



МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ  
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Прикладная математика»

### **Программирование в Delphi: структура данных**

Методические указания к лабораторной работе № 6  
по курсам «Информатика», «Алгоритмические языки  
и программирование»

Автор  
Е.Н. Ладоса, Д.С. Цымбалов, О.В. Яценко,  
Е.В. Бочарова

Ростов-на-Дону, 2018



## Аннотация

Описаны приёмы и средства организации нестандартных информационных структур в Object Pascal. Целью работы ставится освоение студентами технологии структурирования данных с использованием среды Delphi. Предназначены для студентов всех специальностей факультета «Информатика и вычислительная техника».

## Автор

Доцент, к.т.н.  
Ладоша Е.Н.

Старший преподаватель кафедры  
«Электроника и электротехника»  
Цымбалов Д.С.

Доцент, к.ф.-м.н.  
Яценко О.В.

Студент ДГТУ  
Бочарова Е.В.



## Цель работы

Цель работы – изучить **приёмы и средства структурирования данных**. Отдельно рассмотреть статистические и динамические структуры, массивы однородных данных, а также способы их сортировки.

## Структура данных

Как видно из названия, **структура данных** – это некая структура, содержащая данные. В области компьютерного программирования структурам данных и методам их построения посвящено довольно много книг. В лабораторной работе рассматривается большинство таких структур. Ввиду обширности темы работы задумана лишь как введение в полный курс структур данных.

## Статические и динамические структуры данных

**Статическая структура** данных имеет фиксированный тип и размер. Для ее изменения программист должен изменить исходный код программы. В то же время размер (а иногда и тип) **динамической структуры** данных может изменяться в процессе выполнения программы.

Структура данных может быть **статической** или **динамической**. Тип и размер статической структуры данных фиксированы, изменить их можно только путем изменения исходного кода. В то же время размер и тип динамической структуры данных может изменяться при выполнении программы, т.е. динамически. Одни структуры данных могут быть только динамическими, так как создаются во время выполнения. Другие могут быть объявлены как статические или как динамические.

## Перечислимые типы и множества

Допустим, переменной `schoolYear` нужно присваивать значения, определяющие этап обучения некоего студента. В США для студентов колледжей принята следующая градация: `freshman` (первокурсник), `sophomore` (второкурсник), `junior` (младший) и `senior` (старший). Во многих компьютерных языках для присвоения таких значений необходимо или применить их нумерацию (т.е. условиться, что 1 – это `freshman`, 2 – это `sophomore` и т.д.), или объявить переменную как строковую и присваивать ей значения `freshman`, `sophomore` и т.д. Оба способа имеют существенные недостатки. Нумерация усложняет использование программы, так как человек плохо запоминает числа, а при использовании строковых переменных возникают проблемы неоднозначного написания и недопустимых значений. Object Pascal предоставляет программи-

сту лучший способ решения этой задачи: **перечислимые типы**.

Перечислимый тип является упорядоченным набором значений, определенных программистом. Эти значения можно представить себе, как последовательность натуральных чисел (начиная с единицы), каждому из которых присвоено определенное название. Таким образом, перечислимый тип является пользовательским типом данных (т.е. созданным программистом, как пользователем языка).

В Object Pascal типы данных определяются с помощью ключевого слова `type`. Для определения перечислимого типа используется следующий синтаксис:

```
type имя_типа = (значение1, ... , значениеN) ;
```

Входящие в синтаксис `имя_типа`, `значение1` ... `значениеN` должны быть правильными идентификаторами. Например, оператор

```
type
```

```
Year = (Freshman, Sophomore, Junior, Senior);
```

определяет перечислимый тип `Year`, значениями которого являются элементы `Freshman`, `Sophomore`, `Junior` и `Senior`.

В определении перечислимого типа значения от `значение1` до `значениеN` являются константами типа `имя_типа`. Если эти идентификаторы в этой же области видимости используются для других целей, то возникнет конфликт имен.

Перечислимый тип можно объявить и без ключевого слова `type`:

```
var
```

```
year1, year2: (Freshman, Sophomore, Junior, Senior);
```

При попытке объявить эти переменные отдельно

```
var year1: (Freshman, Sophomore, Junior, Senior);
```

```
var year2: (Freshman, Sophomore, Junior, Senior);
```

компилятор вернет сообщение об ошибке вследствие конфликта имен.

Наиболее предпочтительным способом объявления является создание (т.е. определение) пользовательского типа.

```
type
```

```
Year = (Freshman, Sophomore, Junior, Senior);
```

```
var
```

```
year1, year2: Year;
```

Поэтому, всегда создавайте перечислимые типы (как и другие пользовательские типы данных) с помощью ключевого слова `type`. Это обеспечит совместимость типов переменных и предотвратит ошибки компиляции.

Понятия **множества** в Object Pascal и в математике довольно похожи.

Множество состоит из группы значений одного и того же порядкового типа данных. Значения в множестве не упорядочены, и ни одно значение не может входить в множество более одного раза.

Как и перечислимый тип, множество является пользовательским типом данных.

В Object Pascal множество определяется с помощью ключевых слов `set of`.

type

```
имя_множества = set of имя_базового_типа;
```

Имя\_множества должно быть правильным идентификатором Object Pascal. Диапазон допустимых значений элементов множества совпадает со всеми допустимыми значениями базового типа, который должен быть порядковым типом данных. Базовый тип может быть также пользовательским типом данных, например перечислимым. Множество может содержать любые значения базового типа. Оно может быть также пустым множеством, обозначаемым в Object Pascal как `[]`. Кроме того, в Object Pascal на размер множеств наложен следующее ограничение: оно не может содержать более 256 значений. Поэтому порядковые значения базового типа должны находиться в диапазоне от 0 до 255.

В связи с этим ограничением множества иногда определяются с помощью поддиапазонов, обозначаемых двумя точками (`..`).

Например, оператор

type

```
LowercaseSet = set of 'a'..'z';
```

определяет тип множества с именем `LowercaseSet`, значениями которого являются буквы английского алфавита от `a` до `z` в нижнем регистре. Тип данных можно также присвоить поддиапазону, т.е. определить поддиапазон как тип с указанным именем.

type

```
LowercaseLetters = 'a' .. 'z' ;
```

```
LowercaseSet = set of LowercaseLetters;
```

Эти два объявления типа множества `LowercaseSet` эквивалентны.

Множество чисел можно объявить так:

type

```
numberSet = 0..255;
```

```
numbers = set of numberSet;
```

Определив, таким образом, тип, можно объявлять переменные, являющиеся множествами этого типа, а затем присваивать этим переменным значения множеств. В Object Pascal значение множества записывается как заключенный в квадратные скобки список его элементов, разделенных запятыми.

[значение1, значение2, . . . , значениеN]

Значения элементов множества, естественно, должны входить в определенный поддиапазон базового типа. Не путайте значения множества и значения элементов множества! Значение множества – это некоторый набор элементов. В следующем фрагменте кода объявляется и создается множество, содержащее буквы a, b, c, m, n, x, y и z (тип LowercaseSet считается предварительно определенным):

```
var
    myLetters:    LowercaseSet;
begin
    ...
    myLetters    :=    ['a'..'c1', 'm', 'n1', 'x'..'z'];
    ...
end;
```

Подобно тому как числовые переменные ассоциированы с арифметическими операциями, множества ассоциированы с набором операций над множествами, перечисленными в табл. 1. Например, оператор in проверяет вхождение элемента в множество: значение выражения 'a' in myLetters равно True, если элемент 'a' входит в множество myLetters, в противном случае оно равно False.

Таблица 1 - Операции над множествами

Оператор	Операция	Типы операндов	Тип результата	Пример
+	Объединение	Множество	Множество	set1+set2
-	Разность	Множество	Множество	set1-set2
*	Пересечение	Множество	Множество	set1*set2
<=	Подмножество	Множество	Boolean	subSet<=set1
>=	Надмножество	Множество	Boolean	superSet>=set1
=	Равенство	Множество	Boolean	set1 =set2
<>	Неравенство	Множество	Boolean	set1<>set2
in	Вхождение	Множество, порядковый тип	Boolean	value in set

Операторы +, - и \* подчиняются следующим правилам:

- Порядковое значение X входит в множество set1+set2, если и только если X входит в set1 или в set2 (или в оба эти множества). Значение X входит в set1-set2, если и только если X входит в set1, но не входит в set2. Значение X входит в set1 \*set2, если и только если X входит как в set1, так и в set2.

- Результаты операций  $+$ ,  $-$  и  $*$  принадлежат множеству  $A \dots B$ , где  $A$  – элемент с наименьшим порядковым значением данного типа, а  $B$  – элемент с наибольшим порядковым значением.

Операторы  $\leq$ ,  $\geq$ ,  $=$  и  $\langle \rangle$  подчиняются следующим правилам:

- Значение `subSet<=set1` равно `True`, если каждый элемент множества `subSet` является элементом множества `set1`. Значение `superSet>=set1` эквивалентно значению выражения `set1<=superSet`. Значение `set1=set2` равно `True`, если множества `set1` и `set2` содержат одни и те же элементы, в противном случае истинно значение выражения `set1<>set2`.
- Значение `X in set1` равно `True`, если элемент `X` входит в множество `set1`.

Выполните следующий пример.

Необходимо составить программу для вычисления среднего балла, полученного абитуриентом на вступительных экзаменах по математике, физике и химии.

Решение.

Будем использовать тип-диапазон для контроля вводимых оценок.

```
program Project2;

{$APPTYPE CONSOLE}

uses
  SysUtils;
type ball = 2..5;
var  math,phiz,chem :ball;
     ave           :real;
begin
  writeln('Enter math, phiz, chem');
  readln(math,phiz,chem);
  ave:=(math+phiz+chem)/3;
  writeln('ave = ', ave:5:2);
  readln;
end.
```

## Массивы

**Массив** представляет собой последовательность индексированных элементов одного и того же типа. Каждый элемент массива имеет уникальный индекс.

Массив с одним индексом называется **одномерным**. Математическим эквивалентом одномерного массива является вектор. **Двухмерный** массив имеет два индекса, его математический эквивалент – матрица. Существуют также **много-**



**мерные** массивы (n-мерные). Количество индексов многомерного массива больше двух. В большинстве языков программирования массивы записываются аналогично тому, как это принято в математике. Например,  $i$ -и элемент вектора  $x$  принято записывать как  $x_i$ , а  $i$ -й элемент массива  $x$  – как  $x[i]$ . Редактор кода Delphi использует стандартный набор символов ASCII, в который не входят нижние индексы, поэтому индекс вектора заменен выражением в квадратных скобках.

### Статические массивы

В Object Pascal массивы объявляются с помощью ключевых слов `array` `of`. Синтаксис объявления N-мерного **статического массива** имеет вид

```
var  
имя массива: array[тип_индекса1, ... , тип_индексаN] of базовый тип;
```

Обратите внимание: в этом синтаксисе квадратные скобки являются символами объявления массива, а не символами, обозначающими необязательную фразу синтаксиса.

Имя массива может быть любым правильным идентификатором. Базовый тип – это тип, назначаемый каждому элементу массива. Каждый индекс должен иметь порядковый тип. Общее количество элементов массива равно произведению количества элементов по каждому индексу. В качестве типа индекса чаще всего используется целый поддиапазон, однако типом индекса может быть также, например, перечислимый тип.

Для одномерного массива в объявлении используется только один тип индекса, поэтому синтаксис объявления одномерного массива имеет вид

```
var  
имя_одномерного_массива: array[тип_индекса] of базовый_тип;
```

Например, оператор

```
var  
sampleArray: array[1..10] of Integer;
```

объявляет массив `sampleArray`, содержащий 10 чисел типа `Integer`, причем значение индексов изменяется от 1 до 10.

В следующем фрагменте кода объявляются двухмерный и многомерный массивы:

```
type  
Year = (Freshman, Sophomore, Junior, Senior);  
var  
smp2DArray: array[1..10, 1..5] of Integer;
```



```
smp4DArray: array[Year, 1..20, 'A'..'F', 1..3] of Char;
```

Двухмерный массив можно представить себе как массив массивов. Аналогично этому, многомерный массив – это массив массивов, имеющих размерность на единицу меньше (например, трехмерный массив – это массив массивов массивов). Поэтому следующие объявления эквивалентны предыдущему фрагменту кода:

```
type
```

```
Year = (Freshman, Sophomore, Junior, Senior);
```

```
var
```

```
smp2DArray: array[1..10] of array[1..5] of Integer;
```

```
smp4DArray: array[Year] of array[1..20] of  
array['A'..'F'] of array[1..3] of Char;
```

В обоих случаях массив `smp2DArray` содержит  $10 \times 5 = 50$  элементов типа `Integer`, а `smp4DArray` –  $4 \times 20 \times 6 \times 3 = 1440$  элементов типа `Char`. Выбор способа объявления массива определяется стилем программирования. Однако помните: ваш стиль программирования должен быть неизменным на протяжении всего периода разработки программы.

Каждый элемент массива фактически является переменной базового типа. К этой переменной (т.е. к элементу массива) можно обращаться с помощью набора индексов. Для получения доступа к элементу массива можно использовать одну из следующих форм:

```
имя_массива[значение_индекса1, ..., значение_индексаN]
```

или

```
имя_массива[значение_индекса1] ... [значение_индексаN]
```

Например, следующий оператор присваивает значение 10 пятому элементу массива `sampleArray`:

```
sampleArray[5] := 10;
```

Как и в случае двух разных способов объявления массивов, операторы

```
smp2DArray[5, 2] := 7;
```

```
smp4DArray[Junior, 10, 'A', 1] := 'X';
```

эквивалентны операторам

```
smp2DArray[5][2] := 7;
```

```
smp4DArray[Junior][10]['A'][1] := 'X';
```

Если статический массив является формальным или фактическим параметром либо создается больше одного массива данного типа, то используйте для создания массива ключевое слово `type`. Проиллюстрируем это на примере. Приведем следующий фрагмент кода:

```
var
```

```
intArray1: array[1..10] of Integer;  
intArray2: array[1..10] of Integer;  
count: Integer;  
begin  
  for count := 1 to 10 do begin  
    intArray1[count] := count;  
  end;  
  intArray2 := intArray1;  
end;
```

Для определения эквивалентности типов переменных компилятор Object Pascal использует имена типов. Поэтому типы массивов `intArray1` и `intArray2` несовместимы. Эти массивы имеют одинаковое физическое строение, но их типы разные. Поэтому компилятор сгенерирует ошибку несовместимости типов в операторе `intArray2 := intArray1`. Чтобы исправить ошибку, нужно записать этот фрагмент кода следующим образом:

```
type  
  TenInts = array[1..10] of Integer;  
var  
  intArray1: TenInts;  
  intArray2: TenInts;  
  count: Integer;  
begin  
  for count := 1 to 10 do begin  
    intArray1[count] := count;  
  end;  
  intArray2 := intArray1;  
end;
```

### Динамические массивы

Для объявления динамического массива используется оператор `array of` без задания индексов. Синтаксис объявления **одномерного динамического массива** имеет вид

```
var  
  имя_динамического_массива: array of базовый_тип;
```

Задание размера массива и выделение для него памяти выполняется с помощью процедуры `SetLength()`.

```
SetLength(имя_динамического_массива, длина);
```

В этом синтаксисе выражение `длина` должно быть целого типа. После выполнения такого оператора значение индекса динамического массива может изменять-

ся от 0 до длина-1. Процедуру `SetLength()` в процессе выполнения программы можно вызывать произвольное количество раз. Каждый вызов приводит к изменению длины массива, причем содержимое массива сохраняется. Если при вызове `SetLength()` длина массива увеличивается, то добавленные элементы заполняются произвольными значениями, так называемым мусором. Если длина массива уменьшается, то содержимое отброшенных элементов теряется.

**Динамические массивы** являются динамическими структурами данных, поэтому по окончании работы с ними в программе должно быть предусмотрено их явное удаление из памяти компьютера. Процесс удаления ненужных динамических переменных из памяти компьютера иногда называют уборкой мусора. В Object Pascal программисту предоставлены три метода удаления динамических массивов.

1. Путем установки нулевой длины динамического массива:  
`SetLength(имя_динамического_массива, 0);`

2. Присвоением массиву (не его элементам, а индексной переменной, носящей имя массива) значения `nil`. В Object Pascal зарезервированное слово `nil` используется в качестве значения индексных (указательных) переменных. Если указательная переменная имеет значение `nil`, то она не указывает ни на какой объект. Таким образом, удалить динамический массив из памяти можно, воспользовавшись оператором – `имя_динамического_массива := nil;`

3. С помощью встроенной процедуры `Finalize()`:  
`Finalize(имя_динамического_массива);`

Для работы с динамическими массивами Object Pascal предоставляет еще несколько полезных функций. Функция `Copy()` возвращает заданную часть динамического массива.

`Copy(имя_динамического_массива, начальное_значение_индекса, количество_копируемых_элементов);`

Функции `High()` и `Low()` возвращают наибольшее и наименьшее значения индексов динамического массива, т.е. `High()` возвращает значение длина-1, а `Low()` – значение 0. Если динамический массив имеет нулевую длину, то функция `High()` возвращает -1. Синтаксис этих функций имеет вид

`High(имя_динамического_массива);`

`Low(имя_динамического_массива);`

Мы рассмотрели создание и использование одномерного динамического массива. Но как создать **многомерный динамический массив**? Как вы помните, многомерный массив – это не что иное, как массив массивов. Поэтому синтаксис объявления массива с несколькими индексами имеет вид

`var`

имя\_многомерного\_динамического\_массива :

array of array of ... array of базовый\_тип;

Структуру многомерных динамических массивов чаще всего делают прямоугольной. Однако это совсем не обязательно. Например, для решения некоторых задач используются так называемые треугольные массивы, в которых один из индексов всегда равен или больше другого индекса. Если заранее известно, что в задаче используются только элементы, расположенные на диагонали или под диагональю массива, то нет никакого смысла расходовать память на элементы, расположенные над диагональю. На рис. 1 показана физическая структура такого массива. Обратите внимание: строки массива содержат разное количество элементов. Как видите, многомерные динамические массивы – довольно гибкие структуры данных. В приведенном ниже фрагменте кода создается двухмерный треугольный массив, структура которого показана на рис. 1.

```
var
    smp2DdynArray: array of array of Integer;
    count: Integer;
begin
    SetLength(smp2DdynArray, 3);
    for count = 1 to n do begin
        SetLength(smp2DdynArray[count - 1], count * 2);
    end;
    {В этом месте массив smp2DdynArray можно использовать}
end;
```

Первый индекс	Второй индекс					
	0	1	2	3	4	5
0						
1						
2						

Рис. 1. Пример треугольного динамического массива

Имя динамического массива обозначает переменную, фактически являющуюся указателем. Это значит, что такая переменная содержит не данные, а адрес первого элемента массива. Говорят, что она указывает на первый элемент массива. Индексы определяют смещение требуемого элемента массива относительно первого. Рассмотрим следующий фрагмент кода:

```
var
    array1, array2: array of Integer;
    check: Boolean;
```

```
begin
  SetLength(array1, 1);
  SetLength(array2, 1);
  array1[0] := 5;
  array2[0] := 5;
  check := (array1 = array2);
  ...
end ;
```

Какое значение принимает переменная `check`? Если бы это был статический массив, то значение `check` было бы равно `True`, потому что массивы содержат одну и ту же информацию. Однако для динамических массивов (как в данном случае) значение `check` равно `False`, так как указатели `array1` и `array2` ссылаются на разные области памяти. Значение указателя равно адресу первого по счету элемента массива, а поскольку эти массивы хранятся в разных местах, то, естественно, указатели имеют разное значение. Для сравнения динамических массивов необходимо сравнивать в цикле каждую пару элементов отдельно. Похожая проблема возникает и в следующем фрагменте кода. Можете ли вы обнаружить ее?

```
var
  array1, array2: array of Integer;
begin
  SetLength(array1, 1);
  SetLength(array2, 1);
  array1[0] := 5;
  array2 := array1;
  array2[0] := 10;
end;
```

Какие значения принимают элементы `array1[0]` и `array2[0]`? Обратите внимание: значение `array1` присвоено `array2`. Другими словами, `array2` теперь ссылается на ту же область памяти, что и `array1`. Поэтому оба элемента `array1[0]` и `array2[0]` ссылаются на одну и ту же область памяти, содержащую значение 10 (присвоенное этой области последним оператором).

## Задачи

1. Объявить массив из 10 целых чисел. Инициализировать массив значениями введенными с клавиатуры. Найти в массиве первое отрицательное число и вывести его на экран. Отсортировать элементы массива любым из приведенных

ниже способов. Вывести результат на экран. В программе должны быть использованы циклы всех 3-х типов.

2. С клавиатуры вводится строка. Напечатать по одному разу все буквы, входящие в эту строку. При решении задачи использовать множественный тип.

## Контрольные вопросы

1. Чем отличаются статические структуры данных от динамических?
2. Что такое перечислимый тип? Приведите пример перечислимого типа.
3. Что такое множество в Object Pascal? Приведите пример множества.
4. Напишите операторы объявления одномерного, двумерного, трехмерного и семимерного массивов.
5. Что является математическим эквивалентом одномерного массива? Двухмерного массива?

## Задачи для самостоятельного выполнения

### Циклы с параметром (циклы ДЛЯ)

1. Подсчитать число и сумму положительных, число и произведение отрицательных элементов заданного массива  $A(N)$ .
  2. Заданные векторы  $X(N)$  и  $Y(N)$  преобразовать по правилу: большее из  $x_i$  и  $y_i$  принять в качестве нового значения  $x_i$ , а меньшее - в качестве нового значения  $y_i$ .
  3. Вычислить сумму квадратов всех элементов заданного массива  $X(N)$ , за исключением элементов, кратных пяти.
  4. В заданном массиве  $A(N)$  поменять местами наибольший и наименьший элементы.
  5. Осуществить циклический сдвиг компонент заданного вектора  $A(N)$  вправо на две позиции, то есть получить вектор  $A = (a_{N-1}, a_N, a_1, a_2, \dots, a_{N-2})$ .
  6. Вывести на печать номера элементов заданного массива  $Y(N)$ , удовлетворяющих условию  $0 < y_i < 1$ .
  7. В заданном массиве  $A(N)$  положительные элементы уменьшить вдвое, а отрицательные заменить на значения их индексов.
  8. Образовать массив  $B$ , состоящий из положительных элементов заданного массива  $A(N)$ , больших пяти. Вывести на печать образованный массив и число его элементов.
- Сжать заданный массив  $A(N)$  отбрасыванием нулевых элементов.

**Вложенные циклы с параметром (циклы for)**

1. Дана матрица  $A(N,M)$ . Найти её наибольший элемент и номера строки и столбца, на пересечении которых он находится.
2. В каждой строке заданной матрицы  $A(N,M)$  вычислить сумму, количество и среднее арифметическое положительных элементов.
3. Для заданной целочисленной матрицы  $A(N,M)$  определить, является ли сумма её элементов чётным числом, и вывести на печать соответствующий текст.
4. Дана целочисленная матрица  $A(N,M)$ . Вычислить сумму и произведение тех её элементов, которые при делении на два дают нечётное число.
5. В заданной матрице  $A(N, M)$  поменять местами столбцы с номерами  $P$  и  $Q$ .
6. Дана матрица  $A(N,M)$ . Получить вектор  $X(M)$ , равный  $P$ -й строке матрицы, и вектор  $Y(N)$ , равный  $Q$ -му столбцу матрицы.
7. Дана матрица  $A(N,M)$ . Поменять местами её наибольший и наименьший элементы.
8. Дана матрица  $A(N,N)$ . Переписать элементы её главной диагонали в одномерный массив  $Y(N)$  и разделить их на максимальный элемент главной диагонали.
9. Найти наибольший элемент побочной диагонали заданной матрицы  $A(N, N)$  и вывести на печать всю строку, в которой он находится.
10. Дана матрица  $A(N,N)$  и целое  $P$ . Преобразовать матрицу по правилу: строку с номером  $P$  сделать столбцом с номером  $P$ , а столбец с номером  $P$  сделать строкой с номером  $P$ .
11. В заданном массиве  $A(N,N)$  вычислить две суммы элементов, расположенных выше и ниже побочной диагонали.

**Циклы с условием (циклы ПОКА)**

1. Проверить, есть ли в заданной целочисленной последовательности  $a_1, a_2, \dots, a_N$  элементы, равные нулю. Если есть, найти номер первого из них, если нет – выдать соответствующий текст.
2. Выяснить, имеются ли в заданном векторе  $A(N)$  два подряд идущих нулевых элемента.
3. Выяснить, имеются ли в заданном целочисленном векторе  $A(N)$  три подряд идущих элемента одного знака.
4. Имеется последовательность чисел  $a_1, a_2, \dots, a_N$ . Найти сумму первых из них (считая слева направо), произведение которых не превышает заданно-



го числа  $M$ .

5. Определить, имеются ли среди элементов побочной диагонали заданной целочисленной матрицы  $A(N,N)$  числа, равные нулю.
6. Если в заданном целочисленном векторе  $A(N)$  есть элементы со значением, равным заданному числу  $B$ , то переменной  $C$  присвоить значение, равное сумме всех элементов, предшествующих первому по порядку такому элементу; в противном случае вывести соответствующий текст.
7. Дана последовательность из  $N$  целых чисел. Определить, со скольких положительных чисел она начинается.
8. Дано натуральное  $N$ . Выяснить, сколько цифр оно содержит.
9. Найти сумму цифр заданного натурального числа.
10. Цифры заданного натурального числа записать в обратном порядке.

### **Вложенные циклы с условием (циклы ПОКА)**

1. В заданной целочисленной матрице  $A(N,M)$  вывести на печать индексы первого положительного элемента, кратного заданному числу  $K$ . Если таких элементов в матрице нет, то вывести соответствующий текст.  
Элементы матриц просматривать слева направо и сверху вниз.
2. В заданной целочисленной матрице  $A(N,M)$  заменить первый отрицательный элемент максимальным элементом матрицы. Если отрицательных элементов нет, то вывести соответствующий текст.
3. В заданной матрице  $A(N,N)$  обнулить строку, в которой находится первый отрицательный элемент. Элементы матриц просматривать слева направо и сверху вниз.
4. Из заданной матрицы  $A(N,N)$  удалить строку и столбец, в которых находится первый элемент, равный нулю. Полученную матрицу уплотнить.  
Элементы матриц просматривать слева направо и сверху вниз.
5. Если в заданной матрице  $A(N,N)$  есть хотя бы один элемент, больший ста, то элементы обеих диагоналей заменить нулями.

### **Комбинация циклов с параметром и с условием**

1. Дана матрица  $A(N,N)$ . Переменной  $B$  присвоить значение, равное количеству строк матрицы  $A$ , содержащих хотя бы одну нулевую компоненту.
2. Дана матрица  $B(N,N)$ . Получить вектор  $A(N)$ , компоненты которого находятся по правилу:  $A_i$  равно первому по порядку положительному элементу в  $i$ -ой строке матрицы (если таких элементов в строке нет, то принять  $A_i = -1$ ).

3. В заданной матрице  $A(N, M)$  найти количество строк, не содержащих отрицательных чисел.
4. Дана матрица  $A(N, M)$ . Построить вектор  $B(N)$ , элементы  $B_i$  которого равны единице, если элементы  $i$ -ой строки образуют упорядоченную по убыванию или по возрастанию последовательность, и нулю во всех остальных случаях.
5. Подсчитать количество различных (не повторяющихся) чисел, встречающихся в заданном целочисленном массиве  $A(N)$ .
6. Подсчитать количество различных (не повторяющихся) чисел, встречающихся в заданной целочисленной матрице  $A(N, M)$ .

### Приложение. Методы сортировки массивов.

#### МЕТОД ПРЯМОГО ВКЛЮЧЕНИЯ

Идея алгоритма состоит в следующем. Элементы массива просматриваются по одному, и каждый следующий элемент из числа еще неупорядоченных вставляется в подходящее место среди ранее упорядоченных элементов. Очевидно, что на  $N$ -ом шаге упорядоченными окажутся ровно  $N$  элементов.

#### Пример.

Пусть задано 8 чисел: 27, 412, 71, 81, 59, 14, 273, 87. На каждом шаге берем очередной элемент и сравниваем его поочередно с ранее упорядоченными, пока не встретится больший данного. В таблице звездочкой отмечена граница упорядоченных элементов.

Итерация	Массив
нач.сост.	027 412 071 081 059 014 273 087
1	027* 412 071 081 059 014 273 087
2	027 412* 071 081 059 014 273 087
3	027 071 412* 081 059 014 273 087
4	027 071 081 412* 059 014 273 087
5	027 059 071 081 412* 014 273 087

6	014 027 059 071 081 412* 273 087
7	014 027 059 071 081 273 412* 087
8	014 027 059 071 081 087 273 412*

### МЕТОД ПРЯМОГО ВЫБОРА (ЛИНЕЙНОЙ СОРТИРОВКИ)

Пусть задан массив из N чисел. Метод предлагает для сортировки массивов последовательность шагов, на каждом из которых из массива выбирается и ставится на свое место один элемент.

а) Находим в массиве наименьший элемент.

б) Наименьший элемент меняем местами с первым. Найденный элемент уже занял то место, на котором он должен находиться после упорядочения. Поэтому дальше его можно не рассматривать и продолжать сортировку в укороченном на один элемент массиве.

в) Последовательно повторяем пункты а) и б) для массивов, укорачиваемых каждый раз на один элемент до тех пор, пока в новом массиве не останется один (самый большой элемент), который уже будет расположен на своем месте.

#### Пример.

Для приведенного выше массива из 8 чисел последовательность сортировки описывается следующей таблицей (звездочка - граница отсортированной части).

Итерация	Массив
нач.сост.	27 412 71 81 59 14 273 87
1	14* 412 71 81 59 27 273 87
2	14 27* 71 81 59 412 273 87
3	14 27 59* 81 71 412 273 87
4	14 27 59 71* 81 412 273 87
5	14 27 59 71 81* 412 273 87
6	14 27 59 71 81 87* 273 412
7	14 27 59 71 81 87 273* 412
8	14 27 59 71 81 87 273 412*

Основное различие между методами прямого включения и прямого выбора состоит в том, что в первом случае для вставки элемента просматривается уже упорядоченная часть, а во втором случае - еще неупорядоченная.

### Недостатки метода сортировки выбором

В методе не отслеживается, каком состоянии находится исходный массив. Количество необходимых операций сравнения и проходов не зависит от первоначального расположения элементов. Оно зависит только от числа элементов в массиве. Т.е. оно одно и то же, как в случае изначально упорядоченного массива, так и для самого произвольного расположения элементов. Только для изначально упорядоченного массива не потребуется выполнять перестановки элементов.

### МЕТОД ПУЗЫРЬКА (ПРЯМОГО ОБМЕНА)

Метод пузырька получил свое название благодаря тому, что на каждом шаге наименьший из неупорядоченный элемент "всплывает" к левому краю. Идея метода основана на том, что в упорядоченном массиве к а ж д о е значение находится в правильном положении относительно непосредственно следующего за ним. Этот факт является основой метода прямого обмена, основанного на сравнении соседних элементов.

Метод реализуется следующим алгоритмом.

а) На первом шаге все элементы массива, начиная с последнего, сравниваются с предшествующим. Если элемент  $a[j]$  оказывается меньше, чем элемент  $a[j-1]$ , то элементы меняются местами. Далее элемент, стоящий теперь на  $(j-1)$ -ом месте, сравнивается с элементом  $a[j-2]$  и т.д. После завершения первого шага можно гарантировать, что наименьший элемент массива будет самым первым (помещен в самый левый край массива).

б) Далее выполняются те же действия, но процесс заканчивается перед первым (вторым, третьим и т.д.) элементом, который уже находится на своем месте. Таким образом, на каждом новом шаге наименьший из неотсортированных элементов попадает в самую левую позицию неотсортированной части массива. Процесс заканчивается, когда не будет зарегистрирован проход, на котором не было ни одной перестановки.

#### Пример.

При сортировке методом пузырька последовательность шагов будет следующей.

Итерация	Массив
нач. сост.	27 412 71 81 59 14 273 87
1	14* 27 412 71 81 59 87 273
2	14 27* 59 412 71 81 87 273
3	14 27 59* 71 412 81 87 273

4	14 27 59 71* 81 412 87 273
5	14 27 59 71 81* 87 412 273
6	14 27 59 71 81 87* 273 412
7	14 27 59 71 81 87 273* 412

На первом шаге последовательно выполняются следующие перестановки: 87<-->273, 14<-->59, 14<-->81, 14<-->71, 14<-->412, 14<-->27.

На втором шаге значение 59 последовательно меняется местами с 81, 71 и 412.

Метод прямого обмена по своей идее похож метод прямых вставок. Так как в нем акцент делается на поиске наименьших (наиболее "легких" элементов), то эти элементы всплывают значительно быстрее чем самые тяжелые опускаются на дно.

Метод прямого обмена работает тем эффективнее, чем правильней расположены элементы в исходном состоянии массива. Не существует способа, который бы позволил предсказать, за сколько проходов N элементов будут полностью отсортированы. Если массив упорядочен с самого начала, то потребуется всего один проход. Если элементы расположены в обратном порядке (по убыванию), то N проходов.

### Список использованной литературы

1. Фаронов В.В. Delphi 3. Учебный курс. М.: «Нолидж», 1998. 400 с.
2. Галисеев Г.В. Программирование в среде Delphi 8 for .NET. М.: Издательский дом «Вильямс», 2004. 304 с.
3. Павловска Т.А. Паскаль. Программирование на языке высокого уровня. СПб.: Питер, 2003. 393 с.
4. Абрамов С.А. и др. Задачи по программированию. М.: Наука, 1988. 224 с.