



СЕРГЕЙ МЕЛЬНИКОВ



DELPHI и TURBO PASCAL НА ЗАНИМАТЕЛЬНЫХ ПРИМЕРАХ



Применяем алгоритмы перебора с возвратами

Находим все анаграммы и квадраты слов

Решаем олимпиадные задачи

Моделируем невероятный парадокс
и просчитываем головоломки

Создаем игры в Turbo Pascal и Delphi, используя
Flash, Windows API и DirectX

Используем библиотеку PCRE в Delphi для
поиска/замены текста, синтаксического
разбора и вычисления арифметических выражений

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.

Для учащихся, студентов и программистов

УДК 681.3.06
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	<i>Екатерина Кондукова</i>
Зам. главного редактора	<i>Игорь Шишигин</i>
Зав. редакцией	<i>Григорий Добин</i>
Редактор	<i>Анна Кузьмина</i>
Компьютерная верстка	<i>Натальи Смирновой</i>
Корректор	<i>Наталья Першакова</i>
Дизайн обложки	<i>Иины Тачиной</i>
Зав. производством	<i>Николай Тверских</i>

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

© Мельников С. В., 2006
© Оформление, издательство "БХВ-Петербург", 2006

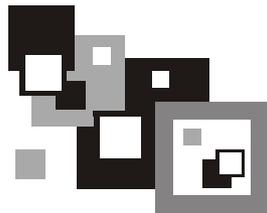
Оглавление

Введение.....	1
Обзор книги.....	1
Глава 1. Практика в Turbo Pascal.....	3
1.1. Преодоление ошибки "Runtime error 200".....	3
1.2. Процедуры задержки времени.....	4
1.3. Перестановки, размещения и сочетания.....	8
1.3.1. Вывод перестановок.....	9
1.3.2. Размещения.....	25
1.3.3. Сочетания и их вывод.....	25
1.4. Моделируем один удивительный парадокс.....	29
1.5. Словесные игры и рекорды.....	34
1.5.1. Анаграммы.....	35
1.5.2. Квадраты слов.....	40
1.6. Перебор с возвратами.....	57
1.6.1. Поиск гамильтоновых циклов.....	58
1.6.2. Блоха на клетчатой бумаге.....	64
1.6.3. Просчитываем головоломку Full Board.....	69
1.6.4. Создаем свой оригинальный "Ним".....	87
Глава 2. Практика на форме Delphi. Игра Calculator.....	115
2.1. Постановка задачи.....	115
2.2. Создание основной формы игры.....	116
2.3. Программирование игры.....	119
2.4. Создание формы для вывода правил игры.....	123
Глава 3. В Win API под DirectX.....	143
3.1. Шаблон минимальной программы Delphi.....	143
3.2. Шаблон второй программы.....	149
3.3. Переходим в полноэкранный режим DirectDraw.....	155
3.4. Создание поверхностей DirectDraw.....	159

3.5. Запись точек на поверхность DirectDraw	165
3.6. Переключение поверхностей DirectDraw	169
3.7. Написание игровой программы.....	190
3.7.1. Файл glob.pas.....	194
3.7.2. Файл dominoes.dpr	197
3.7.3. Модуль Midi.pas.....	207
3.7.4. Модуль Sound.pas	210
3.7.5. Модуль Mouse.pas.....	219
3.7.6. Модуль Fonts.pas.....	222
3.7.7. Модуль DDrawUnit.pas.....	227
3.7.8. Модуль Util.pas	235
3.7.9. Модуль Game.pas	250
Глава 4. Интеграция Flash и Delphi.....	299
4.1. Создаем во Flash игру Calculator	300
4.1.1. Добавляем звук	328
4.2. Устанавливаем компонент <i>Shockwave Flash</i>	330
4.3. Создаем форму и код программы	331
4.3.1. Отменяем показ меню и реакцию кнопок на нажатие клавиши <Tab>.....	332
4.3.2. Присоединяем swf-файл к программе.....	334
4.3.3. Применяем функцию <i>fscommand</i>	334
4.3.4. Передаем в программу Delphi параметры для <i>IsWin</i>	335
4.3.5. Пишем функцию <i>IsWin</i> и остальной код Unit1.pas.....	336
4.3.6. Получаем результат хода во Flash-игре.....	345
Глава 5. Применение PCRE в Delphi.....	355
5.1. Синтаксис языка PCRE.....	356
5.1.1. Модификаторы регулярных выражений	357
5.1.2. Квантификаторы в регулярных выражениях.....	358
5.1.3. Символьные классы	358
5.1.4. Альтернативные шаблоны	359
5.1.5. Метасимволы и специальные символы	360
5.1.6. Мнимые символы (якоря или условия).....	362
5.1.7. Захватывающие и незахватывающие скобки	363
5.1.8. Ссылки на найденный текст	365
5.1.9. Условные подшаблоны	367
5.1.10. Комментарии в регулярных выражениях.....	368

5.1.11. "Жадность" квантификаторов и ее ограничение	368
5.1.12. Перебор с возвратами	369
5.1.13. Атомарная группировка	370
5.1.14. Захватывающие квантификаторы	373
5.2. Программная реализация PCRE	374
5.2.1. Простые примеры использования PCRE	378
5.2.2. Именованные захватывающие скобки	382
5.2.3. Пример программы для замены текста	383
5.2.4. Пример преобразования ссылок в теги $\langle a \dots \rangle$	388
5.2.5. Поиск вложенных конструкций	394
5.2.6. Лексический разбор текста	400
5.2.7. Лексический разбор с несколькими шаблонами	403
5.2.8. Обработка текста по заданной грамматике	408

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 сохраняют 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 9x. 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 $1190 \cdot 2$ связана с частотой счета 1,19 МГц и тем, что счетчик считает по два. Для DOS и Windows 9x 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 9x. 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.

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

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.
      END.
    END.
  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;
        end;
      end;
    end;
  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 $N, (N - 1), \dots, 2, 1$. Первая перестановка упорядочена в словарном порядке, т. е. числа расположены по возрастанию. Если бы мы имели не числа, а буквы a, b, c, \dots , то это было бы более наглядно.

Чтобы в массиве m по данной перестановке получить следующую, сделаем 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 Perl;  
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] }
il:=i+1;
i2:=n-1;
while il < 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 `perl.exe` (`perl.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: `perl > out.txt`. Если файл `out.txt` существует, то он будет уничтожен и воссоздан заново. Если вы хотите дописать к файлу, то используйте формат `perl >> 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` строк

```
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