Е.А. Корчевская, С.А. Ермоченко

О С Н О В Ы ПРОГРАММИРОВАНИЯ И АЛГОРИТМИЗАЦИИ

Учебно-методическое пособие

2009

УДК 004.43(075)
ББК 32.973.26-018.1я73
К70

Авторы: доцент кафедры прикладной математики и механики УО «ВГУ им. П.М. Машерова», кандидат физико-математических наук **Е.А. Корчевская**; преподаватель кафедры прикладной математики и механики УО «ВГУ им. П.М. Машерова» **С.А. Ермоченко**

Р е ц е н з е н т:
заведующий кафедрой информатики и информационных технологий
УО «ВГУ им. П.М. Машерова», кандидат физико-математических наук, доцент *А.И. Бочкин*

Научный редактор: заведующий кафедрой прикладной математики и механики УО «ВГУ им. П.М. Машерова», кандидат физико-математических наук, доцент *Л.В. Маркова*

В учебно-методическом пособии изложены основы программирования и алгоритмизации на языке Object Pascal. Излагаются общие методические указания, которых следует придерживаться при изучении теоретического материала, выполнении практических заданий. Приведено много примеров и задач, решения которых сопровождаются соответствующими методическими указаниями.

Предназначается для студентов специальностей физико-математического профиля.

УДК 004.43(075)
ББК 32.973.26-018.1я73

С О Д Е Р Ж А Н И Е

В В Е Д Е Н И Е

В учебно-методическом пособии собраны теоретические сведения и примеры из практики программирования на языке Object Pascal. Материал разбит на 11 разделов, в которых изложены теоретические сведения по программированию на языке Object Pascal. Также приведены примеры решения наиболее часто встречающихся на практике задач. Все примеры тестировались в среде Borland Delphi 7.0, которая использовалась для создания консольных приложений операционной системы Windows.

В отдельном разделе «Лабораторный практикум» приведен список задач по большинству рассмотренных в учебно-методическом пособии тем.

Материал учебного издания соответствует рабочей программе курса «ЭВМ и программирование» первого семестра. Учебно-методическое пособие может применяться как преподавателями для выявления уровня усвоения материала студентами и для проведения контроля их знаний, так и студентами как дополнительный материал к курсу лекций и как материал для самостоятельной работы.

Данное издание не является справочником по языку Object Pascal или по среде разработки Borland Delphi. Здесь приведены лишь базовые знания, необходимые для усвоения начального уровня программирования и понимания основ составления алгоритмов. Характер подобранного материала позволяет использовать учебное издание и для изучения основ других языков программирования, так как все приведенные в нем примеры, лишь демонстрируют общие принципы программирования, не зависящие от языка программирования или платформы разработки. Язык Object Pascal, рассмотренный в учебно-методическом пособии, взят лишь как наиболее понятный для демонстрации всех базовых алгоритмов.

1. АЛФАВИТ И ИДЕНТИФИКАТОРЫ ЯЗЫКА OBJECT PASCAL

Алфавит языка Object Pascal включает буквы, цифры, шестнадцатеричные цифры, специальные символы, пробелы и зарезервированные слова.

Буквы − это буквы латинского алфавита от `a` до `z` и от `A` до `Z`, а также знак подчеркивания «_». В языке нет различия между заглавными и строчными буквами алфавита, если только они не входят в символьные и строковые выражения.

Цифры − арабские цифры от `0` до `9`.

Каждая ***шестнадцатеричная цифра*** имеет значение от 0 до 15. Первые 10 значений обозначаются арабскими цифрами `0` − `9`, остальные шесть латинскими буквами `A` − `F` или `a` − `f`.

Специальные символы Delphi − это символы:

+ − * / = , ′ . : ; < > ( ) { } [ ] ^ @ $ #

К специальным символам относятся также следующие пары символов: **<> <= >= := (* *) (. .) //**

В программе эти пары символов нельзя разделять пробелами, если они используются как знаки операций отношения или ограничители комментария. Символы `(.` и `.)` могут употребляться соответственно вместо `[` и `]`.

К ***пробелам*** относятся любые символы в диапазоне кодов от 0 до 32. Эти символы рассматриваются ограничители идентификаторов, констант, чисел, зарезервированных слов. Несколько следующих друг за другом пробелов считаются одним пробелом (последнее не относится к строковым константам).

Зарезервированные слова − это английские слова, указывающие компилятору на необходимость выполнения определенных действий. Зарезервированные слова не могут использоваться в программе ни для каких иных целей кроме тех, для которых они предназначены. Например, зарезервированное слово **`begin`** означает для компилятора начало составного оператора. Программист не может создать в программе переменную с именем `begin`, константу `begin`, метку `begin` или вообще какой бы то ни было другой элемент программы с именем `begin`.

Например: **and, array, begin, case, const, constructor, destructor, div, do, downto, else, end** и др.

Зарезервированные слова в окне кода Delphi выделяются жирным шрифтом.

Комментарии ничего не значат для компилятора, он их игнорирует. Наличие комментариев делает программу понятнее, т.к. с их по-

мощью поясняются те или иные места программы. В Delphi в качестве ограничителей комментария используются символы: `{ }`, `(* *)`, `//`.

Идентификаторы − это имена констант, переменных, меток, типов, объектов, классов, свойств, процедур, функций, модулей, программ и полей в записях. Идентификаторы могут иметь произвольную длину. Идентификатор всегда начинается буквой, за которой могут следовать буквы и цифры. Буквой считается также символ подчеркивания, поэтому идентификатор может начинаться этим символом и даже состоять только из одного или нескольких символов подчеркивания. Пробелы и специальные символы алфавита не могут входить в идентификатор.

Примеры **правильных** идентификаторов:

```
_a
date_sep_2007
```

Примеры **неправильных** идентификаторов:

```
if // зарезервированное слово
Point*// содержит специальный символ
My Program// содержит пробел
```

2. ТИПЫ ДАННЫХ

Типы данных − это специальные конструкции языка, которые рассматриваются компилятором как образцы для создания других элементов программы, таких как переменные, константы и функции. Любой тип определяет две важные для компилятора вещи: объем памяти, выделяемый для размещения элемента (константы, переменной или возвращаемого функцией результата), и набор допустимых действий, которые программист может совершать над элементами данного типа. Любой определяемый программистом идентификатор должен быть описан в разделе описаний (перед началом исполняемых операторов). Это означает, что компилятор должен знать тот тип (образец), по которому создается определяемый идентификатором элемент.

Рассмотрим простые типы данных, к которым можно отнести порядковые и вещественные типы.

2.1. Порядковые типы

Порядковые типы отличаются тем, что каждый из них имеет конечное количество возможных значений. Эти значения можно определенным образом упорядочить и, следовательно, с каждым из них можно сопоставить некоторое целое число − порядковый номер значения.

К порядковым типам относятся:

Целые.

Логические.

Символьный.

Перечисленный.

Тип-диапазон.

К любому из них применима функция `ord (X)`, которая возвращает порядковый номер значения выражения `x`. Для целых типов функция `ord(x)` возвращает само значение `x`, т.е. `ord(x) = x` для `x`, принадлежащего любому целому типу. Применение `ord(x)` к логическому, символьному и перечисленному типам дает положительное целое число в диапазоне от 0 до 1 (логический тип), от 0 до 255 (символьный), от 0 до 65 535 (перечисленный). Тип-диапазон сохраняет все свойства базового порядкового типа, поэтому результат применения к нему функции `Ord(X)` зависит от свойств этого типа.

К порядковым типам можно также применять следующие функции:

`Pred(X)` − возвращает предыдущее значение порядкового типа (значение, которое соответствует порядковому номеру `Ord(X) - 1`), т.е.

```
Ord(Pred(X)) = Ord(X) - 1
```

Succ(X) − возвращает следующее значение порядкового типа, которое соответствует порядковому номеру `Ord(x) +1`, т.е.

```
Ord(Succ(X))= Ord(X)+1
```

2.1.1. Целочисленные типы данных

В переменных целого типа информация представлена в виде чисел, не имеющих дробной части. Они используются для математических вычислений и любых других операций, где нужна работа с числами.

Существует несколько видов целых типов данных. Они в основном отличаются только размером отводимой переменным памяти для хранения целых чисел.

В таблице перечислены все типы целых чисел. В примечании указано, какого типа могут быть эти числа − со знаком или без (т.е. только положительные или могут быть как положительными, так и отрицательными).

В зависимости от объема памяти, отводимого переменной для хранения данных, определяется максимальное число, которое можно записать в эту переменную.

Название	**Длина, байт**	**Диапазон значений**	**Примечание**
Byte	1	0 – 255	Без знака
ShortInt	1	-128 – +127	Знаковое
SmallInt	2	-32 768 – +32 767	Знаковое
Word	2	0 – 65 535	Без знака
Integer	4	-2 147 483 648 – +2 147 483 647	Знаковое
LongInt	4	-2 147 483 648 – +2 147 483 647	Знаковое
LongWord	4	0 – 4 294 967 295	Без знака
Int64	8	-2^{63} – $+2^{63}-1$	Знаковое
Cardinal	4	0 – 4 294 967 295	Без знака

2.1.2. Логические типы

К логическим относятся типы:

Boolean
ByteBool
Bool
WordBool
LongBool

Типы `Boolean` и `ByteBool` занимают по одному байту каждый, `Bool` и `WordBool` – по 2, `LongBool` – 4 байта. Значениями логического типа может быть одна предварительно объявленных констант `False` (Ложь) или `True` (Истина).

2.1.3. Символьный тип

Значениями символьного типа является множество всех символов компьютера. Каждому символу приписывается целое число, которое служит кодом внутреннего представления символа, его возвращает функция `ord()`.

Переменная символьного типа может получить значение в результате выполнения инструкции присваивания или ввода (`read()`, `readln()`). Если переменная символьного типа получает значение в результате выполнения операции присваивания, то справа от знака `:=` должно стоять выражение символьного типа, например, переменная типа `char` или символьная константа – символ, заключенный в кавычки.

К символьному типу применимы операции отношения. Переменную символьного типа можно сравнить с другой переменной символьного типа или с символьной константой. Сравнение основано на том, что каждому символу поставлено в соответствие число.

Например:

'0'<'1'<...<'9'<...'A'<'B'<...<'Z'<'a'<'b'<...<'z'

Символам русского алфавита соответствуют числа большие, чем символам латинского алфавита, при этом справедливо следующее:

'А'<'Б'<...<'Я'<'а'<'б'<...<'я'

Каждый символ кодируется числом. Однако не все символы есть на клавиатуре. Например, на клавиатуре нет символов, с помощью которых рисуются рамки. Если в программе нужно вывести на экран символ, которого нет на клавиатуре, то можно воспользоваться функцией `chr()`, возвращающей в качестве значения символ, код которого указывается при обращении к функции.

В Delphi используется 8-битная расширенная таблица символов, где задействованы все 8 бит (ANSI-таблица). Эта таблица берется из самой операционной системы Windows. Таким образом, количество символов и их расположение зависит от ОС.

Для того чтобы удовлетворить все национальности, ввели поддержку UNICODE (16-битная таблица символов). В ней первые 8 бит совпадают с таблицей ANSI, а остальные являются специфичными.

Язык Delphi поддерживает три символьных типа: `Ansichar`, `Char` и `Widechar`:

тип **`Ansichar`** – это символы в кодировке ANSI, которым соответствуют числа в диапазоне от 0 до 255;

тип **`Widechar`** – это символы в кодировке Unicode, им соответствуют числа от 0 до 65 535;

универсальный символьный тип **`Char`**, который эквивалентен `Ansichar`.

К символьному типу применимы встроенные функции:

`chr (B)` – функция типа `char`; преобразует выражение `B` типа `Byte` в символ и возвращает его своим значением;

`upCase(CH)` – функция типа `char`; возвращает прописную букву, если `CH` – строчная латинская буква, в противном случае возвращает сам символ `CH`.

2.1.4. Перечисленный тип

Перечисленный тип задается перечислением тех значений, которые он может получать. Каждое значение именуется некоторым идентификатором и располагается в списке, обрамленном круглыми скобками, например:

```
type
 TColor = (Red,Yellow,Green);
```

Примеры:

```
type
 TDayOfWeek = (MON, TUE, WED, THU, FRI, SAT,
SUN);

var
 ThisDay, LastDay: TDayOfWeek;
```

Соответствие между значениями перечисленного типа и порядковыми номерами этих значений устанавливается порядком перечисления: первое значение в списке получает порядковый номер 0, второе – 1 и т.д. Максимальная мощность перечисленного типа составляет 65 536 значений, поэтому фактически перечисленный тип задает некоторое подмножество целого типа `word` и может рассматриваться как компактное объявление сразу группы целочисленных констант со значениями 0,1 и т.д.

Пусть, например, заданы такие перечисленные типы:

```
type
 colors =(black, red, white);
```

```
ordenal=(one, two, three);
days = (Monday, Tuesday, Wednesday);
```

С точки зрения мощности и внутреннего представления все три типа эквивалентны:

```
Ord(black)=0,..., Ord(white)=2,
Ord(one)=0,..., Ord(three)=2,
Ord(Monday)=0,..., Ord(Wednesday)=2.
```

Помимо указания значений, которые может принимать переменная, описание типа задает, как значения соотносятся друг с другом. Считается, что самый левый элемент списка значений является минимальным, а самый правый − максимальным. Для элементов типа `DayOfWeek` справедливо:

```
MON < TUE < WED < THU < FRI < SAT < SUN
```

Во время компиляции Delphi проверяет соответствие типа переменной типу выражения, которое присваивается переменной. Если тип выражения не может быть приведен к типу переменной, то выводится сообщение об ошибке.

В Delphi допускается и обратное преобразование: любое выражение типа word можно преобразовать в значение перечисленного типа, если только значение целочисленного выражения не превышает мощности этого типа.

Например, для рассмотренного выше объявления типов эквивалентны следующие присваивания:

```
col := black;
col := colors(0);
```

Переменные любого перечисленного типа можно объявлять без предварительного описания этого типа,

Например:

```
var
col: (black, white, green);
```

2.1.5. Тип-диапазон

Тип-диапазон есть подмножество своего базового типа, в качестве которого может выступать любой порядковый тип, кроме типа-диапазона.

Тип-диапазон задается границами своих значений внутри базового типа:

<мин.знач.>..<макс.знач.>

Пример:

```
type
TIndex = 0..100;
TRusChar = 'А' .. 'я';
```

Тип-диапазон необязательно описывать в разделе type, а можно указывать непосредственно при объявлении переменной, например:

```
var
date : 1..31;
ichr : 'A'..'Z';
```

В стандартную библиотеку Delphi включены две функции, поддерживающие работу с типами-диапазонами:

High(x) – возвращает максимальное значение типа-диапазона, к которому принадлежит переменная x;

Low(x) – возвращает минимальное значение типа-диапазона.

В качестве базового можно использовать перечисленный тип, созданный программистом. В следующем фрагменте на основе типа TMonth объявлен интервальный тип TSammer:

```
type
 TMonth = (Jan, Feb, Mar, Apr, May, Jun, Jul,
Aug, Sep, Oct, Nov, Dec);
 TSammer = Jun.. Aug;
```

2.2. Вещественные типы

В отличие от порядковых типов, значения которых всегда сопоставляются с рядом целых чисел и, следовательно, представляются в ПК абсолютно точно, значения вещественных типов определяют произвольное число лишь с некоторой конечной точностью, зависящей от внутреннего формата вещественного числа. Вещественные или дробные типы данных предназначены для хранения чисел с плавающей точкой. Некоторые считают, что лучше использовать именно такие типы вместо целочисленных. Это заблуждение. Операции с плавающей точкой отнимают у процессора больше времени и требуют больше памяти. Поэтому используйте переменные этого типа только там, где это действительно необходимо.

Тип	Диапазон	Значащих цифр	Байтов
Real48	$2.9*10^{-39}$…$1.7*10^{38}$	11-12	06
Real	$5.0*10^{-324}$ …$1.7*10^{308}$	15-16	08
Single	$1.5 * 10^{-45}$…$3.4* 10^{38}$	7-8	04
Double	$5.0*10^{-324}$ …$1.7*10^{308}$	15-16	08
Extended	$3.6*10^{-4951}$ …$1.1*10^{4932}$	19-20	10
Comp	-2^{63} …$2^{63}-1$	19-20	08
Currency	-922 337 203 685 477.5808 … 922 337 203 685 477.5807	19-20	08

Вещественное число в Delphi занимает от 4 до 10 смежных байт.

Мантисса имеет длину от 23 (для Single) до 63 (для Extended) двоичных разрядов. Десятичная точка (запятая) подразумевается перед левым (старшим) разрядом мантиссы, но при действиях с числом ее положение сдвигается влево или вправо в соответствии с двоичным порядком числа, хранящимся в экспоненциальной части, поэтому действия над вещественными числами называют арифметикой с плавающей точкой (запятой).

Особое положение в Delphi занимают типы Comp и Currency, которые трактуются как вещественные числа с дробными частями фиксированной длины: в Comp дробная часть имеет длину 0 разрядов, т.е. просто отсутствует, в Currency длина дробной части − 4 десятичных разряда. Фактически, оба типа определяют большое целое число со знаком (во внутреннем представлении они занимают 8 смежных байтов). В то же время в выражениях типы Comp и Currency полностью совместимы с любыми другими вещественными типами: над ними определены все вещественные операции, они могут использоваться как аргументы математических функций и т.д.

Очень важно помнить, что вещественные числа не равны целым. Например, вещественное число 3.0 не будет равно целому числу 3. Для того чтобы сравнить оба этих числа, нужно округлить вещественное число.

Например, если у вас есть какая-то формула, в которой используется деление, то результат ее выполнения будет почти всегда дробным, даже если вы уверены в целостности ответа. Например, вы дели-

те 10 на 2, и должен получиться результат 5. Хотя результат целое число, компилятор будет представлять его как дробное.

```
var
 i: Integer;
begin
 i:=10/2;
end;
```

Если вы попытаетесь откомпилировать такой код, то увидите следующую запись об ошибке: 'Incompatible types: 'Integer' and 'Extended' (Несовместимые типы: целое число и дробное).

Тут появляется два выхода:

1. Записывать результат в переменную вещественного типа.
2. Использовать функции преобразования.

Функции преобразования

`Round(n)`	Целое, полученное путем округления `n` по известным правилам
`Trunc(n)`	Целое, полученное путем отбрасывания дробной части `n`
`Frac(n)`	Дробное, представляющее собой дробную часть вещественного `n`
`Int(n)`	Дробное, представляющее собой целую часть вещественного `n`

2.3. Строковый тип данных

Строки могут быть представлены следующими типами:

1. **`Shortstring`** или **`String[N], N<=255`** (короткая строка),
2. **`String`** (длинная строка)**,**
3. **`Widestring`** (широкая строка)**.**

Используя операции =, <, >, <=, >=, переменную строкового типа можно сравнить с другой переменной строкового типа или со строковой константой. Строки сравниваются посимвольно, начиная с первого символа с учетом внутренней кодировки символа. Если одна строка меньше другой по длине, то недостающие символы короткой строки заменяются значением `#0`. Если все символы сравниваемых строк одинаковые, то такие строки считаются равными.

Приведем несколько полезных при работе со строками функций и процедур.

Функция **Concat(S1, S2, …, Sn)** возвращает строку, представляющую собой сцепление строк-параметров S1, S2, …, Sn.

Функция **length(Строка)** возвращает длину строки. У этой функции один параметр − выражение строкового типа.

Процедура **delete(Строка, p, n)** позволяет удалить часть строки, где:

Строка − переменная или константа строкового типа;

p − номер символа, с которого начинается удаляемая подстрока;

n − длина удаляемой подстроки.

Функция **pos(Подстрока, Строка)** позволяет определить положение подстроки в строке, где:

Подстрока − строковая константа или переменная, которую надо найти;

Строка – строка, в которой ведется поиск.

Результатом является порядковый номер, начиная с которого найдена нужная строка.

Функция **copy(Строка, p, n)** позволяет выделить фрагмент строки, где:

строка − выражение строкового типа, содержащее строку, фрагмент которой надо получить;

p – номер первого символа, с которого начинается выделяемая подстрока;

n – длина выделяемой подстроки.

Функция **Insert(Строка1, Строка2, p)** вставляет одну строку в другую, начиная с указанного символа, где:

Строка1 – строка, которую надо вставить,

Строка2 – строка, в которую надо вставить,

p – позиция, куда надо вставить.

Процедура **Val(Строка, X, Code)** преобразует строку символов **Строка** во внутреннее представление целой или вещественной переменной **X**, которое определяется типом этой переменной. Фактически процедура позволяет преобразовать изображение числа в число. Параметр **Code** содержит ноль, если преобразование прошло успешно, и тогда в X помещается результат преобразования, в противном случае он содержит номер позиции в строке, где обнаружен ошибочный символ, и в этом случае содержимое **X** не меняется.

Процедура **Str(X, Строка)** преобразует число **X** любого вещественного или целого типов в строку символов **Строка**.

3. ОБЩАЯ СТРУКТУРА ПРОГРАММ

Текст программы начинается зарезервированным словом **Program** и заканчивается словом **end** со следующей за ним точкой.

Условно программу можно разделить на 3 части:

1. Заголовок программы.
2. Раздел описаний.
3. Тело программы.

Слово **Program** со следующим за ним именем программы и точки с запятой образуют **заголовок программы**.

Далее следует **раздел описаний**, в котором описываются используемые в программе идентификаторы. Идентификаторы обозначают элементы программ, такие как типы, переменные, процедуры, функции.

Тело программы заключается в операторные скобки **begin end**.

```
Program Имя;
Uses {список использованных модулей}
Const (*список использованных констант*)
Label //используемые метки
Type //объявляемые типы
Var //объявляемые глобальные переменные
Procedure Имя;
Function Имя; // описание подпрограмм
Begin
{исполняемые операторы}
End.
```

В разделе **uses** программист сообщает компилятору о модулях, которые необходимо рассматривать как составные части программы.

Константы определяют области памяти, которые не могут изменять своего значения в ходе работы программы. Как и любые другие элементы программы, константы могут иметь свои собственные имена.

```
const
 K = 1024;
 M = K*K;
```

В качестве констант в Delphi могут использоваться целые, вещественные и шестнадцатеричные числа, логические константы, символы, строки символов, конструкторы множеств.

Целые числа записываются со знаком или без него по обычным правилам.

Вещественные числа записываются со знаком или без него с использованием десятичной точки и/или экспоненциальной части. Экспоненциальная часть начинается символом `e` или `E`, за которым могут следовать знаки `+` или `-` и десятичный порядок. Символ `e` (`E`) означает десятичный порядок и имеет смысл «умножить на 10 в степени».

Например: `3.14E5` – 3,14 умножить на 10 в степени 5;

`-17e-2` – минус 17 умножить на 10 в степени минус 2.

Шестнадцатеричное число состоит из шестнадцатеричных цифр, которым предшествует знак доллара `$` (код символа 36). Диапазон шестнадцатеричных чисел – от `$FFFFFFFFFFFFFFFF` до `$7FFFFFFFFFFFFFFF`.

Логическая константа – это либо слово `False` (Ложь), либо слово `True` (Истина).

Символьная константа – это любой символ ПК, заключенный в апострофы

`'z'` – символ «z»;

`'Ф'` – символ «Ф».

Допускается использование записи символа путем указания его внутреннего кода, которому предшествует символ # (код 35), например:

`#97` – символ a;

`#90` – символ Z;

`#39` – символ '.

Строковая константа – любая последовательность символов (кроме символа CR – возврат каретки), заключенная в апострофы.

Например:

`'Ошибка!'`

Строку можно составлять из кодов нужных символов с предшествующими каждому коду символами #.

Например:

`#83#121#109#98#11#108` – symbol.

Также в строке можно чередовать части, записанные в обрамляющих апострофах, с частями, записанными кодами. Таким способом можно вставлять в строки любые управляющие символы, в том числе и символ CR.

Конструктор множества − список элементов множества, обрамленный квадратными скобками.

Например:

```
[1,2,4,7,12]
[blue, black]
```

В Delphi разрешается в объявлении констант использовать произвольные выражения, операндами которых могут быть ранее объявленные нетипизированные константы, имена типов и объектов, а также некоторые функции от них.

Например:

```
const
 lg=1/Ln10;
```

Метки − это имена операторов программы. Метки используются очень редко и только для того, чтобы программист смог указать компилятору, какой оператор программы должен выполняться следующим.

```
Label
 Loop;
```

Переменные связаны с изменяемыми областями памяти, т.е. с такими ее участками, содержимое которых будет меняться в ходе работы программы. В отличие от констант, переменные всегда объявляются в программе, этого после идентификатора переменной ставится двоеточие и имя типа, по образу которого должна строиться переменная.

```
var
 V: Integer;
 m: Byte;
```

Здесь идентификатор `V` объявляется как переменная типа `integer`, а идентификатор `m` − как переменная типа `Byte`.

Подпрограммы − это специальным образом оформленные фрагменты программы. Особенностью подпрограмм является их значительная независимость от остального текста программы. Таким образом, подпрограммы являются средством структурирования программ, т.е. расчленения программ на ряд во многом независимых фрагментов. Структурирование неизбежно для крупных программных

проектов, поэтому подпрограммы используются в Delphi-программах очень часто.

В Delphi есть два сорта подпрограмм: процедуры и функции. Функция отличается от процедуры только тем, что ее идентификатор можно наряду с константами и переменными использовать в выражениях, т.к. функция имеет выходной результат определенного типа.

Директивы компилятора в программе обозначаются в фигурных скобках: `{$}`. Это специальным образом написанный фрагмент кода, который указывает на необходимость подключения какого-либо файла, необходимость использования сопроцессора и т.д.

Например: `{$APPTYPE CONSOLE}`, Эта директива предназначена для компилятора. Следуя ее указаниям, компилятор генерирует исполняемую программу как консольное приложение.

4. ВЫРАЖЕНИЯ

Основными элементами, из которых конструируется исполняемая часть программы, являются константы, переменные и обращения к функциям. Каждый из этих элементов характеризуется своим значением и принадлежит к какому-либо типу данных. С помощью знаков операций и скобок из них можно составлять выражения, которые фактически представляют собой правила получения новых значений.

Алгебраические операции

Оператор	**Действие**
+	Сложение
-	Вычитание
*	Умножение
/	Деление
DIV	Деление нацело
MOD	Вычисление остатка от деления

Правила определения типа выражения

Оператор	**Тип операндов**	**Тип выражения**
*, +, -	Хотя бы один из операндов real	real
*, +, -	Оба операнда integer	integer
/	real или integer	Всегда real
DIV, MOD	Всегда integer	Всегда integer

Операции отношения

Операция	Действие	Тип операндов	Тип результата
=	Равно	Любой простой или строковый	Логический
<>	Не равно	То же	То же
<	Меньше	То же	Логический
<=	Меньше или равно	То же	То же
>	Больше	То же	То же
>=	Больше или равно	То же	То же

В Delphi определены следующие логические операции:

not – логическое НЕ;

and – логическое И;

or – логическое ИЛИ;

xor – исключительное ИЛИ.

Логические операции применимы к операндам целого и логического типов. Если операнды − целые числа, то результат логической операции есть то же целое число, биты которого (двоичные разряды) формируются из битов операндов по правилам, указанным в таблице:

Операнд 1	Операнд 2	not	and	or	xor
1		0			
0		1			
0	0		0	0	0
0	1		0	1	1
1	0		0	1	1
1	1		1	1	0

Логические операции над логическими данными дают результат логического типа по правилам:

Операнд 1	Операнд 2	not	and	or	xor
True		False			
False		True			
False	False		False	False	False
False	True		False	True	True
True	False		False	True	True
True	True		True	True	False

К логическим же в Delphi обычно относятся и две сдвиговые операции над целыми числами:

i shl j − сдвиг содержимого i на j разрядов влево; освободившиеся

младшие разряды заполняются нулями;

i shr j − сдвиг содержимого i на j разрядов вправо; освободившиеся

старшие разряды заполняются нулями.

В этих операциях i и j − выражения любого целого типа.

Математические функции

Функция	**Значение**
Abs (n)	Абсолютное значение n
Sqrt (n)	Квадратный корень из n
Sqr (n)	Квадрат n
Sin (n)	Синус n
Cos (n)	Косинус n
Arctan (n)	Арктангенс n

`Exp(n)`	Экспонента n
`Ln(n)`	Натуральный логарифм n
`Rardom(n)`	Случайное целое число в диапазоне от 0 до n- 1

При вычислении значений выражений следует учитывать, что операторы имеют разный приоритет. Приоритет операций убывает в следующем порядке:

унарные: `not`;

мультипликативные: `*, /, div, mod, and, shl, shr`;

аддитивные: `+, -, or, xor`;

отношения: `=, <>, <, >, <=, >=`.

Т.е. наивысшим приоритетом обладают унарные операции, низшим − операции отношения. Порядок выполнения нескольких операций равного приоритета устанавливается компилятором из условия оптимизации кода программы.

5. ВВОД-ВЫВОД В КОНСОЛЬНЫХ ПРИЛОЖЕНИЯХ

5.1. Операторы WRITE и WRITELN

Оператор write предназначен для вывода на экран монитора сообщений и значений переменных. После слова write в скобках задается список имен переменных. Кроме имен переменных в список можно включить сообщение – текст, заключенный в апострофы.

Примеры:

```
Write (Summa);
Write ('Результат вычислений');
Write ('Корни уравнения. x1=',x1,' x2=',x2);
```

После имени переменной через двоеточие можно поместить описание (формат) поля вывода значения переменной.

Для переменной типа integer формат – это целое число, определяющее ширину поля вывода (количество позиций на экране). Например, write (d:5) показывает, что для вывода значения переменной d используется 5 позиций. Если число занимает меньше позиций, чем указано в формате, то неиспользуемые позиции заполняются пробелами, а само изображение выравнивается по правой границе поля.

Для переменных типа real формат представляет собой два целых числа, разделенных двоеточием. Первое число определяет ширину поля вывода, второе – число цифр, стоящих справа от десятичной точки. Если задать только ширину поля, то на экране появится число, представленное в формате с плавающей точкой.

Если ширины поля, указанной в формате, недостаточно для вывода значения переменной, то выводится число в формате с плавающей точкой и десятью цифрами после запятой (все поле вывода в этом случае занимает 17 позиций).

После выполнения оператора write курсор остается в той позиции экрана, в которую он переместился после вывода последнего символа.

Оператор writeln отличается от write только тем, что после вывода сообщения или значений переменных курсор переходит в начало следующей строки.

5.2. Операторы READ и READLN

Оператор read предназначен для ввода с клавиатуры значений переменных (исходных данных).

```
read(Переменная1, Переменная2, ... ПеременнаяN);
```

где Переменнаяi – имя переменной, значение которой должно быть введено с клавиатуры во время выполнения программы.

Примеры:

```
read(a);
read(Cena, Kol);
```

При выполнении оператора read происходит следующее: программа приостанавливает свою работу и ждет, пока на клавиатуре будут набраны нужные данные и нажата клавиша <Enter>.

Один оператор read позволяет получить значения нескольких переменных. При этом вводимые числа должны быть набраны в одной строке и разделены пробелами.

Оператор readln отличается от read тем, что после ввода значений переменных курсор должен перейти к началу следующей строки.

Чтобы «подсказать» пользователю, какие данные ожидает от него программа, перед каждым оператором read или readln следует располагать инструкцию write, например, фрагмент программы вычисления стоимости покупки может быть таким:

```
Writelnt ('Введите исходные данные.');
Write ('Цена изделия:');
Readin (Cena);
Write ('Количество в партии:');
Readln (Kol);
Write ('Скидка:');
Readln (Skidka);
```

Если тип данных, вводимых с клавиатуры, не соответствует или не может быть приведен к типу переменных, имена которых указаны в процедуре read (readln), то программа аварийно завершает работу и на экран выводится сообщение об ошибке.

6. УПРАВЛЯЮЩИЕ СТРУКТУРЫ ЯЗЫКА OBJECT PASCAL

6.1. Оператор присваивания

В результате присваивания переменная получает значение.

Имя:= Выражение,

где **Имя** – имя переменной, значение которой изменяется в результате выполнения присваивания.

Примеры:

```
Counter:=0;
D:=b*b-4*a*c;
```

6.2. Составной оператор и пустой оператор

Составной оператор − это последовательность произвольных операторов программы, заключенная в операторные скобки − зарезервированные слова `begin...end`.

Delphi не накладывает никаких ограничений на характер операторов, входящих в составной оператор. Среди них могут быть и другие составные операторы − язык Delphi допускает произвольную глубину их вложенности.

Весь раздел операторов, обрамленный словами `begin ... end`, представляет собой один составной оператор. Поскольку зарезервированное `end` является закрывающей операторной скобкой, оно одновременно указывает и конец предыдущего оператора, поэтому ставить перед ним символ `;` не обязательно. Наличие точки с запятой перед `end` означает, что между последним оператором и операторной скобкой `end` располагается ***пустой оператор.*** В основном пустой оператор используется для передачи управления в конец составного оператора.

6.3. Условный оператор

Условный оператор позволяет проверить некоторое условие и в зависимости от результатов проверки выполнить то или иное действие. Таким образом, условный оператор − это средство ветвления вычислительного процесса.

Структура условного оператора имеет следующий вид:

```
if <условие>
 then <оператор1>
```

else <оператор2>;

где **if, then, else** – зарезервированные слова (если, то, иначе);

<условие> – произвольное выражение логического типа;

<оператор1>, <оператор2> – любые операторы языка Delphi.

Условный оператор работает по следующему алгоритму. Вначале вычисляется условное выражение <условие>. Если результат есть True (Истина), то выполняется <оператор1>, а <оператор2> пропускается; если результат есть False (Ложь), наоборот, <оператор1> пропускается, а выполняется <оператор2>.

Условными называются выражения, имеющие одно из двух возможных значений: истина или ложь. Такие выражения чаще всего получаются при сравнении переменных с помощью операций отношения =, >, >=, <, <=.

Сложные логические выражения составляются с использованием логических операций.

Например:

```
if (a > b) and (b <> 0)
 then ...
```

В отличие от большинства других языков программирования, в Delphi приоритет операций отношения меньше, чем у логических операций, поэтому отдельные составные части сложного логического выражения заключаются в скобки.

Например, такая запись предыдущего оператора будет неверной:

```
if a > b and b <> 0
 then ...
```

Часть else <оператор2> условного оператора может быть опущена. Тогда при значении True условного выражения выполняется <оператор1>, в противном случае этот оператор пропускается.

Пример. Решение квадратного уравнения.

```
program kvadrUr;
uses
 SysUtils;
Var
 a,b,c: real; //Коэффициенты уравнения
```

```
      x1,x2: real; //Корни уравнения
      d: real; //Дискриминант
    begin
      Writeln('Введите значение коэффициентов');
      Readln(a,b,c); {вводим значение коэффициен-
тов}
      d:=b*b-4*a*c; {вычисляем дискриминант}
      if d>=0 {если значение дискриминанта больше
нуля}
      then
      begin
      x1:=-b+sqrt(d)/(2*a);
      {то вычисляем корни уравнения}
      x2:=-b-sqrt(d)/(2*a);
      Writeln('x1=',x1:4:2,' x2=', x2:4:2); {и вы-
водим их}
      end
      else
      Writeln(' Корней нет'); {иначе выдаем сооб-
щение}
      end.
```

6.4. Операторы повторений

В языке Delphi имеются три различных оператора, с помощью которых можно запрограммировать повторяющиеся фрагменты программ.

6.4.1. Счетный оператор цикла FOR

Счетный оператор цикла **FOR** имеет такую структуру:

for <параметр_цикла>:= <нач_знач> **to** <кон_знач> **do** <оператор>;

Здесь

for, to, do – зарезервированные слова (для, до, выполнить);

<параметр_цикла> – переменная типа Integer (точнее, любого порядкового типа);

<нач_знач> – начальное значение – выражение того же типа;

<кон_знач> – конечное значение – выражение того же типа;

<оператор> – произвольный оператор Delphi.

При выполнении оператора **for** вначале вычисляется выражение <нач_знач> и осуществляется присваивание:

<параметр_цикла>:= <нач_знач>.

После этого циклически повторяется:

проверка условия <параметр_цикла> <= <кон_знач>; если условие не выполнено, оператор **for** завершает свою работу;

выполнение оператора <оператор>;

наращивание переменной <параметр_цикла> на единицу.

Условие, управляющее работой оператора **for**, проверяется перед выполнением оператора <оператор>: если условие не выполняется в самом начале работы оператора **for**, исполняемый оператор не будет выполнен ни разу. Шаг наращивания параметра цикла строго постоянен и равен (+1). Существует другая форма оператора:

for <пар_ цик> := <нач_знач> **downto** <кон_знач> do <оператор>;

Замена зарезервированного слова **to** на **downto** означает, что шаг наращивания параметра цикла равен (-1), а управляющее условие приобретает вид <параметр_цикла> >= <кон_знач>.

Пример. Программа осуществляет ввод произвольного целого числа и вычисление суммы всех чисел от 0 до N.

```
program SummPol_Otr;
uses
 SysUtils;
var
 Sum,i,N: Integer;
begin
 Writeln('Введите N');
 Readln(N);
 Sum:=0; // Начальное значение Sum
 if N>=0 // Если число положительное
 then
 //Цикл формирования суммы положительных чи-
сел
 for i:=1 to N do
 Sum:=Sum+i
 else
 //Цикл формирования суммы отрицательных чи-
сел
 for i:=-1 downto N do
 Sum:=Sum+i;
 Writeln (Sum);
end.
```

6.4.2. Оператор цикла WHILE с предпроверкой условия

While <условие> **do** <оператор>;

Здесь **while, do** – зарезервированные слова (пока [выполняется условие], делать);

<условие> – выражение логического типа;

<оператор> – произвольный оператор Delphi.

Если выражение <условие> имеет значение True, то выполняется <оператор>, после чего вычисление выражения <условие> и его проверка повторяются. Если <условие> имеет значение False, оператор while прекращает свою работу.

Пример: Вычислить значение числа π с задаваемой с клавиатуры точностью.

Вычисление значения числа π основано на том, что сумма ряда

1-1/3+1/5-1/7+1/9+… приближается к значению π/4 при достаточно большом количестве членов ряда. Каждый член ряда вычисляется по формуле 1/(2*n-1) и домножается на –1, если n-четное. Вычисление заканчивается тогда, когда значение очередного члена ряда становится меньше, чем заданная точность вычислений.

```
program pi;
uses
 SysUtils;
var
 p,t,elem: real;
 n: integer;
begin
 p:=0; {Вычисляемое значение pi}
 n:=1; {номер члена ряда}
 elem:=1; {начальное значение}
 Writeln ('Введите точность вычислений');
 Readln(t);
 While elem>=t do
 begin
 elem:=1/(2*n-1);
 if (n mod 2)=0
 then
 p:=p-elem
 else
 p:=p+elem;
 n:=n+1;
 end;
 p:=p*4;
```

```
      Writeln('  Значение  pi  с  точностью  ',
t:16:10, ' равно ', p:9:5);
      end.
```

6.4.3. Оператор цикла `REPEAT ... UNTIL` *с постпроверкой условия:*

repeat <тело_цикла> **until** <условие>;

Здесь **repeat, until** – зарезервированные слова (повторять [до тех пор], пока [не будет выполнено условие]);

<тело_цикла> – произвольная последовательность операторов Delphi;

<условие> – выражение логического типа.

Операторы <тело_цикла> выполняются хотя бы один раз, после чего вычисляется выражение <условие>: если его значение есть False, операторы

<тело_цикла> повторяются, в противном случае оператор **repeat...until** завершает свою работу.

Для правильного выхода из цикла условие выхода должно меняться внутри операторов, составляющих тело цикла **while** или **repeat...until.**

Для гибкого управления циклическими операторами for, while и repeat в состав Delphi включены две процедуры без параметров:

BREAK – реализует немедленный выход из цикла; действие процедуры заключается в передаче управления оператору, стоящему сразу за концом циклического оператора;

CONTINUE – обеспечивает досрочное завершение очередного прохода цикла; эквивалент передачи управления в самый конец циклического оператора.

Пример. Выяснить, является ли введенное с клавиатуры целое число простым. Число называется простым, если оно делится только на единицу и на себя.

Проверить является ли число n простым, можно делением числа на 2,3 и т.д. до n и проверкой остатка после каждого деления. Если остаток равен нулю, то следовательно найдено число, на которое n делится без остатка. Сравнив n и полученное т.о. число, можно определить, является ли n простым.

```
program Prostoe_chislo;
uses
 SysUtils;
```

```
var
 n,d,r:integer;
begin
 Writeln (' Введите целое число');
 Readln(n);
 d:=2;
 repeat
 r:=n mod d;
 if r<>0
 then d:=d+1;
 until r=0;
 if d=n
 then
 Writeln (' Простое число')
 else
 Writeln (' Не простое число');
 end.
```

6.5. Оператор выбора

Оператор выбора позволяет выбрать одно из нескольких возможных продолжений программы. Параметром, по которому осуществляется выбор, служит ***ключ выбора*** – выражение любого порядкового типа.

Структура оператора выбора такова:

case <ключ_выбора> **of** <список_выбора> [else <операторы>] end;

Здесь **case, of, else, end** – зарезервированные слова (случай, из, иначе, конец);

<ключ_выбора> – ключ выбора (выражение порядкового типа);

<список_выбора> – одна или более конструкций вида:

<константа_выбора>:<оператор>;

где:

<константа_выбора> – константа того же типа, что и выражение <ключ_выбора>;

<оператор> – произвольный оператор Delphi.

Оператор выбора работает следующим образом. Вначале вычисляется значение выражения <ключ_выбора>, а затем в последовательности операторов <список_выбора> отыскивается такой, которому предшествует константа, равная вычисленному значению. Найденный оператор выполняется, после чего оператор выбора завершает свою работу. Если в списке выбора не будет найдена константа, соответствующая вычисленному значению ключа выбора,

управление передается операторам, стоящим за словом **else**. Часть **else** <операторы> можно опускать. Тогда при отсутствии в списке выбора нужной константы ничего не произойдет, и оператор выбора просто завершит свою работу.

Пример: Программа дописывает после числа слово “рубль” в правильной форме. Окончание поясняющего слова определяется последней цифрой числа.

Последняя цифра числа	Поясняющий текст
0,5,6,7,8,9	рублей
1	рубль
2,3,4	рубля

Исключение составляют числа, оканчивающиеся на 11,12,13,14.

```
program Rubl;
uses
 SysUtils;
var
 n:integer;
 r:integer;
begin
 Writeln('Введите количество рублей');
 Readln(n);
 if n>100
 then
 n:=n mod 100;
 if (n>=11)and (n<=14)
 then
 Writeln('рублей')
 else
 begin
 r:=n mod 10;
 case r of
 0,5..9: Writeln ('рублей');
 1: Writeln ('рубль');
 2..4: Writeln ('рубля');
 end;
 end;
 end.
```

6.6. Метки и операторы перехода

Рассмотренных операторов вполне достаточно для написания программ любой сложности. Современная технология структурного программирования основана на принципе «программировать без goto»: считается, что злоупотребление операторами перехода затрудняет понимание программы, делает ее запутанной и сложной в отладке.

Тем не менее, в некоторых случаях использование операторов перехода может упростить программу.

Оператор перехода имеет вид:

goto <метка>;

Здесь **goto** − зарезервированное слово (перейти [на метку]);

<метка> − метка.

Метка в Delphi − это произвольный идентификатор, позволяющий именовать некоторый оператор программы и таким образом ссылаться на него. Метка располагается непосредственно перед помечаемым оператором и отделяется от него двоеточием. Перед тем как появиться в программе, метка должна быть описана.

Описание меток состоит из зарезервированного слова **label** (метка), за которым список меток:

```
label
 loop, lb1, lb2;
begin
 …
 goto lb1;
 loop:
 …
 goto lb2;
end;
```

Действие оператора **goto** состоит в передаче управления соответствующему меченному оператору.

При использовании меток необходимо руководствоваться следующими правилами:

− метка, на которую ссылается оператор **goto**, должна быть описана в разделе описаний и она обязательно должна встретиться где-нибудь в теле программы;

− метки, описанные в подпрограмме, локализуются в ней, поэтому передача управления извне подпрограммы на метку внутри нее невозможна.

7. МАССИВЫ

7.1. Одномерные массивы

Необходимость в массивах возникает всегда, когда при решении задач приходится иметь дело с большим, но конечным количеством однотипных данных. Массив − это просто последовательность переменных одного типа.

Например, массив целых чисел будет выглядеть так: 15, 23, 36, 41.

Массив, как и любая переменная программы, перед использованием должен быть объявлен в разделе объявления переменных. В общем виде инструкция объявления массива выглядит следующим образом:

```
Имя: array [нижний_индекс..верхний_индекс] of тип
```

где:

имя − имя массива;

array − зарезервированное слово языка Delphi, обозначающее, что объявляемое имя является именем массива;

нижний_индекс и верхний_индекс − целые константы, определяющие диапазон изменения индекса элементов массива и, неявно, количество элементов (размер) массива;

тип − тип элементов массива.

Как определяется длина массива? Очень просто, это даже похоже на геометрическое определение. Например, пусть нужен массив из 12 значений. Длина такого массива может быть [0..11] или [1..12]. В квадратных скобках вы должны поставить начальное и конечное значение массива, а между ними две точки. Тип данных может быть любой.

Примеры объявления массивов:

```
temper: array [1..31] of real;
koef: array [0. .2] of integer;
name: array [1..30] of char;
```

Рассмотрим пример объявления массива из 100 целых чисел.

```
var
b:array [0..99] of Integer;
begin
 b[0]:=1;
 b[1]:=2;
end;
```

В этом же примере показано, как осуществить доступ к элементам массива. Чтобы получить доступ к какому-то элементу, нужно написать имя переменной массива и после этого в квадратных скобках написать номер элемента, к которому нужно получить доступ.

При объявлении массива удобно использовать именованные константы.

```
const
 NT = 18;
 team: array [0..NT] of char;
```

Типичными операциями при работе с массивами являются:

- ввод массива;
- вывод массива;
- поиск максимального или минимального элемента массива;
- поиск заданного элемента массива;
- сортировка массива.

Ввод массива. Под вводом массива понимается ввод значений элементов массива. Ввод удобно реализовать при помощи оператора повторения **for.** Чтобы пользователь программы знал, ввод какого элемента массива ожидает программа, следует организовать вывод подсказок перед вводом очередного элемента массива. В подсказке обычно указывают индекс элемента массива.

Вывод массива. Под выводом массива понимается вывод на экран значений элементов массива. Если в программе необходимо вывести значения всех элементов массива, то для этого удобно использовать `for`, переменная-счетчик которой может быть использована как индекс элемента массива.

```
program Vvod_vivod;
{Программа формирует и выводит массив по следующему правилу: новые элементы равны исходным элементам, к которым добавляется сумма элементов массива. Исходный массив является целочисленным и вводится с клавиатуры.}
uses
 SysUtils;
const
 N=10; // Количество элементов массива
var
 m: array [1..N] of Integer; // Массив чисел
 i: Integer; // Индекс массива
 sum: Integer; // Сумма элементов массива
```

```
begin
 for i := 1 to N do // Цикл ввода массива
 begin
 Write(i,' элемент', '->');
 Readln (m[i]);
 end;
 sum := m[1];
 // Задаем начальное значение переменной
 for i := 2 to N do
 //Цикл вычисления суммы всех элементов мас-
сива
 sum := sum + m[i];
 for i := 1 to N do
 //Цикл вычисления новых элементов массива и
 //вывода массива
 begin
 m[i]:=m[i]+sum;
 Writeln (i, '-', m[i]);
 end;
 Readln
end.
```

Поиск минимального (максимального) элемента массива. Задачу поиска минимального элемента массива рассмотрим на примере массива целых чисел.

Алгоритм поиска минимального (максимального) элемента массива довольно очевиден: сначала делается предположение, что первый элемент массива является минимальным (максимальным), затем остальные элементы массива последовательно сравниваются с этим элементом. Если во время очередной проверки обнаруживается, что проверяемый элемент меньше (больше) принятого за минимальный (максимальный), то этот элемент становится минимальным (максимальным) и продолжается проверка оставшихся элементов.

```
program Max_min;
 {Программа создает массив из N случайных це-
лых чисел, равномерно распределенных в диапазоне
от 0 до MAX_VALUE-1, подсчитывает минимальное и
максимальное из них.}
uses
 SysUtils;
const
 N = 10; // Количество элементов массива
```

```
      MAX_VALUE = 100+1; {Диапазон значений слу-
чайных чисел}
     var
      m: array [1..N] of Integer; // Массив чисел
      i: Integer;          // Индекс массива
      max, min: Integer; {Максимальное и минималь-
ное число}
     begin
      for i := 1 to N do
      begin
      m[i]:= Random(MAX_VALUE); {Наполняем массив
случайными числами}
      Writeln (i, '-', m[i]); {вывод массива}
      end;
      max := m[1];{Задаем начальное значение мак-
симального значения}
      min := m[1];{Задаем начальное значение мини-
мального значения}
      for i := 2 to N do {Цикл поиска минимального
и максимального}
      if m[i] < min
      then
      min := m[i]
      else
      if m[i] > max
      then
      max := m[i];
      Writeln('max= ', max);
      Writeln('min= ', min);
      Readln;
     end.
```

Поиск заданного элемента массива. При решении многих задач возникает необходимость определить, содержит ли массив определенную информацию или нет. Например, проверить, есть ли в списке студентов фамилия Петров. Задачи такого типа называются поиском в массиве.

Для организации поиска в массиве могут быть использованы различные алгоритмы. Наиболее простой – это алгоритм простого перебора. Поиск осуществляется последовательным сравнением элементов массива с образцом до тех пор, пока не будет найден элемент, равный образцу, или не будут проверены все элементы. Алгоритм простого перебора применяется, если элементы массива не упорядочены.

Ниже приведен текст программы поиска в массиве целых чисел. Перебор элементов массива осуществляется инструкцией `repeat`, в теле которой инструкция `if` сравнивает текущий элемент массива с образцом и присваивает переменной `found` значение `true`, если текущий элемент и образец равны.

Цикл завершается, если в массиве обнаружен элемент, равный образцу (в этом случае значение переменной `found` равно `true`), или если проверены все элементы массива. По завершении цикла по значению переменной `found` можно определить, успешен поиск или нет.

```
program Poisk_v_massive;
uses
 SysUtils;
const
 SIZE=5;
var
 a: array[1..SIZE] of integer; //массив
 obr: integer; // образец для поиска
 found: boolean; {TRUE — совпадение образца с
элементом массива}
 i: integer; // индекс элемента массива
begin
 Writeln('Введите массив');
 for i:=1 to SIZE do // ввод массива
 begin
 Write(i,' элемент', '->');
 Readln (a[i]);
 end;
 Writeln('Введите образец для поиска');
 Readln (obr); // ввод образца для поиска
 // поиск
 found := FALSE; {пусть нужного элемента в
массиве нет}
 i:= 1; {Проверяем с первого элемента масси-
ва}
 repeat
 if a[i] = obr
 then found := TRUE {Совпадение с образцом}
 else i := i+1; {Переход к следующему элемен-
ту}
 until (i > SIZE) or (found = TRUE); {Завер-
шаем, если совпадение с образцом или проверен по-
следний элемент массива}
```

```
      if found
      then
      Writeln ('Совпадение с элементом номер', i,
#10, #13 , 'Поиск успешен')
      else
      Writeln (' Совпадений нет');
      Readln;
     end.
```

7.2. Двумерные массивы

В повседневной жизни довольно часто приходится иметь дело с информацией, которая представлена в табличной форме. Например, результат деятельности некоторой фирмы, торгующей автомобилями, может быть представлен в виде таблицы.

	Январь	Февраль	Март	...	Ноябрь	Декабрь
ваз2106						
ваз2107						
ваз2108						
ваз2109						
ваз2110						
ваз2111						

Колонки и (или) строки таблицы, как правило, состоят из однородной информации. Поэтому в программе, обрабатывающей табличные данные, имеет смысл использовать массивы для хранения и обработки таблиц. Так, приведенная выше таблица может быть представлена как совокупность одномерных массивов:

```
vaz2106: array [1..12] of integer;
vaz2107: array [1..12] of integer;
vaz2108: array [1..12] of integer;
vaz2109: array [1..12] of integer;
vaz2110: array [1..12] of integer;
vaz2111: array [1..12] of integer;
```

Каждый из приведенных массивов может хранить информацию о количестве проданных автомобилей одной марки, причем значение элемента массива отражает количество проданных машин в соответствующем месяце.

Возможно и такое представление таблицы:

```
jan: array [1..6] of integer;
feb: array [1..6] of integer;
mar: array [1..6] of integer;
dec: array [1..6] of integer;
```

В этом случае каждый массив предназначен для хранения информации о количестве проданных за месяц автомобилей, причем значение элемента массива отражает проданное количество автомобилей одной марки.

Если вся таблица содержит однородную информацию, то такая таблица может быть представлена как двумерный массив.

В общем виде инструкция объявления двумерного массива выглядит так:

```
Имя: array [ НГ1..ВГ1,НГ2..ВГ2] of Тип;
```

где:

Имя – имя массива;

array – ключевое слово языка Delphi, указывающее, что объявляемый элемент данных является массивом;

НГ1, ВГ1, НГ2, ВГ2 – целые константы, определяющие диапазон изменения индексов и, следовательно, число элементов массива;

Тип – тип элементов массива.

Таблица может быть представлена в виде двумерного массива следующим образом:

```
itog: array [1..12, 1..6] of integer
```

Для того чтобы использовать элемент массива, нужно указать имя массива и индексы элемента. Первый индекс обычно соответствует номеру строки таблицы, второй – номеру колонки. Так, элемент itog [2,3] содержит число проданных в марте (третий месяц) автомобилей марки ваз2107.

При работе с таблицами (массивами) удобно использовать инструкцию for. Следующий фрагмент программы вычисляет сумму элементов массива (общее количество автомобилей, проданных за год).

```
s:=0;
for i := 1 to 6 do // шесть моделей автомобилей
 for j := 1 to 12 do //12 месяцев
 s := s + itog[i,j];
```

8. МНОЖЕСТВА

Множества − это наборы логически связанных друг с другом объектов. Количество элементов, входящих в множество, может меняться в пределах от 0 до 256 (множество, не содержащее элементов, называется *пустым).*

Описание типа множества имеет вид:

<имя типа> = **set of** <базовый тип>;

Здесь <имя типа> − правильный идентификатор;

set, of − зарезервированные слова *(множество, из)*

<базовый тип> − базовый тип элементов множества.

Пример определения и задания множеств:

```
type
 digitChar = set of '0'..'9';
 digit = set of 0..9;
var
 s1. s2. s3: digitChar;
 s4. s5. s6: digit;
 begin
 …
 s1:= ['1', '2', '3'];
 s2:= ['3', '2', '1'];
 s3:= ['2', '3'];
 s4:= [0..3 , 6];
 s5:= [4, 5];
 s6:= [3..9];
 …
end.
```

В этом примере множества s1 и s2 эквивалентны, а множество S3 включено в S2, но не эквивалентно ему.

Над множествами определены следующие операции:

[ ] –пустое множество;

***** − пересечение множеств; результат содержит элементы, общие для обоих множеств;

+ − объединение множеств; результат содержит элементы первого множества, дополненные недостающими элементами из второго множества;

- — разность множеств; результат содержит элементы из первого множества, которые не принадлежат второму:

= — проверка эквивалентности; возвращает `True`, если оба множества эквивалентны;

<> — проверка неэквивалентности; возвращает `True`, если оба множества неэквивалентны;

<= — проверка вхождения; возвращает `True`, если первое множество включено во второе;

>= — проверка вхождения; возвращает `True`, если второе множество включено в первое;

IN — проверка принадлежности; в этой бинарной операции первый элемент — выражение, а второй — множество одного и того же типа; возвращает `True`, если выражение имеет значение, принадлежащее множеству.

9. ЗАПИСИ

В практике программирования довольно часто приходится иметь дело с данными, которые естественным образом состоят из других данных. Например, сведения об учащемся содержат фамилию, имя, отчество, число, месяц и год рождения, домашний адрес и другие данные. Для представления подобной информации в языке используется структура, которая носит название запись (`record`).

С одной стороны, запись можно рассматривать как единую структуру, а с другой − как набор отдельных элементов, компонентов. Характерной особенностью записи является то, что составляющие ее компоненты могут быть разного типа.

Запись − это структура данных, состоящая из отдельных именованных компонентов разного типа, называемых полями.

Объявление записи.

В общем виде объявление типа «запись» выглядит так:

```
Имя = record
 Поле_1 : Тип_1;
 Поле_2 : Тип_2;
 Поле_К : Тип_К;
end;
```

где:

`Имя` − имя типа «запись»;

`record` − зарезервированное слово языка Delphi, означающее, что далее следует объявление компонентов (полей) записи;

`поле_i` и `тип_i` − имя и тип i-го компонента (поля) записи, где i=1, ..., k;

`end` − зарезервированное слово языка Delphi, означающее, что список полей закончен.

Пример объявлений:

```
type
TPerson = record
f_name: string[20];
l_name: string[20];
day: integer;
month: integer;
year: integer;
address: string[50];
end;
```

После объявления типа записи можно объявить переменную-запись:

```
var
 student : TPerson;
```

Для того чтобы получить доступ к элементу (полю) переменной-записи (записи), нужно указать имя записи и имя поля, разделив их точкой. **Например**, инструкции

```
Writeln('Имя: ', student.f_name);
Writeln('Адрес: ', student.address);
```

выводят на экран содержимое полей `f_name` (имя) и `address` (адрес) переменной-записи `student`.

Чтобы упростить доступ к полям записи, используется оператор присоединения: **`with`**.

Инструкция `with` позволяет использовать в тексте программы имена полей без указания имени переменной-записи. В общем виде инструкция `with` выглядит следующим образом:

```
 with Имя do
 begin
 {операторы программы }
 end;
```

где:

`имя` – имя переменной-записи;

`with, do` – зарезервированные слова языка Delphi *(с, делать),* означающее, что далее, до слова `end`, при обращении к полям записи “Имя” имя записи можно не указывать.

Например, если в программе объявлена запись

```
student: record // информация о студенте
f_name: string[30]; // фамилия
l_name: string[20]; // имя
address: string[50]; // адрес
end;
```

то вместо инструкций

```
student.f_name := 'Петров';
student.l_name := 'Петр';
student.address := 'Минск';
```

можно записать:

```
with student do
 begin
 f_name := 'Петров';
 f_name := 'Петр';
 address := 'Минск';
 end;
```

10. ФАЙЛЫ

10.1. Доступ к файлам

Любой файл имеет три характерные особенности. Во-первых, у него есть имя, что дает возможность программе работать одновременно с несколькими файлами. Во-вторых, он содержит компоненты одного типа. Типом компонентов может быть любой тип Delphi, кроме файлов. Иными словами нельзя создать «файл файлов». В-третьих, длина вновь создаваемого файла никак не оговаривается при его объявлении и ограничивается только емкостью устройств внешней памяти.

Файловый тип можно задать одним из трех способов:

```
<имя> = File of <тип>;
<имя> = TextFile;
 <имя> = File;
```

Здесь <имя> – имя файлового типа (правильный идентификатор);

File, of – зарезервированные слова *(файл, из);*

TextFile – имя стандартного типа текстовых файлов;

<тип> – любой тип Delphi, кроме файлов.

В зависимости от способа объявления можно выделить три вида файлов:

типизированные файлы (задаются предложением File of...);

текстовые файлы (определяются типом TextFile);

нетипизированные файлы (определяются типом File).

Примеры:

```
res: file of char; // файл символов
koef: file of real; // файл вещественных чи-
сел
f: TextFile;
f3: file;
```

Имя файла задается вызовом процедуры AssignFile, связывающей файловую переменную с конкретным файлом.

Описание процедуры AssignFile выглядит следующим образом:

```
AssignFile (<ф.п.>, <имя файла>);
```

Здесь <ф.п.> – файловая переменная (правильный идентификатор, объявленный в программе как переменная файлового типа);

<имя файла> – текстовое выражение, содержащее имя файла и, если это необходимо, маршрут доступа к нему, т.е. состоять не только непосредственно из имени файла, но и включать путь к файлу (имя диска, каталогов и подкаталогов).

Инициировать файл означает указать для этого файла направление передачи данных. В Delphi можно открыть файл для чтения, для записи информации, а также для чтения и записи одновременно. Для чтения файл инициируется с помощью стандартной процедуры Reset:

```
Reset(<ф.п.>);
```

Здесь <ф.п. > – файловая переменная, связанная ранее процедурой AssignFile с уже существующим файлом. При выполнении этой процедуры дисковый файл подготавливается к чтению информации. В результате специальная переменная-указатель, связанная с этим файлом, будет указывать на начало файла, т.е. на компонент порядковым номером 0.

Стандартная процедура

```
Rewrite(<ф.п.>);
```

инициирует запись информации в файл, связанный с файловой переменной <ф.п.>. Процедурой Rewrite нельзя инициировать запись информации в ранее существовавший дисковый файл: при выполнении этой процедуры старый файл (если он был) уничтожается и никаких сообщений об этом в программу не передается. Новый файл подготавливается к приему информации, и его указатель принимает значение 0.

Стандартная процедура

```
Append(<ф.п.>);
```

инициирует запись в ранее существовавший текстовый файл для его расширения, при этом указатель файла устанавливается в его конец. Процедура Append применима только к текстовым файлам, т.е. их файловая переменная должна иметь тип TextFile.

10.2. Подпрограммы для работы с файлами

Procedure CloseFile(var F); Закрывает файл, однако связь файловой переменной F с именем файла, установленная ранее процедурой AssignFile, сохраняется.

Function EOF(**var** F):Boolean; Тестирует конец файла и возвращает True, если файловый указатель стоит в конце файла. При записи это означает, что очередной компонент будет добавлен в конец файла, при чтении – что файл исчерпан

Procedure Erase(var F); Уничтожает файл F. Перед выполнением процедуры необходимо закрыть файл.

Function FileExists (const FileName: String): Boolean; Возвращает True, если файл с именем FileName существует.

Procedure Rename (var F; NewName: String); Переименовывает файл F; NewName – строковое выражение, содержащее новое имя файла. Перед выполнением процедуры необходимо закрыть файл.

10.3. Текстовые файлы

Текстовые файлы предназначены для хранения текстовой информации. Текстовый файл трактуется в Delphi как совокупность строк переменной длины. Доступ к каждой строке возможен лишь последовательно, начиная с первой. При создании текстового файла в конце каждой строки ставится специальный признак EOLN (End Of LiNe – конец строки), а в конце всего файла – признак EOF (End Of File – конец файла). Эти признаки можно протестировать одноименными логическими функциями. При формировании текстовых файлов используются следующие системные соглашения:

EOLN – последовательность кодов #13 и #10;

EOF – код #26.

Для доступа к записям применяются процедуры Read, ReadLn, Write, WriteLn. Они отличаются возможностью обращения к ним с переменным количеством фактических параметров, в качестве которых могут использоваться символы, строки и числа. Первым параметром в любой из перечисленных процедур должна стоять файловая переменная. Обращение осуществляется к дисковому файлу, связанному с переменной процедурой AssignFile.

Подпрограммы для работы с текстовыми файлами

ПОДПРОГРАММЫ	ВЫПОЛНЯЕМОЕ ДЕЙСТВИЕ
Function Eoln(var F:TextFile): Boolean;	Тестирует маркер конца строки и возвращает True, если конец строки достигнут
Procedure Read(var F: TextFile; V1[,V2,..., Vn]);	Читает из текстового файла последовательность символьных представлений переменных Vi типа Char, String, а также любого целого или вещественного типа, игнорируя признаки EOLN
Procedure ReadLn(var F: TextFile;	Читает из текстового файла последовательность символьных представле-

ПОДПРОГРАММЫ	ВЫПОЛНЯЕМОЕ ДЕЙСТВИЕ
`[V1[,V2,..., Vn]]);`	ний переменных `Vi` типа `Char`, `String`, а также любого целого или вещественного типа с учетом границ строк
`Function` `SeekEof (var F : TextFile): Boolean;`	Пропускает все пробелы, знаки табуляции и маркеры конца строки EOLN до маркера конца файла EOF или до первого значащего символа и возвращает `True`, если маркер EOF обнаружен
`Function` `SeekEoln (var F: TextFile): Boolean;`	Пропускает все пробелы и знаки табуляции до маркера конца строки EOLN или до первого значащего символа и возвращает `True`, если маркер обнаружен
`Procedure` `Write( var F: TextFile; P1 [, P2,..., Pn]);`	Записывает символьные представления параметров `Pi` в текстовый файл
`Procedure` `WriteLn (var F: TextFile ; [P1 [, P2,..., Pn]]);`	Записывает символьные представления параметров `Pi` и признак конца строки EOLN в текстовый файл

10.4. Типизированные файлы

Длина любого компонента типизированного файла строго постоянна, что дает возможность организовать прямой доступ к каждому из них (т.е. доступ к компоненту по его порядковому номеру).

Перед первым обращением к процедурам ввода/вывода указательфайла стоит в его начале и указывает на первый компонент с номером 0. После каждого чтения или записи указатель сдвигается к следующему компоненту файла. Переменные в списках ввода/вывода должны иметь тот же тип, что и компоненты файла. Если этих переменных в списке несколько, указатель будет смещаться после каждой операции обмена данными между переменными и дисковым файлом.

Подпрограммы для работы с типизированными файлами

ПОДПРОГРАММЫ	ВЫПОЛНЯЕМОЕ ДЕЙСТВИЕ
Function FilePos(var F): LongInt;	Возвращает текущую позицию в файле, т.е. номер компонента, который будет обрабатываться следующей операцией ввода/вывода
Function FileSize (var F):LongInt;	Возвращает количество компонентов файла. Чтобы переместить указатель в конец типизированного можно написать: seek (FileVar, FileSize(FileVar));
Procedure Seek(var F; N: LongInt);	Смещает указатель файла F к требуемому компоненту: N – номер компонента файла (первый компонент файла имеет номер 0)
Procedure Read **(var** F, V1, ...,Vn**);**	Читает данные из типизированного файла F:Vi – переменные такого же типа, что и компоненты файла
ProcedureWrite **(var** F,P1, ...,Pn**);**	Записывает данные в типизированный файл F: Pi– выражения такого же типа, что и компоненты файла

10.5. Нетипизированные файлы

Нетипизированные файлы объявляются как файловые переменные типа File и отличаются тем, что для них не указан тип компонентов. Отсутствие типа делает эти файлы, с одной стороны, совместимыми с любыми другими файлами, а с другой − позволяет организовать высокоскоростной обмен данными между диском и памятью.

При инициации нетипизированные файла процедурами Reset или Rewrite можно указать длину записи нетипизированного файла в байтах:

```
var
 F: File;
begin
 …
 AssignFile(F,'myfile.dat');
 Reset(f,512);
 …
end.
```

При работе с нетипизированными файлами могут применяться все процедуры и функции, доступные типизированным файлам, за исключением Read и write, которые заменяются соответственно процедурами BlockRead и BlockWrite:

```
Procedure BlockRead (var F: File; var Buf;
Count: Integer [; var AmtTransferred: Integer]);
Procedure BlockWrite (var F: File; var Buf;
Count: Integer [; var AmtTransferred: Integer]);
```

Здесь

Buf – буфер: имя переменной, которая будет участвовать в обмене данными с дисками;

count – количество записей, которые должны быть прочитаны или записаны за одно обращение к диску.

AmtTransferred – необязательный параметр, содержащий при выходе из процедуры количество фактически обработанных записей.

11. ПОДПРОГРАММЫ

Подпрограмма – это небольшая программа, которая решает часть общей задачи. В языке Delphi есть два вида подпрограмм – процедура и функция.

Процедуры и функции представляют собой относительно самостоятельные фрагменты программы, оформленные особым образом и снабженные именем. Упоминание этого имени в тексте программы называется *вызовом* и процедуры (функции). Отличие функции от процедуры заключается в том, что результатом исполнения операторов, образующих тело функции, всегда является некоторое значение, поэтому обращение к функции можно использовать в соответствующих выражениях наряду с переменными и константами.

Процедура и функция – это два способы оформления подпрограммы или фрагмента программы, предназначенного для решения части общей задачи. Одну и ту же подпрограмму можно оформить как процедуру или как функцию. Если подпрограмма должна изменить значение только одной переменной основной программы, то ее предпочтительно оформлять как функцию, в остальных случаях – как процедуру.

Как правило, подпрограмма имеет параметры. Различают формальные и фактические параметры. Параметры, которые указываются в объявлении функции, называются формальными. Параметры, которые указываются в инструкции вызова процедуры, называются фактическими.

Каждую процедуру и функцию программисту необходимо описать в разделе описаний. Описать подпрограмму – это значит указать ее заголовок и тело. В заголовке объявляются имя подпрограммы и формальные параметры, если они есть. Для функции, кроме того, указывается тип возвращаемого ею результата. За заголовком следует тело подпрограммы, которое, подобно программе, состоит из раздела описаний и раздела исполняемых операторов. В разделе описаний подпрограммы могут встретиться описания подпрограмм низшего уровня, а в них – описания других подпрограмм и т. д.

Заголовок процедуры имеет вид:

procedure <имя> [(<сп.ф.п.>)];

Заголовок функции:

function <имя> [(<сп.ф.п.>)] : <тип>;

Здесь <имя> – имя подпрограммы (правильный идентификатор);

<сп.ф.п.> – список формальных параметров;

<тип> – тип возвращаемого функцией результата.

Список формальных параметров необязателен и может отсутствовать. Если же он есть, то в нем должны быть перечислены имена формальных параметров и их типы, например:

```
Procedure SB(a: Real; b: Integer; c: Char);
```

Программа состоит из основной программы и, возможно, процедур и функций. Каждая из них содержит раздел объявления переменных. Переменные, объявленные в основной программе, доступны всем операторам программы, в том числе операторам процедур и функций программиста. Такие переменные называются ***глобальными***. Переменные, объявленные в процедуре или функции программиста, называются ***локальными***.

Рекурсия – это такой способ организации вычислительного процесса, при котором подпрограмма в ходе выполнения составляющих ее операторов обращается сама к себе.

Рассмотрим классический пример – вычисление факториала. При выполнении правильно организованной рекурсивной подпрограммы осуществляется многократный переход от некоторого текущего уровня организации алгоритма к нижнему уровню последовательно до тех пор, пока, наконец, не будет получено тривиальное решение поставленной задачи. В нашем случае решение при N=0 тривиально (факториал 0 равен l) и используется для остановки рекурсии.

```
program Faktorial;
uses
 SysUtils;
var
 n:Word;
 f: Extended;
Function Factorial(k: Word): Extended;
 begin
 if k = 0
 then
 Result:=1
 else
 Result:= k * Factorial(k-1)
 end;
begin
 Readln(n);
 f:= Factorial(n);
 Writeln(f:8:2);
 Readln;
end.
```

ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Лабораторная работа № 1. Линейные алгоритмы

1. Дана колба, наполненная водой (имеющая форму шара заданного диаметра). Необходимо определить, сколько полных пробирок можно наполнить водой из этой колбы (пробирки цилиндрические с заданной высотой и диаметром основания). Задан также суммарный процент потери жидкости при переливании.

2. Задан расход краски в литрах на единицу площади. Необходимо подсчитать, сколько банок краски указанного объема нужно для окраски наружной поверхности заданного количества ведер (усеченный конус с известными диаметрами оснований и высотой), если известен процент потери краски.

3. Даны геометрические размеры некоторой комнаты (ширина, высота и длина) и размеры проемов в этой комнате (ширина с высотой окна и ширина с высотой двери). Необходимо рассчитать, сколько плиток с заданным размером (ширина и длина) нужно для отделки стен этой комнаты, если известен процент потери суммарной площади плиток (т. е. если суммарная площадь стен равна 70 $м^2$, а процент потери 10%, тогда необходимо взять 70 / (100% - 10%) $м^2$ плитки). В качестве результата вывести количество плиток (целое число).

4. Необходимо рассчитать, сколько листов бумаги заданного размера (ширина и длина и толщина) может получиться из ствола дерева с заданными геометрическими размерами (усеченный конус с заданной высотой и диаметрами оснований). Задан также процент выхода объема бумаги от исходного объема древесины.

5. Из рулона ткани заданной ширины и неограниченной длины вырезаются круглые детали заданного радиуса. Необходимо определить минимальную длину отреза ткани и процент полезной площади ткани (процент суммарной площади заготовок от общей площади отреза) для введенного количества деталей.

6. Дан золотой слиток, имеющий форму усеченной четырехугольной пирамиды (заданы размеры верхнего и нижнего основания и высота пирамиды). Необходимо определить количество монет (с заданным диаметром и толщиной), которые можно отчеканить из этого слитка, если известно, сколько процентов от объема монетки составляет рельеф на ней.

7. Дана веревка указанной длины и цилиндр указанного диаметра. Необходимо вычислить, сколько полных витков получится при наматывании веревки на цилиндр, если известен процент потери длины веревки, возникающий от увеличения диаметра витков при накладывании слоев веревки друг на друга.

8. Дан текст, состоящий из заданного количества слов (целое число). Известно среднее количество слов на одной странице (может быть дробным числом), а также процент увеличения количества слов на одной странице за счет использования переносов слов. Необходимо вывести количество страниц текста (целое число).

9. В ведро, имеющее форму усеченного конуса с заданными диаметрами оснований и высотой, помещают шары определенного диаметра. Известен процент увеличения объема шаров за счет их неплотного укладывания в ведро. Определить максимальное количество шаров в ведре.

10. В доску (плоскость) вбито три гвоздя (три точки на плоскости, заданные своими координатами). На эти гвозди (на треугольник, образованный гвоздями) наматывается нитка заданной длины (по периметру этого треугольника). Определить длину оставшейся от наматывания нити, учитывая, что некий заданный процент длины нити уходит на покрытие скруглений образованного треугольника (за счет диаметров самих гвоздей).

11. Автомобиль проезжает заданное расстояние. Известен диаметр его колеса. Определить, сколько полных оборотов совершит колесо автомобиля, если известно, сколько процентов составляет расстояние, на котором колесо «пробуксовывало», от всего, пройденного автомобилем, расстояния.

12. Известна толщина одного листа бумаги. Задана также толщина папки, в которую складывают эти листы. Определить, сколько потребуется папок для раскладывания заданного количества листов, если известен процент уплотнения толщины стопки листов.

Лабораторная работа № 2. Ветвления

1. Найти точки пересечения двух окружностей на плоскости с заданными координатами центров и радиусами.

2. Найти точки пересечения отрезка (заданного координатами своих концов) и окружности (с заданными координатами центра и радиусом). Рассматривается плоская задача.

3. Найти точки пересечения двух отрезков, заданных координатами своих концов в плоскости.

4. Дана парабола (своими коэффициентами) и отрезок (координатами своих концов), найти точки их пересечения.

5. Определить взаимное расположение точки плоскости (заданной двумя координатами) и треугольника (заданного координатами вершин).

6. Даны четыре точки в плоскости. Определить, образуют ли они четырехугольник.

7. Даны четыре точки в плоскости. Определить, образуют ли они трапецию.

8. Даны четыре точки в плоскости. Определить, образуют ли они параллелограмм.

9. Даны четыре точки в плоскости. Определить, образуют ли они прямоугольник.

10. Даны четыре точки в плоскости. Определить, образуют ли они ромб.

11. Даны четыре точки в плоскости. Определить, образуют ли они квадрат.

12. Определить взаимное расположение прямой, заданной своими коэффициентами, и отрезка, заданного координатами концов.

Лабораторная работа № 3. Циклы

1. Вычислить сумму ряда с заданной точностью:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

2. Вычислить сумму ряда с заданной точностью:

$$\sin x = \sum_{n=1}^{\infty} (-1)^{n-1} \frac{x^{2n-1}}{(2n-1)!}$$

3. Вычислить сумму ряда с заданной точностью:

$$\cos x = \sum_{n=0}^{\infty} (-1)^{n} \frac{x^{2n}}{(2n)!}$$

4. Вычислить сумму ряда с заданной точностью:

$$\ln(1+x) = \sum_{n=1}^{\infty} (-1)^{n-1} \frac{x^n}{n}$$

5. Вычислить сумму ряда с заданной точностью:

$$\operatorname{arctg} x = \sum_{n=1}^{\infty} (-1)^{n-1} \frac{x^{2n-1}}{2n-1}$$

6. Вычислить сумму ряда с заданной точностью:

$$\operatorname{sh} x = \sum_{n=1}^{\infty} \frac{x^{2n-1}}{(2n-1)!}$$

7. Вычислить сумму ряда с заданной точностью:

$$\operatorname{ch} x = \sum_{n=0}^{\infty} \frac{x^{2n}}{(2n)!}$$

8. Вычислить сумму ряда с заданной точностью:

$$\ln \frac{1+x}{1-x} = 2\sum_{n=1}^{\infty} \frac{x^{2n-1}}{2n-1}$$

9. Вычислить сумму ряда с заданной точностью:

$$\frac{1}{1+x} = \sum_{n=0}^{\infty} (-1)^n x^n$$

10. Вычислить сумму ряда с заданной точностью:

$$\frac{1}{(1-x)^2} = \sum_{n=0}^{\infty} (n+1)x^n$$

11. Вычислить сумму ряда с заданной точностью:

$$\sqrt{1+x} = 1 + \frac{x}{2} + \sum_{n=2}^{\infty} (-1)^{n-1} \frac{(2n-3)!!}{(2n)!!} x^n$$

12. Вычислить сумму ряда с заданной точностью:

$$\frac{1}{\sqrt{1+x}} = 1 + \sum_{n=1}^{\infty} (-1)^n \frac{(2n-1)!!}{(2n)!!} x^n$$

Лабораторная работа № 4. Массивы

1. Заполнить массив заданного размера квадратами натуральных чисел. Найти среднее арифметическое элементов массива.

2. Заполнить массив заданного размера значениями функции $\sin x$ для аргумента, начиная с определенного значения, с заданным шагом (в радианах). Найти максимальное и минимальное значение массива.

3. Заполнить массив заданного размера факториалами натуральных чисел. Нормализовать массив, разделив каждый его элемент на среднее арифметическое массива. Вывести полученный массив.

4. Заполнить массив заданного размера значениями функции $\cos x$ для аргумента, начиная с определенного значения, с заданным шагом (в градусах). Найти среднее арифметическое максимального и минимального значений массива.

5. Заполнить массив заданного размера последовательностью Фиббоначи (первые два элемента массива вводятся с клавиатуры, все последующие вычисляются как сумма двух предыдущих). Вывести отношение элемента массива к следующему элементу массива.

6. Заполнить массив заданного размера значениями функции $x \sin x$ для аргумента, начиная с определенного значения, с заданным шагом (в градусах). Найти разность максимального и минимального значений массива.

7. Заполнить массив заданного размера сходящейся геометрической прогрессией с заданными параметрами, найти сумму элементов массива и оценить разность этой суммы с суммой самой прогрессии.

8. Заполнить массив заданного размера значениями функции $x \cos x$ для аргумента, начиная с определенного значения, с заданным шагом (в радианах). Найти разность максимального и минимального значений массива.

9. Заполнить массив заданного размера значениями функции $x + \sin x$ для аргумента, начиная с определенного значения, с заданным шагом (в градусах). Найти среднее геометрическое элементов массива.

10. Заполнить массив заданного размера значениями функции $x + \cos x$ для аргумента, начиная с определенного значения, с заданным шагом (в радианах). Найти среднее геометрическое элементов массива.

11. Заполнить массив заданного размера значениями функции $\frac{\sin x}{x}$ для аргумента, начиная с определенного значения, с заданным шагом (в радианах). Найти максимальную по модулю разность между соседними элементами массива.

12. Заполнить массив заданного размера значениями функции $\frac{\cos x}{x}$ для аргумента, начиная с определенного значения, с заданным шагом (в градусах). Найти минимальную по модулю разность между соседними элементами массива.

Лабораторная работа № 5. Двумерные массивы

1. Во введенном двумерном массиве переставить циклически все столбцы влево.
2. Во введенном двумерном массиве переставить циклически все столбцы вправо.
3. Во введенном двумерном массиве переставить циклически все строки вниз.
4. Во введенном двумерном массиве переставить циклически все строки вверх.
5. Распечатать столбец квадратной матрицы, соответствующий максимальному элементу главной диагонали.
6. Распечатать строку квадратной матрицы, соответствующую минимальному элементу побочной диагонали.
7. Найти среднее арифметическое максимальных элементов каждой строки матрицы.
8. Найти среднее геометрическое минимальных элементов каждого столбца матрицы.
9. Найти минимум из максимальных элементов каждой строки матрицы.
10. Найти максимум из минимальных элементов каждого столбца матрицы.
11. В квадратной матрице найти максимум среди всех элементов под главной диагональю.
12. В квадратной матрице найти минимум среди всех элементов под побочной диагональю.

Лабораторная работа № 6. Строки

1. Введенная строка представляет собой вектор десятичных чисел, разделенных символом «**;**». Для двух таких строк вычислить скалярное произведение векторов.
2. Введенная строка представляет собой дату в формате ДД.ММ.ГГГГ или ГГГГ/ММ/ДД (три целых числа, разделенных точками или символами «/»). Для нескольких таких строк (каждая может быть задана своим форматом) определить, какая дата самая ранняя, выводить сформированную по таким же правилам новую строку.
3. Введенная строка представляет собой комплексное число. Для двух строк найти сумму и разность комплексных чисел, выводить сформированную по таким же правилам новую строку.
4. Введенная строка представляет собой комплексное число. Для двух строк найти произведение комплексных чисел, выводить сформированную по таким же правилам новую строку.

5. Введенная строка представляет собой комплексное число. Для двух строк найти частное комплексных чисел, выводить сформированную по таким же правилам новую строку.

6. Введенная строка представляет собой время в 24-часовом формате ЧЧ:ММ:СС (три целых числа, разделенные двоеточием), либо в 12-часовом формате ЧЧ:ММ:СС AM/PM (после времени присутствует после пробела либо строка AM – до полудня, – либо строка PM – после полудня, – причем в любом регистре). Вывести разницу во времени в 24-часовом формате (сформировав предварительно соответствующую строку).

7. Проверить правильность расстановки скобок вида «()», «[]», или «{}» во введенной строке.

8. Удалите все подстроки во введенной строке между символами «<« и «>« вместе с самими символами, оставшиеся такие символы замените на строки «<» и «>» соответственно.

9. Вводится строка – шаблон вида ##,#0.00### (используются только символы решетки, ноль, запятая и точка). Введенное десятичное число преобразуйте в строку согласно введенному шаблону. Здесь точка – разделитель целой и дробной части; запятая – разделитель групп разрядов, заменяется пробелом; ноль – обязательная цифра; решетка – необязательная цифра.

10. Разбейте строку на отдельные слова, каждое слово может отделяться от другого одним или несколькими пробелами и знаками препинания. Предусмотрите различные варианты.

11. Строка представляет собой арифметическое выражение из чисел и знаков «+» или «–», вычислить введенное выражение (скобок выражение не содержит).

12. Строка представляет собой арифметическое выражение из чисел, скобок и знака деления. Вычислить введенное выражение.

Лабораторная работа № 7. Записи

1. Опишите запись «Контакт» с соответствующими полями, характеризующими запись в телефонной книге. Заполните массив таких записей. По введенному телефону или фамилии выдавайте найденные в массиве записи.

2. Опишите запись «Книга» с соответствующими полями, характеризующими запись в каталоге книг библиотека. Заполните массив таких записей. По введенному автору или названию выдавайте найденные в массиве записи.

3. Опишите запись «Автомобиль» с соответствующими полями, характеризующими запись в базе данных автомобилей ГАИ. За-

полните массив таких записей. По введенному номеру или марке автомобиле выдавайте найденные в массиве записи.

4. Опишите запись «Лекарство» с соответствующими полями, характеризующими запись в списке лекарств аптеки. Заполните массив таких записей. По введенному названию или показанию к применению выдавайте найденные в массиве записи.

5. Опишите запись «Товар» с соответствующими полями, характеризующими запись в перечне товаров магазина. Заполните массив таких записей. По введенному названию или типу товара выдавайте найденные в массиве записи.

6. Опишите запись «Студент» с соответствующими полями, характеризующими запись в телефонной книге. Заполните массив таких записей. По введенному номеру группы или фамилии выдавайте найденные в массиве записи.

7. Опишите запись «Поезд» с соответствующими полями, характеризующими запись в расписании движения поездов. Заполните массив таких записей. По введенному пункту назначения и времени отправления выдавайте найденные в массиве записи.

8. Опишите запись «Больной» с соответствующими полями, характеризующими запись в списке пациентов больницы. Заполните массив таких записей. По введенной фамилии или диагнозу выдавайте найденные в массиве записи.

9. Опишите запись «Диск» с соответствующими полями, характеризующими запись в базе данных пункта проката. Заполните массив таких записей. По введенному названию или типу диска выдавайте найденные в массиве записи.

10. Опишите запись «Издание» с соответствующими полями, характеризующими запись в каталоге периодических изданий почты. Заполните массив таких записей. По введенному названию или разделу выдавайте найденные в массиве записи.

11. Опишите запись «Экспонат» с соответствующими полями, характеризующими запись в архиве экспонатов музея. Заполните массив таких записей. По введенному названию или эпохе, к которой относиться экспонат, выдавайте найденные в массиве записи.

12. Опишите запись «Служащий» с соответствующими полями, характеризующими запись в реестре служащих предприятия. Заполните массив таких записей. По введенной фамилии или занимаемой должности выдавайте найденные в массиве записи.

Лабораторная работа № 8. Файлы

Задание А (типизированные файлы)

Модифицируйте лабораторную работу №7 так, чтобы вся информация хранилась в файле. Реализуйте бесконечный цикл, в котором будет считываться с клавиатуры команда (или код команды), а далее, в зависимости от введенной команды, будет либо добавляться какая-то запись, либо удаляться, либо просматриваться существующие.

Задание Б (текстовые файлы)

Модифицируйте лабораторную работу №6 так, чтобы все входные строки (произвольное количество), считывались из одного текстового файла, а результаты записывались в другой текстовый файл.

Лабораторная работа № 9. Подпрограммы

Модифицируйте лабораторную работу № 3 так, чтобы расчет значения ряда при конкретном значении *x* производился отдельной функцией. В главной программе, используя описанную функцию, выведите таблицу значений ряда для аргумента, изменяющегося в введенных пределах с заданным шагом, точность вычислений также ввести с клавиатуры.

СПИСОК ЛИТЕРАТУРЫ

1. Фаронов, В.В. Delphi. Программирование на языке высокого уровня: учеб. для студ. вузов, обуч. по напр. подготовки дипломированных спец. «Информатика и вычислительная техника» / В.В. Фаронов [и др.]. – СПб.: Питер, 2005. – 640 с.
2. Долинский, М.С. Алгоритмизация и программирование на Turbo Pascal: от простых до олимпиадных задач: учеб. пособие / М.С. Долинский [и др.]. – СПб.: Питер, 2005. – 237 с.
3. Павловская, Т.А. Паскаль. Программирование на языке высокого уровня: учеб. для студ. вузов, обуч. по напр. подготовки бакалавров и магистров «Информатика и вычислительная техника» и напр. подготовки дипломированных специалистов «Информатика и вычислительная техника» / Т.А. Павловская [и др.]. – СПб.: Питер, 2004. – 393 с.
4. Лукин, С.Н. Турбо-Паскаль 7.0: Самоучитель для начинающих / С.Н. Лукин. – М.: Диалог-МИФИ, 2000. – 384 с.
5. Фаронов, В.В. Delphi 4: учебный курс / В.В. Фаронов. – М.: Нолидж, 1999. – 464 с.
6. Вирт, Н. Алгоритмы и структуры данных; пер. с англ. / Н. Вирт. – М.: Мир, 1989. – 360 с.
7. Кнут, Д. Искусство программирования / Д. Кнут. – М.: Мир, 1978.