Массивы

Предположим, что программа работает с большим количеством однотипных данных. Скажем около ста разных целых чисел нужно обработать, выполнив над ними те или иные вычисления. Как вы себе представляете 100 переменных в программе? И для каждой переменной нужно написать одно и тоже выражение вычисления значения? Это очень неэффективно.

Есть более простое решение. Это использование такой структуры (типа) данных как **массив**. Массив представляет собой последовательность ячеек памяти, в которых хранятся однотипные данные. При этом существует всего одно имя переменной связанной с массивом, а обращение к конкретной ячейке происходит по ее индексу (номеру) в массиве.

Нужно четко понимать, что индекс ячейки массива не является ее содержимым. Содержимым являются хранимые в ячейках данные, а индексы только указывают на них. Действия в программе над массивом осуществляются путем использования имени переменной, связанной с областью данных, отведенной под массив.

Итак, *массив – это именованная группа однотипных данных, хранящихся в последовательных ячейках памяти*. Каждая ячейка содержит элемент массива. Элементы нумеруются по порядку, но необязательно начиная с единицы (хотя в языке программирования Pascal чаще всего именно с нее). Порядковый номер элемента массива называется индексом этого элемента.

Помним, **все элементы определенного массива имеют один и тот же тип**. У разных массивов типы данных могут различаться. Например, один массив может состоять из чисел типа **integer**, а другой – из чисел типа **real**.

Индексы элементов массива обычно целые числа, однако могут быть и символами, а также описываться другими порядковыми типами.

Массив можно создать несколькими способами.

Обращение к определенному элементу массива осуществляется путем указания имени переменной массива и в квадратных скобках индекса элемента.

Простой массив является **одномерным**. Он представляет собой линейную структуру.

```
var ch: array [1..11] of char;
    h: char;
    i: integer;

begin
    for i := 1 to 11 do read (ch[i]);

    for i := 1 to 11 do write (ch[i]:3);

readln
end.
```

В примере выделяется область памяти под массив из 11 символов. Их индексы от 1 до 11. В процессе выполнения программы пользователь вводит 11 любых символов (например, *'q', 'w', 'e', '2', 't', '9', 'u', 'I', 'I', 'o', 'p'*), которые записываются в ячейки массива. Текущее значение переменной *i* в цикле **for** используется в качестве индекса массива. Второй цикл **for** отвечает за вывод элементов массива на экран.

Одномерный массив можно представить как линейную структуру, в которой элементы следуют друг за другом. Однако бывают более сложные структуры данных. Например, двумерные массивы, которые можно описать как таблицу, в ячейках которой располагаются значения. Для обращения к данным массива указывается номера их строк и столбцов. Часто табличные массивы называют **матрицами**.

Обычно двумерные массивы на языке программирования Pascal описываются так:

array [1..m, 1..n] of базовый_тип

При этом описание может быть в разделе **type** и тогда создается новый тип, который можно использовать при объявлении переменных. Или массив может быть описан непосредственно в разделе переменных.*m* и *n* – это константы, их можно опустить и вставить конкретные значения, но лучше так не делать. Обычно подразумевают,

что в интервале от 1 до *m* определяется количество строк, а в интервале от 1 до *n* – количество столбцов массива.

1 вариант – описание массива через раздел **type**:

```
const
        M = 10;
        N = 5;
type
        matrix = array [1..M, 1..N] of integer;
var
        a: matrix;
```

2 вариант – описание массива в разделе переменных:

```
const
        M = 10;
        N = 5;
var
        a: array [1..M, 1..N] of integer;
```

Рассмотрим простой пример работы с двумерным массивом. Сначала заполним его данными, а затем выведем их на экран в виде таблицы.

```
var
    matrix: array[1..3,1..5] of integer;
    i, j: integer;

begin
    writeln ('Введите 15 чисел: ');

    for i := 1 to 3 do
        for j := 1 to 5 do
            read (matrix[i,j]);

    for i := 1 to 3 do begin
        for j := 1 to 5 do
            write (matrix[i,j], ' ');
        writeln
    end;

readln
end.
```

Размерность массива (т.е. количество содержащихся в нем значений) определяется произведением количества строк на количество столбцов. В примере выше в массив помещается 15 значений.

Когда пользователь вводит очередное число, то процедура **read**считывает его и помещает в ячейку с текущими индексами *i* и *j*. Когда *i*равна единице, значение *j* меняется пять раз, и, значит, заполняется первая строка таблицы. Когда *i* равна двум, значение *j* снова меняется пять раз и заполняется вторая строка

таблицы. Аналогично заполняется третья строка таблицы. Внутренний цикл **for** в общей сложности совершает 15 итераций, внешний только 3.

Как пользователь вводит значения – не важно. Он может их разделять либо пробелом, либо переходом на новую строку.

Вывод значений двумерного массива организован в виде таблицы. Выводятся 3 строки по 5 чисел в каждой. Внутри строк числа разделяются пробелом.

На самом деле, это не совсем корректно написанная программа. Мы несколько раз используем цифры 3 и 5. А что если мы захотим поменять размерность массива? Придется просмотреть всю программу (представьте, что она очень большая) и исправить значения. Это неэффективно. Поэтому в программе следует использовать константы. В случае чего их значения можно поменять всего лишь в одном месте.

Вторая проблема – это «кривость» выводимой на экран таблицы значений матрицы, в случае если есть значения разной разрядности (однозначное, двузначное и т.д. числа). Неплохо бы под каждое число отводить равное количество знаков.

Вот так может выглядеть подправленный вариант программы:

```
const
    M = 3;
    N = 5;

var
    matrix: array[1..M,1..N] of integer;
    i, j: integer;

begin
    writeln ('Введите 15 чисел: ');

    for i := 1 to M do
        for j := 1 to N do
            read (matrix[i,j]);

    for i := 1 to M do begin
        for j := 1 to N do
            write (matrix[i,j]:5);
        writeln
    end;

readln
end.
```