Лицензия

Авторские права на публикуемые материалы принадлежат автору книги Осипову Александру Викторовичу. Публикация данных материалов не предполагает извлечения какой-либо коммерческой выгоды.

Публикуемые материалы защищены действующим законодательством об авторском праве. Все предусмотренные этим законодательством права на опубликованные материалы принадлежат их автору.

Официальным источником для распространения материалов является Интернет-сайт //pascalabc.net, ссылка на который при цитировании обязательна. Разрешается свободно копировать и распространять исключительно на безвозмездной основе опубликованные материалы при условии сохранения их в неизменном виде и с указанием авторства. Передача материалов третьим лицам разрешается при условии сохранения в них страницы с настоящей лицензией. Исключение делается для учебных заведений: при подготовке раздаточного материала допускается страницу с лицензией не включать. Любые другие способы распространения опубликованных материалов при отсутствии письменного разрешения автора запрещены.

Запрещается любым организациям осуществлять любого рода лицензирование опубликованного материала и осуществлять какую бы то ни было иную связанную с авторскими правами деятельность без письменного разрешения автора.

8 Многомерные массивы

В части 5 были рассмотрены одномерные массивы. Но массив может иметь и большее число измерений.

Если M(x,y) - точка на плоскости, в декартовой системе координат она имеет две координаты. Точка M(x,y,z) имеет три координаты. Если рассмотреть положение точки в трехмерном пространстве, оно может оказаться изменяющимся во времени t, и точка M(x,y,z,t) будет описываться четырьмя координатами. Плоская или объемная фигура – это совокупность точек, так что она в рассмотренных случаях будет описываться двухмерным, трехмерным или четырехмерным массивом. Можно придумать варианты, когда понадобится массив и с еще большим числом измерений. Из многомерных массивов наиболее популярны двухмерные (можно также писать двумерные) массивы. В PascalABC.NET двухмерные динамические массивы называются ***матрицами***.

Матрица имеет два измерения, называемые строками и столбцами по аналогии с матрицами в математике. Матрица всегда имеет прямоугольную форму: количество элементов в каждой строке постоянно и количество элементов в каждом столбце тоже постоянно. Внешне матрицу всегда можно представить в виде таблицы.

Пусть имеется матрица, состоящая из m строк и n столбцов. В этом случае говорят, что она имеет размер m×n. Не путайте размер с размерностью: в матрице размерность (число измерений массива) равна двум. Если элемент матрицы A находится на пересечении строки i и столбца j, его записывают как $A_{i,j}$ или A[i, j].

8.1 Динамические двухмерные массивы (матрицы)

Как вы уже знаете, массивы бывают статические и динамические. Размеры (т.е. количество элементов) по каждому измерению статического массива должны быть известны компилятору, иначе он не сможет зарезервировать нужный объем памяти компьютера. В случае динамического массива мы сообщаем компилятору лишь количество измерений массива. Конечно, в любом случае, мы сообщаем еще и тип элементов массива. Динамический массив создается во время работы программы, и тогда же определяются его размеры. Процедура SetLength, с помощью которой можно в процессе выполнения программы задать или поменять размер одномерного массива, работает и для двухмерных массивов. Но если с первоначальным заданием размера массива все просто, то с его переопределением есть проблема. Мы разберемся с этим позже.

8.1.1 Описание и создание матриц

Пусть у нас имеется двухмерный массив размером 4×3. В PascalABC.NET динамические массивы нумеруются с нуля, поэтому получаются четыре строки с номерами от 0 до 3 и три столбца, нумерованные от 0 до 2.

$$\begin{pmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \\ a_{30} & a_{31} & a_{32} \end{pmatrix}$$

Каждую строку двухмерного массива можно представить, как обычный массив. Массив из таких строк образует массив массивов. Построить его несложно.

```
begin
  var A: array of array of integer;
  var (m, n) := (4, 3); // число строк и столбцов
  SetLength(A, m); // распределим память под m строк
  for var i := 0 to m - 1 do
    SetLength(A[i], n); // в каждой строке создадим массив из n элементов
  A[3][2] := 43; // строка с индексом 3, в ней элемент индексом 2
  A[1, 0] := 21; // строка с индексом 1, в ней элемент индексом 0
  Writeln(A) // [[0,0,0],[21,0,0],[0,0,0],[0,0,43]]
end.
```

При обращении к элементу такого «двухмерного массива» допускается запись как двух индексов по отдельности, так и совместно. Можно обойтись без этих нагромождений, описав «нормальный» двухмерный массив.

```
begin
  var A: array [,] of integer; // обратите внимание на запятую
  var (m, n) := (4, 3); // число строк и колонок
  SetLength(A, m, n);
  A[3, 2] := 43; // писать A[3][2] тут не допускается
  A[1, 0] := 21;
  Writeln(A) // [[0,0,0],[21,0,0],[0,0,0],[0,0,43]]
end.
```

Для двухмерных массивов, подобно одномерным, можно совмещать описание с выделением памяти. Для этого массив создается с использованием ключевого слова **new**.

```
var a: array [,] of integer := new integer[4, 3]; // с описанием
var b := new real[4, 3]; // с автовыведением типа
```

Можно также выполнить инициализацию, добавив ***конструктор массива***.

```
var a := new integer[4, 3] ((1, 2, 3), (4, 5, 6), (7, 8, 9), (10, 11, 12));
var b := new real[2, 3] ((2.1, 3.7, 5), (1, 2, 3));
var d: array of array of integer := ((1, 2, 3), (4, 5), (6, 7, 8));
```

После такой инициализации массив a будет содержать следующие значения

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{pmatrix}$$

Конструктор двухмерного массива имеет два уровня скобок. Первый (внешний) уровень ограничивает сам конструктор, второй (внутренний) – ограничивает каждую строку. В приведенном примере массив a имеет четыре строки и три колонки. Соответственно, конструктор содержит четыре группы по три значения в каждой. Пример с инициализацией массива b напоминает нам о том, что заданные в конструкторе значения автоматически приводятся к нужному типу.

Массив d инициализируется как массив массивов. Отметьте, что в данном случае конструктор позволяет создать и инициализировать непрямоугольный массив: во второй строке два элемента, а в остальных по три. К элементу массива d можно также обращаться в виде d[i][j] или d[i, j]. Тем не менее, у массивов d и b разный тип данных. А это означает что их, например, нельзя передавать в подпрограммы один вместо другого.

8.1.2 Генерация матриц

Генерация матриц подразумевает присваивание элементам матрицы значений, задаваемых генератором псевдослучайных чисел или некоторым выражением. В случае, когда нужно присвоить всем элементам матрицы одинаковые значения, говорят о ***заливке*** матрицы. Генерация матриц в PascalABC.NET подразумевает вызов некоторой функции, возвращающей двухмерный динамический массив необходимых размеров. Это позволяет заранее не описывать матрицы, а пользоваться автоматическим выведением типа.

8.1.2.1 Заполнение случайными значениями

Имеются два генератора, заполняющие матрицу целыми, либо вещественными значениями.

- `MatrRandom(m, n, a, b)` – заполнение матрицы размера m×n целыми числами из интервала [a;b]. Имеется синоним MatrRandomInteger ;
- `MatrRandomReal(m, n, a, b)` – заполнение матрицы размера m×n вещественными числами из интервала [a;b].

Имеются следующие значения параметров, принимаемые по умолчанию: m=5, n=5, a=0, b=100.

Следующий пример иллюстрирует создание матриц, а также удобный способ организации их вывода при помощи расширения .Print. Традиционно, это расши-

рение «пропускает через себя» данные, сохраняя их тип. Поэтому, если у вас «синдром одной строки», можете смело встраивать .Print в цепочку.

```
begin // p08001
  var a := MatrRandom(6, 9, -50, 50);
  a.Println;
  Println;
  var b := MatrRandomReal(4, 3, -5, 5);
  b.Println
end.
```

Отметим аккуратный вывод, который дает .Print. По умолчанию под вывод целочисленного значения отводятся четыре позиции, под элемент вещественного типа – семь позиций, в двух из которых размещается дробная часть.

```
  39   8  50 -21  15 -42  12  25  19
 -38 -35   7 -37  27   4 -12  44 -12
   0 -49  45  13  11 -12 -43  46  39
  -7  19  18  46 -46  -8  40 -49 -13

  -4.30  -1.49  -4.66
   0.62   3.09  -0.52
  -4.76   3.59   1.23
   1.63   1.03   4.25
```

Если нужна иная разметка вывода, можно указать количество позиций явно.

```
begin // p08002
  MatrRandom(6, 9, -50, 50).Println(6);
  Println;
  MatrRandomReal(4, 3, -5, 5).Println(11, 7)
end.
```

```
19   -32     6    27    35    40   -22    10    30
23   -18   -35    32    16   -41    34    32    48
33   -31   -16    25    -1    34    42   -11    18
-1    11   -23    -5   -17    -9    43    29    47
 2   -16   -48   -23     2     7    18   -47   -39
26    -2   -43    -2   -28   -20    -7    36    47

 -3.5468194  4.4170922  2.0108772
  2.2497611  3.4569828 -0.4248126
 -2.7983750  2.9217856  4.9491709
  0.2331057 -2.2594955  4.9472349
```

8.1.2.2 Заполнение фиксированным значением

`MatrFill(m, n, x)` возвращает матрицу размера m×n, заполненную значением выражения x. Тип элементов матрицы будет совпадать с типом значения x. Например, вызов `MatrFill(5, 5, '*')` вернет матрицу размером 5×5 типа char, каждый элемент которой будет содержать звездочку.

8.1.2.3 Заполнение значениями, зависящими от индексов

`MatrGen(m, n, (i, j) -> f(i, j))` возвращает матрицу размера m×n, каждый элемент которой заполняется значением некоторой функции от своих индексов. Например, квадратная матрица с единичными элементами на диагонали от $A_{0,0}$ до $A_{5,5}$ и нулями в остальных элементах строится вызовом

```
var a := MatrGen(6, 6, (i, j) -> i = j ? 1 : 0);
```

А вот так получается таблица умножения (привет, Турбо Паскаль!):

```
begin // p08003
  MatrGen(10, 10, (i, j) -> (i + 1) * (j + 1)).Println
end.
```

```
 1   2   3   4   5   6   7   8   9  10
 2   4   6   8  10  12  14  16  18  20
 3   6   9  12  15  18  21  24  27  30
 4   8  12  16  20  24  28  32  36  40
 5  10  15  20  25  30  35  40  45  50
 6  12  18  24  30  36  42  48  54  60
 7  14  21  28  35  42  49  56  63  70
 8  16  24  32  40  48  56  64  72  80
 9  18  27  36  45  54  63  72  81  90
10  20  30  40  50  60  70  80  90 100
```

8.1.2.4 Заполнение на основе одномерного массива

- `Matr(m, n, a)` – заполнение матрицы размера m×n значениями элементов одномерного массива a. Предполагается, что этот массив имеет длину m×n. Заполнение матрицы производится в порядке по строкам;
- `Matr(m, n, a0,0, a0,1, … am-1,n-1)` – заполнение матрицы размера m×n перечисленными значениями элементов.

При несовпадении количества предлагаемых для заполнения элементов с размером матрицы генерируется исключения с выдачей сообщения «Ошибка времени выполнения: Количество инициализирующих элементов не совпадает с количеством элементов матрицы»

Пример. Заполнение матрицы размером 3×8 первыми 24 числами Фибоначчи.

```
begin // p08004
  var a := Matr(3, 8, ArrGen(24, 1, 1, (i, j)-> i + j)).Println(6)
end.
```

```
     1      1      2      3      5      8     13     21
    34     55     89    144    233    377    610    987
  1597   2584   4181   6765 10946 17711 28657 46368
```

Подобный прием удобно использовать для оперативного формирования и вывода различных таблиц.

8.1.3 Клавиатурный ввод значений элементов матриц

В базовом Паскале имеется единственный способ ввести значения элементов матрицы: организовать перебор всех элементов во вложенных циклах с последовательным присваиванием введенного значения очередному элементу. Порядок ввода можно задать как по строкам (второй индекс меняется быстрее первого), так и по столбцам (первый индекс меняется быстрее).

```
begin
  var (m, n) := (3, 2);
  var a := new real[m, n];
  for var i := 0 to m - 1 do
    for var j := 0 to n - 1 do
      Read(a[i, j]);
  a.Println
end.
```

```
2.5 -1.4 3.9 0.15 2 0.03
   2.50  -1.40
   3.90   0.15
   2.00   0.03
```

Чтобы каждый раз не писать эти типовые вложенные циклы, в PascalABC.NET введены функции ReadMatrInteger(m, n) и ReadMatrReal(m, n), возвращающие матрицу размера m×n типа **integer** или **real** соответственно, заполненную принятыми с клавиатуры значениями. Для прочих типов данных придется пользоваться приведенным выше решением на базе вложенных циклов.

```
begin
  var (m, n) := (3, 2);
  var a := ReadMatrReal(m, n);
  a.Println
end.
```

Поскольку массив динамический, компилятор заранее не распределяет под него память. Это позволяет не фиксировать в программе значения m и n, а вводить их с клавиатуры или получать каким-либо иным путем.

8.1.4 Вывод матриц (.Print)

Расширение .Print (и его разновидность .Println) для матриц имеют некоторые особенности. Во-первых, они возвращают не последовательность, а матрицу тех же размеров. Во-вторых, необязательным параметром здесь является не строка-разделитель, а количество позиций, отводимых под длину поля для выводимого элемента. И главное – вывод осуществляется построчно с сохранением формы матрицы.

- .Print(n) – для всех типов элементов матриц, кроме вещественных, выводит каждый элемент в n позициях. По умолчанию n=4 ;
- .Print(n, k) – для вещественных элементов отводит под вывод значения n позиций, сохраняя k знаков в дробной части (с округлением). По умолчанию n=7, k=2.

Недокументированная способность расширения .Print возвращать после вывода исходную матрицу дает возможность компактно записывать алгоритмы, встраивая .Println в цепочку, но в этом есть некоторое «шаманство». Оно отчасти допустимо для учебных задачек, а в практику программирования входить не должно.

В отличие от базового Паскаля, не допускающего указывать имя массива в операторе вывода Write, PascalABC.NET позволяет указывать в качестве параметров Write/Print любые массивы. Следует заметить, что получаемый в этом случае вывод пригоден в основном для целей отладки.

8.1.5 Переопределение размеров матрицы

Переопределить размеры матрицы можно двумя способами. Первый состоит в определении новой матрицы такого же типа в правой части оператора присваивания. Поскольку матрица – это динамический массив, левая и правая части оператора присваивания будут совместимы, что позволит передать ссылку на вновь созданную матрицу. Второй способ состоит в использовании процедуры SetLength.

Рассмотрим использование процедуры SetLength изменяющей размер матрицы с 3×4 на 2×7. Первоначально в матрице содержится 12 элементов, а после изменения размеров их станет 14, поэтому значения двух элементов не будут определены. Но мы помним, что все данные числового типа, не получившие значений, будут инициализированы нулями.

Казалось бы, достаточно просто задать новые размеры матрицы при помощи SetLength (мы так уже поступали с одномерными массивами) – и задача решена. Посмотрим, так ли это.

```
begin // p08005
  var a := MatrGen(3, 4, (i, j)-> 10 * (i + 1) + j + 1);
  a.Println;
  Println;
  SetLength(a, 2, 7);
  a.Println
end.
```

Но результат оказывается вовсе не таким, как ожидался.

```
 11  12  13  14
 21  22  23  24
 31  32  33  34

 11  12  13  14   0   0   0
 21  22  23  24   0   0   0
```

Если визуально наложить полученную матрицу на исходную, станет хорошо видно, что сохранились значения элементов, принадлежащие общей части обеих матриц. Прочие элементы оказались обнулены.

Секрета тут нет. Матрицы в памяти компьютера хранятся построчно, т.е. второй индекс у элементов увеличивается быстрее первого. Для исходной матрицы в памяти были значения (11 12 13 14) (21 22 23 24) (31 32 33 34). После того, как SetLength перераспределила память, в каждой строке стало семь элементов вместо четырех. Первая строка превратилась в (11 12 13 14 0 0 0), вторая – в (21 22 23 24 0 0 0), а третьей строки в новой матрице просто нет.

Но все же, как быть, если нам надо создать новую матрицу без потери данных? Для этого нужно воспользоваться способом, который был упомянут первым. Превратить матрицу в одномерный массив, а потом с помощью Matr (подраздел 8.1.2.4) сформировать из него двумерный массив. И выполнить присваивание.

```
begin // p08006
  var a := MatrGen(3, 4, (i, j)-> 10 * (i + 1) + j + 1);
  a.Println;
  Println;
  a := Matr(2, 7, a.ElementsByRow.ToArray + Arr(0, 0));
  a.Println
end.
```

```
11  12  13  14
21  22  23  24
31  32  33  34

11  12  13  14  21  22  23
24  31  32  33  34   0   0
```

Здесь пришлось немного забежать вперед, и использовать расширение .ElementsByRow. Отметим пока, что оно позволяет получить из матрицы последовательность значений элементов в порядке по строкам.

8.1.6 Получение сведений о текуших размерах матрицы

- a.RowCount – расширение, возвращающее количество строк в матрице;
- a.ColCount – расширение, возвращающее количество столбцов в матрице.

Очень полезная вещь в процедурах и функциях. Позволяет не передавать в них лишние параметры. Да и в цикле перебрать элементы строки от 0 до a.ColCount-1 тоже вполне удобно.

8.1.7 Выборка элементов матрицы

Выбирать можно строку, столбец и все элементы в порядке прохода матрицы по строкам или столбцам. Результат может быть одномерным массивом или последовательностью. Все эти операции обеспечивает набор расширений.

- a.Col(k) – возвращает в виде массива колонку матрицы a с номером k (отсчет номеров ведется от нуля);
- a.ColSeq(k) – возвращает в виде последовательности колонку матрицы a с номером k;

- a.Cols – возвращает последовательность колонок матрицы, в которой каждая колонка, в свою очередь, является последовательностью;
- a.Row(k) – возвращает в виде массива строку матрицы a с номером k (отсчет номеров ведется от нуля);
- a.RowSeq(k) – возвращает в виде последовательности строку матрицы a с номером k;
- a.Rows – возвращает последовательность строк матрицы, в которой каждая колонка, в свою очередь, является последовательностью;
- a.ElementsByCol – возвращает последовательность элементов матрицы, выбирая их по колонкам;
- a.ElementsByRow – возвращает последовательность элементов матрицы, выбирая их по строкам;
- a.ElementsWithIndices – возвращает последовательность трехэлементных кортежей, в которой каждый элемент формируется на основе Aij и имеет структуру вида (значение, индекс i, индекс j).

Ниже в демонстрационных целях приводится программа, делающая различные операции с матрицами.

```
function NewMatrix(m, n: integer): array[,] of integer; // p08007
begin
  Result := MatrRandom(m, n, -50, 50);
  Result.Println;
  Println('-' * 4 * n)
end;

begin
  var a := NewMatrix(3, 5);
  var b := NewMatrix(2, 7);
  Println('Min(a) =', a.ElementsByRow.Min);
  Println('Среднее по b:', b.ElementsByRow.Average);
  Print('Сумма по колонкам в a:');
  a.Cols.Select(t->t.Sum).Println;
  Print('Сумма кубов модулей элементов полследней строки a:');
  a.RowSeq(a.RowCount-1).Select(t->Abs(t**3)).Sum.Println;
  Print('Максимальный элемент в b:');
  var max:=b.ElementsWithIndices.MaxBy(t->t[0]);
  $'B[{max[1]+1},{max[2]+1}] = {max[0]}'.Println
end.
```

```
   8 -39 -26  -1 -39
 -19   7 -31 -48   2
  10 -32  43 -44 -30
--------------------
 -31 -38 -46   0  42 -36  -5
  21 -46 -49  48  -2  48  21
---------------------------
Min(a) = -48
Среднее по b: -5.21428571428571
Сумма по колонкам в a: -1 -64 -14 -93 -67
Сумма кубов модулей элементов последней строки a: 225459
Максимальный элемент в b: B[2,4] = 48
```

8.1.8 Перебор всех элементов матрицы (.foreach)

Процедура .foreach является методом расширения, позволяющим последовательно перебрать все элементы матрицы в порядке «по строкам». Имеются две разновидности, отличающиеся набором параметров лямбда-процедур:

- a.ForEach(T-> ()) применяет указанное лямбда-процедурой действие к каждому элементу матрицы a;
- a.ForEach((T,i,j)-> ()) применяет указанное лямбда-процедурой действие к каждому элементу матрицы a, предоставляя доступ к его индексам i и j.

Например, организовать вывод элементов массива в желаемом оформлении можно следующим образом.

```
begin
  var a := MatrRandom(3, 5, -9999, 9999);
  a.ForEach((v, i, j)->
    begin $'{v,10:# ##0}'.Print; if j = a.ColCount-1 then Println end)
end.
```

```
   322      4 799      9 543     -1 905     -3 759
 4 847      6 600     -9 545      8 367      8 560
-1 762      5 268     -3 794     -6 507     -4 447
```

8.1.9 Модификация строк и столбцов

Помимо выборки отдельных строк и столбцов, в PascalABC.NET включены методы расширения для их замены, обмена местами выбранной пары строк/столбцов и преобразования всех элементов матрицы.

- a.ConvertAll(T -> T1) – функция, возвращающая матрицу типа T1, в которой каждый элемент матрицы a типа T преобразован в соответствии с заданным лямбда-выражением;
- a.ConvertAll((T, i ,j) -> T1) – функция, возвращающая матрицу, в которой каждый элемент матрицы a преобразован в соответствии с заданным лямбда-выражением, в котором участвуют индексы строки и столбца;
- a.Fill((i, j) -> T) – процедура, заполняющая каждый элемент матрицы a в соответствии с заданным лямбда-выражением, в котором участвуют индексы строки и столбца;
- a.SetCol(k, v) – процедура, заменяющая в матрице a элементы столбца с индексом k элементами массива или последовательности v того же типа;
- a.SetRow(k, v) – процедура, заменяющая в матрице a элементы строки с индексом k элементами массива или последовательности v того же типа;
- a.SwapCols(k1, k2) – процедура, обменивающая местами колонки с индексами k1 и k2;
- a.SwapRows(k1, k2) – процедура, обменивающая местами строки с индексами k1 и k2;
- a.Transform(T -> T) – процедура, преобразующая каждый элемент матрицы a в соответствии с заданным лямбда-выражением;

- a.Transform((T, i ,j) -> T) – процедура, преобразующая каждый элемент матрицы a в соответствии с заданным лямбда-выражением, в котором участвуют индексы строки и столбца.

В качестве примера создадим целочисленную матрицу A размером 7×7 из случайных чисел на интервале [-50;50] и найдем среднее значение m среди всех ее элементов. Далее, заменим все элементы, отличающиеся по абсолютной величине от m больше чем на 20, суммой индекса строки и столбца такого элемента, взятой со случайным знаком. В полученной матрице заполним нулями все строки, в которых положительных элементов будет больше, чем прочих. Выведем исходную матрицу, найденное значение m, матрицу, полученную после выборочной замены элементов и результирующую матрицу.

```
begin // p08008
  var a := MatrRandom(7, 7, -50, 50);
  a.Println;
  Println('-' * 7 * 4);
  var m := a.ElementsByRow.Average;
  Println('m =', m);
  var rs: () -> integer := () -> Random(1, 2) = 1 ? -1 : 1;
  a.Transform((t, i, j) -> Abs(t - m) > 20 ? rs * (i + j) : t);
  a.Println;
  Println('-' * 7 * 4);
  var b := a.Rows.Select(row -> row.Where(t -> t > 0).Count).ToArray;
  var nCols := a.ColCount;
  var c := ArrFill(nCols, 0);
  for var i := 0 to a.RowCount - 1 do
    if 2 * b[i] > nCols then
      a.SetRow(i, c);
  a.Println
end.
```

```
 -45   5  47   0  10 -22  36
  -4  -2 -23 -36 -16   1  31
 -10 -21  49  20 -40  13  20
 -34 -20  50 -22  47 -45 -50
  -5  39  45  24   7   0  26
   5  -9  -9   6 -13  38 -19
 -24 -33 -50 -21 -16 -23  37
----------------------------
m = -1.14285714285714
   0   5   2   0  10  -5   6
  -4  -2   3   4 -16   1   7
 -10 -21  -4   5  -6  13   8
  -3 -20   5  -6   7  -8  -9
  -5   5  -6  -7   7   0  10
   5  -9  -9   6 -13  10 -19
   6  -7   8 -21 -16  11 -12
----------------------------
```

```
   0    0    0    0    0    0    0
   0    0    0    0    0    0    0
 -10  -21   -4    5   -6   13    8
  -3  -20    5   -6    7   -8   -9
  -5    5   -6   -7    7    0   10
   5   -9   -9    6  -13   10  -19
   6   -7    8  -21  -16   11  -12
```

Остановимся на некоторых операторах приведенной программы. Переменная rs – это имя лямбда-функции без параметров, возвращающей случайное значение типа integer, равное 1 или -1. Процедура Transform преобразует матрицу по указанному в условии задачи правилу: если для некоторого элемента t матрицы Abs(t - m) > 20, то значение элемента заменяется суммой его индексов, умноженной на значение функции rs. Значения элемента массива b[i] – это количество положительных элементов в строке и индексом i. Массив с содержит нули, его размер равен количеству столбцов в матрице. Строки матрицы последовательно просматриваются в цикле **for**, поскольку нам понадобятся индексы строк в случае замены. В теле этого цикла проверяется заданное условие и при его выполнении делается замена строки посредством SetRow. В самом деле, если b[i] – количество положительных элементов, а nCols – их общее количество, исходное условие выглядит как b[i] > nCols – b[i] или 2*b[i] > nCols.

8.1.10 Транспонирование матрицы

Транспонированием матрицы в математике называется замена ее строк столбцами. Исходная матрица размером m×n превращается в матрицу размером n×m.

$$\begin{pmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \\ a_{30} & a_{31} & a_{32} \end{pmatrix} \rightarrow \begin{pmatrix} a_{00} & a_{10} & a_{20} & a_{30} \\ a_{01} & a_{11} & a_{21} & a_{31} \\ a_{02} & a_{12} & a_{22} & a_{32} \end{pmatrix}$$

Операция транспонирования матрицы a может быть выполнена посредством функции Transpose(a), возвращающей транспонированную матрицу.

8.1.11 О матричных операциях

Как отмечалось ранее, матрицы в PascalABC.NET – это всего лишь название двухмерных динамических массивов. В математике под матрицами понимают совсем иные объекты, для которых определен набор соответствующих операций и операторов. Некоторая их часть реализована для класса Matrix в библиотеке численных методов NumLibABC (часть 11), входящей в состав стандартной поставки PascalABC.NET.

8.2 Статические двухмерные массивы

Как и в случае с одномерными массивами, статический двухмерный массив описывается с указанием границ индексов. Границы должны быть заданы и для строк, и для столбцов. Память под статический массив распределяется на этапе компиляции, выделяется при загрузке программы, и в дальнейшем не может быть перераспределена.

Статический двухмерный массив описывается в виде

```
var ИмяМассива: array[m1..n1, m2..n2] of Тип; или
var ИмяМассива: array[t1, t2] of Тип;
```

Конструкция вида m..n описывает минимальное и максимальное значение, которое может принимать индекс массива, причем допускаются и отрицательные значения. Эта конструкция задается константой порядкового типа. Количество элементов в массиве можно вычислить по формуле (n1-m1+1)×(n2-m2+1).

```
var a: array[0..12, 1,,4] of byte;
var b, c: array[-5..8, 0..6] of real;
```

Описание массива можно совместить с инициализацией его элементов. Как и для динамических массивов, поскольку тип массива задан, можно при инициализации массива вещественного типа указывать в списке значения целочисленных типов.

```
begin // p08009
  var a: array[1..3, 1..4] of integer := ((1, 2, 3, 4),
      (5, 6, 7, 8), (9, 10, 11, 12));
  Println(a);
  var b: array[0..2, 1..3] of real := ((1.2, 5, -3.05),
      (-4, -7, 1), (15, 7, 7));
  Println(b);
  var c: array [0..2, 1..2] of char := (('a', 'b'),
      ('c', 'd'), ('e', 'f'));
  Println(c)
end.
```

```
[[1,2,3,4],[5,6,7,8],[9,10,11,12]]
[[1.2,5,-3.05],[-4,-7,1],[15,7,7]]
[[a,b],[c,d],[e,f]]
```

Двухмерные статические массивы в PascalABC.NET оставлены в целях совместимости с базовым Паскалем.

8.3 Массивы размерности выше двух

Общепринятое название массивов, имеющих размерность выше двух, – многомерные массивы. PascalABC.NET не имеет специальных средств работы с многомерными массивами – реализованы лишь их описание, создание, инициализация и вывод. Также, многомерные массивы можно присваивать и передавать в качестве параметров.

При описании и инициализации многомерных массивов нужно задавать границы их индексов по каждому измерению и соответствующий им конструктор массива.

```
begin // p08010
  var a: array[1..2, 1..3, 1..2] of integer := (((1, 2), (3, 4), (5, 6)),
      ((7, 8), (9, 10), (11, 12)));
  Println(a);
  a[1, 2, 2] := 0;
  Println(a);
end.
```

```
[[[1,2],[3,4],[5,6]],[[7,8],[9,10],[11,12]]]
[[[1,2],[3,0],[5,6]],[[7,8],[9,10],[11,12]]]
```

8.4 Примеры решения задач с матрицами

Задача 1. Переставить строки матрицы C размера m×n: первую с последней, вторую с предпоследней и т.д. В полученной матрице умножить каждый элемент третьей сверху строки на сумму элементов предпоследнего столбца. Значения m и n ввести с клавиатуры, элементы матрицы получить по формуле $C_{ij} = \sin(2*i-j)$ и округлить до трех знаков после запятой.

Здесь единственный подвох – случай с нечетным количеством строк m. Нужно сделать m/2 перестановок строк. «Третья сверху» строка будет иметь индекс 2, а предпоследний столбец – индекс n-2.

```
begin // p08011
  var (m, n) := ReadInteger2;
  var c := MatrGen(m, n, (i, j)-> Round(Sin(2 * i - j), 3));
  c.Println(7, 3);
  Println('-' * 7 * n); // разделитель
  for var i := 0 to (m - 1) div 2 do
    c.SwapRows(i, m - i - 1);
  c.Println(7, 3);
  Println('-' * 7 * n);
  var s := c.Col(n - 2).Sum;
  c.SetRow(2, c.Row(2).Select(t -> t * s));
  c.Println(7, 3)
end.
```

```
5 8
  0.000 -0.841 -0.909 -0.141  0.757  0.959  0.279 -0.657
  0.909  0.841  0.000 -0.841 -0.909 -0.141  0.757  0.959
 -0.757  0.141  0.909  0.841  0.000 -0.841 -0.909 -0.141
 -0.279 -0.959 -0.757  0.141  0.909  0.841  0.000 -0.841
  0.989  0.657 -0.279 -0.959 -0.757  0.141  0.909  0.841
--------------------------------------------------------
  0.989  0.657 -0.279 -0.959 -0.757  0.141  0.909  0.841
 -0.279 -0.959 -0.757  0.141  0.909  0.841  0.000 -0.841
 -0.757  0.141  0.909  0.841  0.000 -0.841 -0.909 -0.141
  0.909  0.841  0.000 -0.841 -0.909 -0.141  0.757  0.959
  0.000 -0.841 -0.909 -0.141  0.757  0.959  0.279 -0.657
--------------------------------------------------------
  0.989  0.657 -0.279 -0.959 -0.757  0.141  0.909  0.841
 -0.279 -0.959 -0.757  0.141  0.909  0.841  0.000 -0.841
 -0.784  0.146  0.942  0.871  0.000 -0.871 -0.942 -0.146
  0.909  0.841  0.000 -0.841 -0.909 -0.141  0.757  0.959
  0.000 -0.841 -0.909 -0.141  0.757  0.959  0.279 -0.657
```

Задача 2. Упорядочить столбцы матрицы A размера m×n таким образом, чтобы элементы второй сверху строки образовали невозрастающую лексикографическую последовательность. Значения m и n ввести с клавиатуры, элементы матрицы заполнить случайными строчными буквами латинского алфавита.

В латинском алфавите 26 букв и в кодовой таблице строчные буквы следуют от «a» до «z» непосредственно друг за другом в алфавитном порядке. Поэтому достаточно сгенерировать случайное число r в диапазоне [0..25] и выбрать символ с кодом буквы «a», увеличенным на r. Далее следует принять решение, производить ли перестановку столбцов в существующей матрице или определить новую, а затем поместить в нее столбцы исходной матрицы в нужном порядке. В первом случае придется реализовать алгоритм обменной сортировки, но менять местами не элементы, а целиком столбцы. Во втором случае можно создать на базе элементов второй строки последовательность кортежей, в которые войдет значение элемента и его индекс в строке, отсортировать элементы этой последовательности по невозрастанию значений, а затем скопировать в новую матрицу колонки в порядке, указанном индексами из состава кортежей. Любое решение обладает как достоинствами, так и недостатками. Здесь будут приведены оба варианта, а выбор останется читателю.

Вариант А. Сортировка матрица на месте.

```
begin // p08012
  var (m, n) := ReadInteger2;
  var a := MatrGen(m, n, (i, j)-> Chr(Ord('a') + Random(0, 25)));
  a.Println(2);
  Println;
  for var i := n - 2 downto 0 do
    for var j := 0 to i do
      if a[1, j] < a[1, j + 1] then a.SwapCols(j, j + 1);
  a.Println(2)
end.
```

```
5 8
 e z z x k k m g
 r d t l e x b s
 d x q j e e q k
 o o q t l f l s
 m d m v q m k k

 k z g e x k z m
 x t s r l e d b
 e q k d j e x q
 f q s o t l o l
 m m k m v q d k
```

Вариант Б. Сортировка с использованием вектора индексов.

```
begin // p08013
  var (m, n) := ReadInteger2;
  var a := MatrGen(m, n, (i, j)-> Chr(Ord('a') + Random(0, 25)));
  a.Println(2);
  Println;
  var v := a.Row(1).Select((v, i)-> (v, i))
      .OrderByDescending(t -> t[0]).ToArray;
  var b := new char[m, n];
  for var i := 0 to n - 1 do
    b.SetCol(i, a.Col(v[i][1]));
  a := b;
  a.Println(2)
end.
```

Это вариант для тех, кто не помнит, как реализовать алгоритм обменной сортировки.

Задача 3. Найдена «на просторах Интернет», но не зря же говорят, что все пути ведут в Рим – это задача Matrix27 из задачника М.Э. Абрамяна. На youtube.com выложен ролик, где в течение тринадцати (!) минут автор добросовестно пыталась изложить решение этой задачи в среде PascalABC.NET, но… средствами базового Паскаля. Буду честным: я добросовестно пытался заставить себя выслушать предлагаемый ход решения задачи методом проб и ошибок, но это оказалось сверхзадачей. Замечу лишь, что итоговое решение содержало 30 строк кода. Итак, условие:

Дана матрица размером M×N. Найти максимальный среди минимальных элементов ее строк.

Как в предложенном решении, матрицу будем заполнять целочисленными значениями на интервале [0;20] с помощью датчика случайных чисел. Выведем исходную матрицу. Далее построим последовательность, содержащую минимальные элементы каждой строки и найдем значение максимального элемента этой последовательности.

```
begin // p08014
  var (m, n) := ReadInteger2('Введите M и N:');
  var a := MatrRandom(m, n, 0, 20);
  a.Println(3);
  Println(3 * n * '-'); // разделитель
  a.Rows.Select(row -> row.Min).Max.Println
end.
```

```
Введите M и N: 7 12
 20  6 10 14 15 18  8 10 20  5  1 11
  3  1  9  1 10 11  3 18  6  0 12  5
 18 16  5 18  8 15 20 17  9 18 19  4
  2 12 11 15  8 16  3  6 14  0  6  3
 14  4 11  1 20 15  2 20 15  4 17 19
 12 10  1  4  5  5  2  5  3  5 20 18
 20  8 10 10 17  5 11 17 10  2  4  3
------------------------------------
4
```

Семь строчек. А никаких не тридцать. Из них одна – предпоследняя – решает поставленную задачу, а прочие носят вспомогательные функции. Программа пишется за пару минут и не требует запутанных объяснений. Это – PascalABC.NET.