ой вершины связанные с ней вершины, мы должны запоминать номер той `j`-ой вершины, на которой мы остановились, чтобы при возврате назад в случае неудачи в поиске цикла программа продолжила перебор в `i`-ой строке со следующей, (`j+1`)-ой вершины, иначе программа зациклится. Для этой цели нам послужит массив `graphind: array[0..MAXN-1] of integer`. Обходя граф, надо запоминать номера пройденных вершин, которые могут войти в цикл. Эти номера будем собирать в массиве `circuit: array[0..MAXN-1] of byte`. В переменной `circuitind` будем запоминать индекс последней пройденной вершины: `circuit[circuitind]` — это ее номер.

Переменная `firstvert` (от англ. *first vertex* — первая вершина) будет означать номер вершины, с которой мы начинаем обход графа. Казалось бы, зачем нужна эта переменная? Начнем с вершины `0`, и все дела. Так как в гамильтонов цикл входят все вершины, то достаточно сделать обход графа, начав с какой-то одной вершины. Это верно, а смысл `firstvert` я поясню позже.

Переменная `numstr` будет хранить номер текущей строки в матрице смежности вершин, по которой мы будем двигаться; счетчик найденных циклов будем хранить в переменной `numcircuits`. Остальные переменные являются вспомогательными для ввода данных, их назначение ясно из текста программы. В частности, `n` — это действительное число вершин графа, это число задает пользователь.

Переходим к описанию алгоритма программы.

1. Спрашиваем у пользователя число `n` вершин в графе и вводим в массив `graph` матрицу смежности вершин, задаваемую пользователем. Делаем присваивания `firstvert:=0` и `numcircuits:=0`.
2. Делаем присваивания `graphind[firstvert]:=-1`, `circuitind:=0`, `numstr:=firstvert` и `circuit[0]:=numstr`. Последнее присваивание означает, что мы в своем пути по графу уже пришли в вершину номер `numstr`. `-1` мы присваиваем потому, что будем увеличивать `graphind[numstr]` на `1` перед началом цикла поиска в строке `numstr` очередной единицы.
3. **Метка `10`.** Вся программа заключена в этот цикл. Наращиваем `graphind[numstr]` на `1`.
4. Цикл поиска следующей единицы в строке `numstr`. Начиная от столбца `graphind[numstr]` и до столбца `n-1`, ищем единицу в строке `graph[numstr]`. Если в `i`-ом столбце мы нашли единицу, то запоминаем номер этого столбца (это также номер вершины, связанной с вершиной номер `numstr`): `graphind[numstr]:=i`, проверяем, была ли уже пройдена вершина `i` раньше (для этого у нас есть массив `circuit` и индекс `circuitind` последней пройденной вершины). Если `i` содержится в мас-

сиве `circuitind`, то проверяем условие `(i = firstvert) and (circuitind = n-1)`, которое означает, что мы нашли цикл. Если мы его нашли, то выводим все вершины этого цикла из массива `circuit`, наращиваем на `1 numcircuits` и отступаем на две вершины назад: `circuitind:=n-3; numstr:=circuit[circuitind]`, после чего переходим на метку `10` для продолжения поиска единицы в строке `numstr`. Если цикл мы не нашли, то просто переходим на метку `10`.

5. Если вершина `i` не была еще пройдена, то заносим `i` в массив `circuit`: `Inc(circuitind), circuit[circuitind]:=i` и переходим в строку `i` для перебора вершин, связанных с вершиной `i`: `numstr:=i, graphind[numstr]:=-1, goto 10`.
6. Наконец, если мы в цикле из п. 4 в строке `numstr` не нашли больше единиц, то, если `circuitind` равен `0`, мы обошли весь граф, а если нет, то переходим в предыдущую вершину графа (возвращаемся на шаг назад) для поиска следующей вершины, связанной с вершиной номер `numstr`: `Dec(circuitind), numstr:=circuit[circuitind], goto 10`.

А теперь вопрос: что будет, если на главной диагонали будут стоять единицы? Не попадет ли наша программа в петлю? Нет, ничего страшного не произойдет: в п. 4 мы найдем, что эта петлевая вершина была уже пройдена, и не пойдем в нее второй раз.

В листинге 1.15 представлен полный текст программы выдачи всех гамильтоновых циклов.

Листинг 1.15. Программа выдачи всех гамильтоновых циклов

```
{ Программа поиска гамильтоновых циклов в орграфе }
{$A-,B-,G+,I+,R-,S-}
{$M 16384,0,655360}
{ Вывод всех гамильтоновых циклов в графе }
program Hamilton;
label
 1,2,10,EX;
const
 maxn=30;
var
 { Матрица смежности вершин графа }
 graph: array[0..maxn-1,0..maxn-1] of byte;
 { Индексы в graph, от которых идет поиск следующей вершины }
 graphind: array[0..maxn-1] of integer;
```

```
 { Массив для запоминания найденных вершин }
 circuit: array[0..maxn-1] of byte;
 firstvert,circuitind,numstr,i,j,n,nf: word;
 { Счетчик найденных циклов }
 numcircuits: longint;
 str: string[maxn];

BEGIN
      Writeln('Программа поиска гамильтоновых циклов в орграфе.');
      repeat
       Writeln('Введите число вершин в графе (3 < n <= ',maxn,')');
       Readln(n);
       if (n < 3) or (n > maxn) then Writeln('Ошибка ввода.');
      until (n > 2) and (n <= maxn);
      { Для форматирования вывода }
      nf:=(n-1) div 10+2;
      Writeln('Введите матрицу смежности вершин графа.');
      Writeln('Это ',n,' строк, целиком состоящих из 0 и 1.');
      for i:=0 to n-1 do
       begin
1:     str:='';
       Readln(str);
       if Length(str) <> n then goto 2;
       for j:=0 to n-1 do
        begin
        if (str[j+1] <> '0') and (str[j+1] <> '1') then
         begin
2:       Writeln('Ошибка. Повторите ввод предыд. строки.');
         goto 1;
         end;
        graph[i,j]:=byte(str[j+1])-byte('0');
        end;
       end;

      firstvert:=0;
      numcircuits:=0;
```

```
      graphind[firstvert]:=-1;
      circuitind:=0;
      numstr:=firstvert;
      circuit[0]:=numstr;
      repeat
       { Выходим на следующую вершину в текущей строке graph }
10:    Inc(graphind[numstr]);
       { Ищем в строке graph[numstr] от текущей вершины единицу }
       for i:=graphind[numstr] to n-1 do
        if graph[numstr,i] = 1 then
         begin
         { Запоминаем номер вершины, с которой связана
           текущая вершина }
         graphind[numstr]:=i;
         { Проверяем, была ли пройдена вершина i }
         for j:=0 to circuitind do
          if i = circuit[j] then
           { Да, мы уже были в ней }
           begin
           { Если мы прошли n вершин и пришли в начальную
             вершину, то мы нашли цикл }
           if (i = firstvert) and (circuitind = n-1) then
            begin
            { Вывод цикла }
            for i:=0 to n-1 do Write(circuit[i]:nf);
            Writeln(firstvert:nf);
            Inc(numcircuits);
            { Делаем возврат на 2 вершины назад }
            circuitind:=n-3;
            numstr:=circuit[circuitind];
            end;
           { Продолжаем поиск в строке graph[numstr] }
           goto 10;
           end;
         { Вершина i еще не была пройдена. Запоминаем ее }
         Inc(circuitind);
```

```
          circuit[circuitind]:=i;
          { Переходим в строку i для перебора вершин, связанных
            с вершиной i }
          numstr:=i;
          graphind[numstr]:=-1;
          { И повторяем цикл поиска в строке graph[numstr] }
          goto 10;
          end;
        { Не нашли единиц в строке graph[numstr] от i-ой вершины }
        if circuitind = 0 then
         { Обошли весь граф }
         begin
         Writeln('Всего циклов: ',numcircuits);
         Writeln('Все циклы, состоящие лишь из ребер, посчитаны дважды!');
         goto EX;
         end;
        { Продолжаем поиск в предыдущей строке graph }
        Dec(circuitind);
        numstr:=circuit[circuitind];
      until FALSE;
EX:   Writeln('Press Enter to Exit...');
      Readln;
END.
```

У нашей программы есть один дефект: все циклы, целиком состоящие из ребер, обходятся, считаются и выводятся дважды — когда программа идет по ним в одну и другую сторону. Ведь мы не запоминаем найденные циклы, их может быть очень много. Предлагаю читателю в качестве домашнего задания сделать запоминание циклов, корректный их подсчет и вывод.

Граф называется *полным*, если в нем каждая вершина соединена с каждой из остальных. Что будет, если задать этой программе такой граф? Она тогда выдаст все перестановки `n` первых целых неотрицательных чисел, которые начинаются с нуля, точнее с числа, которое записано в `firstvert`! Значит, чтобы получить все перестановки, нужно обернуть основной цикл программы в цикл, в котором `firstvert` изменяется от `0` до `n-1`:

```
numcircuits:=0;

for firstvert:=0 to n-1 do
 begin
```

```
  graphind[firstvert]:=-1;
  circuitind:=0;
...
```

А вместо проверки

```
if circuitind = 0 then
 { Обошли весь граф }
 begin
 Writeln('Всего циклов: ',numcircuits);
 Writeln('Все циклы, состоящие лишь из ребер, посчитаны дважды!');
 goto EX;
 end;
```

надо написать:

```
if circuitind = 0 then goto 100;
```

Метку `100` поставим перед закрывающей скобкой `end` введенного цикла по `firstvert`:

```
        { Продолжаем поиск в предыдущей строке graph }
        Dec(circuitind);
        numstr:=circuit[circuitind];
       until FALSE;
100:  end;
```

В секции вывода цикла уберем оператор `Writeln(firstvert:nf)`, чтобы не повторять вывод номера начальной вершины. Такая программа при вводе матрицы, целиком состоящей из единиц, будет выдавать все перестановки точь-в-точь как это делает программа per1.pas. Об этом я и говорил в *разд. 1.3.1*. Сама программа находится в файле hper1.pas.

1.6.2. Блоха на клетчатой бумаге

На большом листе бумаги в клетку сидит блоха. При каждом прыжке она перемещается в одну из восьми соседних клеток. Сколькими способами она может сделать N прыжков, если она не прыгает в клетки, где была ранее?

Эту мою задачку олимпиадного типа мы решим с помощью маленькой рекурсивной подпрограммы. Но сначала поговорим немного о рекурсии. *Рекурсивными* называют такие функции, которые вызывают самих себя напрямую из своего тела или косвенно через вызовы (цепочки вызовов) других функций, обращающихся к вызвавшей их функции. Короче говоря, мы еще не вышли из данной функции, а уже снова в нее вошли.

При знакомстве с рекурсивными функциями обычно приводится классический пример вычисления факториала, т. е. произведения всех натуральных чисел до N включительно: $N! = 1 \times 2 \times ... \times N$, что, конечно, можно сделать с помощью простого цикла. Но не все рекурсивные функции можно так же легко раскрутить в последовательность итераций. Применительно к Turbo Pascal функция вычисления факториала выгладит так:

```
function Fact(n: word): longint;
begin
  if (n < 2) then Fact:=1 else Fact:=n*Fact(n-1);
end;
```

Рассмотрим на примере, как работает эта функция. Пусть `n` равно трем. Если в начало функции вставить предложение

```
Writeln('n=',n);
```

а в основную программу

```
Writeln(Fact(3));
```

то получим такой вывод:

```
n=3
n=2
n=1
6
```

`6` — это результат вычислений. При первом обращении функция `Fact` видит, что `n` не меньше двух и выполняет предложение `3*Fact(2)`. При втором обращении с аргументом `2` происходит выполнение предложения `2*Fact(1)`. И лишь на третьем уровне вложенности наша функция может сделать расчет без помощи других вызовов. Она просто возвращает единицу. Но возвращает она ее не в основную программу, а на предыдущий уровень, где предложение `2*Fact(1)` превращается в `2*1` и вычисляется. Этот результат, равный двум, передается на самый верхний уровень вызова, где вычисляется предложение `3*Fact(2)`, которое превращается в `3*2`. Это число возвращается в основную программу и отображается на экране. В случае рекурсии промежуточные результаты вычислений запоминаются во фреймах стека в экземплярах локальных переменных этой функции. В нерекурсивной программе hamilton.pas для запоминания этих промежуточных значений мы использовали массивы `graphind`, `circuit`, скалярные переменные `circuitind`, `numstr` и счетчики циклов.

Вернемся к нашей блохе. Для отметки клеток, в которых она была ранее, заведем двумерный массив `matr`. Чтобы как-то определиться с его размерами, зададим максимальное число прыжков в константе `MAXOFJUMPS=10`. То-

гда матрица из клеток для отметки прыжков определится как `matr: array[0..2*MAXOFJUMPS,0..2*MAXOFJUMPS] of byte`. Блоха начнет прыгать из центра, т. е. из клетки `matr[numofjumps,numofjumps]`. За `MAXOFJUMPS` прыжков она не выйдет за пределы массива `matr`.

Первоначально все элементы массива `matr` содержат нули. Как только блоха появляется в какой-нибудь клетке, мы записываем в соответствующий элемент массива `1`.

Чтобы перебрать в цикле все восемь направлений прыжков из клетки, введем два массива:

```
ADDX: array[0..7] of integer = (0,-1,-1,-1,0,1,1,1);
ADDY: array[0..7] of integer = (-1,-1,0,1,1,1,0,-1);
```

Нулевые элементы массивов задают направление вверх, следующие элементы — влево, вверх и влево и т. д. против часовой стрелки. (Мы как обычно в программировании предполагаем, что ось *y* направлена вниз.) Например, если блоха из клетки `matr[x,y]` прыгает вверх, то она попадает в клетку `matr[x+ADDX[0],y+ADDY[0]]`. Действительное число прыжков `numofjumps` будем получать от пользователя. Оно должно быть в пределах от `1` до `MAXOFJUMPS`. В силу симметрии достаточно просчитать лишь два начальных направления — одно для прыжка через сторону клетки и другое через вершину. Ответ получится умножением числа прыжков для этих начальных направлений на четыре. В переменной `count` будем накапливать результирующее число прыжков, которое после умножения на четыре будем выдавать как ответ. Процедуру для подсчета числа способов совершения `jumpscount` прыжков, первый из которых имеет данное направление, назовем `FleaJump`. Она будет получать три параметра:

- `x`-координату в массиве `matrix`, где сейчас сидит блоха;
- `y`-координату в массиве `matrix`, где сейчас сидит блоха;
- направление следующего прыжка (`0` — вверх, `1` — влево вверх, ..., `7` — вправо вверх).

Эта процедура будет рекурсивной и станет перебирать все варианты совершения `numofjumps` прыжков, начиная с заданного направления для первого прыжка. Например, при обращении `FleaJump(9,10,0)` эта процедура добавит к глобальной переменной `count` число вариантов совершения `numofjumps` прыжков, начиная из клетки с координатами `x=9`, `y=10` и с направлением `0` (вверх) для начального прыжка. Общий алгоритм работы процедуры для вызова `FleaJump(x,y,direction: integer)` такой:

1. Делаем прыжок из клетки (`x`, `y`) в направлении `direction` и увеличиваем `jumpscount` на единицу.

2. Проверяем, совершили ли мы уже `numofjumps` прыжков. Если `jumpscount = numofjumps`, то наращиваем `count` на единицу, возвращаемся в предыдущую клетку и выходим из процедуры.
3. Перебираем прыжки по всем восьми направлениям из новой клетки, вызывая для этого `FleaJump`.

В результате двух вызовов `FleaJump` из основной программы для направлений `0` и `1` мы в переменной `count` будем иметь число способов совершить `numofjumps` прыжков для этих двух направлений. Умножив его на четыре, выведем результат работы программы.

В листинге 1.16 представлен текст этой программы.

Листинг 1.16. Программа подсчета числа вариантов прыжков блохи

```
{$A-,B-,G+,F-,I-,R-,S-}
program Flea;
label EX;
const
 MAXOFJUMPS = 10;
 { Начальное направление - вверх, далее - против часовой стрелки }
 ADDX: array[0..7] of integer = (0,-1,-1,-1,0,1,1,1);
 ADDY: array[0..7] of integer = (-1,-1,0,1,1,1,0,-1);
var
 matr: array[0..2*MAXOFJUMPS,0..2*MAXOFJUMPS] of byte;
 i,jumpscount,numofjumps: integer;
count: longint;

{ Рекурсивная. Делает прыжок из клетки (x,y) по верному
  направлению direction }
procedure  FleaJump(x,y,direction: integer);
label 10;
var
 i: integer;
begin
      { Делаем прыжок }
      Inc(x,ADDX[direction]);
      Inc(y,ADDY[direction]);
```

```
      matr[x,y]:=1;
      Inc(jumpscount);
      { Совершили numofjumps прыжков? }
      if jumpscount = numofjumps then
       begin
       { Да, добавляем к результату 1 и возвращаемся
         в предыдущую клетку для дальнейшего перебора вариантов }
       Inc(count);
       goto 10;
       end;
      { Перебираем все прыжки из новой клетки }
      for i:=0 to 7 do
       if matr[x+ADDX[i],y+ADDY[i]] = 0 then FleaJump(x,y,i);
      { Возврат прыжка назад }
10:   matr[x,y]:=0;
      Dec (jumpscount);
end; {FleaJump}

BEGIN
      { Ввод числа прыжков }
      Writeln('Введите число прыжков от 1 до ',MAXOFJUMPS,':');
      Readln(numofjumps);
      if (numofjumps < 1) or (numofjumps > MAXOFJUMPS) then
       begin
       Writeln('Ошибка ввода');
       goto EX;
       end;
      { Заполняем нулями массив matr }
      FillChar (matr,sizeof(matr),0);
      { Обнуляем результат }
      count:=0;
      { Счетчик прыжков для каждого обращения к FleaJump }
      jumpscount:=0;
      { Сажаем блоху в центр matr }
      matr[numofjumps,numofjumps]:=1;
```

```
      { Суммируем количество прыжков для направлений 0 и 1 }
      for i:=0 to 1 do FleaJump(numofjumps,numofjumps,i);
      Writeln('Число вариантов для ',numofjumps,' прыжков = ',count*4);
EX:   Writeln('Press Enter to Exit...');
      Readln;
end.
```

"Почему переменная `jumpscount` обнуляется лишь раз в программе, а не каждый раз перед обращением к `FleaJump`?" — может спросить внимательный читатель. Потому что эта процедура вначале увеличивает этот счетчик на единицу, а перед выходом уменьшает тоже на один, поэтому в основной программе он всегда равен нулю.

Запустив программу, с интересом увидим, что подсчет числа вариантов для совершения десяти прыжков уже требует нескольких секунд.

1.6.3. Просчитываем головоломку Full Board

В этом разделе мы рассмотрим интересную головоломку известного американского создателя головоломок Эриха Фридмана (Erich Friedman). Она носит название Full Board. Правила ее просты: имеется квадрат, разделенный на клетки, некоторые клетки заняты. Нам надо выбрать свободную клетку и, начав из нее движение, обойти все свободные клетки, побывав в каждой из них ровно по одному разу. В занятые клетки заходить нельзя. Движение осуществляется с помощью ходов. При каждом ходе фишка движется от клетки к клетке через их стороны до упора в занятую клетку, край доски или клетку, где фишка уже была.

Эта головоломка заинтересовала меня простотой правил, и я сделал для нее свой онлайновый вариант на Flash. Вы можете найти его на моем игровом сайте **www.gameintellect.com**. Также вы можете запустить игру с компакт-диска, который прилагается к книге. Для этого загрузите в браузер файл fullboard.html.

Я подскажу, как решается **Level 1**: щелкните мышью на центральной клетке в верхней строке — в этой клетке появится голубая фишка. Затем щелкните в какой-нибудь клетке той же строки справа от этой фишки — фишка переместится вправо до упора. Затем щелкайте в какой-либо пустой клетке перед фишкой, и она будет перемещаться по указанному вами направлению. На рис. 1.1 вы видите позицию из **Level 1** за один ход до решения.

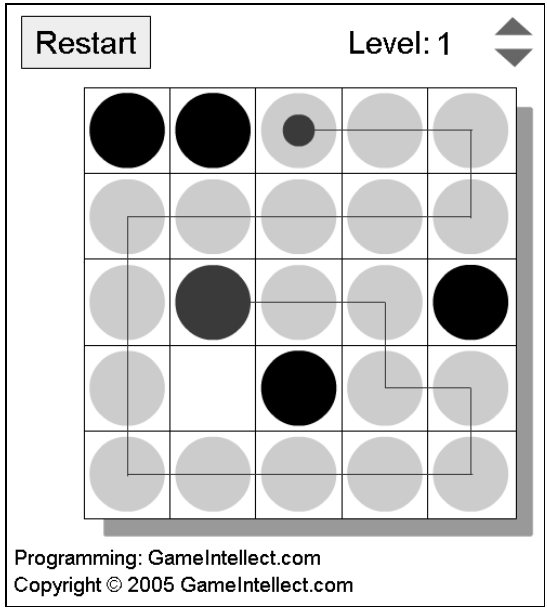


Рис. 1.1. Первый уровень головоломки Full Board

Автор нашел девять позиций с единственным решением. Встречаются поля размером от 5 × 5 до 7 × 7. Я тоже попробовал создать свою позицию для решения, это **Level 5**. Но моя радость творчества оказалась обманчивой: я нашел для этой позиции второе решение. Да, тяжела профессия составителя головоломок. Сколько времени порой надо потратить, чтобы найти какую-либо позицию с единственным решением. И тогда я решил поручить эту задачу электронному мозгу — компьютеру. Уж он-то щелкает подобные задачки как орехи, и голова у него не болит, если... если ему дадут правильную программу действий. Вот за ее создание мы сейчас и возьмемся.

Я иногда удивляюсь разработчикам игр: скрестил такой изобретатель го с рэндзю и говорит программисту: "Введи эту игру в свой компьютер, и пусть он скажет, кто побеждает при сильнейшей игре с обеих сторон". Если бы у меня был такой компьютер, разве стал бы я писать эту книгу? Ха, я бы сделал пластическую операцию по сужению разреза глаз, взял этот компьютер под мышку — и прямиком в Японию! Там за программу по го, которая играет хотя бы в силу первого дана, программиста сделают почетным гражданином Японии, а за программу, обыгрывающую любого человека, — императором или, как минимум, премьер-министром. Я бы уже сидел на мягкой циновке, попивал сакэ, клевал палочками рис и другие продукты японской кухни и общался с молоденькими гейшами, щурясь от удовольствия.

Итак, формулировка общей задачи: пользователь дает размер стороны квадрата и количество занятых клеток. На выходе получаются все позиции с единственным решением по правилам Full Board. В ходе работы пусть программа фильтрует найденные позиции и не выдает одну и ту же, повернутую на некоторый угол или еще с какой-нибудь симметрией. При этом позиции не должны запоминаться, а сразу выдаваться те, которые нам нужны. Как вам такая задачка? Выглядит солидно. В такой программе лучше применить рекурсию, иначе она слишком усложнится.

Переходим к обсуждению общего алгоритма программы. Задаем максимальную длину стороны квадрата `MAX=10` (вы можете увеличить ее хоть до `100`) и создаем доску `board` как одномерный массив байтов с границами от `0` до `100-1`. На всякий случай для ширины и высоты доски заведем отдельные переменные — `width` и `height`, ведь, вообще говоря, игровое поле может быть и не квадратным. (В этом случае код программы надо будет изменить.) Число клеток у рассматриваемого игрового поля равно `n:=width*height`, где `numengaged` — число занятых клеток. Запрашиваем длину стороны квадрата и число занятых клеток и заполняем массив `board` единицами для занятых клеток и нулями для свободных.

Немного о том, в каком виде программа будет выдавать решение. Очевидно, каждая позиция будет представляться `n` строками длиной по `n` символов. Занятые клетки будем отмечать звездочками. Как быть с показом этого единственного пути по всем свободным клеткам? Сделаем так: начальную клетку обозначим буквой `a`, все клетки, пройденные за первый ход, — буквами `b`, пройденные за второй ход — буквой `c` и т. д.

Еще нам надо будет последовательно получить все позиции $n \times n$ со всевозможной конфигурацией занятых клеток. Для каждой такой позиции мы хитрым рекурсивным перебором будем выяснять, имеет ли она единственное решение. Если имеет, то мы выведем его и перейдем к следующей конфигурации занятых клеток. Тут встает вопрос о переборе всех таких конфигураций. Изучим это на примере с доской размером 4×4 и тремя занятыми полями. Вот как будут выглядеть эти исходные позиции:

```
1110
0000
0000
0000

1101
0000
0000
0000
```

```
1100
1000
0000
0000
...
1100
0000
0000
0001

1011
0000
0000
0000
...
0000
0000
0000
0111
```

Ба, да это же размещение трех предметов по 16-ти местам! Это мы уже проходили. Только в этом примере я изобразил массив `board` в виде двумерного, но на самом деле он в программе одномерный ввиду независимости программы от размеров доски. Поэтому матрицы, вообще говоря, должны быть вытянуты в одну строку:

```
1110000000000000
1101000000000000
1100100000000000
...
1100000000000001
1011000000000000
...
0000000000000111
```

Условимся, что первый (левый) элемент этих массивов будет иметь индекс `0`, а последний (правый) индекс `n-1`. В программе для генерации всех размещений используем функцию `Next`, которую мы уже написали.

А теперь обратите внимание на первую и последнюю позиции. Они симметричны и получаются одна из другой поворотом доски на 180°. Очевидно, решения тоже будут симметричны. Мы должны от всех симметричных позиций оставлять по одной.

Всего квадрат имеет восемь симметрий, которые образуют группу: это повороты на 0°, 90°, 180° и 270° и отражения относительно больших диагоналей и отрезков, соединяющих середины противоположных сторон. Ну, поворот на 0° (тождественная симметрия) нас здесь не волнует, а все остальное надо учитывать. Как же выбирать одну позицию из группы симметричных? Поступим так: будем рассматривать массив `board` как двоичное число, и из всех симметричных друг другу позиций будем рассматривать только ту, для которой это число будет наибольшим. Например, из двух позиций

```
1110000000000000
```

и

```
0000000000000111
```

мы оставим первую.

Каким образом мы это будем делать? Когда функция `Next` выдаст нам очередную позицию, мы из нее во вспомогательном массиве будем получать остальные семь симметричных ей позиций и каждый раз сравнивать их по величине этого двоичного числа. Если какое-либо из этих семи чисел окажется больше исходного, то мы не станем рассматривать такую позицию и будем вызывать функцию `Next` дальше. Этот случай будет говорить о том, что одну из этой группы симметричных позиций мы уже рассмотрели ранее, ведь функция `Next` выдает позиции в порядке уменьшения этого двоичного числа.

Основной цикл программы здесь получается простым:

```
repeat
...
until not Next(board,n);
```

Итак:

1. Получаем начальную позицию в массиве `board` в соответствии с величинами `n` и `numengaged`.
2. Здесь начало основного цикла. Проверяем `board` на наличие симметричных уже пройденных позиций функцией `AlreadyPassed`. Если мы уже обрабатывали симметричную позицию, то переходим на повтор основного цикла. Присвоим переменной `solnum 0`, это будет счетчик найденных решений для данной позиции. Если он будет равен единице, то такая позиция нам подходит, и мы ее выведем на экран.

3. Цикл по всем пустым полям массива `board`. Для каждого такого поля пишем в него символ `a` и четырежды вызываем процедуру `Step`, которая делает ход в заданном направлении (влево, вверх, вправо и вниз) и затем рекурсивно перебирает все ходы до конца. Каждый раз, найдя путь через все пустые клетки, `Step` увеличит на `1` глобальную переменную `solnum`. Если после окончания цикла четырех шагов в разных направлениях окажется, что `solnum=1`, то мы выводим эту позицию. Внутри процедуры `Step` тоже есть проверка переменной `solnum`, чтобы не делать лишнего перебора после нахождения второго решения. Мы применим новаторский прием для перехода из этой процедуры в основную программу сразу на повтор основного цикла программы. В конце этого цикла надо предусмотреть очистку массива `board` от букв, чтобы он принял первозданный вид перед очередным обращением к функции `Next`.

А теперь рассмотрим работу процедуры `Step`, которая является центральной в нашей программе. Она получает такие параметры:

- направление движения — байт `lurd` (left, up, down, right), который содержит, соответственно, `0`, `1`, `2` или `3`;
- массив `m` типа `mb=array[0..MAX*MAX-1] of byte`. Это наша доска, которая передается в стеке, благодаря чему `Step` начинает работать с копией массива `board`, в котором уже стоит буква `a` в клетке, из которой начинается движение. Когда эта рекурсивная процедура продолжит обращаться к себе самой, то она при этом будет передавать экземпляры массива `m`, в котором будут находиться ходы `b`, `c` и т. д. При возврате в случае попадания в тупик в место процедуры `Step`, откуда был произведен вызов, фрейм стека с последним значением массива `m` и последним неудачным ходом освободится, и `Step` продолжит вызывать себя с предыдущего хода. Если в отладчике вы посмотрите на работу этой процедуры, то каждый раз будете видеть текущие экземпляры локальных переменных;
- переменная `ind` — это индекс пустой клетки, в которую программа поставила букву `a`. Из этой клетки `Step` продолжит делать ходы;
- переменная `numfree` — число оставшихся пустых клеток. При вызове `Step` из основной программы `numfree` будет иметь значение `width*height-numengaged-1`. Далее `Step` при каждом ходе будет уменьшать `numfree` на число клеток, через которое прошел очередной сделанный ход. Если после очередного хода окажется, что `numfree=0`, то решение найдено, и надо проверять значение переменной `solnum`. Если ходить больше некуда, а `numfree > 0`, то мы попали в тупик, и надо делать возврат в место вызова процедуры `Step` для отмены последнего хода;
- переменная `putnum`, которая содержит как бы номер следующего хода. Но мы вместо цифр используем буквы `a`, `b`, `c`, ..., потому что цифр нам не хватит для показа ходов при выводе позиции, а появление двузначных

чисел сделает вывод неудобным для понимания. Перед обращением к себе `Step` увеличивает на `1` этот байт.

Алгоритм работы процедуры `Step` таков:

1. Начинается процедура оператором `case lurd of`, который имеет четыре ветви в зависимости от заданного в этом байте направления хода. Все его ветви аналогичны, поэтому мы рассмотрим ветвь, помеченную нулем. Итак, если `lurd=0`, то следует сделать ход влево из клетки, имеющей индекс `ind` в массиве `m`. Но вначале надо проверить, что слева от этой клетки находится свободная клетка. Эта проверка разбивается на две: данная клетка `ind` — не самая левая, и элемент `m[ind-1]` равен нулю. Здесь важен порядок следования этих проверок и тот факт, что транслируется наша программа с опцией `$B-`. При этой опции логические условия вычисляются по короткой схеме, поэтому, если выяснится, что клетка `m[ind]` находится в левом столбце, то проверки, что `m[ind-1]` равно нулю, не будет. А если бы логические выражения вычислялись полностью, то мы в этом месте могли бы выйти за пределы массива `m` (при `ind=0`). Первое условие выглядит так: `ind mod width > 0`. Действительно, `ind` делится без остатка на ширину доски только для клеток левого столбца, которые имеют индексы `0`, `width`, `width*2`, `width*3`, ..., `width*(n-1)`. Если мы имеем слева пустую клетку, то выполняем цикл `repeat`, в котором двигаемся на клетку влево до упора: вычитаем из `ind` единицу, присваиваем `m[ind]:=putnum`, уменьшаем `numfree` на `1`. Условие выхода из этого цикла `repeat` будет `until (ind mod width = 0) or (m[ind-1] <> 0)`, оно должно быть понятно. После этого цикла мы проверяем условием `numfree=0`. Если оно выполняется, то мы обошли всю доску и переходим на метку `10`. Если нет, то перебираем ходы вверх и вниз из новой клетки, делая пару вызовов процедуры `Step`: `Step(1,m,ind,numfree,putnum+1)` и `Step(3,m,ind,numfree,putnum+1)` и затем выходим из этого уровня рекурсии оператором `Exit`. Таким образом мы перебираем все ходы из новой клетки `m[ind-1]`, ведь после хода влево нет ходов влево и вправо, а остальные ходы мы просматриваем до конца, потому что такова наша процедура `Step`.
2. Если в п. 1 условие `if (ind mod width > 0) and (m[ind-1] = 0)` не выполняется, то ход в цикле `repeat` пропускается, и мы переходим к п. 3, куда попадаем, если больше нет хода.
3. Если `numfree = 0`, то выходим из данного уровня рекурсии процедуры `Step`.
4. **Метка `10`.** Наращиваем `solnum` на `1`. Если теперь `solnum > 1`, то надо совсем уходить из процедуры `Step`, потому что заданная этой процедуре позиция имеет два решения. Как мы это будем делать, расскажу чуть позже. Остается вариант, когда `solnum` равно `1`. В этом случае надо за-

помнить массив m в глобальном массиве bsol того же типа. Turbo Pascal перепишет массив по одному оператору присваивания bsol:=m. И здесь опять выход из данного уровня этой рекурсивной процедуры, который транслятор вставляет автоматически при встрече завершающего оператора end.

Если читатель раньше не сталкивался с рекурсивными подпрограммами и не совсем понял этот алгоритм — не беда. Человеческий ум плохо приспособлен к рекурсивным размышлениям ввиду отсутствия у него стека для запоминания предыдущих состояний, и на втором уровне вложенности мы уже не помним, в какое состояние надо возвращаться. Тут важен опыт, и мы в этой книге еще будем иметь случай применить этот красивый прием в игровой программе.

А сейчас выясним, как и куда надо "совсем уходить" из процедуры Step при обнаружении второго решения. Напомню, что основной цикл программы заканчивается условием MNEXT: until not Next(board,n). На метку MNEXT мы переходим, если функция AlreadyPassed обнаружила, что симметричная текущей позиция была уже рассмотрена. В случае прыжка из процедуры Step через все ее рекурсивные вызовы в основную программу сразу прыгать к метке MNEXT было бы неправильно, ведь у нас в массиве board где-то осталась буква a, которую надо заменить нулем. Поэтому мы перед этой меткой поставим метку MCLEAR, где и будем это делать. Для этого в цикле по j по пустым клеткам board после оператора board[j]:=byte('a') поставим оператор j1:=j, чтобы запомнить индекс клетки с буквой a. Turbo Pascal не имеет оптимизаций, он не держит счетчики циклов в регистрах процессора, поэтому вне цикла j1 будет равняться j. Но надеяться на это — плохая практика, т. к. при портировании программы в другие системы программирования, которые поддерживают оптимизацию, мы за пределами цикла по j будем иметь в этой переменной не последнее значение счетчика цикла, а что-то другое.

Как же мы будем делать этот прыжок? Очень просто: Turbo Pascal при входе в процедуру или функцию использует регистры BP и SP для создания фрейма стека с локальными переменными. В основной программе значения этих регистров постоянны. Поэтому в начале программы мы запомним значения этих регистров в глобальных переменных regbp и regsp. А чтобы перейти на какой-либо оператор в основной программе, надо в регистр IP (Instruction pointer, указатель команды) процессора записать смещение в сегменте кода этого оператора. Для этого в том же месте основной программы запоминаем в глобальной переменной regip смещение метки MCLEAR оператором ассемблера mov regip,offset MCLEAR. Для этого метка должна начинаться с буквы, чтобы транслятор не заругался. Так как переход у нас близкий, внутри сегмента кода, то модифицировать регистр сегмента кода CS не нужно, и переход из процедуры Step на метку MCLEAR осуществляется командой ас-

семблера `jmp [regip]`. Вы можете это проверить в отладчике: при нажатии клавиши <F8> на этой команде подсветка для текущей команды, как ни в чем не бывало, переместится на строчку `MCLEAR: board[j1]:=0`. Мы видим, что наш трюк прозрачен для отладчика, это приятно.

Остается рассмотреть процедуры симметричного отражения доски и исключения повторных с точностью до симметрии позиций. Для нас достаточно иметь процедуру `Rotate90`, вращающую доску на 90° против часовой стрелки, процедуру `Diagonal0` для отражения относительно главной диагонали, `Diagonal1` — относительно побочной, `Horizontal` для отражения относительно средней горизонтали (верх на низ) и `Vertical` — относительно средней вертикали (правую половину на левую). Все эти процедуры получают три параметра: `var`-параметры `matr`, `matr1` и длину стороны квадрата (или матрицы) `n`. В `matr` мы передаем массив `board`, а на выходе в `matr1` получаем этот массив после соответствующего преобразования. Для этих процедур существенно то, что доска у нас квадратная. Работу этих процедур обсуждать не будем, метод совершения преобразований можно понять на конкретных примерах с ручкой и бумагой. Трудность здесь в том, что матрицы у нас с произвольной длиной стороны, и поэтому массивы под них используются одномерные.

Аналогично не будем разбирать алгоритм короткой функции `GreaterOrEqual`, которая сравнивает массивы `b1` и `b` типа `mb`, каждый из которых имеет размер `n` байтов, рассматривая их как двоичные числа, у которых старшие разряды начинаются с меньших индексов. Если число в `b1` больше или равно `b`, то возвращается `TRUE`, иначе `FALSE`.

Функция `AlreadyPassed` получает на входе сам массив доски в параметре `b`, а длину стороны в `m`. Далее эта функция получает в локальных массивах всевозможные симметричные преобразования массива `b` и каждый раз вызывает функцию `GreaterOrEqual`, сравнивая по величине исходный массив с полученным. Если полученный массив оказывается не меньше исходного `b`, то возвращается `TRUE`, а после всех преобразований и проверок возвращается `FALSE`. Это говорит о том, что массив `b` для очередной позиции, выданной функцией `Next`, надо исследовать на предмет существования единственного решения для первого хода из всех его пустых клеток.

В листинге 1.17 представлен текст программы fullboar.pas.

Листинг 1.17. Программа выдачи всех позиций ровно с одним решением в Full Board

```
{$A-,B-,G+,F-,I-,R+,S-}
{$M 32768,0,655360}
program FullBoard;
```

```
LABEL MCLEAR,MNEXT;
CONST
 MIN=4;
 MAX=10;
TYPE
 { Игровое поле максимум на MAX*MAX клеток }
 mb=array[0..MAX*MAX-1] of byte;
VAR
 board: mb;
 bsol: mb;
 i,j,j1,k,l,
 { Ширина игрового поля }
 width,
 { Высота игрового поля }
 height,
 { Число клеток поля, n = width*height }
 n,
 { Число занятых клеток }
 numengaged,
 solnum,regbp,regsp,regip: word;
 numpos: longint;
 berr: boolean;

{  Выдача следующего сочетания без повторения
   в массиве m[0], ..., m[n-1].
   Если оно найдено, то возвращается TRUE, если заданное сочетание
   было последним, массив m не меняется и возвращается FALSE. Пустые
   места в массиве m содержат нули, занятые места содержат 1 }
function Next(var m: mb; n: word): boolean;
LABEL 10;
VAR i,j,ct1: word;

begin
      { Ищем от конца m пару m[i] = 0 и m[i-1] > 0 }
      ct1:=0;
      for i:=n-1 downto 1 do
```

```
       begin
       if m[i] = 1 then
        begin
        Inc(ct1);
        goto 10;
        end;
       if m[i-1] = 1 then
        { Продвигаем m[i-1] на шаг вправо }
        begin
        m[i]:=1;
        m[i-1]:=0;
        { Переписать правые элементы? }
        if ct1 > 0 then
         { Выводим j на 1-е место m[j] = 1 }
         begin
         Inc(i);
         for j:=i to i+ct1-1 do m[j]:=1;
         for j:=i+ct1 to n-1 do m[j]:=0;
         end;
        Next:=TRUE;
        Exit;
        end;
10:    end;
      { Последнее сочетание }
      Next:=FALSE;
end; {Next}

{ Делает один ход в данном направлении с рекурсией }
procedure Step(lurd: byte; m: mb; ind,numfree: word; putnum: byte);
LABEL 10;
VAR
 i: word;

begin
      case lurd of
```

```
0:      if (ind mod width > 0) and (m[ind-1] = 0) then
         { Делаем ход влево из клетки номер ind }
         begin
         repeat
          Dec(ind);
          m[ind]:=putnum;
          Dec(numfree);
         until (ind mod width = 0) or (m[ind-1] <> 0);
         if numfree = 0 then goto 10;
         { Перебираем ходы из новой клетки }
         Step(1,m,ind,numfree,putnum+1);
         Step(3,m,ind,numfree,putnum+1);
         Exit;
         end;
1:      if (ind >= width) and (m[ind-width] = 0) then
         { Делаем ход вверх из клетки номер ind }
         begin
         repeat
          Dec(ind,width);
          m[ind]:=putnum;
          Dec(numfree);
         until (ind < width) or (m[ind-width] <> 0);
         if numfree = 0 then goto 10;
         { Перебираем ходы из новой клетки }
         Step(0,m,ind,numfree,putnum+1);
         Step(2,m,ind,numfree,putnum+1);
         Exit;
         end;
2:      if (ind mod width < width-1) and (m[ind+1] = 0) then
         { Делаем ход вправо из клетки номер ind }
         begin
         repeat
          Inc(ind);
          m[ind]:=putnum;
          Dec(numfree);
         until (ind mod width = width-1) or (m[ind+1] <> 0);
```

```
        if numfree = 0 then goto 10;
        { Перебираем ходы из новой клетки }
        Step(1,m,ind,numfree,putnum+1);
        Step(3,m,ind,numfree,putnum+1);
        Exit;
        end;
3:      if (ind < width*(height-1)) and (m[ind+width] = 0) then
        { Делаем ход вниз из клетки номер ind }
        begin
        repeat
         Inc(ind,width);
         m[ind]:=putnum;
         Dec(numfree);
        until (ind >= width*(height-1)) or (m[ind+width] <> 0);
        if numfree = 0 then goto 10;
        { Перебираем ходы из новой клетки }
        Step(0,m,ind,numfree,putnum+1);
        Step(2,m,ind,numfree,putnum+1);
        Exit;
        end;
      end; {case}
      { Нет хода }
      { Нашли решение? }
      if numfree = 0 then
       { Да }
       begin
10:    Inc(solnum);
       { Оно не первое для этой позиции? }
       if solnum > 1 then
        { Не первое. Переходим к метке MCLEAR основной программы }
        asm
        mov bp,regbp
        mov sp,regsp
        jmp [regip]
        end;
```

```
       { Это первое решение для данной позиции. Запоминаем его }
       bsol:=m;
       end;
end; {Step}

{ Вращение квадратной матрицы matr со стороной n против часовой стрелки
  на 90 градусов и запись результата в matr1 }
procedure Rotate90(var matr,matr1: mb; n : word);
VAR i,j: word;

begin
      for i:=0 to n-1 do
       for j:=0 to n-1 do
        matr1[n*(n-j-1)+i]:=matr[n*i+j];
end; {Rotate90}

{ Отражение относительно главной диагонали из matr в matr1 }
procedure Diagonal0(var matr,matr1: mb; n : word);
VAR i,j: word;

begin
      for i:=0 to n-1 do
       for j:=0 to n-1 do
        matr1[n*j+i]:=matr[n*i+j];
end; {Diagonal0}

{ Отражение относительно побочной диагонали из matr в matr1 }
procedure Diagonal1(var matr,matr1: mb; n : word);
VAR i,j: word;

begin
      for i:=0 to n-1 do
       for j:=0 to n-1 do
        matr1[n*(n-j)-i-1]:=matr[n*i+j];
end; {Diagonal1}
```

```
{ Отражение относительно средней горизонтали из matr в matr1 }
procedure Horizontal(var matr,matr1: mb; n : word);
VAR i,j: word;

begin
      for i:=0 to n-1 do
       for j:=0 to n-1 do
        matr1[n*(n-i-1)+j]:=matr[n*i+j];
end; {Horizontal}

{ Отражение относительно средней вертикали из matr в matr1 }
procedure Vertical(var matr,matr1: mb; n : word);
VAR i,j: word;

begin
      for i:=0 to n-1 do
       for j:=0 to n-1 do
        matr1[n*(i+1)-j-1]:=matr[n*i+j];
end; {Vertical}

{ Выясняем, больше или равно число b1 чем b. Размер чисел - n
  двоичных разрядов по одному разряду в байте. Хранятся числа,
  начиная со старших разрядов. Если да, возвращаем TRUE }
function GreaterOrEqual(var b1,b: mb; n: word): boolean;
VAR
 i: word;

begin
      GreaterOrEqual:=TRUE;
      for i:=0 to n-1 do
       begin
       if b1[i] > b[i] then Exit;
       if b[i] > b1[i] then
        begin
        GreaterOrEqual:=FALSE;
        Exit;
```

```
        end;
       end;
end; {GreaterOrEqual}

{ Выясняем, получали ли мы уже позицию, заданную в b,
  размером m*m клеток }
function AlreadyPassed(b: mb; m: word):boolean;
VAR
 b1,b2: mb;
 n: word;

begin
      AlreadyPassed:=TRUE;
      n:=m*m;
      Rotate90(b,b1,m);
      if GreaterOrEqual(b1,b,n) then Exit;
      Rotate90(b1,b2,m);
      if GreaterOrEqual(b2,b,n) then Exit;
      Rotate90(b2,b1,m);
      if GreaterOrEqual(b1,b,n) then Exit;
      Diagonal0(b,b1,m);
      if GreaterOrEqual(b1,b,n) then Exit;
      Diagonal1(b,b1,m);
      if GreaterOrEqual(b1,b,n) then Exit;
      Horizontal(b,b1,m);
      if GreaterOrEqual(b1,b,n) then Exit;
      Vertical(b,b1,m);
      if GreaterOrEqual(b1,b,n) then Exit;
      AlreadyPassed:=FALSE;
end; {AlreadyPassed}

BEGIN
      { Получаем параметры программы }
      berr:=FALSE;
      repeat
       if berr then Writeln('Ошибка ввода.');
```

```
 Write('Введите через пробел длину стороны квадрата '+
  'и число занятых клеток: ');
 width:=0;
 numengaged:=0;
 Readln(width,numengaged);
 berr:=(width < MIN) or (width > MAX) or (numengaged < 1) or
  (numengaged > width*(width-1));
until not berr;
height:=width;
n:=width*height;
{ Готовим адрес перехода для процедуры Step }
asm
mov regbp,bp
mov regsp,sp
mov regip,offset MCLEAR
end;
numpos:=0;
for i:=numengaged to numengaged do
 begin
 { Создаем очередную позицию. Занятым клеткам присваиваем 1 }
 FillChar(board,sizeof(board),#0);
 FillChar(board,i,#1);
 repeat
  { Проверяем на наличие симметричных уже полученных позиций }
  if AlreadyPassed(board,width) then goto MNEXT;
  { Обрабатываем очередную позицию }
  solnum:=0;
  for j:=0 to n-1 do
   if board[j] = 0 then
    begin
    { Выбираем начальное поле }
    board[j]:=byte('a');
    j1:=j;
    { Влево, вверх, вправо, вниз. Ширина, высота }
    for k:=0 to 3 do
     begin
     Step(k,board,j,width*height-i-1,byte('b'));
```

```
          if solnum > 1 then goto MCLEAR;
          end;
         board[j]:=0;
         end;
        if solnum = 1 then
         begin
         for k:=0 to height-1 do
          begin
          for l:=0 to width-1 do
           if bsol[k*height+l] = 1 then
            Write('*') else
             Write(char(bsol[k*height+l]));
          Writeln;
          end;
         Writeln('===');
         Inc(numpos);
         end;
       goto MNEXT;
       { Обнуляем клетку с буквой a }
MCLEAR: board[j1]:=0;
MNEXT: until not Next(board,n);
       end;
     Writeln('Всего найдено позиций: ',numpos);
     Writeln('Press Enter to Exit...');
     Readln;
END.
```

Внимательный читатель наверно заметил, что основной цикл программы обернут в цикл `for i:=numengaged to numengaged do`. Это позволяет за один вызов программы получать все решения с разным числом занятых клеток, изменяя начальное и конечное значения счетчика цикла `i`. Можно сделать еще один внешний цикл по длине стороны `width` доски и получить сразу все позиции для разных досок.

Замечание

Программа fullboar.pas размещена на компакт-диске исключительно с целью обучения и не может быть использована в любых иных целях. Вы не имеете права публиковать где бы то ни было и в любой форме как текст этой програм-

мы, так и выдаваемые ею позиции. Также вы не имеете права изменять текст этой программы кроме как с целью ее изучения. Если вам интересна эта головоломка, возьмите ее вариант Hex Board с шестиугольными полями, напишите аналогичную моей программу и публикуйте свои результаты. Не забудьте при этом указать, что оригинальная идея принадлежит Эриху Фридману. Его страницу с головоломками вы найдете по адресу **http://www.stetson.edu/~efriedma/puzzle.html**.

Я думаю, читатель понял, как трактовать полученные решения: начинать надо с буквы `a` и далее двигаться по буквам `b`, `c` и т. д. до последней буквы.

Домашнее задание

Выдаваемые программой позиции для доски одного и того же размера с тем же числом занятых полей неравнозначны по сложности решения. Например, позиция

```
a**ed
bhifd
bhjfd
bggfd
bcccc
```

в верхней левой клетке имеет тупик, и поэтому маршрут должен либо начинаться, либо кончаться в этой клетке. Это облегчает решение. В позиции

```
****a
deeeb
dhifb
dggfb
ccccb
```

кроме того, все занятые поля расположились в один ряд. Это тоже ее не украшает. Интуитивно кажется, что трудность решения определяется отсутствием тупиков, большим числом поворотов и тем, много ли раз ходы упираются в уже пройденные клетки, в зависимости от размеров доски и числа занятых клеток. Разработайте эвристический алгоритм, который по заданным позициям будет выдавать число, означающее уровень сложности позиции.

1.6.4. Создаем свой оригинальный "Ним"

Еще в древнем Китае была известна такая игра: имеются три кучки камней; двое играющих поочередно берут камни из этих кучек, причем при каждом ходе играющий может взять любое ненулевое число камней из любой одной кучки. Выигрывает тот, кто возьмет последний камень.

Эта игра известна во множестве разновидностей. Когда-то во времена "золотой лихорадки" посетители раскладывали на стойке бара в три ряда монеты, чтобы определить, кто должен платить за виски. Естественно, это делал проигравший. Аналогично другим логическим играм, как, например, шахматы, здесь надо было считать варианты: если я сделаю такой ход, то противник может походить так, тогда я могу ответить так или так... В общем, на третьем ходу уже трудно понять, кто как может походить, особенно учитывая количество выпитого. Но в начале XX века один математик исследовал эту игру и нашел простое арифметическое правило для определения, выигрышна данная позиция или нет, а если выигрышна, то можно было просто найти ходы, ведущие к победе.

На компакт-диске, прилагаемом к данной книге, имеется моя программа на Flash, играющая в этот "Ним". Для игры откройте в браузере файл nim.html. В предыдущей своей книге "Создание игр во Flash MX"[3] я описал программирование этой игры на ActionScript. Книгу вы можете увидеть на сайте магазина books.ru на странице **www.books.ru/shop/books/323172** или найти во всех онлайновых магазинах на странице **www.findbook.ru** по ISBN 5-94157-629-3. Здесь же я изложу правило определения выигрышности позиции, и мы рассмотрим другой вариант "Нима", свободный от этого недостатка.

На рис. 1.2 вы видите окно проигрывателя Flash с этой игрой.

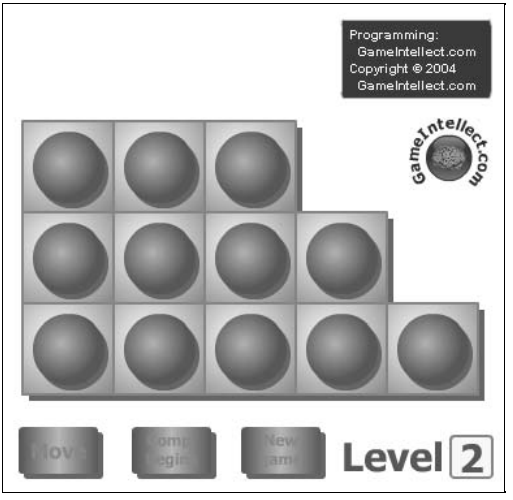


Рис. 1.2. Окно Flash-игры "Ним"

[3] Мельников С. В. Создание игр во Flash MX. — СПб.: БХВ-Петербург, 2005.

Для выполнения хода надо отметить убираемые фишки, щелкнув на них мышью, и затем щелкнуть по кнопке **Move** (Ход). Щелкните в начале игры по кнопке **Comp. begins** для передачи первого хода компьютеру.

Чтобы понять несложную выигрывающую стратегию, немного переформулируем игру на математический лад. Запишем числа фишек в этих трех строках в двоичной системе счисления, дополним некоторые из них левыми нулями для того, чтобы все числа имели одно и то же количество цифр, и подпишем их друг под другом. Мы имеем числа 3, 4 и 5:

```
011
100
101
```

Ответ на вопрос, выигрышна ли эта позиция, мы получим, рассматривая столбцы этой таблицы двоичных цифр. Если в каком-либо столбце количество единиц нечетно, то игрок, чья очередь ходить, имеет выигрыш. Мы видим, что во втором столбце стоит одна единица, а это значит, что исходная позиция в этом варианте "Нима" выигрышна.

Как же сделать этот выигрышный ход? Для этого надо взять такое количество фишек из какой-то строки, чтобы во всех столбцах количество единиц стало четным. Это всегда можно сделать. В данном случае надо вычеркнуть единицу из первой строки и второго столбца, т. е. взять из верхней строки две фишки. Это единственный выигрывающий ход: записывая любым другим образом в любой из строк меньшее число, чем там имеется, мы не получим четное число единиц во всех столбцах. Соперник в ответном ходе неизбежно изменит четность числа единиц хотя бы в одном столбце, и следующим ходом мы опять эту четность восстановим и т. д. вплоть до того момента, когда после нашего хода во всех строках не останутся нули. Это правило работает также и в общем случае при любом числе кучек и любом числе камней в каждой кучке. Если выбирать случайным образом число кучек и число камней в каждой из них, то выигрывать в абсолютном большинстве случаев будет первый игрок.

После того как становится известной выигрывающая стратегия, игра перестает представлять интерес (а игрок, владеющий ею, рискует заболеть белой горячкой от неумеренного потребления виски за чужой счет). Поэтому прогрессивная часть человечества придумывает варианты игр, свободные от этого недостатка. Вот один из моих вариантов "Нима": имеется несколько карточек, на каждой из которых написано по четыре случайных числа из диапазона от `0` до `9`. Вместо десяти чисел можно нарисовать десять различных предметов или взять десять различных цветов. За один ход можно выбрать любое число, которое есть на оставшихся карточках, и взять все карточки, на которых оно написано. Если после вас уже нечего брать, считайте, что вы выиграли.

Рассмотрим пример, как надо играть. Пусть мы имеем такую позицию с пятью карточками:

```
7 5   8 2   6 9   0 3   9 3
3 6   1 7   1 0   7 8   4 7
```

Когда-то на днях мехмата в НГУ прозвучала такая шутка: студентам будет прочитан курс лекций на тему "Методы отчисления". Лекция первая: "Метод последовательного исключения". Мы здесь будем искать выигрывающий ход этим методом.

Сначала сделаем ход 0, убрав 3-ю и 4-ю карточки. Получим такую позицию:

```
7 5   8 2   9 3
3 6   1 7   4 7
```

В ней цифра 7 написана на всех карточках, соответственно, сделав этот ход, соперник выиграет партию. Нет, ход 0 не годится. Тогда не годится и ход 7 из-за ответа 0. По аналогичной причине не годится ход 1 (из-за цифры 3) и ход 3 из-за хода 1. Попробуем ход 2, имеем:

```
7 5   6 9   0 3   9 3
3 6   1 0   7 8   4 7
```

Второй игрок может сделать ход 4 и снять последнюю карточку. Оставшиеся две карточки можно взять ровно двумя ходами, а это означает, что ходы 2 и 4 тоже проигрывают. Перебираем дальше и делаем ход 5:

```
8 2   6 9   0 3   9 3
1 7   1 0   7 8   4 7
```

Отвечая 2, соперник опять выигрывает, т. к. оставшиеся три карточки можно взять ровно двумя ходами. Ход 6 сразу проигрывает из-за ответа 7. Делаем ход 8:

```
7 5   6 9   9 3
3 6   1 0   4 7
```

Легко убедиться, что все ходы в этой позиции проигрывают. Вот он выигрывающий ход: 8! Он здесь единственный такой.

Мы напишем программу, которая будет играть в такую игру с числом карточек от четырех до шестнадцати. Ввиду того, что могут быть проблемы с совместимостью некоторых графических карт с картой VGA по регистрам, я не буду использовать свою графическую библиотеку для режима VGA 640 × 480, 16 цветов, которую я в свое время написал для изготовления игр под DOS на Turbo Pascal. Напишем эту игру в стандартном текстовом режиме 80 × 25, 16 цветов без использования мыши. Отсутствие всего лишнего будет способствовать большей наглядности алгоритма. Программа также не будет ис-

пользовать какие-либо модули Turbo Pascal кроме встроенного модуля `System`, который присоединяется к каждой программе автоматически. Запись в видеопамять будем делать напрямую, используя адрес `$B800:0` начала видеопамяти в цветном текстовом режиме. Каждому символу на экране соответствует пара байтов в видеопамяти, в первом из которых записан код символа (число от `0` до `255`), а во втором — атрибуты этого символа, т. е. цвет его символа и цвет фона. Цвет символа кодируется младшей четверкой битов, а цвет фона — следующей тройкой, старший бит — бит мерцания; если он установлен, символ неприятно мерцает, меняя свой цвет на цвет фона и обратно. Кодировка цвета соответствует схеме RGB (red, green, blue — красный, зеленый, синий). Синему цвету соответствует младший, седьмой бит, зеленому — шестой, а красному — пятый. Четвертый бит — бит `I` интенсивности цвета. Цвета фона кодируются аналогично, но бит интенсивности для фона в стандартном режиме работы видеоконтроллера задействован под мерцание, поэтому цвета у фона темные. Строки на экране адресуются в видеопамяти последовательно слева направо и сверху вниз, за `n`-ой строкой сразу идет (`n+1`)-я. Пример: если записать слово `$0C30` по адресу `$B800:0` (`memw[$B800:0]:=$1C30`), то мы увидим на месте первого символа экрана ярко-красный нолик на синем фоне, потому что `$30` — ASCII-код нуля, а `$1C` (в двоичной системе это `11100`) задает бит `B` для цвета фона, бит интенсивности `I` и бит `R` для цвета символа. При очистке экрана каждому слову присваивается значение `$720` — пробел стандартного серого цвета на черном фоне, как было заведено в DOS.

Опишем теперь внешний вид и интерфейс игры. При старте программа попросит ввести число карточек, уровень сложности игры (глубина просчета вариантов программой) и кто начинает игру. После ввода на некоторое время возникает подсказка

```
F1 — новая игра
F2 — передача хода компьютеру
ESC — выход
```

и программа выводит заданное количество карточек. Для удобства восприятия каждая цифра имеет свой цвет. Также программа ведет запись ходов и после того, как она досчиталась до выигрыша, объявляет игроку, что тот проиграл. В случае его выигрыша она объявляет об этом и поздравляет игрока.

Сначала я поясню некоторые вызовы функций BIOS, которые будут нам нужны для установки курсора в нужную позицию, чтения нажатой клавиши из буфера клавиатуры и проверки этого буфера на наличие кодов клавиш, которые могли быть нажаты. Эти коротенькие процедуры написаны на встроенном ассемблере.

Для установки текстового курсора в позицию (x, y) (левый верхний угол экрана имеет координаты (0, 0), правый нижний — (79, 24)) надо вызвать функцию 2 прерывания 10H. Номер функции передается в регистре AH.

На входе:

AH 2

BH — номер страницы видеопамяти. Мы будем использовать лишь нулевую страницу.

DH — номер строки y.

DL — номер столбца x.

Текст самой процедуры вы видите в листинге 1.18.

Листинг 1.18. Установка текстового курсора

```
{ Установка курсора в позицию x,y (0-79,0-24) }
procedure SetCursorPos(x,y: byte); assembler;
      asm
      mov  ah,2
      xor  bh,bh
      mov  dl,x
      mov  dh,y
      int  10H
      end; {SetCursorPos}
```

Чтение нажатой клавиши в глобальную переменную key типа word представлено в листинге 1.19. BIOS при чтении клавиши возвращает слово, старший байт которого содержит скан-код нажатой клавиши, а младший — код символа.

Листинг 1.19. Чтение нажатой клавиши из буфера клавиатуры BIOS

```
{ Чтение клавиши в key. }
procedure GetKey; assembler;
      asm
      xor  ax,ax
      int  16H
      mov  key,ax
      end; {GetKey}
```

Иногда надо просто узнать, была ли нажата клавиша (и какая именно) без чтения ее из буфера клавиатуры (листинг 1.20). Ведь функция чтения клавиши приостановит работу программы на время, пока буфер будет пуст. Эта функция при наличии нажатой клавиши устанавливает флаг нуля `ZF` и возвращает код первой нажатой клавиши в регистре `AX`.

Листинг 1.20. Проверка нажатой клавиши в буфере клавиатуры BIOS

```
{ Проверка наличия символа в буфере клавиатуры. Если символ есть,
  то KeyPress равен символу и ZF = 0, иначе 0 и ZF = 1 }
function KeyPress : word; assembler;
      asm
      mov  ah,1
      int  16H
      mov  dx,ax
      mov  ax,0
      jz   @1
      mov  ax,dx
@1:   end; {KeyPress}
```

На основе последних двух функций написана процедура очистки буфера клавиатуры (листинг 1.21).

Листинг 1.21. Очистка буфера клавиатуры BIOS

```
{ Очистка буфера клавиатуры }
procedure ClearKeyBuff; assembler;
      asm
@1:   call KeyPress
      jz   @2        { нет символа }
      call GetKey    { читаем его }
      jmp  @1
@2:   mov  key,0
      end; {ClearKeyBuff}
```

Процедура `Delay55`, которую мы будем использовать в игре, нам уже знакома.

Очистка экрана перед началом новой игры будет выполняться процедурой `ClearScr`. Она во все позиции экранной памяти записывает слово `$720` (листинг 1.22).

Листинг 1.22. Очистка экрана (видеопамяти)

```
{ Очистка экрана }
procedure ClearScr;
var
 i: word;

begin
      for i:=0 to 1999 do MemW[$B800:i*2]:=$720;
end; {ClearScreen}
```

Еще нам надо будет выводить строку символов с заданным цветом символа и фона в данных экранных координатах `x` и `y` (листинг 1.23).

Листинг 1.23. Вывод строки в видеопамять

```
{ Вывод на экран строки str с координат (x,y) с цветом фона bkgcol
  и цветом символов fgcol }
procedure OutStr(x,y,bkgcol,fgrcol: word; str: string);
var
 i,l,offs,attrib: word;

begin
      offs:=(y*80+x)*2;
      attrib:=(bkgcol shl 4+fgrcol) shl 8;
      l:=Length(str);
      for i:=1 to l do
       begin
       MemW[$B800:offs]:=attrib+word(str[i]);
       Inc(offs,2);
       end;
end; {OutStr}
```

Процедура `OutStr` по координатам `x` и `y` вычисляет смещение `offs` в видеопамяти для первого символа этой строки, а по цвету символа и фона — байт атрибута (это старший байт слова `attrib`) для символов и затем в цикле выводит сформированные символы. Вот и все базовые функции этой игры.

А теперь обсудим кодировку данных для этой программы, которую назовем `OurNim`. Константы `BOARDX`, `BOARDY` задают экранные координаты левой верхней цифры первой карточки, а константы `HEADERX`, `HEADERY` — координаты заголовка для показа ходов человека и компьютера. Константы `RESULTX`, `RESULTY` обозначают место вывода фразы о выигрыше или проигрыше.

Константа `CHARSNUM=10` задает число символов, используемых в карточках, а сами символы представлены в массиве `CHARS: '0123456789'`. Массив `COLORS: (4,5,6,9,10,11,12,13,14,15)` задает десять цветов по одному цвету для каждой цифры; например, для нуля используется цвет символа `4` (темно-красный).

Назначение массива `DEPTHS: (3,99)` станет ясно позднее. Пока лишь скажу, что это глубины просчета вариантов для первого и второго уровня игры. На первом уровне на три хода вперед, на втором — до конца игры, т. к. игра не может продолжаться дольше десяти ходов. Это означает, что наша программа на втором уровне будет играть с максимальной силой, и если первый ход будет за компьютером и начальная позиция окажется выигрышной, то мы получим сообщение, что проиграли, еще не начав игру. Если вы считаете, что на первом уровне программа играет слишком сильно, замените в этом массиве `3` на `2`.

Массив `MARK: (0,0,0,0,$F,$1F,$3F,$7F,$FF,$1FF,$3FF,$7FF,$FFF,$1FFF,$3FFF,$7FFF,$FFFF)` в первых четырех элементах содержит заглушки, чтобы индексы начинались с нуля (это удобно при переносе программы на язык C). Остальные 12 чисел содержат битовое представление (отметки) для начального набора карточек (квадратов). Как вы помните, программа позволяет задавать от четырех до шестнадцати карточек для игры, а в процессе игры некоторые карточки будут изыматься (вычеркиваться) игроками. Вот этот массив и позволяет судить, какие карточки в игре, а какие уже вычеркнуты. Приведу пример: пусть в игре участвует шесть карточек. Тогда перед началом игры переменная `cardnum` получит значение `6`, а переменной `mark0` присвоится число `MARK[cardnum]` или `$3F`. В двоичной системе `$3F` это `111111`, т. е. как раз шесть единичных битов. Младший бит означает наличие самой левой карточки, следующий за ним — карточки справа от нее и т. д. Если кто-то (игрок или компьютер) возьмет, к примеру, первую и вторую слева карточки, то переменная `mark0` станет равна `111100` или `$3C`. Таким образом, программа всегда знает: какие карточки в игре, и когда игра закончена.

Сами карточки кодируются массивом `squares: array[0..15,0..3] of byte`. Например, левая цифра в верхнем ряду в первой карточке кодируется элементом `squares[0,0]`, правая цифра во втором ряду — элементом `squares[0,3]`. Цифры в этом массиве кодируются в двоичном виде, а не ASCII-кодами десятичных цифр.

Переменные movenum и strnum содержат номер хода и относительный номер строки, в которой программа должна вывести очередной ход.

Переменная depth0 содержит глубину просчета вариантов (3 или 99).

Константы с кодами управляющих клавиш ESC=$11B, F1=$3B00 и F2=$3C00 служат для проверки нажатия на эти клавиши. Эти значения возвращает BIOS, и их можно посмотреть с помощью функции GetKey.

Рассмотрим остальные вспомогательные процедуры.

procedure ShowResult(num: word) при num=0 выводит сообщение о проигрыше игрока, иначе — о его выигрыше. Рассматривать ее текст нет необходимости.

Еще нам понадобится функция, сообщающая, имеется ли в позиции, кодируемой битами слова currmark, карточка с данной двоичной цифрой digit. Вы видите ее в листинге 1.24.

Листинг 1.24. Функция FindSquareByDigit

```
{ Есть ли квадрат, содержащий цифру digit? }
function FindSquareByDigit(currmark: word; digit: byte): boolean;
var  i,j: byte;

begin
      FindSquareByDigit:=TRUE;
      for i:=0 to cardnum-1 do
       if currmark and (1 shl i) > 0 then
        for j:=0 to 3 do
         if squares[i,j] = digit then Exit;
      FindSquareByDigit:=FALSE;
end; {FindSquareByDigit}
```

На входе эта функция получает слово currmark — текущую отметку наличия карточек (см. описание массива MARK) и искомую цифру digit. В цикле по i по числу карточек cardnum перебираются биты в слове currmark, начиная с младшего, и для каждого единичного бита в цикле по j в i-ой карточке ищется цифра digit. Как только эта цифра найдена, функция заканчивает работу и возвращает TRUE. Если digit в действующих карточках не найдена, возвращается FALSE.

Процедура PaintSquare будет выводить на экран квадрат по его номеру i (листинг 1.25).

Листинг 1.25. Процедура `PaintSquare`

```
{ Вывод квадрата по его номеру 0..15. Удаленные квадраты
  выводятся серым цветом (7), остальные - в соответствии с
  массивом COLORS }
procedure PaintSquare(i: integer);
var
  s,i1,j1,x,y: integer;
  color1: byte;

begin
      for i1:=0 to 1 do
       for j1:=0 to 1 do
        begin
        s:=squares[i,i1*2+j1];
        x:=BOARDX+i*3+j1;
        y:=BOARDY+i1;
        if mark0 and (1 shl i) = 0 then color1:=7
         else color1:=COLORS[s];
        OutStr(x,y,0,color1,CHARS[s]);
        end;
end; {PaintSquare}
```

Еще нам понадобится функция, которая убирает все карточки (квадраты), содержащие заданную цифру. Она будет возвращать число удаленных карточек. Так как эта функция будет вызываться для выполнения хода игрока и компьютера в основной программе, а также внутри функции, считающей варианты, то в ней имеется параметр `show`, который указывает, отображать ли этот ход на экране. В листинге 1.26 представлен текст этой функции.

Листинг 1.26. Функция `RemoveSquare`

```
{ Убираем квадраты в позиции currmark с цифрой digit. Возвращаем
  количество удаленных квадратов. Если show > 0, то стираем
  их на экране и показываем ход }
function RemoveSquare(var currmark: word; digit,show: byte): byte;
label 10;
```

```
var
  sqremoved,i,j: byte;

begin
      { Число удаляемых квадратов }
      sqremoved:=0;
      { Цикл по всем квадратам }
      for i:=0 to cardnum-1 do
       begin
       { i-й квадрат в игре? }
       if currmark and (1 shl i) > 0 then
        begin
        { Цикл по 4-м цифрам в i-ом квадрате }
        for j:=0 to 3 do
         { Нашли цифру digit в i-ом квадрате? }
         if squares[i,j] = digit then
          { Да, удаляем ее }
          begin
          Inc(sqremoved);
          Dec(currmark,1 shl i);
          if show > 0 then
           { Отмечаем убранный квадрат и переходим на продолжение
             цикла по всем квадратам }
           begin
           PaintSquare(i);
           goto 10;
           end;
          end;
        end;
10:    end;
      if (show > 0) and (sqremoved > 0) then
       { Показываем ход }
       begin
       OutStr(HEADERX+4+movenum and 1*10,HEADERY+strnum,0,7,
        CHARS[digit]);
       { Наращиваем номер строки для вывода хода }
```

```
        if Odd(movenum) then Inc(strnum);
        { Увеличиваем число сделанных ходов }
        Inc(movenum);
        end;
      RemoveSquare:=sqremoved;
end; {RemoveSquare}
```

Позиция в нашей игре задается одним словом `currmark` — битовым массивом имеющихся карточек, `digit` — цифра, все карточки с которой надо удалить. Обратите внимание, что переменная `currmark` передается по адресу, как `var`-параметр. Это означает, что вызывающий код получит в переданной переменной измененную этим ходом позицию, из которой удалятся все биты, соответствующие карточкам, содержащим цифру `digit`. При этом сам набор карточек на протяжении одной игры не меняется и хранится в массиве `squares` по четыре цифры в одной карточке.

А сейчас самая трудная часть программирования — написание маленькой функции, которая будет давать ответ, выигрывает ли данный ход в позиции или нет. Вдумчивый читатель, вероятно, уже догадался, что эта функция будет рекурсивной. На входе она должна получать два параметра — текущую позицию `currmark` и цифру `digit`, все карточки с которой надо удалить. На выходе эта функция будет возвращать признак, выигрывает ли заданный ход `digit` в данной позиции `currmark` или нет. Но сначала мы напишем вариант этой функции `IsWin`, который играет в полную силу и просчитывает варианты до выяснения поставленного вопроса. Удивительно, но эта функция, в которой сосредоточен искусственный интеллект, обыгрывающий любого человека, состоит всего из восьми операторов!

Мы будем предполагать, что перед вызовом этой функции в игре имеется хотя бы одна карточка. Вот алгоритм этой замечательной функции:

1. Выполняем заданный ход, делая вызов `RemoveSquare(currmark,digit,0)`. Если после совершения этого хода оказывается, что не осталось квадратов (`currmark = 0`), то заданный ход выигрывает, и мы выходим из функции и возвращаем `TRUE`.
2. Так как еще есть карточки в игре, то надо перебрать в цикле все ответные ходы соперника, вызывая саму себя. Если при каком-то вызове `IsWin` вернет `TRUE`, то это означает, что соперник имеет выигрыш в ответ на наш ход `digit`. Поэтому мы должны выйти из функции с возвратом `FALSE`. Если цикл окончен, то это означает, что у соперника нет ответного выигрывающего хода, а это значит, что наш ход `digit` выигрывает, и мы возвращаем `TRUE`.

В листинге 1.27 приведен текст этой функции.

Листинг 1.27. Функция IsWin

```
{ Выясняем, можно ли выиграть в позиции currmark, если вычеркнуть
  все квадраты с символом digit? Если можно, возвращаем TRUE,
  если нельзя, возвращаем FALSE }
function IsWin(currmark: word; digit: byte): boolean;
LABEL  WON,LOST;
var
  i: word;

begin
      { Делаем ход digit в текущей позиции currmark }
      RemoveSquare(currmark,digit,0);
      { Если после этого хода не осталось квадратов, то
        этот ход выигрышный }
      if currmark = 0 then goto WON;
      { Перебираем все ответные ходы соперника }
      for i:=0 to CHARSNUM-1 do
       { Есть цифра i на каком-то квадрате? }
       if FindSquareByDigit(currmark,i) then
        { Да, убираем все квадраты, содержащие цифру i }
        if IsWin(currmark,i) then goto LOST;
WON:  IsWin:=TRUE;
      Exit;
LOST: IsWin:=FALSE;
end; {IsWin}
```

Вы можете сыграть с программой, которая считает варианты с помощью этой функции, это файлы ournim1.exe и ournim1.pas. Здесь компьютер своего выигрыша не упустит.

Но ведь это не слишком интересно — играть с таким соперником. Если он начинает игру, то в большинстве случаев он будет объявлять, что вы проиграли, сделав первый ход. Поэтому мы смоделируем игру человека и введем два уровня игры — первый и второй. На первом уровне программа будет считать на три хода вперед, а на втором — аж на `99`, что в нашей игре будет означать бесконечность и игру в полную силу. Помните массив `DEPTHS: array[0..1] of byte = (3,99)`? Вот для этого он нам и понадобится. Вы можете задать промежуточные глубины счета вариантов. Обычно это делает-

ся в играх, где партия состоит из многих ходов, а у нас 3—4 хода — и конец игры.

Если на первоначальный запрос программы об уровне своей игры человек выберет уровень `1`, то глобальная переменная `depth0` получит значение `3` (иначе — `99`). Кроме того, будет использоваться глобальная переменная `DEPTH`, которая вне функции `IsWin` будет равна нулю. При входе в `IsWin` первым делом `DEPTH` будет увеличиваться на единицу, а перед выходом из `IsWin DEPTH` будет уменьшаться на `1`.

Теперь `IsWin` у нас будет возвращать уже не логическое значение, а число, потому что появляется третий тип результата ее работы: программа не досчитала до конца игры и не смогла определить, выигрывает заданный ход или нет. Пусть в этом случае `IsWin` возвращает `2`. В случае выигрыша пусть возвращается `1`, а при проигрыше — `0`.

Итак, `IsWin` нарастила переменную `DEPTH`. Если теперь `DEPTH > depth0`, то надо выйти и вернуть `2`, т. к. лимит на глубину счета вариантов исчерпан. Иначе, как и в предыдущем случае, делаем заданный ход, и если после него не осталось карточек, то выходим с возвратом единицы. Дальше нам понадобится новая локальная переменная `res` (результат хода соперника). Вначале присваиваем `res` ноль, что предварительно означает проигрыш соперника. Затем аналогично перебираем все ответные ходы соперника в цикле. Если здесь `IsWin` вернет единицу, то мы делаем выход и возвращаем ноль. Если `IsWin` после какого-то ответного хода возвращает `2`, то мы запоминаем этот факт в переменной `res` (`res:=2`). По окончании работы этого цикла будет уже два исхода: если `res` окажется равным нулю, то все ходы соперника проигрывают, и мы возвращаем `1`, а если `res` оказывается равным двум, то у соперника был ход, при котором программа пока еще не может досчитаться до результата игры. В этом случае мы возвращаем `2`. Вот и весь человеческий алгоритм игры. Если читатель понял этот в принципе нехитрый алгоритм, до которого я когда-то давно доходил своим умом, то ему не составит особого труда запрограммировать аналогичные логические игры, такие как различные варианты "Нима", "Крестиков-ноликов" и других, где партия длится не такое большое число ходов, как в шахматах или го. В этих случаях надо применять также другие стратегии, ведь время перебора вариантов очень быстро растет с ростом глубины перебора.

Остается только рассмотреть основной текст нашей программы, вернее, алгоритм ее работы.

1. Устанавливем текстовый режим 80 × 25, 16 цветов. Это может понадобиться, потому что иногда у пользователя, работающего в консольной программе типа Far или Dos Navigator, может быть задано нестандартное число символов по горизонтали и вертикали. Запрашиваем параметры программы (уровень игры и количество карточек), проверяем их, прячем курсор и выводим на время подсказку по клавишам управления.

2. **Метка NEW.** Здесь начинается цикл новой игры. Вызываем процедуру `Randomize`, очищаем экран и выводим заголовок для показа сделанных ходов. В переменной `mark0` создаем битовое представление начальной позиции в соответствии с числом карточек `numcadrs`, введенным игроком. Счетчику сделанных ходов `movenum` присваиваем ноль, счетчику строк `strnum` — единицу переменной `depth0` (это глубина счета вариантов) присваиваем `3` или `99` в зависимости от того, какой уровень игры задал человек. Далее мы заполняем случайными цифрами массив `squares` и выводим сформированные карточки на экран.
3. **Метка PLAYERMOVE.** Ждем и читаем нажатую клавишу (<0>, <1>, ..., <9>). Если нажата клавиша <Esc> — выходим из программы, если клавиша <F1>, то переходим к метке **NEW**. Если нажата клавиша <F2> и `movenum=0`, то наращиваем `movenum` на `1` и передаем первый ход компьютеру, переходя к метке **COMPMOVE**. Иначе проверяем, нажата ли клавиша из имеющегося набора игровых символов (у нас это десятичные цифры) и есть ли карточка с заданной цифрой. Если ход некорректный, то переходим на метку **PLAYERMOVE**. Иначе убираем все карточки с введенной игроком цифрой. Если после этого хода `mark0=0`, то объявляем о выигрыше игрока, ждем нажатия клавиши, выходим из программы, если нажата клавиша <Esc>, иначе переходим к метке **NEW**.
4. **Метка COMPMOVE.** Здесь ход компьютера. Делаем небольшую задержку времени, чтобы программа после хода человека не отвечала мгновенно. Выполняем цикл перебора всех ходов. Если среди них попадается выигрышный, то выполняем этот ход, показываем его, а также фразу о проигрыше игрока. Если `mark0=0`, то конец игре, и мы читаем клавишу и выходим из программы или переходим на метку **PLAYERMOVE** в зависимости от нажатой клавиши (<Esc> или нет). Если цикл перебора всех ходов закончился, это означает, что программа не нашла выигрыша. Как в этом случае побольше насолить игроку? Ну, конечно, надо убрать минимум карточек, чтобы ему труднее было найти выигрыш. Для этого опять перебираем все ходы и для каждого подсчитываем, сколько карточек при этом снимается. Делаем минимальный ход и переходим к метке **PLAYERMOVE**.

В листинге 1.28 представлен полный текст программы ournim.pas, которая, конечно, также присутствует на компакт-диске.

Листинг 1.28. Игровая программа ournim.pas

```
{$A-,B-,G+,I-,R-,S-}
{$M 16384,0,655360}
program OurNim;
```

```
label
 NEW,PLAYERMOVE,VALIDATEMOVE,COMPMOVE;
const
 BOARDX = 2;
 BOARDY = 10;
 HEADERX = 55;
 HEADERY = 10;
 RESULTX = 55;
 RESULTY = 5;
 CHARSNUM = 10;
 DEPTHS: array[0..1] of byte = (3,99);
 COLORS: array[0..CHARSNUM-1] of byte =
  (4,5,6,9,10,11,12,13,14,15);
 CHARS: array[0..CHARSNUM-1] of char = '0123456789';
 MARK: array[0..16] of word = (0,0,0,0,
  $F,$1F,$3F,$7F,$FF,$1FF,$3FF,$7FF,$FFF,$1FFF,$3FFF,$7FFF,$FFFF);
 DEPTH: byte = 0;
 { Коды управляющих клавиш }
 ESC = $11B;
 F1 = $3B00;
 F2 = $3C00;

var
 i,j,i1,j1,x,y,k: integer;
 depth0,movenum,strnum,cardnum,depthind: byte;
 mark0,mark1,key: word;
 squares: array[0..15,0..3] of byte;

{ Установка курсора в позицию (x,y) - (0-79, 0-24) }
procedure SetCursorPos(x,y: byte); assembler;
      asm
      mov  ah,2
      xor  bh,bh
      mov  dl,x
      mov  dh,y
      int  10H
      end; {SetCursorPos}
```

```
{ Задержка времени интервалами по 55 мс.
  Число интервалов задается в ms55 }
procedure Delay55(ms55 : longint);
VAR
 biosct : longint absolute $40:$6C;
 ct : longint;
begin
      ct:=biosct+ms55;
      repeat until ct <= biosct;
end; {Delay55}

{ Чтение клавиши в key. }
procedure GetKey; assembler;
      asm
      xor  ax,ax
      int  16H
      mov  key,ax
      end; {GetKey}

{ Проверка наличия символа в буфере клавиатуры. Если символ есть,
  то KeyPress равен символу и ZF = 0, иначе 0 и ZF = 1 }
function KeyPress : word; assembler;
      asm
      mov  ah,1
      int  16H
      mov  dx,ax
      mov  ax,0
      jz   @1
      mov  ax,dx
@1:   end; {KeyPress}

{ Очистка буфера клавиатуры }
procedure ClearKeyBuff; assembler;
      asm
@1:   call KeyPress
      jz   @2        { нет символа }
```

```
      call GetKey    { читаем его }
      jmp  @1
@2:   mov  key,0
      end; {ClearKeyBuff}

{ Очистка экрана }
procedure ClearScr;
var
 i: word;

begin
      for i:=0 to 1999 do MemW[$B800:i*2]:=$720;
end; {ClearScreen}

{ Вывод на экран строки str с координат (x,y) с цветом фона bkgcol
  и цветом символов fgcol }
procedure OutStr(x,y,bkgcol,fgrcol: word; str: string);
var
 i,l,offs,attrib: word;

begin
      offs:=(y*80+x)*2;
      attrib:=(bkgcol shl 4+fgrcol) shl 8;
      l:=Length(str);
      for i:=1 to l do
       begin
       MemW[$B800:offs]:=attrib+word(str[i]);
       Inc(offs,2);
       end;
end; {OutStr}

{ Сообщение о выигрыше/проигрыше при num=0/1 }
procedure ShowResult(num: word);
begin
      OutStr(RESULTX,RESULTY,0,7,'                      ');
      if num = 0 then
```

```
       OutStr(RESULTX,RESULTY,0,12,'Увы, Вы проиграли') else
        OutStr(RESULTX,RESULTY,0,10,'Ух ты! Вы выиграли!')
end; {ShowResult}

{ Есть ли квадрат, содержащий цифру digit? }
function FindSquareByDigit(currmark: word; digit: byte): boolean;
var  i,j: byte;

begin
      FindSquareByDigit:=TRUE;
      for i:=0 to cardnum-1 do
       if currmark and (1 shl i) > 0 then
        for j:=0 to 3 do
         if squares[i,j] = digit then Exit;
      FindSquareByDigit:=FALSE;
end; {FindSquareByDigit}

{ Вывод квадрата по его номеру 0..15. Удаленные квадраты
  выводятся серым цветом (7), остальные - в соответствии с
  массивом COLORS }
procedure PaintSquare(i: integer);
var
  s,i1,j1,x,y: integer;
  color1: byte;

begin
      for i1:=0 to 1 do
       for j1:=0 to 1 do
        begin
        s:=squares[i,i1*2+j1];
        x:=BOARDX+i*3+j1;
        y:=BOARDY+i1;
        if mark0 and (1 shl i) = 0 then color1:=7
         else color1:=COLORS[s];
        OutStr(x,y,0,color1,CHARS[s]);
        end;
end; {PaintSquare}
```

```
{ Убираем квадраты в позиции currmark с цифрой digit. Возвращаем
  количество удаленных квадратов. Если show > 0, то стираем
  их на экране и показываем ход }
function RemoveSquare(var currmark: word; digit,show: byte): byte;
label 10;
var
  sqremoved,i,j: byte;

begin
      { Число удаляемых квадратов }
      sqremoved:=0;
      { Цикл по всем квадратам }
      for i:=0 to cardnum-1 do
       begin
       { i-й квадрат в игре? }
       if currmark and (1 shl i) > 0 then
        begin
        { Цикл по 4-м цифрам в i-ом квадрате }
        for j:=0 to 3 do
         { Нашли цифру digit в i-ом квадрате? }
         if squares[i,j] = digit then
          { Да, удаляем ее }
          begin
          Inc(sqremoved);
          Dec(currmark,1 shl i);
          if show > 0 then
           { Отмечаем убранный квадрат и переходим на продолжение
             цикла по всем квадратам }
           begin
           PaintSquare(i);
           goto 10;
           end;
          end;
        end;
10:    end;
      if (show > 0) and (sqremoved > 0) then
```

```
        { Показываем ход }
        begin
        OutStr(HEADERX+4+movenum and 1*10,HEADERY+strnum,0,7,
         CHARS[digit]);
        { Наращиваем номер строки для вывода хода }
        if Odd(movenum) then Inc(strnum);
        { Увеличиваем число сделанных ходов }
        Inc(movenum);
        end;
       RemoveSquare:=sqremoved;
end; {RemoveSquare}

{ Выясняем, можно ли выиграть в заданной позиции?
  Если можно, возвращаем 1,
  если нельзя, возвращаем 0,
  если не просчитали позицию до конца, возвращаем 2 }
function IsWin(currmark: word; digit: byte): word;
LABEL  100,WON,LOST,NOTKNOWN;
var
 i,res,tmpres: word;

begin
       { Наращиваем глубину счета вариантов }
       Inc(DEPTH);
       { Если она больше, чем задано игроком, то дальше
         варианты не рассматриваем }
       if DEPTH > depth0 then
        begin
        IsWin:=2;
        goto 100;
        end;
       { Делаем ход в текущей позиции currmark }
       RemoveSquare(currmark,digit,0);
       { Если после этого хода не осталось квадратов, то
         этот ход выигрышный }
       if currmark = 0 then goto WON;
```

```
      { Предварительно засчитываем сопернику проигрыш }
      res:=0;
      { Перебираем все ответные ходы соперника }
      for i:=0 to CHARSNUM-1 do
       begin
       { Есть цифра i на каком-то квадрате? }
       if FindSquareByDigit(currmark,i) then
        begin
        { Да, убираем все квадраты, содержащие цифру i }
        tmpres:=IsWin(currmark,i);
        { Если соперник выиграл, то заданная позиция
          была проигрышной }
        if tmpres = 1 then goto LOST;
        { Если соперник на этом ходе не досчитался
          до выигрыша, запомним это в переменной res }
        if tmpres = 2 then res:=2;
        end;
       end;
      { Соперник не досчитался до выигрыша на каком-то ходе.
        Значит, результат игры неизвестен }
      if res = 2 then goto NOTKNOWN;
      { Все ответные ходы соперника проиграли, значит, данная
        позиция выигрышна }
WON:  IsWin:=1;
      goto 100;
LOST: IsWin:=0;
      goto 100;
NOTKNOWN:
      IsWin:=2;
      { Возвращаемся в предыдущую позицию }
100:  Dec(DEPTH);
end; {IsWin}

BEGIN
      { Устанавливем текстовый режим 80x25, 16 цветов }
      asm
```

```
      mov  ax,3
      int  10H
      end;
      { Запрашиваем параметры игры }
      SetCursorPos(0,24);
      Writeln('Введите через пробел количество карточек (4-16),');
      Write('и уровень сложности моей игры (1 или 2): ');
      ClearkeyBuff;
      Readln(cardnum,depthind);
      if (cardnum < 4) or (cardnum > 16) or
       (depthind < 1) or (depthind > 2) then Exit;
      Dec(depthind);
      { Прячем курсор, устанавливая его на 25-ю строку }
      SetCursorPos(0,25);
      { Показываем на время подсказку }
      ClearScr;
      OutStr(26,9,0,7,'F1 - новая игра');
      OutStr(26,10,0,7,'F2 - передача хода компьютеру');
      OutStr(26,11,0,7,'ESC - выход');
      Delay55(30);
      ClearkeyBuff;
      { Цикл новой игры }
NEW:  Randomize;
      ClearScr;
      OutStr(HEADERX,HEADERY,0,7,'Ваши ходы: Мои ходы:');
      { Битовое представление квадратов (позиция игры) }
      mark0:=MARK[cardnum];
      { Счетчик ходов }
      movenum:=0;
      { Счетчик строк для вывода ходов }
      strnum:=1;
      { Глубина счета вариантов }
      depth0:=DEPTHS[depthind];
      { Создаем и выводим на экран квадраты }
      for i:=0 to cardnum-1 do
```

```
       begin
       { Цикл создания i-го квадрата }
       for j:=0 to 3 do
        begin
        k:=Random(CHARSNUM);
        { Уменьшаем вероятность появления одинаковых
          символов на одном квадрате }
        for i1:=1 to j do
         if squares[i,i1-1] = k then k:=Random(CHARSNUM);
        squares[i,j]:=k;
        end;
       PaintSquare(i);
       end;
      { Чтение хода игрока }
      repeat
PLAYERMOVE:
       GetKey;
       if key = ESC then Exit;
       if key = F1 then goto NEW;
       if (key = F2) and (movenum = 0) then
        { Передача первого хода компьютеру }
        begin
        Inc(movenum);
        goto COMPMOVE;
        end;
       { Преобразуем key в код введенного символа }
       key:=word(UpCase(chr(key and $FF)));
       { Проверяем на соответствие набору символов }
       for i:=0 to CHARSNUM-1 do
        if key = word(CHARS[i]) then goto VALIDATEMOVE;
      until FALSE;
      { Получаем в key номер выбранного символа (0-9) }
VALIDATEMOVE:
      key:=i;
      { Проверка, имеется ли такой ход }
```

```
      if (mark0 > 0) and (RemoveSquare(mark0,key,1) = 0) then
       goto PLAYERMOVE;
      if mark0 = 0 then
       begin
       ShowResult(1);
       GetKey;
       if key = ESC then Exit;
       goto NEW;
       end;
COMPMOVE:
      Delay55(8);
      for i:=0 to CHARSNUM-1 do
       if FindSquareByDigit(mark0,i) and (IsWin(mark0,i) = 1) then
        begin
        ShowResult(0);
        RemoveSquare(mark0,i,1);
        if mark0 = 0 then
         begin
         GetKey;
         if key = ESC then Exit;
         goto NEW;
         end;
        goto PLAYERMOVE;
        end;
      { Нет выигрыша. Убираем минимум квадратов, чтобы игроку
        труднее было считать варианты }
      k:=100;
      for i:=0 to CHARSNUM-1 do
       begin
       mark1:=mark0;
       j1:=RemoveSquare(mark1,i,0);
       if (j1 > 0) and (j1 < k) then
        begin
        k:=j1;
        j:=i;
        end;
       end;
```

```
      { j - цифра, написанная на минимальном числе квадратов }
      RemoveSquare(mark0,j,1);
      goto PLAYERMOVE;
END.
```

А теперь — очередное задание.

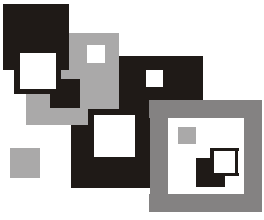
Домашнее задание

1. Напишите игровую программу для игры "Цепочка". Имеется $n > 14$ карточек, расположенных в виде цепочки (ряда). При первом ходе переворачивается одна любая карточка. При последующих ходах — одна или две соседние. Выигрывает тот, кто оставит сопернику одну неперевернутую карточку.
2. Из спичек сложена квадратная решетка 4×4 квадрата. Игроки поочередно забирают по одной спичке. Выигрывает тот, после чьего хода не останется ни одного прямоугольника, сложенного из спичек. Запрограммируйте эту игру.
3. Напишите программу для такой игры: соперники по очереди ставят на шахматную доску $n \times n$ по своему выбору ферзя или коня так, чтобы ни одна фигура не угрожала никакой другой. Проигрывает тот, кто не может сделать ход.

Замечание

Исходные файлы для игры "Ним" размещены на компакт-диске исключительно с целью обучения и не могут быть использованы в любых иных целях. Вы не имеете права публиковать где бы то ни было и в любой форме текст этой программы. Также вы не имеете права использовать принадлежащую мне как автору идею этой игры.

Глава 2



Практика на форме Delphi. Игра Calculator

- Создаем оригинальную логическую игру стандартными компонентами Delphi

В этой главе мы создадим простую логическую игру с помощью компонентов из палитры Delphi. Вид у нее, конечно, будет не ахти какой завлекательный, но после того, как программа отлажена и работает, ее можно украшать до бесконечности, это не сложно. Для нас же здесь важно, чтобы программа работала правильно и была доступна для понимания и изучения. Кроме того, реальные программы имеют очень большие размеры листинга, который один потянет на книжку.

2.1. Постановка задачи

С появлением микрокалькуляторов появились также игры для них. Я также решил внести вклад в это дело, тем более, что мне нравятся миниатюрные игры с простыми правилами и непростой теорией; и придумал игру с виртуальным калькулятором. Этот калькулятор можно назвать бухгалтерским, потому что он выполняет только вычитание. Он имеет девять кнопок в три строки и три столбца с цифрами от 1 до 9 и индикатор. Перед каждой игрой цифры на кнопках перемешиваются случайным образом, а на индикаторе появляется случайное число в диапазоне где-то от 30 до 35. Правила игры таковы: первый игрок может нажать любую из девяти кнопок. После каждого нажатия число на индикаторе уменьшается на цифру, написанную на этой кнопке. Второй игрок может нажать любую из трех кнопок в том столбце, в котором делал предыдущий ход первый игрок, а первый игрок в свои последующие ходы может нажимать любую из трех кнопок в той строке, в которой второй игрок нажимал кнопку последний раз. Проигрывает

тот игрок, после чьего хода на индикаторе появится отрицательное, т. е. меньшее нуля, число.

Рассмотрим пример поиска выигрывающего хода в простой позиции. Пусть на индикаторе написано число 15:

```
 15

1 5 2

7 8 9

6 3 4
```

Перебор ходов начнем с больших чисел, т. к. при этом быстрее становится понятен результат игры. Итак, нажимаем кнопку с цифрой 9. На индикаторе остается число 6, а второй игрок теперь имеет ходы 2, 9 и 4:

```
  6

1 5 2

7 8 9

6 3 4
```

На девятку ему нажимать нет смысла ввиду проигрыша. А если он нажмет кнопку с цифрой 4, то на индикаторе останется число 2, а у нас — ходы 6, 3 и 4. После каждого из них на индикаторе окажется отрицательное число, поэтому ход 9 проигрывает.

Попробуем ход 8. После этого хода позиция станет такой:

```
  7

1 5 2

7 8 9

6 3 4
```

Ход 8 не годится, смотрим ход 5. На индикаторе остается 2, и мы в ответ ходим 2 и выигрываем. А если оппонент сходит 3, то на индикаторе установится число 4 и мы конечно же сразу сходим 4 и опять выиграем. Вот он, выигрывающий ход: 8! Проверьте, что это единственный выигрывающий ход.

2.2. Создание основной формы игры

После этой короткой теоретической подготовки запускаем имеющуюся у нас версию Delphi и видим заготовку для будущей игры в виде формы, текста для Unit1.pas окно **Object Inspektor** (Инспектор объектов) и быть может вверху над ним **Object Treeview** (Дерево объектов). Для клавиш мы будем

использовать компонент **SpeedButton** (кнопка для инструментальных панелей), расположенный на вкладке **Additional** (Дополнительная) палитры компонентов, для индикатора — компонент **Panel** (Панель) на вкладке **Standard** (Стандартная). Еще нам будут нужны кнопки **New Game** для начала новой игры, **Comp. begins** для передачи первого хода компьютеру, **Level** для задания уровня сложности игры компьютера, **Help** для вывода окна с автором и правилами игры и **Exit** для выхода из игры. Уровень игры будем отображать с помощью компонента **Label** (Метка) той же палитры **Standard**. Все эти кнопки будут иметь тип **Button** (Кнопка) стандартной палитры компонентов. Для отображения ходов удобно использовать элемент **Memo** (Примечание) из той же палитры, для вывода результата игры — опять **Label**.

Итак, где-то в памяти у нас уже сидит Delphi и ждет, когда мы начнем утюжить коврик мышью и нажимать разные кнопки. Оставим имя `Form1` для формы, но ее заголовок переименуем в `Calculator`. Этот текст введите в поле `Caption` инспектора объектов. Свойство **Position** установим в `poDefault`, позволяя Windows выбирать место на экране для окна нашей игры. В меню **File** выберите команду **Save All** и запомните оба файла проекта в надежном месте, попутно переименовав файл проекта в calc.dpr.

Для большего удобства можете уменьшить шаг решетки на форме с восьми до четырех точек. Для этого откройте окно **Tools | Environment Options** (Сервис | Опции окружения) и на вкладке **Designer** (Дизайнер) установите **Grid size** (Размер решетки) 4 и 4.

Начнем с панели. Щелкните на кнопке **Panel** в панели компонентов и затем где-либо на форме. В результате на форме окажется панель. В окне инспектора объектов очистите поле **Caption** (Заголовок) (мы будем заполнять его программно), в поле **Name** (Имя) пусть остается `Panel1`, отступ слева **Left** = = `24`, отступ сверху **Top** = `24`, ширина **Width** = `141`, высота **Height** = `41`. Далее раскройте меню **Font** (Шрифт) в инспекторе объектов, и в нем — подменю **Style** (Стиль). Установите стиль **fsBold** (Стиль шрифта жирный) в `True` (для этого можно сделать двойной щелчок на этом выпадающем списке), в поле **Size** (Размер шрифта) введите `28`. Чтобы посмотреть на результат, введите в поле **Caption** несколько цифр, и они тут же отобразятся на панели.

Выберите компонент **Memo** на этой же стандартной панели инструментов и щелкните мышью на форме справа от панели. В инспекторе объектов щелкните на поле **Lines** и на кнопке с многоточием справа от надписи **(TStrings)**. В открывшемся окне **String List Editor** (Редактор списка строк) вместо строки `Memo1` введите строку `Your moves: My moves:` (или свой вариант). Выберите моноширный шрифт, например, Courier New, размер шрифта 12. Выберите ширину компонента **Memo1** в `218` точек (лишь бы эта строка располагалась на одной линии), в поле для высоты введите `218`, отступ слева `180`, сверху — `24`. Вообще для отмены переноса длинных строк можно уста-

новить свойство **WordWrap** в `False`. Чтобы игрок не мог редактировать текст в этом компоненте, установите параметр **ReadOnly** (Только чтение) в `True`.

Выберите в палитре компонент **Button** и сделайте справа от поля **Memo1** пять кнопок с именами (**Name**) `btNew`, `btCompBegins`, `btLevel`, `btHelp`, `btExit` и надписями (**Caption**), соответственно, `New Game`, `Comp. begins`, `Level`, `Help` и `Exit`. Отступ слева для всех кнопок задайте в `411` точек, а по высоте расположите их равномерно с шагом `48` точек. Справа от кнопки `btLevel` создайте метку (это компонент с именем **Label**), шрифт выберите жирный размером `12`. У меня отступ слева получился `496`, а сверху — `122`. Для передвижения выделенного компонента на одну точку используйте клавиши со стрелками и нажатой клавишей <Ctrl>; с нажатой клавишей <Shift> можно плавно менять размер компонента. В свойство **Caption** этого компонента мы будем вписывать уровень игры `1`, `2`, ..., `9` при щелчке на кнопке **Level**. Имя этой метке дадим `laLevel`.

Теперь перейдем на вкладку **Additional** и выберем компонент **SpeedButton**. Щелкните слева под панелью и поставьте эту кнопку. Задайте ей квадратный размер в `45` точек, а шрифт жирный размером `28`. Затем с помощью комбинации клавиш <Ctrl>+<C> или через контекстное или общее меню **Edit** (Правка) скопируйте эту кнопку в буфер обмена и потом нажатием комбинации клавиш <Ctrl>+<V> или опять через меню создайте еще восемь таких же кнопок. Расположите их под панелью в три строки и три столбца с шагом в `48` точек. Дайте им последовательные имена `k00`, `k01`, `k02`; `k10`, `k11`, `k12`; `k20`, `k21`, `k22`, двигаясь от левой верхней кнопки по строкам. В имени у нас первая цифра означает номер столбца кнопки, а вторая — номер ее строки. В поле **Caption** этих кнопок мы программно будем вписывать цифры от `1` до `9`.

Уменьшим теперь размер формы для нашей игры, потянув за ее правый нижний угол или с помощью инспектора объектов. Допустим, зададим ей размер `525` на `290`. А чтобы игрок не мог менять ее размер и распахивать на весь экран, **BorderStyle** (Стиль рамки формы) установите в `bsSingle`, а в **BorderIcons** (Кнопки заголовка) установите **biMaximize** в `FALSE`. На рис. 2.1 вы видите, что получилось у меня на дизайнерской форме. Сам калькулятор притулился где-то сбоку. Вы можете расположить эти компоненты на свой вкус, это не влияет на работоспособность программы.

Впрочем, строку `Your moves: My moves:` вводить в компонент `Memo1` не обязательно, она будет возникать в процедуре `NewGame`.

Если нажать клавишу <F9>, то заготовка для будущей игры заработает и покажет эту форму. Кроме показа себя наша форма еще умеет сворачиваться и прекращать свое существование. А ну-ка, какой размер у exe-файла? 400 Кбайт? Впечатляет, особенно в сравнении с нашими программами для DOS. Ничего, сейчас размеры ОЗУ и дисков большие, выдержат. Зато мы освобо-

ждены от рутинной работы по созданию и программированию поведения элементов управления.

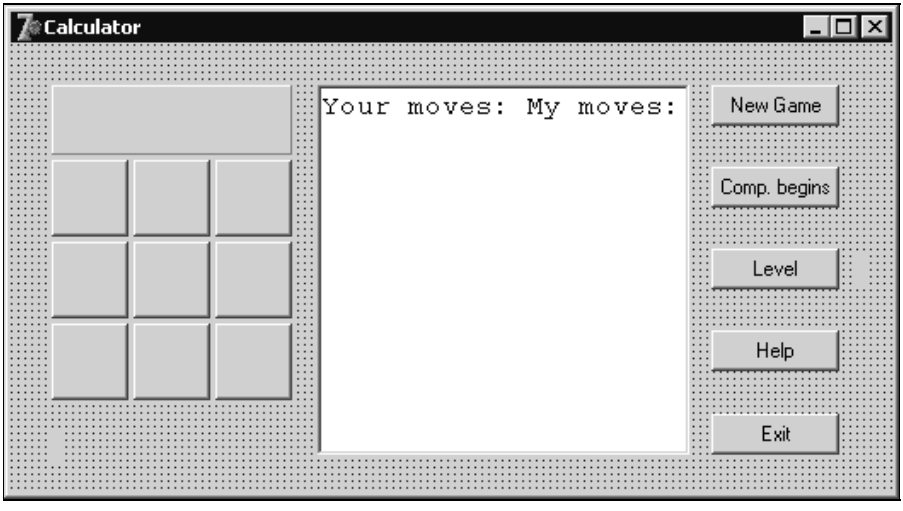


Рис. 2.1. Дизайнерская форма для игры Calculator

2.3. Программирование игры

Вот мы и приступаем собственно к программированию. Начнем со вспомогательных функций и секции данных. Для этого можно было бы отвести отдельный файл, но у нас программа небольшая, и мы разместим все в файле Unit1.pas. Вверху вы видите строки

```
var
 Form1: TForm1;
```

Когда мы будем добавлять свои переменные, то продолжим эту секцию `var`. Для констант сделаем секцию `const` выше этой секции `var`. Вспомогательные процедуры начнем размещать сразу за строкой `{$R *.dfm}`, чтобы они располагались раньше методов компонентов, которые будут их вызывать.

Разместим в секции констант массив глубин счета вариантов, число его элементов — это количество уровней игры программы:

```
const
 DEPTHS : array[0..8] of byte = (2,3,4,5,6,7,8,9,99);
```

В этой игре будет аж девять уровней силы игры компьютера. На последнем, девятом, уровне программа будет играть наисильнейшим образом. Дальше будут идти различные признаки во избежание в программе "магических чисел" — числовых литералов, значение которых трудно понять.

- `INTOSTR=0` — ход делается в строке;
- `INTOCOL=1` — ход делается в столбце;
- `RESLOST=0` — результат игры — проигрыш игрока;
- `RESWON=1` — результат игры — выигрыш игрока;
- `RESNOTKNOWN=2` — результат игры пока не известен;
- `HUMAN_MOVE=0` — этот ход делает человек;
- `COMP_MOVE=1` — этот ход делает компьютер.

Секцию `var` продолжите переменными как в листинге 2.1.

Листинг 2.1. Глобальные переменные программы

```
depth: byte=0;
depth0: byte;
tabl: array[0..2,0..2] of byte=((1,2,3),(4,5,6),(7,8,9));
indicator: integer;
level: integer=2;
movenum,oldstrnum,oldcolnum,compbegins: integer;
isplayersmove: boolean;
```

Переменные `depth` и `depth0` уже знакомы нам по игре "Ним". Первая используется в функции `IsWin` для ограничения глубины перебора ходов, а вторая содержит эту глубину перебора в зависимости от выбора уровня игры кнопкой `btLevel`. `tabl` — массив, содержащий цифры на кнопках калькулятора в порядке слева направо и сверху вниз. Эти цифры будут перемешиваться в начале каждой игры. `indicator` — значение на индикаторе калькулятора. `level` — уровень игры, но в массиве `DEPTHS` индексы начинаются с нуля, поэтому `level` будет изменяться от `0` до `8`, а отображаемый уровень возле кнопки `btLevel` будет на единицу больше. Первоначально для него выбрано значение `2` (показано будет `3`). Остальные переменные имеют такой смысл:

- `movenum` — счетчик сделанных ходов;
- `oldstrnum` — номер строки кнопок, в которой сделан предыдущий ход. В начале игры значение не определено;
- `oldcolnum` — аналогично для столбца кнопок;

- `compbegins` — игру начал компьютер;
- `isplayersmove` — если `TRUE`, то сейчас очередь хода за игроком, иначе — за компьютером.

После строки `{$R *.dfm}` пишем процедуры для будущей игры. Опережающее описание процедуры выполнения хода за компьютер `procedure CompMove; forward;` нужно будет для того, чтобы развязать ее с процедурой `DoMove`, которая выполняет все ходы, т. к. они вызывают друг друга. Далее идут мелкие вспомогательные процедуры, которые мы не рассматриваем ввиду их простоты:

- `SwapByte` — обмен заданных байтов местами;
- `ShowLevel` — показ уровня игры и установка `depth0`;
- `ShowIndicator` — показ числа на индикаторе;
- `ShowResult` — показ результата игры. Здесь задается цвет шрифта в виде двойного слова, которое имеет формат `$BBGGRR` (байты интенсивности синего, зеленого и красного цвета).

Рассмотрим процедуру отображения цифр на кнопках. После того как компьютер сделает ход, цифры в ряду, в котором может делать ход человек, будут отображаться для удобства красным цветом. Для этого `ShowDigits` будет получать два параметра — номер строки и столбца. Только один из них будет иметь значение от `0` до `2` (а другой будет равен `-1`). Вот в этом ряду, номер которого больше `-1`, цифры будут иметь красный цвет. Еще одна особенность: в цикле по строкам и столбцам клавиш мы должны будем по их индексам `i` и `j` находить ссылку на компонент данной кнопки. Мы не зря давали имена кнопкам в виде `k<`*`номер строки кнопки`*`><`*`номер столбца кнопки`*`>`. Теперь по индексам кнопки мы легко получим имя ее компонента. А по имени компонента с помощью метода формы `FindComponent` найдем ссылку `sb` на компонент, носящий это имя, а через нее доберемся до цвета шрифта и надписи на кнопке. Оператор

```
sb.Font.Color:=integer((i = redstr) or (j = redcol))*$B0;
```

это хитрое сокращение от операторов

```
sb.Font.Color:=0;
if (i = redstr) or (j = redcol) then sb.Font.Color:=$B0;
```

Выбирайте вариант, который вам больше нравится.

Функция `IsWin` аналогична одноименной функции из программы ournim.pas.

Процедура `DoMove` делает заданный ход как для человека, так и для компьютера. После обновления индикатора и поля `Memo1` она вызывает метод `Application.ProcessMessages`, чтобы наша программа могла обработать со-

общения и отрисовать новые значения. Иначе мы не будем видеть изменения индикатора после наших ходов, и кнопка, по которой мы щелкнули, будет оставаться в нажатом состоянии, пока компьютер не сделает ответный ход или не закончится игра.

Процедура `CompMove` ищет выигрывающий ход за компьютер. При этом она перебирает ходы в порядке убывания цифр, чтобы сделать игру как можно короче. Представьте, что выигрывает как ход `1`, так и ход `9`. Конечно, лучше сделать ход `9`, чтобы не мучить игрока. При поиске выигрывающего хода запоминается ход наименьшей цифрой с неизвестным результатом (`IsWin` не досчиталась до конца игры). Если цикл перебора всех ходов будет закончен, а выигрыш не найден, то этот лучший ход с неизвестным результатом и будет сделан. Если и такого хода не будет обнаружено (`IsWin` сказала, что все ходы компьютера проигрывают), то компьютер из возможных ходов отыщет ход с наименьшей цифрой и сделает его, если после него на индикаторе не окажется отрицательное число, иначе машина поднимет руки кверху за шаг от проигрыша.

Процедура `NewGame` вызывается из обработчика события `FormCreate`, а также при щелчке по кнопке `btNew`. Она перемешивает цифры в массиве `tabl` и приводит нашу форму и ключевые переменные программы в первоначальный вид.

Теперь пойдут события и методы формы `Form1`. В инспекторе объектов для формы `Form1` щелкните на вкладке **Events** (События) и затем сделайте двойной щелчок на поле для **OnCreate**. В этом поле появится слово `FormCreate`, и Delphi создаст в тексте модуля Unit1.pas заготовку обработчика для этого события. Вам останется вписать в него вызовы `ShowLevel` и `NewGame`.

Затем сделайте двойные щелчки по кнопкам `btNew`, `btCompBegins`, `btLevel`, `btHelp` и `btExit`. Впишите в них команды, как в листинге 2.2 (кроме кнопки `btHelp`, которую мы обработаем позже).

Листинг 2.2. Код для кнопок

```
// Щелчок на кнопке "Comp. begins"
procedure TForm1.btCompBeginsClick(Sender: TObject);
begin
      if movenum = 0 then CompMove;
end;

// Щелчок на кнопке "New Game"
procedure TForm1.btNewClick(Sender: TObject);
```

```
begin
      NewGame;
end;

// Щелчок на кнопке "Level"
procedure TForm1.btLevelClick(Sender: TObject);
begin
      if movenum = 0 then
       begin
       level:=(level+1) mod 9;
       ShowLevel;
       end;
end;

// Щелчок на кнопке "Exit"
procedure TForm1.btExitClick(Sender: TObject);
begin
      Close;
end;
```

Далее нам нужна будет процедура, обрабатывающая щелчки по кнопкам `SpeedButton` калькулятора. Она называется `SBClick`. Сами обработчики кнопок вызывают ее, передавая свои координаты. Для этого сделайте двойные щелчки по порядку на каждой кнопке и оформите обработчики их событий `onClick`.

2.4. Создание формы для вывода правил игры

Для вывода правил игры сделаем отдельную форму. Для этого в меню **File** (Файл) выберем команду **New | Form** (Создать | Форма). На экране появится форма с именем **Form2**. Переименуйте ее в `fmHelp`, а в поле **Caption** запишите `Help`. Свойства **BorderIcons** и **BorderStyle** установите такими же, как и у основной формы. Выберите меню **Project | Options** (Проект | Настройка) и откройте окно проекта нашей программы. В нем войдите на вкладку **Forms**. В окне **Auto-create forms** (Автоматически созданные формы) вы видите обе формы. Это означает, что при старте программы форма `fmHelp` также будет

создана и станет занимать память. Если в программе много форм, то она будет медленно стартовать и занимать лишнюю память, поэтому мы в порядке выработки общего стиля программирования оставим в этом окне только основную форму, а остальные будем перемещать в **Available forms** (Доступные формы). Выберите форму `fmHelp` и щелкните по кнопке со стрелкой вправо, перемещая ее в это окно. Щелкните по кнопке **OK**. Результат вы видите на рис. 2.2.

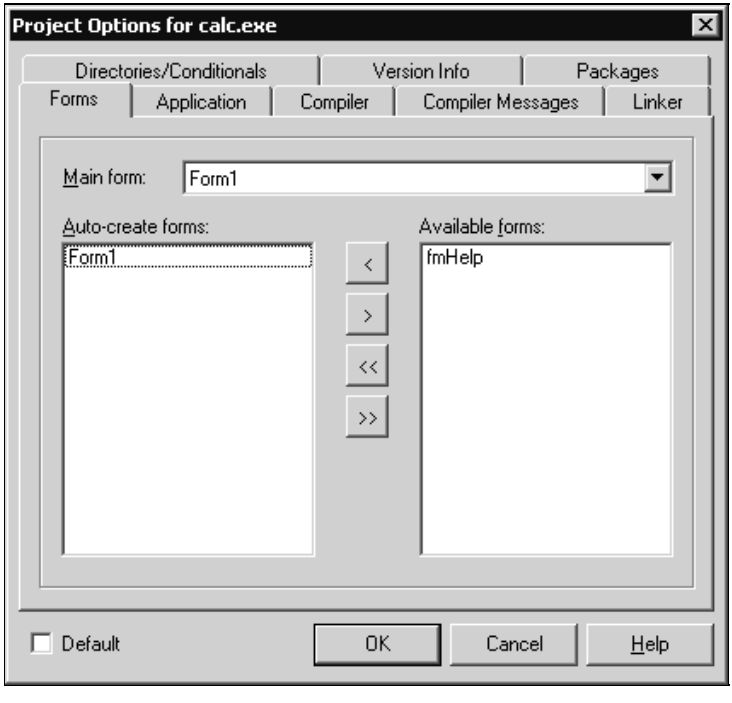


Рис. 2.2. Окно опций проекта игры Calculator

В обработчик `FormCreate` вставьте код, представленный в листинге 2.3.

Листинг 2.3. Код для создания формы `fmHelp`

```
if not Assigned(fmHelp) then fmHelp:=TfmHelp.Create(Form1);
  with fmHelp do
    begin
    left:=Form1.Left;
    top:=Form1.Top;
```

```
ShowModal;
Free;
fmHelp:=NIL;
end;
```

В этом коде у нас создается новое модальное окно с текстом справки, и его экранные координаты устанавливаются такими, как у основной формы. Пока мы его не закроем, мы не сможем работать с программой.

Создайте в форме компонент `Memo1`, задайте его свойства **WordWrap** = `True` и размер шрифта `10`. Введите в поле `Memo1` следующий текст:

```
Игра Calculator, Copyright© Сергей Мельников, 2006.

Эта игра создана специально для книги "Turbo Pascal и Delphi на
занимательных примерах". Правила ее просты: первый игрок
может нажать любую из девяти кнопок. После каждого нажатия
число на индикаторе уменьшается на цифру, написанную на этой
кнопке. Второй игрок может нажать любую из трех кнопок в
том столбце, в котором делал предыдущий ход первый игрок,
а первый игрок в свои последующие ходы может нажимать
любую из трех кнопок в той строке, в которой второй игрок
нажимал кнопку последний раз. Проигрывает тот игрок, после
чьего хода на индикаторе появится отрицательное число.
```

Символ © набираем так: нажимаем клавишу <NumLock>, затем при нажатой клавише <Alt> набираем на дополнительной клавиатуре код этого символа: 0169, отпускаем <Alt>, выключаем индикатор NumLock.

Разместите справа от `Memo1` кнопку `btOK` с надписью **OK** для закрытия этой формы аналогично кнопке `btExit` основной формы. Но не создавайте для нее обработчик события `onClick`, а вместо этого установите свойство **ModalResult** в `mrOK`. Теперь эта кнопка сама будет закрывать форму. Так можно было поступить и в основной форме.

Для удобства сделаем, чтобы это окно закрывалось при нажатии клавиши <Esc>. Для этого установим свойство формы **KeyPreview** в `True`, чтобы форма могла обрабатывать нажатия клавиш до их отсылки в активный компонент. На вкладке **Events** сделайте двойной щелчок на поле справа от свойства **OnKeyPress** и создайте обработчик события `FormKeyPress`. Введите в нем такой код:

```
if key = char(VK_ESCAPE) then Close;
```

Аналогично можно сделать, чтобы эта форма с правилами игры открывалась при нажатии клавиши <F1>.

В листинге 2.4 — текст модуля Unit1.pas.

Листинг 2.4. Модуль Unit1.pas

```
{$B-}
unit Unit1;

interface

uses
 Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
 Forms, Dialogs, Buttons, ExtCtrls, StdCtrls;

type
  TForm1 = class(TForm)
    k00: TSpeedButton;
    k01: TSpeedButton;
    k02: TSpeedButton;
    k12: TSpeedButton;
    k10: TSpeedButton;
    k11: TSpeedButton;
    k22: TSpeedButton;
    k20: TSpeedButton;
    k21: TSpeedButton;
    Panel1: TPanel;
    Memo1: TMemo;
    btNew: TButton;
    btCompBegins: TButton;
    btLevel: TButton;
    btHelp: TButton;
    btExit: TButton;
    laLevel: TLabel;
    laResult: TLabel;
    procedure FormCreate(Sender: TObject);
    procedure btNewClick(Sender: TObject);
```

```
    procedure btLevelClick(Sender: TObject);
    procedure btCompBeginsClick(Sender: TObject);
    procedure btExitClick(Sender: TObject);
    procedure k00Click(Sender: TObject);
    procedure k01Click(Sender: TObject);
    procedure k02Click(Sender: TObject);
    procedure k10Click(Sender: TObject);
    procedure k11Click(Sender: TObject);
    procedure k12Click(Sender: TObject);
    procedure k20Click(Sender: TObject);
    procedure k21Click(Sender: TObject);
    procedure k22Click(Sender: TObject);
    procedure btHelpClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

const
 DEPTHS: array[0..8] of byte=(2,3,4,5,6,7,8,9,99);
 INTOSTR=0;
 INTOCOL=1;
 RESLOST=0;
 RESWON=1;
 RESNOTKNOWN=2;
 HUMAN_MOVE=0;
 COMP_MOVE=1;

var
 Form1: TForm1;
 depth: byte=0;
 depth0: byte;
 tabl: array[0..2,0..2] of byte=((1,2,3),(4,5,6),(7,8,9));
 indicator: integer;
 level: integer=2;
```

```
 movenum,oldstrnum,oldcolnum,compbegins: integer;
 isplayersmove: boolean;

implementation

uses Unit2;

{$R *.dfm}

procedure CompMove; forward;

// Обмен байтов b1 и b2 местами
procedure SwapByte(var b1,b2: byte);
var
 b: byte;

begin
      b:=b1;
      b1:=b2;
      b2:=b;
end; {SwapByte}

// Показ уровня игры и установка depth0
procedure ShowLevel;
begin
      Form1.laLevel.Caption:=IntToStr(level+1);
      // Глубина счета вариантов
      depth0:=DEPTHS[level];
end; {ShowLevel}

// Показываем число на индикаторе
procedure ShowIndicator;
begin
      Form1.Panel1.Caption:=IntToStr(indicator);
end; {ShowIndicator}
```

```
// Показ результата игры
procedure ShowResult(resind: integer);
const
 RES: array[0..2] of string=('you lost','you won!','none');
 COLORS: array[0..2] of integer=($E00000,$B0,0);

begin
      with Form1.laResult do
       begin
       Font.Color:=COLORS[resind];
       Caption:='Result: '+RES[resind];
       end;
end; {ShowResult}

{ Отображение цифр на кнопках. Цифры в строке redstr и столбце redcol
  выводятся красным цветом }
procedure ShowDigits(redstr,redcol: integer);
var
 i,j: integer;
 sb: TSpeedButton;

begin
      for i:=0 to 2 do
       for j:=0 to 2 do
        begin
        // Находим адрес нужного компонента кнопки по его имени
        sb:=TSpeedButton(Form1.FindComponent(Format('k%d%d',[i,j])));
        sb.Font.Color:=integer((i = redstr) or (j = redcol))*$B0;
        sb.Caption:=IntToStr(tabl[i,j]);
        end;
end; {ShowDigits}

{ Выяснение результата хода strnum, colnum. Если str_col=tstr, то игрок,
  делающий этот ход, ходит в строке, иначе - в столбце. Результат:
  0 - проигрыш, 1 - выигрыш, 2 - неизвестно. }
```

```
function IsWin(indicator,strnum,colnum,str_col: integer): byte;
label
 WON,LOST,NOTKNOWN,EX;
var
 i,newindicator,tmpres,res: integer;

begin
      // Ограничиваем глубину рекурсии
      Inc(depth);
      if depth > depth0 then goto NOTKNOWN;
      // Делаем заданный ход
      newindicator:=indicator-tabl[strnum,colnum];
      if newindicator < 0 then goto LOST;
      if newindicator = 0 then goto WON;
      // Результат этого хода неизвестен. Перебираем ходы за соперника
      res:=RESLOST;
      for i:=0 to 2 do
       begin
       if str_col = INTOSTR then
        // Перебираем ходы в столбце colnum
        tmpres:=IsWin(newindicator,i,colnum,INTOCOL)
        else
         // Перебираем ходы в строке strnum
         tmpres:=IsWin(newindicator,strnum,i,INTOSTR);
       if tmpres = RESWON then goto LOST;
       if tmpres = RESNOTKNOWN then res:=RESNOTKNOWN;
       end;
      if res = RESNOTKNOWN then goto NOTKNOWN;
WON:  IsWin:=RESWON;
      goto EX;
NOTKNOWN:
      IsWin:=RESNOTKNOWN;
      goto EX;
LOST: IsWin:=RESLOST;
EX:   Dec (depth);
end; {IsWin}
```

```
{ Выполнение хода strnum, colnum. Если iscompmove=0, то это ход игрока,
  если 1 - компьютера }
procedure DoMove(strnum,colnum,iscompmove: integer);
begin
      // Корректируем и выводим индикатор
      Dec(indicator,tabl[strnum,colnum]);
      ShowIndicator;
      // Выводим ход
      if iscompmove = COMP_MOVE then
       // Ход компьютера добавляем в последнюю строку Memo1
       with Form1.Memo1 do
        begin
        if movenum = 0 then Lines.Append('     ');
        Lines[Lines.Count-1]:=Lines[Lines.Count-1]+'           '+
        IntToStr(tabl[strnum,colnum]);
        end
       else
        // Ход игрока пишем в новую строку
        Form1.Memo1.Lines.Append('    '+IntToStr(tabl[strnum,colnum]));
      // Даем отрисоваться индикатору и выведенному ходу
      Application.ProcessMessages;
      // Конец игры?
      if indicator < 0 then
       begin
       ShowResult(iscompmove);
       Exit;
       end;
      // Отрабатываем сделанный ход
      Inc(movenum);
      oldstrnum:=strnum;
      oldcolnum:=colnum;
      isplayersmove:=not isplayersmove;
      if not isplayersmove then
       // Делаем ответный ход компьютера
       CompMove
```

```
       else
        // После хода компьютера отображаем красным ряд кнопок
        if compbegins = 1 then
         ShowDigits(-1,colnum)
         else
          ShowDigits(strnum,-1);
end; {DoMove}

// Ход компьютера
procedure CompMove;
var
 i,j,res,newindicator,i1,j1,i2,j2: integer;
 ind: array[0..8,0..1] of integer;

begin
      // Начинает игру компьютер?
      if movenum = 0 then compbegins:=1;
      // Сейчас очередь ходить компьютеру
      isplayersmove:=FALSE;
      // Делаем задержку времени перед ходом
      Sleep(800);
      if indicator = 0 then
       // Если на индикаторе 0, то заканчиваем игру
       begin
       ShowResult(RESWON);
       Exit;
       end;
      { В i1, j1, newindicator будем запоминать ход i, j с неизвестным
        результатом и наибольшим числом на индикаторе после этого хода }
      newindicator:=-1;
      { В массиве ind запомним индексы цифр из массива tabl
        по возрастанию цифр }
      for res:=0 to 8 do
       for i:=0 to 2 do
        for j:=0 to 2 do
         if tabl[i,j] = res+1 then
```

```
    begin
    ind[res,0]:=i;
    ind[res,1]:=j;
    end;
// Ищем выигрывающий ход во всех трех рядах цифр
for i:=8 downto 0 do
 begin
 // i2, j2 - координаты следующей по убыванию цифры
 i2:=ind[i,0];
 j2:=ind[i,1];
 { Если это не первый ход, то не рассматриваем ходы
   в ненужном ряду }
 if (movenum > 0) and
  ((i2 <> oldstrnum) and (compbegins = 1) or
   (j2 <> oldcolnum) and (compbegins = 0)) then continue;
 // Рассматриваем ход i2, j2 компьютера
 res:=IsWin(indicator,i2,j2,1-compbegins);
 // Он выигрышный?
 if res = RESWON then
  // Да, делаем его и выходим
  begin
  ShowResult(RESLOST);
  DoMove(i2,j2,COMP_MOVE);
  Exit;
  end;
 { Ход i2, j2 не выигрывает. Это ход с неизвестным
   результатом и меньшей цифрой, чем запомненный? }
 if (res = RESNOTKNOWN) and
  (indicator-tabl[i2,j2] > newindicator) then
  // Да, запоминаем его
  begin
  newindicator:=indicator-tabl[i2,j2];
  i1:=i2;
  j1:=j2;
  end;
 end;
```

```
      { Выигрыш не найден. Если есть ход с неизвестным результатом,
        то делаем его }
      if newindicator > -1 then
       begin
       DoMove(i1,j1,COMP_MOVE);
       Exit;
       end else
        // Иначе пытаемся сделать ход наименьшей цифрой
        for i:=0 to 8 do
         begin
         // Рассматриваем ход i2, j2
         i2:=ind[i,0];
         j2:=ind[i,1];
         { Если это не первый ход, то не рассматриваем ходы
           в ненужном ряду }
         if (movenum > 0) and
          ((i2 <> oldstrnum) and (compbegins = 1) or
           (j2 <> oldcolnum) and (compbegins = 0)) then continue;
         { Если после этого хода индикатор > 0, то делаем его
           и выходим }
         if indicator-tabl[i2,j2] > 0 then
          begin
          DoMove(i2,j2,COMP_MOVE);
          Exit;
          end else
           // Иначе выходим из цикла и показываем выигрыш игрока
           break;
         end;
      ShowResult(RESWON);
end; {CompMove}

// Старт новой игры
procedure NewGame;
```

```
var
 i,j: integer;

begin
      Randomize;
      // Перемешиваем числа в массиве tabl
      for i:=0 to 2 do
       for j:=0 to 2 do SwapByte(tabl[i,j],tabl[Random(3),Random(3)]);
      // Отображаем черным цветом цифры на кнопках
      ShowDigits(-1,-1);
      // Устанавливаем число на индикаторе в диапазоне 30..35
      indicator:=Random(6)+30;
      ShowIndicator;
      // Счетчик сделанных ходов
      movenum:=0;
      // Очищаем вывод ходов в Memo1
      Form1.Memo1.Lines.Clear;
      Form1.Memo1.Lines.Append('Your moves: My moves:');
      // Разрешаем игроку начать игру
      isplayersmove:=TRUE;
      // Результат игры пока неизвестен
      ShowResult(RESNOTKNOWN);
end; {NewGame}

// Начальная установка программы
procedure TForm1.FormCreate(Sender: TObject);
begin
      ShowLevel;
      NewGame;
end;

// Щелчок на кнопке "Comp. begins"
procedure TForm1.btCompBeginsClick(Sender: TObject);
begin
      if movenum = 0 then CompMove;
end;
```

```
// Щелчок на кнопке "New Game"
procedure TForm1.btNewClick(Sender: TObject);
begin
      NewGame;
end;

// Щелчок на кнопке "Level"
procedure TForm1.btLevelClick(Sender: TObject);
begin
      if movenum = 0 then
       begin
       level:=(level+1) mod 9;
       ShowLevel;
       end;
end;

// Щелчок на кнопке "Help"
procedure TForm1.btHelpClick(Sender: TObject);
begin
      if not Assigned(fmHelp) then fmHelp:=TfmHelp.Create(Form1);
        with fmHelp do
         begin
         left:=Form1.Left;
         top:=Form1.Top;
         ShowModal;
         Free;
         fmHelp:=NIL;
         end;
end;

// Щелчок на кнопке "Exit"
procedure TForm1.btExitClick(Sender: TObject);
begin
      Close;
end;
```

```
// Обработка щелчка на кнопке в строке strnum и столбце colnum
procedure SBClick(strnum,colnum: integer);
begin
      // Сейчас не ход игрока или игра закончена?
      if not isplayersmove or (indicator < 0) then Exit;
      // Проверяем правильность хода игрока
      if movenum > 0 then
       begin
       // Ход должен быть в том же столбце, что и предыдущий
       if (compbegins = 1) and (colnum <> oldcolnum) then Exit;
       // Ход должен быть в той же строке, что и предыдущий
       if (compbegins = 0) and (strnum <> oldstrnum) then Exit;
       end
      else
       // Игру начал человек
       compbegins:=0;
      // Выполняем ход strnum, colnum
      DoMove(strnum,colnum,HUMAN_MOVE);
end; {Click}

// Щелчки на кнопках калькулятора. Передаем строку и столбец кнопки
procedure TForm1.k00Click(Sender: TObject);
begin
      SBClick(0,0);
end;

procedure TForm1.k01Click(Sender: TObject);
begin
      SBClick(0,1);
end;

procedure TForm1.k02Click(Sender: TObject);
begin
      SBClick(0,2);
end;
```

```
procedure TForm1.k10Click(Sender: TObject);
begin
      SBClick(1,0);
end;

procedure TForm1.k11Click(Sender: TObject);
begin
      SBClick(1,1);
end;

procedure TForm1.k12Click(Sender: TObject);
begin
      SBClick(1,2);
end;

procedure TForm1.k20Click(Sender: TObject);
begin
      SBClick(2,0);
end;

procedure TForm1.k21Click(Sender: TObject);
begin
      SBClick(2,1);
end;

procedure TForm1.k22Click(Sender: TObject);
begin
      SBClick(2,2);
end;

end.
```

Листинг 2.5 — для модуля Unit2.pas.

Листинг 2.5. Модуль Unit2.pas

```
unit Unit2;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls;

type
  TfmHelp = class(TForm)
    Memo1: TMemo;
    btOK: TButton;
    procedure FormKeyPress(Sender: TObject; var Key: Char);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  fmHelp: TfmHelp;

implementation

{$R *.dfm}

procedure TfmHelp.FormKeyPress(Sender: TObject; var Key: Char);
begin
      if key = char(VK_ESCAPE) then Close;
end;

end.
```

На рис. 2.3 вы видите форму для модуля Unit2.pas.

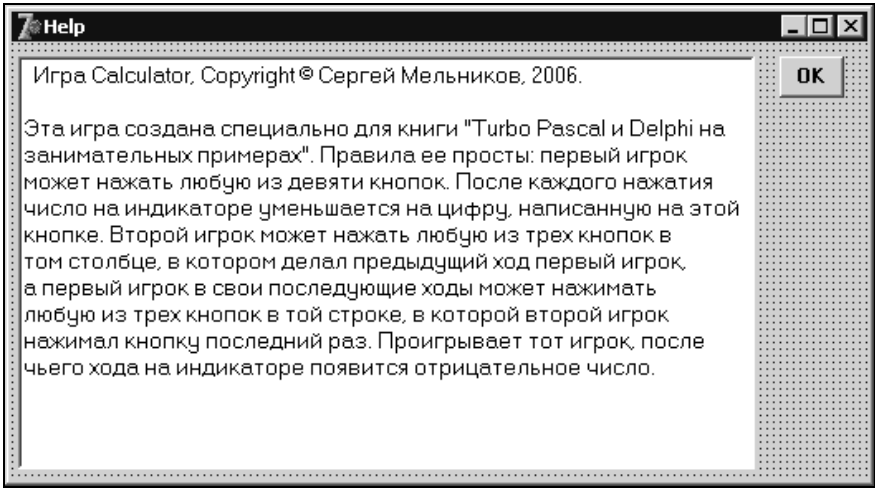


Рис. 2.3. Форма `fmHelp` для игры Calculator

Домашнее задание

1. Чтобы для красоты выделить в рамку надписи с результатом и уровнем игры, замените компоненты **Label** на **Static Text** (Статический текст) — находится на вкладке **Additional** — и настройте его свойства, начинающиеся со слова **Bevel**.
2. Сдвиньте вверх кнопки справа от поля **Memo1** и создайте на освободившемся месте вывод общего счета за все игры с последнего старта программы.
3. Сделайте возможность управления программой с клавиатуры, чтобы играть можно было также нажатием цифровых клавиш. У кнопок справа сделайте "горячие символы" (которые будут подчеркнуты при работе программы). Для этого поставьте в поле **Caption** символ `&` перед "горячей буквой", например: `&New Game`. После этого нажатие клавиши <N> запустит новую игру аналогично щелчку по кнопке `btNew`.
4. Цифры на кнопках, выполненные шрифтом 28-го размера, выглядят не очень привлекательно. Вы можете использовать картинки из bmp-файлов для отображения на кнопках, которые для этого имеют свойство **Glyph** (Глиф). Например, так, как представлено в листинге 2.6.

Листинг 2.6. Код для использования картинок на кнопках

```
var
 bm: TBitMap;
...
```

```
bm:=TBitMap.Create;
bm.LoadFromFile('d:\new32.bmp');
SpeedButton1.Glyph:=bm;
bm.Free;
```

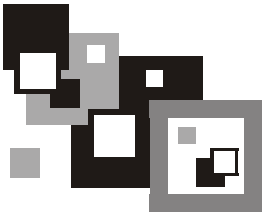
Также можно загружать bmp-файлы из ресурсов.

Существуют также бесплатные компоненты с исходными текстами для кнопок произвольной формы. Их и многое другое вы можете найти на сайте Максима Пересады **www.torry.net**. На сайте Виталия Невзорова **http://www.drkb.ru** имеется большая и очень полезная база знаний по Delphi в виде chm-файла. При скачивании этого файла подправьте URL, оставив в нем только ссылку на сам этот файл, потому что редирект из php-скрипта на него не происходит.

Радикальным решением является использование картинки поверх всей формы. Вместо стандартных компонентов используются изображения кнопок и других объектов управления, как костер и котелок на холсте в коморке старого папы Карло. Определение, на каком объекте пользователь щелкнул мышью, происходит при анализе координат указателя. Для таких игр библиотека классов, из-за которой exe-файлы имеют столь большие размеры, в общем-то не нужна. Другие способы улучшения внешнего вида окна программы мы еще рассмотрим.

Исходные и результирующие файлы для игры Calculator размещены на прилагаемом к книге компакт-диске исключительно с целью обучения и не могут быть использованы в любых иных целях. Вы не имеете права публиковать где бы то ни было и в любой форме как текст этой программы, так и результирующий exe-файл. Также вы не имеете права использовать принадлежащую мне, как автору, идею этой игры.

Глава 3



В Win API под DirectX

- Шаблон программы в чистом Win API
- Оконная функция и обработка сообщений
- Создание поверхностей DirectDraw, переход в полноэкранный режим
- Вывод BMP-файла на поверхность DirectDraw
- Свой указатель мыши
- WAV-звук и MIDI-музыка
- Создаем оригинальную головоломку с нуля

3.1. Шаблон минимальной программы Delphi

Если вы привыкли использовать визуальные компоненты Delphi, то этот материал покажется вам незнакомым, но с ним стоит познакомиться, чтобы получить представление о настоящем программировании под Windows. Библиотека VCL (Visual Component Library, библиотека визуальных компонентов) Delphi служит удобной оболочкой для Windows API (Application Program Interface, программный интерфейс приложений), которая существенно ускоряет процесс штамповки программ, но иногда бывает нужно создать небольшую программу или библиотеку без сотни с лишним килобайтов неиспользуемого кода. Поэтому сейчас мы познакомимся с тем, как работает программа под управлением Windows. Читателю, который перешагнул через рекурсивные функции, прыгающую блоху и поиск позиций в головоломке Full Board, здесь уже не должно быть очень страшно. В листинге 3.1 представлена минимальная Delphi-программа, создающая окно.

Листинг 3.1. Минимальная Delphi-программа

```
{ Шаблон минимальной программы в Win API }
{$A-,B-,I-,R-,S-,X+,H+}
program Window1;
uses
 Windows, Messages;
const
  AppName = 'Window1';
var
  Message: TMsg;
  hWindow: HWnd;

function WindowProc(Window: HWnd; AMessage: UInt;
 WParam: WParam; LParam: LParam): LResult; stdcall; export;
begin
  WindowProc:=0;
  case AMessage of
wm_Destroy:
   begin
   PostQuitMessage(0);
   Exit;
   end;
  end; {case}
  WindowProc:=DefWindowProc(Window, AMessage, WParam, LParam);
end;

{ Регистрируем класс окна }
function WinRegister: Boolean;
var
  WindowClass: TWndClass;

begin
  WindowClass.Style:=cs_hRedraw or cs_vRedraw;
  WindowClass.lpfnWndProc:=@WindowProc;
  WindowClass.cbClsExtra:=0;
```

```
  WindowClass.cbWndExtra:=0;
  WindowClass.hInstance:=HInstance;
  WindowClass.hIcon:=LoadIcon(0, idi_Application);
  WindowClass.hCursor:=LoadCursor(0, idc_Arrow);
  WindowClass.hbrBackground:=HBrush(Color_Window);
  WindowClass.lpszMenuName:=nil;
  WindowClass.lpszClassName:=AppName;
  Result:=RegisterClass(WindowClass) <> 0;
end;

{ Создаем окно программы }
function WinCreate: HWnd;
var
  hWindow: HWnd;

begin
  hWindow:=CreateWindow(AppName,'Our Window',
   ws_OverlappedWindow,
   Integer(cw_UseDefault), Integer(cw_UseDefault),
   Integer(cw_UseDefault), Integer(cw_UseDefault),
   0, 0, HInstance, nil);
  if hWindow <> 0 then
   begin
   ShowWindow(hWindow, SW_SHOW);
   UpdateWindow(hWindow);
   end;
  Result:=hWindow;
end;

BEGIN
  if not WinRegister then
   begin
   MessageBox(0, 'WinRegister failed', nil, mb_Ok);
   Exit;
   end;
  hWindow:=WinCreate;
```

```
  if hWindow = 0 then
   begin
   MessageBox(0, 'WinCreate failed', nil, mb_Ok);
   Exit;
   end;
  // Цикл обработки сообщений
  while GetMessage(Message, 0, 0, 0) do
   begin
   TranslateMessage(Message);
   DispatchMessage(Message);
   end;
  Halt(Message.wParam);
END.
```

Вы можете транслировать файлы без помощи среды разработки Delphi. Для этого в каталоге Bin имеется автономный транслятор из командной строки dcc32.exe. Ему надо передать имя dpr-файла: `dcc32 window1.dpr`. Настройте ваш файл-менеджер, чтобы он запускал транслятор при нажатии клавиши <Enter> на файле с расширением dpr, передавал ему имя этого dpr-файла, и работайте.

Эта программа создает окно с кнопками оконного меню. Она может сворачиваться, распахиваться на весь экран и закрываться. Для начала неплохо, особенно, если учесть, что размер exe-файла всего 15 Кбайт. Как же работает эта программа? Когда Windows передает управление программе, то она записывает в ее стек несколько параметров, среди которых:

- `Hinstance` — уникальное четырехбайтовое число, идентифицирующее данный экземпляр приложения;
- указатель на командную строку, которой была запущена эта программа.

В программах на C и ассемблере пишется функция с именем `WinMain`, которую вызывает Windows. В программе на Delphi работает стартовый код, который извлекает эти параметры из стека, и они становятся доступны через глобальную переменную `Hinstance` и функцию `ParamStr`.

Фирма Borland купила у Microsoft лицензию на их hlp-файлы, которые включены в поставку Delphi. Вы их можете найти в каталоге \Program Files\Common Files\Borland Shared\MSHelp. Посмотрите файл win32.hlp или win32sdk.hlp. Там содержится большая справка по функциям и структурам Windows API. В строке ввода для поиска по ключевым словам наберите слово `winmain`, нажмите клавишу <Enter> и читайте подробности. К сожале-

нию, эта справка написана применительно к языку C, т. к. ядро самой ОС Windows тоже написано на этом языке.

Первым делом наша программа регистрирует класс своего окна, обращаясь к функции `WinRegister`. Эта функция заполняет структуру `WindowClass`, а затем вызывает функцию Windows API `RegisterClass`, передавая ей эту структуру. В частности, в этой структуре передается указатель на оконную функцию `WindowProc` обработки сообщений, `Hinstance`, имя класса окна, которое у нас записано в переменной `AppName`, идентификаторы курсора и значка, кисть для закраски фона окна. Цвет фона окна у нас задан константой `Color_Window`. Эта и ей подобные константы определены в файле в каталоге Delphi Source\Rtl\Win\Windows.pas. Это просто число `5`. Оно соответствует выбору пользователя, когда тот задает фон окна в Панели управления для свойств экрана. Но программа может задать фон окна цветом других интерфейсных элементов Windows, подставляя другие константы вместо `Color_Window`, а также закрасить в любой цвет или образцом заданного изображения, создав соответствующую кисть. Для закраски окна в белый цвет подставьте вместо строки

```
WindowClass.hbrBackground:=HBrush(Color_Window);
```

строку

```
WindowClass.hbrBackground:=HBrush(GetStockObject(WHITE_BRUSH));
```

Теперь Windows будет использовать эту кисть для закраски фона нашего окна. Так как меню у нашего окна нет, то в поле структуры `lpszMenuName` передается `nil`.

При успешной регистрации класса окна функция `RegisterClass` возвратит ненулевое значение. В ином случае не стоит запускать программу — скорее всего, случилась ошибка в вызове функции `RegisterClass`.

После успешной регистрации класса окна основной код нашей программы обращается к функции `WinCreate`, которая вызывает функцию Windows API `CreateWindow`. В частности, этой функции передается имя класса нашего будущего окна, строка, которая будет отображаться в его заголовке, стиль, положение и размеры окна. При успешном создании окна возвращается handle этого окна. Это уникальное четырехбайтовое число, которое сопоставляется нашему окну. В случае успеха `WinCreate` вызывает функцию ядра Windows `ShowWindow` для вывода окна поверх остальных окон и функцию `UpdateWindow` для перерисовки окна.

Далее в нашей программе стоит цикл обработки сообщений. Сообщения — это небольшие структуры данных, с помощью которых Windows управляет работой всех программ. У окон, так же как и у потоков, есть свои очереди сообщений, в которые поступают сообщения от Windows и от других прикладных программ. Получив квант процессорного времени, программа с

помощью функции Windows API `GetMessage` извлекает из очереди окна следующее сообщение, помещает его в структуру `Message`, обрабатывает его и возвращает управление Windows. Но сначала с помощью функции `TranslateMessage` сообщения, связанные с клавиатурой, транслируются, и при нажатии алфавитно-цифровой клавиши вырабатывается сообщение `wm_Char`, которое учитывает раскладку клавиатуры, а также состояние клавиш <Caps Lock> и <Shift>. В сообщении `wm_Char` вместо виртуального кода клавиши присутствует код соответствующего символа. Функция `DispatchMessage` направляет сообщение оконной функции, которая у нас называется `WindowProc`. Windows и другие программы также могут передавать сообщения напрямую этой функции, минуя очередь сообщений.

Оконная функция имеет тип вызова C и получает четыре жестко заданных в Windows параметра: handle окна, код сообщения Windows и два дополнительных параметра. Все аргументы функции и возвращаемый ею результат — четырехбайтовые числа. Мы могли бы определить тип всех этих чисел как `Integer`, и транслятор ни слова бы не имел против этого.

Окно получает сотни типов различных сообщений, но помнить все их и обрабатывать самостоятельно нет необходимости. Для этого существует функция `DefWindowProc`, которая обработает необработанные сообщения, осуществляя стандартное поведение окна. Попробуйте в оператор `case` оконной функции после строки

```
case AMessage of
```

вставить строку

```
WM_ERASEBKGND: Exit;
```

Мы таким путем игнорируем сообщение, что окно должно перерисовать свой фон вследствие его появления на экране, увеличения размеров, открытия его закрытых другими окнами частей. И теперь никто больше не закрашивает окно.

Наша оконная функция самостоятельно обрабатывает лишь одно сообщение — `wm_Destroy`, которое посылается окну, чтобы оно уничтожило себя. Это сообщение не обрабатывается `DefWindowProc`. В ответ на него программа должна освободить занимаемые ресурсы и сделать вызов `PostQuitMessage(0)`, который сгенерирует сообщение `wm_Quit`. Функция `GetMessage`, приняв это сообщение, возвратит `FALSE`, и произойдет выход из цикла обработки сообщений. В переменной `Message.wParam` обычно находится код завершения программы. Впрочем, если после цикла не будет стоять `Halt`, то программа все равно нормально завершится, потому что транслятор вставляет в конец программы вызов функции Windows API `ExitProcess`, который и завершит наш процесс.

А когда программа может инициализировать свои структуры, запрашивать и распределять память? Это можно сделать в основном коде перед циклом обработки сообщений или при получении сообщения `wm_Create`. Оно посылается после создания окна, но до его появления на экране.

Что произойдет, если пользователь забудет, что программа уже запущена, и запустит ее еще раз? Появится второе такое же окно, и в системе будут работать две копии одной программы. Это бывает не всегда допустимо, обычно вместо этого первый экземпляр программы выводит свое окно поверх всех остальных и передает ему фокус. Сейчас мы это проделаем. Для этого заведите глобальную переменную `var h: HWnd`. В самое начало программы после строки

```
BEGIN
```

вставьте такой код:

```
// При повторном запуске программы активизируем запущенный экземпляр
h:=FindWindow(AppName,'Our Window');
if h > 0 then
  begin
  SendMessage(h,WM_SETFOCUS,h,0);
  ShowWindow(h,SW_SHOWNORMAL);
  Exit;
  end;
```

Функция `FindWindow` ищет окно по имени его класса и тексту окна и в успешном случае возвращает его handle. Тогда мы посылаем этому окну сообщение о передаче ему фокуса, помещаем его поверх остальных окон через обращение `ShowWindow` и выходим из программы. Теперь у нас будет работать лишь один экземпляр приложения. Для этого окно не должно изменять свой текст заголовка.

Можно искать окно только по имени его класса, тогда вместо текста заголовка окна надо передать `nil`. Разумеется, имя класса должно быть уникальным.

3.2. Шаблон второй программы

Теперь немного нарастим функциональность нашей программы. В играх может понадобиться обрабатывать еще и такие сообщения:

- `wm_ActivateApp` — окно становится активным;
- `wm_LbuttonDown`/`wm_LButtonUp` — нажата/отпущена левая кнопка мыши;

- wm_RbuttonDown/wm_RButtonUp — нажата/отпущена правая кнопка мыши;
- wm_MouseMove — указатель мыши переместился над нашим окном;
- wm_KeyDown/wm_KeyUp — нажата/отпущена клавиша;
- wm_Char — нажата алфавитно-цифровая клавиша. Это сообщение не отменяет сообщений wm_KeyDown и wm_KeyUp.

При получении сообщения типа buttonDown/buttonUp, а также сообщения wm_MouseMove в младшем слове параметра LParam передается *x*-координата "горячей точки" указателя в координатах рабочей области нашего окна. Рабочая область — это как раз тот прямоугольник, который мы просили Windows закрашивать цветом Color_Window. В старшем слове LParam передается соответствующая *y*-координата.

Кроме того, в структуре Message для каждого сообщения, которое извлечено из очереди сообщений Windows (а как я говорил, сообщение оконной функции можно послать, минуя эту очередь), присутствуют такие полезные поля, как:

- Message.time — время отправки данного сообщения. Оно измеряется в миллисекундах с момента начала работы Windows. Такое же время дает функция GetTickCount;
- Message.pt.x и Message.pt.y — координаты указателя в момент отправки сообщения. Это уже экранные координаты, а не оконные.

Посмотрите в hlp-файлах описание сообщения wm_KeyDown. В параметрах к этому сообщению передается много интересной информации, которая может быть полезна для аркадной игры.

В листинге 3.2 вы видите текст этой второй программы. Обработка некоторых сообщений закомментирована, чтобы не было проблем с частым появлением модального окна MessageBox. Функция MessageBeep воспроизводит простой звук (если звуки Windows разрешены).

Листинг 3.2. Вторая Delphi-программа

```
{ Шаблон второй программы в Win API }
{$A-,B-,I-,R-,S-,X+,H+}
program Window2;
uses
 Windows,Messages,SysUtils;
const
  AppName = 'Window2';
var
  Message: TMsg;
```

```
  hWindow: HWnd;
  s : string;

function WindowProc(Window: HWnd; AMessage: UInt;
 WParam: WParam; LParam: LParam): LResult; stdcall; export;
begin
  WindowProc:=0;
  case AMessage of
(*wm_ActivateApp:
   begin
   MessageBox (hWindow, nil, 'wm_ActivateApp', MB_OK);
   end;
*)
wm_LButtonDown:
    begin
    s:='X='+IntToStr(LoWord(LParam))+#10+
       'Y='+IntToStr(HiWord(LParam))+#10+
       'Time='+IntToStr(Message.time)+' ms'+#10+
       'Screen X='+IntToStr(Message.pt.x)+#10+
       'Screen Y='+IntToStr(Message.pt.y);
    MessageBox (hWindow, PChar(s), 'wm_LButtonDown', MB_OK);
    end;
wm_LButtonUp:
    begin
    s:='x='+IntToStr(LoWord(LParam))+#10+
       'Y='+IntToStr(HiWord(LParam));
    MessageBox (hWindow, PChar(s), 'wm_LButtonUp', MB_OK);
    end;
wm_RButtonDown:
    begin
    s:='X='+IntToStr(LoWord(LParam))+#10+
       'Y='+IntToStr(HiWord(LParam));
    MessageBox (hWindow, PChar(s), 'wm_RButtonDown', MB_OK);
    end;
wm_RButtonUp:
    begin
```

```
    s:='X='+IntToStr(LoWord(LParam))+#10+
       'Y='+IntToStr(HiWord(LParam));
    MessageBox (hWindow, PChar(s), 'wm_RButtonUp', MB_OK);
    end;
(*wm_MouseMove:
    begin
    s:='X='+IntToStr(LoWord(LParam))+#10+
       'Y='+IntToStr(HiWord(LParam));
    MessageBox (hWindow, PChar(s), 'wm_MouseMove', MB_OK);
    end;
*)
wm_KeyDown:
   begin
   s:='Virtual key='+IntToStr(WParam)+#10+
      'Scan code='+IntToStr(LParam shr 16 and $FF);
   MessageBeep($FFFFFFFF);
   MessageBox(hWindow, PChar(s), 'wm_KeyDown', MB_OK);
   end;
(*wm_KeyUp:
   begin
   s:='Virtual key='+IntToStr(WParam);
   MessageBox(hWindow, PChar(s), 'wm_KeyUp', MB_OK);
   end;
*)
wm_Char:
   begin
   s:=IntToStr(WParam);
   MessageBox (hWindow, PChar(s), 'wm_Char', MB_OK);
   end;
wm_Destroy:
   begin
   MessageBox (hWindow, nil, 'wm_Destroy', MB_OK);
   PostQuitMessage(0);
   Exit;
   end;
  end; {case}
```

```
  WindowProc:=DefWindowProc(Window, AMessage, WParam, LParam);
end;

{ Регистрируем класс окна }
function WinRegister: Boolean;
var
  WindowClass: TWndClass;

begin
  WindowClass.Style:=cs_hRedraw or cs_vRedraw;
  WindowClass.lpfnWndProc:=@WindowProc;
  WindowClass.cbClsExtra:=0;
  WindowClass.cbWndExtra:=0;
  WindowClass.hInstance:=HInstance;
  WindowClass.hIcon:=LoadIcon(0, idi_Application);
  WindowClass.hCursor:=LoadCursor(0, idc_Arrow);
  WindowClass.hbrBackground:=HBrush(Color_Window);
  WindowClass.lpszMenuName:=nil;
  WindowClass.lpszClassName:=AppName;
  Result:=RegisterClass(WindowClass) <> 0;
end;

{ Создаем окно программы }
function WinCreate: HWnd;
var
  hWindow: HWnd;

begin
  hWindow:=CreateWindow(AppName,'Our Window',
   ws_OverlappedWindow,
   Integer(cw_UseDefault), Integer(cw_UseDefault),
   Integer(cw_UseDefault), Integer(cw_UseDefault),
   0, 0, HInstance, nil);
  if hWindow <> 0 then
```

```
   begin
   ShowWindow(hWindow, SW_SHOW);
   UpdateWindow(hWindow);
   end;
  Result:=hWindow;
end;

BEGIN
  // При повторном запуске программы активизируем запущенный экземпляр
  hWindow:=FindWindow(AppName,'Our Window');
  if hWindow > 0 then
    begin
    SendMessage(hWindow,WM_SETFOCUS,hWindow,0);
    ShowWindow(hWindow,SW_SHOWNORMAL);
    Exit;
    end;
  if not WinRegister then
   begin
   MessageBox(0,'WinRegister failed',nil,mb_Ok);
   Exit;
   end;
  hWindow:=WinCreate;
  if hWindow = 0 then
   begin
   MessageBox(0,'WinCreate failed',nil,mb_Ok);
   Exit;
   end;
  // Цикл обработки сообщений
  while GetMessage(Message, 0, 0, 0) do
   begin
   TranslateMessage(Message);
   DispatchMessage(Message);
   end;
  Halt(Message.wParam);
END.
```

3.3. Переходим в полноэкранный режим DirectDraw

В этом разделе мы рассмотрим переход в полноэкранный режим DirectDraw (full screen mode) с помощью встроенной в Windows библиотеки для работы с видеокартой, минуя систему GDI (Graphics Device Interface, интерфейс графических устройств). Мы будем использовать базовые функции DirectDraw версии 3, который шел с Windows 95 OSR1 от 1996 г., а это значит, что наша программа должна будет работать всегда и везде, в том числе под Windows NT 4.0 и далее. Мы будем пользоваться интерфейсными файлами ddraw.pas, d3dtypes.pas и dsound.pas, которые Blake Stone перевел из соответствующих заголовочных файлов для языка C. Эти файлы используют технологию OLE (Object Linking and Embedding). Существуют другие интерпретации этих файлов, которые не используют технологию OLE. Вы можете найти варианты этих интерфейсных файлов на сайте **www.delphi-jedi.org**. Также на этом сайте есть библиотека DelphiX — оболочка высокого уровня для DirectX и много чего еще интересного, и все это бесплатно и в исходных кодах.

Для компиляции этих заголовочных файлов нужен файл ole2.pas или ole2.dcu. В каталоге Delphi содержатся эти файлы, но ole2.dcu лежит отдельно от других скомпилированных модулей. Его надо переместить из подпапки Lib\Delphi2 к другим таким же файлам в подпапку Lib. Также можно положить его или исходный файл ole2.pas из подпапки Source\Rtl\Win в каталог с файлами игры.

Замечание

К сожалению, я не нашел автора заголовочных файлов ddraw.pas, d3dtypes.pas и dsound.pas, чтобы спросить у него разрешение на размещение этих файлов на компакт-диске. Издательство не размещает авторские файлы без согласия авторов. Недостающие файлы вы можете скачать по ссылке **http://www.gameintellect.com/bhv/headers.zip**.

Для продолжения работы берем за основу файл window1.dpr и добавляем в него такой код:

1. В список используемых модулей включаем ddraw.pas. Для трансляции этой программы понадобится также модуль d3dtypes.pas. Теперь эта строка выглядит так:

```
uses
 Windows, Messages, DDraw;
```

2. Добавим константы для размеров экрана:

```
SCRX = 800;
SCRY = 600;
```

3. Продолжаем секцию `var` в начале программы и вставляем описание таких глобальных переменных:

```
DirectDraw: IDirectDraw;
i: integer;
HALCaps, HELCaps: DDCaps;
Result: HResult;
```

Мы здесь создаем ссылку на DirectDraw-объект, методами и свойствами которого будем пользоваться. Типы переменных и значения констант также описаны в интерфейсном файле ddraw.pas.

4. В операторе `case` для обработки сообщений вставляем такую ветку:

```
wm_EraseBkgnd, wm_SetCursor:
  begin
    WindowProc:=1;
    Exit;
  end;
```

Этим мы говорим умалчиваемому обработчику сообщений Windows, что обрабатывать эти сообщения не нужно. (Впрочем, я не заметил, что это как-то влияет на работу программы.)

5. Перед циклом обработки сообщений вставляем такой код:

```
{ Создаем DirectDraw COM object }
Result:=DirectDrawCreate(NIL,DirectDraw,NIL);
if Result <> DD_OK then
  begin
    MessageBox(hWindow, 'Can not create DirectDraw object',
               nil, mb_Ok);
    Exit;
  end;
{ Устанавливаем exclusive cooperative level }
Result:=DirectDraw.SetCooperativeLevel (hWindow,
  DDSCL_EXCLUSIVE or DDSCL_FULLSCREEN);
if Result <> DD_OK then
  begin
    MessageBox(hWindow, 'SetCooperativeLevel failed', nil, mb_Ok);
    Exit;
  end;
```

```
{ Если нет аппаратного ускорения, то выход }
FillChar(HALCaps,SizeOf(DDCaps),0);
FillChar(HELCaps,SizeOf(DDCaps),0);
HALCaps.dwSize:=SizeOf(DDCaps);
HELCaps.dwSize:=SizeOf(DDCaps);
DirectDraw.GetCaps(HALCaps, HELCaps);
if HALCaps.dwCaps and DDCAPS_BLT <> DDCAPS_BLT then
  begin
    MessageBox(hWindow, 'BltFast does not work', nil, mb_Ok);
    Exit;
  end;
{ Устанавливаем графический режим True Color с 32 или 24 битами
  на точку }
for i:=4 downto 3 do
  begin
    Result:=DirectDraw.SetDisplayMode(800,600,i*8);
    if Result = DD_OK then break;
    if (Result <> DD_OK) and (i = 3) then
      begin
        MessageBox(hWindow, 'Can not set display mode', nil, mb_Ok);
        Exit;
      end;
  end;
```

Здесь мы вначале создаем DirectDraw-объект, затем устанавливаем эксклюзивный режим работы, чтобы наша программа монопольно владела аппаратурой видеокарты. Это нужно для работы в полноэкранном режиме. Далее проверяем, поддерживает ли видеокарта аппаратное ускорение `DirectDraw`. Если нет, то мы для рисования не сможем воспользоваться функцией `BltFast`, которая работает в сотни раз быстрее `BitBlt` и функций GDI. На всех современных видеокартах должно быть аппаратное ускорение `DirectDraw`. Следующий шаг — установка графического режима True Color с разрешением экрана 800×600 экранных точек. Мы пробуем вначале установить режим с 32 битами на точку, а в случае неудачи (чего не должно быть) с 24 битами на точку. Мне встречались видеокарты, которые поддерживали оба режима, но аппаратное ускорение было лишь в режиме 32 бита на точку. После этого, вообще говоря, в обработку сооб-

щения wm_Destroy надо вставить команды восстановления предыдущего графического режима и освобождения объекта DirectDraw:

```
if Assigned (DirectDraw) then
  begin
    Directdraw.RestoreDisplayMode;
    DirectDraw.FlipToGDISurface;
    { Освобождаем DirectDraw COM object }
    if Assigned (DirectDraw) then DirectDraw.Release;
    DirectDraw:=NIL;
  end;
```

Хотя, если этого не сделать, Windows все равно сделает это все за нас.

Если запустить эту программу, то вы увидите окно, распахнутое на весь экран, размеры которого 800×600 точек. Будет присутствовать курсор в виде стрелки. При переключении <Alt>+<Tab> и восстановлении окна программы Windows отрисует заголовок окна и закрасит его в серый (или установленный у вас) цвет. Это окно можно уменьшить в размерах, тогда за ним покажется экран GDI рабочего стола и другие окна. Если щелкнуть вне нашего окна, оно свернется на панель задач. Это происходит из-за эксклюзивного режима работы, установленного нами. В нем либо окно находится в фокусе, либо оно сворачивается в кнопку. Интересно задать разрешение 320×200 точек и посмотреть, что получится. У меня панель задач отображается нормально.

Но такое окно, которое имеет заголовок и рамку, нам в полноэкранном приложении не нужно. Нам нужен доступ ко всему экрану как в DOS, ничего постороннего на нем быть не должно. Для этого заменим вызов CreateWindow для создания окна в функции WinCreate на такой:

```
hWindow:=CreateWindowEx(WS_EX_TOPMOST, AppName, AppName,
   ws_PopUp, 0, 0, GetSystemMetrics(SM_CXSCREEN),
   GetSystemMetrics(SM_CYSCREEN), 0, 0, HInstance, nil);
```

Такой вызов создает окно на весь экран, которое всегда расположено наверху. Имя класса и текст окна берутся из AppName. С учетом этого вставим сразу после BEGIN такой код для предотвращения повторного запуска программы:

```
// При повторном запуске программы активизируем запущенный экземпляр
hWindow:=FindWindow(AppName, AppName);
if hWindow > 0 then
  begin
    SendMessage(hWindow, WM_SETFOCUS, hWindow, 0);
```

```
    ShowWindow(hWindow, SW_SHOWNORMAL);
    Exit;
  end;
```

Для порядка в функции `WinRegister` заменим строку

```
WindowClass.hbrBackground:=HBrush(Color_Window);
```

на строку

```
WindowClass.hbrBackground:=HBrush(GetStockObject(NULL_BRUSH));
```

которая создает для закраски фона специальную кисть с целью отмены закраски. Напомню, что мы отменили закраску фона функцией `DefWindowProc` в обработке сообщения `wm_EraseBkgnd`.

Сразу перед циклом обработки сообщений введите строки

```
// Системный курсор нам не нужен, прячем его
ShowCursor(FALSE);
```

Эта программа находится в каталоге "chapter3\Переход в полноэкранный режим" на компакт-диске. Если ее запустить, то мы увидим черный экран. Правда после переключения на другую задачу и возврат у меня иногда внизу экрана появляется серая линия от панели задач. (У меня эта панель автоматически скрывается.) Для завершения программы нажмите комбинацию клавиш <Alt>+<F4>.

Замечание

Кроме достоинств у полноэкранных программ есть также один недостаток: их трудно отлаживать. Если вы попытаетесь выполнять такую программу по шагам или установите точку прерывания, то получите замерзший экран программы, т. к. переключения на экран отладчика не будет, программа по-прежнему будет монопольно владеть экраном. В этом случае останется только перегрузить систему, а в Windows 9*x* в этом случае оставалось только нажать на кнопку Reset.

Для обычного домашнего пользователя остаются методы, какими пользовались наши отцы и деды: вывод значений переменных в файл или на экран. Но вам не придется подключать фотосчитыватель или загрузчик с перфокарт для старта системы. Это радует. В общем, для написания полноэкранных программ требуется опыт программирования и понимание того, как будет работать созданный вами код.

3.4. Создание поверхностей DirectDraw

Но это еще не все. Нам нужна хотя бы одна поверхность DirectDraw, которая ассоциируется с экраном, чтобы было на чем рисовать.

Поверхности бывают следующих видов:

- *первичная поверхность* (primary surface), которая ассоциируется с экраном. То, что выводится на эту поверхность, немедленно отображается на экране;
- *внеэкранная поверхность* (offscreen surface), которая также расположена в видеопамяти, но не отображается на экране. Таких поверхностей можно создать несколько (пока очередная попытка создания не вернет ошибку). Эти поверхности обычно хранят спрайты и изображения, которые с помощью метода `BltFast` объекта поверхности можно очень быстро копировать с одной поверхности на любую другую. Также на внеэкранной поверхности мы будем хранить фон под нашим красивым указателем для его перерисовки по сообщению `wm_MouseMove`;
- *поверхность*, которая находится *в оперативной памяти* (memory surface). Для программиста она в принципе не отличается от offscreen surface, но аппаратного ускорения для работы с этой поверхностью может и не быть;
- *комплексная поверхность переключения* (flipping surface). Это экранная поверхность и одна или несколько внеэкранных поверхностей, между которыми можно делать переключение. Используется в игре, в которой много движущихся спрайтов. Здесь технология такая: вместо отрисовки всего на внеэкранной поверхности и копирования ее содержимого на экранную поверхность все рисуется, но копирование не делается, а вместо этого применяется метод `Flip` поверхности, в результате чего экранной поверхностью становится та, на которой готовилось изображение. В видеоконтроллере есть регистр начала отображения, который содержит адрес видеопамяти, начиная с которого содержимое отображается на экране. Стоит в него записать другое значение, как изображение моментально изменится (после обратного хода лучей электронных пушек по экрану). Программа может при каждом кадре копировать фон во весь экран с одной поверхности на другую, потом копировать поверх него все спрайты и изображения, которых могут быть сотни, и все это будет происходить с частотой 85 и выше кадров в секунду. Вообще-то в DirectDraw предусмотрено создание вторичной поверхности в ОЗУ, если в видеопамяти места недостаточно. В этом случае переключение методом `Flip` будет только эмулировать переключение страниц, а на самом деле содержимое вторичной поверхности будет копироваться на экранную поверхность. Нас такой вариант не интересует, потому что у современных видеокарт хватает памяти не на одну вторичную поверхность.

Замечание

В комплект Windows входит средство диагностики DirectX — программа dxdiag.exe. В ней на вкладке **Если ничего не помогло** есть кнопка **Частота...**, с помощью которой вы можете изменить частоту кадров в режиме DirectDraw.

(Эта частота по умолчанию может быть установлена в 60 Гц, а этого мало). Также установку этой частоты может поддерживать сервисная программа, поставляемая вместе с видеокартой.

На основе программы window1.dpr из предыдущего раздела продолжим работать в DirectDraw.

В секцию `var` основной программы добавьте следующие строки:

```
DDSurfaceDesc: ddraw.DDSurfaceDesc;
ColorKey: ddraw.DDColorKey;
PrimSurf: IDirectDrawSurface;
```

Это структура описания поверхности, переменная для задания цвета прозрачности и объект первичной (экранной) поверхности.

Перед строкой для скрытия системного курсора вставьте следующие строки:

```
{ Заполняем DirectDrawSurface descriptor для поверхности }
FillChar(DDSurfaceDesc, SizeOf(DDSurfaceDesc), 0);
with DDSurfaceDesc do
  begin
    dwSize:=SizeOf(DDSurfaceDesc);
    dwFlags:=DDSD_CAPS;
    ddSCaps.dwCaps:=DDSCAPS_PrimarySurface;
  end;
Result:=DirectDraw.CreateSurface(DDSurfaceDesc, PrimSurf, NIL);
if Result <> DD_OK then Exit;
{ Устанавливаем ключевые цвета поверхности в 0 }
ColorKey.dwColorSpaceLowValue:=0;
ColorKey.dwColorSpaceHighValue:=0;
Result:=PrimSurf.SetColorKey(DDCKEY_SRCBLT, ColorKey);
if Result <> DD_OK then Exit;
```

Я устанавливаю ключевой цвет (он же — цвет прозрачности) для всех поверхностей в `0` (черный). Вообще-то в описании этой функции сказано, что может быть диапазон прозрачных цветов, но это по-моему не работает: можно задать лишь один ключевой цвет. Теперь мы можем выводить на поверхность спрайты с аппаратным ускорением, черные точки в них будут пропускаться и сквозь них будет виден фон. А как же быть с черными точками самого спрайта? Их можно заменить на темный, но ненулевой цвет, скажем, байт для голубой составляющей установить в `1`. Такой голубой цвет никто не отличит от черного.

Ширина всех поверхностей должна быть такой, каков размер экрана в точках, а высота внеэкранных поверхностей может быть и меньше высоты экрана.

Аналогично создаются поверхности других типов. Для этого в структуре `DDSurfaceDesc` указывается их тип.

В нашем примере в случае каких-либо ошибок осуществляется простой выход из программы, в реальной игре это все будет находиться в специальном модуле, и перед выходом из программы мы будем освобождать ресурсы, хотя Windows после завершения программы все равно не оставит их занятыми.

Пришла пора вывести на экран какое-нибудь изображение. Объект поверхности имеет дисплейный контекст, и можно его получить методом `GetDC` и рисовать на этой поверхности с помощью функций GDI. Для примера нарисуем эллипс. Для хранения handle полученного дисплейного контекста заведем глобальную переменную `DC: HDC`. Перед циклом обработки сообщений вставьте такие строки:

```
PrimSurf.GetDC(DC);
Ellipse(DC, 100, 100, 300, 300);
PrimSurf.ReleaseDC(DC);
```

Запустив эту программу, мы увидим белый круг, потому что по умолчанию в контексте устройства установлен белый цвет для кисти. А что произойдет, если мы переключимся на другие окна, а наше окно свернем? После восстановления окна нашей программы мы увидим черный экран и, возможно, линию от панели задач. "Куда делся круг? Верните немедленно!" — воскликнет взволнованный читатель, выключив `WinAmp`. "А кто тебе должен его рисовать? — спросят с ухмылкой Windows и Delphi. — Это не визуальное программирование, где восстановление изображения окна берет на себя Delphi". Поэтому не спешите разбивать стекло молотком или делать компот из жидких кристаллов. Здесь все делает программа, а Windows только посылает ей сообщения типа `wm_EraseBkgnd` и `wm_Paint`, которые мы не обрабатываем. Но не торопитесь в обработчик сообщения `wm_Paint` ставить рисование этого проклятого круга, вы получите ошибку. Дело в том, что поверхности после деактивизации окна теряются, и память, распределенная для них, освобождается. Если немного подумать, то это и понятно: ведь видеокарта с ее памятью и аппаратурой ускорителя является общим на все программы ресурсом. Поэтому его надо освобождать для других программ. Поверхности, создаваемые в ОЗУ, продолжают хранить то, что на них было нарисовано, но лучше на это не полагаться, а перерисовывать все поверхности.

Вообще-то наша полноэкранная игра тоже не будет обрабатывать эти сообщения, как и рисовать методами GDI (кроме вывода BMP-файла на поверхность), она будет пользоваться методами поверхностей. При попытке

рисования методами поверхности мы получим сообщение DDERR_SURFACELOST, — это и означает, что наши поверхности надо пересоздать и восстановить все, что на них было. Кроме того, в игре в цикле обработки сообщений мы будем проверять, не потеряны ли поверхности вызовом метода IsLost первичной поверхности. Иначе программа не восстановит изображения, пока ей не понадобится что-нибудь нарисовать, и игрок будет некоторое время наблюдать черный экран. Если результат не равен DD_OK (это всего лишь константа 0, описанная в файле ddraw.pas), то поверхности DirectDraw потеряны. Но бесполезно их восстанавливать, если наша программа не активна. Метод Restore будет возвращать ошибку. Полноэкранная игра всегда должна знать, активна она или нет. Для этого заведем глобальную логическую переменную bIsActive. В оператор case обработки сообщений добавим такую ветку:

```
wm_ActivateApp:
    { Запоминаем в bIsActive активно ли наше приложение.
      Оно может быть свернуто в кнопку на панели задач }
    bIsActive:=boolean(wParam);
```

Теперь эта переменная после получения первого сообщения wm_ActivateApp будет иметь значение TRUE тогда и только тогда, когда наша программа активна. А цикл обработки сообщений перепишем так:

```
while GetMessage(Message, 0, 0, 0) do
  begin
    if bIsActive and (PrimSurf.IsLost <> DD_OK) then
      begin
        if PrimSurf.Restore <> DD_OK then Halt;
        PrimSurf.GetDC(DC);
        Ellipse(DC, 100, 100, 300, 300);
        PrimSurf.ReleaseDC(DC);
      end;
    TranslateMessage(Message);
    DispatchMessage(Message);
  end;
```

Сейчас я еще покажу, как можно напрямую обращаться к памяти поверхностей DirectDraw, и этот пример будет закончен. Вообще, это не рекомендуется, но я это использовал в игре, чтобы выводить текст своим растровым шрифтом, заданным цветом и с тенью для большего удобства чтения. Эта процедура, конечно, не такая быстрая, как работа через аппаратный ускоритель, особенно чтение из видеопамяти. Для того чтобы поверхность была доступна для чтения/записи, надо выполнить ее метод Lock (залочить). По-

сле того как мы что-то запишем и/или прочитаем из этой поверхности, следует вызвать метод `UnLock`. Пока поверхность заблокирована, нельзя обращаться к ней методами этой поверхности — рисовать на ней, копировать прямоугольники и т. д. Еще неплохо бы знать формат точки на поверхности — сколько в ней байтов, где байт, отвечающий за интенсивность голубого, зеленого и красного цвета. Если в точке 4 байта, то старший байт не используется, если 3, то все они в деле. Кроме того, разные производители видеокарт по-разному хранят в этих байтах значения цветов RGB — у одних в младшем байте может быть голубой цвет, а у других — красный. Как сговорились, чтобы усложнять программистам жизнь!

Мы не пользуемся графическим режимом с двумя байтами на точку, но и там та же картина: у одних видеокарт используются все 16 битов — по 5 на красный и голубой и 6 на зеленый цвет, у других — старший бит не используется, а в младших пятнадцати хранятся три пятерки битов RGB... или BGR, голубой и красный переставлены местами. В палитровом режиме с 8 битами на точку аппаратное ускорение особенно заметно, но этот режим тоже устарел ввиду малого числа цветов. Хотя вы можете его использовать, но не забывайте также восстанавливать палитру соответствующими методами DirectDraw после восстановления поверхностей.

Для получения формата точки существует метод поверхности `GetPixelFormat`, смотрите также структуру `DDPIXELFORMAT` в файле ddraw.pas. Но мы в игре, которую скоро начнем писать, сделаем по-другому: где-то в уголке BMP-файла со вспомогательными изображениями нарисуем точки всеми цветами, которые будем использовать в шрифтах. Затем прочитаем их в программе с поверхности. Число байтов на точку можно узнать в цикле установки графического режима:

```
for i:=4 downto 3 do
```

Заведем глобальную переменную `bpp` типа `integer` (от англ. *bytes per pixel* — число байтов на точку) и перепишем этот цикл так:

```
for i:=4 downto 3 do
  begin
    Result:=DirectDraw.SetDisplayMode(800,600,i*8);
    if Result = DD_OK then
      begin
        bpp:=i;
        break;
      end;
    if (Result <> DD_OK) and (i = 3) then
      begin
```

```
        MessageBox(hWindow, 'Can not set display mode', nil, mb_Ok);
        Exit;
      end;
  end;
```

Если теперь запустить этот пример и свернуть его окно, а потом опять развернуть, то вновь появится изображение так необходимого нам белого круга. Заработало!

Если теперь в цикл обработки сообщений вставить этот код для рисования эллипса, то экран с эллипсом будет восстанавливаться после переключения на другую задачу:

```
// Цикл обработки сообщений
while GetMessage(Message, 0, 0, 0) do
  begin
    if bIsActive and (PrimSurf.IsLost <> DD_OK) then
      begin
        if PrimSurf.Restore <> DD_OK then Halt;
        PrimSurf.GetDC(DC);
        Ellipse(DC, 100, 100, 300, 300);
        PrimSurf.ReleaseDC(DC);
      end;
    TranslateMessage(Message);
    DispatchMessage(Message);
  end;
```

В игре для этой цели будет специальная процедура `DrawSurfaces`.

Эта программа находится в каталоге "chapter3\Создание поверхностей DDraw" на компакт-диске.

3.5. Запись точек на поверхность DirectDraw

Продолжим программу из предыдущего раздела и заполним экран точками со случайным цветом. Продолжим секцию `var` программы и добавим в нее такие строки:

```
psurf: pointer;
pitch,j: integer;
pb: ^byte;
```

Здесь:

- `psurf` — указатель на начало памяти экранной поверхности;
- `pitch` — длина физической скан-линии в байтах;
- `pb` и `j` — вспомогательные переменные.

Надо заметить, что если ширина экрана 800 точек и в каждой точке 4 байта, то это еще не значит, что вторая скан-линия начинается через 3200 байтов от начала первой. Дело в том, что на некоторых видеокартах драйверы создают за концом каждой скан-линии несколько десятков байтов в качестве какого-то кэша для своего собственного использования. Прикладная программа не должна писать в этот кэш. При блокировке поверхности программа получает в качестве ответа не только указатель на начало поверхности, но и физическую длину скан-линии в байтах, также это можно назвать шагом между одноименными точками соседних скан-линий. При вызове метода `UnLock` программа передает этот указатель `psurf`, чтобы вернуть поверхность в распоряжение DirectDraw. Вообще-то метод `Lock` также предусматривает блокировку только части поверхности прямоугольной формы, но мы эту возможность не будем использовать.

Так как мы часто будем использовать блокировку и разблокировку поверхностей, то напишем для этого специальные функции `SurfLock` и `SurfUnLock`. В листинге 3.3 представлен текст функции `SurfLock`.

Листинг 3.3. Блокировка поверхности

```
{ Блокировка поверхности surf }
function SurfLock(surf: IDirectDrawSurface;
 surftype, pixx, pixy: integer): pointer;
var
  DDSurfDesc: ddraw.DDSurfaceDesc;
  DDResult: dword;

begin
  // Очищаем структуру для поверхности
  FillChar (DDSurfDesc, sizeof(DDSurfDesc), 0);
  with DDSurfDesc do
    begin
    // Передаем ее размер
    dwSize:=sizeof(DDSurfDesc);
    dwFlags:=DDSD_CAPS or DDSD_HEIGHT or DDSD_WIDTH;
```

```
    // Для каждого типа поверхности свой параметр
    case surftype of
PRIMARYSURF:
      ddsCaps.dwCaps:=DDSCAPS_FRONTBUFFER;
OFFSCREENSURF,FLIPSURF:
      ddsCaps.dwCaps:=DDSCAPS_OFFSCREENPLAIN;
MEMORYSURF:
      ddsCaps.dwCaps:=DDSCAPS_OVERLAY;
    end; {case}
    dwWidth:=pixx;
    dwHeight:=pixy;
    end;
  // Блокируем для чтения и записи
  DDResult:=surf.Lock(nil, DDSurfDesc,
   DDLOCK_SURFACEMEMORYPTR or DDLOCK_WRITEONLY or
   DDLOCK_READONLY or DDLOCK_WAIT, 0);
  if DDResult <> DD_OK then Halt;
  // Возвращаем указатель на поверхность
  Result:=DDSurfDesc.lpSurface;
  { Также запоминаем шаг между началами соседних скан-линий в байтах }
  pitch:=DDSurfDesc.lpitch;
end; {SurfLock}
```

На входе она получает такие параметры:

- указатель на объект поверхности;
- тип поверхности (экранная, внеэкранная, в памяти);
- размер поверхности в точках по горизонтали и вертикали.

Еще одно замечание: методы вывода на поверхность возвращают управление в программу сразу, но аппаратура видеокарты еще некоторое время может быть занята отрисовкой. Поэтому некоторые методы поверхности могут возвращать код ошибки `DDERR_WASSTILLDRAWING`, что означает, аппаратура занята рисованием. При получении этой ошибки надо повторять вызов данной функции. В функции `SurfLock` при блокировке поверхности мы передаем бит `DDLOCK_WAIT`, чтобы это ожидание, когда аппаратура освободится, было внутри метода `Lock`. Поэтому обрабатывать эту ошибку нам не надо.

Процедура разблокировки поверхности выглядит намного проще (листинг 3.4).

Листинг 3.4. Разблокировка поверхности

```
procedure SurfUnLock(surf: IDirectDrawSurface; psurf: pointer);
var
 res: dword;

begin
  res:=surf.UnLock(psurf);
  if (res <> DD_OK) and (res <> DDERR_NOTLOCKED) then Halt;
end; {SurfUnlock}
```

Она получает указатель на память поверхности, который возвращает функция `SurfLock`. Если мы получаем ошибку `DDERR_NOTLOCKED`, то мы пытались разблокировать незаблокированную поверхность. В случае другой ошибки останавливаем нашу учебную программу.

После того как в основной программе мы нарисовали эллипс, вставим такой код:

```
// Заполняем экранную поверхность случайными точками
Sleep(1000);
psurf:=SurfLock(PrimSurf, SCRX, SCRY, PRIMARYSURF);
pb:=psurf;
// Цикл по числу скан-линий
for i:=0 to SCRY-1 do
  begin
    // Цикл по i-ой скан-линии
    for j:=0 to SCRX-1 do
      begin
        // Выводим 3 байта цвета
        pb^:=Random(256);
        Inc(dword(pb));
        pb^:=Random(256);
        Inc(dword(pb));
        pb^:=Random(256);
        Inc(dword(pb));
```

```
        // Переходим к следующей точке
        Inc(dword(pb), bpp-3);
      end;
    // Переходим к следующей скан-линии
    Inc(dword(pb), pitch-bpp*SCRX);
  end;
  SurfUnLock(PrimSurf, psurf);
```

Здесь мы присваиваем переменной `pb` адрес экранной поверхности и используем эту переменную для вывода точек побайтно (хотя можно было бы писать в видеопамять словами и двойными словами, лишь бы не выйти за границы поверхности). При переходе к следующей точке учитываем, что она может содержать 3 или 4 байта. При переходе к следующей скан-линии учитываем физический размер скан-линии в байтах. Чтение точек происходит аналогично.

Если вы запустите эту программу, то увидите, что заполнение экрана происходит практически мгновенно. После нажатия комбинации клавиш <Alt>+ +<Tab> восстанавливается только эллипс, но я думаю, что это уже понятно читателю. Смотрите этот пример в каталоге "chapter3\Запись точек на поверхность DDraw".

Для полноты изложения теории работы в DirectDraw нужно еще привести пример переключения поверхностей и вывод на них спрайтов, а также вывод BMP-файла на поверхность. Исходный текст этого примера я приведу полностью, потому что он во многом отличается от предыдущих примеров.

3.6. Переключение поверхностей DirectDraw

В листинге 3.5 вы видите полный текст программы, выводящей 100 спрайтов и переключающей поверхности. Некоторые участки кода оформлены в виде функций, а данные — в виде структур для большего удобства и предотвращения дублирования. Пробегитесь по нему взглядом, а после этого обсудим некоторые детали.

Листинг 3.5. Программа вывода спрайтов с переключением поверхностей DirectDraw

```
{ Переключение поверхностей DirectDraw }
{$A-,B-,I-,R-,S-,X+,H+}
```

```
program Window1;
uses
 Windows, Messages, DDraw;
const
  AppName = 'Window1';
  { Типы поверхностей }
  // Экранная
  PRIMARYSURF = 0;
  // Экранная с переключением страниц
  FLIPSURF = 1;
  // Внеэкранная видеопамять
  OFFSCREENSURF = 2;
  // Поверхность в оперативной памяти
  MEMORYSURF = 3;
  // Максимальное число используемых поверхностей
  MAXSURF = 3;
  // Размеры экрана
  SCRX = 800;
  SCRY = 600;
  // Размеры спрайта
  SPRWIDTH = 105;
  SPRHEIGHT = 81;
  // Число спрайтов
  SPRNUM = 100;
var
  Message: TMsg;
  hWindow: HWnd;
  DirectDraw: IDirectDraw;
  i,bpp: integer;
  HALCaps, HELCaps: DDCaps;
  Result: HResult;
  bIsActive: boolean;
  path: string;
  tick,lasttick: TLargeInteger;
  SurfRec: array[1..MAXSURF] of record
    surf: IDirectDrawSurface;
```

```
    psurf: pointer;
    surftype,pixx,pixy,pitch: integer;
    exist: boolean;
    end;
  sprites: array[0..SPRNUM-1] of record
   x,y,dx,dy,width,height: integer;
   end;

procedure DestroyAllSurf; forward;
procedure DrawSurf; forward;

function WindowProc(Window: HWnd; AMessage: UInt;
 WParam: WParam; LParam: LParam): LResult; stdcall; export;
begin
  WindowProc:=0;
  case AMessage of
wm_ActivateApp:
    { Запоминаем в bIsActive, активно ли наше приложение.
      Оно может быть свернуто в кнопку на панели задач }
    bIsActive:=boolean(wParam);
wm_EraseBkgnd, wm_SetCursor:
   begin
   WindowProc:=1;
   Exit;
   end;
wm_Destroy:
   begin
   { Восстанавливаем предыдущий графический режим и освобождаем объект
     DirectDraw }
   DestroyAllSurf;
   PostQuitMessage(0);
   Exit;
   end;
  end; {case}
  WindowProc:=DefWindowProc(Window, AMessage, WParam, LParam);
end;
```

```
{ Регистрируем класс окна }
function WinRegister: Boolean;
var
  WindowClass: TWndClass;

begin
  WindowClass.Style:=cs_hRedraw or cs_vRedraw;
  WindowClass.lpfnWndProc:=@WindowProc;
  WindowClass.cbClsExtra:=0;
  WindowClass.cbWndExtra:=0;
  WindowClass.hInstance:=HInstance;
  WindowClass.hIcon:=LoadIcon(0, idi_Application);
  WindowClass.hCursor:=LoadCursor(0, idc_Arrow);
  WindowClass.hbrBackground:=HBrush(GetStockObject(NULL_BRUSH));
  WindowClass.lpszMenuName:=nil;
  WindowClass.lpszClassName:=AppName;
  Result:=RegisterClass(WindowClass) <> 0;
end;

{ Создаем окно программы }
function WinCreate: HWnd;
var
  hWindow: HWnd;

begin
  hWindow:=CreateWindowEx(WS_EX_TOPMOST, AppName, AppName,
   ws_PopUp, 0, 0, GetSystemMetrics(SM_CXSCREEN),
   GetSystemMetrics(SM_CYSCREEN), 0, 0, HInstance, nil);
  if hWindow <> 0 then
   begin
   ShowWindow(hWindow, SW_SHOW);
   UpdateWindow(hWindow);
   end;
  Result:=hWindow;
end;
```

```
{ Создание поверхности nsurf размером pixx, pixy точек. Если video=
  TRUE, то поверхность создается в видеопамяти. Если video=flip=TRUE
  (только для экранной поверхности), то создается комплексная
  поверхность номер nsurf вместе с поверхностью переключения nsurf+1,
  иначе создается одна поверхность переднего плана nsurf. При успешном
  завершении возвращается DD_OK, иначе ничего не производится и
  возвращается ошибка с кодом из DirectDraw }
function CreateSurf(hWindow: hWnd; nsurf,pixx,pixy,surftype: integer;
  flip: boolean): HResult;
VAR
  DDSurfaceDesc: ddraw.DDSurfaceDesc;
  ColorKey: ddraw.DDColorKey;

begin
  Result:=100;
  if (nsurf > MAXSURF) or flip and (surftype = PRIMARYSURF) and
    (nsurf > MAXSURF-1) then Exit;
  { Заполняем DirectDrawSurface descriptor для заданной поверхности }
  FillChar(DDSurfaceDesc,sizeof(DDSurfaceDesc),0);
  with DDSurfaceDesc do
    begin
    dwSize:=sizeof(DDSurfaceDesc);
    dwFlags:=DDSD_CAPS or DDSD_HEIGHT or DDSD_WIDTH;
    if (surftype = OFFSCREENSURF) or (surftype = FLIPSURF) then
      ddsCaps.dwCaps:=DDSCAPS_OFFSCREENPLAIN;
    if surftype = MEMORYSURF then
      ddSCaps.dwCaps:=DDSCAPS_SYSTEMMEMORY;
    if surftype = PRIMARYSURF then
      begin
      dwFlags:=DDSD_CAPS;
      ddSCaps.dwCaps:=DDSCAPS_PRIMARYSURFACE;
      if flip then
        { Создаем поверхность переключения }
        begin
        dwFlags:=DDSD_CAPS or DDSD_BACKBUFFERCOUNT;
        ddSCaps.dwCaps:=DDSCAPS_COMPLEX or DDSCAPS_FLIP or
```

```
        DDSCAPS_PRIMARYSURFACE;
      dwBackBufferCount:=1;
      end;
    end else
      begin
      dwWidth:=pixx;
      dwHeight:=pixy;
      end;
  Result:=DirectDraw.CreateSurface(DDSurfaceDesc,
    surfrec[nsurf].surf,NIL);
  if Result <> DD_OK then Exit;
  if flip and (surftype = PRIMARYSURF) then
    { Проверяем создание поверхности переключения }
    begin
    DDSCaps.dwCaps:=DDSCAPS_BACKBUFFER;
    if surfrec[nsurf].surf.GetAttachedSurface(DDSCaps,
      surfrec[nsurf+1].surf) <> DD_OK then Exit;
    end;
  end;
{ Устанавливаем ключевой цвет поверхности в 0 }
ColorKey.dwColorSpaceLowValue:=0;
ColorKey.dwColorSpaceHighValue:=0;
Result:=surfrec[nsurf].surf.SetColorKey(DDCKEY_SRCBLT,ColorKey);
if Result <> DD_OK then Exit;
surfrec[nsurf].exist:=TRUE;
surfrec[nsurf].surftype:=surftype;
surfrec[nsurf].pixx:=pixx;
surfrec[nsurf].pixy:=pixy;
if flip and (surftype = PRIMARYSURF) then
  begin
  Result:=surfrec[nsurf+1].surf.SetColorKey(DDCKEY_SRCBLT,ColorKey);
  if Result <> DD_OK then Exit;
  surfrec[nsurf+1].exist:=TRUE;
  surfrec[nsurf+1].surftype:=FLIPSURF;
  surfrec[nsurf+1].pixx:=pixx;
```

```
    surfrec[nsurf+1].pixy:=pixy;
    end;
end; {CreateSurf}

// Уничтожает все поверхности и объект DirectDraw
procedure DestroyAllSurf;
VAR
  i: integer;

begin
  if Assigned(DirectDraw) then
   begin
   DirectDraw.FlipToGDISurface;
   Directdraw.RestoreDisplayMode;
   end;
  for i:=1 to MAXSURF do
    if surfrec[i].exist and Assigned(surfrec[i].surf) and
      (surfrec[i].surftype <> FLIPSURF) then
      begin
      surfrec[i].surf.release;
      surfrec[i].exist:=FALSE;
      end;
  { Освобождаем DirectDraw COM object }
  if Assigned(DirectDraw) then DirectDraw.Release;
  DirectDraw:=NIL;
end;

{  Восстановление потерянной памяти поверхностей }
function RestoreSurfaces: HResult;
VAR
  i: integer;

begin
  Result:=DD_OK;
  for i:=1 to MAXSURF do
    if surfrec[i].exist and (surfrec[i].surftype <> FLIPSURF) then
```

```
      begin
      Result:=surfrec[i].surf.Restore;
      if Result <> DD_OK then Halt;
      end;
  // Загружаем картинку в 3-ю поверхность
  LoadBitMap(0,0,0,0,3,path+'diamond.bmp');
  DrawSurf;
end; {RestoreSurfaces}

// Для облегчения обработки ошибок
function RespHandler(DDResult: HResult): boolean;
begin
  case DWord(DDResult) of
    DD_OK: Result:=TRUE;
    DDERR_SURFACELOST: Result:=RestoreSurfaces <> DD_OK;
     else Result:=DWord(DDResult) <> DDERR_WASSTILLDRAWING;
  end;
end;

{ Блокировка поверхности nsurf }
function SurfLock(nsurf: integer): pointer;
VAR
  DDSurfDesc: ddraw.DDSurfaceDesc;

begin
  FillChar(DDSurfDesc,sizeof(DDSurfDesc),0);
  with DDSurfDesc do
    begin
    dwSize:=sizeof(DDSurfDesc);
    dwFlags:=DDSD_CAPS or DDSD_HEIGHT or DDSD_WIDTH;
    case surfrec[nsurf].surftype of
PRIMARYSURF:            ddsCaps.dwCaps:=DDSCAPS_FRONTBUFFER;
OFFSCREENSURF,FLIPSURF: ddsCaps.dwCaps:=DDSCAPS_OFFSCREENPLAIN;
MEMORYSURF:             ddsCaps.dwCaps:=DDSCAPS_OVERLAY;
    end; {case}
    dwWidth:=surfrec[nsurf].pixx;
```

```
    dwHeight:=surfrec[nsurf].pixy;
    end;
  repeat
  until RespHandler(surfrec[nsurf].surf.Lock
    (nil,DDSurfDesc,DDLOCK_SURFACEMEMORYPTR or DDLOCK_WRITEONLY or
    DDLOCK_READONLY or DDLOCK_WAIT,0));
  Result:=DDSurfDesc.lpSurface;
  surfrec[nsurf].psurf:=Result;
  surfrec[nsurf].pitch:=DDSurfDesc.lpitch;
end; {SurfLock}

{ Разблокировка поверхности nsurf }
procedure SurfUnLock(nsurf: integer);
begin
  repeat
  until RespHandler(surfrec[nsurf].surf.UnLock(surfrec[nsurf].psurf));
end; {SurfUnlock}

// Получение объекта TRect по координатам прямоугольника
function Rect(ALeft,ATop,ARight,ABottom: Integer): TRect;
VAR
 ARect: TRect;

begin
  ARect.left:=Aleft;
  ARect.Top:=ATop;
  ARect.right:=Aright;
  ARect.bottom:=ABottom;
  Result:=ARect;
end; {Rect}

{ Копирование прямоугольника с координатами xs1, ys1, xs2, ys2
  в координаты xd, yd с поверхности номер nsurf1 на nsurf2 }
procedure CopyRec(xs1,ys1,xs2,ys2,xd,yd,nsurf1,nsurf2: integer);
VAR
  ARect: TRect;
```

```
begin
  ARect:=Rect(xs1,ys1,xs2+1,ys2+1);
    repeat
    until RespHandler(surfrec[nsurf2].surf.Bltfast
      (xd,yd,surfrec[nsurf1].surf,ARect,DDBLTFAST_NOCOLORKEY));
end; {CopyRec}

{ Копирование спрайта с координатами xs1, ys1, xs2, ys2
  в координаты xd, yd с поверхности номер nsurf1 на nsurf2 }
procedure CopySpr(xs1,ys1,xs2,ys2,xd,yd,nsurf1,nsurf2: integer);
VAR
  ARect: TRect;

begin
  ARect:=Rect(xs1,ys1,xs2+1,ys2+1);
    repeat
    until RespHandler(surfrec[nsurf2].surf.Bltfast
      (xd,yd,surfrec[nsurf1].surf,ARect,DDBLTFAST_SRCCOLORKEY));
end; {CopySpr}

// Переключение экранной поверхности
procedure Flip;
begin
  // Ожидаем освобождение аппаратуры видеокарты
  repeat
  until DWord(surfrec[1].surf.GetFlipStatus(DDGBS_ISBLTDONE)) <>
    DDERR_WASSTILLDRAWING;
  // Переключаем страницу во время обратного хода луча по экрану
  repeat
  until RespHandler(surfrec[1].surf.flip(NIL,DDFLIP_WAIT));
end; {Flip}

// Рисование всей поверхности
procedure DrawSurf;
VAR
 i: integer;
```

```
begin
  { Очищаем 2-ю поверхность, копируя на нее нижнюю скан-линию
    из 3-й поверхности }
  for i:=0 to SCRY-1 do CopyRec(0,SCRY-1,SCRX-1,SCRY-1,0,i,3,2);
  // Перемещаем все спрайты на шаг
  for i:=0 to SPRNUM-1 do
   with sprites[i] do
    begin
    if (dx > 0) and (x+width > SCRX-3) or
     (dx < 0) and (x < 2) then dx:=-dx;
    if (dy > 0) and (y+height > SCRY-3) or
     (dy < 0) and (y < 2) then dy:=-dy;
    Inc(x,dx);
    Inc(y,dy);
    CopySpr(0,0,width-1,height-1,x,y,3,2);
    end;
  // Переключаем страницы
  Flip;
end; {DrawSurf}

{ Загрузка данного BMP-файла на поверхность DDraw }
procedure LoadBitMap(x1,y1,x2,y2,nsurf: integer; BitMapName: string);
VAR
  Bitmap: HBitmap;
  BM: Windows.TBitmap;
  DC,ImageDC: HDC;

begin
  Bitmap:=LoadImage(0,PChar(BitMapName),IMAGE_BITMAP,0,0,
   LR_LOADFROMFILE);
  if BitMap = 0 then
    begin
    DestroyAllSurf;
    Halt;
    end;
```

```
  { Получаем размер картинки }
  GetObject(Bitmap,SizeOf(BM),@BM);
  ImageDC:=CreateCompatibleDC(0);
  SelectObject(ImageDC,BitMap);
  surfrec[nsurf].surf.GetDC(DC);
  BitBlt(DC,x2,y2,BM.bmwidth,BM.bmheight,ImageDC,x1,y1,SRCCOPY);
  surfrec[nsurf].surf.ReleaseDC(DC);
  DeleteDC(ImageDC);
  DeleteObject(Bitmap);
end; {LoadBitMap}

BEGIN
  // При повторном запуске программы активизируем запущенный экземпляр
  hWindow:=FindWindow(AppName, AppName);
  if hWindow > 0 then
    begin
    SendMessage(hWindow, WM_SETFOCUS, hWindow, 0);
    ShowWindow(hWindow, SW_SHOWNORMAL);
    Exit;
    end;
  if not WinRegister then
   begin
   MessageBox(0, 'WinRegister failed', nil, mb_Ok);
   Exit;
   end;
  hWindow:=WinCreate;
  if hWindow = 0 then
   begin
   MessageBox(0, 'WinCreate failed', nil, mb_Ok);
   Exit;
   end;
  { Создаем DirectDraw COM object }
  Result:=DirectDrawCreate(NIL,DirectDraw,NIL);
  if Result <> DD_OK then
   begin
   MessageBox(hWindow, 'Can not create DirectDraw object',
```

```
  nil, mb_Ok);
 Exit;
 end;
{ Устанавливаем exclusive cooperative level }
Result:=DirectDraw.SetCooperativeLevel(hWindow,
  DDSCL_EXCLUSIVE or DDSCL_FULLSCREEN);
if Result <> DD_OK then
 begin
 MessageBox(hWindow, 'SetCooperativeLevel failed', nil, mb_Ok);
 Exit;
 end;
{ Если нет аппаратного ускорения, то выход }
FillChar(HALCaps,SizeOf(DDCaps),0);
FillChar(HELCaps,SizeOf(DDCaps),0);
HALCaps.dwSize:=SizeOf(DDCaps);
HELCaps.dwSize:=SizeOf(DDCaps);
DirectDraw.GetCaps(HALCaps, HELCaps);
if HALCaps.dwCaps and DDCAPS_BLT <> DDCAPS_BLT then
  begin
  MessageBox(hWindow, 'BltFast does not work', nil, mb_Ok);
  Exit;
  end;
{ Устанавливаем графический режим True Color с 32 или 24 битами
  на точку }
for i:=4 downto 3 do
  begin
  Result:=DirectDraw.SetDisplayMode(SCRX,SCRY,i*8);
  if Result = DD_OK then
   begin
   bpp:=i;
   break;
   end;
  if (Result <> DD_OK) and (i = 3) then
   begin
   MessageBox(hWindow, 'Can not set display mode', nil, mb_Ok);
   Exit;
```

```
   end;
  end;
// Устанавливаем признак, что поверхности не существуют
for i:=1 to MAXSURF do surfrec[i].exist:=FALSE;
{ Создаем комплексную экранную поверхность и поверхность для
  спрайтов }
if CreateSurf(hWindow,1,SCRX,SCRY,PRIMARYSURF,TRUE) <> DD_OK then
 Halt;
if CreateSurf(hWindow,3,SCRX,SCRY,OFFSCREENSURF,FALSE) <> DD_OK then
 Halt;
// Системный курсор нам не нужен, прячем его
ShowCursor(FALSE);
// Узнаем путь к файлам программы
path:=ParamStr(0);
i:=Length(path);
repeat
 Dec(i);
until path[i] = '\';
SetLength(path,i);
// Загружаем картинку в 3-ю поверхность
LoadBitMap(0,0,0,0,3,path+'diamond.bmp');
{ Создаем 100 спрайтов с положительными и отрицательными скоростями
  1 и 2 точки на кадр }
Randomize;
for i:=0 to SPRNUM-1 do
 with sprites[i] do
  begin
  x:=Random(SCRX-SPRWIDTH);
  y:=Random(SCRY-SPRHEIGHT);
  dx:=Random(4)-1;
  if dx < 1 then Dec(dx);
  dy:=Random(4)-1;
  if dy < 1 then Dec(dy);
  width:=SPRWIDTH;
  height:=SPRHEIGHT;
  end;
```

```
  lasttick:=GetTickCount;
  // Цикл работы программы и обработки сообщений
  repeat
   tick:=GetTickCount;
   { Если не прошло 5 мс со времени последней активности программы,
     то передаем кванты времени Windows }
   if tick < lasttick+5 then
     begin
     Sleep(2);
     continue;
     end;
   lasttick:=tick;
   if bIsActive then
    begin
    if surfrec[1].surf.IsLost <> DD_OK then RestoreSurfaces;
    DrawSurf;
    end;
   // Получение сообщения без ожидания его появления в очереди
   if PeekMessage(Message,0,0,0,PM_REMOVE) then
     begin
     if Message.message = wm_Quit then Halt(Message.wParam);
     TranslateMessage(Message);
     DispatchMessage(Message);
     end;
  until FALSE;
END.
```

Итак, начнем с секции констант. У нас появились константы для обозначения типов поверхностей (`PRIMARYSURF`, `FLIPSURF`, `OFFSCREENSURF`, `MEMORYSURF`), которые мы используем для обращения к методам DirectDraw при их создании и блокировке. Максимальное число используемых поверхностей `MAXSURF=3` применяется для задания размеров структуры `SurfRec` параметров используемых поверхностей. Эти параметры означают следующее:

- `surf` — ссылка на объект `IDirectDrawSurface`;
- `psurf` — адрес начала поверхности после ее блокировки;
- `surftype` — тип этой поверхности;

- `pixx`, `pixy` — горизонтальный и вертикальный размеры этой поверхности;
- `pitch` — шаг в байтах между соседними скан-линиями;
- `exist` — признак того, что поверхность была создана.

В этом примере мы используем три поверхности: экранную типа `PRIMARYSURF`, присоединенную к ней поверхность переключения `FLIPSURF` и внеэкранную поверхность типа `OFFSCREENSURF` для хранения образца спрайта, который мы загрузим на нее из BMP-файла. В ваших проектах вы установите переменную `MAXSURF` в соответствии со своими планами использования видеопамяти. Размеры спрайта `SPRWIDTH` и `SPRHEIGHT` мы берем, просматривая этот файл в программе вывода графики (например, ACDSee, а можно также воспользоваться программой Paint (файл mspaint.exe из каталога Windows\System32)). Число спрайтов `SPRNUM` определяет размер структуры `sprites` описания всех этих спрайтов на экране. Смысл полей этой структуры такой:

- экранные координаты `x`, `y` левого верхнего угла ограничивающего прямоугольника;
- приращение координат `dx`, `dy` в точках для следующего кадра. У нас это будут случайно выбранные числа из множества {`-2`, `-1`, `1`, `2`}. Когда спрайт подойдет к краю экрана, соответствующая покадровая скорость поменяет свой знак. Учтите, что при попытке вывести спрайт, который не помещается полностью на поверхности, он нарисован не будет;
- `width` и `height` — это ширина и высота спрайта, то же, что и `SPRWIDTH` и `SPRHEIGHT`. У нас здесь все спрайты одинаковые.

Опережающие описания

```
procedure DestroyAllSurf; forward;
procedure DrawSurf; forward;
procedure LoadBitMap(x1,y1,x2,y2,nsurf: integer; BitMapName: string);
  forward;
```

нужны нам для того, чтобы сообщить компилятору прототипы вызовов этих процедур для генерации кода по их вызову. Не всегда описания функций можно расположить так, чтобы все они встречались раньше кода, их вызывающего. Вот здесь и приходит на помощь опережающее описание.

Теперь поверхности получают номера от единицы до трех. Это их индексы в структуре `SurfRec`. Создание поверхности данного номера, размера и типа у нас делает функция `CreateSurf`. Она же устанавливает ключевой цвет в ноль. Уничтожает все поверхности и освобождает объект DirectDraw COM процедура `DestroyAllSurf`. Функция `RestoreSurfaces` восстанавливает потерянную память всех поверхностей, а также перезагружает в третью поверхность картинку с алмазом, который и является выводимым спрайтом.

Таким путем наш пример продолжает, как ни в чем не бывало, выполняться после переключения экрана на другие задачи. Функция `RespHandler` используется для сокращения кода при вызове методов поверхности, при получении кода возврата `DDERR_WASSTILLDRAWING` мы должны повторять вызов этого метода, а при получении ошибки `DDERR_SURFACELOST` — вызвать функцию `RestoreSurfaces`. При блокировке поверхности функцией `SurfLock` в соответствующий элемент структуры `SurfRec` записывается адрес этой поверхности и шаг в байтах между соседними скан-линиями.

Функция `Rect` принимает координаты `ALeft`, `ATop`, `ARight` и `ABottom` (слева, сверху, справа, внизу) прямоугольника на поверхности и возвращает структуру Windows типа `TRect`, которую надо задавать методу рисования поверхности `BltFast`. Процедура

```
procedure CopyRec(xs1,ys1,xs2,ys2,xd,yd,nsurf1,nsurf2: integer)
```

принимает параметры:

- `xs1`, `ys1` — координаты левого верхнего угла исходного прямоугольника;
- `xs2`, `ys2` — координаты правого нижнего угла исходного прямоугольника;
- `xd`, `yd` — координаты левого верхнего угла прямоугольника назначения;
- `nsurf1`, `nsurf2` — номера поверхности-источника и поверхности назначения

и выполняет копирование обозначенного прямоугольника с одной поверхности на другую (или на ту же самую поверхность). При этом копируются все точки, включая точки с ключевым цветом. Процедура `CopySpr`

```
procedure CopySpr(xs1,ys1,xs2,ys2,xd,yd,nsurf1,nsurf2: integer)
```

аналогична предыдущей, но она копирует только точки, не имеющие ключевой цвет, т. е. выводит спрайт. Отличие состоит в передаче методу `BltFast` бита `DDBLTFAST_SRCCOLORKEY`, который заставляет пропускать при копировании точки, имеющие ключевой цвет, для поверхности-источника.

Процедура `Flip` (без параметров) переключает экранную поверхность на поверхность переключения. Заметьте, что при этом номера этих поверхностей в структуре `SurfRec` не меняются: как имела экранная поверхность индекс один, так она его и будет иметь. Для этого вызывается метод `Flip` первичной поверхности. Перед переключением страницы вызывается ее метод

```
GetFlipStatus(DDGBS_ISBLTDONE)
```

для проверки, освободилась ли аппаратура видеокарты. Если этот метод вернет код `DDERR_WASSTILLDRAWING`, то это значит, что аппаратура занята рисованием. Если в этот момент вызвать метод `Flip`, то это ожидание освобождения аппаратуры будет происходить внутри драйвера видеокарты, а он работает в нулевом кольце защиты, и прервать это ожидание никто не в си-

лах. В результате программа, интенсивно использующая аппаратуру видеокарты на компьютере с быстрым центральным процессором, чуть ли не 100% времени будет проводить в ожидании освобождения аппаратуры видеокарты и будет плохо отвечать на события. В этом же цикле

```
repeat
until DWord(surfrec[1].surf.GetFlipStatus(DDGBS_ISBLTDONE)) <>
  DDERR_WASSTILLDRAWING;
```

можно поместить код для проверки возникновения событий. В самом вызове `Flip` мы передаем бит `DDFLIP_WAIT`, который говорит о том, что переключение поверхности надо производить во время обратного хода луча по экрану. В этом примере мы явно не используем метод DirectDraw для ожидания этого обратного хода, потому что рисуем на внеэкранных поверхностях. А выглядит этот метод так:

```
DirectDraw.WaitForVerticalBlank(DDWAITVB_BLOCKBEGIN,0)
```

Этот вызов возвращает управление, когда начинается обратный ход луча по экрану. Вызов

```
DirectDraw.WaitForVerticalBlank(DDWAITVB_BLOCKEND,0)
```

делает возврат, когда заканчивается обратный ход луча и начинается прямой ход луча во время рисования изображения. Иногда используется тот или другой вызов. Смотрите результаты на экране и выбирайте тот, при котором изображение будет более стабильным. Особенно обратите внимание на вывод изображений в верхней части экрана, откуда лучи начинают рисовать кадр.

Процедура `DrawSurf` (без параметров) копирует все 100 спрайтов из образца на третьей поверхности во вторую поверхность. Но перед этим ведь надо очистить эту вторую поверхность. Для этого выполняется цикл

```
{ Очищаем 2-ю поверхность, копируя на нее нижнюю скан-линию
  из 3-й поверхности }
for i:=0 to SCRY-1 do CopyRec(0,SCRY-1,SCRX-1,SCRY-1,0,i,3,2);
```

который размножает нижнюю скан-линию из третьей поверхности на всю вторую поверхность. Это, конечно, не самый экономный вариант — шестьсот раз вызывать копирование одной скан-линии, но этим я хотел показать, с какой скоростью происходит аппаратное копирование видеопамяти. Затем каждый спрайт перемещается на свой покадровый шаг `dx`, `dy` с проверкой его выхода за пределы поверхности, и для каждого спрайта выполняется его копирование с третьей поверхности на вторую. Затем происходит переключение страниц.

Процедура `LoadBitMap`

```
procedure LoadBitMap(x1,y1,x2,y2,nsurf: integer; BitMapName: string)
```

выводит изображение из BMP-файла с полным путем и именем BitMapName в координаты x1, y1 поверхности. Используемая для этого функция GDI BitBlt вообще-то может выводить любой прямоугольник из данной картинки, но мы эту возможность не используем и выводим всю картинку целиком. Параметры x2 и y2 можно было бы не задавать. Эта процедура подчиняется правилам работы с дисплейным контекстом устройства (Display Context, DC). Это структура Windows, которая абстрагирует различные графические устройства от прикладных программ. Программы задают такие параметры, как кисть, карандаш, образец заполнения для данного дисплейного контекста, а драйвер выбранного устройства по этой структуре выполняет заданную операцию. Как видим, на поверхности DirectDraw можно рисовать обычными функциями из набора GDI, надо только получить у поверхности ее контекст. Впрочем, круг на поверхности мы уже рисовали.

В основной части программы мы, как и прежде, создаем объект DirectDraw COM, создаем три поверхности тремя вызовами CreateSurf, прячем указатель и получаем путь к файлу BMP с бриллиантом:

```
// Узнаем путь к файлам программы
path:=ParamStr(0);
i:=Length(path);
repeat
  Dec(i);
until path[i] = '\';
```

Надеюсь, читатель не станет использовать вызов Application.ExeName?

Для извлечения пути из полного имени файла существует функция из модуля SysUtils, но я получил этот путь вручную. Зачем присоединять этот модуль для использования одной небольшой функции? Одно его подключение в программу (без его использования!) увеличит размер exe-файла на целых 30 Кбайт.

Далее идет загрузка картинки и создание спрайтов. Здесь нет чего-то особенного. А вот цикл обработки сообщений изменился. В чем дело, чем старый был плох? Во-первых, для чтения сообщения вместо GetMessage мы теперь используем более мощную функцию PeekMessage, которая не задерживает выполнение программы, если в очереди ее окна нет сообщений. Иначе наши дорогие бриллианты шевелились бы лишь тогда, когда мы двигали бы мышь или нажимали на всякие там кнопки (проверьте это!). Из-за этого изменилась система завершения программы: теперь мы проверяем в сообщении одноименное поле message и ловим сообщение wm_Quit. Также PeekMessage может читать сообщения без их извлечения из очереди, что иногда бывает нужно.

Перед циклом работы программы мы засекаем текущее время:

```
lasttick:=GetTickCount;
```

Для этой переменной мы используем восьмибитовые числа типа `TLargeInteger`: а вдруг у кого-то Windows продержалась дольше 49,7 суток? Тогда число миллисекунд, прошедших с момента старта системы, уже не поместится в двойное слово. В цикле

```
// Цикл работы программы и обработки сообщений
repeat
  tick:=GetTickCount;
  { Если не прошло 5 мс со времени последней активности программы,
    то передаем кванты времени Windows }
  if tick < lasttick+5 then
    begin
      Sleep(2);
      continue;
    end;
    lasttick:=tick;
    if bIsActive then
      begin
        if surfrec[1].surf.IsLost <> DD_OK then RestoreSurfaces;
        DrawSurf;
      end;
    // Получение сообщения без ожидания его появления в очереди
    if PeekMessage(Message,0,0,0,PM_REMOVE) then
      begin
        if Message.message = wm_Quit then Halt(Message.wParam);
        TranslateMessage(Message);
        DispatchMessage(Message);
      end;
until FALSE;
```

мы даем нашей программе поработать не чаще одного раза в пять миллисекунд. Это ограничение не обязательно, но иначе диспетчер задач Windows, который вызывается по нажатию комбинации клавиш <Ctrl>+ +<Alt>+<Del> (только не в Windows 9*x*!), показывал бы стопроцентную загрузку процессора даже тогда, когда окно нашей программы находится в свернутом состоянии.

И далее в цикле у нас определяется, активна ли программа, и если да, то проверяется, не потеряны ли поверхности. При необходимости они восстанавливаются.

(Вообще-то вызов

```
if surfrec[1].surf.IsLost <> DD_OK then RestoreSurfaces;
```

в этом примере лишний. Рисование происходит и так достаточно часто, поэтому отлов потери поверхностей произойдет быстро.) Затем происходит вызов `DrawSurf` для рисования всех спрайтов и переключения страниц. Это происходит, по возможности, раз в пять-шесть миллисекунд. Без ожидания времени скорость движения спрайтов определялась бы числом показа кадров в секунду, т. е. зависела бы от используемого монитора, а это неправильная практика. Для ознакомления с таймером высокого разрешения (1,19 МГц) смотрите в справке Windows описания функций `QueryPerformanceCounter` и `QueryPerformanceFrequency`.

Вы можете также использовать для измерения времени счетчик тактов центрального процессора командой встроенного ассемблера `rdtsc` *(см. главу 1)*. В листинге 3.6 приведен пример того, как я посчитал, сколько этот счетчик натикает за восемь минут.

Листинг 3.6. Использование высокоточного счета времени

```
{ Узнаем, сколько насчитает счетчик rdtsc за 8 минут }
procedure  Rdtsc8;
VAR
  freq,count1,count2,rdcount1,rdcount2 : TLargeInteger;
  OldPriorityClass,OldThreadPriority : dword;
  hProcess,hThread : THandle;
begin
        hProcess:=GetCurrentProcess ();
        hThread:=GetCurrentThread ();
        OldPriorityClass:=GetPriorityClass (hProcess);
        OldThreadPriority:=GetThreadPriority (hThread);
        SetPriorityClass (hProcess,REALTIME_PRIORITY_CLASS);
        SetThreadPriority (hThread,THREAD_PRIORITY_TIME_CRITICAL);
        QueryPerformanceFrequency (freq);
        repeat
          QueryPerformanceCounter (count1);
          asm
```

```
          db    0Fh, 31h
          mov   dword ptr rdcount1,eax
          mov   dword ptr rdcount1+4,edx
          end;
          { Ждем, пока счетчик не натикает 0.1 с }
          repeat
            QueryPerformanceCounter (count2);
          until count2 >= count1+freq div 10;
        until count2-count1-freq div 10 < 100;
        asm
        db      0Fh, 31h
        mov     dword ptr rdcount2,eax
        mov     dword ptr rdcount2+4,edx
        end;
        SetPriorityClass (hProcess,OldPriorityClass);
        SetThreadPriority (hThread,OldThreadPriority);
        rd8min:=(rdcount2-rdcount1)*10*60*8;
end; {Rdtsc8}
```

Методика здесь такая: нашему процессу и потоку в нем назначаются наивысшие приоритеты, как у драйверов и программ, критичных ко времени отклика. Это следует делать только на короткое время, иначе Windows не сможет нормально работать. Затем через обращение к `QueryPerformance-Counter` засекается, сколько тактов делает процессор за заданный интервал времени. Отсюда узнаем число тактов в секунду. В конце восстанавливаем приоритеты. `db 0Fh,31h` — это и есть код команды `rdtsc`. Delphi 7 этот мнемокод уже понимает.

3.7. Написание игровой программы

Наконец, осталось обобщить полученные сведения и навыки и написать что-нибудь полезное, например, какую-нибудь игру.

В одной из книг всемирно известного популяризатора науки Мартина Гарднера (Martin Gardner) упоминалась интересная головоломка, которую придумал польский кинокритик Лех Пияновский (Leh Pijanowski). Она мне понравилась. В ней комплект домино из 28 костей разложен в виде прямоугольника 7 × 8, границы между костями не показаны. Требуется

восстановить эти границы. При решении этой задачи используются такие соображения.

1. Относительно половинок кости, содержащих очки 2, 3 и 6, можно сразу сказать, расположены ли их кости горизонтально или вертикально, потому что они несимметричны относительно поворота на 90°. Если кость лежит вертикально, то линия точек для 2 и 3 идет из правого верха квадратика к левому низу. Такими их делают на производстве. Поэтому такие кости отметить легче, особенно если они прилегают к краям прямоугольника. Ну и, конечно, неплохо запоминать, какие кости уже отмечены, хотя бы для дублей и шестерок.
2. Соображение четности: иногда при неправильной отметке костей в какой-то ограниченной области остается один изолированный неотмеченный квадратик.
3. Если к примеру, квадратик 3 граничит с квадратиком 2 лишь в одном месте допустимым образом, то кость 2—3 находится именно здесь.
4. Если например, квадратик 2 примыкает к половинкам 4 и 5, а кость 2—4 уже обозначена, то здесь лежит кость 2—5.

Некоторые книжные авторы создают позиции, имеющие единственное решение, и предлагают решать позиции по науке: сначала доказать, что данная кость лежит именно здесь, а потом уже ее отмечать. Я не ставил перед собой такой задачи.

Сначала я решил, что в таком виде игра будет слишком проста, и придумал вариант с цифрами, которые написаны на квадратах вместо точек. Здесь очки 2, 3, и 6 не обладают особенностью, которая облегчает решение. Но если разместить 56 таких половинок в виде прямоугольника 7 × 8, то трудно будет отметить какую-либо кость ввиду нескольких вариантов. Некоторые из них могут привести в тупик, и тогда надо будет возвращаться назад. Этого в реальной игре не хотелось бы делать, и я придумал другую форму игровой доски — в виде квадрата 8 × 8, у которого с каждой стороны вынуто по два квадратика так, что получается фигура с четырьмя осями симметрии. Такая фигура облегчает решение, т. к. клетки посередине сторон играют роль угловых и могут соединяться с соседями лишь двумя способами. В этом легко убедиться, попробовав поиграть. Такая форма позиции при практической проверке оказалась хорошо сбалансированной: практически при любом случайном размещении квадратиков можно было решить позицию без возвратов назад. Поэтому я написал игровую программу для такой позиции. Позже для полноты я добавил игру на прямоугольнике 7 × 8, о котором говорил вначале. Конечно, в компьютерном варианте это игра на время, и я даже получил в обоих вариантах сумасшедшие по скорости результаты. Правда, если честно признаться, я иной раз делал ходы наобум — получится, так получится, а нет — начну новую игру. И еще я выбирал позицию полегче для установления рекордов, щелкая мышью на кнопке **NEW GAME**, если я

не видел, как можно быстро начать отмечать камни. Попробуйте превзойти эти мои рекорды. Еще позднее я сделал подсказку: если водить мышью над квадратиками, то возможная кость, над центром которой находится указатель, выделяется рамкой. Если указатель уходит, то рамка удаляется. И еще я ускорил выделение кости: раньше для этого надо было нажать кнопку над одной половинкой, переместить указатель на другую половинку и отпустить кнопку. Теперь же надо сделать щелчок примерно на середине кости, которую игрок желает отметить. Аналогично снимается отметка с кости, и таким же легким путем кости отмечаются по-другому. Играть стало еще удобнее и быстрее, поэтому мои старые рекорды нуждаются в обновлении.

Программа использует два музыкальных MIDI-файла. Эти мелодии сочинил питерский композитор Константин Елгазин (**http://www.e-studios.info**). В то время он еще учился в школе. Звуковые WAV-файлы подготовил я сам, подобрав и отредактировав то, что было в наличии. Для игры такого уровня достаточно иметь звуки с 8-битной оцифровкой и частотой дискретизации 22 050 Гц. Графика готовилась в трехмерном редакторе, кота и фоновое изображение bkgr.bmp предоставил 3D-аниматор и художник Николай Крисанов.

Вначале я некоторое время думал, как показать ход времени. И решил, что будет очень остроумно, если этот нарисованный в книжке кот будет есть бесконечную цепочку сарделек, по одной сардельке на каждый такт счетчика. А при нажатии кнопки **NEW GAME** слева из книжки будет появляться рука, которая вытаскивает изо рта все эти сардельки назад, а кот при этом недовольно кричит, и затем начинается все с начала. Сама идея нарисованного оживающего кота пришла из сказки Джанни Родари "Джельсомино в стране лжецов". В этой сказке был такой чудесный художник, что нарисованные им персонажи тут же оживали. Но редактор с "двух коров" (**www.tucows.com**) не понял юмора и выразил свое недовольство "летающими сардельками". Пришлось их убрать, оставив кота голодным. Позже я также убрал мошек с бутерброда и чайной ложки, т. к. их приняли за мух и попросили убрать мух с еды.

Рассмотрим графические файлы, все файлы для игры расположены в каталоге chapter3\Dominoes на компакт-диске.

Файл dice.bmp содержит неотмеченные и отмеченные квадратики для обеих ориентаций, причем для второго уровня игры достаточно изображения одной ориентации ввиду симметрии. Справа вверху от костей видны два разделителя половинок кости — для горизонтального и вертикального положения. Еще правее видна каретка в виде вертикальной желтой линии, она понадобится для ввода имени игрока, попавшего в таблицу рекордов. Еще правее нанесен ряд точек разных цветов. Это цвета, которые используются для вывода на поверхность напрямую, обычно для надписей шрифтами. Ниже имеются две фазы мяуканья кота и окантовки для отмеченной кости (они же — маркеры для подсказки). Ниже костей находятся 48 фаз анимы-

рованного указателя. Он тоже сделан в 3D-редакторе, а в качестве текстуры использовано красивое изображение, которое дает старая маленькая рекурсивная DOS-программка для Pascal 4 под названием plasma.pas. Сразу за последней фазой имеется пустой прямоугольник для хранения фона экрана под указателем. Когда мы прячем указатель, на его месте выводится этот прямоугольник. Далее идут элементы и фон для панели правил игры, цифры для показа времени, панель для ввода имени игрока, часть заголовка игры и некоторые другие мелочи.

Файл record.bmp содержит фон для таблицы рекордов. Текст вписывается в нее по точкам заранее приготовленным шрифтом.

Файл balls.bmp хранит фазы шарика, рисующего слово Domino в заставке игры. Этот шарик сделан аналогично указателю. Сначала надпись рисовалась постепенно, но потом я стал выводить ее на внеэкранной поверхности и копировать на поверхность экрана мгновенно. В файле ballpos закодированы координаты положения этого шарика, когда он рисует название игры.

Эти файлы создает программа:

- records — зашифрованная таблица рекордов;
- dominoes.cfg — конфигурационный файл (разрешен ли звук, музыка, уровень последней игры).

Запустив программу на выполнение, вы увидите картинку, как на рис. 3.1.

Рис. 3.1. Экран игры на втором уровне

Вверху находится строка **RESULT** (Результат). Когда все кости будут отмечены, то по этой строке и характерному звуку вы узнаете, правильно ли вы это сделали. Иногда можно получить одну кость отмеченной дважды, а какая-то кость будет пропущена. Программа не следит, правильно ли вы отмечаете кости, но на первом уровне игры она не допускает такое обозначение костей, которого не встречается в природе (с очками 2, 3 и 6). Других ограничений нет, и можно отметить кость так, как не может быть по соображениям четности.

Ниже расположена строка **LEVEL** (Уровень). На самом деле это не два уровня, а две родственные игры. Можно выбрать ту, которая вам нужна, щелкнув мышью на цифре **1** или **2**.

Еще ниже вы видите знак вопроса, щелкнув по которому, можно прочитать правила игры, и надпись **EXIT** для выхода из игры. Перед выходом отображается таблица рекордов.

О кнопке **NEW GAME** я уже говорил. А слева от кота нарисовано слово **Pause**. Если щелкнуть по нему, то окно программы свернется на панель задач. Таким образом предотвращается обдумывание ходов с выключенными часами. Обратите также внимание, что если переключиться на другую программу, а потом вернуться назад в игру, то время продолжит свой ход так, как будто не было задержки в игре.

Надписи **Sound** и **Midi** позволяют включать и выключать звук и музыку.

Взгляните теперь на саму позицию. Вначале была отмечена кость 4—5, затем автоматически 1—6, 0—2 и 0—3. И мы попали в тупичок, получив изолированный квадратик с цифрой 3. Это я и имел в виду, когда говорил, что боковые клетки играют роль угловых и имеют лишь двух соседей.

Настала пора ознакомиться с исходными файлами этого проекта. Читатель (или читатели, если эту книгу купили несколько человек) должен понимать, что я не могу подробно, строка за строкой, разбирать и комментировать исходный код программы. Это скучно, и файлы содержат достаточно подробные комментарии. Возможно, когда-нибудь этот доскональный разбор войдет в мое полное собрание сочинений и электронных писем. А в этой книге я ограничусь общими описаниями и замечаниями по ходу дела.

3.7.1. Файл glob.pas

Этот файл хранит глобальные константы и переменные, которые могут использоваться всеми модулями игры. В листинге 3.7 представлен его текст.

Листинг 3.7. Модуль glob.pas

```
{$A-,B-,I-,R-,S-,X+,H+}
UNIT  Glob;
```

```
INTERFACE
USES  Windows;
TYPE
  tmb = array[0..10000000] of byte;
  tpmb = ^tmb;
  tmw = array[0..10000000] of word;
  tpmw = ^tmw;
  { Типы состояния игры (программы): заставка, идет игра,
    экран после конца игры, показ правил игры, ввод имени
    игрока, показ таблицы рекордов }
  tGameState = (ISSPLASH,GAMES,GAMEOVER,HELP,NAME,TREC);
  { Типы событий при работе программы: нажата клавиша,
    нажата левая кнопка мыши, правая кнопка, отпущена левая кнопка,
    события от таймера. Реально обрабатываются лишь первые два
    и последнее события }
  tGEvent = (keyd,lbdown,rbdown,lbup,tic);
CONST
  { Номер последней используемой поверхности DirectDraw, считая с 1 }
  MAXSURF = 3;
  // Название нашей программы
  AppName = 'Domino Dilemma';
  // Разрешение экрана
  SCRX = 800;
  SCRY = 600;
  // Коды управляющих клавиш
  ESC = chr(vk_escape);
  BACK = chr(vk_back);
  RETURN = chr(vk_return);
VAR
  hWindow : hWnd;
  biSActive : boolean;
  AMessage: TMsg;
  trig : word = 0;
  // Уровень игры
  level : byte = 1;
  biosct,col0 : word;
```

```
  // Число байтов на пиксел
  bytepp : dword;
  path : string;
  // Состояние игры, вначале - заставка
  GameState : tGameState = ISSPLASH;
  // Запрос на игру мелодии
  midirequest : boolean = FALSE;
  // Управление звуком и MIDI
  soundenable,soundprohib,midienable,midiprohib : boolean;
  // Счетчики времени
  tc,tick,lasttick,qpfreq,qpfreq2 : TLargeInteger;

IMPLEMENTATION

end.
```

Самая главная переменная игры — это `GameState`, текущее состояние игры. Для того чтобы знать, что надо делать при обработке того или иного события, программа должна представлять, в каком состоянии она находится. Эта переменная имеет перечисляемый тип:

```
tGameState=(ISSPLASH,GAMES,GAMEOVER,HELP,NAME,TREC)
```

В начале работы игра показывает заставку, поэтому стартовое значение у `GameState` — это `ISSPLASH`. Во время заставки программа обрабатывает лишь нажатие клавиши <Esc>, осуществляя свое завершение. Заставка сама по себе переходит в игру соответствующего уровня `level`, который запоминается в файле.

Состояние `GAMES` означает, что идет игра. То есть комплект домино разложен и игра еще не закончена. В этом состоянии программа при нажатии клавиши <Esc> переходит в состояние `TREC` и показывает таблицу рекордов. При нажатии на левую кнопку мыши программа по координатам указателя смотрит, находится ли он над меню (при этом вызывается процедура для обработки меню `CheckMenu`). Иначе игра проверяет, отмечена этим нажатием какая-либо кость или с нее снята отметка.

Состояние `GAMEOVER` означает, что весь комплект костей отмечен верно и игра закончена.

Состояние `HELP` — это показ правил игры.

Состояние `NAME` — это ввод имени игрока для таблицы рекордов на соответствующей панели с кареткой.

TREC — показ таблицы рекордов, после которого происходит завершение программы.

В процедуру обработки сообщений передается тип возникшего события. Эти типы перечислены в шаблоне

```
tGEvent = (keyd,lbdown,rbdown,lbup,tic)
```

Это соответственно, клавиша нажата, левая кнопка мыши нажата, правая кнопка мыши нажата, левая кнопка мыши отпущена, сообщение от таймера.

3.7.2. Файл dominoes.dpr

Это текст основного файла программы. Он представлен в листинге 3.8.

Листинг 3.8. Файл dominoes.dpr

```
{$A-,B-,I-,R-,S-,X+,H+}
program  Dominoes;
uses
  Windows,
  Messages,
  DDraw,
  Dsound,
  MMSystem,
  Glob in 'Glob.pas',
  Ddrawunit in 'Ddrawunit.pas',
  Mouse in 'Mouse.pas',
  Game in 'Game.pas',
  Fonts in 'Fonts.pas',
  Util in 'Util.pas',
  Sound in 'Sound.pas',
  Midi in 'Midi.pas';

{$R *.res}

LABEL ERR;
VAR
  Message: TMsg;
  i : integer;
  trig : word = 0;
```

```
function WindowProc (Window: HWnd; AMessage, WParam,
 LParam: integer): integer; stdcall; export;
LABEL  10;

begin
  WindowProc:=0;
  case AMessage of
  // Сообщение об активизации/дезактивизации нашего окна
wm_activateapp:
    begin
    { Запоминаем в bIsActive активно ли наше приложение.
      Оно может быть свернуто в кнопку на панели задач }
    bIsActive:=boolean(wParam);
    // Если окно не активно, то нельзя рисовать курсор
    if not bIsActive then movis:=FALSE;
    end;
wm_EraseBkgnd, wm_SetCursor:
    begin
    WindowProc:=1;
    Exit;
    end;
    // Сообщение от таймера. (Они возникают 18.2 раза в секунду)
wm_timer:
    begin
    Inc (biosct);
    if bIsActive then Process (tic,#0,0,0);
    end;
    // Нажата левая кнопка мыши
wm_lbuttondown:
    begin
    moleft:=TRUE;
    if bIsActive then Process (lbdown,#0,LoWord(lParam),HiWord(lParam));
    end;
(*
    // Нажата правая кнопка мыши
wm_rbuttondown:
```

```
    begin
    moright:=TRUE;
    if bIsActive then Process (rbdown,#0,LoWord(lParam),HiWord(lParam));
    end;
    // Отпущена левая кнопка мыши
wm_lbuttonup:
    begin
    moleft:=FALSE;
    if bIsActive then Process (lbup,#0,LoWord(lParam),HiWord(lParam));
    end;
    // Отпущена правая кнопка мыши
wm_rbuttonup:
    moright:=FALSE;
*)
    // Указатель мыши переместился
wm_MouseMove:
    if not moprohib and mostart and bIsActive then
      begin
      mophase:=(mophase+trig) mod mophases;
      trig:=trig xor 1;
      // Запоминаем координаты "горячей точки" указателя
      mox:=LoWord(lParam);
      moy:=HiWord(lparam);
      if movis then
        // Перерисовываем его с другой картинкой
        begin
        DirectDraw.WaitForVerticalBlank (DDWAITVB_BLOCKEND,0);
        MouseOff;
        if GameState = GAMES then
          // Рисуем и убираем подсказку для отметки костей
          if level = 1 then
            begin
            if MouseInBox (PX1,PY1,SQX*8,SQY*7) then Check1 else
              if hintx > -1 then RemoveHint1;
            end else
```

```
              if MouseInBox (PX2,PY2,SQX*8,SQY*8) then Check2 else
                if hintx > -1 then RemoveHint2;
        MouseOn;
        end;
      end;
    // Пришел код нажатой алфавитно-цифровой клавиши
wm_Char:
    if bIsActive then Process (keyd,char(wParam),0,0);
    { Сообщение от проигрывателя MIDI, что он закончил играть
      мелодию. Играем очередную MIDI-мелодию }
mm_MCINOTIFY:
    PlayMidi;
    { Пришло сообщение об уничтожении нашего окна.
      Освобождаем занятые программой ресурсы }
wm_Destroy:
    begin
    // MIDI
    StopMidi;
    CloseMidi;
    // DirectSound
    DestroySound;
    // Восстанавливаем предыдущий графический режим
    Directdraw.RestoreDisplayMode;
    // Уничтожаем поверхности DirectDraw
    DestroyAllSurf;
    // Уничтожаем таймер
    KillTimer (hWindow,1);
    // Посылаем нашей программе сообщение wm_Quit
    PostQuitMessage(0);
    Exit;
    end;
  end; {case}
  { Переправляем стандартной функции обработки сообщений
    необработанные сообщения }
  WindowProc:=DefWindowProc(Window,AMessage,WParam,LParam);
end; {WindowProc}
```

```
{ Регистрируем наш класс окна }
function  WinRegister: Boolean;
VAR
  WindowClass : TWndClass;

begin
  WindowClass.Style:=0;
  // Регистрируем нашу оконную функцию
  WindowClass.lpfnWndProc:=@WindowProc;
  WindowClass.cbClsExtra:=0;
  WindowClass.cbWndExtra:=0;
  // Связываем наше окно с идентификатором нашей программы
  WindowClass.hInstance:=HInstance;
  // Регистрируем наш значок
  WindowClass.hIcon:=LoadIcon(hInstance,'MAINICON');
  // Курсор в виде стрелки
  WindowClass.hCursor:=LoadCursor(0, idc_Arrow);
  // Цвет кисти для закраски фона - прозрачный (без закраски)
  WindowClass.hbrBackground:=HBrush(GetStockObject(NULL_BRUSH));
  // Стандартного Windows-меню нет
  WindowClass.lpszMenuName:=nil;
  // Название программы
  WindowClass.lpszClassName:=AppName;
  Result:=RegisterClass(WindowClass) <> 0;
end;

{ Создаем окно }
function WinCreate: HWnd;
begin
  hWindow:=CreateWindowEx (WS_EX_TOPMOST,AppName,AppName,ws_PopUp,0,0,
   GetSystemMetrics(SM_CXSCREEN),GetSystemMetrics (SM_CYSCREEN),
   0,0,HInstance,nil);
  if hWindow <> 0 then
    begin
    // Отображаем окно
    ShowWindow (hWindow,SW_SHOW);
```

```
    // Даем команду перерисовать его
    UpdateWindow (hWindow);
    end;
  Result:=hWindow;
end;

BEGIN
    // При повторном запуске программы активизируем запущенный экземпляр
    hWindow:=FindWindow(AppName,AppName);
    if hWindow > 0 then
      begin
      SendMessage(hWindow,WM_SETFOCUS,hWindow,0);
      ShowWindow(hWindow,SW_SHOWNORMAL);
      Halt;
      end;
    if not WinRegister then Exit;
    // Запоминаем handle нашего окна
    hWindow:=WinCreate;
    if hWindow = 0 then Exit;
    { Узнаем путь к своим файлам }
    path:=ParamStr(0);
    for i:=Length(path) downto 1 do
    if path[i] <> '\' then SetLength(path,Length(path)-1) else break;
    { Создаем DirectDraw COM object и выбираем режим с нужным bpp
      (bytes per pixel) }
      if DDrawInit (hWindow,SCRX,SCRY) <> DD_OK then
        begin
ERR:    MessageBox(HWindow,'Can''t create Directdraw object',AppName,
          mb_OK or mb_ICONSTOP);
        Halt;
        end;
    { Создаем экранную поверхность, поверхность для картинок и
      вспомогательную поверхность }
    if CreateSurf (hWindow,1,SCRX,SCRY,PRIMARYSURF,FALSE) <> DD_OK then
      goto ERR;
```

```
if (CreateSurf (hWindow,2,SCRX,SCRY,OFFSCREENSURF,FALSE) <> DD_OK)
 and (CreateSurf (hWindow,2,SCRX,SCRY,MEMORYSURF,FALSE) <> DD_OK)
 then goto ERR;
if (CreateSurf (hWindow,3,SCRX,SCRY,OFFSCREENSURF,FALSE) <> DD_OK)
 and (CreateSurf (hWindow,3,SCRX,SCRY,MEMORYSURF,FALSE) <> DD_OK)
 then goto ERR;
// Системный курсор нам не нужен, прячем его
ShowCursor(FALSE);
// Читаем конфигурационный файл
GetCfg;
// Если в нем звук разрешен, то инициализируем его
if soundenable then InitSound;
// Если в нем MIDI разрешено, то инициализируем его
if midienable then InitMidi;
{ Вычисляем частоту системного таймера }
QueryPerformanceFrequency(qpfreq);
qpfreq2:=qpfreq div 2;
Splash;
InitGame;
// Запоминаем показание счетчика тиков таймера
tick:=GetQPcount;
lasttick:=tick;
// Основной цикл работы программы
repeat
  if bIsActive then
    begin
    { Если первичная поверхность потеряна (при Alt+Tab, Ctrl+Esc),
    то восстанавливаем все поверхности DirectDraw }
    if surfrec[1].surf.IsLost <> DD_OK then RestoreSurfaces;
    { Даем программе работать только 2 раза в секунду для показа
      мигания каретки при вводе имени }
    tick:=GetQPCount;
    if tick > lasttick+qpfreq2 then
      begin
      Inc (tc,qpfreq2);
      Go;
```

```
          lasttick:=tick;
          end;
        end;
      // Извлекаем сообщение из очереди нашего окна
      if GetMessage (Message,0,0,0) then
        begin
        // Обрабатываем сообщение (вырабатываем wm_Char и т. д.)
        TranslateMessage (Message);
        DispatchMessage (Message);
        end else
         Halt (Message.wParam);
    until FALSE;
end.
```

Многое из него нам уже знакомо. В функции `WindowProc` появился новый обработчик события `wm_timer`. Это события от таймера, который, аналогично часам DOS, тикает с частотой 18,2 раза в секунду. Такие же таймеры создаются в интегрированной среде разработки Delphi. В Windows API таймер создается функцией `SetTimer`, а уничтожается вызовом `KillTimer`.

Из событий мыши реально используются лишь `wm_lbuttondown` и `wm_MouseMove`, обработка остальных закомментирована. Когда нажата левая кнопка мыши, мы запоминаем это в переменной `moleft` (реально игра эту переменную не использует) и вызываем процедуру `Process` для обработки событий игры:

```
if bIsActive then Process (lbdown,#0,LoWord(lParam),HiWord(lParam));
```

В первом параметре передается код события, во втором — код нажатой клавиши или нулевой символ, если это событие не связано с клавиатурой, в последних двух — координаты указателя мыши (для событий мыши).

Перемещение указателя `wm_MouseMove` требует его перерисовку, за которую отвечает модуль mouse.pas. У нас используется многофазный спрайт мыши, поэтому номер текущей фазы мы храним в переменной `mophase`. Кроме того, показом указателя управляют переменные:

- `moprohib` (показывать указатель запрещено);
- `mostart` (`TRUE`, если указатель впервые появился на экране);
- в `mox` и `moy` программа запоминает координаты указателя;
- переменная `trig` замедляет смену фаз в два раза, чтобы указатель не менялся при каждом перемещении на одну точку;
- `movis` — логический признак того, что указатель виден.

Если указатель не виден, то программа просто запоминает координаты указателя в mox и moy и циклически меняет номер фазы указателя. Если указатель виден (movis установлена в TRUE), то идет ветвление по уровню игры (переменной level). Если игрок играет на первом уровне, то с помощью функции MouseInBox мы проверяем, находится ли "горячая точка" указателя над игровым полем или нет. Если да, то вызываем процедуру Check1 для дальнейшего анализа относительно показа или сокрытия подсказки (выделения соответствующей возможной кости). Иначе, если указатель не над доской, то надо убрать подсказку (hint), которая показывается, если переменная hintx больше или равна нулю. Это делает процедура RemoveHint1. Аналогичный код отрабатывает при игре на втором уровне. И все это происходит при обратном ходе лучей по экрану. Перед всеми этими проверками программа прячет указатель кодом

```
DirectDraw.WaitForVerticalBlank (DDWAITVB_BLOCKEND,0);
MouseOff;
```

(ожидается, пока не начнется обратный ход лучей, и прячется указатель). А после этого указатель рисуется.

При поступлении события wm_Char вызываем процедуру Process с передачей типа события и виртуального кода нажатой клавиши:

```
if bIsActive then Process (keyd,char(wParam),0,0);
```

Если нажималась клавиша <Esc>, то:

- во время игры и после ее окончания программа переходит к показу таблицы рекордов;
- во время показа таблицы рекордов завершается программа;
- во время показа правил игры происходит возврат в предыдущее состояние (GAMES или GAMEOVER);
- если происходит ввод имени пользователя, то просто передается код нажатой клавиши в метод EName.Enter.

Если нажата другая клавиша, то при показе правил игры, таблицы рекордов или ввода имени происходит то же самое, что и в случае нажатия клавиши <Esc>.

Если нажата левая кнопка мыши, то:

- во время показа правил игры происходит возврат в игру или состояние GAMEOVER;
- при показе рекордов происходит завершение программы;
- если состояние программы GAMES, GAMEOVER или NAME, то вызывается процедура CheckMenu для проверки, не щелкнул ли игрок на каком-либо пункте меню. Если да, то этот щелчок отрабатывается. Кроме того, если

состояние игры GAMES, то вызывается процедура MouseSquare1 или MouseSquare2 с помощью элемента MouseSquare[level] — массива указателей на эти процедуры.

Если приходит событие от таймера wm_timer, то наращивается счетчик biosct тиков таймера BIOS, и если программа активна, делается вызов

```
if bIsActive then Process (tic,#0,0,0);
```

А в этой процедуре выполняется ветвь

```
tic:      if GameState = GAMES then STime;
```

Процедура STime показывает время игры, если оно не достигло значения 999, и играет довольно неприятный звук, если текущее время игрока превысило худшее время из последней строки таблицы рекордов, предупреждая, что игрок уже не попадает в нее.

Вместо обработки сообщения wm_timer можно было бы засекать время внутри цикла обработки сообщений.

Появилось новое сообщение mm_MCINOTIFY. Это сообщение посылает драйвер MCI, когда он закончил играть заданную мелодию. По этому сообщению программа циклически запускает следующую мелодию вызовом PlayMidi.

Сообщение wm_Destroy мы уже обсуждали в примерах.

Программа создает для своих нужд три поверхности DirectDraw, обрабатывает свой миниатюрный конфигурационный файл вызовом GetCfg, узнавая, разрешен ли звук, музыка и какой должен быть уровень игры. Затем в случае надобности инициализируется звук и MIDI вызовами

```
if soundenable then InitSound;
if midienable then InitMidi;
```

Наша игра в основном работает по получаемым сообщениям, но есть одно состояние, когда она должна работать также и при их отсутствии, это состояние NAME, когда программа показывает мерцающую каретку в строке ввода имени игрока. Для этого вычисляется частота системного таймера вызовом

```
QueryPerformanceFrequency(qpfreq);
```

и программа узнает, сколько тиков этого таймера происходит за полсекунды. В цикле обработки сообщений через каждые полсекунды вызывается процедура Go, которая делает каретку попеременно то видимой, то невидимой. Конечно, эту работу можно было бы повесить на обработчик сообщения wm_timer и не вычислять частоту системного таймера. Но в динамической игре бывает нужно перемещать спрайты чаще, чем приходят сообщения от таймера, вот здесь не обойтись без счетчика высокого разрешения.

Перед входом в цикл обработки сообщений вызывается процедура показа заставки `Splash` и за ней — инициализация игры `InitGame`.

3.7.3. Модуль Midi.pas

Этот модуль (листинг 3.9) ищет файлы с расширением mid в каталоге игры и затем играет эти мелодии циклически, если разрешена музыка. Для этого используются вызовы Windows API.

Листинг 3.9. Модуль Midi.pas

```
{$A-,B-,I-,R-,S-,X+,H+}
UNIT Midi;
INTERFACE
USES  Windows;

function   InitMidi : boolean;
procedure  FindMidi;
procedure  StopMidi;
procedure  CloseMidi;
procedure  PlayMidi;

IMPLEMENTATION
USES  SysUtils,MMSystem,DDrawUnit,Glob;

CONST
  // Максимальное число MIDI-файлов
  MAXMIDI = 10;
VAR
  midifiles : array[1.. MAXMIDI] of string;
  numfiles : integer = 0;
  curmidi : integer = 0;

VAR
  mciOpenParms : TMCI_Open_Parms;
  mciPlayParms : TMCI_Play_Parms;
  wDeviceID    : word;
  s : string;
```

```
{ Запись имен MIDI-файлов из каталога path в midifiles }
procedure  FindMidi;
LABEL 10;
VAR
  sr : TSearchRec;
begin
    numfiles:=0;
    if FindFirst (PChar(path+'*.mid'),faReadOnly+faArchive,sr) <> 0
     then goto 10;
    repeat
      if numfiles >= MAXMIDI then break;
      Inc (numfiles);
      midifiles[numfiles]:=sr.name;
    until FindNext (sr) <> 0;
10: FindClose (sr);
    IOResult;
end; {FindMidi}

// Создает по номеру следующего файла его полное имя
function  MakeName : PChar;
begin
  Inc (curmidi);
  if curmidi > numfiles then curmidi:=1;
  s:=path+midifiles[curmidi];
  MakeName:=@s[1];
end; {MakeName}

// Инициализация MIDI-устройства
function  InitMidi : boolean;
begin
  InitMidi:=FALSE;
  { Есть MIDI-устройство? }
  if midiOutGetNumDevs = 0 then Exit;
  InitMidi:=TRUE;
  midienable:=TRUE;
  midiprohib:=FALSE;
end;
```

```
// Останов проигрывания мелодии
procedure  StopMidi;
begin
  if midienable then
    mciSendCommand (wDeviceID,mci_stop,0,Integer(0));
end; {StopMidi}

// Закрытие MIDI-устройства
procedure  CloseMidi;
begin
  if midienable then
    mciSendCommand (wDeviceID,mci_close,0,Integer(0));
end; {StopMidi}

// Играет очередной MIDI-файл
procedure  PlayMidi;
begin
  if not midienable then Exit;
  // На всякий случай освободим проигрыватель
  StopMidi;
  CloseMidi;
  FillChar (mciOpenParms,SizeOf(TMCI_Open_Parms),0);
  FillChar (mciPlayParms,SizeOf(TMCI_Play_Parms),0);
  with mciOpenParms do
    begin
    lpStrDeviceType:='sequencer';
    lpstrElementName:=MakeName;
    end;
  // Посылаем команду играть файл
  if mciSendCommand (0,MCI_OPEN,MCI_OPEN_TYPE or MCI_OPEN_ELEMENT,
    Integer(@mciOpenParms)) = 0 then
    begin
    wDeviceID:=mciOpenParms.wDeviceID;
    mciPlayParms.dwCallback:=hWindow;
```

```
    mciSendCommand(wDeviceID,MCI_PLAY,MCI_NOTIFY,
      Integer(@mciPlayParms));
    end;
end; {PlayMidi}

end.
```

Для поиска и записи имен mid-файлов в массив `midifiles` вызывается процедура `FindMidi`. Для этой цели она использует вызовы `FindFirst` и `FindNext` из библиотечного модуля `SysUtils`. Вы можете также использовать непосредственно вызовы Windows API `FindFirstFile` и `FindNextFile`. Delphi 7 при трансляции почему-то предупреждает, что константы `faReadOnly` и `faArchive` специфичны для платформы, а если заменить их на аналогичные битовые флаги `FILE_ATTRIBUTE_ARCHIVE` и `FILE_ATTRIBUTE_READONLY` из справки Windows для этих процедур Windows API, то такого предупреждения уже не возникает.

Функция `MakeName` возвращает полное имя очередного файла для проигрывания.

Функция `InitMidi` просто проверяет наличие в системе MIDI-устройства вызовом `midiOutGetNumDevs`. Инициализировать его не нужно, оно готово к работе.

Управление драйвером MCI осуществляется посылкой ему команд функцией `mciSendCommand`. Например, для останова проигрывания мелодии надо послать константу `mci_stop` вызовом

```
mciSendCommand (wDeviceID,mci_stop,0,Integer(0))
```

Идентификатор `wDeviceID` программа получает от вызова `mciSendCommand` в процедуре `PlayMidi` для проигрывания мелодии. Среди параметров надо задать handle нашего окна:

```
mciPlayParms.dwCallback:=hWindow;
```

и константу `MCI_NOTIFY` для получения мультимедиасообщений `mm_MCINOTIFY` при окончании очередной мелодии. Больше в этом модуле ничего сложного нет.

3.7.4. Модуль Sound.pas

Модуль Sound.pas (листинг 3.10) обеспечивает нам воспроизведение звуковых файлов формата WAV.

Листинг 3.10. Модуль Sound.pas

```
{ Модуль для проигрывания WAV-файлов через DSound }
{$A-,B-,I-,R-,S-,X+,H+}
UNIT  Sound;
INTERFACE

function   InitSound : boolean;
procedure  PlaySound (numf : longint);
procedure  DestroySound;

IMPLEMENTATION

USES
  Windows,DSound,MMSystem,Glob,Util;
CONST
  MAXWAWS = 10;         { Число вторичных звуковых буферов }

VAR
  DirectSound           : IDirectSound;
  DirectSoundBuffer     : IDirectSoundBuffer;
  SecondarySoundBuffer  : array[0..MAXWAWS-1] of IDirectSoundBuffer;
  nextbuf               : integer = 0;
  s                     : string;

function  MakeName (numf : integer) : string;
VAR
  i : integer;

begin
  s:='00.wav';
  i:=2;
  while numf > 0 do
    begin
    Inc (s[i],numf mod 10);
    numf:=numf div 10;
```

```
    Dec (i);
    end;
  s:=path+s;
  MakeName:=s;
end; {MakeName}

function  AppWriteDataToBuffer (Buffer: IDirectSoundBuffer;
  OffSet: DWord; var SoundData; SoundBytes: DWord) : boolean;
VAR
  AudioPtr1,AudioPtr2 : Pointer;
  AudioBytes1,AudioBytes2 : DWord;
  H : dword;
  Temp : Pointer;

begin
  AppWriteDataToBuffer:=FALSE;
  H:=Buffer.Lock (OffSet, SoundBytes, AudioPtr1, AudioBytes1,
    AudioPtr2, AudioBytes2, 0);
  if H = DSERR_BUFFERLOST  then
    begin
    Buffer.Restore;
    if Buffer.Lock (OffSet, SoundBytes, AudioPtr1, AudioBytes1,
      AudioPtr2, AudioBytes2, 0) <> DS_OK then Exit;
    end else
      { Ошибка в Lock }
      if H <> DS_OK then Exit;
  Temp:=@SoundData;
  Move(Temp^, AudioPtr1^, AudioBytes1);
  if AudioPtr2 <> nil then
    begin
    Temp:=@SoundData;  Inc(Integer(Temp), AudioBytes1);
    Move(Temp^, AudioPtr2^, AudioBytes2);
    end;
  { Ошибка в UnLock }
  if Buffer.UnLock(AudioPtr1,AudioBytes1,AudioPtr2,
    AudioBytes2) <> DS_OK then Exit;
```

```
  AppWriteDataToBuffer:=TRUE;
end;

function  AppCreateWritePrimaryBuffer : boolean;
VAR
  BufferDesc : DSBUFFERDESC;
  PCM : TWaveFormatEx;

begin
  AppCreateWritePrimaryBuffer:=FALSE;
  FillChar(BufferDesc, SizeOf(DSBUFFERDESC),0);
  FillChar(PCM, SizeOf(TWaveFormatEx),0);
  with BufferDesc do
    begin
    PCM.wFormatTag:=WAVE_FORMAT_PCM;
    PCM.nChannels:=1;
    { Частота дискретизации звука }
    PCM.nSamplesPerSec:=22050;
    PCM.nBlockAlign:=4;
    PCM.nAvgBytesPerSec:=PCM.nSamplesPerSec*PCM.nBlockAlign;
    { Устанавливаем 16-битовый звук, для 8-битовых WAV-файлов
      это тоже работает }
    PCM.wBitsPerSample:=16;
    PCM.cbSize:=0;
    dwSize:=SizeOf(DSBUFFERDESC);
    dwFlags:=DSBCAPS_PRIMARYBUFFER;
    dwBufferBytes:=0;
    lpwfxFormat:=nil;
    end;
  { На некоторых звуковых картах установка CooperativeLevel
    не проходит, но программа работает }
  if (DirectSound.SetCooperativeLevel (hInstance,
    DSSCL_PRIORITY) <> DS_OK) then;
  if DirectSound.CreateSoundBuffer (BufferDesc,DirectSoundBuffer,
    nil) <> DS_OK then
    begin
```

```
    Message ('Can''t initialize sound. Playing without sound.',
      MB_OK);
    Exit;
    end;
  { Это не на всех звуковых платах проходит, но не влияет
    на работу программы }
  if (DirectSoundBuffer.SetFormat(PCM) <> DS_OK) then;
  { Это тоже может не пройти, но не влияет на работу }
  if DirectSound.SetCooperativeLevel (hwindow,DSSCL_NORMAL)
    <> DS_OK then;
  AppCreateWritePrimaryBuffer:=TRUE;
end;

function AppCreateWriteSecondaryBuffer (var Buffer :
  IDirectSoundBuffer; SamplesPerSec : Integer; Bits : Word;
  isStereo : Boolean; Time : dword) : boolean;
VAR
  BufferDesc : DSBUFFERDESC;
  PCM : TWaveFormatEx;

begin
  AppCreateWriteSecondaryBuffer:=FALSE;
  FillChar(BufferDesc, SizeOf(DSBUFFERDESC),0);
  FillChar(PCM, SizeOf(TWaveFormatEx),0);
  with BufferDesc do
    begin
    PCM.wFormatTag:=WAVE_FORMAT_PCM;
    if isStereo then PCM.nChannels:=2 else PCM.nChannels:=1;
    PCM.nSamplesPerSec:=SamplesPerSec;
    PCM.nBlockAlign:=(Bits div 8)*PCM.nChannels;
    PCM.nAvgBytesPerSec:=PCM.nSamplesPerSec*PCM.nBlockAlign;
    PCM.wBitsPerSample:=Bits;
    PCM.cbSize:=0;
    dwSize:=SizeOf(DSBUFFERDESC);
    dwFlags:=DSBCAPS_STATIC;
    dwBufferBytes:=Time*PCM.nAvgBytesPerSec;
```

```
    lpwfxFormat:=@PCM;
    end;
  if DirectSound.CreateSoundBuffer (BufferDesc,Buffer,nil)
    <> DS_OK then
    begin
    Message ('Can''t create sound buffer. Playing without sound.',
      MB_OK);
    Exit;
    end;
  AppCreateWriteSecondaryBuffer:=TRUE;
end;

function  CreateSecondaryBufferFromWav (var Buffer:
  IDirectSoundBuffer; Name : string) : boolean;
LABEL  10;
VAR
  Data : PChar;
  F : File;
  BufferDesc : DSBUFFERDESC;
  PCM : TWaveFormatEx;
  FWord : Word;
  FDWord,DataSize : DWord;
  Chunk : String[4];
  Pos : Integer;

begin
  CreateSecondaryBufferFromWav:=FALSE;
  AssignFile (F,name);  Reset (F,1);
  if IOResult <> 0 then Exit;
  FillChar(BufferDesc, SizeOf(DSBUFFERDESC), 0);
  FillChar(PCM, SizeOf(TWaveFormatEx), 0);
  with BufferDesc do
    begin
    PCM.wFormatTag:=WAVE_FORMAT_PCM;
    Seek(F,$16);
    BlockRead (F,FWord, SizeOf(Word));
```

```
  CheckIO;
  PCM.nChannels:=FWord;
  BlockRead(F,FDWord, SizeOf(DWord));
  CheckIO;
  PCM.nSamplesPerSec:=FDWord;
  BlockRead(F,FDWord, SizeOf(DWord));
  CheckIO;
  PCM.nAvgBytesPerSec:=FDWord;
  BlockRead(F,FWord, SizeOf(Word));
  CheckIO;
  PCM.nBlockAlign:=FWord;
  BlockRead (F,FWord, SizeOf(Word));
  CheckIO;
  PCM.wBitsPerSample:=FWord;
  PCM.cbSize:=0;
  dwSize:=SizeOf(DSBUFFERDESC);
  dwFlags:=DSBCAPS_STATIC;
  Pos:=$22;
  SetLength(Chunk,4);
  repeat
    Seek(F,Pos);
    BlockRead(F,Chunk[1], 4);
    CheckIO;
    Inc(Pos);
  until Chunk = 'data';
  Seek(F,Pos + 3);
  CheckIO;
  BlockRead (F,DataSize, SizeOf(DWord));  CheckIO;
  GetMem(Data,DataSize);
  BlockRead(F,Data^, DataSize);
  CheckIO;
  dwBufferBytes:=DataSize;
  lpwfxFormat:=@PCM;
  end;
{ Не могу создать звуковой буфер }
if DirectSound.CreateSoundBuffer (BufferDesc, Buffer, nil)
```

```
    <> DS_OK then goto 10;
  if not AppWriteDataToBuffer(Buffer, 0, Data^, DataSize) then
    goto 10;
  CreateSecondaryBufferFromWav:=TRUE;
10:
  FreeMem(Data, DataSize);
  CloseFile (F);
end; {CreateSecondaryBufferFromWav}

// Играет звуковой файл по номеру в его имени
procedure  PlaySound (numf : longint);
LABEL 10;

begin
  if not soundenable then Exit;
  if Assigned (SecondarySoundBuffer[nextbuf]) then
    begin
    SecondarySoundBuffer[nextbuf].Stop;
    SecondarySoundBuffer[nextbuf].Release;
    end else goto 10;
  if not CreateSecondaryBufferFromWav (SecondarySoundBuffer[nextbuf],
    MakeName (numf)) then
    begin
10: soundenable:=FALSE;
    soundprohib:=TRUE;
    Exit;
    end;
  { Попытка сыграть WAV }
  if SecondarySoundBuffer[nextbuf].Play(0,0,0) <> DS_OK then goto 10;
  nextbuf:=(nextbuf+1) mod MAXWAWS;
end;

{ Инициализация объекта DSound }
function  InitSound : boolean;
VAR  i : integer;
begin
```

```
  InitSound:=FALSE;
  if DirectSoundCreate (NIL,DirectSound,NIL) <> DS_OK then Exit;
  AppCreateWritePrimaryBuffer;
  for i:=0 to MAXWAWS-1 do
    if not AppCreateWriteSecondaryBuffer (SecondarySoundBuffer[i],
      22050,8,False,1) then Exit;
  InitSound:=TRUE;
  soundenable:=TRUE;
  soundprohib:=FALSE;
end; {InitSound}

{ Освобождаем звуковые буферы }
procedure  DestroySound;
VAR
  i : integer;

begin
  if soundprohib then Exit;
  for i:=0 to MAXWAWS-1 do
    if Assigned (SecondarySoundBuffer[i]) then
      SecondarySoundBuffer[i].Release;
  if Assigned (DirectSoundBuffer) then DirectSoundBuffer.Release;
  { Освобождаем DirectSound COM object }
  if Assigned (DirectSound) then DirectSound.Release;
  DirectSound:=NIL;
end;

end.
```

Изучать теорию воспроизведения звука через библитотеку DirectSound и формат файлов WAV нет нужды, модуль от этого лучше работать не станет. Это просто пример из документации, приспособленный для своих нужд. В простых случаях вместо DirectSound вы можете использовать вызов Windows API `sndPlaySound`.

В модуле Sound.pas задана константа

```
MAXWAWS = 10;
```

Это число вторичных звуковых буферов, а также число одновременно проигрываемых звуков, которые смешиваются смесителем.

Для удобства программа использует файлы с именами вида *XX*.wav, где *XX* — десятичный номер звука. В этом случае нашей процедуре `PlaySound` можно передавать в качестве параметра не строку с именем файла, а просто число. А функция `MakeName` переводит его в полное имя файла, которое задается соответствующей процедуре библиотеки DirectSound.

Поясню некоторые моменты.

В функции `AppCreateWritePrimaryBuffer PCM.nChannels:=1` означает, что играемый звук монофонический, `PCM.nSamplesPerSec:=22050` означает частоту дискретизации звука, которую показывает звуковой редактор или некоторые программы воспроизведения звуков. В разбираемой игре используются 8-битные звуки с частотой дискретизации не выше 22 050 Гц, которые дают нормальное качество звучания. Можно задавать в качестве параметров более высокие характеристики (16 бит и большую частоту дисткретизации), это будет работать.

Я заметил, что метод `DirectSoundBuffer.SetFormat(PCM)` на некоторых звуковых картах возвращает ошибку, но это не влияет на работу, поэтому я игнорирую возвращаемое им значение. То же самое относится к вызову `DirectSound.SetCooperativeLevel(hwindow,DSSCL_NORMAL)`. Почему-то не на всех звуковых картах установка кооперативного режима завершается без ошибки. Вероятно, это относится к старым звуковым картам.

3.7.5. Модуль Mouse.pas

С помощью модуля Mouse.pas (листинг 3.11) игра отображает и прячет собственный указатель мыши, а также проверяет нахождение "горячей точки" указателя в заданном прямоугольнике.

Листинг 3.11. Модуль Mouse.pas

```
{$A-,B-,I-,R-,S-,X+,H+}
UNIT  Mouse;
INTERFACE
USES  Windows;

VAR
  mohotx,mohoty,moline,modx,mody,mophase,mophases,moperline,
  mox,moy,mooldx,mooldy,moolddx,moolddy,mobkgx,mobkgy : integer;
```

```
  // Показывается ли указатель
  movis : boolean = FALSE;
  // Запрещено ли показывать указатель
  moprohib : boolean = TRUE;
  mostart : boolean = FALSE;
  moleft : boolean = FALSE;
  moright : boolean = FALSE;
  psurf,mosurf : integer;

procedure  MouseOn;
procedure  MouseOff;
function  MouseInBox (x,y,dx,dy : integer) : boolean;

IMPLEMENTATION
USES
  Glob,DDraw,DdrawUnit,Util;

// Рисует курсор. Координаты левого верхнего угла - mox, moy
procedure  MouseOn;
VAR
  ARect1,ARect2 : Trect;
  x2,y2,dx1,dy1 : integer;

begin
  if movis or moprohib then Exit;
  moprohib:=TRUE;
  mooldx:=mox-mohotx;
  mooldy:=moy-mohoty;
  x2:=mooldx+modx;
  y2:=mooldy+mody;
  moolddx:=modx;
  moolddy:=mody;
  dx1:=0;
  dy1:=0;
  if mooldx < 0 then
```

```
    begin
    Inc (moolddx,mooldx);
    dx1:=-mooldx;
    mooldx:=0;
    end;
  if mooldy < 0 then
    begin
    Inc (moolddy,mooldy);
    dy1:=-mooldy;
    mooldy:=0;
    end;
  if x2 > SCRX then Dec (moolddx,x2-SCRX);
  if y2 > SCRY then Dec (moolddy,y2-SCRY);
  ARect1.left:=mophase mod moperline*modx+dx1;
  ARect1.top:=mophase div moperline*mody+moline+dy1;
  ARect1.right:=ARect1.left+moolddx;
  ARect1.bottom:=ARect1.top+moolddy;
  // Восстанавливаем фон под указателем
  ARect2:=Rect (mooldx,mooldy,mooldx+moolddx,mooldy+moolddy);
  repeat
  until RespHandler (surfrec[mosurf].surf.BltFast(mobkgx,mobkgy,
   surfrec[psurf].surf,ARect2,DDBLTFAST_NOCOLORKEY));
  // Рисуем указатель на новом месте
  repeat
  until RespHandler(surfrec[psurf].surf.BltFast(mooldx,mooldy,
   surfrec[mosurf].surf,ARect1,DDBLTFAST_SRCCOLORKEY));
  moprohib:=FALSE;
  movis:=TRUE;
end;

// Убирает указатель мыши
procedure  MouseOff;
VAR
  ARect : TRect;

begin
  if moprohib or not movis then Exit;
```

```
  ARect:=Rect (mobkgx,mobkgy,mobkgx+moolddx,mobkgy+moolddy);
  repeat
  until RespHandler(surfrec[psurf].surf.BltFast(mooldx,mooldy,
   surfrec[mosurf].surf,ARect,DDBLTFAST_NOCOLORKEY));
  movis:=FALSE;
end;

{ Определяет, находится ли "горячая точка" мыши в прямоугольнике
  с координатами левого верхнего угла x, y и длинами dx и dy точек }
function  MouseInBox (x,y,dx,dy : integer) : boolean;
begin
  MouseInBox:=(mox >= x) and (moy >= y) and (mox < x+dx)
                         and (moy < y+dy);
end; {MouseInBox}

end.
```

В процедуре `MouseOn`, которая выводит спрайт указателя, происходит нудное определение, виден ли спрайт полностью или его часть выходит за правый или нижний край экрана. В этом случае нужно выводить не весь прямоугольник спрайта, а его часть. Правило такое: "горячая точка" всегда должна быть на экране, причем пользователь должен иметь возможность поместить ее в любое место.

3.7.6. Модуль Fonts.pas

Аналогичный модуль я использовал еще в старых играх под DOS, а здесь он позволяет выводить текст своим шрифтом и с тенью для лучшей читабельности. Я не рекомендую работать напрямую с видеопамятью, используйте для этого спрайты букв и метод `BltFast`.

В игре Domino Dilemma используется растровый шрифт гарнитуры Times, подготовленный в файле times16 специальной программой, которая читает нарисованные символы из BMP-файла. Первые пять байтов в файле times16 служебные. В нулевом и первом байтах хранится ширина в байтах линии растра каждого символа. Если какая-либо буква, например, W, имеет ширину больше восьми точек, то для всех символов нужно использовать два байта на одну линию точечного представления символа. Следующие два байта — это высота знакоместа символа или число скан-линий в нем. И последний байт заголовка — числовой код первого символа в этом шрифте.

Если программе не нужно выводить символы с кодами меньше пробела, то это будет число 32 — ASCII-код пробела. Далее в файле идут битовые представления линий для всех заявленных символов до символа с кодом 255. С помощью шестнадцатеричного редактора вы можете легко понять и воспроизвести графические образы символов.

Для использования шрифта его файл целиком читается в ОЗУ и процедуре для вывода текста задается указатель на эту область памяти. Этот модуль также может вычислять ширину графического представления заданной строки в точках. `x` и `y` означают экранные координаты левого верхнего угла матрицы символа, в данном случае прямоугольника 16 × 21.

Текст модуля представлен в листинге 3.12.

Листинг 3.12. Модуль Fonts.pas

```
{$A-,B-,I-,R-,S-,X+,H+}
UNIT  Fonts;
INTERFACE

{ Вывод строки s шрифтом из pf цветом col с тенью в координатах
  x,y поверхности с адресом psurf. letterspace - число точек,
  разделяющих соседние символы, linespace - шаг в точках между
  началами соседних линий в поверхности вывода }
procedure  TextXY (x,y,linespace,letterspace,col: integer;
 pf,psurf: pointer; s: string);
{ Выдача ширины данного текста в точках с учетом
  пропусков + 1 точка на тень }
function  TextWidth (letterspace: integer; pf: pointer;
 s: string): integer;

IMPLEMENTATION
USES  Glob,DDRawunit,Windows;

procedure  TextXY (x,y,linespace,letterspace,col: integer;
 pf,psurf: pointer; s: string);
VAR
  i,j,l,k,symbolwidth,symbolheight,symbolheight1,ofscr,
   ofscr0,offnt,code0,tcode,widthmulheight,l8_1,linespace1,
   stepblack,ls: integer;
  lin: byte;
```

```
begin
  symbolwidth:=tpmw(pf)^[0];
  symbolheight:=tpmw(pf)^[1];
  code0:=tpmb(pf)^[4];
  ofscr:=linespace*y+x*integer(bytepp);
  widthmulheight:=symbolheight*symbolwidth;
  linespace1:=linespace-symbolwidth*8*integer(bytepp);
  { Если col = 0, то тень не делаем }
  stepblack:=linespace+integer(bytepp);
  if col = 0 then stepblack:=0;
  { Цикл по символам }
  ls:=Length(s);
  for i:=1 to ls do
    begin
    tcode:=integer(s[i])-code0;
    symbolheight1:=0;
    ofscr0:=ofscr;
    offnt:=5+tcode*widthmulheight;
    { Цикл по линиям i-го символа }
    for j:=0 to symbolheight-1 do
      begin
      l8_1:=1;
      { Цикл по байтам j-ой линии i-го символа }
      for l:=0 to symbolwidth-1 do
        begin
        lin:=tpmb(pf)^[offnt];
        Inc (offnt);
        { Цикл по битам l-го байта j-ой линии i-го символа }
        for k:=0 to 7 do
          begin
          if lin and $80 > 0 then
            begin
            tpmb(psurf)^[ofscr+stepblack]:=1;
            tpmb(psurf)^[ofscr+stepblack+1]:=1;
            tpmb(psurf)^[ofscr+stepblack+2]:=1;
            tpmb(psurf)^[ofscr]:=col;
```

```
            tpmb(psurf)^[ofscr+1]:=col shr 8;
            tpmb(psurf)^[ofscr+2]:=col shr 16;
            if symbolheight1 < k+l8_1 then symbolheight1:=k+l8_1;
            end;
          asm
          shl lin,1
          end;
          Inc (ofscr,integer(bytepp));
          end;
        Inc (l8_1,8);
        end;
      Inc (ofscr,linespace1);
      end;
    if symbolheight1 = 0 then
     symbolheight1:=symbolwidth*4-letterspace;
    ofscr:=ofscr0+(symbolheight1+letterspace+1)*integer(bytepp);
    end;
end; {TextXY}

{ Вычисление ширины в точках для строки s и шрифта pf^ с
  пропуском между символами letterspace }
function  TextWidth (letterspace: integer; pf: pointer;
 s: string): integer;
VAR
  i,j,l,k,symbolwidth,symbolheight,symbolheight1,offnt,code0,
   tcode,widthmulheight,l8_1,res,ls: integer;
  lin: byte;
begin
  symbolwidth:=tpmw(pf)^[0];
  symbolheight:=tpmw(pf)^[1];
  code0:=tpmb(pf)^[4];
  widthmulheight:=symbolheight*symbolwidth;
  res:=0;
  { Цикл по символам }
  ls:=Length(s);
  for i:=1 to ls do
```

```
    begin
    tcode:=integer(s[i])-code0;
    symbolheight1:=0;
    offnt:=5+tcode*widthmulheight;
    { Цикл по линиям i-го символа }
    for j:=0 to symbolheight-1 do
      begin
      l8_1:=1;
      { Цикл по байтам j-ой линии i-го символа }
      for l:=0 to symbolwidth-1 do
        begin
        lin:=tpmb(pf)^[offnt];
        Inc (offnt);
        { Цикл по битам l-го байта j-ой линии i-го символа }
        for k:=0 to 7 do
          begin
          if (lin and $80 > 0) and (symbolheight1 < k+l8_1) then
            symbolheight1:=k+l8_1;
          asm
          shl lin,1
          end;
          end;
        Inc (l8_1,8);
        end;
      end;
    if symbolheight1 = 0 then
     symbolheight1:=symbolwidth*4-letterspace+1;
    Inc (res,symbolheight1+letterspace);
    if s[i] <> ' ' then Inc (res);
    end;
  TextWidth:=res;
end; {TextWidth}

end.
```

3.7.7. Модуль DDrawUnit.pas

Этот модуль (листинг 3.13) является оболочкой для использования библиотеки DirectDraw. Его процедуры и функции уже знакомы нам по предыдущим примерам.

Листинг 3.13. Модуль DDrawUnit.pas

```
{$A-,B-,I-,R-,S-,X+,H+}
UNIT  DDRawUnit;
INTERFACE
USES  Windows,DDraw,Glob;

CONST
  { Типы поверхностей }
  // Экранная
  PRIMARYSURF = 0;
  // Экранная с переключением страниц
  FLIPSURF = 1;
  // Внеэкранная видеопамять
  OFFSCREENSURF = 2;
  // Поверхность в оперативной памяти
  MEMORYSURF = 3;
VAR
  SurfRec : array[1..MAXSURF] of record
    surf : IDirectDrawSurface;
    psurf : pointer;
    surftype,pixx,pixy,pitch : integer;
    exist : boolean;
    end;
  DirectDraw : IDirectDraw;

  function  DDrawInit (hWindow : hWnd; scrx,scry : integer)
    : HResult;
  function  CreateSurf (hWindow : hWnd; nsurf,pixx,pixy,surftype
    : integer; flip : boolean) : HResult;
  procedure  DestroySurf (nsurf : integer);
```

```
  procedure  DestroyAllSurf;
  function  RespHandler( DDResult : HResult ) : boolean;
  function  RestoreSurfaces : HResult;
  { Блокировка поверхности nsurf }
  function  SurfLock (nsurf : integer) : pointer;
  { Разблокировка поверхности nsurf }
  procedure  SurfUnLock (nsurf : integer);

{ Возвращает стуктуру типа Trect по координатам углов
  прямоугольника }
function  Rect (ALeft,ATop,ARight,ABottom : Integer) : Trect;
{ Копирует прямоугольник с координатами углов
  xs1, ys1, xs2, ys2 с поверхности nsurf1 в координаты xd, yd
  поверхности nsurf2 }
procedure CopyRec(xs1,ys1,xs2,ys2,xd,yd,nsurf1,nsurf2: integer);
{ Копирует прямоугольник за исключением точек цвета 0 с
  координатами углов xs1, ys1, xs2, ys2 с поверхности nsurf1
  в координаты xd, yd поверхности nsurf2 }
procedure CopySpr(xs1,ys1,xs2,ys2,xd,yd,nsurf1,nsurf2: integer);

IMPLEMENTATION

USES Mouse,Game,Util;

{ Инициализация COM-объекта IDirectDraw и установка полноэкранного
  графического режима scrx, scry. bpp подбирается! При успешном
  завершении приложение переключается в заданный полноэкранный режим
  и возвращается DD_OK, иначе ничего не производится и возвращается
  ошибка с кодом из DirectDraw }
function  DDrawInit (hWindow : hWnd; scrx,scry : integer) : HResult;
VAR
  i : integer;
  HALCaps, HELCaps : DDCaps;

begin
  for i:=1 to MAXSURF do surfrec[i].exist:=FALSE;
```

```
  { Создаем DirectDraw COM object }
  Result:=DirectDrawCreate (NIL,DirectDraw,NIL);
  if Result <> DD_OK then Exit;
  { Устанавливаем exclusive cooperative level }
  Result:=DirectDraw.SetCooperativeLevel (hWindow,
    DDSCL_EXCLUSIVE or DDSCL_FULLSCREEN);
  if Result <> DD_OK then Exit;
  { Если нет аппаратного ускорения, то выход }
  FillChar (HALCaps,SizeOf(DDCaps),0);
  FillChar (HELCaps,SizeOf(DDCaps),0);
  HALCaps.dwSize:=SizeOf(DDCaps);
  HELCaps.dwSize:=SizeOf(DDCaps);
  DirectDraw.GetCaps (HALCaps,HELCaps);
  if HALCaps.dwCaps and DDCAPS_BLT <> DDCAPS_BLT then
    begin
    Result:=$100;
    Exit;
    end;
  { Устанавливаем графический режим True Color с 32 или 24 битами
    на точку }
  for i:=4 downto 3 do
    begin
    bytepp:=i;
    Result:=DirectDraw.SetDisplayMode (scrx,scry,bytepp*8);
    if Result = DD_OK then Exit;
    end;
end; {DDrawInit}

{ Создание поверхности numsurf размером pixx, pixy точек. Если
  video=TRUE, то поверхность создается в видеопамяти. Если
  video=flip=TRUE (только для экранной поверхности), то создается
  комплексная поверхность номер numsurf вместе с поверхностью
  переключения numsurf+1, иначе создается одна поверхность переднего
  плана numsurf. При возможности видеопамяти numsurf также создается
  в ней. При успешном завершении возвращается DD_OK, иначе ничего не
  производится и возвращается ошибка с кодом из DirectDraw. }
```

```
function  CreateSurf (hWindow : hWnd; nsurf,pixx,pixy,
  surftype : integer; flip : boolean) : HResult;
TYPE
  mw = array[0..1000] of word;
  tmw = ^mw;
VAR
   DDSurfaceDesc : ddraw.DDSurfaceDesc;
   ColorKey      : ddraw.DDColorKey;

begin
  Result:=100;
  if (nsurf > MAXSURF) or flip and (surftype = PRIMARYSURF) and
    (nsurf > MAXSURF-1) then Exit;
  { Заполняем DirectDrawSurface descriptor для заданной поверхности }
  FillChar (DDSurfaceDesc,sizeof(DDSurfaceDesc),0);
  with DDSurfaceDesc do
    begin
    dwSize:=sizeof(DDSurfaceDesc);
    dwFlags:=DDSD_CAPS or DDSD_HEIGHT or DDSD_WIDTH;
    if (surftype = OFFSCREENSURF) or (surftype = FLIPSURF) then
      ddsCaps.dwCaps:=DDSCAPS_OFFSCREENPLAIN;
    if surftype = MEMORYSURF then
      ddSCaps.dwCaps:=DDSCAPS_SYSTEMMEMORY;
    if surftype = PRIMARYSURF then
      begin
      dwFlags:=DDSD_CAPS;
      ddSCaps.dwCaps:=DDSCAPS_PRIMARYSURFACE;
      if flip then
        { Создаем поверхность переключения }
        begin
        dwFlags:=DDSD_CAPS or DDSD_BACKBUFFERCOUNT;
        ddSCaps.dwCaps:=DDSCAPS_COMPLEX or DDSCAPS_FLIP or
          DDSCAPS_PRIMARYSURFACE;
        dwBackBufferCount:=1;
        end;
      end else
```

```
        begin
        dwWidth:=pixx;
        dwHeight:=pixy;
        end;
    Result:=DirectDraw.CreateSurface (DDSurfaceDesc,
      surfrec[nsurf].surf,NIL);
    if Result <> DD_OK then Exit;
    if flip and (surftype = PRIMARYSURF) then
      { Проверяем создание поверхности переключения }
      begin
      DDSCaps.dwCaps:=DDSCAPS_BACKBUFFER;
      if surfrec[nsurf].surf.GetAttachedSurface
        (DDSCaps,surfrec[nsurf+1].surf) <> DD_OK then Exit;
      end;
    end;
  { Устанавливаем ключевой цвет поверхности в 0 }
  ColorKey.dwColorSpaceLowValue:=0;
  ColorKey.dwColorSpaceHighValue:=0;
  Result:=surfrec[nsurf].surf.SetColorKey (DDCKEY_SRCBLT,ColorKey);
  if Result <> DD_OK then Exit;
  surfrec[nsurf].exist:=TRUE;
  surfrec[nsurf].surftype:=surftype;
  surfrec[nsurf].pixx:=pixx;
  surfrec[nsurf].pixy:=pixy;
  if flip and (surftype = PRIMARYSURF) then
    begin
    Result:=surfrec[nsurf+1].surf.SetColorKey
      (DDCKEY_SRCBLT,ColorKey);
    if Result <> DD_OK then Exit;
    surfrec[nsurf+1].exist:=TRUE;
    surfrec[nsurf+1].surftype:=FLIPSURF;
    surfrec[nsurf+1].pixx:=pixx;
    surfrec[nsurf+1].pixy:=pixy;
    end;
end; {CreateSurf}
```

```
{ Уничтожает поверхность с номером nsurf }
procedure  DestroySurf (nsurf : integer);
begin
  if surfrec[nsurf].exist and Assigned (surfrec[nsurf].surf) then
    begin
    surfrec[nsurf].surf.release;
    surfrec[nsurf].exist:=FALSE;
    end;
end; {DestroySurf}

{ Уничтожает все задействованные поверхности }
procedure  DestroyAllSurf;
VAR
  i : integer;

begin
  if Assigned (DirectDraw) then DirectDraw.FlipToGDISurface;
  for i:=1 to MAXSURF do
    if surfrec[i].exist and Assigned (surfrec[i].surf) and
      (surfrec[i].surftype <> FLIPSURF) then
      begin
      surfrec[i].surf.release;
      surfrec[i].exist:=FALSE;
      end;
  { Освобождаем DirectDraw COM object }
  if Assigned (DirectDraw) then DirectDraw.Release;
  DirectDraw:=NIL;
end; {DestroyAllSurf}

{  Восстановление потерянной памяти поверхностей}
function  RestoreSurfaces : HResult;
VAR
  moproh : boolean;
  i : integer;

begin
  moproh:=moprohib;
```

```
  moprohib:=TRUE;
  Result:=DD_OK;
  for i:=1 to MAXSURF do
    if surfrec[i].exist and (surfrec[i].surftype <> FLIPSURF) then
      begin
      Result:=surfrec[i].surf.Restore;
      if Result <> DD_OK then Halt;
      end;
  moprohib:=moproh;
  DrawSurf;
end; {RestoreSurfaces}

function  RespHandler(DDResult : HResult) : boolean;
begin
  case dword(DDResult) of
    DD_OK: Result:=true;
    DDERR_SURFACELOST: Result:=RestoreSurfaces <> DD_OK;
      else Result:=dword(DDResult) <> DDERR_WASSTILLDRAWING;
  end; {case}
end; {RespHandler}

{ Блокировка поверхности nsurf }
function  SurfLock (nsurf : integer) : pointer;
VAR
  DDSurfDesc : ddraw.DDSurfaceDesc;
begin
  FillChar (DDSurfDesc,sizeof(DDSurfDesc),0);
  with DDSurfDesc do
    begin
    dwSize:=sizeof(DDSurfDesc);
    dwFlags:=DDSD_CAPS or DDSD_HEIGHT or DDSD_WIDTH;
    case surfrec[nsurf].surftype of
PRIMARYSURF:
      ddsCaps.dwCaps:=DDSCAPS_FRONTBUFFER;
OFFSCREENSURF,FLIPSURF:
      ddsCaps.dwCaps:=DDSCAPS_OFFSCREENPLAIN;
```

```
MEMORYSURF:
      ddsCaps.dwCaps:=DDSCAPS_OVERLAY;
    end; {case}
    dwWidth:=surfrec[nsurf].pixx;  dwHeight:=surfrec[nsurf].pixy;
    end;
  repeat
  until RespHandler (surfrec[nsurf].surf.Lock
    (nil,DDSurfDesc,DDLOCK_SURFACEMEMORYPTR or DDLOCK_WRITEONLY or
    DDLOCK_READONLY or DDLOCK_WAIT,0));
  // Возвращаем указатель на поверхность
  Result:=DDSurfDesc.lpSurface;
  // И запоминаем его в структуре для этой поверхности
  surfrec[nsurf].psurf:=Result;
  { Также запоминаем шаг между началами соседних скан-линий
    в байтах }
  surfrec[nsurf].pitch:=DDSurfDesc.lpitch;
end; {SurfLock}

{ Разблокировка поверхности nsurf }
procedure  SurfUnLock (nsurf : integer);
begin
  repeat
  until RespHandler(surfrec[nsurf].surf.UnLock(surfrec[nsurf].psurf));
end; {SurfUnlock}

function  Rect (ALeft,ATop,ARight,ABottom : Integer) : Trect;
VAR
  ARect : TRect;

begin
  ARect.left:=Aleft;
  ARect.Top:=ATop;
  ARect.right:=Aright;
  ARect.bottom:=ABottom;
  Result:=ARect;
end; {Rect}
```

```
procedure  CopyRec (xs1,ys1,xs2,ys2,xd,yd,nsurf1,nsurf2 : integer);
VAR
  ARect : TRect;

begin
  ARect:=Rect (xs1,ys1,xs2+1,ys2+1);
    repeat
    until RespHandler (surfrec[nsurf2].surf.Bltfast
      (xd,yd,surfrec[nsurf1].surf,ARect,DDBLTFAST_NOCOLORKEY));
end; {CopyRec}

procedure  CopySpr (xs1,ys1,xs2,ys2,xd,yd,nsurf1,nsurf2 : integer);
VAR
  ARect : TRect;

begin
  ARect:=Rect (xs1,ys1,xs2+1,ys2+1);
    repeat
    until RespHandler (surfrec[nsurf2].surf.Bltfast
      (xd,yd,surfrec[nsurf1].surf,ARect,DDBLTFAST_SRCCOLORKEY));
end; {CopySpr}

end.
```

3.7.8. Модуль Util.pas

В модуле Util.pas (листинг 3.14) собраны вспомогательные процедуры для того, чтобы они не загромождали модуль Game.pas.

Листинг 3.14. Модуль Util.pas

```
{$A-,B-,I-,R-,S-,X+,H+}
UNIT  Util;
INTERFACE
USES  Windows,DDraw,Glob;

{ Возврат меньшего из двух чисел }
function   Min (a1,a2 : integer) : integer;
```

```
{ Возврат большего из двух чисел }
function   Max (a1,a2 : integer) : integer;
{ Выдаем строку s и кнопки "Да" и "Нет". Возврат IDYES или IDNO }
function  Message (s : PChar; attr : integer) : integer;
{ Перевод integer в символы в заданное число позиций (< 33) по
  основанию r. Минус может ставиться при r=10. Если z > 0, то
  пишутся незначащие нули }
function   IntStrFix (a,r,pos,z : integer) : string;
{ Рисуем прямоугольник с левым верхним углом в точке x0, y0
  со сторонами dlx и dly точек
  цветом col в поверхности по адресу psurf }
procedure  FillRec (x0,y0,dlx,dly,col,pitch : integer;
  psurf : pointer);
{ Возврат структуры TRect по координатам вершин прямоугольника }
function   Rect (ALeft,ATop,ARight,ABottom : Integer) : TRect;
{ Обмен байтов местами }
procedure  SwapB (arg1,arg2 : tpmb);
{ Обмен слов местами }
procedure  SwapW (arg1,arg2 : tpmw);
{ Обмен троек байтов местами }
procedure  SwapD (arg1,arg2 : tpmw);
{ Показ уровня игры }
procedure  SetLevel (lev : byte; nsurf : integer);
{ Выводит сообщение о результате игры. Если correct,
  то устанавливает gamestate=GAMEOVER }
procedure  Msg (arg : word; nsurf : integer);
{ Проверка результата файловой операции через IOResult. Если есть
  ошибка, программа аварийно завершается }
procedure  CheckIO;
{ Чтение и обработка конфигурационного файла dominoes.cfg }
procedure  GetCfg;
{ Запись конфигурации в файл dominoes.cfg }
procedure  SaveCfg;
{ Загрузка данного BMP-файла на поверхность DirectDraw }
procedure  LoadBitMap (x1,y1,x2,y2,nsurf : integer;
  BitMapName : string);
```

```
{ Ожидание примерно ms миллисекунд с отработкой (перерисовыванием)
  движения указателя. Если dellbd = TRUE, то нажатия на левую кнопку
  игнорируются. Курсор должен быть инициализирован! }
procedure  Wait (ms : TLargeInteger; dellbd : boolean);
{ Возврат значения счетчика системного таймера (1.19 МГц) }
function  GetQPCount : TLargeInteger;
// Запись файла с именем name по адресу p^
procedure  LoadFileToPoint (name : string; var p : pointer);
{ Запись в файл name ct байтов с адреса buf^. }
procedure  SaveFile (name : string; var buf; ct : integer);
{ Загрузка файла в переменную p^, число байтов ct,
  если такой файл есть }
procedure  LoadFile (name : string; var p; ct : integer);
{ Показ флажка, что звук разрешен, в поверхности номер nsurf }
procedure  ShowSound (nsurf : integer);
{ Показ флажка, что MIDI разрешено, в поверхности номер nsurf }
procedure  ShowMidi (nsurf : integer);
{ Запись поверхности nsurf в файл BMP для отладки. Закомментировано }
// procedure  DiskSurf (nsurf : integer; name : string);

IMPLEMENTATION
USES  Messages,Mouse,DDrawunit,Game,Fonts,Midi,Sound;

function  Min (a1,a2 : integer) : integer;
begin
  if a1 < a2 then Result:=a1 else Result:=a2;
end; {Min}

function  Max (a1,a2 : integer) : integer;
begin
  if a1 > a2 then Result:=a1 else Result:=a2;
end; {Max}

{ Выдаем строку s и кнопки "Да" и "Нет". Возврат IDYES или IDNO }
function  Message (s : PChar; attr : integer) : integer;
VAR
  mv : boolean;
```

```
begin
  mv:=movis;
  if mv then MouseOff;
  DirectDraw.FlipToGDISurface;
  ShowCursor (TRUE);
  Message:=MessageBox (hWindow,s,AppName,attr);
  ShowCursor (FALSE);
  if mv then MouseOn;
end; {Message}

{ Перевод integer в символы в заданное число позиций (< 33) по
  основанию r. Минус может ставиться при r=10. Если z > 0, то
  пишутся незначащие нули }
function  IntStrFix (a,r,pos,z : integer) : string;
LABEL  10;
 VAR
   i,sign,rest : byte;
   s : string[32];

begin
  sign:=0;
  s[0]:=char(pos);
  for i:=1 to pos do s[i]:='X';
  if a < 0 then
    begin
    sign:=1;
    a:=-a;
    end;
  repeat
    if pos = 0 then goto 10;
    rest:=a mod r;
    if rest > 9 then Inc(rest,byte('A')-10) else
      Inc(rest,byte('0'));
    s[pos]:=char(rest);
    a:=a div r;
    Dec (pos);
```

```
  until a = 0;
  if pos = 0 then goto 10;
  i:=pos;
  while i > 0 do
    begin
    s[i]:=' ';
    if z > 0 then s[i]:='0';
    Dec (i);
    end;
  if z > 0 then pos:=1;
  if (sign = 1) and (r = 10) then s[pos]:='-';
10:
  IntStrFix:=s;
end; {IntStrFix}

{ Рисуем прямоугольник в поверхности по адресу psurf }
procedure  FillRec (x0,y0,dlx,dly,col,pitch : integer;
  psurf : pointer);
begin
    asm
    pushad
    mov     eax,pitch
    mul     y0
    mov     ecx,bytepp
@0: add     eax,x0
    loop    @0
    add     eax,psurf
    mov     edi,eax
    mov     eax,col
    // ebx - счетчик линий
    mov     ebx,dly
    mov     edx,pitch
    mov     ecx,bytepp
    // В edx будет шаг на следующую линию
@1: sub     edx,dlx
    loop    @1
```

```
    mov     esi,bytepp
    // В esi - добавок для перехода на следующую точку
    sub     esi,3
    cld
@2: mov     ecx,dlx
@3: stosb
    ror     eax,8
    stosb
    ror     eax,8
    stosb
    ror     eax,16
    add     edi,esi
    loop    @3
    add     edi,edx
    dec     ebx
    jnz     @2
    popad
    end;
end; {FillRect}

function  Rect (ALeft,ATop,ARight,ABottom : Integer) : Trect;
VAR
  ARect : TRect;

begin
  ARect.left:=Aleft;
  ARect.Top:=ATop;
  ARect.right:=Aright;
  ARect.bottom:=ABottom;
  Result:=ARect;
end; {Rect}

{ Обмен байтов местами }
procedure  SwapB (arg1,arg2 : tpmb);
VAR
  t : byte;
```

```
begin
  t:=arg1^[0];
  arg1^[0]:=arg2^[0];
  arg2^[0]:=t;
end; {SwapB}

{ Обмен слов местами }
procedure  SwapW (arg1,arg2 : tpmw);
VAR
  t : word;

begin
  t:=arg1^[0];
  arg1^[0]:=arg2^[0];
  arg2^[0]:=t;
end; {SwapW}

{ Обмен местами тройки байтов }
procedure  SwapD (arg1,arg2 : tpmw);
VAR
  t1 : word;
  t2 : byte;

begin
  t1:=arg1^[0];
  arg1^[0]:=arg2^[0];
  arg2^[0]:=t1;
  t2:=tpmb(arg1)^[2];
  tpmb(arg1)^[2]:=tpmb(arg2)^[2];
  tpmb(arg2)^[2]:=t2;
end; {SwapD}

procedure  SetLevel (lev : byte; nsurf : integer);
VAR
  mv : boolean;
```

```
begin
  mv:=movis;  MouseOff;
  CopyRec (291+(lev-1)*72,251,362+(lev-1)*72,284,562,36,2,nsurf);
  level:=lev;
  if mv then MouseOn;
end;

{ Выводит сообщение о результате игры. Если correct,
  то устанавливает gamestate=GAMEOVER }
procedure  Msg (arg : word; nsurf : integer);
CONST
  KY : array[0..2] of word = (265,284,246);
VAR
  mv : boolean;

begin
  mv:=movis;
  if nsurf = 1 then MouseOff;
  ismsg:=FALSE;
  if arg < 2 then ismsg:=TRUE;
  CopyRec (0,KY[arg],226,KY[arg]+18,566,14,2,nsurf);
  nummsg:=arg;
  if mv and (nsurf = 1) then MouseOn;
end; {Msg}

procedure CheckIO;
begin
  if IOResult = 0 then Exit;
  if movis then MouseOff;
  ShowCursor (TRUE);
  MessageBox (hWindow,'I/O error',AppName,mb_OK or mb_ICONSTOP);
  Halt;
end; {CheckIO}

{ Читаем из файла soundenable, midienable, lang и level }
procedure  GetCfg;
```

```
VAR
  f : file;
  cfg : array[0..3] of byte;

begin
  soundenable:=FALSE;
  soundprohib:=TRUE;
  midienable:=FALSE;
  midiprohib:=TRUE;
  level:=1;
  AssignFile (f,path+'dominoes.cfg');
  Reset (f,1);
  if (IOResult = 0) and (FileSize (f) = 4) then
    begin
    BlockRead (f,cfg,4);
    CloseFile (f);
    if cfg[0] = 1 then
      begin
      soundenable:=TRUE;
      soundprohib:=FALSE;
      end;
    if cfg[1] = 1 then
      begin
      midienable:=TRUE;
      midiprohib:=FALSE;
      end;
    if cfg[3] = 2 then level:=2;
    end else
      begin
      soundenable:=TRUE;
      soundprohib:=FALSE;
      midienable:=TRUE;
      midiprohib:=FALSE;
      end;
end; {GetCfg}
```

```
{ Сохраняем в файле soundenable, midienable, lang и level }
procedure  SaveCfg;
VAR
  f : file;
  cfg : array[0..3] of byte;

begin
  cfg[0]:=byte(soundenable);
  cfg[1]:=byte(midienable);
  cfg[2]:=0;
  cfg[3]:=level;
  AssignFile (f,path+'dominoes.cfg');
  Rewrite (f,1);
  CheckIO;
  BlockWrite (f,cfg,4);
  CheckIO;
  CloseFile (f);
end; {SaveCfg}

{ Загрузка данного BMP-файла на поверхность DirectDraw }
procedure  LoadBitMap (x1,y1,x2,y2,nsurf : integer;
  BitMapName : string);
VAR
  Bitmap : HBitmap;
  BM : Windows.TBitmap;
  DC,ImageDC : HDC;

begin
  Bitmap:=LoadImage(0,PChar(BitMapName),IMAGE_BITMAP,
                    0,0,LR_LOADFROMFILE);
   if BitMap = 0 then
    begin
    Message ('File not found. Please reinstall '+
     Appname,mb_OK or mb_ICONSTOP);
    StopMidi;
    CloseMidi;
```

```
    DestroySound;
    Directdraw.RestoreDisplayMode;
    DestroyAllSurf;
    KillTimer (hWindow,1);
    Halt;
    end;
  { Получаем размер картинки }
  GetObject (Bitmap,SizeOf(BM),@BM);
  ImageDC:=CreateCompatibleDC (0);
  SelectObject (ImageDC,BitMap);
  surfrec[nsurf].surf.GetDC (DC);
  BitBlt (DC,x2,y2,BM.bmwidth,BM.bmheight,ImageDC,x1,y1,SRCCOPY);
  surfrec[nsurf].surf.ReleaseDC (DC);
  DeleteDC (ImageDC);
  DeleteObject (Bitmap);
end; {LoadBitMap}

{ Ожидание примерно ms миллисекунд и перерисовывание курсора.
  Если dellbd = TRUE, то нажатия на левую кнопку игнорируются.
  Курсор должен быть инициализирован! }
procedure  Wait (ms : TLargeInteger; dellbd : boolean);
VAR
  time : TLargeInteger;
  mox1,moy1 : smallint;

begin
  time:=GetQPCount;
  mox1:=mox;
  moy1:=moy;
  repeat
    Sleep (1);
    if PeekMessage (Amessage,0,wm_MouseMove,wm_MouseMove,PM_REMOVE) or
      dellbd and PeekMessage (Amessage,0,wm_LButtonDown,
        wm_LButtonDown,PM_REMOVE) then
      begin
      mophase:=(mophase+trig) mod mophases;
```

```
      trig:=trig xor 1;
      mox:=LoWord(Amessage.lParam);
      moy:=Hiword(Amessage.lparam);
      if movis and ((mox <> mox1) or (moy <> moy1)) then
        begin
        DirectDraw.WaitForVerticalBlank (DDWAITVB_BLOCKBEGIN,0);
        MouseOff;
        MouseOn;
        mox1:=mox;
        moy1:=moy;
        end;
      end;
  until GetQPCount >= time+qpfreq div 1000*ms;
end; {Wait}

{ Возврат значения счетчика системного таймера (1.19 МГц) }
function  GetQPCount : TLargeInteger;
VAR
  count : TLargeInteger;

begin
  QueryPerformanceCounter (count);
  Result:=count;
end; {GetQPCount}

procedure  LoadFileToPoint (name : string; var p : pointer);
VAR
  f : file;
  size : integer;

begin
 AssignFile (f,path+name);
 Reset (f,1);
 CheckIO;
 size:=FileSize (f);
 GetMem (p,size);
```

```
 BlockRead (f,p^,size);
 CheckIO;
 CloseFile (f);
end; {LoadFileToPoint}

{ Запись в файл name ct байтов с адреса buf^. }
procedure  SaveFile (name : string; var buf; ct : integer);
VAR
  f : file;

begin
  AssignFile (f,path+name);
  Rewrite (f,1);
  CheckIO;
  BlockWrite (f,buf,ct);
  CheckIO;
  CloseFile (f);
end; {SaveFile}

{ Загрузка файла в переменную p, число байтов ct,
  если такой файл есть }
procedure  LoadFile (name : string; var p; ct : integer);
VAR
  f : file;

begin
  AssignFile (f,path+name);
  Reset (f,1);
  if IOResult = 2 then Exit;
  BlockRead (f,p,ct);
  CheckIO;
  CloseFile (f);
end; {LoadFile}

procedure  ShowSound (nsurf : integer);
begin
```

```
  if nsurf = 1 then MouseOff;
  CopyRec (680+10*byte(soundenable),173,689+10*byte(soundenable),
    186,710,560,2,nsurf);
  if nsurf = 1 then MouseOn;
end; {ShowSound}

procedure  ShowMidi (nsurf : integer);
begin
  if nsurf = 1 then MouseOff;
  CopyRec (680+10*byte(midienable),173,689+10*byte(midienable),
    186,778,569,2,nsurf);
  if nsurf = 1 then MouseOn;
end; {ShowMidi}
(*
{ Запись поверхности nsurf в файл BMP }
procedure  DiskSurf (nsurf : integer; name : string);
VAR
  { Этот и следующий заголовок не разрывать! }
  bmhead : record
    btype : word;
    bsize : integer;
    bres1 : word;
    bres2 : word;
    boffs : integer;
    end;
  bminfo : record
    bisize     : integer;
    biwidth    : integer;
    biheight   : integer;
    biplanes   : word;
    bibitcount : word;
    bicompress : integer;
    bisizeim   : integer;
    bixpm      : integer;
```

```
    biypm      : integer;
    biclrused  : integer;
    biclrimp   : integer;
    end;
VAR
  f : file;
  i,j,k,strlength : dword;
  p,p1,p2 : ^byte;

begin
  AssignFile (f,name);
  Rewrite (f,1);
  FillChar (bmhead,sizeof(bmhead),0);
  FillChar (bminfo,sizeof(bminfo),0);
  strlength:=SCRX*3;
  bmhead.btype:=$4D42;
  bmhead.bsize:=strlength*SCRY+54;
  bmhead.boffs:=54;
  bminfo.bisize:=40;
  bminfo.biwidth:=SCRX;
  bminfo.biheight:=SCRY;
  bminfo.biplanes:=1;
  bminfo.bibitcount:=24;
  bminfo.bisizeim:=SCRX*SCRY*3;
  GetMem (p,SCRX*SCRY*3);
  p1:=p;
  { Переписываем r,g,b с поверхности в p^ в обратном порядке строк }
  SurfLock (nsurf);
  for i:=SCRY-1 downto 0 do
    begin
    dword(p2):=dword(surfrec[nsurf].psurf)+dword(surfrec[nsurf].pitch)*i;
    for j:=0 to SCRX-1 do
      begin
      for k:=1 to 3 do
        begin
```

```
        p1^:=p2^;
        Inc (dword(p1));
        Inc (dword(p2));
        end;
      Inc (dword(p2),bytepp-3);
      end;
    end;
  SurfUnLock (nsurf);
  BlockWrite (f,bmhead,sizeof(bmhead));
  BlockWrite (f,bminfo,sizeof(bminfo));
  BlockWrite (f,p^,SCRX*SCRY*3);
  CloseFile (f);
  FreeMem (p);
end; {DiskSurf}
*)
end.
```

Это обычные рутинные процедуры, которые необходимы для поддержки работы программы. Их назначение ясно из комментариев. Процедура `FillRec`, которая рисует закрашенный прямоугольник, использует прямой доступ к видеопамяти. Она может быть заменена аналогичной процедурой, использующей метод поверхности `BltFast`. Процедура `DiskSurf` закомментирована, используйте ее для отладки.

3.7.9. Модуль Game.pas

Это основной модуль для игры (листинг 3.15). Назначение его процедур ясно из комментариев.

В процедуре `Action` остался код от DOS-варианта игры, в котором можно было играть с помощью клавиш. Теперь в этом нет необходимости, мыши есть у всех.

Листинг 3.15. Модуль Game.pas

```
{$A-,B-,I-,R-,S-,X+,H+}
UNIT  Game;
INTERFACE
USES  Glob;
```

```
// Отрисовка всех используемых поверхностей для всех состояний игры
procedure  DrawSurf;
// Инициализация игры
procedure  InitGame;
// Показ заставки игры
procedure  Splash;
// Начало следующей игры
procedure  NewGame (bkgr : integer);
// Обработка сообщений от оконной функции
procedure  Process (gevent : tGEvent; key : char; mx,my : word);
// Работа программы каждые полсекунды для отображения мерцающей каретки
procedure  Go;
// Вызывается при движении мыши над игровым полем при level=1.
procedure  Check1;
// Вызывается при движении мыши над игровым полем при level=2.
procedure  Check2;
// Убирает подсказку у указателя, какая кость будет отмечена для level=1
procedure  RemoveHint1;
// Убирает подсказку у указателя, какая кость будет отмечена для level=2
procedure  RemoveHint2;

CONST
  { Экранные координаты левого верхнего угла игрового поля для level = 1
    и level = 2 }
  PX2 = 160;
  PY2 = 100;
  PX1 = PX2;
  PY1 = PY2+20;
  { Координаты в рабочей поверхности и размеры панелей }
  HELPX = 0;
  HELPY = 187;
  HELPDX = 359;
  HELPDY = 253;
  NAMEX = 0;
  NAMEY = 120;
  NAMEDX = 453;
```

```
  NAMEDY = 67;
  { Ширина и высота квадрата для level = 2 }
  SQX = 50;
  SQY = 50;
VAR
  biosct,ct,ct1 : word;
  pf : pointer;
  nummsg : word = 2;
  ismsg : boolean = FALSE;
  is_music5 : boolean = FALSE;
  startgame : boolean = TRUE;
  startmidi : boolean = FALSE;
  resgame : word = 0;
  oldstate : tGameState;
  hintx,hinty,lightnapr : integer;

IMPLEMENTATION
USES
  Windows,Messages,MMSystem,ShellAPI,DDraw,Ddrawunit,Mouse,Fonts,
   Util,Sound,Midi;
TYPE
  TStr = ^shortstring;

  // Объект для ввода имени в таблицу рекордов
  TEName = object
  public
   // Подготовка ввода имени
   procedure  Init;
   // Обработка введенного символа ch
   procedure  Enter (ch : char);
   // Отображение панели с тем, что введено
   procedure  Display (sound,nsurf : integer);
  private
   // Строка с введенными символами
   sname : shortstring;
  end;
```

```
// Объект для показа каретки при вводе имени
TECaret = object
public
 // Подготовка к показу каретки
 procedure  Init (pos1,wx1,wy1,fx1,fy1,px1,py1,dl1,chrspace1 : integer;
   str1 : TStr);
 // Сохранение фона под кареткой
 procedure  SaveBkgnd (newpos : integer);
 // Прячет каретку
 procedure  Hide;
 // Показывает каретку в позиции newpos
 procedure  Show (newpos : integer);
private
 { Номер символа, за которым курсор
   pos
   Координаты курсора в рабочей поверхности
   wx,wy
   Координаты для сохранения фона под курсором
   fx,fy
   Начальная x- и y-координата курсора на экране
   px,py
   Длина курсора минус 1
   dl
   Пропуск между символами строки
   chrspace
   Триггер для мерцания курсора
   trig }
 pos,wx,wy,fx,fy,px,py,dl,chrspace,trig : integer;
 str : TStr;
end;
procedure  MakeBoard1;  forward;
procedure  MakeBoard2;  forward;
procedure  MarcOn1;     forward;
procedure  MarcOn2;     forward;
procedure  MarcOff1;    forward;
procedure  MarcOff2;    forward;
```

```
  procedure  MouseSquare1; forward;
  procedure  MouseSquare2; forward;

TYPE
  { Связь данного квадратика в масссиве board: не связан, связан с
    левым, правым, верхним, нижним соседом }
  tconnect = (none,left,right,up,down);
  { Направление возможной связи квадратика для level = 1 }
  tdirect = (none1,leftright,updown);
  { Положение кости домино при заполнении игрового поля }
  tposture = (indefinite,horizontal,vertical,any);
CONST
  MAXCOLS = 30; // 7
  // Надо ли обновлять таблицу рекордов
  IS_REC : boolean = FALSE;
  // Ссылки на процедуры создания игровой позиции для обоих уровней
  MAKEBOARD   : array[1..2] of procedure = (MakeBoard1,MakeBoard2);
  // Ссылки на процедуры снятия отметки с кости
  MARCOFF     : array[1..2] of procedure = (MarcOff1,MarcOff2);
  // Ссылки на процедуры отметки кости
  MARCON      : array[1..2] of procedure = (MarcOn1,MarcOn2);
  { Ссылки на процедуры вычисления координат квадрата (половины кости),
    на котором было нажатие кнопки мыши }
  MOUSESQUARE : array[1..2] of procedure = (MouseSquare1,MouseSquare2);
VAR
  // Начальное максимальное время игры для определения рекорда
  maxres : array[1..2] of word = (999,999);
  x1,x2,y1,y2 : integer;
  game1,start : boolean;
  posture : tposture;
  { Значения очков на всех костях домино }
  die : array[1..28,1..2] of byte;
  { Логические координаты квадратов, на которых происходит нажатие и
    отпускание кнопки мыши. Не редактировать эту строку! }
  sqpressx,sqpressy,sqfreex,sqfreey : integer;
```

```
  { Эти же координаты собраны в массив координат квадратов, на
    которых нажимают и отпускают кнопку. Соответствие:
    sqpressfree[0] = sqpressx, sqpressfree[1] = sqpressy,
    sqpressfree[2] = sqfreex, sqpressfree[3] = sqfreey }
  sqpressfree : array[0..3] of integer absolute sqpressx;
  { Если test[i] = x, то кость с порядковым номером i отмечена игроком
    x раз (в разных местах игрового поля) }
  test : array[0..27] of byte;
  { Игровое поле 8x8 }
  board : array[1..8,1..8] of record
          value : byte;
          connect : tconnect;
          direct : tdirect;
          end;
  { Таблица рекордов }
  rec : array [1..2,0..4] of record
           fam : string[25];
           res : word;
           end;
  mrec : array[1..140] of word absolute rec;
  EName : TEname;
  ECaret : TECaret;
  // Образцы используемых цветов, прочитанные с поверхности DirectDraw
  cols : array[0..MAXCOLS] of dword;
  PREVSTATE : Tgamestate = GAMES;

{ Зашифровка/расшифровка таблицы рекордов}
procedure  Crypt;
VAR  i : smallint;
begin
  for i:=1 to 140 do MREC[i]:=MREC[i] xor (i-not i);
end; {Crypt}

// Показываем/убираем (1/0) таблицу рекордов
procedure  ShowRec (arg,sound : byte; nsurf : integer);
```

```
VAR
  i,j,y : integer;
  s: string;

begin
  if arg = 1 then
    begin
    if (GameState <> NAME) and (GameState <> TREC) then
      for i:=1 to 2 do
        if maxres[i] < rec[i,4].res then
          { Запрос имени }
          begin
          EName.Init;
          EName.Display (1,1);
          Exit;
          end;
    if sound = 1 then PlaySound (6);
    if nsurf = 1 then MouseOff;
    LoadBitMap (0,0,165,36,nsurf,path+'record.bmp');
    Sleep(10);
    { Вписываем таблицу }
    SurfLock (nsurf);
    for i:=0 to 1 do
      for j:=0 to 4 do
        begin
        s:='';
        y:=200+i*218+j*25;
        TextXY (200,y,surfrec[nsurf].pitch,0,cols[0],pf,
         surfrec[nsurf].psurf,chr(j+49)+'.');
        if rec[i+1,j].fam[0] > #0 then
          begin
          s:=rec[i+1,j].fam;
          while TextWidth (0,pf,s) > 255 do SetLength(s,Length(s)-1);
          end;
        TextXY (221,y,surfrec[nsurf].pitch,0,cols[i+1],pf,
         surfrec[nsurf].psurf,s);
```

```
        s:=IntStrFix (rec[i+1,j].res,10,3,0);
        TextXY (520-TextWidth(0,pf,s),y,surfrec[nsurf].pitch,0,
         cols[i+3],pf,surfrec[nsurf].psurf,s);
        end;
    SurfUnLock (nsurf);
    if nsurf = 1 then MouseOn;
    GameState:=TREC;
    end
     else
      // Если убираем таблицу рекордов, то выход из программы
      begin
      Wait (300,TRUE);
      PlaySound (13);
      Wait (700,TRUE);
      SaveCfg;
      PostMessage (hWindow,wm_Close,0,0);
      end;
end; {ShowRec}

{ Методы EName для ввода имени }
procedure  TEName.Init;
begin
        // Строка для ввода имени игрока
        sname:='';
        // Инициализируем каретку
        ECaret.Init (0,764,6,765,6,199,544,19,0,@sname);
end; {EName.Init}

// Получен символ ch при вводе имени
procedure  TEName.Enter (ch : char);
VAR
  i,j,k,x,x1 : integer;
begin
       case ch of
        // Нажата клавиша <Backspace>
BACK:   if Length(sname) > 0 then
```

```
          // Убираем последний символ
          begin
          DirectDraw.WaitForVerticalBlank (DDWAITVB_BLOCKEND,0);
          MouseOff;
          Ecaret.Hide;
          x1:=199+TextWidth (0,pf,sname)-1;
          SetLength(sname,Length(sname)-1);
          x:=199+TextWidth (0,pf,sname);
          CopyRec (x+283,343,x1+283,365,x,544,2,1);
          Ecaret.Show (Length(sname));
          MouseOn;
          end;
        // Нажата клавиша <Esc> или <Enter>
ESC,RETURN:
        begin
        DirectDraw.WaitForVerticalBlank (DDWAITVB_BLOCKEND,0);
        MouseOff;
        CopyRec (17,528,426,581,17,528,3,1);
        MouseOn;
        { Записываем фамилию в таблицу }
        for i:=1 to 2 do
          for j:=0 to 4 do
            if maxres[i] < rec[i,j].res then
              begin
              for k:=4 downto j+1 do
                rec[i,k]:=rec[i,k-1];
              rec[i,j].fam:=sname;
              rec[i,j].res:=maxres[i];
              break;
              end;
        { Записываем рекорды на диск }
        Crypt;
        SaveFile ('records',rec,280);
        Crypt;
        ShowRec (1,1,1);
        end else
```

```
          // Нажата другая клавиша
          if (ch in [' '..#255]) and (Length (sname) < 25) and
            (TextWidth (0,pf,sname+ch) < 209) then
            // Выводим заданный символ и добавляем его к sname
            begin
            DirectDraw.WaitForVerticalBlank (DDWAITVB_BLOCKEND,0);
            MouseOff;
            Ecaret.Hide;
            SurfLock (1);
            TextXY (199+TextWidth (0,pf,sname),544,surfrec[1].pitch,
              0,cols[2],pf,surfrec[1].psurf,ch);
            SurfUnLock (1);
            sname:=sname+ch;
            Ecaret.Show (Length(sname));
            MouseOn;
            end;
       end; {case}
end; {EName.Enter}

// Показ панели с именем игрока
procedure  TEName.Display (sound,nsurf : integer);
begin
        if sound = 1 then PlaySound (8);
        if nsurf = 1 then
          begin
          DirectDraw.WaitForVerticalBlank (DDWAITVB_BLOCKEND,0);
          MouseOff;
          end;
        CopyRec (300,327,709,380,17,528,2,nsurf);
        SurfLock (nsurf);
        TextXY (199,544,surfrec[nsurf].pitch,0,cols[2],pf,
         surfrec[nsurf].psurf,sname);
        SurfUnLock (nsurf);
        if nsurf = 1 then ECaret.Show (ECaret.pos);
        if nsurf = 1 then MouseOn;
```

```
        GameState:=NAME;
end; {EName.Display}

{ Рисуем фон в поверхности surf }
procedure  DrawBackgr (nsurf : integer);
begin
  LoadBitMap (0,0,0,0,nsurf,path+'bkgr.bmp');
end; {DrawBackgr}

{ Показ хода времени }
procedure  ShowTime (nsurf : integer);
CONST
  XZ = 690;
  YZ = 323;
VAR
  i,dig : integer;
  mv : boolean;
  digit : array[0..2] of byte;
  s : string[3];

begin
  Str (ct:3,s);
  for i:=0 to 2 do
    if s[i+1] = ' ' then digit[i]:=10 else
      digit[i]:=byte(s[i+1])-48;
  mv:=movis;
  if mv then
    begin
    DirectDraw.WaitForVerticalBlank (DDWAITVB_BLOCKEND,0);
    MouseOff;
    end;
  for i:=2 downto 0 do
      begin
      dig:=digit[i];
      CopyRec (288+dig*29,215,288+dig*29+27,215+34,
       690+i*35,323+i*3,2,nsurf);
      end;
```

```
  if mv then MouseOn;
end; {ShowTime}

{ Обработка сообщений от таймера }
procedure  STime;
begin
  if ct < 999 then
    begin
    if biosct-ct1 > 9 then
      begin
      ct1:=biosct;
      Inc (ct);
      ShowTime (1);
      { Игрок уже не попадает в таблицу рекордов? }
      if not is_music5 and (ct >= rec[level,4].res) then
        begin
        PlaySound (10);
        is_music5:=TRUE;
        end;
      end;
    end;
end; {STime}

{ Показываем/убираем (1/0) правила игры }
procedure  ShowHelp (arg,sound : byte; nsurf : integer);
VAR
  s : string[80];
  i,j : integer;
  f1 : text;

begin
  if arg = 0 then
    begin
    gamestate:=PREVSTATE;
    DrawSurf;
    Exit;
    end;
```

```
// Запоминаем состояние игры до показа правил игры
if GameState <> HELP then prevstate:=GameState;
GameState:=HELP;
// Если это не восстановление экрана игры, то проигрываем звук
if nsurf = 1 then
  begin
  PlaySound (3);
  MouseOff;
  end;
for i:=0 to 5 do
  for j:=0 to 9 do
    CopyRec (227,251,227+63,251+63,80+j*64,100+i*64,2,nsurf);
for j:=0 to 9 do
  CopyRec (227,251,227+63,251+35,80+j*64,484,2,nsurf);
AssignFile (f1,path+'help.txt');
Reset (f1);
CheckIO;
i:=23;
SurfLock (nsurf);
repeat
  Readln (f1,s);
  CheckIO;
  TextXY (110,90+i,surfrec[nsurf].pitch,1,cols[4],pf,
    surfrec[nsurf].psurf,s);
  Inc (i,23);
until Eof (f1);
SurfUnLock (nsurf);
CloseFile (f1);
{ Кость с курсором }
CopyRec (607,214,706,263,269,265,2,nsurf);
CopyRec (150,100,249,149,540,265,2,nsurf);
CopySpr (700,62,799,111,540,265,2,nsurf);
CopySpr (700,6,704,41,587,272,2,nsurf);
{ Верно и неверно }
CopyRec (100,100,199,149,269,423,2,nsurf);
CopySpr (700,62,799,111,269,423,2,nsurf);
```

```
  CopySpr (700,6,704,41,316,430,2,nsurf);
  CopyRec (450,100,549,149,540,423,2,nsurf);
  CopySpr (700,62,799,111,540,423,2,nsurf);
  CopySpr (700,6,704,41,587,430,2,nsurf);
  { Стрелка }
  CopySpr (640,150,676,163,437,284,2,nsurf);
  if nsurf = 1 then MouseOn;
end; {ShowHelp}

// Вывод разделителя кости в level 1
procedure  Splitter1 (x,y : word; direct : tdirect; nsurf : integer);
begin
  if direct = updown then
    { Разделение по вертикали }
    begin
    CopySpr (705,0,746,4,PX1+(x-1)*SQX+1,PY1+(y-1)*SQY+48,2,nsurf);
    CopySpr (700,112,749,211,PX1+(x-1)*SQX,PY1+(y-1)*SQY,2,nsurf);
    end else
      { Разделение по горизонтали }
      begin
      CopySpr (700,0,704,41,PX1+(x-1)*SQX+47,PY1+(y-1)*SQY+1,2,nsurf);
      CopySpr (700,62,799,111,PX1+(x-1)*SQX,PY1+(y-1)*SQY,2,nsurf);
      end;
end; {Splitter1}

// Вывод разделителя кости в level 2
procedure  Splitter2 (x,y : word; direct : tdirect; nsurf : integer);
begin
  if direct = updown then
    { Разделение по вертикали }
    begin
    CopySpr (700,0,704,41,PX2+(x-1)*SQX+47,PY2+(y-1)*SQY+1,2,nsurf);
    CopySpr (700,62,799,111,PX2+(x-1)*SQX,PY2+(y-1)*SQY,2,nsurf);
    end else
```

```
      { Разделение по горизонтали }
      begin
      CopySpr (710,0,746,5,PX2+(x-1)*SQX+7,PY2+(y-1)*SQY+47,2,nsurf);
      CopySpr (700,112,749,211,PX2+(x-1)*SQX,PY2+(y-1)*SQY,2,nsurf);
      end;
end; {Splitter2}

{ Рисование игрового поля после <Alt>+<Tab> }
procedure  DrawBoard (level : word; nsurf : integer);
VAR
  i,j : word;
  direct : tdirect;
  connect : tconnect;
  x1,y1,x2,y2 : integer;

begin
  if level = 1 then
    begin
    for i:=0 to 6 do
      for j:=0 to 7 do
        begin
        direct:=board[i+1,j+1].direct;
        if direct = none1 then direct:=leftright;
        connect:=board[i+1,j+1].connect;
        x1:=board[i+1,j+1].value*SQX;
        if connect in [up,down] then Inc (x1,350);
        x2:=x1+SQX-1;
        y1:=50*word(direct = updown);
        if connect <> none then y1:=100;
        y2:=y1+SQY-1;
        CopyRec (x1,y1,x2,y2,PX1+j*SQX,PY1+i*SQY,2,nsurf);
        if connect = up then Splitter1 (j+1,i,updown,nsurf);
        if connect = left then Splitter1 (j,i+1,leftright,nsurf);
        end;
    end else
```

```
      begin
      for i:=0 to 7 do
        for j:=0 to 7 do
          begin
          x1:=350+board[i+1,j+1].value*SQX;
          x2:=x1+SQX-1;
          y1:=SQY*word(board[i+1,j+1].connect <> none);
          y2:=y1+SQY-1;
          if board[i+1,j+1].value < 10 then
            CopyRec (x1,y1,x2,y2,PX2+j*SQX,PY2+i*SQY,2,nsurf);
          connect:=board[i+1,j+1].connect;
          if connect = left then Splitter2 (j,i+1,updown,nsurf);
          if connect = up then Splitter2 (j+1,i,leftright,nsurf);
          end;
      end;
  hintx:=-1;
end; {DrawBoard}

// Восстановление всех поверхностей DirectDraw
procedure  DrawSurf;
begin
  MouseOff;
  LoadBitMap (0,0,0,0,2,path+'dice.bmp');
  SetCursorPos (mox,moy);
  if GameState in [GAMES,GAMEOVER,HELP,NAME,TREC] then
    begin
    { Рисуем фон }
    DrawBackgr (3);
    ShowTime (3);
    ShowSound (3);
    ShowMidi (3);
    SetLevel (level,3);
    Msg (nummsg,3);
    DrawBoard (level,3);
    if GameState = NAME then EName.Display (0,3);
    if GameState = TREC then ShowRec (1,0,3);
```

```
    if GameState = HELP then ShowHelp (1,0,3);
    end;
  DirectDraw.WaitForVerticalBlank (DDWAITVB_BLOCKEND,0);
  CopyRec (0,0,SCRX-1,SCRY-1,0,0,3,1);
  if GameState = NAME then ECaret.Show (ECaret.pos);
  moprohib:=FALSE;
  if gamestate in [games,gameover,help] then
  movis:=FALSE;
  MouseOn;
  DrawBackgr (3);
end; {DrawSurf}

// Показывает заставку игры
procedure  Splash;
LABEL  100;
VAR
  f : file;
  bb : array[0..1600] of byte;
  o : byte;
  i,x,y,sx,sy : smallint;
  x1,y1,x2,y2 : integer;
  p : tpmb;
  r,g,b : dword;

begin
  { Инициализируем указатель мыши }
  // Ширина указателя
  modx:=32;
  // Высота указателя
  mody:=32;
  // Координаты горячей точки
  mohotx:=0;
  mohoty:=0;
  // Фаза указателя
  mophase:=0;
```

```
{ Число изображений указателя в одной строке в файле
  dice.bmp }
moperline:=20;
{ Номер скан-линии, на которой начинаются изображения указателей
  в файле dice.bmp }
moline:=150;
{ X-координата прямоугольника для сохранения фона под указетелем
  в файле dice.bmp }
mobkgx:=32*8;
{ Y-координата прямоугольника для сохранения фона под указетелем
  в файле dice.bmp }
mobkgy:=moline+mody*2;
{ Если TRUE, то показ указателя запрещен }
moprohib:=FALSE;
{ Число фаз указателя }
mophases:=48;
{ Номер экранной поверхности DirectDraw }
psurf:=1;
{ Номер поверхности, на которой находятся изображения указателей }
mosurf:=2;
{ Загружаем в surfrec[2].surf шарики }
LoadBitMap (0,0,0,0,2,path+'balls.bmp');
{ Читаем координаты шариков }
AssignFile (f,path+'ballpos');
Reset (f,1);
CheckIO;
BlockRead (f,bb,FileSize (f));
CheckIO;
CloseFile (f);
{ Рисуем фон }
DrawBackgr (3);
CopyRec (0,0,SCRX-1,SCRY-1,0,0,3,1);
i:=0;
x:=135;
y:=160;
Wait (300,TRUE);
```

```
PlaySound (5);
Wait (1500,TRUE);
{ Очищаем surfrec[2].surf, начиная с 96-ой линии }
SurfLock (2);
FillRec (0,96,SCRX,SCRY-96,0,surfrec[2].pitch,surfrec[2].psurf);
SurfUnLock (2);
// Рисуем шариками слово "Domino"
repeat
  x1:=i mod 96 mod 45*14;
  y1:=i mod 96 div 45*14;
  x2:=x1+14-1;
  y2:=y1+14-1;
  CopySpr (x1,y1,x2,y2,x,y,2,2);
  o:=bb[i];
  asm
  mov  al,o;
  shl  al,4;
  sar  al,4;
  cbw;
  mov  sx,ax
  mov  al,o;
  sar  al,4;
  cbw
  mov  sy,ax
  end;
  Inc (x,sx);
  Inc (y,sy);
  Inc (i);
until (i=1289);
// Ждем 0.2 секунды с игнорированием нажатий на левую кнопку мыши
Wait (200,TRUE);
CopySpr (x1,y1,x2,y2,318,205,2,2);
CopySpr (72,118,552,297,150,150,2,1);
LoadBitMap (0,0,0,0,2,path+'dice.bmp');
// Копируем слово "Dilemma"
CopyRec (300,381,607,448,187,323,2,1);
```

```
  { Запоминаем цвета для TextXY }
  SurfLock (2);
  dword(p):=dword(surfrec[2].psurf)+770*bytepp;
  for i:=0 to MAXCOLS do
    begin
    b:=p^[0];
    g:=p^[1];
    r:=p^[2];
    cols[i]:=b+g shl 8+r shl 16;
    Inc (dword(p),bytepp);
    end;
  SurfUnLock (2);
  GameState:=GAMES;
100:
  // Ждем 4 секунды с игнорированием нажатий на левую кнопку мыши
  Wait (4000,TRUE);
end; {Splash}

{ Обработка нажатия и отпускания клавиши и кнопки мыши в игре }
procedure  Action;
LABEL  10,20,30,100;
CONST
  DX : array[1..2] of byte = (130,135);
  { Корректировка для вычисления порядкового номера (0, 1, ..., 27)
    кости в массиве test. Пусть min - меньшее значение очков на
    кости, а max - большее. Тогда номер кости
    в test = min*6+max-CORRECT[min] }
  CORRECT : array[0..6] of byte = (0,0,1,3,6,10,15);
  { Для установки связи в обратную сторону (от другого
    квадратика к данному) }
  LRUD : array[left..down] of tconnect = (right,left,down,up);
VAR
  direct1,direct2 : tdirect;
  i,j,time_result : integer;
  max,min,coincidence : byte;
  connection : tconnect;
```

```
begin
  { Обрабатываем кость. Если игрок отметил несмежные квадраты,
    то переход на 10 }
  if (sqpressx < 0) or (sqfreex < 0) or (Abs (sqpressx-sqfreex)+
    Abs (sqpressy-sqfreey) <> 1) then Exit;
  { Если игра без мыши, то надо проверить на выход
    за поле для level = 2 }
  if (board[sqpressy,sqpressx].value = 10) or
    (board[sqfreey,sqfreex].value = 10) then Exit;
  { Проверим соответствие направлений точек на квадратиках
    (для level = 1) }
  direct1:=board[sqpressy,sqpressx].direct;
  direct2:=board[sqfreey,sqfreex].direct;
10:
  if (sqpressx <> sqfreex) and
    ((direct1 = updown) or (direct2 = updown)) then Exit;
  if (sqpressy <> sqfreey) and
    ((direct1 = leftright) or (direct2 = leftright)) then Exit;
  { Совпадает ли отмеченная кость с уже отмеченной костью? }
  coincidence:=0;
  case board[sqpressy,sqpressx].connect of
left:
    if sqfreex+1 = sqpressx then coincidence:=1;
right:
    if sqpressx+1 = sqfreex then coincidence:=1;
up:
    if sqfreey+1 = sqpressy then coincidence:=1;
down:
    if sqpressy+1 = sqfreey then coincidence:=1;
  end; {case}
  DirectDraw.WaitForVerticalBlank (DDWAITVB_BLOCKEND,0);
  MouseOff;
  { Корректировка связей в board. Цикл по 2-м отмеченным
    квадратикам }
  for i:=0 to 1 do
    begin
```

```
    { Вычисляем координаты x1, y1 отмеченного квадратика
      и x2, y2 связанного с ним квадратика (если такой есть) }
    x1:=sqpressfree[i*2];
    y1:=sqpressfree[i*2+1];
    x2:=x1;
    y2:=y1;
    connection:=board[y1,x1].connect;
    case connection of
left:  Dec (x2);
right: Inc (x2);
up:    Dec (y2);
down:  Inc (y2);
    end; {case}
    { Удаляем эту кость, если есть связь, из массивов board
      и test }
    if connection > none then
      begin
      min:=board[y1,x1].value;
      max:=board[y2,x2].value;
      if min > max then SwapB (@min,@max);
      board[y1,x1].connect:=none;
      board[y2,x2].connect:=none;
      Dec (test[min*6+max-CORRECT[min]]);
      MARCOFF[level];
      end;
    end;
  { Если нет совпадения, то отмечаем кость и заносим ее номер
    в test }
  if coincidence = 0 then
    begin
    min:=board[sqpressy,sqpressx].value;
    max:=board[sqfreey,sqfreex].value;
    if min > max then SwapB (@min,@max);
    Inc (test[min*6+max-CORRECT[min]]);
    MARCON[level];
    MouseOn;
```

```
    { Отмечаем новую связь в board }
    if sqpressx-sqfreex = 1 then connection:=left;
    if sqfreex-sqpressx = 1 then connection:=right;
    if sqpressy-sqfreey = 1 then connection:=up;
    if sqfreey-sqpressy = 1 then connection:=down;
    board[sqpressy,sqpressx].connect:=connection;
    board[sqfreey,sqfreex].connect:=LRUD[connection];
    { Все кости правильно отмечены? }
    for i:=0 to 27 do
      if test[i] <> 1 then goto 20;
    { Да, конец партии }
    { Останавливаем часы }
    time_result:=ct;
    Msg (0,1);
    GameState:=GAMEOVER;
    Wait (200,TRUE);
    PlaySound (9);
    Wait (300,TRUE);
    if (time_result < rec[level,0].res) and
      (time_result < maxres[level]) then
      { Новый чемпион }
      begin
      Wait (200,TRUE);
      PlaySound (2);
      { В maxres[level] получаем минимальное время за все партии }
      if time_result < maxres[level] then
        maxres[level]:=time_result;
      end else
        Wait (200,TRUE);
    { В maxres[level] получаем минимальное время за все партии }
    if time_result < maxres[level] then
      maxres[level]:=time_result;
    Exit;
    end;
  { Если все кости отмечены, то это неправильно }
```

```
20:
  MouseOn;
  for i:=1 to 6+level do
    for j:=1 to 8 do
      if (board[i,j].value < 7) and
        (board[i,j].connect = none) then goto 30;
  Msg (1,1);
  PlaySound (4);
  Wait (500,TRUE);
30:
  Msg (2,1);
  Exit;
100:
  MouseOff;
end; {Action}

{ Если bkgr > 0, то восстанавливать фон под костями }
procedure  NewGame (bkgr : integer);
LABEL  1;
CONST
  KX : array[0..2] of integer = (74,189,230);
VAR
  i,j,k : integer;

begin
  sqpressx:=-1;
  Msg (2,1);
  game1:=TRUE;
  gamestate:=GAMES;
  startgame:=FALSE;
  { Цикл новой игры. Убираем сообщение о результате. Рисуем поле }
1:
  Msg (2,1);
  { Заполняем кости домино значениями }
  k:=1;
  for i:=0 to 6 do
```

```
    for j:=i to 6 do
      begin
      die[k,1]:=i;
      die[k,2]:=j;
      if Random (2) = 0 then SwapB (@die[k,1],@die[k,2]);
      Inc (k);
      end;
  { Перемешиваем их }
  for i:=1 to 28 do
    SwapW (@die[i],@die[Random (28)+1]);
  start:=TRUE;
  if bkgr > 0 then
    begin
    DirectDraw.WaitForVerticalBlank (DDWAITVB_BLOCKEND,0);
    MouseOff;
    CopyRec (PX2,PY2,PX2+399,PY2+399,PX2,PY2,3,1);
    end;
  MAKEBOARD[level];
  game1:=FALSE;
  { Очищаем массив для отслеживания конца игры.
    test[0]=1, если 0-0 отмечена, и т. д. по порядку костей:
    0-1, 0-2, ..., 0-6, 1-1, 1-2, ..., 1-6, ..., 6-6 }
  for i:=0 to 27 do test[i]:=0;
  { ct1 наращивается на 1 по каждому прерыванию от таймера.
    ct - значение на часах (это количество очков для данной партии) }
  PlaySound (1);
  Wait (300,TRUE);
  ct1:=biosct;
  ct:=0;
  is_music5:=FALSE;
  hintx:=-1;
  SetCursorPos (mox,moy);
  ShowTime (1);
  if not startmidi then
    begin
    PlayMidi;
```

```
    startmidi:=TRUE;
    end;
end; {NewGame}

// Проверка выбора пунктов меню
procedure  CheckMenu;
begin
  if MouseInBox (564,36,30,30) and (level = 2) or
    MouseInBox (599,36,30,30) and (level = 1) then
      begin
      SetLevel (3-level,1);
      NewGame (1);
      Exit;
      end;
  if MouseInBox (651,50,41,41) then
    begin
    oldstate:=gamestate;
    ShowHelp (1,1,1);
    Exit;
    end;
  if MouseInBox (642,112,149,29) then
    begin
    NewGame (0);
    Exit;
    end;
  if MouseInBox (713,59,67,26) then
    begin
    ShowRec (1,1,1);
    Exit;
    end;
  { Sound }
  if MouseInBox (650,552,75,30) then
    begin
    if soundprohib then InitSound else
      soundenable:=not soundenable;
    ShowSound (1);
    end;
```

```
  if MouseInBox (728,559,65,30) then
    // MIDI
    begin
    if midiprohib then
      begin
      if InitMidi then midirequest:=TRUE;
      end else
        begin
        midirequest:=TRUE;
        midienable:=not midienable;
        end;
    if midienable and midirequest then
      begin
      PlayMidi;
      midirequest:=FALSE;
      end;
    if not midienable then
      begin
      midienable:=TRUE;
      StopMidi;
      CloseMidi;
      midienable:=FALSE;
      midirequest:=TRUE;
      end;
    ShowMidi (1);
    end;
  { Pause }
  if MouseInBox (638,455,35,86) then
    ShowWindow (hWindow,SW_MINIMIZE);
end; {CheckMenu}

// Обработка вызовов из оконной функции
procedure  Process (gevent : tGEvent; key : char; mx,my : word);
begin
     case gevent of
```

```
       // Нажата клавиша
keyd:  case key of
  ESC:   case GameState of
    ISSPLASH:       PostMessage (hWindow,wm_Close,0,0);
    GAMES,GAMEOVER: ShowRec (1,1,1);
    HELP:           ShowHelp (0,0,1);
    TREC:           ShowRec (0,0,1);
    NAME:           EName.Enter (key);
         end; {case}
        else
          begin
          if GameState = TREC then ShowRec (0,0,1);
          if GameState = NAME then EName.Enter (key);
          if GameState = HELP then
            begin
            ShowHelp (0,0,1);
            Exit;
            end;
          end;
       end; {case}
          // Нажата левая кнопка мыши
lbdown:   begin
          if GameState = HELP then
            begin
            ShowHelp (0,0,1);
            Exit;
            end;
          if GameState = TREC then
            begin ShowRec (0,0,1);
            Exit;
            end;
          if (GameState = GAMES) or (GameState = GAMEOVER) then
           CheckMenu;
          { Pause, sound, midi }
          if (GameState = NAME) and (Mouse.moy > 450) then CheckMenu;
          if GameState = GAMES then
```

```
            begin
            MouseSquare[level];
            if (sqpressx > 0) and (sqpressy > 0) then Action;
            end;
          end;
tic:      if GameState = GAMES then STime;
     end; {case}
end; {Process}

// Инициализация игры
procedure  InitGame;
VAR
  f : file;
  point : TPoint;
  i,j : integer;

begin
  Randomize;
  FindMidi;
  { Загружаем шрифты }
  LoadFileToPoint ('times16',pf);
  { Читаем картинки и курсоры }
  { Рисуем фон, корректируем lang, level, sound и midi }
  CopyRec (0,0,SCRX-1,SCRY-1,0,0,3,1);
  CopyRec (291+(level-1)*72,251,362+(level-1)*72,284,562,36,2,1);
  CopyRec (706,6,706+57,6+18,717,35,2,1);
  CopyRec (680+10*byte(soundenable),173,689+10*byte(soundenable),186,
    710,560,2,1);
  CopyRec (680+10*byte(midienable),173,689+10*byte(midienable),186,
    778,569,2,1);
  { Читаем и расшифровываем таблицу рекордов }
  AssignFile (f,path+'records');
  Reset (f,1);
  if IOResult = 0 then
    begin
    BlockRead (f,rec,280);
```

```
    Crypt;
    CloseFile (f);
    end else
      begin
      rec[1,0].fam:='Author';
      rec[1,0].res:=30;
      rec[2,0].fam:=rec[1,0].fam;
      rec[2,0].res:=73;
      for i:=1 to 2 do
        for j:=1 to 4 do
          begin
          rec[i,j].fam[0]:=#0;
          rec[i,j].res:=999;
          end;
      end;
  { Кот мяукает }
  CopyRec (741,45,781,60,661,421,2,1);
  PlaySound (12);  Wait (1200,TRUE);
  CopyRec (700,45,740,60,661,421,2,1);
  Wait (500,TRUE);
  { Ставим курсор }
  GetCursorPos (point);
  mox:=point.x;
  moy:=point.y;
  mostart:=TRUE;
  MouseOn;
  SetTimer (hWindow,1,55,NIL);
  NewGame (0);
end; {InitGame}

{ Снятие отметки с кости с логическими координатами x1, y1, x2, y2.
  Для level = 1. }
procedure  MarcOff1;
VAR
  xx1,xx2,yy1,yy2 : word;
  direct : tdirect;
  x,y : integer;
```

```
begin
  xx1:=x1;  xx2:=x2;
  if xx1 > xx2 then SwapW (@xx1,@xx2);
  yy1:=y1;
  yy2:=y2;
  if yy1 > yy2 then SwapW (@yy1,@yy2);
  direct:=board[yy1,xx1].direct;
  if direct = none1 then direct:=leftright;
  x:=board[yy1,xx1].value*SQX;  y:=SQX*(word(direct)-1);
  CopyRec (x,y,x+SQX-1,y+SQY-1,PX1+(xx1-1)*SQX,PY1+(yy1-1)*SQY,2,1);
  direct:=board[yy2,xx2].direct;
  if direct = none1 then direct:=leftright;
  x:=board[yy2,xx2].value*SQX;  y:=SQX*(word(direct)-1);
  CopyRec (x,y,x+SQX-1,y+SQY-1,PX1+(xx2-1)*SQX,PY1+(yy2-1)*SQY,2,1);
end; {MarcOff1}

{ Снятие отметки с кости с логическими координатами x1, y1, x2, y2.
  Для level = 2. }
procedure  MarcOff2;
VAR
  x : integer;

begin
  x:=350+board[y1,x1].value*SQX;
  CopyRec (x,0,x+SQX-1,SQY-1,PX2+(x1-1)*SQX,PY2+(y1-1)*SQY,2,1);
  x:=350+board[y2,x2].value*SQX;
  CopyRec (x,0,x+SQX-1,SQY-1,PX2+(x2-1)*SQX,PY2+(y2-1)*SQY,2,1);
end; {MarcOff2}

{ Установка отметки кости с логическими координатами
  sqpressx, sqpressy, sqfreex, sqfreey. Для level = 1. }
procedure  MarcOn1;
VAR
  x1,y1,x2,y2 : word;
  direct : tdirect;
  x : integer;
```

```
begin
  direct:=leftright;
  x1:=sqpressx;  x2:=sqfreex;
  if x1 > x2 then SwapW (@x1,@x2);
  y1:=sqpressy;  y2:=sqfreey;
  if y1 > y2 then SwapW (@y1,@y2);
  if y1 < y2 then direct:=updown;
  x:=board[y1,x1].value*SQX+350*(word(direct)-1);
  CopySpr (x,100,x+SQX-1,100+SQY-1,PX1+(x1-1)*SQX,PY1+(y1-1)*SQY,2,1);
  x:=board[y2,x2].value*SQX+350*(word(direct)-1);
  CopySpr (x,100,x+SQX-1,100+SQY-1,PX1+(x2-1)*SQX,PY1+(y2-1)*SQY,2,1);
  { Рисуем перекладину }
  Splitter1 (x1,y1,direct,1);
  PlaySound (14);
  hintx:=-1;
end; {MarcOn1}

{ Установка отметки кости с логическими координатами
  sqpressx, sqpressy, sqfreex, sqfreey. Для level = 2. }
procedure  MarcOn2;
VAR
  i,j,x : integer;

begin
  x:=350+board[sqpressy,sqpressx].value*SQX;
  CopySpr (x,SQY,x+SQX-1,SQY*2-1,PX2+(sqpressx-1)*SQX,
    PY2+(sqpressy-1)*SQY,2,1);
  x:=350+board[sqfreey,sqfreex].value*SQX;
  CopySpr (x,SQY,x+SQX-1,SQY*2-1,PX2+(sqfreex-1)*SQX,
    PY2+(sqfreey-1)*SQY,2,1);
  { Выводим перекладину }
  j:=sqpressx;  if j > sqfreex then j:=sqfreex;
  i:=sqpressy;  if i > sqfreey then i:=sqfreey;
  if sqpressy = sqfreey then
    { Вертикально }
    Splitter2 (j,i,updown,1) else
```

```
      { Горизонтально }
      Splitter2 (j,i,1eftright,1);
  PlaySound (14);
  hintx:=-1;
end; {MarcOn2}


{ Вычисления координат квадрата (1-8, 1-7) при нажатии кнопки. Если
  курсор за пределами поля, то соответствующая x-координата = -1.
  Для level = 1. }
procedure  MouseSquare1;
VAR
  i,j : word;

begin
  sqpressfree[0]:=-1;
  sqpressfree[2]:=-1;
  { Для горизонтальных костей }
  for j:=0 to 6 do
    for i:=0 to 6 do
      if MouseInBox (PX1+42+j*50,PY1+8+i*50,16,34) then
        begin
        sqpressfree[0]:=j+1;
        sqpressfree[1]:=i+1;
        sqpressfree[2]:=j+2;
        sqpressfree[3]:=i+1;
        Exit;
        end;
  { Для вертикальных костей }
  for j:=0 to 7 do
    for i:=0 to 5 do
      if MouseInBox (PX1+8+j*50,PY1+42+i*50,34,16) then
        begin
        sqpressfree[0]:=j+1;
        sqpressfree[1]:=i+1;
        sqpressfree[2]:=j+1;
```

```
        sqpressfree[3]:=i+2;
        Exit;
        end;
end; {MouseSquare1}

{ Вычисления координат квадрата (1-8, 1-8) при отпускании (arg = 0)/
  нажатии (arg = 1) кнопки. Если курсор за пределами игрового поля,
  то соответствующая x-координата = -1. Для level = 2. }
procedure  MouseSquare2;
VAR
  i,j : word;

begin
  sqpressfree[0]:=-1;
  sqpressfree[2]:=-1;
  { Для горизонтальных костей }
  for j:=0 to 6 do
    for i:=0 to 7 do
      begin
      if (i in [0,7]) and (j in [2..4]) then continue;
      if (i in [3,4]) and (j in [0,6]) then continue;
      if MouseInBox (PX2+42+j*50,PY2+8+i*50,16,34) then
        begin
        sqpressfree[0]:=j+1;
        sqpressfree[1]:=i+1;
        sqpressfree[2]:=j+2;
        sqpressfree[3]:=i+1;
        Exit;
        end;
      end;
  { Для вертикальных костей }
  for j:=0 to 7 do
    for i:=0 to 6 do
      begin
      if (i in [0,6]) and (j in [3,4]) then continue;
      if (i in [2..4]) and (j in [0,7]) then continue;
```

```
      if MouseInBox (PX2+8+j*50,PY2+42+i*50,34,16) then
        begin
        sqpressfree[0]:=j+1;
        sqpressfree[1]:=i+1;
        sqpressfree[2]:=j+1;
        sqpressfree[3]:=i+2;
        Exit;
        end;
      end;
end; {MouseSquare2}

// Вызывается при движении мыши над игровым полем при level=1.
procedure  Check1;
VAR
  i,j : integer;

begin
{ Для горизонтальных костей }
for j:=0 to 6 do
  for i:=0 to 6 do
    if MouseInBox (PX1+42+j*50,PY1+8+i*50,16,34) then
      begin
      { Эта пара уже подсвечена? }
      if (hintx = j) and (hinty = i) and (lightnapr = 0) then Exit;
      { Другая пара подсвечена? }
      if hintx > -1 then RemoveHint1;
      { Ни один квадратик этой пары не отмечен,
        и эту пару можно отметить? }
      if (board[i+1,j+1].connect = none) and
        (board[i+1,j+2].connect = none) and
        (byte(board[i+1,j+1].direct) < 2) and
        (byte(board[i+1,j+2].direct) < 2) then
        { Подсвечиваем эту пару }
        begin
        CopySpr (700,62,799,111,PX1+j*SQX,PY1+i*SQY,2,1);
        hintx:=j;
```

```
        hinty:=i;
        lightnapr:=0;
        end;
      Exit;
      end;
{ Для вертикальных костей }
for j:=0 to 7 do
  for i:=0 to 5 do
    if MouseInBox (PX1+8+j*50,PY1+42+i*50,34,16) then
      begin
      { Эта пара уже подсвечена? }
      if (hintx = j) and (hinty = i) and (lightnapr = 1) then Exit;
      { Другая пара подсвечена? }
      if hintx > -1 then RemoveHint1;
      { Ни один квадратик этой пары не отмечен,
        и эту пару можно отметить? }
      if (board[i+1,j+1].connect = none) and
        (board[i+2,j+1].connect = none) and
        (board[i+1,j+1].direct in [none1,updown]) and
        (board[i+2,j+1].direct in [none1,updown]) then
        { Подсвечиваем эту пару }
        begin
        CopySpr (700,112,749,211,PX1+j*SQX,PY1+i*SQY,2,1);
        hintx:=j;
        hinty:=i;
        lightnapr:=1;
        end;
      Exit;
      end;
{ Курсор не находится над активной зоной костей }
if hintx > -1 then RemoveHint1;
end; {Check1}

// Вызывается при движении мыши над игровым полем при level=2.
procedure  Check2;
```

```
VAR
  i,j : integer;

begin
{ Для горизонтальных костей }
for j:=0 to 6 do
  for i:=0 to 7 do
    begin
    if (j in [2..4]) and (i in [0,7]) or (j in [0,6])
      and (i in [3,4]) then continue;
    if MouseInBox (PX2+42+j*50,PY2+8+i*50,16,34) then
      begin
      { Эта пара уже подсвечена? }
      if (hintx = j) and (hinty = i) and (lightnapr = 0) then Exit;
      { Другая пара подсвечена? }
      if hintx > -1 then RemoveHint2;
      { Ни один квадратик этой пары не отмечен? }
      if (board[i+1,j+1].connect = none) and
        (board[i+1,j+2].connect = none) then
        { Подсвечиваем эту пару }
        begin
        CopySpr (700,62,799,111,PX2+j*SQX,PY2+i*SQY,2,1);
        hintx:=j;  hinty:=i;  lightnapr:=0;
        end;
      Exit;
      end;
    end;
{ Для вертикальных костей }
for j:=0 to 7 do
  for i:=0 to 6 do
    begin
    if (j in [3,4]) and (i in [0,6]) or (j in [0,7]) and
      (i in [2..4]) then continue;
    if MouseInBox (PX2+8+j*50,PY2+42+i*50,34,16) then
      begin
      { Эта пара уже подсвечена? }
```

```
      if (hintx = j) and (hinty = i) and (lightnapr = 1) then Exit;
      { Другая пара подсвечена? }
      if hintx > -1 then RemoveHint2;
      { Ни один квадратик этой пары не отмечен? }
      if (board[i+1,j+1].connect = none) and
        (board[i+2,j+1].connect = none) then
        { Подсвечиваем эту пару }
        begin
        CopySpr (700,112,749,211,PX2+j*SQX,PY2+i*SQY,2,1);
        hintx:=j;  hinty:=i;  lightnapr:=1;
        end;
      Exit;
      end;
    end;
{ Курсор не находится над активной зоной костей }
if hintx > -1 then RemoveHint2;
end; {Check2}

{ Снятие подсказки при level=1. Курсор не трогать! }
procedure  RemoveHint1;
VAR
  x,y,addx,addy : integer;

begin
  x:=board[hinty+1,hintx+1].value*SQX;
  if board[hinty+1,hintx+1].direct = updown then y:=SQY else y:=0;
  CopyRec (x,y,x+SQX-1,y+SQY-1,PX1+hintx*SQX,PY1+hinty*SQY,2,1);
  if lightnapr = 0 then
    begin
    addx:=1;
    addy:=0;
    end else
      begin
      addx:=0;
      addy:=1;
      end;
```

```
  x:=board[hinty+1+addy,hintx+1+addx].value*SQX;
  y:=0;
  if board[hinty+1+addy,hintx+1+addx].direct = updown then y:=SQY;
  CopyRec (x,y,x+SQX-1,y+SQY-1,
   PX1+(hintx+addx)*SQX,PY1+(hinty+addy)*SQY,2,1);
  hintx:=-1;
end; {RemoveHint1}

{ Снятие подсказки при level=2. Курсор не трогать! }
procedure  RemoveHint2;
VAR
  x,y,addx,addy : integer;

begin
  x:=350+board[hinty+1,hintx+1].value*SQX;
  y:=0;
  CopyRec (x,y,x+SQX-1,y+SQY-1,PX2+hintx*SQX,PY2+hinty*SQY,2,1);
  if lightnapr = 0 then
    begin
    addx:=1;
    addy:=0;
    end else
      begin
      addx:=0;
      addy:=1;
      end;
  x:=350+board[hinty+1+addy,hintx+1+addx].value*SQX;
  y:=0;
  CopyRec (x,y,x+SQX-1,y+SQY-1,
   PX2+(hintx+addx)*SQX,PY2+(hinty+addy)*SQY,2,1);
  hintx:=-1;
end; {RemoveHint2}

{ Обмен левой и правой половин массива board (для level = 2) }
procedure  Left_Right;
```

```
VAR
 i,j : integer;

begin
  for i:=1 to 8 do
    for j:=1 to 4 do SwapB (@board[i,j],@board[i,9-j]);
end; {Left_Right}

{ Обмен верхней и нижней половин массива board (для level = 2) }
procedure  Up_Down;
VAR
 i,j : integer;

begin
  for i:=1 to 4 do
    for j:=1 to 8 do SwapB (@board[i,j],@board[9-i,j]);
end; {Up_Down}

{ Транспонирование массива board (для level = 2) }
procedure  Trans;
VAR
 i,j : integer;

begin
  for i:=1 to 7 do
    for j:=i+1 to 8 do SwapB (@board[i,j],@board[j,i]);
end; {Trans}

{ Заполнение игрового поля (массива board) костями домино
  случайным образом. Для level = 1. }
procedure  MakeBoard1;
LABEL  2;
VAR
  i,j,k,r : word;
  direct : tdirect;
  x,y : integer;
  mo : boolean;
```

```
begin
    r:=Random (2);
    { Начальное заполнение игрового поля (массива board). }
2:  for i:=1 to 7 do
      for j:=1 to 8 do
        begin
        board[i,j].value:=9;  board[i,j].connect:=none;
        board[i,j].direct:=none1;
        end;
    { Выкладываем кости в board }
    k:=1;
    if r = 0 then
      { Кладем сверху вниз и слева направо }
      begin
      for i:=1 to 7 do
        for j:=1 to 8 do
          if board[i,j].value = 9 then
            begin
            { Выбираем случайное положение данной кости }
            posture:=indefinite;
            if (j < 8) and (board[i,j+1].value = 9) then
              posture:=horizontal;
            if i < 7 then byte(posture):=byte(posture) or
              byte(vertical);
            if posture = any then posture:=tposture(Random (2)+1);
            { Определили положение кости? }
            if posture = indefinite then goto 2;
            board[i,j].value:=die[k,1];
            if posture = horizontal then
             board[i,j+1].value:=die[k,2] else
               board[i+1,j].value:=die[k,2];
            { Задаем разрешенные направления для квадратиков 2, 3 и 6 }
            if posture = horizontal then
              begin
              if board[i,j].value in [2,3,6] then
                board[i,j].direct:=leftright;
```

```
        if board[i,j+1].value in [2,3,6] then
          board[i,j+1].direct:=leftright;
        end else
          begin
          if board[i,j].value in [2,3,6] then
            board[i,j].direct:=updown;
          if board[i+1,j].value in [2,3,6] then
            board[i+1,j].direct:=updown;
          end;
      Inc (k);
      end;
end
else
  { Кладем слева направо и сверху вниз }
  for j:=1 to 8 do
    for i:=1 to 7 do
      if board[i,j].value = 9 then
        begin
        { Выбираем случайное положение данной кости }
        posture:=indefinite;
        if (i < 7) and (board[i+1,j].value = 9) then
          posture:=vertical;
        if j < 8 then byte(posture):=byte(posture) or
          byte(horizontal);
        if posture = any then posture:=tposture(Random (2)+1);
        { Определили положение кости? }
        if posture = indefinite then goto 2;
        board[i,j].value:=die[k,1];
        if posture = horizontal then
          board[i,j+1].value:=die[k,2] else
            board[i+1,j].value:=die[k,2];
        { Задаем разрешенные направления для квадратиков 2, 3 и 6 }
        if posture = horizontal then
          begin
          if board[i,j].value in [2,3,6] then
            board[i,j].direct:=leftright;
```

```
                if board[i,j+1].value in [2,3,6] then
                  board[i,j+1].direct:=leftright;
                end else
                  begin
                  if board[i,j].value in [2,3,6] then
                  board[i,j].direct:=updown;
                  if board[i+1,j].value in [2,3,6] then
                  board[i+1,j].direct:=updown;
                  end;
              Inc (k);
              end;
    { Вертим массив board случайным образом }
    case Random (4) of
      { Обмен левой и правой половин }
0:    for i:=1 to 7 do
        for j:=1 to 4 do SwapD (@board[i,j],@board[i,9-j]);
      { Обмен верхней и нижней половин }
1:    for i:=1 to 3 do
        for j:=1 to 8 do SwapD (@board[i,j],@board[8-i,j]);
2:    begin
      { Обмен правой и левой, затем верхней и нижней половин }
      for i:=1 to 7 do
        for j:=1 to 4 do SwapD (@board[i,j],@board[i,9-j]);
      for i:=1 to 3 do
        for j:=1 to 8 do SwapD (@board[i,j],@board[8-i,j]);
      end;
    end; {case}
    mo:=movis;
    if mo then
      begin
      DirectDraw.WaitForVerticalBlank (DDWAITVB_BLOCKEND,0);
      MouseOff;
      end;
    { Рисуем игровое поле }
    for i:=0 to 6 do
      for j:=0 to 7 do
```

```
        begin
        direct:=board[i+1,j+1].direct;
        if direct = none1 then direct:=leftright;
        x:=board[i+1,j+1].value*SQX;  y:=SQY*(word(direct)-1);
        CopyRec (x,y,x+SQX-1,y+SQY-1,PX1+j*SQX,PY1+i*SQY,2,1);
        end;
    MouseOn;
end; {MakeBoard1}

{ Заполнение игрового поля (массива board) костями домино
  случайным образом. Для level = 2. }
procedure  MakeBoard2;
LABEL  2;
VAR
  i,j,k,x : integer;
  mo : boolean;

begin
   { Начальное заполнение игрового поля (массива board). }
2: for i:=1 to 8 do
     for j:=1 to 8 do
       begin
       board[i,j].connect:=none;
       board[i,j].value:=9;
       board[i,j].direct:=none1;
       { Отмечаем недоступные клетки игрового поля через 10 }
       if ((i = 1) or (i = 8)) and ((j = 4) or (j = 5)) or
           ((j = 1) or (j = 8)) and ((i = 4) or (i = 5)) then
           board[i,j].value:=10;
       end;
   { Выкладываем кости в board }
   k:=1;
   for i:=1 to 8 do
     for j:=1 to 8 do
       if board[i,j].value = 9 then
```

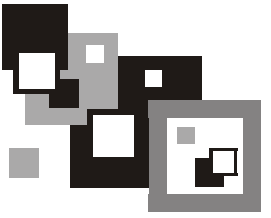
```
        begin
        { Выбираем случайное положение данной кости }
        posture:=indefinite;
        if (j < 8) and (board[i,j+1].value = 9) then
         posture:=horizontal;
        if (i < 8) and (board[i+1,j].value = 9) then
        byte(posture):=byte(posture) or byte(vertical);
        if posture = any then posture:=tposture(Random (2)+1);
        { Определили положение кости? }
        if posture = indefinite then goto 2;
        board[i,j].value:=die[k,1];
        if posture = horizontal then board[i,j+1].value:=die[k,2] else
         board[i+1,j].value:=die[k,2];
        Inc (k);
        end;
   { Вертим массив board случайным образом }
   case Random (8) of
0:   Trans;
1:   begin
     Trans;
     Left_Right;
     end;
2:   Left_Right;
3:   begin
     Left_Right;
     Up_Down;
     end;
4:   begin
     Trans;
     Left_Right;
     Up_Down;
     end;
5:   begin
     Trans;
     Up_Down;
     end;
```

```
6:   Up_Down;
    end; {case}
    mo:=movis;
    if mo then
      begin
      DirectDraw.WaitForVerticalBlank (DDWAITVB_BLOCKEND,0);
      MouseOff;
      end;
    { Рисуем игровое поле }
    for i:=0 to 7 do
      for j:=0 to 7 do
        begin
        x:=350+board[i+1,j+1].value*SQX;
        if board[i+1,j+1].value < 10 then
        CopyRec (x,0,x+SQX-1,SQY-1,PX2+j*SQX,PY2+i*SQY,2,1);
        end;
    MouseOn;
end; {MakeBoard2}

// Инициализация каретки
procedure  TECaret.Init (pos1,wx1,wy1,fx1,fy1,px1,py1,dl1,
  chrspace1 : integer; str1 : TStr);
begin
  pos:=pos1;
  wx:=wx1;
  wy:=wy1;
  fx:=fx1;
  fy:=fy1;
  px:=px1;
  py:=py1;
  dl:=dl1-1;
  chrspace:=chrspace1;
  str:=str1;
  trig:=0;
end; {TECaret.Init}
```

```
// Сохранение фона под кареткой
procedure  TECaret.SaveBkgnd (newpos : integer);
VAR
  x : integer;
  s : shortstring;

begin
  Move (str^,s,integer(str^[0])+1);
  s[0]:=char(newpos);
  x:=px+TextWidth (chrspace,pf,s);
  CopyRec (x,py,x,py+dl,fx,fy,1,2);
end; {TECaret.SaveBkgnd}

// Прячем каретку
procedure  TECaret.Hide;
VAR
  x : integer;
  s : shortstring;
begin
  Move (str^,s,integer(str^[0])+1);
  s[0]:=char(pos);
  x:=px+TextWidth (chrspace,pf,s);
  CopyRec (fx,fy,fx,fy+dl,x,py,2,1);
end; {TECaret.Hide}

// Показываем каретку
procedure  TECaret.Show (newpos : integer);
VAR
  x : integer;
  s : shortstring;
begin
  SaveBkgnd (newpos);
  Move (str^,s,integer(str^[0])+1);
  s[0]:=char(newpos);
  x:=px+TextWidth (chrspace,pf,s);
  CopyRec (wx,wy,wx,wy+dl,x,py,2,1);
```

```
  pos:=newpos;
  trig:=0;
end; {TECaret.Show}

// Постоянная работа программы по счетчику времени (2 раза в секунду)
procedure  Go;
begin
  if GameState = NAME then
    // Мигание каретки
    begin
    Inc (ECaret.trig);
    if ECaret.trig = 1 then
      begin
      DirectDraw.WaitForVerticalBlank (DDWAITVB_BLOCKBEGIN,0);
      MouseOff;
      ECaret.Hide;
      MouseOn;
      end;
    if ECaret.trig = 2 then
      begin
      DirectDraw.WaitForVerticalBlank (DDWAITVB_BLOCKBEGIN,0);
      MouseOff;
      ECaret.Show (ECaret.pos);
      MouseOn;
      end;
    end;
end; {Go}

end.
```

Глава 4



Интеграция Flash и Delphi

- Создаем игру Calculator на Flash
- Вставка ожидания перед ходом компьютера
- Создаем оболочку на Delphi для swf-файла
- Связь Flash-ролика с внешней программой

В этой главе мы создадим графику и вспомогательные функции для уже сделанной с помощью компонентов Delphi игры Calculator. Delphi-программа будет запускать на выполнение swf-файл с графикой на Flash и думать над ходами, а Flash будет лишь отображать ходы и выводить нужные надписи. SWF в переводе означает Small Web Format (малый Web-формат). Это выходной формат, в котором Flash создает файлы с графикой, программным кодом и звуком. Файлы SWF-формата могут визуализироваться проигрывателем Flash. На Web-страницах эти файлы проигрывает ActiveX-компонент, установленный в Windows, а кроме этого существует еще автономный проигрыватель Flash. Все эти проигрыватели находятся в папке Players внутри папки с установленной системой Flash.

Зачем нужно делать графику на Flash и управлять ей из Delphi? Вы ведь видели, что графика в нашей игре получилась не совсем игровая, а Flash в этом оформительском отношении дает фантастические возможности. Не выходя из среды разработки Flash, можно сделать законченную игру с потрясающей графикой и звуком, и это делается проще, чем в среде разработки Delphi (вы когда-нибудь пробовали самостоятельно кодировать антиалиасинг, вращение изображения или морфинг одного изображения в другое?). Кроме того, мы приобретем опыт и расширим наши возможности в программировании.

При всей своей привлекательности Flash имеет тот недостаток, что программы в ней выполняются не напрямую центральным процессором, а интерпретируются проигрывателем примерно так же, как интерпретируется

код JavaScript на страницах браузера или байт-код Perl и PHP их интерпретаторами. Поэтому на Flash пока не создашь программы, которая бы сильно играла, скажем, в шахматы. Кроме того, Flash-программа, в общем, не может создавать файлы и вызывать на выполнение другие программы из соображений безопасности. Вот здесь и приходит на помощь оболочка Delphi, которая может считать варианты в сто раз быстрее, чем Flash-проигрыватель, и к тому же работать с файлами.

Для наших целей вы можете использовать Flash, начиная с версии 5 и до самой последней. В этом примере главное для нас — научиться обмениваться информацией с Flash-игрой. Ввиду того, что в данной главе присутствует программирование на ActionScript, читатель должен немного ознакомиться с этим языком, который похож на JavaScript. Среда разработки Flash содержит справку по языку, функциям и объектам. По ходу дела я буду пояснять назначение создаваемого кода.

4.1. Создаем во Flash игру Calculator

Запустите на выполнение вашу версию среды разработки Flash и создайте, если необходимо, новый документ Flash. Вы увидите примерно такую картину, как на рис. 4.1.

Слева вы видите панель инструментов (**Tools**), вверху — *временну́ю диаграмму* (**Timeline**). На *рабочем поле* редактора находится *рабочая область*, это содержимое *кадра* фильма. У нас для него выбран белый цвет. Это то, что будет отображаться на форме Delphi. Справа расположено окно *библиотеки*, которое появляется и исчезает по нажатию клавиши <F11>. Снизу — *окно свойств* выбранного объекта (**Properties**), которое можно вывести и спрятать нажатием комбинации клавиш <Ctrl>+<F3>. Конечно, всем этим можно управлять также через меню **Window**.

На шкале времени уже имеется слой под названием **Layer 1**, в первом кадре которого стоит незакрашенный кружок — признак того, что этот кадр — ключевой и что в нем ничего нет. Это шкала времени *главной временной диаграммы*. В каждом кадре может быть свое содержимое, которое меняется, если щелчком мыши войти в какой-либо кадр или ухватить розовый прямоугольник и потянуть его вдоль временной диаграммы. У каждого *клипа фильма* (**Movie clip**) есть своя временная диаграмма, которая работает синхронно с главной временной диаграммой. Временные диаграммы и кадры делают возможным создание анимации. Ниже панели со слоями и над рабочей областью вы видите маленькую панель, на которой написано **12.0 fps**. Это частота кадров нашего будущего фильма (**Frames Per Second**). Изменить ее можно, сделав двойной щелчок по этой панели. Здесь же можно поменять цвет фона и размеры окна фильма. Установите размер окна

(**Dimensions**) в 460 × 370 точек, цвет фона — белый, **Frame rate** (Число кадров в секунду) установите в 15.

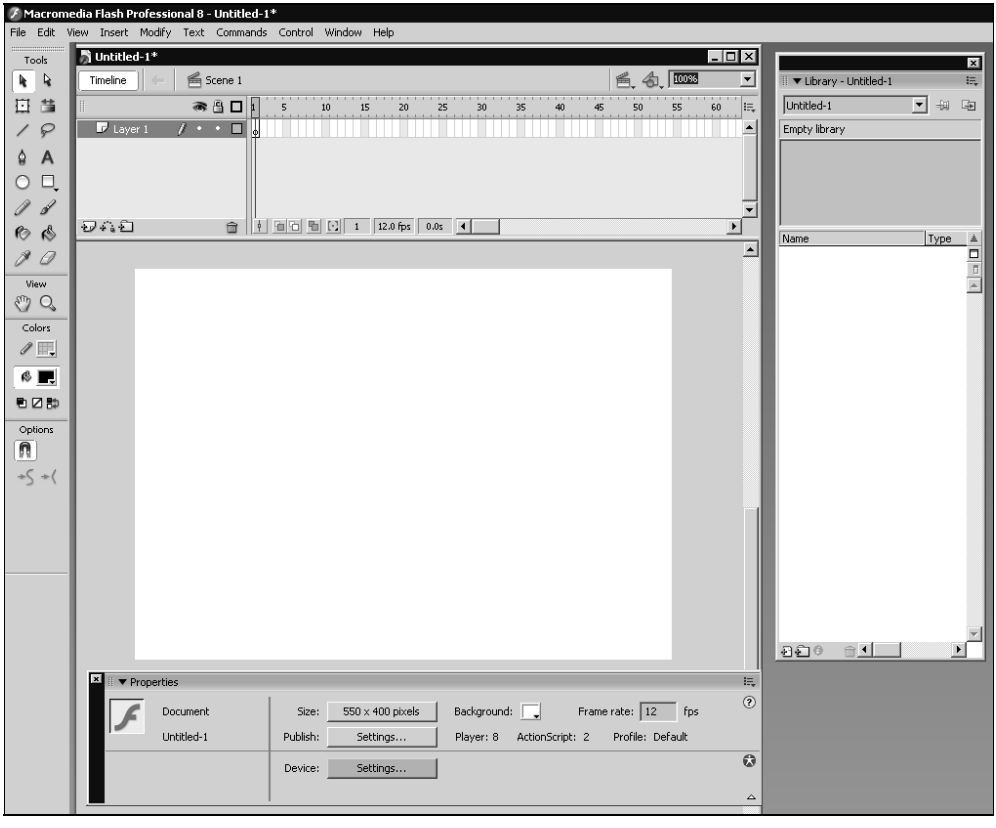


Рис. 4.1. Редактор Flash с пустым документом (для Flash 8)

Теперь давайте сохраним все это в виде fla-файла. Выберите в меню **File | Save** и в нужном каталоге сохраните файл под именем calculator.fla.

Зайдите в меню **File | Publish Settings** (Файл | Установки публикации) и на вкладке **Formats** нажмите кнопку **Use Default Names** (Использовать имена по умолчанию). В столбце **Type** оставьте флажок только у **Flash (.swf)**, HTML-документ нам создавать не нужно. На вкладке **Flash** выберите версию swf-файла **Flash 5**, чтобы наша программа для своей работы не требовала скачивать новый Flash-проигрыватель. Нажмите кнопку **OK**.

Мы сделаем графику так, как будто этот калькулятор нарисован на тетрадном клетчатом листе чернилами. Белый цвет фона для этого нам подходит.

Сначала сделаем разлиновку листа в клетку. Для этого создадим графический объект — линию и потом будем ее многократно использовать.

Выберите на панели инструментов **Line Tool**, на панели свойств (которая появляется и исчезает при нажатии комбинации клавиш <Ctrl>+<F3>) выберите тип линии (**Stroke Style**) **hairline** (Волосяная линия). Это линия толщиной в одну точку, которая не масштабируется при увеличении. У окошечка с изображением карандаша выберите цвет линии **#D5F2FF**. В меню **View | Snapping** (Просмотр | Привязка) оставьте флажок только у **Snap to Pixels** (Привязываться к точкам). На всякий случай щелкните во временной диаграмме на первом кадре и проведите через все рабочее поле горизонтальную линию. На самом деле эта линия может иметь любую длину, лишь бы она была горизонтальной. Длину и координаты левого верха можно задавать в числовых полях панели свойств выбранного объекта, выделив перед этим линию инструментом **Selection** (это черная стрелка на панели инструментов). Задайте этой линии длину на весь кадр и координаты $x = 0$, $y = 30$. Далее выделите эту прямую и продублируйте ее, нажав комбинацию клавиш <Ctrl>+<D>. Двигайте новую линию с помощью клавиш со стрелками. Если при этом нажать клавишу <Shift>, то перемещение будет сразу на 10 точек. Установите ее на 30 точек ниже первой прямой. Аналогично выделите обе линии, продублируйте их и разместите под уже имеющимися линиями. Поступая так дальше, заполните кадр горизонтальными прямыми.

Вертикальные прямые будем делать в другом слое, чтобы они не рассеклись на мелкие отрезки горизонтальными прямыми. Для создания нового слоя щелкните на прямоугольничке **Insert Layer** (листок с синим плюсиком) слева внизу временной диаграммы. Над слоем **Layer 1** возникнет слой **Layer 2**. Чем выше слой, тем ближе он к наблюдателю. Щелкните на первом кадре слоя **Layer 1**, чтобы выделить в нем все содержимое. Для этого можно также щелкнуть на рабочем поле и нажать комбинацию клавиш <Ctrl>+<A>. Скопируйте все это в буфер обмена нажатием комбинации клавиш <Ctrl>+<C>. Для того чтобы изображение в слое 1 не мешало рисованию, скройте этот слой, щелкнув справа от его названия на кружочке под изображением глаза. Перейдите в первый кадр слоя 2 и нажмите комбинацию клавиш <Ctrl>+ +<V> (<Ctrl>+<Shift>+<V> скопирует выделенные объекты в те же координаты). Все эти действия можно было также сделать с помощью меню **Edit**. Пока все эти линии выделены, нажмите комбинацию клавиш <Ctrl>+ +<Alt>+<S>, задайте угол поворота в 90° и поверните все линии. Выровняйте и размножьте их, задайте им нужную длину. Должно получиться так, как на рис. 4.2.

Сделаем заготовку для кнопок нашей игры. Это будет простой прямоугольник, как бы нарисованный чернилами и закрашенный белым цветом. Для этого нажмите комбинацию клавиш <Ctrl>+<F8>, чтобы открыть окно **Create New Symbol** (Создать новый символ), дайте ему имя `box`, тип поведения выберите **Graphic** (Графика) и нажмите кнопку **OK**. Откроется окно редак-

тирования этого символа, а в библиотеке появится первый символ **box**. Мы могли бы дать этому символу поведение **Movie clip** (Клип фильма). А изменить поведение и некоторые свойства символа, а также удалить его можно, щелкнув на его значке в библиотеке правой кнопкой.

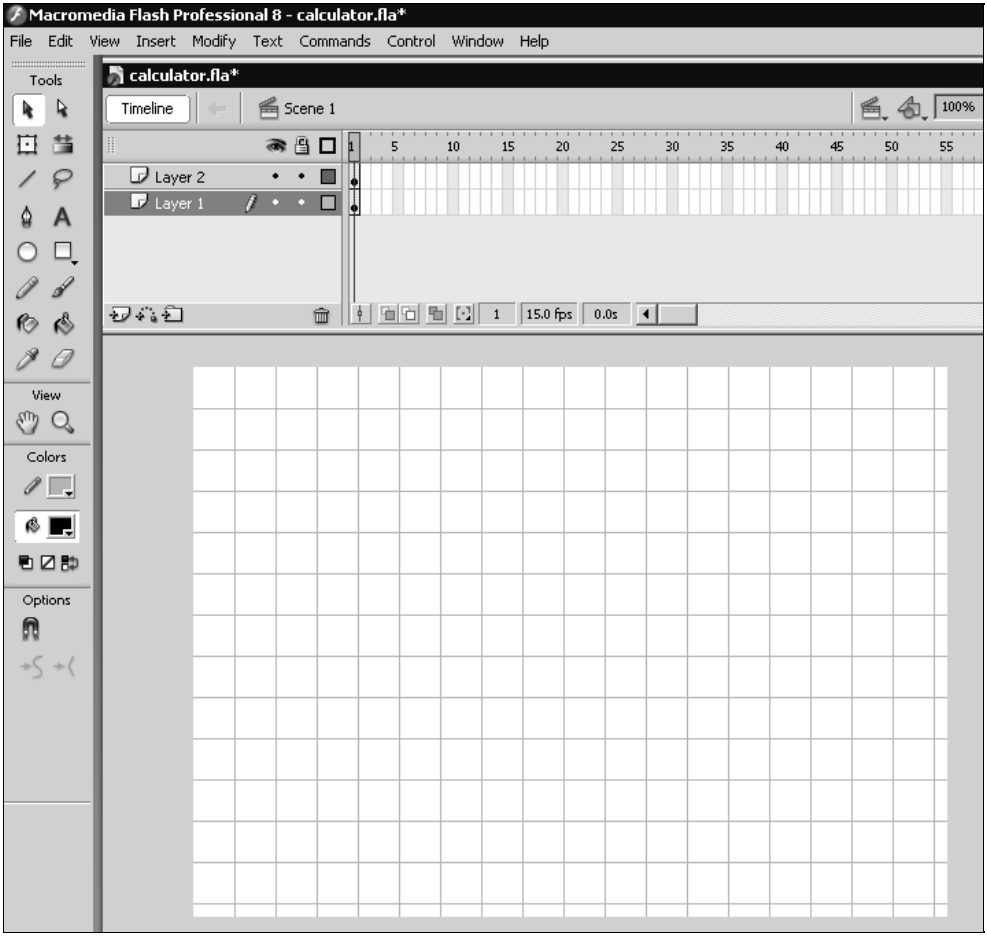


Рис. 4.2. Разлиновка тетрадного листа

Для удобства рисования вы можете через меню **View | Grid** (Вид | Сетка) отобразить сетку, установить для нее другой шаг и цвет. Но здесь она нам не понадобится.

Выберите инструмент **Rectangle** (Прямоугольник), задайте цвет штриха **#0066CC** или другой чернильный оттенок, для цвета заливки выберите пере-

черкнутый квадратик (он также присутствует на панели **Tools** под панелькой выбора цвета заливки). Ниже инструментов щелкните по закругленному уголку и в окне **Rectangle Settings** (Установки прямоугольника) задайте **Corner radius** (Радиус закругления) `5 points`. В панели свойств **Properties** дайте толщину линии (**Stroke height**) `3`, стиль штриха (**Stroke Style**) из выпадающего списка образцов выберите третий после **Solid** (Сплошной), он также будет третьим снизу. Это прерывистая линия, как бы нарисованная от руки. После всех этих настроек нарисуйте возле центра рабочего поля прямоугольник в общем-то произвольного размера. На рис. 4.3 смотрите, что получилось у меня.

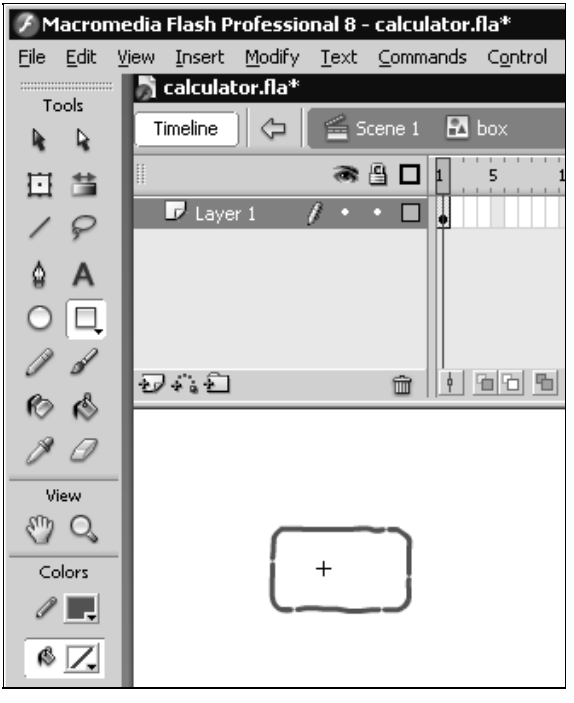


Рис. 4.3. Символ **box**

Этот крестик в центре рабочего стола является точкой регистрации клипа, относительно нее он деформируется и передвигается, а ее координаты являются координатами самого объекта. Как правило, мы будем выравнивать все нарисованные объекты по центру рабочего стола. Для этого существует меню **Modify | Align**, но удобнее выровнять клавишами. Для этого надо выделить объект, щелкнув на нем инструментом **Selection** (Выделение) и нажать комбинацию клавиш <Ctrl>+<Alt>+<2>, а потом <Ctrl>+<Alt>+<5>.

Также можно выравнивать объекты путем задания им новых координат в панели свойств выделенного объекта (которая возникает и прячется по нажатию комбинации клавиш <Ctrl>+<F3>).

Чтобы вернуться к основной сцене, щелкните на надписи **Scene 1** над временной шкалой.

Теперь сделаем девять кнопок для калькулятора. Это будут клипы, в которые мы программным путем будем вписывать цифры от 1 до 9. Нажмите комбинацию клавиш <Ctrl>+<F8>, в поле **Name** (Имя) введите `btCalc`, тип поведения выберите **Movie clip** и нажмите кнопку **OK**. В рабочее поле для только что созданного символа перетащите из библиотеки символ **box**, выделите его и дайте ему в панели свойств размер 50 × 50 точек и можете задать координаты точки привязки (которая изображается крестиком) `-25`, `-25`, чтобы квадратик оказался по центру. Для ввода текста выберите инструмент **Text** (это кнопка с буквой **A**), в панели свойств этого текста выберите шрифт Comic Sans MS, цвет ему дайте тот же **#0066CC**, размер (**Font size**) `32`, справа выберите **Align Center** (Выравнивать по центру), слева в выпадающем списке **Text Type** (Тип Текста) выберите **Dynamic Text** (Динамический текст), чтобы можно было определять надпись программно, в поле **Var** (Variable, переменная) введите имя `digit` (цифра) и отожмите кнопку **Selectable** (Выбираемый), чтобы нельзя было выделять цифры, протаскивая по ним указатель мыши. Для создания поля для текста щелкните указателем при активном инструменте **Text** внутри прямоугольника и растяните его за кружочек почти на весь прямоугольник, не меняя высоты поля. Выберите инструмент **Selection** (черную стрелку) и выровняйте это поле, чтобы оно было посередине прямоугольника. На рис. 4.4 вы видите, что должно получиться.

Не забывайте своевременно нажимать комбинацию клавиш <Ctrl>+<S>, сохраняя плоды кропотливого труда, а то вдруг отключат электричество.

Также нам нужны цифры красного цвета для показа возможных вариантов ходов игрока. Поэтому над слоем **Layer 1** сделаем слой **Layer 2** и в нем через буфер обмена продублируем на том же месте текстовое поле из слоя **Layer 1**. Но цвет ему зададим **#C833AA** и в поле **Var** напишем для него идентификатор `digit1`.

В пятой версии Flash клипы не могут работать как кнопки, поэтому для реакции кнопок калькулятора на щелчки кнопкой мыши разместим в этом клипе экземпляр кнопки, которую сейчас и сделаем.

Нажмите комбинацию клавиш <Ctrl>+<F8> и создайте объект типа **Button** с именем `b`. Эта кнопка будет невидимой, поэтому в кадрах **Up**, **Over** и **Down** рисовать ничего не будем. Эти кадры предназначены, соответственно, для вида кнопки в пассивном состоянии, когда указатель находится над ней, и когда она нажата. Но кнопка должна реагировать на щелчки. Для обозначения области, над которой должен находиться указатель, чтобы во время

щелчка кнопка реагировала, существует кадр **Hit**. Эта область не показывается, а выполняет техническую роль. Если этой области нет, то она принимается за фигуру, нарисованную в кадре **Up**. Щелкните в слое **Layer 1** на кадре **Hit** и нарисуйте закрашенный прямоугольник без контура такого же размера, как и клип кнопки калькулятора и выровняйте его по центру. Кнопка готова.

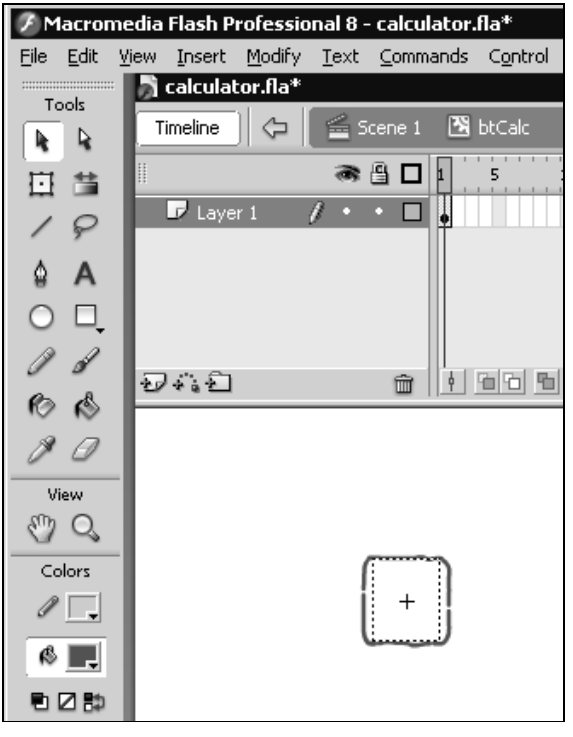


Рис. 4.4. Клип `btCalc`

Вернитесь к редактированию символа `btCalc`, создайте над слоем **Layer 2** слой **Layer 3** и в его первом кадре перетащите на клип кнопку `b`, разместив ее по центру.

Нам конечно же не терпится проверить, как это все работает. Сейчас увидим. Перейдите к сцене и над слоем **Layer 2** создайте слой **Layer 3**. Порядок слоев легко меняется их перетаскиванием. В слое **Layer 3** будем располагать все кнопки и панели нашей будущей игры. Над слоем **Layer 3** создайте слой **Actions** (Действия). Для переименования слоя дважды щелкните на его имени и введите новое. К кадрам этого слоя мы будем прикреплять код ActionScript.

Когда фильм (т. е. swf-файл) загружается с диска или из сети, то проигрыватель начинает работу по его отображению сразу, не дожидаясь, когда загрузятся все кадры. Поэтому, вообще говоря, чтобы не было обращений из программ к незагруженным объектам, нужно делать ожидание загрузки. Сейчас мы сделаем ожидание загрузки всех кадров. А пока все они не загрузятся, фильм будет циклиться между первым и вторым кадрами. Но давайте сначала сделаем второй кадр у всех слоев. Для этого выделите второй кадр в слое **Actions** нажатием на левую кнопку, затем установите указатель на второй кадр в нижнем слое и, удерживая нажатой клавишу <Shift>, щелкните на втором кадре нижнего слоя. В результате вторые кадры во всех слоях выделятся. Нажмите клавишу <F5>, чтобы сделать пустые кадры во всех выделенных слоях. Далее выделите второй кадр слоя **Actions** и нажмите клавишу <F6>, делая его ключевым, потому что он будет содержать код ActionScript. Вызовите окно ввода кода нажатием клавиши <F9>, в этом окне задвиньте влево панель с подсказками по вводу кода для непрограммистов, чтобы осталось только окно текстового редактора (этот режим использования окна кода еще называется **Expert mode**) и введите такой код:

```
if (!_framesloaded || _framesloaded != _totalframes) gotoAndPlay(1);
```

Этот условный оператор переводит головку проигрывателя командой `gotoAndPlay(1)` в первый кадр до тех пор, пока глобальная переменная `_framesloaded`, равная числу загруженных кадров, не сравняется с другой глобальной переменной `_totalframes`, которая означает число кадров этого фильма. Перейдя в первый кадр, проигрыватель из него продолжит воспроизведение фильма и опять перейдет во второй кадр. Этот условный оператор выполнится опять и так до тех пор, пока все кадры не загрузятся. Это стандартный код.

Если вы хотите изменить установки для окна ввода кода, то вызовите окно установок через меню **Edit | Preferences** или комбинацией клавиш <Ctrl>+<U>.

Перетащите из библиотеки на рабочий стол сцены в третий слой символ `btCalc` и в окне его свойств в поле **Instance Name** (Имя экземпляра) дайте имя этому экземпляру `bt00`. Цифры в имени означают номер строки и номер столбца кнопки калькулятора. Вернитесь в окно кода для второго кадра, щелкнув на этом кадре, и допишите такой код:

```
bt00.digit='1';

stop();
```

В первой строке мы присваиваем текстовому полю с именем `digit` клипа `bt00` значение `0`, а во второй останавливаем головку проигрывателя в этом кадре, иначе она из него пойдет в первый кадр и так будет ходить между ними циклически.

Для тестирования фильма нажмите комбинацию клавиш <Ctrl>+<Enter>. Опа, вы видите единичку в центре прямоугольника? Значит, вы все сделали, как надо. Можете убрать этот отладочный код, но оператор `stop()` оставьте, он войдет в конечный код игры.

Перетащите из библиотеки этот же клип еще восемь раз и создайте все девять кнопок калькулятора. Дайте им соответствующие имена как показано ниже:

```
bt00  bt01  bt02
bt10  bt11  bt12
bt20  bt21  bt22
```

Выровнять эти кнопки можно с помощью панели **Align** (Выравнивание), которая вызывается комбинацией клавиш <Ctrl>+<K>. Надо с нажатой клавишей <Shift> выделить ряд из трех кнопок и нажать соответствующую пиктограмму в этой панели. Также можно задавать общую координату **X** или **Y** всем выделенным объектам. Если вы ошиблись, отменяйте предыдущие шаги комбинацией клавиш <Ctrl>+<Z>. Я сделал шаг между кнопками равным семидесяти точкам.

Над слоями вы видите также символ замка́; как и в редакторе Photoshop, он запрещает редактирование в слое, где находится. Используйте его, чтобы случайно не задеть объекты из других слоев.

Замечание

В настройках **Preferences** можно выбрать шрифт и другие параметры для редактора кода. У меня в этих настройках стоит шрифт Courier New Cyr 10.

Теперь сделаем реакцию на отпускание кнопки мыши над кнопками калькулятора. Это делается путем привязывания кода ActionScript к экземплярам кнопки `b` в символе `btCalc`. Сделайте двойной щелчок на значке символа `btCalc` в библиотеке, выделите на нем экземпляр кнопки и в окне редактора кода напишите для нее такой код:

```
on(release) { _root.click(_name.substr(2,1),_name.substr(3,1)) }
```

Это обработчик события `release` — отпускания кнопки мыши над экземпляром кнопки `b`. Префикс `_root` означает, что вызываемая функция `click` будет находиться на главной временной диаграмме (во втором кадре, где и вся программа). Символ подчерка перед свойством объекта означает, что это его родное свойство, о котором известно проигрывателю Flash. Программист может добавлять свои свойства уже без подчерка вначале.

Как эта функция будет определять, над какой кнопкой калькулятора было отпускание кнопки мыши? Для этого мы и задавали имена экземплярам символа `btCalc` в определенном порядке. Свойство `_name` хозяина кнопки

как раз и есть это имя. Теперь в справочнике по ActionScript, нажав клавишу <F1>, смотрите на объект `String`. У него есть метод `substr`, который возвращает подстроку, начиная с данного индекса и данной длины. `_name` — это строка, поэтому она имеет метод `substr`. С его помощью мы выделяем из имени экземпляра символа `btCalc` его первую и вторую цифры и передаем их функции `click`, которую сейчас напишем.

Выделите второй кадр основной временной диаграммы и в редакторе кода добавьте к нему такой код:

```
// Щелчок на кнопке калькулятора в строке y и столбце x
function click(y,x)
{ trace('y='+y+' '+'x='+x);
}
```

Запустите игру на выполнение и пощелкайте по кнопкам калькулятора. В окне вывода (**Output**) вы должны видеть сообщения о координатах кнопки, над которой отпустили кнопку мыши. Уберите этот отладочный код, оставив саму эту функцию с пустым телом для дальнейшего его заполнения.

Следующий шаг — делаем индикатор. Если делать его на основе графики `box`, то придется увеличить его ширину и высоту. От этого толщина контура тоже станет больше. Поэтому сделаем отдельный клип для индикатора с теми же установками, с какими мы рисовали объект `box`.

Нажмите комбинацию клавиш <Ctrl>+<F8> и создайте клип с именем `indicator`. Дайте ему размеры 190 × 70, создайте аналогичное динамическое текстовое поле посередине клипа, размер шрифта возьмите побольше, например, `40`. В поле **Var** введите `ind` — имя для этого текстового поля. Индикатор готов. Остается перетянуть его в первый кадр третьего слоя сцены, там где расположены экземпляры кнопки `btCalc`, на рабочий стол выше этих кнопок и дать экземпляру то же имя `indicator`, через которое мы будем обращаться к его текстовому полю. Если ко второму кадру сцены добавить код

```
indicator.ind='35';
```

то в окне фильма мы увидим это число на индикаторе. У вас все в порядке? Тогда двигаемся дальше.

В том же втором кадре сцены разместим функцию `newGame`, которая будет размещать цифры на кнопках и число на индикаторе. Среда разработки Flash содержит справку по работе в редакторе и по языку ActionScript. Для получения справки по функциям этого языка нажмите клавишу <F1> и в открывшемся окне **Help** из выпадающего списка выберите **ActionScript 2.0** и дальше **Global Functions**.

В самое начало кода основной программы вставьте описание переменных:

```
var
 // Уровень игры, считая с 0 по 8
 level=2,
 // Признак, что начал игру компьютер
 compbegins,
 // Признак, что сделан первый ход
 gamestarted,
 // Признак, что игра закончена
 gameended,
 // Текущее значение на индикаторе
 ind,
 // Признак, что сейчас очередь хода человека
 isplayersmove,
 // Номер ряда, в котором игрок может делать ход
 numline,
 // Массив цифр, написанных на кнопках
 tabl=[[0,0,0],[0,0,0],[0,0,0]];
```

Вставьте код для функции `newGame`, как в листинге 4.1.

Листинг 4.1. Функция newGame

```
// Начало новой игры
function newGame()
{ var
   d=[1,2,3,4,5,6,7,8,9],
   i,
   j,
   temp;

  // Устанавливаем уровень игры в текстовом поле
  leveltext=level+1;
  // Признак, что начал игру компьютер/сделан первый ход/игра закончена
  compbegins=gamestarted=gameended=0;
  // Разрешаем игроку ходить первым
  isplayersmove=1;
```

```
  // Убираем сообщения
  showYouWon(0);
  showYouLost(0);
  showGameOver(0);
  // Перемешиваем массив d
  for (i=0; i < 9; i++)
   { // Получаем случайное целое число из диапазона от 0 до 8
     j=Math.floor(Math.random()*9);
     temp=d[i];
     d[i]=d[j];
     d[j]=temp;
   }
  // Размещаем случайные числа на кнопках
  for (i=0; i < 3; i++)
   for (j=0; j < 3; j++)
    { eval('bt'+i+j).digit=tabl[i][j]=d[i*3+j];
    }
  // И на индикаторе
  ind=Math.floor(Math.random()*6)+30;
  indicator.ind=ind;
  // Убираем красные цифры
  markRed();
}
```

Это конечный вид функции, поэтому на вызовы еще не существующих функций не обращайте внимания.

В этой функции создается массив `d`, состоящий из девяти чисел. Затем его элементы с помощью методов `random` и `floor` объекта `Math` перемешиваются. Метод `random` возвращает случайное число в диапазоне [0; 1[, а `floor` округляет число до ближайшего меньшего целого. Еще в функции `newGame` используется глобальная функция `eval`, которая по имени объекта выдает ссылку на него, так что потом можно пользоваться его свойствами и методами. Здесь в `eval('bt'+i+j)` `i` и `j` автоматически преобразуются в строку ввиду того, что первый элемент выражения `bt` — строка. И таким путем мы получаем все имена экземпляров кнопок по их индексам. В ActionScript символ `+` используется также для конкатенации строк как и в языке Pascal.

Вставьте в этот же кадр вызов функции `newGame`, отредактировав первую строку кода:

```
if (!_framesloaded || _framesloaded != _totalframes) gotoAndPlay(1);
 else newGame();
```

чтобы увидеть картинку начала игры. В ActionScript порядок следования функций не имеет значения, поэтому вызов функции может находиться раньше ее определения.

Далее сделаем кнопки **New Game**, **Comp. begins**, **Level**, **Help** и **Exit**. Так как они будут одинакового размера, то создадим графический объект `box1` аналогичный `box`, но размером 120 × 35 точек с прозрачной заливкой. Если вы сделаете только контур, то на таких кнопках можно будет щелкнуть, когда указатель находится на тексте внутри кнопки или на контуре. Есть еще один выход: на временной диаграмме кнопки имеется кадр **Hit**. Если кнопка не сплошная (например, это текст с URL), то в этом кадре можно нарисовать фигуру (обычно это прямоугольник), находясь над которой, указатель будет принимать вид руки, а кнопка будет реагировать на щелчки. Сам этот прямоугольник будет невидим в фильме.

Для порядка выравнивайте нарисованные объекты по центру рабочей области. Это удобно делать комбинацией клавиш <Ctrl>+<Alt>+<2> (по горизонтали) и <Ctrl>+<Alt>+<5> (по вертикали), когда объект для выравнивания выделен. На основе фигуры `box1` сделайте кнопки `btNew`, `btComp`, `btLevel`, `btHelp`, `btExit`. Текст набирайте размером 18, тип текста — **Static text** (Статический текст). В выпадающем списке **Font rendering method** (Метод визуализации шрифта) выберите **Anti-alias for animation** (Антиалиасинг при анимации), остальные установки те же, что и раньше. Антиалиасинг означает сглаживание краев путем вставки на границах объекта частично прозрачных точек. Имена кнопок не играют роли, потому что мы не будем к ним обращаться по именам.

Все эти кнопки надо разместить внутри клипа, чтобы их можно было прятать. Поэтому создайте клип `buttons` и перетяните в него все эти пять кнопок, как показано на рис. 4.5.

Кнопки выровнены по левому верхнему краю, шаг между кнопками равен 61 точке. Перетяните этот клип в сцену в тот же слой, где и все кнопки, и дайте его экземпляру имя `buttons`.

Выделите все имеющиеся объекты в сцене кроме прямых линий и переместите все это так, чтобы индикатор отстоял от левого и верхнего края окна на 30 точек. У меня x-координата правых кнопок равна 280. На рис. 4.6 вы видите окно проигрывателя с фильмом.

Рис. 4.5. Клип `buttons`

Присоединим к кнопке **New Game** код обработчика события отпускания кнопки мыши над этой кнопкой. Для этого выделите кнопку `btNew` в сцене и в редакторе кода наберите такой код:

```
on(release) { newGame() }
```

Далее делаем клип, показывающий правила игры. Для этого опять нажмите комбинацию клавиш <Ctrl>+<F8>, дайте имя символу `help`, тип **Movie clip** и установите флажок **Export for ActionScript** (Экспорт в ActionScript). Имя символа продублируется в поле **Identifier** (Идентификатор) как на рис. 4.7.

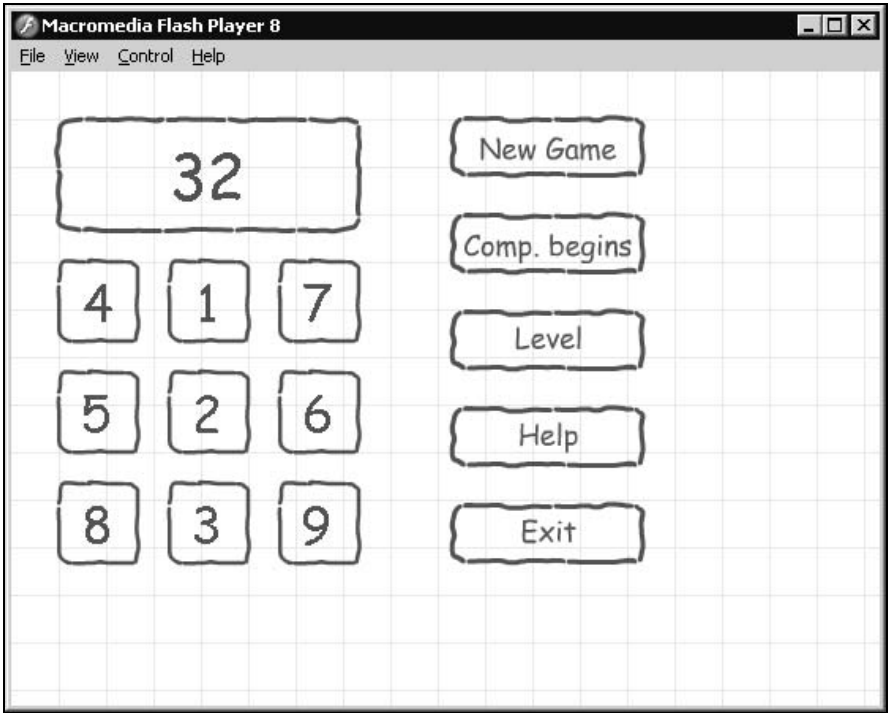


Рис. 4.6. Окно проигрывателя с фильмом

Щелкните по кнопке **OK**.

Выберите инструмент **Text** и создайте текстовое поле. Нам здесь понадобится русифицированный шрифт, чтобы видеть русские буквы, поэтому выберите Arial Cyr, размер шрифта установите равным 13. Наберите такой текст:

```
Игра Calculator, Copyright © Сергей Мельников, 2006.

Эта игра создана специально для книги "Turbo Pascal и Delphi на
занимательных примерах". Правила ее просты: первый игрок
может нажать любую из девяти кнопок. После каждого нажатия
число на индикаторе уменьшается на цифру, написанную на этой
кнопке. Второй игрок может нажать любую из трех кнопок в
том столбце, в котором делал предыдущий ход первый игрок,
а первый игрок в свои последующие ходы может нажимать
любую из трех кнопок в той строке, в которой второй игрок
нажимал кнопку последний раз. Проигрывает тот игрок, после
чьего хода на индикаторе появится отрицательное число.
```

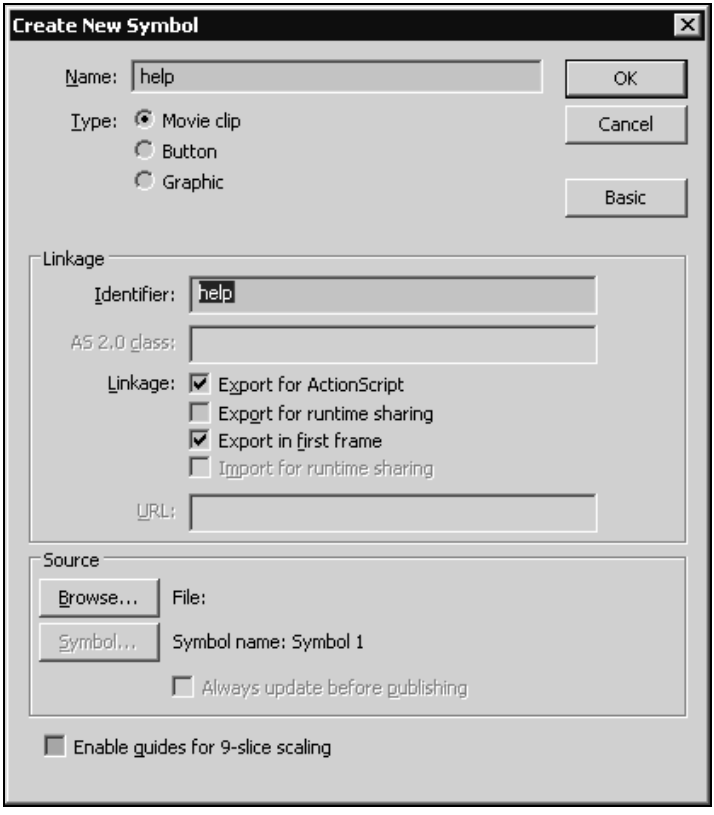


Рис. 4.7. Окно создания символа `help`

Для набора символа © нажмите клавишу <NumLock> и при нажатой клавише <Alt> на дополнительной клавиатуре наберите 4-значный код этого символа: 0169. Размер текстового поля выберите таким, чтобы не было лишних переносов строк. Тип текста может быть как статическим, так и динамическим. В первом случае вы можете выбрать метод визуализации **Anti-alias for animation** и **Use device fonts** (Использовать шрифты устройства). В первом случае текст будет с антиалиасингом, а во втором — нет, и при этом будет использоваться шрифт, установленный на компьютере пользователя. Иначе используемые символы шрифта будут запомнены в swf-файле и его размер увеличится на 5 Кбайт. Вы можете это увидеть, если в окне, отображаемом командой **File | Publish Settings...**, на вкладке **Flash** установите флажок **Generate size report** (Создавать отчет о размере). При этом наряду с созданием swf-файла будет создаваться файл "calculator Report.txt" с информацией о том, что и сколько байтов занимает в swf-файле.

Под текстом правил игры нужна будет кнопка **OK**, чтобы вернуться в игру. Сделайте кнопку `btOK` аналогично остальным кнопкам и перетяните ее из библиотеки в клип `help`. Выделите ее экземпляр в клипе `help` и присоедините к ней такой код:

```
on(release) { _root.toggleHelp() }
```

Он означает, что при нажатии и отпускании кнопки мыши на этой кнопке будет вызвана функция `toggleHelp` из главной временной диаграммы. Она будет прятать правила игры, когда клип `help` виден, и показывать их, когда он не виден.

В листинге 4.2 видна эта функция вместе с аналогичной функцией переключения видимости для всех кнопок, использующихся в игре. Вставьте ее в основную программу.

Листинг 4.2. Функции `toggleHelp` и `toggleButtons`

```
// Прячет или показывает правила игры и все кнопки
function toggleHelp()
{ if (toggleHelp.ishelp == 1)
   { help.removeMovieClip()
     toggleButtons(1);
     toggleHelp.ishelp=0;
   } else
    { attachMovie('help','help',5);
      help._x=230;
      help._y=185;
      toggleButtons(0);
      toggleHelp.ishelp=1;
    }
}

// Прячем при arg == 0 или показываем при arg == 1 кнопки калькулятора
function toggleButtons(arg)
{ buttons._visible=arg;
  for (i=0; i < 3; i++)
   for (j=0; j < 3; j++)
    eval('bt'+i+j)._visible=arg;
}
```

В ActionScript функции являются объектами, поэтому мы можем создавать для них свойства, имена которых пишутся через точку после имени функции. Эти свойства работают аналогично глобальным переменным и хранят свое значение между вызовами функции. Этим мы и воспользовались, создав у функции `toggleHelp` свойство `ishelp` — флаг того, что клип `help` присутствует на рабочем столе. Если он присутствует, мы удаляем его методом клипа `removeMovieClip` и переключаем все кнопки в видимое состояние. Иначе мы с помощью глобальной функции `attachMovie` извлекаем клип `help` из библиотеки и размещаем его на рабочем столе в координатах 230, 185. При этом мы делаем вызов `toggleButtons(0)`, чтобы спрятать все кнопки, иначе они станут "просвечивать" под клипом `help` и будут реагировать на щелчки. Чтобы уметь скрывать кнопки, мы их все разместили в клипах, свойству `_visible` которых присваиваем `0`.

Но нам нужен прямоугольник для фона под текстом и кнопкой, иначе окно фильма будет просвечивать под клипом `help`. Для этого создайте новый слой **Layer 2**, перетяните его ниже слоя **Layer 1** и нарисуйте в нем прямоугольник белого цвета без контура по размеру окна фильма. Выровняйте его по центру рабочей области. Текст и кнопку тоже выровняйте, чтобы хорошо смотрелось. Сравните с тем, что получилось у меня на рис. 4.8. Я на время сделал цвет фона более темным, чтобы выделить этот белый прямоугольник.

Если вы хотите, чтобы под правилами игры также были клетки, то вы можете через буфер обмена вставить их из сцены в клип `help`. Также вы можете создать отдельный клип для клеточного фона этой игры и перетаскивать его из библиотеки в нужные места. Чтобы создать символ из выделенных объектов, надо нажать клавишу <F8> и дать имя этому символу.

Мы упорно идем к нашей цели: во Flash-ролике будет вся функциональность кроме ресурсоемкой функции `isWin`, которую ActionScript не "потянет" ввиду своей медлительности. Эта функция будет находиться в Delphi-оболочке для этого Flash-ролика, который пока еще неизвестным способом будет общаться с этой оболочкой. Сейчас мы сделаем отображение уровней игры с 1 по 9.

Создайте в слое **Layer 5** в первом кадре сцены текстовое поле правее кнопки **Level**. Тип текста должен быть **Dynamic Text**, чтобы можно было менять его значение программно, шрифт — Comic Sans MS, его размер 24, цвет такой же, что и у остального текста, а в поле **Var** введите его идентификатор `leveltext`. После этого мы из любого места можем присваиванием `_root.leveltext=3` устанавливать его значение, а из кода, прикрепленного к кадрам главной временной диаграммы, можно к нему обращаться просто как к `leveltext`.

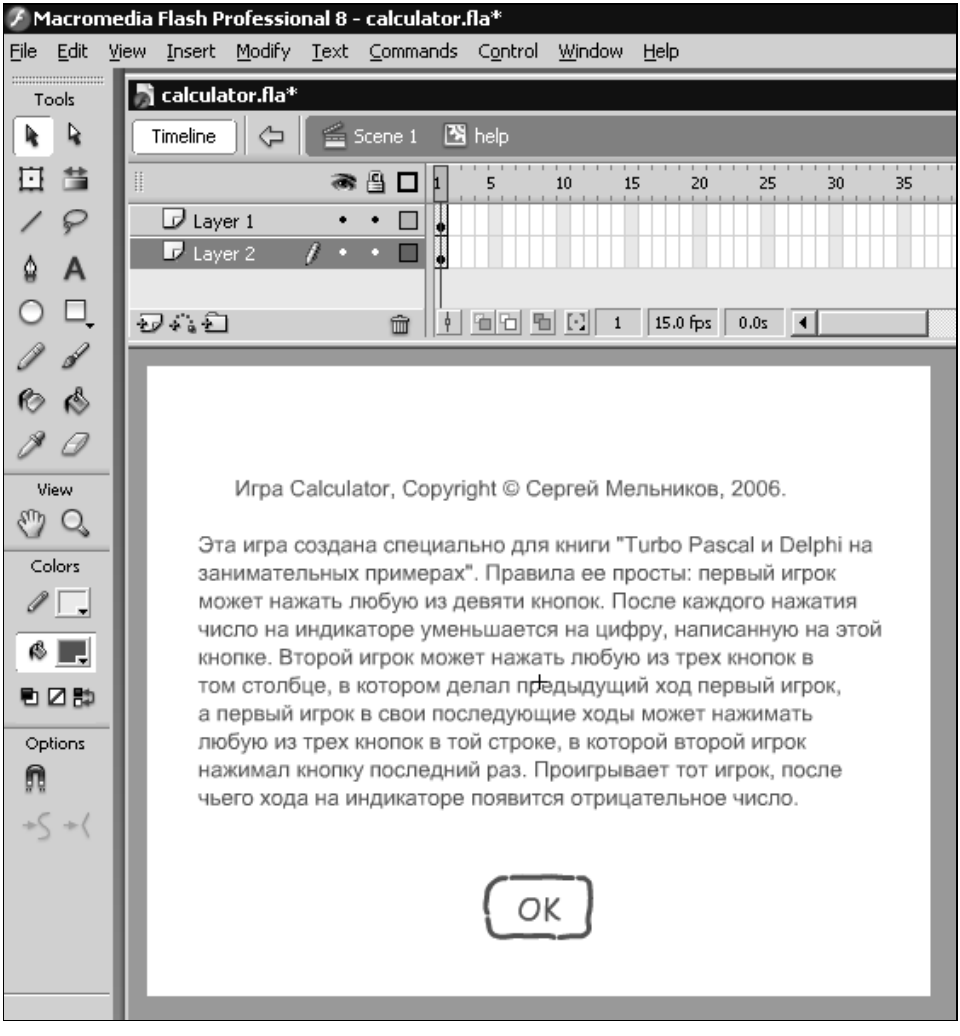


Рис. 4.8. Клип `help`

Если вы помните, `3` — это начальное значение для уровня игры, а счет идет с нуля. Поэтому переменной `level` сразу присвоено значение `2`. Выделите кнопку **Level** и присоедините к ней такой код:

```
on(release)
{ level=(level+1) % 9;
  leveltext=level+1;
}
```

Вы заметили, что мы здесь не использовали префикс `_root`? Это потому, что область видимости из кода, присоединенного к кнопке, — вся временная диаграмма, на которой эта кнопка находится.

Чтобы уровень было видно сразу после загрузки фильма, в функцию `newGame` был вставлен код

```
leveltext=level+1;
```

Вот и уровень игры у нас заработал.

Из клипов остается сделать показ результата игры, показ хода компьютера и информирование, что игра закончена.

Создайте символ с именем `gameover` и экспортом для ActionScript. В нем зеленым цветом, тем же шрифтом размера 18 создайте статический текст **Game Over!**. Выровняйте символ по левому верхнему краю. Аналогично создайте символы с именами `youlost` и `youwon` и экспортом для ActionScript. Надписи в них, соответственно, **Sorry, you lost** синим цветом и **Congratulations, you won!** красным.

Чтобы игрок видел, на какую кнопку нажал компьютер, сделаем анимированный клип с именем `hint` и экспортом этого имени в ActionScript. Это вначале будет маленькое колечко, возникающее над центром нажатой кнопки, которое будет расти и становиться все более прозрачным, пока не исчезнет из глаз.

В первом кадре символа `hint` инструментом **Oval** нарисуйте кольцо без заливки с шириной и высотой 6 точек, цветом штриха **#80B3E6** и прозрачностью **Alpha** 50%, толщина штриха — 1 точка, линия типа **Solid**. Чтобы нарисовать окружность, держите нажатой клавишу <Shift>. Выровняйте кольцо по центру. Выделите на временной диаграмме этого символа 15-й кадр и нажмите клавишу <F6>, чтобы сделать его ключевым. Это кольцо продублируется во все кадры вплоть до 15-го. Выделите это кольцо в 15-м кадре и вызовите окно **Scale and Rotate** комбинацией клавиш <Ctrl>+<Alt>+<S>. В поле **Scale** установите масштаб 1000%, чтобы кольцо увеличилось в 10 раз. Кроме того, в поле **Alpha** в окне свойств объекта введите 5%, чтобы это кольцо было едва видимо. Таким образом, у кольца будет анимироваться размер и прозрачность. Для анимации фигур (shapes) существует тип анимации **shape tween**. Войдите в первый кадр и в окне свойств объекта из выпадающего списка **Tween** выберите **Shape**. Должна возникнуть стрелочка на зеленом фоне к 15-му кадру. Это означает, что анимация создана. Вы можете захватить мышью розовый прямоугольничек над временной шкалой и подвигать его, наблюдая анимацию.

К 15-му кадру прикрепите код

```
this.removeMovieClip();
```

который удаляет клип с рабочего стола по окончании его анимации.

В листинге 4.3 показан весь код основной программы. В нем есть отладочные вызовы функции `trace` для показа ходов и нажатых кнопок. Ходы за компьютер носят характер заглушки и выбираются случайным путем. Этот проект сохранен в каталоге chapter4\1 на компакт-диске.

Листинг 4.3. Первоначальный код основной программы

```
var
 // Уровень игры, считая с 0 по 8
 level=2,
 // Признак, что начал игру компьютер
 compbegins,
 // Признак, что сделан первый ход
 gamestarted,
 // Признак, что игра закончена
 gameended,
 // Текущее значение на индикаторе
 ind,
 // Признак, что сейчас очередь хода человека
 isplayersmove,
 // Номер ряда, в котором игрок может делать ход
 numline,
 // Массив цифр, написанных на кнопках
 tabl=[[0,0,0],[0,0,0],[0,0,0]];

if (!_framesloaded || _framesloaded != _totalframes) gotoAndPlay(1);
 else newGame();

// Щелчок на кнопке калькулятора в строке y и столбце x
function click(y,x)
{ if (!isplayersmove || gameended ||
   gamestarted && (compbegins && numline != x || !compbegins &&
    numline != y)) return;
  trace('y='+y+' '+'x='+x);
  // Выполняем ход x, y
  indicator.ind=ind-=tabl[y][x];
  if (ind == 0)
```

```
    // Игрок выиграл
   { gameended=1;
     showYouWon(1);
     return;
   }
  if (ind < 0)
     // Игрок проиграл
   { gameended=1;
     showYouLost(1);
     return;
   }
  gamestarted=1;
  isplayersmove=0;
  // Ход компьютера в строке y или столбце x
  if (compbegins) compMove(1,y); else compMove(0,x);
}

// Прячем при arg == 0 или показываем при arg == 1 кнопки калькулятора
function toggleButtons(arg)
{ buttons._visible=arg;
  for (i=0; i < 3; i++)
   for (j=0; j < 3; j++)
    eval('bt'+i+j)._visible=arg;
}

// Начало новой игры
function newGame()
{ var
   d=[1,2,3,4,5,6,7,8,9],
   i,
   j,
   temp;

  // Устанавливаем уровень игры в текстовом поле
  leveltext=level+1;
```

```
  // Признак, что начал игру компьютер/сделан первый ход/игра закончена
  compbegins=gamestarted=gameended=0;
  // Разрешаем игроку ходить первым
  isplayersmove=1;
  // Убираем сообщения
  showYouWon(0);
  showYouLost(0);
  showGameOver(0);
  // Перемешиваем массив d
  for (i=0; i < 9; i++)
   { // Получаем случайное целое число из диапазона от 0 до 8
     j=Math.floor(Math.random()*9);
     temp=d[i];
     d[i]=d[j];
     d[j]=temp;
   }
  // Размещаем случайные числа на кнопках
  for (i=0; i < 3; i++)
   for (j=0; j < 3; j++)
    { eval('bt'+i+j).digit=tabl[i][j]=d[i*3+j];
    }
  // И на индикаторе
  ind=Math.floor(Math.random()*6)+30;
  indicator.ind=ind;
  // Убираем красные цифры
  markRed();
}

// Прячет или показывает правила игры и все кнопки
function toggleHelp()
{ if (toggleHelp.ishelp == 1)
   { help.removeMovieClip()
     toggleButtons(1);
     toggleHelp.ishelp=0;
   } else
    { attachMovie('help','help',5);
```

```
      help._x=230;
      help._y=185;
      toggleButtons(0);
      toggleHelp.ishelp=1;
    }
}

// Отмечаем красным ряд номер num. Если row_col == 1, то это строка,
// если row_col == 1, то столбец
function markRed(row_col,num)
{  for (var i=0; i < 3; i++)
    for (var j=0; j < 3; j++)
    { eval('bt'+i+j).digit1='';
      if (row_col == 1 && i == num) eval('bt'+i+j).digit1=tabl[i][j];
      if (row_col == 0 && j == num) eval('bt'+i+j).digit1=tabl[i][j];
    }
}

// Ход компьютера. Если row_col == -1, то это первый ход, если
// 0, то в строке num, если 1, то в столбце num
function compMove(row_col,num)
{ var i,j;

  trace('row_col='+row_col+' '+'num='+num);
  if (row_col == -1)
   { i=Math.floor(Math.random()*3);
     j=Math.floor(Math.random()*3);
      { if (ind-tabl[i][j] < 0)
         { showYouWon(1);
           markRed();
           gameended=1;
           return;
         }
        showHint(i,j);
        trace('Ход '+tabl[i][j]);
        indicator.ind=ind-=tabl[i][j];
        markRed(0,j);
```

```
        numline=j;
        gamestarted=1;
        isplayersmove=1;
        return;
       }
    }
  if (!row_col)
    { i=Math.floor(Math.random()*3);
      showHint(i,num);
      trace('Ход '+tabl[i][num]);
      indicator.ind=ind-=tabl[i][num];
      if (ind < 0)
       { showYouWon(1);
         markRed();
         gameended=1;
         return;
       }
      markRed(1,i);
      numline=i;
    } else
     { j=Math.floor(Math.random()*3);
       indicator.ind=ind-=tabl[num][j];
       showHint(num,j);
       trace('Ход '+tabl[num][j]);
       if (ind < 0)
        { showYouWon(1);
          markRed();
          gameended=1;
          return;
        }
       markRed(0,j);
       numline=j;
     }
  gamestarted=1;
  isplayersmove=1;
}
```

```
// Щелчок по кнопке Comp. begins
function btCompClick()
{ if (gamestarted || gameended) return;
  compbegins=1;
  compMove(-1);
}

// Щелчок по кнопке Level
function btLevelClick()
{ if (!gamestarted || gameended)
   { level=(level+1) % 9;
     leveltext=level+1;
   }
}

// Показ или удаление надписи "You won"
function showYouWon(arg)
{ if (!arg) youwon.removeMovieClip(); else
   { attachMovie('youwon','youwon',2);
     youwon._x=25;
     youwon._y=330;
     showGameOver(1);
   }
}

// Показ или удаление надписи "You lost"
function showYouLost(arg)
{ if (!arg) youlost.removeMovieClip(); else
   { attachMovie('youlost','youlost',2);
     youlost._x=25;
     youlost._y=330;
     if (ind < 0) showGameOver(1);
   }
}

// Показ или удаление надписи "Game over"
function showGameOver(arg)
```

```
{ if (!arg) gameover.removeMovieClip(); else
   { attachMovie('gameover','gameover',3);
     gameover._x=293;
     gameover._y=330;
   }
}

// Показ анимации над нажатой компьютером кнопкой калькулятора
function showHint(i,j)
{ attachMovie('hint','hint',4);
  hint._x=55+70*j;
  hint._y=145+70*i;
}

stop();
```

Если запустить этот вариант игры на выполнение, то мы увидим, что ответные ходы компьютер делает мгновенно, и на индикаторе даже не видно очков после хода человека. Но нам хочется, чтобы после того, как человек щелкнет на какой-либо цифре, на индикаторе отобразился бы результат его хода, потом было бы ожидание на одну секунду, и после этого ход компьютера. Но посмотрите в листинге на код функции `click`: в ней выполняется ход (`x`, `y`) человека, уменьшается значение на индикаторе, затем происходит вызов функции `compMove` хода за компьютер, в которой значение на индикаторе также уменьшается на величину хода компьютера. Проигрыватель Flash не может отобразить промежуточное значение на индикаторе, потому что он не обновляет экран между вызовами функций `click` и `compMove`. Что же делать? Может быть, в функцию `click` перед вызовом `compMove` поставить цикл ожидания:

```
var t=getTimer();
while (getTimer()-t < 1000) {}
```

Это приведет лишь к тому, что конечный результат возникнет на секунду позже. Не поможет тут и глобальная функция `updateAfterEvent()`, которая обновляет экран внутри обработчиков событий, не дожидаясь события наступления следующего кадра. Здесь надо разорвать непосредственную связь между вызовом функции `click` и `compMove`, и с таким методом мы сейчас ознакомимся.

Сначала сделаем клип с именем `wait`. Это будет пустой клип, в нем будет лишь некоторый код ActionScript. К первому кадру этого клипа присоедините код

```
stop();
```

Щелкните на втором кадре, нажмите клавишу <F6>, чтобы создать ключевой кадр, и присоедините к нему код

```
var
 t=getTimer(),
 row_col,
 num;
```

Аналогично создайте код в кадре **4**:

```
if (getTimer()-t < 1000) gotoAndPlay(3);
```

И наконец, в кадре **5** создайте код

```
_root.compMove(row_col,num);
```

Перетяните этот клип из библиотеки на рабочий стол в первый кадр основной диаграммы в слой **Layer 3** и дайте имя экземпляру также `wait`. Номер слоя и положение клипа здесь не играют роли, у меня он находится немного выше и левее графического содержимого кадра.

В функцию `compMove` добавим третий параметр — `waiting`. Если он существует (задан при вызове этой функции), то это означает, что нужно реализовать ожидание на одну секунду перед выполнением хода компьютера. А это ожидание как раз и делается с помощью клипа `wait`. Теперь начало функции `compMove` выглядит так:

```
function compMove(row_col,num,waiting)
{ var i,j;


  if (waiting)
   { wait.row_col=row_col;
     wait.num=num;
     wait.play();
     return;
   }
```

А вызовы из функции `click` теперь будут такими:

```
// Ход компьютера в строке y или столбце x
if (compbegins) compMove(1,y,1); else compMove(0,x,1);
```

Теперь скажу, как эта схема работает. После вызова `compMove(1,y,1)` или аналогичного с ненулевым и непустым третьим параметром логическое условие `if (waiting)` в данной функции становится истинным, и выполняется код внутри этого оператора `if`. Этот код запоминает внутри клипа `wait` первые два параметра функции `compMove` и вызывает метод `play` клипа `wait`, командой `return` передавая управление проигрывателю Flash. Проигрыватель отрабатывает команду `play` для клипа `wait`, и проигрывающая головка, до этого остановленная в первом кадре клипа его командой `stop()`, начинает двигаться по кадрам и выполнять код, который к ним прикреплен. В результате выполняется секундное ожидание в цикле между третьим и четвертым кадрами, после которых из пятого кадра вызывается функция `compMove` с ее собственными параметрами, сохраненными в переменных клипа `wait`. После этого головка Fash-проигрывателя переходит из последнего кадра клипа `wait` в первый кадр и там останавливается все той же командой `stop()`. Клип опять готов выполнить ожидание.

Код цикла в третьем и четвертом кадрах можно было не писать, достаточно передвинуть пятый кадр на место пятнадцатого с помощью мыши. Ведь у нас частота кадров — пятнадцать в секунду. Еще можно было не размещать экземпляр этого клипа постоянно в фильме, а из функции `compMove` вызывать его из библиотеки командой `attachMovie`, предварительно дав ему идентификатор для обращения из ActionScript. Тогда строку кода `stop()` из первого кадра клипа можно было бы убрать, а код в пятом кадре изменить на такой:

```
_root.compMove(row_col,num);
this.removeMovieClip();
```

Вторая строчка удаляла бы этот клип с рабочего стола. Соответствующий код в функции `compMove` в этом случае выглядел бы так:

```
if (waiting)
 { attachMovie('wait','wait',6);
   wait.row_col=row_col;
   wait.num=num;
   return;
 }
```

4.1.1. Добавляем звук

Какая игра без звука? Для оживления игры надо добавить к ней звук, тем более, что в редакторе Flash это сделать легко. Мы хотим озвучить ход человека и ход компьютера.

Звук импортируется так же, как и изображения, с помощью меню **File | Import... | Import to Library...**. Редактор Flash понимает звуки в формате WAV и MP3. Кроме того, при публикации фильма можно устанавливать большее их сжатие. Для этого есть соответствующие пункты в меню **File | Publish Settings...**, которые позволяют перекрыть индивидуальные установки для звуков при их импорте. Установите флажок **Override sound settings** (Перекрыть установки для звуков), нажмите кнопки **Set...** и установите значения:

- **Compression**: **MP3**;
- **Bit rate**: **16 kbps**;
- **Quality**: **Fast**.

Таким путем мы сжимаем эти звуки в MP3 и уменьшаем размер выходного файла.

Импортируйте оба звука, humanmove.wav и playermove.wav, которые находятся в каталоге chapter4\2, в библиотеку. Затем щелкните правой кнопкой мыши на значках этих звуков в библиотеке и в меню **Sound Properties** или **Linkage** установите флажок экспорта в ActionScript и идентификаторы, соответственно, `humanmove` и `playermove`.

Во Flash звуки можно перетаскивать в клипы, и они будут звучать, когда клипы играют, а можно управлять ими программно подобно функции `attachMovie`. Мы будем использовать второй вариант. В нем возможно изменение громкости и панорамирование звуков (см. методы объекта `Sound`).

Продолжите секцию данных основной программы и добавьте к ней такие переменные:

```
humansnd=new Sound();
compsnd=new Sound();
```

Метод `start` объекта `Sound` играет звук. В функцию `showHint` добавьте код

```
compsnd.start();
```

А в функцию `click` после строки

```
indicator.ind=ind-=tabl[y][x];
```

добавьте строку

```
humansnd.start();
```

Но эти звуки надо извлечь из библиотеки методом `attachSound`. Этот код можно вставить в код проверки загрузки игры. Он теперь будет выглядеть так:

```
if (!_framesloaded || _framesloaded != _totalframes) gotoAndPlay(1); else
 { humansnd.attachSound('humanmove');
```

```
    compsnd.attachSound('compmove');
    newGame();
}
```

Теперь при ходах игрока и компьютера будут слышны звуки.

Улучшенный вариант этой игры присутствует в каталоге chapter4\2 на компакт-диске.

Переходим к созданию Delphi-оболочки и функции `isWin` для этой игры. Но сначала надо установить в среде программирования Delphi компонент **Shockwave Flash**, чтобы перетянуть его на форму программы и воспользоваться его методами.

4.2. Устанавливаем компонент *Shockwave Flash*

Для того чтобы в программе на Delphi можно было отображать Flash-фильмы, надо установить компонент **Shockwave Flash**, который будет присутствовать в системе, если установлен проигрыватель Flash. Пятая версия проигрывателя идет в поставке Windows XP (или Internet Explorer), а последнюю версию можно скачать и установить, набрав в браузере URL **http://www.macromedia.com/shockwave/download/download.cgi?P1_Prod_Version=ShockwaveFlash**.

Для установки компонента **Shockwave Flash** запустите Delphi и выберите пункт меню **Component/Import ActiveX Control...** (Компонент/импорт элемента управления ActiveX...). Перед вами откроется диалоговое окно с заголовком **Import ActiveX Control** (Импорт элемента управления). В разделе **Registered Controls** (Зарегистрированные элементы управления) выберите **Shockwave Flash**. В разделе **Pallete Page...** (Страница палитры компонентов) выберите страницу в палитре компонентов, на которой будет располагаться установленный компонент (по умолчанию это **ActiveX**). В разделе **Unit Dir Name...** (Имя каталога модуля) — путь к папке, куда будет установлен компонент. (Здесь проще соглашаться со всем, что предлагается.)

Нажмите кнопку **Install** (Установить). Перед вами появится окно, в котором вам нужно выбрать, в какой пакет будет установлен компонент. Вы можете установить как в уже существующий, так и в новый пакет. Затем перед вами появится окно редактирования выбранного пакета и Delphi вас спросит: "...Package will be rebuilt. Continue?" Нажмите кнопку **Yes**. Все готово, теперь можно использовать Flash в наших приложениях.

Новый модуль ShockwaveFlashObjects_TLB.dcu и соответствующий ему файл ShockwaveFlashObjects_TLB.pas появятся в подкаталоге Delphi Imports. Этот

модуль будет нужен при создании приложения, поэтому файл ShockwaveFlashObjects_TLB.dcu должен быть в одном из каталогов, где Delphi хранит библиотечные файлы, чтобы Delphi его нашла при компиляции проекта. Проще всего поместить файл ShockwaveFlashObjects_TLB.pas в каталог вашего текущего проекта.

4.3. Создаем форму и код программы

Итак, создайте каталог для вашего проекта на Delphi, поместите в него файл ShockwaveFlashObjects_TLB.pas, а также файл calculator.swf, который получается трансляцией файла calculator.fla, для чего в среде Flash надо нажать комбинацию клавиш <Shift>+<F12>. Запустите Delphi и выберите в меню **File | New | Application** (Файл | Создать | Приложение). Delphi создаст форму под названием **Form1** для нового приложения.

У нас на этой форме не будет ничего кроме объекта управления **Shockwave Flash**. Для того чтобы перетянуть его значок на форму, прокручивайте палитру компонентов, пока не появится вкладка **ActiveX**. На этой вкладке установите указатель на кнопке **Shockwave Flash** и утопите ее, щелкнув на ней левой кнопкой мыши. Затем щелкните мышью где-нибудь на форме `fmMain` (рис. 4.9).

Запишите в **Object Inspector** (Инспектор объектов) в ее свойстве **Caption** (Заголовок) имя нашей программы `Calculator`. В поле **Name** введите `fmMain`, в поле **BorderStyle** введите `bsSingle`, чтобы нельзя было растягивать форму за края. Для отмены распахивания окна на весь экран раскройте ветку **BorderIcons** и установите **biMaximize** в `False`. В **Object Inspector** (Инспекторе объектов), который вызывается клавишей <F11>, откроется вкладка **Properties** (Свойства) со свойствами объекта **ShockwaveFlash1**. Переименуйте его, написав в поле **Name** `calc`. После этого задайте для объекта `calc` такие свойства: **Top** и **Left** установите в ноль, **Width** установите в 460, **Height** в 370. В свойствах формы введите следующие значения: **ClientHeight** 370, **ClientWidth** 460. Позицию окна **Position** определим как **poScreenCenter**. Пора сохранить наш проект. Выберите меню **File | Save All** (Файл | Сохранить все) и найдите каталог, где у нас лежат уже два файла к нашему проекту. При сохранении самого файла проекта переименуйте его из Project1.dpr в calculator.dpr.

Продолжим установку свойств окна с игрой. В свойстве **Movie** задайте полный путь и имя файла calculator.swf, иначе Delphi при трансляции проекта его не найдет. Теперь можно запустить проект на выполнение. Для этого нажмите клавишу <F9>. Не знаю, как у читателя, а у меня работает.

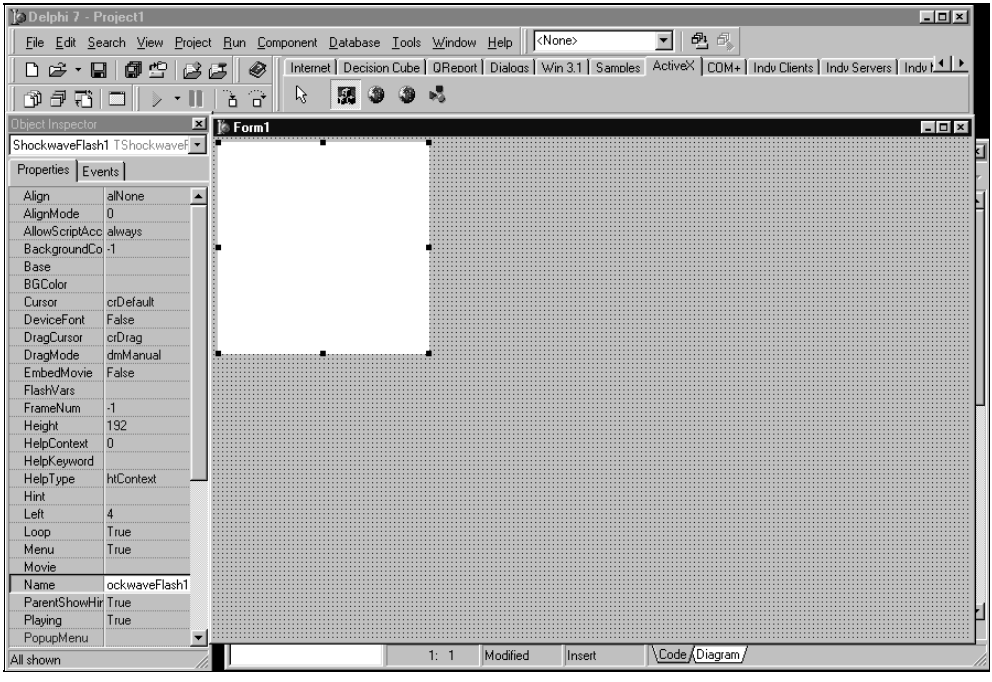


Рис. 4.9. Установка компонента **Shockwave Flash**

4.3.1. Отменяем показ меню и реакцию кнопок на нажатие клавиши <Tab>

Вы заметили, что при нажатии правой кнопки мыши на работающей игре появляется меню проигрывателя Flash? Оно помогает увеличению популярности этого продукта и компании Macromedia, используя продукты разработчиков. Сейчас мы его уберем, и наша игра будет выглядеть как обычное приложение Windows.

На вкладке свойств объекта `calc` есть элементы **Menu** и **Loop**. Можете установить их в `False`. Кстати, наша игра начинает свою работу сразу после загрузки потому, что свойство **Playing** установлено в `True`. Иначе программа должна была бы запускать игру методом `Play` объекта `calc`. Но нас и это вполне устраивает. Есть еще свойство **DeviceFont**. Я не знаю, оказывает ли оно влияние на проигрыватель Flash, ведь такой флажок имеется в свойствах текста во Flash. Но если у нас в игре шрифты берутся с компьютера игрока, то можно установить его в `True`. В **Quality2** должно быть задано **High**, иначе не будет антиалиасинга (сглаживания), хотя нагрузка на центральный

процессор снизится. Качество воспроизведения можно менять во время работы ролика, используя глобальное свойство в ActionScript `_quality`. Оно может иметь строковые значения `'LOW'`, `'MEDIUM'`, `'HIGH'` и `'BEST'`. Стандартное значение — `'HIGH'`.

Однако вы заметили, что меню проигрывателя Flash только уменьшилось, но не исчезло? Для полного его исчезновения найдите на вкладке **Additional** компонентов Delphi компонент **ApplicationEvents**. Он имеет значок факела с исходящими от него тремя стрелками. Перетяните его куда-нибудь на форму. Это невизуальный компонент, поэтому на этапе работы его не будет видно. В окне инспектора объектов на его вкладке **Properties** найдите свойство **onMessage** и сделайте двойной щелчок на поле ввода справа от него. В нем возникнет слово `ApplicationEvents1Message`, а в тексте модуля Unit1.pas появится процедура

```
procedure TfmMain.ApplicationEvents1Message(var Msg: tagMSG;
  var Handled: Boolean);
```

в ее тело введите строку

```
if msg.message = WM_RBUTTONDOWN then Handled:=TRUE;
```

Таким путем мы перехватываем сообщение о нажатии правой кнопки мыши и извещаем диспетчер сообщений Delphi, что оно уже обработано и его не надо передавать в проигрыватель Flash.

Еще раз запустите игру и нажимайте клавишу <Tab>. Вы увидите, что происходит перемещение желтого прямоугольника по всем кнопкам игры. Этот прямоугольник показывает кнопку, находящуюся в фокусе ввода с клавиатуры. Если нажать клавишу <Enter>, то это будет эквивалентно щелчку мышью над активной кнопкой.

Объект **Button** и свойство `_focusrect` появились в шестой версии Flash, поэтому в редакторе Flash мы этот желтый прямоугольник отменить не сможем. Но мы можем перехватить нажатия на клавишу <Tab>. Эта клавиша приравнена к "системным", таким как <Alt> и <F10>, поэтому она не перехватывается установкой свойства формы **KeyPreview** в `True` и перехватом события формы **OnKeyDown**. Вместо этого надо опять воспользоваться вышеприведенным методом и добавить в него оператор

```
if (msg.message = WM_KEYDOWN) and (msg.wparam = vk_TAB) then
 Handled:=TRUE;
```

Как вы помните, `wparam` хранит код нажатой клавиши, а `vk_TAB` — это виртуальный код клавиши <Tab> или просто число 9.

4.3.2. Присоединяем swf-файл к программе

В Delphi 7 появилась приятная возможность: можно включить этот swf-файл в состав exe-файла программы и иметь всего лишь один исполняемый файл. В Delphi 6 это свойство хотя и было, но не работало. Я имею в виду свойство **EmbedMovie** у объекта `calc`. Установите его в `True` и оттранслируйте программу. Вы увидите, что размер exe-файла увеличился примерно на длину swf-файла, и теперь нашей программе не требуется файл calculator.swf.

4.3.3. Применяем функцию *fscommand*

Среди глобальных функций Flash есть функция `fscommand`. Она связывает Flash-фильм с его хозяином. С ее помощью можно, к примеру, запретить проигрывателю Flash масштабирование фильма и попросить его выводить свое меню в минимальном виде. В браузере она передает команды языку сценариев, который работает на этой странице браузера. В нашем случае эта команда передает два своих параметра в программу Delphi, которая запустила этот фильм. Эти параметры — две строки в Unicode, в Delphi им соответствует строка типа `WideString` (широкая строка), все символы которой имеют размер в два байта. Но Delphi в смешанных выражениях с обоими типами строк молча преобразует обычные строки в широкие, так что у программиста здесь нет особых забот.

Помните, мы делали в игре кнопку **Exit**? Сейчас она у нас заработает. Загрузите проект calculator.fla, сделайте двойной щелчок в библиотеке на клипе `buttons`, который содержит эту кнопку, выделите ее и присоедините к ней такой код:

```
on(release) { _root.exitApp() }
```

В текст основной программы вставьте функцию `exitApp`:

```
function exitApp() { fscommand('exit','') }
```

Она будет передавать строку `exit` в нашу Delphi-программу. Второй параметр здесь не нужен, поэтому он пустой. Здесь как бы считается, что первый параметр — это команда, а второй — ее параметры. Пересоздайте swf-файл и скопируйте его в каталог проекта Delphi.

Чтобы получить эту команду, вернитесь в среду разработки Delphi и в окне инспектора объектов взгляните на вкладку **Events** объекта `calc`. На ней присутствует событие **onFSCommand**. Сделайте двойной щелчок на поле ввода справа от него и получите заготовку процедуры

```
procedure TfmMain.calcFSCommand(ASender: TObject; const command,
  args: WideString);
```

Среди ее параметров мы видим эти две широкие строки.

В теле этой процедуры запишите такой оператор:

```
if command = 'exit' then close;
```

Запустите программу на выполнение и щелкните на кнопке **Exit**. Окно игры тотчас исчезнет, словно его и не было. Вот он, этот метод получения информации от игры: в методе `TfmMain.calcFSCommand` надо сделать ветвление с помощью условных операторов по строке `command`.

4.3.4. Передаем в программу Delphi параметры для *IsWin*

Для задействования Delphi-функции `isWin`, которая будет искать выигрыш, надо передать в программу Delphi:

- массив `tabl`;
- число на индикаторе `ind`;
- признак `row_col`, равный:
  - `-1`, если это первый ход;
  - `0`, если надо перебирать ходы в строке;
  - `1`, если надо смотреть ходы в столбце;
- номер строки/столбца `num`, в котором надо перебирать ходы (если `row_col` не равно `-1`);
- уровень игры `level`.

Чтобы передать все элементы массива в строке, у массива существует метод `join`. С его помощью мы преобразуем все элементы массива `tabl` по порядку в строку, а в качестве разделителя между ними будет запятая. Аналогично добавим к результирующей строке остальные три переменные. Строку с командой назовем `iswin`.

Для этого в функцию `compMove` после строк

```
function compMove(row_col,num,waiting)
{ var i,j;


  if (waiting)
   { wait.row_col=row_col;
     wait.num=num;
     wait.play();
     return;
   }
```

вставим строку

```
fscommand('iswin',tabl.join()+','+ind+','+row_col+','+num+','+level);
```

В программе на Delphi добавим оператор к процедуре `TfmMain.calcFSCommand`, которая теперь будет выглядеть так:

```
procedure TfmMain.calcFSCommand(ASender: TObject; const command,
  args: WideString);
begin
 if command = 'exit' then close;
 if command = 'iswin' then
   begin
     ShowMessage(args);
   end;
end;
```

Если запустить программу с обновленным swf-файлом, то при каждом ходе компьютера в программе на Delphi мы будем наблюдать в окне все цифры на кнопках, число на индикаторе, признак хода и номер ряда, в котором мог делать ход компьютер. Обратите внимание, что если щелкнуть по кнопке **Comp. begins**, то вместо переменной `num` выводится пустая строка. Это потому, что в программу `compMove` она в этом случае не передается и, значит, имеет значение `undefined`, которое преобразуется в пустую строку.

Нам остается заменить вызов `ShowMessage(args);` на просчет вариантов и отправку во Flash-игру результата (строка и столбец с ходом и признак, выигрышный он или нет). Для этого из уже написанной в Delphi игры Calculator возьмем часть кода, переделаем его для наших нужд и вставим в оболочку.

4.3.5. Пишем функцию *IsWin* и остальной код Unit1.pas

Перенесем из игры Calculator, которую мы создали в *главе 2*, константы и переменные в модуль Unit1.pas:

```
const
  DEPTHS: array[0..8] of byte=(2,3,4,5,6,7,8,9,99);
  INTOSTR=0;
  INTOCOL=1;
  RESLOST=0;
  RESWON=1;
  RESNOTKNOWN=2;
```

```
var
  fmMain: TfmMain;
  depth0,depth: byte;
  indicator,row_col,row_col1,num,level: integer;
  tabl: array[0..2,0..2] of byte;
```

Нам еще понадобятся две вспомогательных функции: `Min3` для вычисления номера наименьшего из трех переданных функции чисел и `IntFromStr` для извлечения числа из строки и перевод его в двоичный вид. Первая функция поможет нам выбирать минимальное число в ряду кнопок, когда все ходы проигрывают, а вторая будет извлекать числа из вызова `fscommand`. Широкую строку `args` мы преобразуем к обычной строке типа `String` присваиванием `str:=args`, где `str` — обычная строка. После этого будем извлекать из нее аргументы.

Программа может читать и записывать значения переменных, существующих на уровне `_root` (в главной временной диаграмме). Для этого используются методы объекта **Shockwave Flash** `GetVariable` и `SetVariable`. `SetVariable` имеет два параметра типа `WideString`: первый — это имя переменной, а второй — ее значение. В файле ShockwaveFlashObjects_TLB.pas есть интерфейс для всех методов этого объекта, но не следует думать, что все они работают.

Оболочка на Delphi, просчитав варианты, запишет в переменные Flash-игры следующие значения:

- в переменную `drow` — номер строки (`0`, `1`, `2`) для совершения хода;
- в переменную `dcol` — номер столбца (`0`, `1`, `2`) для совершения хода;
- в переменную `dres` — результат этого хода:
  - `0` — компьютер проиграл;
  - `1` — компьютер выиграл;
  - `2` — результат игры неизвестен компьютеру.

Эти три переменные нам еще предстоит создать в основной Flash-программе и получить в них результат. А в листинге 4.4 представлен заключительный код модуля Unit1.pas.

Листинг 4.4. Заключительный код модуля Unit1.pas

```
{$A-,B-,H+}
unit Unit1;


interface
```

```
uses
  Windows, Messages, SysUtils, Variants, Classes,Graphics, Controls,
  Forms, Dialogs, OleCtrls, ShockwaveFlashObjects_TLB, AppEvnts;

type
  TfmMain = class(TForm)
  calc: TShockwaveFlash;
    ApplicationEvents1: TApplicationEvents;
    procedure calcFSCommand(ASender: TObject; const command,
      args: WideString);
    procedure ApplicationEvents1Message(var Msg: tagMSG;
      var Handled: Boolean);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

const
  DEPTHS: array[0..8] of byte=(2,3,4,5,6,7,8,9,99);
  INTOSTR=0;
  INTOCOL=1;
  RESLOST=0;
  RESWON=1;
  RESNOTKNOWN=2;

var
  fmMain: TfmMain;
  depth0,depth: byte;
  indicator,row_col,row_col1,num,level: integer;
  tabl: array[0..2,0..2] of byte;

implementation

{$R *.dfm}
```

```
{ Выдает номер (0, 1 или 2) наименьшего из заданных трех
  различных чисел }
function Min3(a,b,c: byte): integer;
begin
      if a < b then
       begin
       if a < c then Result:=0 else Result:=2;
       end else
        if b < c then Result:=1 else Result:=2;
end; {Min3}

{ Извлечение из строки s целого числа и продвижение позиции pos
  через один символ после этого числа }
function IntFromStr(var s:string; var pos: integer): integer;
label EX;
var
 l,sign,res: integer;
 ch: char;

begin
       l:=Length(s);
       // Знак числа пока плюс
       sign:=1;
       // Здесь будет само число
       res:=0;
       // Вышли за строку?
       if pos > l then goto EX;
       while (pos <= l) and (s[pos] in ['-','0'..'9']) do
        begin
        ch:=s[pos];
        if ch = '-' then
         begin
         // Знак уже был прочитан?
         if sign = -1 then break;
         sign:=-1;
         end
```

```
          // Добавляем к результату очередную цифру
          else res:=res*10+byte(ch)-byte('0');
        Inc(pos);
        end;
       // Учитываем знак
       res:=res*sign;
       Inc(pos);
EX:    Result:=res;
end; {IntFromStr}

{ Выяснение результата хода strnum, colnum. Если row_col=1, то игрок,
  делающий этот ход, ходит в строке, если 0 - в столбце. Результат:
  0 - проигрыш, 1 - выигрыш, 2 - неизвестно. }
function IsWin(indicator,strnum,colnum,row_col: integer): byte;
label
 WON,LOST,NOTKNOWN,EX;
var
 i,newindicator,tmpres,res: integer;

begin
      // Ограничиваем глубину рекурсии
      Inc(depth);
      if depth > depth0 then goto NOTKNOWN;
      // Делаем заданный ход
      newindicator:=indicator-tabl[strnum,colnum];
      if newindicator < 0 then goto LOST;
      if newindicator = 0 then goto WON;
      // Результат этого хода неизвестен. Перебираем ходы за соперника
      res:=RESLOST;
      for i:=0 to 2 do
       begin
       if row_col = INTOSTR then
        // Перебираем ходы в столбце colnum
        tmpres:=IsWin(newindicator,i,colnum,INTOCOL)
        else
```

```
          // Перебираем ходы в строке strnum
          tmpres:=IsWin(newindicator,strnum,i,INTOSTR);
        if tmpres = RESWON then goto LOST;
        if tmpres = RESNOTKNOWN then res:=RESNOTKNOWN;
        end;
      if res = RESNOTKNOWN then goto NOTKNOWN;
WON:  IsWin:=RESWON;
      goto EX;
NOTKNOWN:
      IsWin:=RESNOTKNOWN;
      goto EX;
LOST: IsWin:=RESLOST;
EX:   Dec (depth);
end; {IsWin}

// Обработка команд, полученных от fscommand
procedure TfmMain.calcFSCommand(ASender: TObject; const command,
  args: WideString);
var
 i,j,pos,row,col,row1,col1,row2,col2: integer;
 res: byte;
 str: string;

begin
if command = 'exit' then close;

if command = 'iswin' then
 // Получаем параметры
 begin
 str:=args;
 pos:=1;
 // Переводим параметры команды в числа
 for i:=0 to 8 do tabl[i div 3,i mod 3]:=IntFromStr(str,pos);
 indicator:=IntFromStr(str,pos);
 row_col:=IntFromStr(str,pos);
 num:=IntFromStr(str,pos);
 level:=IntFromStr(str,pos);
```

```
// Вычисляем depth0
depth0:=DEPTHS[level];
depth:=0;
// Перебор всех возможных ходов и вызов IsWin
{ Делаем параметр row_col1, т. к. IsWin не понимает значения
  row_col = -1 для первого хода компьютера }
row_col1:=row_col;
if row_col1 < 0 then row_col1:=1;
// Признак, что еще не найден выигрывающий ход
row1:=-1;
// Признак, что еще не найден ход с неизвестным результатом
row2:=-1;
// Перебор всех ходов
for i:=0 to 2 do
 for j:=0 to 2 do
  if (row_col = -1) or
   (row_col = 1) and (i = num) or (row_col = 0) and (j = num) then
   begin
   res:=IsWin(indicator,i,j,1-row_col1);
   if (res = 1) and ((row1 = -1) or (tabl[i,j] > tabl[row1,col1])) then
    // Запоминаем выигрышный ход в row1, col1 с наибольшей цифрой
    begin
    row1:=i;
    col1:=j;
    end;
   if (res = 2) and ((row2 = -1) or (tabl[i,j] < tabl[row2,col2])) then
    { Запоминаем ход с неизвестным результатом в row1, col1
     с наименьшей цифрой }
    begin
    row2:=i;
    col2:=j;
    end;
   end;

  if row1 > -1 then
   // Есть выигрышный ход
```

```
begin
// Переменные для передачи во Flash-игру
res:=1;
row:=row1;
col:=col1;
end
 else
 if row2 > -1 then
  // Есть ход с неизвестным результатом
  begin
  // Переменные для передачи во Flash-игру
  res:=2;
  row:=row2;
  col:=col2;
  end
   else
   if row1+row2 = -2 then
    // Все ходы проигрывают. Делаем ход наименьшей цифрой
    begin
    res:=0;
    if row_col = -1 then
     // Ищем единицу во всем массиве tabl
     begin
     for i:=0 to 2 do
      for j:=0 to 2 do
       if tabl[i,j] = 1 then
        // Выдаем ход
        begin
        row:=i;
        col:=j;
        break;
        end;
     end
      else
      if row_col = 1 then
       // Ищем минимальное число в строке num
```

```
          begin
          row:=num;
          col:=Min3(tabl[num,0],tabl[num,1],tabl[num,2]);
          end
           else
            // Ищем минимальное число в столбце num
            begin
            col:=num;
            row:=Min3(tabl[0,num],tabl[1,num],tabl[2,num]);
            end
       end;

 { Записываем результат в переменные drow, dcol и dres Flash-игры.
   drow, dcol - номер строки и столбца с ходом, dres - результат
   за компьютер: 0 - проиграл, 1 - выиграл, 2 - неизвестно.
 }
 with calc do
  begin
  SetVariable('drow',IntToStr(row));
  SetVariable('dcol',IntToStr(col));
  SetVariable('dres',IntToStr(res));
  end;
 end;
end;

// Перехват и отмена правого щелчка и нажатия клавиши <Tab>
procedure TfmMain.ApplicationEvents1Message(var Msg: tagMSG;
  var Handled: Boolean);
begin
if msg.message = WM_RBUTTONDOWN then Handled:=True;
if (msg.message = WM_KEYDOWN) and (msg.wparam = vk_TAB) then
 Handled:=TRUE;
end;

end.
```

4.3.6. Получаем результат хода во Flash-игре

Вернемся в среду разработки Flash. Нам надо добавить в секцию `var` основной программы три переменные: `dres`, `drow` и `dcol`. Для получения результата следует учитывать, что установка переменных из программы Delphi происходит не мгновенно после вызова `fscommand`, а через два такта работы проигрывателя Flash, т. е. через два события перехода проигрывающей головки в следующий кадр. Поэтому мы не сможем получить результат в этой же функции `compMove`, и нам придется опять прибегнуть к ухищрению, как мы это делали с задержкой и клипом `wait`. Для этого перед вызовом `fscommand` вставим оператор присваивания

```
drow=-1;
```

Значение `-1` будет означать, что оболочка Delphi еще не присвоила этой переменной значения. А когда `drow` приобретет другое значение, то значит, программа на Delphi вернула результат счета и можно делать этот ход.

Из функции `compMove` удаляем код для выполнения хода и переносим его в новую функцию `doCompMove`, которая будет вызываться после получения результата от Delphi. Этой функции нужна будет переменная `row_col` из функции `compMove`, поэтому после вызова `fscommand` продублируем ее в глобальной переменной с тем же именем:

```
_root.row_col=row_col;
```

Создадим пустой клип с именем `delphi` по примеру клипа `wait`. В первом его кадре ставим команду

```
stop();
```

В третьем кадре вставим цикл ожидания получения результатов хода:

```
if (_root.drow == -1) gotoAndPlay(2);
```

А в четвертом кадре напишем вызов функции `doCompMove`:

```
_root.doCompMove();
```

После этого разместим клип `delphi` в том же слое **Layer 3** и первом кадре, что и клип `wait`. Выделим этот клип инструментом **Selection** и в окне его свойств дадим имя экземпляру `delphi`, чтобы он вызывался из функции `compMove`. А в конце этой функции вставляем оператор

```
delphi.play();
```

Во время своей игры этот клип подождет, когда будет получен ответ от кода Delphi, и вызовет функцию `doCompMove`, после чего вернется в исходное состояние.

В листинге 4.5 вы видите заключительный код основной программы.

Листинг 4.5. Заключительный код основной программы Flash-игры

```
var
 // Уровень игры, считая с 0 по 8
 level=2,
 // Признак, что начал игру компьютер
 compbegins,
 // Признак, что сделан первый ход
 gamestarted,
 // Признак, что игра закончена
 gameended,
 // Текущее значение на индикаторе
 ind,
 // Признак, что сейчас очередь хода человека
 isplayersmove,
 // Номер ряда, в котором игрок может делать ход
 numline,
 // Массив цифр, написанных на кнопках
 tabl=[[0,0,0],[0,0,0],[0,0,0]],
 // Звуки ходов
 humansnd=new Sound(),
 compsnd=new Sound(),
 row_col,
 dres,
 drow,
 dcol;

if (!_framesloaded || _framesloaded != _totalframes) gotoAndPlay(1); else
 { // Прикрепляем звуки и начинаем игру
   humansnd.attachSound('humanmove');
   compsnd.attachSound('compmove');
   newGame();
 }

// Игрок щелкнул по кнопке Exit
function exitApp() { fscommand('exit','') }
```

```
// Щелчок на кнопке калькулятора в строке y и столбце x
function click(y,x)
{ if (!isplayersmove || gameended ||
   gamestarted && (compbegins && numline != x || !compbegins &&
   numline != y)) return;
  // Выполняем ход x, y
  indicator.ind=ind-=tabl[y][x];
  humansnd.start();
  if (ind == 0)
   // Игрок выиграл
   { gameended=1;
     showYouWon(1);
     return;
   }
  if (ind < 0)
   // Игрок проиграл
   { gameended=1;
     showYouLost(1);
     return;
   }
  gamestarted=1;
  isplayersmove=0;
  // Ход компьютера в строке y или столбце x
  if (compbegins) compMove(1,y,1); else compMove(0,x,1);
}

// Прячем при arg == 0 или показываем при arg == 1 кнопки калькулятора
function toggleButtons(arg)
{ buttons._visible=arg;
  for (i=0; i < 3; i++)
   for (j=0; j < 3; j++)
    eval('bt'+i+j)._visible=arg;
}

// Начало новой игры
function newGame()
```

```
{ var
   d=[1,2,3,4,5,6,7,8,9],
   i,
   j,
   temp;

  // Устанавливаем уровень игры в текстовом поле
  leveltext=level+1;
  // Признак, что начал игру компьютер/сделан первый ход/игра закончена
  compbegins=gamestarted=gameended=0;
  // Разрешаем игроку ходить первым
  isplayersmove=1;
  // Убираем сообщения
  showYouWon(0);
  showYouLost(0);
  showGameOver(0);
  // Перемешиваем массив d
  for (i=0; i < 9; i++)
   { // Получаем случайное целое число из диапазона от 0 до 8
     j=Math.floor(Math.random()*9);
     temp=d[i];
     d[i]=d[j];
     d[j]=temp;
   }
  // Размещаем случайные числа на кнопках
  for (i=0; i < 3; i++)
   for (j=0; j < 3; j++)
    eval('bt'+i+j).digit=tabl[i][j]=d[i*3+j];
  // И на индикаторе
  ind=Math.floor(Math.random()*6)+30;
  indicator.ind=ind;
  // Убираем красные цифры
  markRed();
}

// Прячет или показывает правила игры и все кнопки
function toggleHelp()
```

```
{ if (toggleHelp.ishelp == 1)
   { help.removeMovieClip()
     toggleButtons(1);
     toggleHelp.ishelp=0;
   } else
    { attachMovie('help','help',5);
      help._x=230;
      help._y=185;
      toggleButtons(0);
      toggleHelp.ishelp=1;
    }
}

// Отмечаем красным ряд номер num. Если row_col == 1, то это строка,
// если row_col == 0, то столбец
function markRed(row_col,num)
{  for (var i=0; i < 3; i++)
    for (var j=0; j < 3; j++)
     { eval('bt'+i+j).digit1='';
       if (row_col == 1 && i == num) eval('bt'+i+j).digit1=tabl[i][j];
       if (row_col == 0 && j == num) eval('bt'+i+j).digit1=tabl[i][j];
     }
}

// Ход компьютера. Если row_col == -1, то это первый ход, если
// 1, то в строке num, если 1, то в столбце num
function compMove(row_col,num,waiting)
{ if (waiting)
   { wait.row_col=row_col;
     wait.num=num;
     wait.play();
     return;
   }
  // Признак, что из Delphi еще не получен результат
  drow=-1;
```

```
  // Отправляет массив tabl, число на индикаторе,
  // -1 - это первый ход,
  // 0  - ход в столбце,
  // 1  - ход в строке.
  // num - номер строки/столбца, где надо перебрать ходы
  fscommand('iswin',tabl.join()+','+ind+','+row_col+','+num+','+level);
  // Ожидаем получение ответа от Delphi-программы
  _root.row_col=row_col;
  delphi.play();
}

// Выполнение хода за компьютер. Вызывается из клипа delphi, в глобальных
// переменных dres, drow, dcol результат просчета программы на Delphi
function doCompMove()
{ // Если в результате хода от Delphi на индикаторе получается
  // отрицательное число, то не делаем этот ход. Заодно в этой проверке
  // присваиваем новое значение переменной ind
  if ((ind-=tabl[drow][dcol]) < 0)
   { showYouWon(1);
     // Убираем подсветку цифр красным цветом
     markRed();
     gameended=1;
     return;
   }
  // Обновляем индикатор
  indicator.ind=ind;
  if (row_col == -1)
   // Это первый ход компьютера
   { showHint(drow,dcol);
     markRed(0,dcol);
     numline=dcol;
     // Если игрок проиграл, заранее сообщаем это, чтобы не мучился
     if (dres == 1) showYouLost(1);
   } else
     if (!row_col)
```

```
      // Это ход в столбце
      { showHint(drow,dcol);
        markRed(1,drow);
        numline=drow;
        // Если игрок проиграл, заранее сообщаем это...
        if (dres == 1) showYouLost(1);
      } else
       // Это ход в строке
       { showHint(drow,dcol);
         markRed(0,dcol);
         numline=dcol;
         // Если игрок проиграл, заранее сообщаем это...
         if (dres == 1) showYouLost(1);
       }
  gamestarted=1;
  isplayersmove=1;
}

// Щелчок по кнопке Comp. begins
function btCompClick()
{ if (gamestarted || gameended) return;
  compbegins=1;
  compMove(-1);
}

// Щелчок по кнопке Level
function btLevelClick()
{ if (!gamestarted || gameended)
   { level=(level+1) % 9;
     leveltext=level+1;
   }
}

// Показ или удаление надписи "You won"
function showYouWon(arg)
{ if (!arg) youwon.removeMovieClip(); else
```

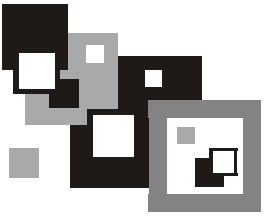
```
   { attachMovie('youwon','youwon',2);
     youwon._x=25;
     youwon._y=330;
     showGameOver(1);
   }
}

// Показ или удаление надписи "You lost"
function showYouLost(arg)
{ if (!arg) youlost.removeMovieClip(); else
   { attachMovie('youlost','youlost',2);
     youlost._x=25;
     youlost._y=330;
     if (ind < 0) showGameOver(1);
   }
}

// Показ или удаление надписи "Game over"
function showGameOver(arg)
{ if (!arg) gameover.removeMovieClip(); else
   { attachMovie('gameover','gameover',3);
     gameover._x=293;
     gameover._y=330;
   }
}

// Показ анимации над нажатой компьютером кнопкой калькулятора
function showHint(i,j)
{ attachMovie('hint','hint',4);
  hint._x=55+70*j;
  hint._y=145+70*i;
  compsnd.start();
}

stop();
```

Все исходные и результирующие файлы присутствуют на компакт-диске в каталоге chapter4.

Замечание

Исходные и результирующие файлы для игры Calculator размещены на компакт-диске исключительно с целью обучения и не могут быть использованы в любых иных целях. Вы не имеете права публиковать где бы то ни было и в любой форме как исходные коды этой программы, так и результирующие swf- и exe-файлы. Также вы не имеете права использовать принадлежащую мне, как автору, идею этой игры.

Глава 5



Применение PCRE в Delphi

- Мощь языка PCRE
- Синтаксис PCRE
- Применение этого языка в ваших проектах Delphi
- Вычисление арифметических выражений по LL(1)-грамматике

Эта глава стоит несколько особняком от остального содержания книги. Но не все же играть? Пора заняться и серьезными делами.

Что это за штука такая — PCRE, и чем она может быть нам полезна? PCRE — это Perl Compatible Regular Expressions (регулярные выражения, совместимые с языком программирования Perl). Регулярные — значит, составленные по правилам. В UNIX эти регулярные выражения являются частью системы, и их можно применять в командах оператора. В фирме Microsoft работает группа по внедрению аналогичной практики в систему Windows, но ориентация этой системы на массового потребителя не дает сделать это так легко.

Все мы сталкивались с шаблонами файлов еще со времен DOS. Попробуйте без их помощи удалить все файлы с расширением bak с диска. А с применением шаблона запросто: `del *.bak /s`. В этих файловых шаблонах звездочка обозначает любое число символов имени, а знак вопроса — ровно один любой такой символ. А если надо удалить файлы по более сложным соответствиям? Тут системы DOS/Windows пасуют: ищите и удаляйте вручную или пользуйтесь соответствующими утилитами. Но чаще требуется не удалять файлы по шаблону, а искать файлы, содержащие в себе последовательность символов, соответствующую шаблону, а также заменять найденные последовательности, возможно, в зависимости от их содержимого. Часто такие задачи возникают у Web-мастеров. Например, придумали в Google параметр тега `a rel=nofollow`. Web-мастеру требуется вставить этот параметр во все или выбранные файлы в конец тега `a`. Или у выбранных файлов,

да еще и в отдельных местах. Это уже работа не для рук. И строковые функции Delphi здесь не сильно помогут. То, что можно сделать на PCRE за полчаса, на Delphi вы будете кодировать неделю и потом еще полгода вылавливать ошибки. Именно для облегчения поиска и замены в тексте и были придуманы эти регулярные выражения. В приличных текстовых редакторах, в том числе в редакторе Delphi, присутствуют те или иные диалекты регулярных выражений. Кроме того, эти выражения используются в таких языках, как Perl, PHP, Java, JavaScript, Python, Ruby, Tcl, VB.NET...

5.1. Синтаксис языка PCRE

Части текста, которые надо найти и, возможно, заменить, идентифицируются путем составления шаблона для этих образцов. *Шаблон* — это обычная строка, которая содержит конструкцию, соответствующую всем разновидностям искомого текста. Например, если вы ищете номера телефонов, то код города может быть взят в скобки или указан без них, может быть отделен рядом пробелов или даже переводом строки. Для PCRE это не проблема. Для этого в нем имеются соответствующие выражения.

Прежде всего, надо уметь разделять строки. Как мы знаем, в Windows строка заканчивается двумя символами с кодами 13 (carriage return, возврат каретки) и 10 (new line, новая строка) в десятичной системе. В UNIX используется лишь символ 10 или `\n` (символ с кодом 13 обозначается `\r`). Обратная косая черта (обратный слэш) используется, чтобы показать: либо это двухсимвольная Escape-последовательность, либо следующий метасимвол является обычным символом. В данном случае обратный слэш показывает, что `n` не представляет из себя буквы `n`, `\n` надо рассматривать как единый символ. Просто дело в том, что на клавиатуре нет столько спецсимволов, чтобы можно было их использовать поодиночке. В конце текста может и не стоять символ `\n`. В PCRE истинный конец текста обозначается через псевдосимвол `\z`. Псевдосимволом он называется потому, что имеет нулевую длину (в отличие от `\n`). Псевдосимвол `\Z` соответствует позиции в истинном конце текста, а также перед символом `\n`, который является последним в тексте. Метасимвол `$` также соответствует концу строки и концу текста, а именно, он совпадает перед каждым символом `\n` (при определенных условиях), а также в истинном конце текста и перед символом `\n`, который является последним в тексте. Символ `\A` соответствует началу текста. Метасимвол `^` соответствует началу строки, это начало текста, а также все позиции сразу после символа `\n` (при определенном условии). Все эти символы совпадают не с текстом, а с позицией в тексте.

Описывать синтаксис регулярных выражений непросто, потому что смысл некоторых символов зависит от контекста. Например, в одном случае `^`

является одним метасимволом, в других — другим, а в третьих — обычным символом. Поэтому мы не будем лезть в дебри кодировки UTF-8, которую поддерживает библиотека PCRE, чтобы не нагружаться ненужной информацией.

5.1.1. Модификаторы регулярных выражений

Введем определение модификаторов регулярного выражения. В Delphi-программе их можно задавать операторами присваивания, но нагляднее вставлять их в сам шаблон. Это следующие буквы:

- `i` — означает, что ищется совпадение без различия строчных и прописных букв (case Insensitive). При этом учитываются также буквы национального алфавита, в русифицированной версии Windows это будут буквы от А до Я и от а до я;
- `m` — multiline (многострочный текст). В этом режиме метасимвол `^` соответствует началу каждой строки, а не только всего текста, как метасимвол `\A`. То есть он совпадает также перед всеми символами `\n`. Метасимвол `$` также совпадает перед каждым символом `\n`, а не только так, как совпадает метасимвол `\Z`;
- `s` — метасимвол . (точка) также совпадает со всеми без исключения символами текста, иначе она совпадает со всеми символами кроме символа конца строки `\n`;
- `x` — extended (расширенный режим). Позволяет делать комментарии внутри шаблона. В этом режиме истинные пробельные символы в шаблоне (т. е. натуральные символы, что записаны в файле, а не Escape-последовательности `\n`, `\r` и т. д.) игнорируются, благодаря чему возможно форматирование и запись шаблона в несколько строк.

Вводятся модификаторы таким путем:

- `(?i)` — игнорирование регистра букв;
- `(?im)` — игнорирование регистра и режим многострочного текста;
- `(?i-sm)` — игнорирование регистра и отмена режимов `s` и `m`.

Область действия этих выражений начинается сразу после их закрывающей скобки и заканчивается следующей закрывающей группирующей скобкой в шаблоне, причем такой, соответствующая открывающая скобка которой стоит левее этого выражения с модификаторами. Например, `(a(?i)b)c` соответствует `abc` и `aBc` и ничему больше.

Если конструкция `(?i)` стоит вне группирующих скобок, то она действует на всю часть шаблона, находящуюся справа от нее.

5.1.2. Квантификаторы в регулярных выражениях

Часто надо повторить какой-то символ или подвыражение в шаблоне. Для этого служат *квантификаторы* (числители). Они записываются в фигурных скобках. Например:

- `a{3}` означает, что символ `a` должен встретиться три раза, то же, что `aaa`;
- `a{0,3}` означает, что символ `a` должен встретиться от нуля до трех раз;
- `ab{1,5}` означает букву `a`, за которой идет от одной до пяти букв `b`;
- `(ab){1,5}` означает, что пара `ab` записана от одного до пяти раз подряд.

Если левое число отсутствует, то вообще-то считается, что на его месте стоит `0`. Но в рассматриваемой версии PCRE эта конструкция не работает должным образом, поэтому не применяйте ее. Если отсутствует правое число, то считается, что на его месте стоит бесконечно большое число. (На самом деле есть ограничение реализацией в 65 535 символов максимум.)

Некоторые квантификаторы встречаются так часто, что для них ввели специальные метасимволы:

- `a?` означает ноль символов или один символ `a`. То же, что `a{0,1}`;
- `a*` означает ноль или больше символов `a`. То же, что `a{0,}`;
- `a+` означает один или больше символов a. То же, что `a{1,}`.

5.1.3. Символьные классы

Предположим, вы ищете в тексте даты с 2006 по 2008 годы. В соответствующем шаблоне на четвертом месте должна стоять конструкция, которая совпадает с цифрами 6, 7 и 8. Для таких вещей создали классы символов, которые записываются так же, как и множества языка Pascal: в квадратных скобках. Искомый шаблон будет таким:

```
200[678]
```

Класс символов обязательно должен совпадать с каким-либо символом, нельзя создавать пустые классы, которые совпадают лишь с пустой подстрокой. Это лежит на совести программиста, компилятор регулярных выражений не всегда может это проверить. Кроме того, класс символов совпадает ровно с одним символом в данной позиции строки, а именно, с любым из перечисленных в классе. Не надо писать класс типа `[<b>]` и ждать, что он совпадет с тегом `<b>` в тексте. На самом деле этот класс совпадает с одним из символов множества `<`, `b`, `>`.

Если диапазон символов велик, а их коды возрастают подряд, то можно воспользоваться конструкцией диапазона в символьном классе. Например, диапазон `a-z` соответствует всем строчным латинским буквам. Как видим,

минус является метасимволом внутри класса. Но если он стоит в самом начале, вторым после символа `^` или в самом конце класса, то он считается обычным символом, т. к. не может определять диапазон. В остальных случаях его внутри класса надо маскировать обратным слэшем так же, как и символ-ограничитель `]`.

Иногда легче перечислить символы, которые не входят в класс. В этом случае применяют инвертированные классы. Такие классы начинаются символом `^` и совпадают со всеми символами, которые в них не перечислены. Например, класс `[^a-z]` совпадает со всеми символами кроме строчных латинских букв.

В конце заметим, что модификатор `i` (нечувствительность к регистру букв) действует на символьные классы. Например, шаблон `(?i)[a]` соответствует как символу `a`, так и символу `A`.

В символьных классах работают сокращения для обозначения классов: `\d`, `\D`, `\w` и т. д., а также Escape-последовательности `\n`, `\r`, `\t`, ... конструкция `\Q...\E` внутри классов не распознается.

5.1.4. Альтернативные шаблоны

Предположим, вы ищете слова stop, quit и exit. Как составить шаблон, который соответствует всем этим словам? Для этого существует конструкция выбора, в которой *альтернативные шаблоны* разделяются вертикальной чертой `|`. В данном случае шаблон выглядит так:

```
stop|quit|exit
```

Заметьте, что вместо этих слов могут стоять любые регулярные выражения, в том числе со своей конструкцией выбора. Но тогда надо как-то отделять эти конструкции, чтобы интерпретатор знал, где они начинаются и заканчиваются. Разделителями являются круглые скобки. Кроме того, если альтернативный шаблон не является единственным, что есть в регулярном выражении, то его надо отделить от того, что было перед ним и идет после него, потому что операция альтернативы `|` имеет самый низкий приоритет. Например, шаблон

```
a(stop|quit|exit)b
```

совпадет со строками

```
astopb
aquitb
aexitb
```

и ни с чем иным. А шаблон

```
astop|quit|exitb
```

совпадает лишь со строками

```
astop
quit
exitb
```

Алгоритм работы альтернативы таков: подойдя к этой конструкции, механизм поиска совпадения начинает пробовать с текущей позиции текста каждый шаблон из альтернативы в порядке их написания (т. е. слева направо). Первый шаблон, совпавший с текстом с текущей позиции (или с текущей позицией текста), захватывает, сколько может, и механизм перемещается в шаблоне за эту конструкцию альтернативы. Если вы запишете такую альтернативу, как

```
a?|ab
```

то она будет соответствовать любому тексту, в том числе и пустой строке, потому что подшаблон

```
a?
```

совпадает со всем на свете! Поэтому будьте внимательны при выборе порядка следования альтернатив, он очень важен. Возможно, в этом шаблоне следовало бы поменять альтернативы местами (`ab|a?`). Тогда он бы совпадал с текстом

```
ab
```

буквой `a` или, на худой конец, с пустой подстрокой.

5.1.5. Метасимволы и специальные символы

В регулярных выражениях некоторые символы ASCII имеют значения метасимволов, но внутри классов на метасимволы распространяются свои правила, поэтому символьные классы надо рассматривать отдельно.

Вне классов следующие символы являются метасимволами:

- `\` — обратный слэш. Маскирует следующий символ, чтобы он не воспринимался метасимволом; или является первым символом в Escape-последовательности;
- `|` — вертикальная черта. Разделитель в альтернативных шаблонах;
- `()` — открывающая и закрывающая круглые скобки. Группируют подшаблоны, служат для захвата текста, совпавшего с подшаблоном в этих скобках;

- `[` — открывающая квадратная скобка. Определяет символьный класс;
- `{` — открывающая фигурная скобка. Определяет числитель для наименьшего возможного подшаблона, стоящего слева от скобки;
- `*` — звездочка. Является квантификатором "ноль или больше";
- `?` — вопрос. Является квантификатором "ноль или один", а также применяется в других случаях в выражениях вида `(?...)`;
- `^` — крышка. Символизирует начало строки;
- `$` — доллар. Совпадает с концом строки;
- `.` — точка. Совпадает с любым символом кроме символа новой строки `\n`. В зоне действия модификатора `s` совпадает со всеми без исключения символами.

Имеются следующие Escape-последовательности:

- `\037` — символ с данным восьмеричным кодом;
- `\a` — символ BEL, звонок, имеет код 7;
- `\c[` — управляющие символы (на клавиатуре это комбинация <Ctrl>+ +<*символ*>. В данном случае это символ ESC, его десятичный код — 27;
- `\d` — десятичная цифра, то же, что `[0-9]`;
- `\D` — любой символ кроме десятичной цифры, то же что `[^0-9]`;
- `\e` — символ ESC;
- `\E` — конец действия команды `\Q`;
- `\f` — перевод страницы, десятичный код 12;
- `\n` — символ новой строки, десятичный код 10;
- `\Q` — вплоть до команды `\E` или до конца регулярного выражения все метасимволы считаются обычными символами. Применяется для вставки ввода пользователя в регулярное выражение;
- `\r` — возврат каретки, десятичный код 13;
- `\s` — соответствует пробельным символам. То же, что `[\040\t\n\r\f]`;
- `\S` — любой непробельный символ, то же, что `[^\s]`;
- `\t` — горизонтальная табуляция, десятичный код 9;
- `\w` — алфавитно-цифровой символ: буква, цифра или знак подчеркивания. Также включает буквы национального алфавита;
- `\W` — то же, что и `[^\w]`;
- `\x1B` — символ с данным шестнадцатеричным кодом.

5.1.6. Мнимые символы (якоря или условия)

Приведем в порядок все *мнимые символы*. Их называют так за то, что они имеют нулевую длину и совпадают не с символами, а с позициями в заданном тексте.

- `^` — соответствует началу текста. С модификатором `m` соответствует началу каждой строки в тексте.
- `$` — совпадает с истинным концом текста, а также перед `\n`, стоящим в истинном конце текста, подобно символу `\Z` (истинность последнего утверждения я при проверке рассматриваемой далее версии PCRE не обнаружил). С модификатором `m` также совпадает с концом каждой строки (после символа `\n`).
- `\A` — совпадает с истинным началом текста.
- `\Z` — совпадает с истинным концом текста, а также перед символом `\n`, стоящим последним в тексте (истинность последнего утверждения я при проверке также не обнаружил).
- `\z` — совпадает с истинным концом текста.
- `\b` — совпадает с границей слова, т. е. между символами `\W\w` или `\w\W`, а также между началом текста и `\w` и после `\w`, стоящим в самом конце текста.
- `\B` — совпадает с отсутствием границы слова, т. е. между символами `\w\w`, `\W\W`, `\A\W`, `\W\z`.
- `\G` — привязка к стартовой позиции поиска. Шаблон, начинающийся этим символом, будет совпадать только в стартовой позиции, т. е. позиции, с которой задан поиск в тексте.

Мнимые символы не имеют смысла внутри символьных классов. Поэтому, к примеру, класс `[\z]` будет соответствовать символу `z`. Так как обратная косая стоит перед обычным символом в классе, то она просто отбрасывается, и это не считается ошибкой (при стандартных опциях библиотеки PCRE). А класс `[\b]` будет соответствовать символу забоя с десятичным кодом 8. Интересно, что в рассматриваемой версии PCRE шаблон `\b` с символом забоя не совпадает.

Существуют еще сложные якоря, в которых используются подшаблоны:

- `(?=`*шаблон*`)` — сразу после данной точки идет фрагмент текста, который соответствует заданному шаблону;
- `(?!`*шаблон*`)` — сразу после данной точки нет текста, который соответствует заданному шаблону;
- `(?<=`*шаблон*`)` — непосредственно перед данной позицией имеется фрагмент текста, который соответствует заданному шаблону;

- `(?<!`*шаблон*`)` — непосредственно перед данной позицией нет фрагмента текста, который соответствует заданному шаблону.

Первые два якоря называют *заглядыванием вперед*, а последние два — *заглядыванием назад*. В двух последних случаях на шаблон накладывается такое условие: этот шаблон должен соответствовать тексту с фиксированной длиной. В частности, этот шаблон не должен содержать квантификаторов кроме фиксированной длины `{n}`. Но он может содержать высокоуровневую альтернативу, которая совпадает со строками разной длины, например: `(?<=abc|abcd)`.

Подшаблоны в этих условных конструкциях не поглощают текст, т. е. если соответствие такому подшаблону найдено, то текущая позиция поиска соответствия в строке не продвигается. Это дает возможность к данной позиции текста применять множество проверочных условий, например, условия

```
a(?=\d)(?!1)
```

проверяют, чтобы после `a` стояла цифра и она не была единицей. Это регулярное выражение можно переписать и в такой эквивалентной форме:

```
a(?=(?!1)\d)
```

Но первая форма более ясная.

5.1.7. Захватывающие и незахватывающие скобки

Текст, соответствующий подшаблону, взятому в круглые скобки, запоминается в элементе массива с индексом, соответствующим номеру этой открывающей захватывающей скобки. Например, имеем шаблон

```
a(b)c
```

и текст

```
ababc
```

Если применить этот шаблон к данной строке, то он совпадет с третьего символа (считая с единицы), и вторая буква `b` окажется в первом элементе захватывающего массива. Имеется ограничение на 99 захватывающих пар скобок.

Если вы хотите применить скобки только для группировки подшаблона (например, чтобы поставить к нему квантификатор), то можно использовать незахватывающую конструкцию вида `(?:`*шаблон*`)`. В этом случае между знаком вопроса и двоеточием можно вставить модификаторы. Например, шаблон

```
(?i:a)
```

соответствует как строке `a`, так и `A`. А шаблон

```
(?i)(?-i:a)
```

соответствует только строке `a`, потому что внутренние квантификаторы отменяют действие внешних своих собратьев.

Незахватывающих скобок в шаблоне может быть 200 пар. "Неужели можно превысить это число?" — спросит читатель. Да, был у меня пример на Perl с поиском и заменой нехороших слов на всех языках в форумах. Регулярное выражение получилось под двести килобайт, и порог в 200 скобок был превышен. Пришлось разбивать шаблон на несколько частей. Тот скрипт был способен найти и обезвредить даже такие утонченные изыски, как `)|(*o$п a`.

Номера захватывающих скобок считаются в порядке появления открывающих скобок в регулярном выражении. Например, имеем шаблон

```
a((?!(a))(b)(?=(c)))
```

и строку

```
abc
```

Поиск соответствия будет таким:

1. Вызовется метод `Match` для поиска соответствия шаблона `a((?!(a))(b)(?=(c)))` строке `abc`. Он должен вернуть истину, если с какой-то произвольной позиции в строке найдется соответствие заданному шаблону. В этом случае захватывающие скобки, вообще говоря, будут захватывать соответствующие фрагменты текста. Если ни с одной позиции соответствия не будет найдено, то обнулится количество захваченных фрагментов и возвратится ложь.
2. Установится соответствие подшаблона `a` подстроке `a`. Текущая позиция поиска соответствия в шаблоне и строке сдвинется на один символ.
3. Откроется первая захватывающая скобка, которая начнет захват совпадения.
4. Для подшаблона `(?!(a))` рекурсивно вызовется метод `Match` для поиска соответствия шаблона `(a)`, начиная обязательно со второй позиции строки, т. е. с символа `b`. Этот метод вернет ложь, поэтому захватывающие скобки ничего не захватят, а условие `(?!(a))` окажется истиной, потому что после первого символа в строке нет символа `a`. Захваченная второй парой скобок подстрока будет пустой, как и бывает всегда в негативных условных проверках. Текущая позиция в шаблоне продвинется за подшаблон `(?!(a))`.
5. Откроется третья захватывающая скобка, позиция в шаблоне продвинется за нее.

6. Найдется соответствие символу `b`, позиции в шаблоне и строке продвинутся еще на один символ.
7. Закроется третья пара захватывающих скобок, она захватила фрагмент `b`. Позиция в шаблоне продвинется за `a((?!(a))(b)`, позиция в строке установится на символе `c`.
8. Начнется проверка условия `(?=(c))` с позиции на символе `c`. Вызовется метод `Match` для подшаблона `(c)`, и он найдет соответствие и захватит фрагмент `c` четвертой парой скобок. Но позиция в тексте не будет продвинута за символ `c`, потому что условия не поглощают текст, хотя, бывает, захватывают то, что лежит впереди или позади.
9. Закроется третья пара скобок, которая захватит фрагмент `b`. Внешний вызов `Match` вернет истину, поэтому первая пара скобок вернет `b`, вторая пусто, третья `b` и четвертая — `c`. На этом работа будет завершена: найдено соответствие с первого символа строки. Если бы оно не было найдено, то работа была бы продолжена следующей итерацией внешнего метода `Match` со второго символа строки и так до конца строки или до нахождения соответствия. Перед каждой итерацией счетчик захваченных фрагментов обнуляется.

5.1.8. Ссылки на найденный текст

Ссылки на найденный текст, иначе называемые *обратными ссылками*, — особенность, делающая поиск более интеллектуальным. Мы можем использовать в шаблоне уже найденные фрагменты текста. Обозначаются обратные ссылки так:

- `\1` для первой пары захватывающих скобок;
- `\2` для второй пары;
- ...;
- `\99` для 99-й пары.

Обратная ссылка представляет собой фрагмент текста, захваченный соответствующей парой скобок.

В случае, если эту конструкцию можно интерпретировать так же как восьмеричное число, приоритет отдается интерпретации как обратной ссылке, поэтому начинайте восьмеричные числа с нуля для большей ясности. Вообще-то сейчас больше принята шестнадцатеричная система записи чисел.

Рассмотрим практический пример: пусть нам надо в HTML-документе искать теги заголовков всех шести уровней `h1`, `h2`, ..., `h6`. Вот шаблон, который соответствует такому тегу:

```
(?is)<(h\d)>.*?<\1>
```

Он найдет текст `<h1>...</H1>`, но не будет соответствовать тексту `<h1>...</h2>`. Смысл конструкции `.*?` мы разберем позже.

А что, если за обратной ссылкой стоит цифра, ведь ее надо будет отделить от ссылки? Да, в этом случае придется взять ссылку в скобки:

```
(?:\1)1
```

Еще одна тонкость: если даже текст был захвачен не в зоне действия модификатора `i`, а обратная ссылка стоит в зоне его действия, то она будет соответствовать этому тексту без учета регистра букв. Шаблон

```
(a)(?i)\1
```

соответствует как строке `aa`, так и строке `aA`. Обратное не верно, шаблон

```
((?i)(a))\1
```

будет соответствовать строке `AA` и строке `aa`, но не будет соответствовать строкам `aA` и `Aa`.

Если обратная ссылка соответствует паре скобок, которая не участвовала в совпадении, то такая ссылка не будет соответствовать ничему, даже пустой подстроке. Пусть мы имеем шаблон

```
(a|(bc))\2
```

и строку a. Из альтернативы в совпадении участвует ветка `a`, ветка `(bc)` пропускается. Поэтому соответствия найдено не будет. Такой шаблон может чему-то соответствовать, только если в совпадении участвует ветка `(bc)`, например, строке `bcbc`.

Если обратная ссылка используется до закрытия соответствующей скобки, то это не считается ошибкой. Например, шаблон

```
(a\1)
```

не будет ничему соответствовать, т. к. эта пара скобок используется впервые и текст для ссылки `\1` еще не существует. Но шаблон

```
^(\1|a)\1$
```

будет соответствовать строке `aa`. С символом `a` совпадет вторая ветка альтернативы, а после выхода за закрывающую скобку обратная ссылка `\1` получит значение `a`.

Вот еще интересная головоломка: шаблон

```
^(a|b\1)+$
```

будет соответствовать строке `aba`. Как такое происходит? Сначала устанавливается соответствие первой ветки альтернативы символу `a`. Затем происходит выход за скобку и обратная ссылка `\1` получает значение `a`. Кванти-

фикатор + заставляет вернуться внутрь скобок и поискать совпадение с возможно большим числом повторов этой пары скобок. Во второй раз совпадает вторая ветка с фрагментом ba. При выходе за скобки в этот раз \1 получает значение ba. Квантификатор опять заставляет вернуться, но на этот раз подшаблон в скобках ни с чем не совпадает, на этом действие плюса заканчивается. В дело вступает условие $, которое совпадает с концом строки.

Заметим еще, что к обратным ссылкам тоже можно применять квантификаторы, как и к другим подшаблонам. PCRE запрещает ставить квантификаторы к условиям (якорям).

5.1.9. Условные подшаблоны

Конструкции с условными подшаблонами задаются в форме

(?(*условие*)*шаблон-да*)

или

(?(*условие*)*шаблон-да*|*шаблон-нет*)

Если условие выполняется, то на место всей этой конструкции подставляется *шаблон-да*, иначе в первом случае подставляется пусто, а во втором — *шаблон-нет*. Первая форма напоминает неполный условный оператор if...then, а вторая полный if...then...else. Условия могут быть трех видов:

- последовательность цифр — номер обратной ссылки. Если пара скобок, соответствующая этой обратной ссылке, участвовала в совпадении, то условие считается истинным, в противном случае — ложным. Здесь надо различать выражения "участвовать в совпадении" и "иметь непустое значение". Пара скобок может участвовать в совпадении, но захватить пустой фрагмент текста;
- сложные условия с заглядыванием вперед или назад. Если такое сложное условие выполняется, то условие в условном операторе получает значение "истина", иначе — "ложь";
- строка R. Это условие истинно, если мы находимся внутри рекурсивного вызова (?R) всего регулярного выражения. Вначале это условие является ложью. О рекурсивных шаблонах мы будем говорить позже.

Теперь примеры. Пусть нам надо извлекать из строки натуральные десятичные и шестнадцатеричные числа. Десятичные числа — это просто последовательность десятичных цифр, а шестнадцатеричные числа — это знак доллара, за которым идут шестнадцатеричные цифры. Общий для этих чисел шаблон выглядит так:

```
(?=(\$)|\d)(?(1)\$[0-9A-Fa-f]++|\d++)
```

Этот шаблон соответствует одному десятичному или шестнадцатеричному числу. Конструкции вида ++ мы обсудим в дальнейшем. Вначале ищется знак доллара или десятичная цифра. Затем, если был найден символ $, используется шаблон соответствия шестнадцатеричному числу, иначе — десятичному числу.

Шаблон

```
(\()?[^()]+(?(1)\))
```

соответствует любому тексту вне круглых скобок, а также тексту в одноуровневых скобках вместе с этими скобками. Если квантификатор ? имел значение 0, то первая пара скобок не участвовала в совпадении и в условном выражении закрывающая скобка не подставляется. Если квантификатор ? имел значение 1, то первая пара скобок захватила символ (и в условном выражении подставляется шаблон для закрывающей скобки.

5.1.10. Комментарии в регулярных выражениях

Комментарий имеет вид

```
(?#все_кроме_закрывающей_круглой_скобки)
```

Здесь все до первой закрывающей круглой скобки является комментарием. Вся эта конструкция перед компиляцией регулярного выражения удаляется. Но лучше использовать модификатор x, который позволяет свободное форматирование регулярного выражения в несколько строк. В этом случае символ # вне символьного класса, который не замаскирован \, означает начало комментария, который идет до конца строки. Правда, в этом случае пробелы в регулярном выражении придется задавать в виде Escape-последовательностей вида \040.

5.1.11. "Жадность" квантификаторов и ее ограничение

По умолчанию квантификаторы "жадны" и ищут совпадение с максимально возможным числом повторов. Например, шаблон

```
((?:ab)*)
```

примененный к строке

```
abababc
```

захватит фрагмент ababab. Шаблон

```
((?:ab){1,2})
```

примененный к строке

```
abababc
```

захватит подстроку `abab`.

Иногда это хорошо, но порой бывает нужно, чтобы квантификатор вел себя наоборот: соответствовал минимально возможному фрагменту. Для этого после квантификатора надо поставить знак вопроса. Например, шаблон

```
((?:ab)*?)
```

примененный к строке

```
abababc
```

захватит пустую строку. Шаблон

```
((?:ab){1,2}?)
```

примененный к строке

```
abababc
```

захватит текст `ab`.

Здесь замечу, что оба варианта ("жадных" и "нежадных") квантификаторов работают по одному алгоритму, просто "жадные" пробуют захватывать текст с наибольшего возможного числа повторов, а "нежадные" — с наименьшего числа.

5.1.12. Перебор с возвратами

И опять мы сталкиваемся с вездесущим перебором с возвратами.

Возьмем регулярное выражение

```
^a*ab$
```

и применим его к строке

```
ab
```

Будет ли найдено соответствие шаблона этому тексту? Да, будет. Вначале `a*` захватит по максимуму — символ `a`. Но затем совпадение не обнаружится. На этом поиск не прекращается, и квантификатор `*` отдаст один символ обратно, захватив ноль символов, а остальная часть шаблона найдет соответствие.

Для каждого квантификатора, который может захватывать переменное число символов, организуется счетчик текущего числа захватов и запоминаются позиции в тексте для каждой итерации счетчика. Если после квантификатора совпадение не обнаружится, то произойдет возврат в шаблоне к послед-

нему участвующему в работе квантификатору (или конструкции выбора), и, если он жадный, то отдаст одно захваченное совпадение, а если нежадный, то он попробует захватить еще одно совпадение. И так до полного исчерпания счетчика. Таким образом обеспечивается нахождение соответствия шаблона тексту, если этот шаблон может соответствовать этому тексту с произвольным легальным числом повторов для своих квантификаторов и произвольным ветвям выбора в конструкциях выбора.

Например, шаблон

```
a*
```

примененный к строке

```
aaa
```

порождает четыре сохраненных состояния:

- счетчик равен 0, позиция в строке перед первым символом;
- счетчик равен 1, позиция в строке перед вторым символом;
- счетчик равен 2, позиция в строке перед третьим символом;
- счетчик равен 3, позиция в конце строки.

Если позже не обнаружатся совпадения, то будет происходить постепенный переход от четвертого состояния к первому. И только после того, как счетчик станет равен нулю и совпадения все равно не обнаружатся, произойдет возврат к предыдущему квантификатору и его сохраненным состояниям. При исчерпании квантификаторов начальная позиция поиска в тексте будет продвинута на один символ, и произойдет повтор всего поиска совпадения со следующей позиции текста. И так до конца текста. Но этот внешний повтор будет выполняться, если шаблон не привязан к началу текста условиями `^` или `\A` или к начальной позиции поиска условием `\G`. Иначе будет зафиксировано несоответствие шаблона тексту без итераций общего поиска.

Хранение этих состояний требует памяти, поэтому есть ограничение на 10 млн состояний для одного регулярного выражения.

А для конструкции выбора происходит запоминание номера совпавшего подшаблона, и при откате назад произойдет примерка следующего справа подшаблона от подшаблона из этой конструкции, совпавшего в последний раз. Если такого подшаблона нет, то откат к квантификаторам или конструкциям выбора продолжится, насколько это возможно.

5.1.13. Атомарная группировка

Представим, что мы ищем в длинной строке

```
aaaaaaaaaac
```

букву `b` по шаблону

```
a*b
```

Пока функция `Match` придет к выводу, что совпадения нет, будет произведена большая и ненужная работа по сохранению состояний квантификатора и по отдаче им по одному символу от всех захваченных букв `a` до нуля. Это не всегда бывает нужно по смыслу поиска, поэтому существует возможность сделать подшаблон как бы единым целым. На этот случай существует специальная конструкция

```
(?>шаблон)
```

В данном случае шаблон переписывается так:

```
(?>a*)b
```

Квантификатор `*` сразу захватит все символы `a`, но отдавать он их не будет. При обнаружении несовпадения после закрывающей скобки произойдет возврат назад, но не внутрь атомарной группировки, а сразу за нее, в данном случае в начало шаблона. Поиск буквы `b` повторится со второго, третьего и т. д. символа, пока система не придет к убеждению, что буквы `b` в данной строке нет. В данном случае для отмены итераций поиска можно привязать шаблон к началу строки:

```
^(?>a*)b
```

После этого вывод о несоответствии шаблона строке будет сделан мгновенно.

Но атомарную группировку надо применять, если мы уверены, что отдача квантификаторами захваченных символов не поможет найти соответствия. Например, мы ищем ссылку на Web-узел **gameintellect.com** в конструкции

```
href = "http://www.gameintellect.com/..."
```

Перед и после знака равенства может идти любое число пробельных символов, ноль или более. Здесь уместно применить *атомарную группировку*:

```
href(?>\s*)=(?>\s*)"(http://www\.gameintellect.com(?>[^"]*)"
```

Ведь если, начиная с кавычек, не найдено соответствие, то отдача обоими квантификаторами пробельных символов не поможет его найти. В случае с закрывающей кавычкой соображение по применению атомарной группировки аналогично. В результате так нужный нам URL будет захвачен скобками.

Но у читателя здесь могут возникнуть такие вопросы:

1. А что, если данный URL заключен в апострофы или вовсе ничем не ограничен? А в иных документах HTML встречаются даже случаи выделения URL обратными апострофами (``)! Как мы тогда найдем эту ссылку?

2. А вдруг Web-мастер забыл закрыть кавычку? Не захватят ли тогда скобки того, что им не положено?

Да, вынужден признать, что это регулярное выражение было составлено в надежде на русское "авось": авось Web-мастер делал этот документ на ясную голову, авось он всегда использует кавычки... Но ведь Internet Explorer както ухитряется показывать документы с ошибками правильно, сам их у себя исправляет и даже не сообщает об этом. О построении более гибких регулярных выражений мы поговорим чуть позже.

Далее замечу, что внутри шаблона с атомарной группировкой возвраты назад выполняются. Если шаблон

```
(?>a*a)
```

применить к строке, состоящей из одних букв `a`, то, не выходя за скобки, система поиска совпадения обнаружит, что квантификатор `*` захватил все буквы, а в шаблоне после этого еще имеется одна буква `a`. И квантификатор пожертвует одной захваченной буквой во имя нахождения соответствия шаблона тексту.

Совсем не так поведет себя шаблон

```
(?>a*)a
```

Соответствие строке из букв `a` найдено не будет. В этом случае и без того жадный квантификатор становится просто каким-то Плюшкиным или, на европейский лад, Гобсеком, и, сжав набранные символы в дрожащем кулачке, ни за что не отдает захваченное (вообще-то отдает, но только при возвратах внутри атомарной группировки).

С поведением максимальных квантификаторов в атомарной группировке мы разобрались. А что происходит в этом случае с минимальными квантификаторами? Они ведут себя аналогично: ни за что не хотят взять хотя бы еще одну порцию соответствующего им текста (но внутри атомарной группировки берут). В результате шаблон

```
^(?>a*?)b
```

не будет соответствовать строке

```
ab
```

Квантификатор `*` захватит ноль символов `a`, а там хоть трава не расти. В шаблоне следующий символ `b`, а в тексте по-прежнему на очереди символ `a`? Все равно его не возьму! Отходите за открывающую скобку и пробуйте дальше. А перед открывающей скобкой стоит привязка к началу строки...

Еще надо рассмотреть поведение конструкции выбора с атомарной группировкой. Ведь в конструкции выбора тоже запоминается, какая ветка совпала, и при возврате назад механизм сопоставления шаблона тексту продол-

жит сопоставление со следующей ветви. Но в случае атомарной группировки откат назад будет произведен опять за всю конструкцию выбора, и следующая ветвь сопоставлена тексту не будет. Вот пример, имеем регулярное выражение

```
^(?>ab|a)b
```

и строку

```
ab
```

Первая ветвь `ab` совпадет со всей строкой, поэтому для буквы `b` за скобкой не будет найдено соответствие. При этой неудаче не произойдет возврат внутрь скобок, и вторая ветка конструкции выбора задействована не будет. Но если рассмотреть шаблон

```
^(?:ab|a)b
```

то при несовпадении всего шаблона при выборе первой альтернативы произойдет откат обратно внутрь скобок, и будет найдено соответствие текста второй ветке плюс букве `b` за скобкой.

5.1.14. Захватывающие квантификаторы

Чтобы не вводить лишних атомарных скобок для простых подшаблонов, используется специальная форма квантификаторов. Если после максимального квантификатора поставить знак `+`, то такой квантификатор становится захватывающим и никогда не отдает захваченный текст. Таким образом, квантификаторы

- `?+`;
- `*+`;
- `++`;
- `{`*мин*`,`*макс*`}+`

являются захватывающими. Минимальные квантификаторы

- `??`;
- `*?`;
- `+?`;
- `{`*мин*`,`*макс*`}?`

не могут быть захватывающими.

Шаблон `a*+` эквивалентен шаблону `(?>a*)`, но может работать более эффективно.

5.2. Программная реализация PCRE

С авторского сайта **www.pcre.org** вы можете скачать документацию, исходные тексты (на языке C) и выполняемые файлы (в том числе библиотеку pcre.dll). Вы можете использовать этот dll-файл в своих программах на Delphi, предварительно создав для него интерфейсный файл.

Программист Jan Goyvaerts сделал на Delphi бесплатный компонент PerlRegex, последнюю версию которого можно скачать с авторского сайта **http://www.regular-expressions.info/delphi.html**. На компакт-диске, прилагаемом к книге, в подкаталоге Lib он также имеется, и вы можете установить его в палитру компонентов Delphi. В файле PerlRegEx.hlp описаны его методы и свойства, а также приведены примеры использования. Этому компоненту не нужен dll-файл, он использует объектные файлы от библиотеки Philip Hazel. Чтобы их получить, надо оттранслировать исходные файлы в системе C++Builder, потому что Microsoft-формат объектных файлов, который выдают трансляторы с языка C, отличается от Borland-формата. Также надо написать интерфейс к оригинальным функциям и системозависимые вызовы типа `memalloc`, которые используют оригинальные библиотечные функции из объектных файлов. Это и сделал Jan Goyvaerts.

Я сделал небольшое исправление в авторском файле PerlRegex.pas, который находится в каталоге Lib, в методах `Match` и `MatchAgain`. Я взял часть кода в комментарий `(*...*)`. Иначе, к примеру, шаблон `$` и другие шаблоны с нулевой длиной совпадения никогда не соответствуют пустой строке.

В листинге 5.1 вы видите содержимое файла Unit1.pas.

Листинг 5.1. Пример использования библиотеки PCRE

```
unit Unit1;

interface

uses
  Windows, SysUtils, Forms, Classes, Controls, StdCtrls, PerlRegEx;

type
  TForm1 = class(TForm)
    PerlRegEx1: TPerlRegEx;
    Button1: TButton;
    re: TEdit;
```

```
    text: TMemo;
    Label1: TLabel;
    Label2: TLabel;
    out: TMemo;
    Label3: TLabel;
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
var i: integer;

begin
with PerlRegEx1 do
 begin
 Subject:=text.Text;
 Start:=1;
 Stop:=Length(Subject);
 RegEx:=re.Text;
 out.Lines.Clear;
 if Match then
  for i:=0 to SubExpressionCount do
   begin
   out.Lines.Append('SubExpressionOffsets['+IntToStr(i)+']='+
    IntToStr(SubExpressionOffsets[i]));
```

```
    out.Lines.Append('SubExpressionLengths['+IntToStr(i)+']='+
     IntToStr(SubExpressionLengths[i]));
    out.Lines.Append('SubExpressions['+IntToStr(i)+']='+
     SubExpressions[i]);
    end;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
PerlRegEx1.State:=[];
end;

end.
```

Чтобы создать этот пример, на форму надо перетащить компонент PCRE и сделать поле **Edit** и два **Memo**-поля. В поле **Edit** можно вводить регулярные выражения, а в первом **Memo**-поле — текст. После щелчка по кнопке **OK** во втором **Memo**-поле в случае нахождения совпадения появляется результат. В элементах массива `SubExpressionOffsets[i]` выводятся смещения для совпадений в элементах:

- `SubExpressionLengths[i]`— длины совпадений;
- `SubExpressions[i]` — текст совпадений.

Нулевой элемент предназначен для совпадения всего регулярного выражения, а элементы 1, 2, ..., 99 — для захватывающих скобок соответствующего номера. Если вы в **Memo**-поле вводите текст на нескольких строках, то имейте в виду, что строки разделяются символами `\r\n`. Например, тексту

```
ab
c
```

соответствует шаблон

```
^ab\r\nc$
```

Компонент не только может искать и захватывать текст, но также заменять найденные фрагменты текста и разбивать текст на части по заданному регулярному выражению аналогично оператору `split` языка Perl.

Мы с помощью примера Project1.exe будем учиться писать регулярные выражения и мыслить их категориями.

Компонент имеет имена для обозначения состояния, которые соответствуют обозначениям состояний оригинальной библиотеки Philip Hazel:

- `preNotBOL` — якорь `^` не совпадает в начале заданного для поиска текста. Можно использовать, если заданный текст является частью другого текста, которая начинается не с начала всего текста;
- `preNotEOL` — якорь `$` не совпадает в конце заданного для поиска текста. Можно использовать, если заданный текст является частью другого текста, которая завершается раньше конца всего текста;
- `preNotEmpty` — совпадения с пустой подстрокой не допускаются. У автора компонента этот бит был установлен по умолчанию, но это не соответствует правилам регулярных выражений языка Perl, поэтому я в обработчике события `FormCreate` отменил это оператором

  ```
  PerlRegEx1.State:=[];
  ```

Также компонент имеет опции:

- `preCaseLess` — независимость от регистра букв. Аналогично `(?i)` в начале шаблона;
- `preMultiLine` — якоря `^` и `$` также совпадают в начале/конце внутренних строк. Аналогично `(?m)` в начале шаблона;
- `preSingleLine` — точка также соответствует символу новой строки `\n`. Соответствует `(?s)` в начале шаблона;
- `preExtended` — позволяет иметь внутри шаблона пробельные символы, которые будут игнорироваться при компиляции. Также разрешает иметь внутри шаблона комментарии в стиле Perl: от символа `#` и до конца строки;
- `preAnchored` — соответствует символу `/A` в начале шаблона. Привязка начала совпадения к началу текста или к концу предыдущего совпадения;
- `preDollarEndOnly` — якорь `$` совпадает лишь в конце текста и не совпадает перед символом `\n`, который стоит в конце текста;
- `preExtra` — обратная косая черта, стоящая перед обычным символом, не являющимся метасимволом, вызывает ошибку при компиляции. Можно использовать для более строгого контроля за написанием регулярного выражения;
- `preUnGreedy` — квантификаторы по умолчанию становятся нежадными.

Эти состояния и опции перекрываются аналогичными модификаторами, стоящими внутри шаблона.

Работать с опциями можно, например, так:

```
PerlRegEx1.Options:=PerlRegEx1.Options+[preCaseLess,preAnchored];
```

5.2.1. Простые примеры использования PCRE

Имея программу Project1.exe, потренируемся составлять регулярные выражения для поиска совпадений.

Пусть нам надо найти (чтобы удалить) пробельные символы в начале строки. Для этого можно использовать шаблон

```
^\s++
```

Шаблон

```
\s++$
```

соответствует пробельным символам, стоящим в конце строки.

Для извлечения целого числа можно использовать шаблон

```
([+-]\d+)
```

Введите этот шаблон в поле **Regex** проекта 1, а в поле **Text** введите какой-нибудь текст, например,

```
это некоторый
текст-123 4
```

Тогда в поле вывода **Output** получим такой результат:

```
SubExpressionOffsets[0]=21
SubExpressionLengths[0]=4
SubExpressions[0]=-123
SubExpressionOffsets[1]=21
SubExpressionLengths[1]=4
SubExpressions[1]=-123
```

Мы видим, что все выражение совпало с 21-ой позиции и что длина совпадения равна четырем. Скобки захватили текст `-123`.

Пусть нам нужно извлечь URL из тега `a`. Например, из текста

```
<A
href = "http://www.gameintellect.com">GameIntellect.com</a>
```

надо извлечь ссылку **http://www.gameintellect.com**. Для этого можно применить выражение

```
(?i)<a\s++href\s*+=\s*+"([^"]+)
```

Действительно, наша программа находит эту ссылку в данном тексте. А если текст будет таким:

```
<a href=http://www.gameintellect.com>GameIntellect.com</a>
```

Нет, если URL задан без кавычек, то он уже не находится. Необходимо усовершенствовать шаблон. Видоизменим его так:

```
(?is)<a\s++href\s*+=\s*+(["']?+)([^"'>\040]+)\1.*?</a>
```

В этом случае в подшаблоне

```
(["']?+)
```

мы берем разделяющие кавычки или апостроф в первые скобки, а затем после вторых скобок, которые захватывают URL, вставляем обратную ссылку, проверяя, чтобы этот же ограничитель стоял после ссылки. Если ограничителя нет, то `\1` будет пустой строкой. Затем с помощью подшаблона

```
.*?
```

мы выходим на конец тега `a`, чтобы потом выйти за его переделы для продолжения поиска других тегов ссылок. Модификатор `s` нужен, чтобы последний подшаблон поглощал также переводы строк. Вместо последнего подшаблона можно было бы использовать подшаблон

```
[^<]*+
```

Для проверки введем теперь такой текст:

```
<a title='Games' href=http://www.gameintellect.com>GamaeIntellect.com</a>
```

Как и следовало ожидать, шаблон перестает находить ссылку. Ведь он считает, что элемент `href` должен быть первым в теге, что по правилам языка HTML не является обязательным. Еще раз модифицируем регулярное выражение:

```
(?is)<a\s++[^>]*?href\s*+=\s*+(["']?+)([^"'>\040]+)\1.*?</a>
```

Мы вставили подшаблон

```
[^>]*?
```

для пропуска символов от `<a` с пробелами до появления `href`. Мы пропускаем все символы кроме `>`, чтобы не выскочить за угловые скобки. С этой целью можно было бы также использовать максимальный квантификатор

```
[^>]*
```

Но он первоначально захватил бы весь текст до символа `>` и отдавал бы его по одному символу, пока у него не остался бы захваченным текст

```
title='Games'
```

плюс один пробел. Минимальный квантификатор приходит к этому результату без перебора с возвратами. Во вторых скобках мы получаем искомую ссылку.

Рассмотрим учебный пример, как надо поступать, если внутри искомого текста может находиться ограничивающий символ. Например, мы внутри формы ищем теги `input`. Язык HTML не запрещает внутри строки использовать угловые скобки, хотя это и не рекомендуется. Предположим, имеется такая форма:

```
<form...>
<input type=text value='>'      >
```

Внутри строки стоит закрывающая угловая скобка. Поиск по следующему регулярному выражению:

```
(?i)(<input\s[^>]++>)
```

споткнется на этой скобке внутри строки и не найдет всего тега. А это может сбить дальнейший разбор текста HTML. Шаблон должен принимать все символы кроме апострофов, кавычек и закрывающих угловых скобок, а также апостроф/кавычку и все до такого же следующего символа. На языке регулярных выражений это записывается так:

```
(?i)(<input\s(?:[^"'>]++|(["'])[^"']*+\2)*+>)
```

Что ж, этот шаблон захватывает тег `input` в первую пару скобок.

А теперь поставим внутрь апострофов кавычку или даже пару кавычек:

```
<form...>
<input type=text value='">"'      >
```

Ага, уже не работает! А Internet Explorer, наверно, и это не собьет. Вообще-то синтаксический разбор документа делает не он, а dll-библиотека в каталоге Windows. Но нам от этого не легче. Сбой происходит внутри подшаблона

```
[^"']*+
```

который должен пропускать только захваченный символ `\2`, а он пропускает также кавычки. Но внутри класса обратные ссылки не работают, ведь за Escape-последовательностью `\2` может скрываться не один, а сколько угодно символов.

Итак, нам нужна конструкция, которая пропускает все символы кроме `\2`. Язык регулярных выражений достаточно гибок, чтобы запрограммировать это действие. Обратимся к конструкции для заглядывания вперед:

```
(?!шаблон)
```

Она может заменить класс символов, имея более широкие возможности. Нам надо в цикле брать символ, за которым не идет символ или строка `\2`. Теперь все регулярное выражение выглядит так:

```
(?i)(<input\s(?:[^"'>]++|(["'])(?:(?!\2).)*+\2)*+>)
```

Я не добавил модификатор `s`, чтобы метасимвол . соответствовал также символу `\n`, ведь в HTML строки вроде бы не могут разрываться символами `\r` и `\n`. Этот шаблон опять находит тег `input`.

Замечание

В языке PCRE есть Escape-последовательность `\C`, которая соответствует строго одному *байту*, если даже он является частью многобайтного символа UTF-8. `\C` можно использовать вместо точки, что ускорит работу интерпретатора, потому что ему не нужно будет проверять каждый байт, не является ли он началом многобайтового символа (в случае поддержки UTF-8, что задается при компиляции оригинальных C-программ).

Но внутри строки может находиться символ, ограничивающий эту строку, если он замаскирован обратной косой чертой: `\'`. Напишем такой пример для нашего случая:

```
<form...>
<input type=text value='">\'"'      >
```

Положение дел довольно искусственное, но это учебный пример. Шаблон опять перестал находить тег `input`. Может быть, такого универсального шаблона на наш пример не бывает в природе?

"No problem!" — говорит PCRE. В общем случае, нам надо внутри строки пропускать также символ `\` вместе со следующим за ним символом. Обратная косая не может стоять в самом конце строки, иначе тогда она маскировала бы ограничитель этой строки. В цикле нам надо кроме символа, за которым не стоит `\2`, также брать символ `\` и следующий за ним символ, т. е. вставить еще одну альтернативу:

```
(?i)(<input\s(?:[^"'>]++|(["'])(?:\\.|(?!\2).)*+\2)*+>)
```

Но при применении альтернатив будьте внимательны при выборе их порядка, если одна из ветвей может соответствовать тексту, предназначенному для другой ветви. В нашем случае альтернатива

```
(?!\2).
```

может соответствовать тексту, для которого предназначена альтернатива

```
\\.
```

Если бы мы поменяли эти ветви местами, то такой шаблон уже не нашел бы этого хитрого тега `input`:

```
(?i)(<input\s(?:[^"'>]++|(["'])(?:(?!\2).|\\.)*+\2)*+>)
```

Получившийся шаблон со многими скобками выглядит сложновато. Его можно записать в расширенном режиме на нескольких строках и с комментариями:

```
(?ix)                 # Нечувствительность к регистру и расширенный режим
(                     # Начинаем захват тега input
  <input\s            # Ключевое слово
    (?:[^"'>]++       # Все кроме строк и конца тега
      |(["'])         # Или начало строки, которое берем в \2
        (?:\\.        # В строке — любой замаскированный символ
          |(?!\2).    # Или любой символ, за которым нет \2
        )*+           # Повторяются сколько угодно раз
      \2              # В конце строки — ее ограничитель
    )*+               # Внешняя альтернатива повторяется сколько угодно раз
  >                   # Потом идет закрывающий тег
)                     # Конец захвата тега input
```

Часто при написании текстов по ошибке одно и то же слово повторяется два раза подряд. Каким шаблоном это можно отловить? Это регулярное выражение выглядит так:

```
\b(\w++)\s++\1\b
```

От границы слова оно начинает захват алфавитно-цифровых символов и продолжает его делать до первого пробельного символа. После одного или более пробельных символов должно повториться это же слово. Вы можете сами усовершенствовать это выражение.

Займемся поиском в тексте адресов URL и e-mail и преобразованием их в ссылки. Эта задача встречается Web-мастерам, например, когда надо подсветить ссылки и почтовые адреса в постах на форуме. Это неформальная задача, она в чем-то интуитивна.

5.2.2. Именованные захватывающие скобки

Пусть мы написали мудреное регулярное выражение, оттестировали его и потираем ручки от творческого удовлетворения. Но вдруг замечаем, что надо было захватить в тексте кое-что еще, т. е. надо вставить еще одну пару захватывающих скобок. И тут может произойти небольшой кошмарик: ведь номера этих скобок могут быть завязаны на номера обратных ссылок и работу программы. PCRE в таких случаях предлагает решение: к захватывающим скобкам можно обращаться не только по номерам, но и по именам.

Синтаксис у этих скобок такой:

`(?P<имя>шаблон)`

Именованные скобки имеют те же номера, как если бы они были обычными захватывающими скобками. И к ним также можно обращаться по номерам. Просто их именам ставятся в соответствие их номера. Эти скобки также можно вкладывать друг в друга, например:

```
(?P<date>(?P<year>(?:\d\d)?\d\d)/(?P<month>\d\d)/(?P<day>\d\d))
```

Этот шаблон соответствует дате, записанной в форме `гг/мм/дд` или `гггг/мм/дд`. Вся дата ловится первой парой захватывающих скобок, год — второй, а день — третьей парой.

В нашем программном примере содержимое этих скобок будет находиться в строках, соответственно

```
PerlRegEx1.SubExpressions[NamedSubExpression('date')]
PerlRegEx1.SubExpressions[NamedSubExpression('year')]
PerlRegEx1.SubExpressions[NamedSubExpression('month')]
PerlRegEx1.SubExpressions[NamedSubExpression('day')]
```

А также в соответствующих нумерованных строках

```
PerlRegEx1.SubExpressionOffsets[i]
```

где `i = 1`, `2`, `3` и `4`.

Но если в программе запросить содержимое скобок с неиспользуемым в шаблоне именем, то в этом случае почему-то выдается один символ из первой пары захватывающих скобок. Я не знаю, чья здесь ошибка — разработчика PCRE или создателя компонента для Delphi. Впрочем, одинаковые имена для разных скобок у Philip Hazel в этой версии уже замечаются и запрещаются, что радует.

5.2.3. Пример программы для замены текста

До сих пор мы только искали текст, но с помощью компонента PCRE можно также заменять найденные образцы на другой текст, который может включать в себя обратные ссылки, а также удалять найденные фрагменты, поскольку удаление — это замена на пустую строку. Этот пример находится в папке Project2 на прилагаемом к книге компакт-диске. Он аналогичен уже рассмотренному примеру, но в нем регулярное выражение можно вводить на нескольких строках благодаря компоненту **Memo**, и добавлено поле ввода типа **Edit** под названием **Replacement**. В листинге 5.2 вы видите текст модуля Unit1.pas.

Листинг 5.2. Пример замены найденных фрагментов текста

```
unit Unit1;

interface

uses
  Windows, SysUtils, Forms, Classes, Controls, StdCtrls, PerlRegEx;

type
  TForm1 = class(TForm)
    PerlRegEx1: TPerlRegEx;
    Button1: TButton;
    text: TMemo;
    Label1: TLabel;
    Label2: TLabel;
    out: TMemo;
    Label3: TLabel;
    re: TMemo;
    Label4: TLabel;
    repl: TEdit;
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}
procedure TForm1.Button1Click(Sender: TObject);
```

```
begin
with PerlRegEx1 do
 begin
 Subject:=text.Text;
 Start:=1;
 Stop:=Length(Subject);
 RegEx:=re.Text;
 Replacement:=repl.Text;
 ReplaceAll;
 out.Text:=Subject;
 end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
PerlRegEx1.State:=[];
end;

end.
```

Для того чтобы собрать в строку все, что присутствует в поле **Memo**, используется свойство `Text`, аналогичное свойству `Text` компонента **Edit**. В этом случае строки разделяются символами `#13#10` или `\r\n` в терминах регулярных выражений. Метод `ReplaceAll` заменяет все фрагменты текста, которые соответствуют заданному регулярному выражению, на то, что введено в поле **Replacement**.

Запустите этот пример и введите такое регулярное выражение:

```
(?i)(\w+)\s+(\w+)\s+(\w+)
```

В поле **Text** введите такой текст:

```
Собака укусила человека
```

В поле **Replacement** введите

```
\3 \2 \1
```

Если нажать кнопку **OK**, то в поле **Output** прочитаем:

```
человека укусила Собака
```

Составленное регулярное выражение соответствует всему предложению в поле **Text**. При этом в захватывающие скобки попадают все слова по порядку. В поле **Replacement** первое и третье слово обмениваются местами.

Теперь замените шаблон на следующий:

```
(?i)(\w+\s*)
```

а в качестве замены введите текст

```
\1
```

Что в итоге должно получиться? А получается тот же самый текст. Происходит это так: внутри метода `ReplaceAll` вначале вызывается метод `Match`, который находит соответствие введенного шаблона тексту `Собака` (с пробелом после него). Затем вызывается метод `Replace` (который вызывается только в случае, если метод `Match` вернул `TRUE` (т. е. нашел соответствие). Он заменяет соответствующий шаблону фрагмент текста на `\1`, т. е. на самого себя. Затем работает цикл

```
while MatchAgain do
 Replace;
```

Он еще два раза заменяет найденные фрагменты на них самих. При этом `MatchAgain` начинает поиск с той позиции, на которой он закончился последний раз. Но вы можете поменять эту позицию, изменив значение свойства `Start`.

Рассмотрим еще два более сложных практических примера на замену (удаление) текста. Недавно один знакомый прислал письмо с просьбой написать регулярные выражения для удаления таких конструкций в коде HTML:

```
<A
HREF=http://ad.doubleclick.net/jump/idg.us.idg.net.standard.rsstower/;pos
=skyscraper;tile=2;sz=160x600;ord=20060417050449?
TARGET=_blank><IMG
SRC=http://ad.doubleclick.net/ad/idg.us.idg.net.standard.rsstower/;pos=sk
yscraper;tile=2;sz=160x600;ord=20060417050449?
WIDTH=160 HEIGHT=600 BORDER=0 ALIGN=right HSPACE=5 VSPACE=5 /></A>
```

и

```
<p class="goog"><map name="google_ad_map_060414104741"> <area
shape="rect"
href="http://imageads.googleadservices.com/pagead/imgclick/060414104741?p
os=0"
coords="1,2,367,28"/> <area shape="rect"
```

```
href="http://services.google.com/feedback/abg"
coords="384,10,453,23"/></map> <img
usemap="#google_ad_map_060414104741" border="0"
src="http://imageads.googleadservices.com/pagead/ads?format=468x30_aff_img&amp;client=ca-pub-1362322931946654&amp;channel=8269939301&amp;output=png&amp;cuid=060414104741&amp;url=http%3A%2F%2Fwifinetnews.com%2Farchives%2F006477.html"/></p>
```

В первом случае ключевым моментом является нахождение фрагмента `ad.doubleclick.net` внутри тега `<a ...>...</a>`, а во втором случае — нахождение `services.google.com` внутри `<p>...</p>`.

Первый случай более простой, мы сначала ищем открывающий тег `<a` подшаблоном

```
<a\s
```

который гарантирует, что это именно тег `<a`, а не тег, начинающийся на букву `a`. Затем подшаблоном

```
[^>]*?ad\.doubleclick\.net
```

мы пропускаем все символы кроме ограничителя тега `>` и выходим за фразу `ad.doubleclick.net`, если она имеется внутри угловых скобок. Остаток тега до `</a>` пропускаем по подшаблону

```
.+?
```

Модификатор `i` нужен, т. к. в HTML можно задавать теги и ссылки как прописными, так и строчными буквами, а модификатор `s` необходим, чтобы метасимвол `.` соответствовал также символу перевода строки, т. к. тег может располагаться на нескольких строках. В сумме получаем такой шаблон:

```
(?is)<a\s[^>]*?ad\.doubleclick\.net.+?</a>
```

Как видим, минимальные квантификаторы используются, когда надо пропустить что-то до заданного после них образца. Перед поглощением очередного символа проверяется, не является ли он началом следующего за квантификатором подшаблона. Из-за этой проверки минимальные квантификаторы работают медленнее максимальных. В нашем случае за ними идет литеральный текст, а в общем случае может следовать любой подшаблон, который при каждой итерации квантификатора будет проверяться на соответствие тексту с текущей позиции.

Во втором случае подшаблон `[^>]*?` уже не обеспечит пропуск текста от `<p` до искомого фрагмента `services.google.com`, потому что на этом пути встречаются закрывающие угловые скобки. Здесь критерий пропуска символов такой: пропускаем все до `services.google.com`, не выходя за `</p>`. На

язык PCRE это переводится с помощью утверждений, проверяющих текст перед шаблоном. Вот это регулярное выражение:

```
(?is)<p\s(?:(?!services\.google\.com)(?!</p>).)++services\.google\.com.+?
</p>
```

После поглощения текста по шаблону `<p\s` перед тем, как взять очередной символ, проверяем в цикле, чтобы с текущей позиции не шел текст, который соответствует подшаблону `services\.google\.com` или подшаблону `</p>`. Только в этом случае берем этот символ с помощью точки. После того как этот цикл остановился, должен идти текст, соответствующий `services\.google\.com`, а затем уже все остальное кроме `</p>`.

В той же папке Project2 есть файл examples.txt, в котором находятся эти примеры и соответствующие им регулярные выражения.

5.2.4. Пример преобразования ссылок в теги *<a*

При обработке таких текстов, как письма и посты на форумах, возникает задача отыскания в тексте ссылок и выделение их в теги. Например, текст

```
Зайдите на gameintellect.com/flash-games и посмотрите список игр
```

Здесь текст `gameintellect.com/flash-games` является ссылкой, несмотря на отсутствие протокола `http://`, который подразумевается по умолчанию. Может быть и такое, что ссылка не отделена пробелом от окружающих слов или после нее идет знак препинания (точка, запятая и т. д.) Желательно, чтобы регулярное выражение это учитывало и не включало такой знак в ссылку. И конечно, оно не должно совпадать там, где ему совпадать не следует. Неплохо было бы, если бы оно также форматировало текст ссылки `href`: протокол, домен и субдомены должны быть записаны строчными буквами. Задача эта непростая и не формализуется. Критерием успешности регулярного выражения является то, как оно справляется с набором тестов, которые провоцируют его к несовпадению или совпадению не в тех местах.

Это регулярное выражение достаточно сложное и громоздкое, и мы будем создавать его по частям. Начнем с протокола.

Протокол может быть `http`, `https` и `ftp`. Для его обнаружения создадим строковую переменную `protocol`:

```
protocol:='(?:(?=[FfHh])(?i:https?+|ftp)://)';
```

Если в тексте следующий символ — это `F`, `f`, `H` или `h`, то этот подшаблон делает проверку следующих за ним символов и, если это протокол, поглощает его вместе с префиксом. Я взял весь шаблон для протокола в скобки, потому что в общем регулярном выражении у этого подшаблона может сто-

ять квантификатор, который должен относиться ко всему этому подшаблону, а не к последнему его символу /.

Для имени хоста запишем такое регулярное выражение в свободном формате:

```
host:='(?x:(?>[A-Za-z0-9]{1,63}\.)'+
          '(?>[A-Za-z0-9]'+
           '(?>[-A-Za-z0-9]{0,62})\.'+
          ')*)';
```

Эта запись соответствует последовательности имен, разделенных точками, она в том числе поглотит префикс www. Также учтено, что длина одного имени (от точки до точки) не может быть больше 63 символов. Скобки для ввода модификатора также отделяют этот шаблон от того, что будет после него, и группируют его данные, ведь в конечном регулярном выражении после подстроки host может стоять квантификатор.

Для поддоменов запишем такой шаблон:

```
subdom:='(?:(?>[A-Za-z0-9](?:[-A-Za-z0-9]{0,61}[A-Za-z0-9])?)\.)+';
```

Введем вспомогательный подшаблон, который будет повторяться в других подшаблонах:

```
wb:='(?![A-Za-z0-9])';
```

Он означает, что слева не должно быть ни буквы, ни цифры.

Для зоны (это может быть com, net, org, ..., co.uk, ...) имеется такой подшаблон:

```
zone:='(?ix:(?(?=[a-z]{3}'+wb+')(?>com|net|org|edu|biz|gov|int|mil)|'+
               '(?(?=[a-z]{2}'+wb+')[a-z]{2}|'+
                '(?(?=[a-z]{4}'+wb+')(?>info|aero|name)|'+
                 '(?(?=[a-z]{6}'+wb+')museum|(?!)'+
                 ')'+
                ')'+
               ')'+
             ')'+
             '(?>\.[a-z]{2}'+wb+')?)';
```

После имени хоста через двоеточие может идти порт:

```
port:='(?::\d{1,5}'+wb+')';
```

А после зоны может идти хвост, который содержит множество всяких параметров, передаваемых с URL:

```
tail:='(?:[/?](?>[^.,"'<>()\[\]{}\s\x7F-\xFF]*)(?:(?>[.,?]+)
(?:[^"'<>()\[\]{}\s\x7F-\xFF]+))*(?<![,.?!-]))';
```

Это одна строка, просто она не помещается по ширине на странице книги.

В конце записано заглядывание назад

```
(?<![,.?!-])
```

которое учитывает, что после URL могут стоять знаки препинания, не входящие в него.

Все регулярное выражение на соответствие URL в тексте выглядит немного страшновато:

```
re.Text:='((?>('+protocol+')(?(2)(?>'+ip+'|'+host+zone+')|'+host+zone+')
(?![A-Za-z0-9])|(?<![A-Za-z0-9_@-])(?<!\.(?!(?i:www)))'+
subdom+zone+'(?![A-Za-z0-9_.-]*@))(?>(?>'+port+'?(?>@'+host+zone+
'(?![A-Za-z0-9_.-]*@))?)?))('+tail+'?)';
```

Это тоже одна строка. Это выражение учитывает заход через прокси-сервер вида

```
http://proxy.com@site.com/
```

Программа должна "подсвечивать" ссылки в тексте. Например, имеем текст

```
Look at:aaa.Museum.
```

Это должно превратиться в

```
Look at:<a href="http://aaa.museum" target=_blank>aaa.Museum</a>.
```

Здесь возможный URL отделяется от прилипших справа и слева символов и оформляется в тег `a`. Внутри строки `href` имя хоста и зона записываются строчными буквами, а в тексте, который будет виден на странице, форматирование не производится. Еще программа должна учесть, что в хвосте URL (имена подкаталогов и параметры) форматировать текст нельзя, т. к. эти слова чувствительны к регистру символов.

Как же мы будем форматировать найденный текст? Для этого у библиотеки PCRE имеется мощное средство — возможность вызывать пользовательскую функцию при достижении в шаблоне определенной точки. В компоненте это доступно через события `onMatch` и `onReplace` объекта `PerlRegex1`. В событии `onReplace` мы форматируем нужную часть найденного текста и пользуемся свойством `ReplaceWith`. Это строка, которой будет заменено найденное. Но если она содержит обратные ссылки `\1`, `\2` и т. д., то перед совершением замены надо поменять их на захваченные фрагменты текста методом `ComputeReplacement`.

Читатель может спросить: а что, если после обратной ссылки в заменяемом тексте будет идти десятичная цифра? Как отделить ее от ссылки? Это интересный вопрос, и его надо задать разработчику этого компонента.

Проект находится в подпапке Project3. В листинге 5.3 вы видите текст модуля Unit1.pas.

Листинг 5.3. Пример поиска и выделения URL в тексте

```
unit Unit1;

interface

uses
  Windows, SysUtils, Forms, Classes, Controls, StdCtrls, PerlRegEx;

type
  TForm1 = class(TForm)
    PerlRegEx1: TPerlRegEx;
    Button1: TButton;
    text: TMemo;
    Label1: TLabel;
    Label2: TLabel;
    out: TMemo;
    Label3: TLabel;
    re: TMemo;
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure PerlRegEx1Replace(Sender: TObject;
                                var ReplaceWith: String);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation
{$R *.dfm}
```

```
var
 protocol,host,subdom,zone,port,tail,wb: string;

procedure TForm1.Button1Click(Sender: TObject);
begin
with PerlRegEx1 do
 begin
 Subject:=text.Text;
 Start:=1;
 Stop:=Length(Subject);
 RegEx:=re.Text;
 ReplaceAll;
 out.Text:=Subject;
 end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
PerlRegEx1.State:=[];

wb:='(?![A-Za-z0-9])';
protocol:='(?:(?=[FfHh])(?i:https?+|ftp)://)';
host:='(?x:(?>[A-Za-z0-9]{1,63}\.)'+
           '(?>[A-Za-z0-9]'+
            '(?>[-A-Za-z0-9]{0,62})\.'+
           ')*)';
subdom:='(?:(?>[A-Za-z0-9](?:[-A-Za-z0-9]{0,61}[A-Za-z0-9])?)\.)+';
zone:='(?ix:(?(?=[a-z]{3}'+wb+')(?>com|net|org|edu|biz|gov|int|mil)|'+
                '(?(?=[a-z]{2}'+wb+')[a-z]{2}|'+
                 '(?(?=[a-z]{4}'+wb+')(?>info|aero|name)|'+
                  '(?(?=[a-z]{6}'+wb+')museum|(?!)'+
                  ')'+
                 ')'+
                ')'+
             ')'+
             '(?>\.[a-z]{2}'+wb+')?)';
```

```
port:='(?::\d{1,5}'+wb+')';
tail:='(?:[/?](?>[^.,"'''<>()\[\]{}\s\x7F-
\xFF]*)(?:(?>[.,?]+)(?:[^"'''<>()\[\]{}\s\x7F-\xFF]+))*(?<![,.?!-]))';


text.Text:=
'URLs:'#13#10+
'Ftp://a.com/AAa'#13#10+
'Look at:aaa.Museum.'#13#10+
'http://www.proxy.com:80@www.site.com/'#13#10+
'http://proxy.com:80@site.com/'#13#10+
'http://proxy.com@site.com/'#13#10+
'aAaa.com.au.rr.ggg'#13#10+
'Zwww.Yahoo.co.uk'#13#10+
'+forum.abcd.de'#13#10+
'www.Abc.eu'#13#10+
'-123.123.123.1234.com/?q=aaa'#13#10+
'http://Abc.Tk'#13#10+
'Ahttp://www.Abc.pt/AAa'#13#10+
'http://abc.au/query/vid.cam.dig/sony.dcrhc15.htm#full_image'#13#10+
'+.Www.old-avto.tk'#13#10+
''#13#10+
'NOT URLs:'#13#10+
'aaa.museumm'#13#10+
'http://aaa.museumm,'#13#10+
'http://-aaa.com'#13#10+
'www._aaa.com'#13#10+
'www.aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.com
'#13#10;


re.Text:='((?>('+protocol+')(?(2)(?>'+host+zone+')|'+host+zone+')(?![A-
Za-z0-9])|(?<![A-Za-z0-9_@-])(?<!\.(?!(?i:www)))'+subdom+zone+'(?![A-Za-
z0-9_.-]*@))(?>(?>'+port+'?(?>@'+host+zone+'(?![A-Za-z0-9_.-
]*@))?)?))('+tail+'?)';
end;


procedure TForm1.PerlRegEx1Replace(Sender: TObject;
                                   var ReplaceWith: String);
```

```
var
 protocol: string;

begin
with PerlRegEx1 do
 begin
 if Length(SubExpressions[2]) = 0 then protocol:='http://' else
  protocol:='';
 Replacement:='<a href="'+protocol+LowerCase(SubExpressions[1])+
   '\3" target=_blank>\1\3</a>';
  ReplaceWith:=ComputeReplacement;
 end;
end;

end.
```

Запустите пример и щелкните по кнопке **OK**. В поле вывода вы увидите преобразованный текст. Вы можете редактировать текст в поле ввода и смотреть результат его обработки. В поле **Regex** записано наше регулярное выражение после подстановки в него строк с подшаблонами. Оно имеет длину 1475 символов! Вряд ли мы составили бы такое выражение без разбивки его на подшаблоны.

5.2.5. Поиск вложенных конструкций

Поиск конструкций произвольного уровня вложенности давно будоражил умы всего человечества. Например, арифметическое выражение может иметь скобки произвольного уровня вложенности, оператор вызова функции в языке Pascal тоже может иметь скобки с произвольным уровнем вложенности: ведь среди параметров функции может быть обращение к функции и т. д. В HTML-документе в ячейке таблицы может находиться своя таблица, а в ее ячейке, в свою очередь, — еще таблица. Такие рекурсивные конструкции невозможно разобрать лишь с помощью вышеописанного инструментария языка PCRE. Для этого надо иметь рекурсивные регулярные выражения. Разработчик языка PCRE создал для этой цели несколько возможностей.

- Специальная конструкция `(?R)` является рекурсивным вызовом всего регулярного выражения. Если при разборе текста система поиска соответствия доходит до этой конструкции в шаблоне, то вместо нее подставляется все регулярное выражение.

- Специальная конструкция `(?число)`, где *число* должно быть больше нуля, заменяет `?число` на подшаблон, стоящий в захватывающих скобках соответствующего номера. Скобки вокруг `?число` являются группирующими. Эта конструкция используется, если нужно рекурсивно подставлять не все регулярное выражение, а его часть.
- Специальная конструкция `(?P>имя)` заменяет `?P>имя` на именованный *подшаблон* с именем *имя*: `(?P<имя>подшаблон)`. Это позволяет вести учет захватывающих скобок по их именам.
- Специальная конструкция `(R)` является условием, которое используется в условных подшаблонах. Оно имеет значение "истина", если в этот момент мы находимся внутри рекурсивного вызова. Иначе оно имеет значение "ложь". Автор довольно скупо описывает эти свои новшества в документации, но здесь, по-видимому, речь идет о рекурсивном вызове как всего регулярного выражения, так и любого подшаблона в захватывающих скобках. В каждом из этих случаев внутри вызова логическая конструкция `(R)` становится истиной.

Теперь рассмотрим примеры использования этих конструкций.

Регулярное выражение

```
\(((?[^()]++|(?R))*)\)
```

соответствует любому числу сбалансированных круглых скобок. Пусть мы имеем арифметическое выражение

```
a*(b+c-(d-e)+f)/h
```

Если применить к нему это регулярное выражение, то оно будет соответствовать ближайшему слева фрагменту в сбалансированных скобках вместе с этими скобками:

```
(b+c-(d-e)+f)
```

В первой паре захватывающих скобок окажется этот фрагмент, но без внешних скобок:

```
b+c-(d-e)+f
```

Во второй паре захватывающих скобок окажется фрагмент

```
+f
```

Это то, что захватил подшаблон

```
[^()]++
```

в последний раз.

Разберем логику работы этого регулярного выражения.

1. Вначале устанавливается соответствие открывающей скобки с третьей позиции заданной строки. Затем открывается первая пара захватывающих скобок, за ней — вторая.
2. Работает первая альтернатива `[^()]++`, которая поглощает фрагмент `b+c-`, и он попадает в первую и вторую пару захватывающих скобок.
3. Вторая пара захватывающих скобок закрывается.
4. Квантификатор `*` заставляет повториться вторую пару захватывающих скобок, и она опять открывается, начиная захват текста. То, что в ней было до этого, уничтожается. Но в этот раз первая альтернатива не находит соответствия, и пробуется вторая альтернатива `(?R)`. Она находит соответствие. То есть разбор начинается с начала всего регулярного выражения. Но внутри рекурсивных вызовов захватывающие скобки работают только как группирующие, ничего не захватывая, иначе возникла бы полная неразбериха, кому что захватывать.
5. Рекурсивный вызов начинает поиск соответствия с начала всего регулярного выражения, и вторая открывающая скобка в тексте соответствует первой открывающей скобке в шаблоне. Эта скобка попадает в обе пары захватывающих скобок.
6. Работает первая альтернатива, захватывающая фрагмент `d-e`, который тоже попадает в обе пары захватывающих скобок.
7. Внутри рекурсивного вызова происходит закрытие второй пары захватывающих скобок, но это не оказывает эффекта на захват текста скобками. На этот раз обе альтернативы не находят совпадения, начиная от первой закрывающей скобки в тексте. Происходит выход за квантификатор внутри рекурсивного вызова, за которым стоит закрывающая захватывающая скобка, тоже не оказывающая влияния, а за ней — литеральная закрывающая скобка, замаскированная обратной косой чертой, которая совпадает со второй закрывающей скобкой в тексте. На этом шаблон заканчивается, и происходит выход из рекурсивного вызова. Дальше идет текст `+f)/h`, а в шаблоне мы находимся на второй закрывающей захватывающей скобке. Вторая пара скобок захватила текст `(d-e)`.
8. Квантификатор заставляет повториться вторую пару захватывающих скобок, и ее содержимое опять стирается. В этот раз работает первая альтернатива, которая захватывает текст `+f`. Он попадает в обе пары захватывающих скобок. Вторая пара скобок закрывается, захватив текст `+f`.
9. Квантификатор `*` заставляет повториться вторую пару захватывающих скобок, но повтора не происходит, т. к. обе альтернативы не соответствуют тексту, начинающемуся с закрывающей скобки.

10. Происходит выход за квантификатор `*` и закрытие первой пары захватывающих скобок, которая захватила текст, находящийся внутри внешней пары скобок: `b+c-(d-e)+f`.
11. Литеральная закрывающая скобка в шаблоне соответствует последней закрывающей скобке в тексте. На этом шаблон заканчивается, он совпал с фрагментом `(b+c-(d-e)+f)`.

Уберите из текста фрагмент `+f` и увидите, что вторая пара захватывающих скобок получит значение `(d-e)`, т. е. то, что она захватила во второй раз.

Давайте слегка видоизменим это регулярное выражение, заменив первый плюс на звездочку:

```
\((([^()]*+|(?R))*)\)
```

Если применить его к этому же тексту, то мы увидим, что все регулярное выражение и первая пара захватывающих скобок совпадают с теми же фрагментами текста, что и раньше, а вторая пара скобок совпадает с пустой строкой. Почему так происходит? Как я уже говорил, подшаблон, который может иметь ноль повторов, совпадает всегда. Поэтому здесь до рекурсивного вызова дело не доходит. Вначале совпадает литеральная открывающая скобка, затем первая альтернатива поглощает текст `b+c-`. Происходит закрытие второй пары захватывающих скобок, квантификатор `*` заставляет повториться конструкцию выбора. Опять совпадает первая альтернатива, но в этот раз с пустым фрагментом. И в этом месте происходит любопытное дело: механизм поиска совпадения видит, что произошло совпадение с пустым фрагментом, т. е. с ничем, и он перемещает текущую позицию в тексте на один символ, иначе будет бесконечный цикл совпадения с пустотой. А если в этом месте текст заканчивается, то этот механизм переходит к следующему подшаблону в регулярном выражении. Благодаря этому, например, шаблон

```
a((a*)*)*
```

благополучно соответствует строке

```
a
```

Поэтому альтернатива `[^()]*+` будет продолжать совпадать до конца текста. Но после всего текста в шаблоне остается еще закрывающая скобка, которой не с чем совпасть. И здесь благодаря перебору с возвратами квантификатор `*` будет отдавать захваченные символы вплоть до последней закрывающей скобки, и после ее отдачи она совпадет с закрывающей литеральной скобкой регулярного выражения.

На самом деле в этом шаблоне не надо заменять плюс на звездочку, т. к. он и без того совпадает с парой скобок, внутри которой ничего нет:

```
()
```

а также с такими же вложенными парами:

```
(()((()))()()
```

Внешний квантификатор `*` делает применение альтернативы не обязательным, и в этом случае происходит совпадение с пустой парой скобок. Еще замечу, что совпадение выражения происходит от первой правильно сбалансированной открывающей скобки до соответствующей ей закрывающей скобки. Например, в тексте

```
(a(b)
```

произойдет совпадение с фрагментом

```
(b)
```

Специальная конструкция `(?`*число*`)` и подобная ей `(?P>`*имя*`)` работают аналогично `(?R)`, но вместо всего регулярного выражения делают подстановку подшаблона с соответствующим номером захватывающих скобок. Это видно на следующем примере. Имеем регулярное выражение:

```
(a|b)c(?1){2}c
```

и текст

```
acaac
```

Альтернатива совпадает с буквой `a`, затем совпадает буква `c`, происходит вызов первой пары захватывающих скобок два раза, и это совпадает с буквами `aa`, затем совпадает буква `c`. В этом шаблоне вместо `(?1)` подставляется альтернатива `(a|b)`, поэтому в месте подстановки происходит совпадение как с `a`, так и с `b`. Благодаря этому все регулярное выражение совпадает со всем текстом также в вариантах

```
acabc
acbac
acbbc
bcaac
bcabc
bcbac
bcbbc
```

Сравните это с шаблоном

```
(a|b)c\1{2}c
```

который совпадет со всем текстом лишь в двух вариантах

```
acaac
bcbbc
```

По поводу конструкции `(R)`, используемой в качестве условия в условных шаблонах, которую не надо путать с рекурсивным вызовом `(?R)`, автор приводит такой пример: пусть нужен шаблон, совпадающий со сбалансированными угловыми скобками произвольного уровня вложенности, но во всех вложенных угловых скобках должны находиться одни лишь цифры. Этот шаблон выглядит так:

```
<(?:(?(R)\d++|[^<>]*+)|(?R))*>
```

Сначала происходит совпадение с первой открывающей угловой скобкой. Затем стоит альтернативный шаблон, состоящий из двух альтернатив:

```
(?(R)\d++|[^<>]*+)
```

и

```
(?R)
```

Вначале условие `(R)` ложно, т. к. мы не внутри рекурсивного вызова `(?R)`. Поэтому в условном шаблоне работает ветка

```
[^<>]*+
```

которая захватывает все кроме угловых скобок. Если встречается угловая скобка, то во внешней конструкции выбора работает альтернатива `(?R)`, которая соответствует тому же, чему и все регулярное выражение, но внутри нее условие `(R)` становится истиной, и работает ветка `\d++`, которая соответствует лишь цифрам. Благодаря этому во вложенных угловых скобках совпавшего текста могут быть только цифры.

Условие `(R)` также становится истиной, если мы находимся внутри рекурсивного вызова захватывающей пары скобок. Например, шаблон

```
^(a(?(R)c)b)(?1)$
```

совпадает с текстом

```
abacb
```

Вначале происходит совпадение с `a`. Условие `(R)` ложно, поэтому подшаблон `c` не подставляется и происходит совпадение с `b`. Затем происходит вызов первой пары захватывающих скобок, условие `(R)` становится истиной и буква `c` подставляется. Скобки захватывают фрагмент `ab`, т. е. внутри вызова `(?1)` вместо `?1` подставляется содержимое этих скобок.

Со строкой

```
abab
```

это выражение не совпадает, что и доказывает работу условия `(R)` с вызовами `(?`*номер*`)` захватывающих скобок. Если внутри первой пары захватываю-

щих скобок имеются еще захватывающие скобки, то внутри вызовов (?1) они не производят захвата текста. Например, шаблон

```
^(a(?(R)c)(\d))(?1)$
```

совпадает со строкой

```
a1ac2
```

При этом первая пара захватывающих скобок захватывает a1, а вторая — цифру 1.

5.2.6. Лексический разбор текста

Условие \G, которое при вызовах метода MatchAgain заставляет совпадать текст с конца последнего совпадения, позволяет делать лексические анализаторы текста. При первом вызове Match якорь \G действует как якорь \A.

Предположим, что мы делаем разбор текста на имена, которые состоят из латинских букв, чисел и символов перевода строк. Мы должны последовательно извлекать из текста эти объекты, называть и выводить их. Если встречается что-то, что не соответствует синтаксису ни одного объекта, то это будет называться unknown (неизвестно). Регулярное выражение будет начинаться с \G и представлять собой высокоуровневую конструкцию выбора:

```
(?x)\G
(?:[\000-\011\013-\040]++|      # все от \0 до пробела кроме \n
   (?P<name>\b[a-zA-Z]++\b)|    # имя
   (?P<number>[+-]?+\d++\b)|    # число
   (?P<newline>\n)|             # \n
   (?P<unknown>\S++)            # все остальное
)
```

Чтобы отделить первую альтернативу из выбора от \G, необходимо конструкцию выбора взять в скобки.

Этот пример, сделанный на основе проекта из папки Project1, находится в папке Project4. В листинге 5.4 вы видите текст модуля Unit1.pas.

Листинг 5.4. Лексический разбор текста

```
unit Unit1;

interface

uses
  Windows, SysUtils, Forms, Classes, Controls, StdCtrls, PerlRegEx;
```

```
type
  TForm1 = class(TForm)
    PerlRegEx1: TPerlRegEx;
    Button1: TButton;
    re: TEdit;
    text: TMemo;
    Label1: TLabel;
    Label2: TLabel;
    out: TMemo;
    Label3: TLabel;
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

const
 enames: array[1..4] of string = ('name','number','newline','unknown');

procedure TForm1.Button1Click(Sender: TObject);
var i,j: integer;

begin
with PerlRegEx1 do
 begin
 Subject:=text.Text;
 Start:=1;
```

```
 Stop:=Length(Subject);
 RegEx:=re.Text;
 out.Lines.Clear;
 j:=0;
 if Match then
  begin
  for i:=1 to SubExpressionCount do
   if SubExpressionOffsets[i] > 0 then
    with out.Lines do
     begin
     Append('Found '+enames[i]);
     if enames[i] <> 'newline' then
      Strings[j]:=Strings[j]+': '+SubExpressions[i];
     Inc(j);
     break;
     end;
  while MatchAgain do
   with out.Lines do
    for i:=1 to SubExpressionCount do
     if SubExpressionOffsets[i] > 0 then
      begin
      Append('Found '+enames[i]);
      if enames[i] <> 'newline' then
       Strings[j]:=Strings[j]+': '+SubExpressions[i];
      Inc(j);
      break;
      end;
  end;
 end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
 PerlRegEx1.State:=[];
end;

end.
```

В этом модуле создается массив строк — имен именованных подшаблонов:

```
enames: array[1..4] of string = ('name','number','newline','unknown');
```

После вызова методов `Match` и `MatchAgain` мы узнаем номер совпавшей захватывающей скобки по признаку, что `SubExpressionOffsets[i]` для этой пары скобок будет больше нуля. Мы из массива `enames` выводим имя этой пары скобок как `enames[i]`, а также сам захваченный ими фрагмент, если он не является символом перевода строки. Если программе задать текст

```
123 abc
-234 def
$@ xyz
```

то она выведет такой результат:

```
Found number: 123
Found name: abc
Found newline
Found number: -234
Found name: def
Found newline
Found unknown: $@
Found name: xyz
Found newline
```

5.2.7. Лексический разбор с несколькими шаблонами

В следующем примере рассмотрим применение нескольких регулярных выражений для последовательного извлечения из текста лексических единиц, соответствующих этим выражениям. В сложных случаях не всегда удобно держать все шаблоны в виде одной большой альтернативы, как в предыдущем примере.

Этот пример, который находится в каталоге Project6, производит разбор заданного текста HTML (простой случай). В начале каждого регулярного выражения, которое используется для разбора документа, стоит якорь `\G`, благодаря чему в следующей итерации цикла поиска соответствия регулярное выражение ищет совпадение от конца предыдущего совпадения, ничего не пропуская. Для корректной работы этой схемы создана вспомогательная функция `Match1`, которая получает в качестве параметра регулярное выражение, которое она будет применять. Текущая позиция в тексте, с которой будет применяться следующее регулярное выражение, хранится в перемен-

ной start1 и корректируется при каждом совпадении, потому что свойство Start не может стоять в правой части оператора присваивания.

Функция Match1 явно вызывает метод Compile для трансляции заданного ей шаблона, потому что методы Match и MatchAgain обращаются к Compile, если это регулярное выражение еще не было оттранслировано, и при этом свойству Start присваивается единица, что не нужно. Фрагмент кода

```
Compile;
Start:=start1;
if MatchAgain then Exit;
```

сохраняет внутри метода MatchAgain присвоенное значение параметру Start.

В этом примере программе разбора задается текст

```
<html>
<body>
<a href=www.gameintellect.com><img src=/img/gibutton.gif width=88
height=31>Our button</a>
&lt;Flash Games&gt;
</body>
</html>
```

В результате программа выдает такие сообщения:

```
Found a tag: <html>
Found a tag: <body>
Found a tag: <a href=www.gameintellect.com>
Found a tag: <img src=/img/gibutton.gif width=88 height=31>
Found a number/word: Our
Found a number/word: button
Found a tag: </a>
Found a HTML entities: &lt;
Found a number/word: Flash
Found a number/word: Games
Found a HTML entities: &gt;
Found a tag: </body>
Found a tag: </html>
  END of parsing.
```

В листинге 5.5 представлен текст модуля Unit1.pas из каталога Project5.

Листинг 5.5. Последовательный лексический разбор по нескольким шаблонам

```
unit Unit1;

interface

uses
  Windows, SysUtils, Forms, Classes, Controls, StdCtrls, PerlRegEx;

type
  TForm1 = class(TForm)
    PerlRegEx1: TPerlRegEx;
    Button1: TButton;
    text: TMemo;
    Label2: TLabel;
    out: TMemo;
    Label3: TLabel;
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);

var
 need_close_a: boolean;
 start1: integer;
```

```
function Match1(re: string): boolean;
begin
Result:=TRUE;
with PerlRegEx1 do
 begin
 RegEx:=re;
 Compile;
 Start:=start1;
 if MatchAgain then Exit;
 end;
Result:=FALSE;
end; {Match1}

begin
need_close_a:=FALSE;
with PerlRegEx1 do
 begin
 Subject:=text.Text;
 start1:=1;
 Stop:=Length(Subject);

 // Ищем соответствие, пока не достигнем конца текста
 while (not Match1('\G\z')) do

  // Ищем теги html, body, /html и /body
  if Match1('\G(</?+html[^>]*+>|</?+body[^>]*+>)\s*+') then
   begin
   out.Lines.Append('Found a tag: '+SubExpressions[1]);
   Inc(start1,SubExpressionLengths[0]);
   end else

    // Ищем тег a
    if Match1('\G(<a\s++[^>]++>)\s*+') then
     begin
     out.Lines.Append('Found a tag: '+SubExpressions[1]);
     Inc(start1,SubExpressionLengths[0]);
```

```
// Если предыдущий тег a не был закрыт, то это ошибка
if need_close_a = TRUE then
 out.Lines.Append('  ERROR: <a tag was not closed!');

// Будет нужен закрывающий тег /a
need_close_a:=TRUE;
end else

 // Ищем тег img
 if Match1('\G(<img\s++[^>]++>)\s*+') then
  begin
  out.Lines.Append('Found a tag: '+SubExpressions[1]);
  Inc(start1,SubExpressionLengths[0]);
  end else

   // Ищем тег /a
   if Match1('\G(</a[^>]*+>)\s*+') then
    begin
    out.Lines.Append('Found a tag: '+SubExpressions[1]);
    Inc(start1,SubExpressionLengths[0]);

    // Если был закрывающий тег /a, то это ошибка
    if need_close_a = FALSE then
     out.Lines.Append('  ERROR: <a tag was not opened!');

    // Не нужен закрывающий тег /a
    need_close_a:=FALSE;
    end else

     // Ищем HTML entities типа &gt;, &copy; &243; &#1234;
     if Match1('\G(&#?+\w{1,5}+;)\s*+') then
      begin
      out.Lines.Append('Found a HTML entities: '+SubExpressions[1]);
      Inc(start1,SubExpressionLengths[0]);
      end else
```

```
            // Ищем слово или число
            if Match1('\G(\w++)\s*+') then
             begin
             out.Lines.Append('Found a number/word: '+SubExpressions[1]);
             Inc(start1,SubExpressionLengths[0]);
             end else

              // Ищем все остальное
              if Match1('\G(\S++)\s*+') then
               begin
               out.Lines.Append('Found unknown: '+SubExpressions[1]);
               Inc(start1,SubExpressionLengths[0]);
               end;

 // В конце документа тег /a должен быть закрыт
 if need_close_a = TRUE then out.Lines.Append(
  'ERROR: a tag was not closed!');
 out.Lines.Append('  END of parsing.')
 end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
 PerlRegEx1.State:=[];
end;

end.
```

Внутри условных операторов могут стоять проверки параметров тегов. А поиск совпадения для вложенных конструкций типа таблиц мы уже делали.

5.2.8. Обработка текста по заданной грамматике

В конце рассмотрим пример разбора и вычисления арифметических выражений по заданной грамматике для этих выражений. Все проходили в школе грамматику естественного языка, на котором разговаривают. У языков

программирования, таких как Pascal, тоже есть свои грамматики. Но поскольку это формальные языки, то и грамматики для них называют формальными, потому что они подчиняются строгим формальным правилам, благодаря чему предложения этих языков трактуются однозначно. (Сравните это, например, с такими фразами естественного языка, как "холм закрыл лес" и "голубые ели".) Существует несколько видов формальных грамматик, мы будем использовать грамматику *LL(1)*. В ней предложение языка разбирается с левой рекурсией, и для выбора очередного правила разбора достаточно знать один следующий символ предложения, который называется направляющим символом. Предложение — это и есть сама программа. Разбор по правилам грамматики устанавливает, соответствует ли это предложение правилам грамматики. Попутно в ходе разбора это предложение может интерпретироваться: выполняться программа, как это делает интерпретатор Basic, или генерироваться машинный код, как это делает транслятор с языка Pascal. Для этой цели в правила грамматики вставляются действия (точнее, их номера). Как только в ходе разбора программы интерпретатор подошел в правиле грамматики к номеру действия, он это действие выполняет. Для этого разбора нужен лексический анализатор и стек (магазин) для запоминания результатов во время работы. Вот, собственно, и вся теория, которую нам надо знать. Регулярные выражения позволяют быстро написать и отладить такую программу, беря на себя мелочи извлечения из текста лексических единиц языка.

В данном пункте мы напишем программу, которая вычисляет арифметические выражения, такие как это:

```
3* (13,2 - 1e0*(1+2) )
```

В результате должен появиться результат `30,6`. Как видим, можно использовать целые числа и числа с плавающей точкой с экспонентой и без. Вообще-то в этом примере имеется плавающая запятая, и это зависит от установок на вашем компьютере в панели управления (языки и региональные стандарты). У кого-то разделителем целой и дробной частей будет точка. Программа будет это учитывать. Delphi имеет для этого переменную `DECIMALSEPARATOR`, которая хранит этот самый разделитель.

Займемся формальной грамматикой этого небольшого языка арифметических выражений. Грамматика — это просто совокупность правил, которые поясняют свою левую часть. Начинаются правила с самого общего понятия, для нас это — арифметическое выражение. Затем последовательно составляются правила для очередных конструкций языка, пока не доходят до самых мелких конструкций, которые целиком состоят из символов языка. У нас символами языка являются:

число `+ - * / ( )`

Лексический анализатор будет последовательно извлекать из текста эти символы и передавать их синтаксическому анализатору на дальнейшую обработку. Синтаксический анализатор будет состоять из подпрограмм по одной для каждого правила вывода. Такой разбор предложения языка называется *нисходящим рекурсивным разбором*.

Если читатель напуган этими научными словечками, то спешу его успокоить: ничего особо сложного здесь нет, и составлять LL(1)-грамматику довольно просто. Мы начнем с правила для всего выражения и постепенно создадим правила для всего языка. Итак, на входе мы имеем арифметическое выражение, которое обозначим через `EXPR`. Из каких наикрупнейших компонентов оно состоит? Такое выражение можно представить как сумму так называемых *термов*, разделенных знаками плюс или минус:

```
EXPR = TERM + TERM - TERM + ...
```

Внутри себя термы не имеют знаков плюс и минус верхнего уровня, т. е. терм нельзя представить как сумму чего-то еще. Если мы умеем вычислять термы, то вычисление выражения является простой задачей: ведь скобок здесь нет, а операции выполняются слева направо.

Из чего же состоят термы? Они состоят из множителей, разделенных операциями `*` или `/`. С точки зрения грамматики `*` и `/` — это просто разделители множителей. Итак,

```
TERM = MULT / MULT * MULT / MULT / MULT * MULT / ...
```

Если мы умеем вычислять множители, то вычисление терма тоже не составляет труда: ведь скобок здесь нет, а операции выполняются слева направо.

Если вы хотите ввести более приоритетную операцию (например, возведение в степень), то можете аналогично продолжить и написать правило для множителя. Но операции возведения в степень выполняются справа налево, а это немного усложняет программу разбора: ей потребуется забегать вперед. Возможно, поэтому в языках программирования, как правило, не вводится такая операция.

Из чего же состоит множитель? И где здесь скобки, которые были в примере выражения? А вот в этом множителе они и спрятаны: множитель — это либо число, либо выражение в скобках:

```
MULT = CONST
```

а так же

```
MULT = ( EXPR )
```

Как видим, наши правила вывода имеют рекурсию: мы начали с выражения, а закончили тем, что минимальный элемент этого выражения может быть опять выражением в скобках. Да, в этом и сила. LL(1)-грамматика допускает рекурсию, но правая часть правила не может начинаться с конструкции,

которая встречалась раньше, в нашем случае это EXPR. Перед EXPR должно что-то стоять, у нас это скобка.

А как записать правило вида

```
EXPR=TERM+TERM-TERM+...
```

ведь многоточие — это неформальное описание? Выходят из этого положения довольно просто: выделяют из выражения первую составляющую конструкцию, а к ней добавляют продолжение выражения:

```
EXPR -> TERM CONT_EXPR
```

Это означает, что выражение состоит из терма и продолжения выражения. Теперь надо создать правила для продолжения выражения. Вот они:

```
CONT_EXPR -> + TERM CONT_EXPR
          -> - TERM CONT_EXPR
          ->
```

"А что там в третьей строке за правило? — может спросить читатель. — В типографии для него не хватило краски?" Нет, все в порядке: это пустое правило, оно означает, что продолжение выражения может и не быть, а формально говоря, оно может быть пустым. Мы видим, что из этих трех правил выводятся выражения с любым числом термов, разделенных знаками + и -. Также терм в выражении может быть единственным.

Пока получается довольно ловко, а что же дальше? Дальше мы совершенно аналогично составляем правила для терма:

```
TERM -> MULT CONT_TERM
CONT_TERM -> * MULT CONT_TERM
          -> / MULT CONT_TERM
          ->
```

ну, и в конце:

```
MULT -> CONST
     -> ( EXPR )
```

Между лексическими элементами языка допускается любое число пробельных символов от нуля и больше. Лексический анализатор будет их игнорировать. Еще в конце выражения потребуем наличия символа =, который и будет обозначать то, что выражение закончилось. Он играет роль END с точкой в конце Pascal-программы. В этом случае придется ввести еще одно правило для арифметического выражения со знаком равенства. Собирая все правила вместе, получим такую небольшую грамматику:

```
EXPR_EQ   -> EXPR =
EXPR      -> TERM CONT_EXPR
```

```
CONT_EXPR -> + TERM CONT_EXPR
          -> - TERM CONT_EXPR
          ->
TERM      -> MULT CONT_TERM
CONT_TERM -> * MULT CONT_TERM
          -> / MULT CONT_TERM
          ->
MULT      -> CONST
          -> ( EXPR )
```

Вообще-то в этой записи все названия кроме `CONST` обычно берутся в угловые скобки, чтобы отличить символы языка от более общих понятий.

Грамматика языка Turbo Pascal занимает несколько десятков килобайтов.

Вручную можно применить эти правила к различным арифметическим выражениям, и вы увидите, что они разбирают такие выражения от одиночного числа до сложнейших конструкций с произвольным уровнем вложенности скобок. Но правда, эта грамматика не допускает унарного плюса и минуса. Этот недостаток можно исправить, например, форматированием строки с выражением: если перед плюсом или минусом нет числа или закрывающей скобки, то вставляем перед ним число `0`.

Наш будущий пример по ходу разбора арифметического выражения должен будет его вычислять. Для этого внутрь грамматических правил надо вставить действия. Эти действия простые, их будет всего пять. Они делятся на запись числа в стек, выполнение операции над двумя верхними элементами стека с выталкиванием их из стека и записью результата этой операции в стек. Если выражение составлено правильно, то в конце разбора с текущей позиции строки, в которой было задано арифметическое выражение, мы будем иметь совпадение с шаблоном `\G\s*+\z`, а в стеке будет находиться единственное число — результат вычисления этого выражения. Иначе программа станет фиксировать ошибку и сообщать о ней с выдачей позиции в строке, где она была обнаружена. Кроме того, пример будет также фиксировать переполнение при делении.

Мы видим, что в трех случаях, в которых правил вывода для одного символа несколько, имеется неопределенность, какое правило выбрать. Выбор следующего правила для разбора определяется следующим символом языка, стоящим в строке. Поэтому такие символы называются *направляющими*. Для каждого варианта правила, если их несколько, эти символы должны быть различными. Далее вы видите эту грамматику с направляющими символами. Их нетрудно установить для каждого правила, если продолжать его разбор

до тех пор, пока в правой части первым не встретится символ языка (литерал).

```
1. EXPR_EQ   -> EXPR =                  CONST (
2. EXPR      -> TERM CONT_EXPR          CONST (
3. CONT_EXPR -> + TERM CONT_EXPR        +
4.           -> - TERM CONT_EXPR        -
5.           ->                         ) =
6. TERM      -> MULT CONT_TERM          CONST (
7. CONT_TERM -> * MULT CONT_TERM        *
8.           -> / MULT CONT_TERM        /
9.           ->                         + - ) =
10. MULT      -> CONST                  CONST
11.          -> ( EXPR )                (
```

Теперь будем разбираться, как найдены эти направляющие символы.

1. Для правил 3, 4, 7, 8, 10 и 11 их можно записать сразу.
2. Правило 6 начинается с `MULT`, поэтому имеет те же направляющие символы, что правила для `MULT` (10 и 11).
3. Правило 2 начинается с `TERM`, поэтому имеет те же направляющие символы, что и правило 6.
4. Правило 1 начинается с `EXPR`, поэтому дублируем в него эти символы из правила 2.
5. С каких символов может начинаться правая часть правила 5? Если `CONT_EXPR` отсутствует, это символы, которые следуют за `CONT_EXPR` (в этом случае они называются символами-последователями). Надо просмотреть остальные правила, в которые входит `CONT_EXPR`. Это правило 2. Из него видно, что за `CONT_EXPR` идут те же символы, что и за `EXPR`. Из правила 1 мы видим, что за `EXPR` может идти `=`, а из правила 11 — что после `EXPR` может стоять `)`. Таким образом, символы-последователи для правила 5 найдены.
6. Найдем символы-последователи для правила 9. Если `CONT_TERM` отсутствует, то смотрим все остальные правила, в которые входит `CONT_TERM`. Это правило 6. Надо найти все символы, которые идут после `TERM`. Смотрим все правила, в правую часть которых входит `TERM`. Это правила 2, 3 и 4. Во всех них после `TERM` идет `CONT_EXPR`. Значит, остается найти направляющие символы для `CONT_EXPR`. Берем их из правил 3, 4 и 5 и записываем в правило 9. Построение грамматики закончено.

Мы видим, что для всех альтернативных правил все направляющие символы и символы-последователи различны. То есть мы построили LL(1)-грамматику.

Остается расставить действия в этой грамматике. Сначала перечислим эти действия и занумеруем их.

1. Записать `CONST` (число) в стек.
2. Выполнить сложение по двум верхним элементам стека с выталкиванием слагаемых из стека и записать результат в стек.
3. Выполнить вычитание по двум верхним элементам стека с выталкиванием операндов из стека и записать результат в стек.
4. Выполнить умножение по двум верхним элементам стека с выталкиванием операндов из стека и записать результат в стек.
5. Выполнить деление по двум верхним элементам стека с выталкиванием операндов из стека и записать результат в стек.

Принцип записи в стек и выполнения операций прост: когда разбор выражения не знает, что надо делать с полученным символом языка, он его записывает в стек. А когда можно выполнить операцию, то он ее выполняет, удаляет операнды из стека и результат записывает в стек.

Поэтому в правило 10 вставим операцию записи константы в стек.

В правилах 7 и 8 после `MULT` у нас на вершине стека будут находиться два числа для этой операции умножения или деления, значит, соответствующую операцию можно выполнить, ведь более приоритетных операций у нас нет, поэтому нас не волнует, что идет справа в `CONT_TERM`. Поставим в эти правила действия 4 и 5.

В правилах 4 и 5 после `TERM` можно вставить действия, соответственно, 2 и 3. В этот момент на вершине в стеке у нас будут два числа — одно это вычисленный предыдущий терм, который стоял перед знаком, а второе — текущий вычисленный терм. Вот она эта грамматика с действиями:

```
1. EXPR_EQ   -> EXPR =                  CONST (
2. EXPR      -> TERM CONT_EXPR          CONST (
3. CONT_EXPR -> + TERM 2 CONT_EXPR      +
4.           -> - TERM 3 CONT_EXPR      -
5.           ->                         ) =
6. TERM      -> MULT CONT_TERM          CONST (
7. CONT_TERM -> * MULT 4 CONT_TERM      *
8.           -> / MULT 5 CONT_TERM      /
9.           ->                         + - ) =
```

```
10. MULT      -> CONST 1              CONST
11.           -> ( EXPR )              (
```

Как я уже писал, каждому символу из левой части правил вывода ставится в соответствие своя процедура, которая осуществляет применение этого правила. Например, процедура для правила 1

```
EXPR_EQ   -> EXPR =                CONST (
```

делает следующее:

- проверяет, что следующий символ в выражении — это `CONST` или `(`;
- если это не так, выводится ошибка и текущая позиция в строке разбора. Если это так, то вызывается процедура разбора `EXPR`, проверяется, что после нее в строке остался знак = (если это не так, то это опять ошибка), далее проверяется, что в стеке осталось ровно одно число (иначе ошибка, которая может быть связана с неправильной вложенностью скобок) и выводится результат вычисления.

Процедура для разбора по правилам

```
3. CONT_EXPR -> + TERM 2 CONT_EXPR     +
4.           -> - TERM 3 CONT_EXPR     -
5.           ->                        ) =
```

работает так:

- если следующий символ в строке — это `+`, то он читается из строки (просто указатель текущего символа увеличивается на число прочитанных байтов), вызывается процедура для правила `TERM`, выполняется действие 2 и вызывается процедура для правила `CONT_EXPR`. После этого — выход из процедуры обработки `CONT_EXPR`;
- если следующий символ в строке — это `-`, то аналогично обрабатывается правило 4;
- если следующий символ в строке — это `)` или `=`, то происходит выход из процедуры обработки `CONT_EXPR`, при этом символ-последователь не извлекается из строки;
- в случае неверного следующего символа выводится сообщение об ошибке.

Возьмем такой пример:

```
2*(3+4)=
```

и разберемся в работе нашей схемы.

1. Вначале работает правило `EXPR_EQ`. Оно вызывает процедуру для применения правила `EXPR`.
2. Правило `EXPR` вызывает правило `TERM`.

3. Правило TERM вызывает правило MULT.
4. Правило MULT извлекает число 2 и записывает его в стек. Происходит возврат в строку 6.
5. В строке 6 далее идет вызов правила CONT_TERM.
6. Правило CONT_TERM извлекает знак * и вызывает правило MULT.
7. Правило MULT видит символ (и выбирает вариант (EXPR) в строке 11. (извлекается из строки и происходит вызов правила EXPR.
8. Правило EXPR вызывает правило TERM.
9. Правило TERM вызывает правило MULT.
10. Правило MULT извлекает из строки число 3 и записывает его в стек. Происходит возврат в строку 6.
11. В строке 6 далее идет вызов CONT_TERM.
12. CONT_TERM видит символ + и выбирает вариант из строки 9. То есть просто делается выход из правила CONT_TERM в правило TERM.
13. Из TERM происходит выход в EXPR.
14. EXPR вызывает правило CONT_EXPR.
15. CONT_EXPR извлекает из строки + и вызывает правило TERM.
16. TERM вызывает правило MULT.
17. MULT извлекает из строки число 4 и записывает его в стек. Происходит возврат в правило TERM.
18. Из TERM происходит вызов CONT_TERM.
19. Из CONT_TERM происходит возврат в TERM.
20. Из TERM делается возврат в CONT_EXPR.
21. CONT_EXPR, которое ранее применило правило 3, выполняет действие 2, извлекает из стека 3 и 4 и записывает 7 в стек, далее вызывает CONT_EXPR.
22. Из CONT_EXPR происходит выход в CONT_EXPR.
23. Из CONT_EXPR происходит выход в EXPR.
24. Из EXPR происходит выход в MUL.
25. MULT выбирает из строки символ) и делает выход в CONT_TERM.
26. CONT_TERM выполняет действие 4, перемножая 2 на 7, извлекает из стека 2 и 7 и записывает в стек 14. Далее происходит выход в CONT_TERM.
27. Из CONT_TERM происходит выход в CONT_TERM.
28. Из CONT_TERM происходит выход в TERM.

29. Из `TERM` происходит выход в `EXPR`.
30. `EXPR` вызывает `CONT_EXPR`.
31. `CONT_EXPR` видит знак = и делает возврат в `EXPR`.
32. `EXPR` делает возврат в `EXPR_EQ`.
33. `EXPR_EQ` выбирает из строки символ =, проверяет, что в строке больше нет символов кроме пробельных, проверяет, что в стеке находится ровно один элемент, и выводит его как результат вычисления выражения.

Вот это мы прокрутили работу правил для небольшого примерчика! Если смотреть изнутри, то даже для такого простого примера происходит много вызовов и возвратов в процедуры. А если представить себя на месте центрального процессора, который все это обрабатывает в машинных кодах, то с ума можно сойти.

Перейдем теперь к самой программе. Она будет использовать следующие глобальный переменные:

```
var
 start1,istack: integer;
 expr: string;
 iserror: boolean;
 stack: array of extended;
```

Здесь:

- `Start1` — это текущая позиция в заданной строке `expr` с арифметическим выражением;
- `istack` — указатель на вершину стека;
- `expr` — строка с арифметическим выражением;
- `iserror` — признак, что произошла синтаксическая ошибка в одном из правил вывода;
- `stack` — открытый массив чисел с плавающей запятой (или точкой). Индексы начинаются с нуля. Играет роль безразмерного стека.

Сначала рассмотрим функцию `Match1`, аналогичную той, что уже применялась в предыдущих примерах. Она представлена в листинге 5.6.

Листинг 5.6. Функция `Match1`

```
{ Поиск соответствия в остатке строки expr[start1] по шаблону re.
  При нахождении соответствия возвращается TRUE, и start1 увеличивается
  на длину найденного, иначе возвращается FALSE }
```

```
function Match1(re: string): boolean;
begin
Result:=TRUE;
with Form1.PerlRegEx1 do
 begin
 RegEx:=re;
 Compile;
 Start:=start1;
 if MatchAgain then
  begin
  Inc(start1,MatchedExpressionLength);
  Exit;
  end;
 end;
Result:=FALSE;
end; {Match1}
```

Эта функция применяет заданное ей регулярное выражение `re`.

Немного коснемся формата чисел. Для перевода чисел из символьного вида в двоичный мы будем использовать функцию `StrToFloat`, которая требует, чтобы у числа была экспонента вида `[eE]-?\d+`. Но записывать в арифметических выражениях все числа в этом виде неудобно, поэтому внутри процедуры для правила `MULT` извлеченное из строки число будет немного форматироваться, чтобы оно имело экспоненту.

В листинге 5.7 приведен текст для процедуры, осуществляющей правило `MULT`.

Листинг 5.7. Процедура для правила `MULT`

```
// Извлечение из позиции expr[start1] множителя/делителя
procedure Parse_mult;
var
 temp: string;

begin
if iserror then Exit;
if Match1('\G\s*+(?:(\d++(?:\Q'+
 DECIMALSEPARATOR+'\E\d++)?+([eE]-?+\d++)?+)|(\())') then
```

```
 with Form1.PerlRegEx1 do
  begin
  if (SubExpressionCount > 0) and (SubExpressionOffsets[1] > 0) then
   // Извлекли число
   begin
   Inc(istack);
   temp:=SubExpressions[1];
   { Если у числа нет экспоненты, то добавляем ее для правильной работы
     функции StrToFloat }
   if SubExpressionCount = 1 then temp:=temp+'e0';
   // Записываем его в стек
   stack[istack]:=StrToFloat(temp);
   end else
    if (SubExpressionCount > 2) and (SubExpressionOffsets[3] > 0) then
     // Извлекли (
     begin
     // Разбираем выражение в скобках
     Parse_expr;
     if iserror then Exit;
     // Извлекаем ) после этого выражения
     if not Match1('\G\s*+\)') then
      begin
      Form1.out.Lines.Append('Error at '+IntToStr(start1));
      iserror:=TRUE;
      istack:=-1;
      Exit;
      end;
     Exit;
     end;
  Exit;
  end;
 Form1.out.Lines.Append('Error at '+IntToStr(start1));
 iserror:=TRUE;
 istack:=-1;
end;
```

Регулярное выражение для этого правила имеет две альтернативы, соответствующие правилам 10 и 11. Вначале подшаблон `\G\s*+` пропускает пробельные символы от текущей позиции в строке. Первая альтернатива захватывает число, а вторая — открывающую скобку. В подшаблоне для числа предусмотрено, что оно может быть целым, с дробной частью и с экспонентой. Разделитель `DECIMALSEPARATOR` обрамляется символами `\Q \E`, чтобы он воспринимался как простой символ, а не как метасимвол (если разделителем будет точка). Впрочем, здесь достаточно было просто обратного слэша перед `DECIMALSEPARATOR`.

Если функция `Match1` нашла совпадение шаблона, то проверяется, какая альтернатива совпала. В частности, если выражение

```
(SubExpressionCount > 0) and (SubExpressionOffsets[1] > 0)
```

истинно, то совпала первая альтернатива, и из строки извлечено число. Это число немного форматируется и записывается в стек. Если совпала вторая альтернатива, то в соответствии с грамматикой передаем управление процедуре `Parse_expr`. Если ничего не совпало, выводим сообщение об ошибке.

Еще разберем процедуру, реализующую правила для `CONT_TERM`. Она приведена в листинге 5.8.

Листинг 5.8. Процедура для правила `CONT_TERM`

```
// Извлечение из позиции expr[start1] продолжения терма
procedure Parse_cont_term;
begin
if iserror then Exit;
if Match1('\G\s*+(?:(\*)|(/)|(?=\+)|(?=-)|(?=\)|(?==)))') then
 with Form1.PerlRegEx1 do
  begin
  if (SubExpressionCount > 0) and (SubExpressionOffsets[1] > 0) then
   // Извлекли *
   begin
   // Обрабатываем множитель
   Parse_mult;
   if iserror then Exit;
   // Выполняем умножение по двум элементам стека
   stack[istack-1]:=stack[istack-1]*stack[istack];
   // Удаляем из стека множитель
   Dec(istack);
```

```
   // Разбираем остаток терма
   Parse_cont_term;
   if iserror then Exit;
   end else
    if (SubExpressionCount > 1) and (SubExpressionOffsets[2] > 0) then
    // Извлекли /
    begin
    // Обрабатываем делитель
    Parse_mult;
    if iserror then Exit;
    // Выполняем деление по двум элементам стека
    try
     stack[istack-1]:=stack[istack-1]/stack[istack];
    except
     // Обрабатываем переполнение при делении
     Form1.out.Lines.Append('Division overflow at '+IntToStr(start1));
     iserror:=TRUE;
     istack:=-1;
     Exit;
    end;
    // Удаляем из стека делитель
    Dec(istack);
    // Разбираем остаток терма
    Parse_cont_term;
    if iserror then Exit;
    end;
  Exit;
 end;
 Form1.out.Lines.Append('Error at '+IntToStr(start1));
 iserror:=TRUE;
 istack:=-1;
end;
```

Здесь регулярное выражение имеет шесть альтернатив. Первая альтернатива извлекает знак умножения, вторая — деления, а остальные четыре альтернативы проверяют символы-последователи для девятой строки с правилами.

Обратите внимание, что символы-последователи не извлекаются из строки с арифметическим выражением.

В случае извлечения символа * вызывается процедура Parse_mult и выполняется умножение по двум верхним элементам стека. Затем вызывается процедура Parse_cont_term в соответствии с грамматикой.

Как видим из этих примеров, программирование правил грамматики является делом техники. Еще замечу, что специальной процедуры для EXPR_EQ я не писал, ее роль выполняет метод Button1Click. В листинге 5.9 представлен текст модуля Unit1.pas из каталога Project6.

Листинг 5.9. Текст модуля Unit1.pas

```
unit Unit1;

interface

uses
  Windows, SysUtils, Forms, Classes, Controls, StdCtrls, PerlRegEx,
  dialogs;

type
  TForm1 = class(TForm)
    PerlRegEx1: TPerlRegEx;
    Button1: TButton;
    text: TMemo;
    Label2: TLabel;
    out: TMemo;
    Label3: TLabel;
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
```

```
implementation

{$R *.dfm}

procedure Parse_expr; forward;

var
 start1,istack: integer;
 expr: string;
 iserror: boolean;
 stack: array of extended;

{ Поиск соответствия в остатке строки expr[start1] по шаблону re.
  При нахождении соответствия возвращается TRUE, и start1 увеличивается
  на длину найденного, иначе возвращается FALSE }
function Match1(re: string): boolean;
begin
Result:=TRUE;
with Form1.PerlRegEx1 do
 begin
 RegEx:=re;
 Compile;
 Start:=start1;
 if MatchAgain then
  begin
  Inc(start1,MatchedExpressionLength);
  Exit;
  end;
 end;
Result:=FALSE;
end; {Match1}

// Извлечение из позиции expr[start1] множителя/делителя
procedure Parse_mult;
var
 temp: string;
```

```
begin
if iserror then Exit;
if Match1('\G\s*+(?:(\d++(?:\Q'+DECIMALSEPARATOR+'\E\d++)?+([eE]-
?+\d++)?+)|(\())') then
 with Form1.PerlRegEx1 do
  begin
  if (SubExpressionCount > 0) and (SubExpressionOffsets[1] > 0) then
   // Извлекли число
   begin
   Inc(istack);
   temp:=SubExpressions[1];
   { Если у числа нет экспоненты, то добавляем ее для правильной работы
     функции StrToFloat }
   if SubExpressionCount = 1 then temp:=temp+'e0';
   // Записываем его в стек
   stack[istack]:=StrToFloat(temp);
   end else
    if (SubExpressionCount > 2) and (SubExpressionOffsets[3] > 0) then
     // Извлекли (
     begin
     // Разбираем выражение в скобках
     Parse_expr;
     if iserror then Exit;
     // Извлекаем ) после этого выражения
     if not Match1('\G\s*+\)') then
      begin
      Form1.out.Lines.Append('Error at '+IntToStr(start1));
      iserror:=TRUE;
      istack:=-1;
      Exit;
      end;
     Exit;
     end;
  Exit;
  end;
```

```
 Form1.out.Lines.Append('Error at '+IntToStr(start1));
 iserror:=TRUE;
 istack:=-1;
end;

// Извлечение из позиции expr[start1] продолжения терма
procedure Parse_cont_term;
begin
if iserror then Exit;
if Match1('\G\s*+(?:(\*)|(/)|(?=\+)|(?=-)|(?=\)|(?==)))') then
 with Form1.PerlRegEx1 do
  begin
  if (SubExpressionCount > 0) and (SubExpressionOffsets[1] > 0) then
   // Извлекли *
   begin
   // Обрабатываем множитель
   Parse_mult;
   if iserror then Exit;
   // Выполняем умножение по двум элементам стека
   stack[istack-1]:=stack[istack-1]*stack[istack];
   // Удаляем из стека множитель
   Dec(istack);
   // Разбираем остаток терма
   Parse_cont_term;
   if iserror then Exit;
   end else
    if (SubExpressionCount > 1) and (SubExpressionOffsets[2] > 0) then
    // Извлекли /
    begin
    // Обрабатываем делитель
    Parse_mult;
    if iserror then Exit;
    // Выполняем деление по двум элементам стека
    try
     stack[istack-1]:=stack[istack-1]/stack[istack];
    except
```

```
     // Обрабатываем переполнение при делении
     Form1.out.Lines.Append('Division overflow at '+IntToStr(start1));
     iserror:=TRUE;
     istack:=-1;
     Exit;
    end;
    // Удаляем из стека делитель
    Dec(istack);
    // Разбираем остаток терма
    Parse_cont_term;
    if iserror then Exit;
    end;
  Exit;
 end;
 Form1.out.Lines.Append('Error at '+IntToStr(start1));
 iserror:=TRUE;
 istack:=-1;
end;

// Извлечение из позиции expr[start1] терма
procedure Parse_term;
begin
if iserror then Exit;
// Разбираем множитель/делитель
Parse_mult;
if iserror then Exit;
// Разбираем остаток терма
Parse_cont_term;
end;

// Извлечение из позиции expr[start1] продолжения выражения
procedure Parse_cont_expr;
begin
if iserror then Exit;
if Match1('\G\s*+(?:(\+)|(-)|(?=\))|(=))') then
 with Form1.PerlRegEx1 do
```

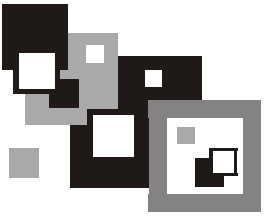
```
begin
if (SubExpressionCount > 0) and (SubExpressionOffsets[1] > 0) then
 // Извлекли +
 begin
 // Получаем терм
 Parse_term;
 if iserror then Exit;
 // Выполняем сложение по двум элементам стека
 stack[istack-1]:=stack[istack-1]+stack[istack];
 // Убираем второе слагаемое
 Dec(istack);
 // Разбираем остаток выражения
 Parse_cont_expr;
 if iserror then Exit;
 end else
  if (SubExpressionCount > 1) and (SubExpressionOffsets[2] > 0) then
  // Извлекли -
  begin
  // Получаем терм
  Parse_term;
  if iserror then Exit;
  // Выполняем вычитание по двум элементам стека
  stack[istack-1]:=stack[istack-1]-stack[istack];
  // Убираем вычитаемое
  Dec(istack);
  // Разбираем остаток выражения
  Parse_cont_expr;
  if iserror then Exit;
  end else
   if (SubExpressionCount > 2) and (SubExpressionOffsets[3] > 0) then
    // Извлекли =
    begin
    if istack <> 0 then
     // В стеке должен остаться только результат
     begin
     Form1.out.Lines.Append('Error at '+IntToStr(start1));
```

```
       iserror:=TRUE;
       istack:=-1;
       Exit;
       end;
      Exit;
      end;
  Exit;
 end;
 Form1.out.Lines.Append('Error at '+IntToStr(start1));
 iserror:=TRUE;
 istack:=-1;
end;

// Разбор выражения, заданного в строке expr
procedure Parse_expr;
begin
if iserror then Exit;
// Разбираем терм
Parse_term;
if iserror then Exit;
// Разбираем остаток выражения
Parse_cont_expr;
end;

// Разбор выражения, заданного в строке expr, и вывод результата
procedure TForm1.Button1Click(Sender: TObject);
begin
out.Lines.Clear;
iserror:=FALSE;
// Инициализируем стек на 100 чисел
SetLength(stack,100);
with PerlRegEx1 do
 begin
 Subject:=text.Text;
 expr:=Subject;
```

```
 // Текущая позиция поиска соответствия
 start1:=1;
 Stop:=Length(Subject);
 // Стек пока пуст
 istack:=-1;
 // Разбираем все выражение
 Parse_expr;
 // Сообщение об ошибке уже выведено?
 if iserror then Exit;
 // В конце в строке expr не должно быть ничего кроме пробелов
 if not Match1('\G\s*+\z') then
  Form1.out.Lines.Append('Error at '+IntToStr(start1)+
   '. May be not matched parens') else
    Form1.out.Lines.Append(FloatToStr(stack[0]));
 end;
// Освобождаем память под стек
Finalize(stack);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
PerlRegEx1.State:=[];
end;

end.
```

Представленный метод синтаксического разбора и вычисления арифметических выражений с помощью LL(1)-грамматики является общим. Он аналогично может применяться для интерпретации или трансляции программ, грамматика которых LL(1). Отличием будет то, что интерпретатор должен будет иметь таблицу меток и идентификаторов и записывать в стек не только числа, но также идентификаторы, метки и знаки сравнения (<, <=, ...).

Приложение



Описание компакт-диска

Все примеры, рассмотренные в книге, можно загрузить с прилагаемого к книге компакт-диска (табл. П1).

Таблица П1. *Структура компакт-диска*

Папка	Содержание
\chapter1	Статьи об исправлении ошибки переполнения при делении; примеры программ задержки времени; исходные тексты и программы для моделирования парадокса; вывода перестановок и сочетаний; вывода гамильтоновых циклов, решения задачи о блохе, игры в ним; вывода позиций для головоломки Full Board; вывода анаграмм и квадратов слов, словарь существительных
\chapter2	Исходный текст и программа для игры Calculator
\chapter3	Исходные тексты и программы для минимальной Windows API Delphi-программы; создания поверхностей DirectDraw; перехода в полноэкранный режим; записи точек на поверхность DirectDraw; переключения поверхностей DirectDraw; исходные файлы и программа для игры Domino Dilemma
\chapter4	Исходные и результирующие файлы для Flash-версии игры Calculator
\chapter5	Delphi-компонент PCRE; исходные и результирующие файлы для примеров

Для использования примеров нужно скопировать соответствующий каталог с компакт-диска на жесткий диск, после чего можно просматривать, редактировать и запускать нужные сценарии.

Замечание

Для компилирования текстов программ к *главе 3* требуются заголовочные файлы DirectX, которые надо скачать с URL **http://www.gameintellect.com/bhv/headers.zip** (29 Кбайт), ввиду того, что не найден их автор и нет его разрешения на публикацию.

Предметный указатель

З

К

Л

М

Н

П

Р

С

Т

Ф

Ш

Э

Я