КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ
ИНСТИТУТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

ПРОГРАММИРОВАНИЕ В СРЕДЕ DELPHI: РАЗРАБОТКА КОНСОЛЬНЫХ ПРИЛОЖЕНИЙ

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

КАЗАНЬ 2012

УДК 519.682

Печатается по решению Редакционно-издательского совета ФГАОУВПО «Казанский (Приволжский) федеральный университет»

методической комиссии института вычислительной математики и информационных технологий
Протокол № 9 от 10 мая 2012 г.

заседания кафедры информатики и вычислительных технологий
Протокол № 9 от 20 апреля 2012 г.

Рецензенты:
канд. физ.-мат. наук, с.н.с. (ИММ КазНЦ РАН) Топорков Д.Ю.
канд. пед. наук, доц. КФУ Фаткуллов И.Р.

Халитова З.Р., Хисматуллина Н.А.

Программирование в среде Delphi:разработка консольных приложений. Учебно-методическое пособие / З.Р. Халитова, Н.А.Хисматуллина. – Казань: Казанский федеральный университет, 2012. – 81 с.

Предлагаемое пособие предназначено для обеспечения аудиторных и самостоятельных занятий студентов направления подготовки «Педагогическое образование» (специальность "информатика"). Оно содержит теоретический и практический материал по программированию на языке Object Pascal.

Оглавление

Введение

В основе системы программирования Delphi для создания Windows-приложений лежит язык объектного программирования Object Pascal. Первая версия Delphi была выпущена фирмой Borland International в 1995 г. Ей предшествовали несколько версий системы Turbo Pascal для MS DOS, Turbo Pascal for Windows и Borland Pascal. Таким образом, Delphi является развитием среды программирования Turbo Pascal для современных условий. Поэтому в Delphi сохранилось многое из того, что можно было делать в системе Turbo Pascal.

Delphi – это система объектно-ориентированного визуального программирования. Delphi использует язык Object Pascal в среде визуальной разработки.

Предлагаемое пособие состоит из трех глав. В первой главе приведены элементы языка программирования Object Pascal и описание простейших типов данных. Во второй главе продолжается знакомство с типами данных, здесь приводятся структуры, позволяющие реализовать сложные алгоритмы, составлять эффективные программы. Материал третьей главы предназначен для обучения проектированию программ с использованием процедур и функций, что является неотъемлемой частью программирования в Delphi. Изложение теоретического материала сопровождается примерами и задачами с подробным объяснением их решений.

ГЛАВА 1. ЭЛЕМЕНТЫ ЯЗЫКА ПРОГРАММИРОВАНИЯ OBJECT PASCAL

1. Начальные сведения о структуре программы. Алфавит языка

Алфавит – это совокупность допустимых в языке символов. Алфавит языка Object Pascal включает в себя буквы латинского и русского алфавитов, арабские цифры, специальные символы и зарезервированные слова. В языке нет различия между прописными и строчными буквами, если только они не входят в символьные и строковые выражения. Специальные символы Object Pascal: + -

* / = , ‘ . : ; < > [ ] { } ^ @ $ # и т.д. Зарезервированные слова: and, array, as, begin, case, class, const, constructor, destructor, div, do, downto, else, end, except, exports, file, finalization, finally, for, function, goto, if, implementation, in, inherited, initialization, inline, interface, is, label, library, mod, nil, not, object, of, or, out, packed, procedure, program, property, record, repeat, set, shl, shr, string, then, try,type, unit, until, uses, var, while, with, xor и др.

Идентификатор

Идентификатор - это последовательность символов произвольной длины, которая может содержать латинские буквы, цифры, символы подчеркивания и начинается с буквы или символа подчеркивания. Имя программы или любого определяемого в ней объекта является идентификатором. Например, последовательности символов *a, ab, c1, program_1* являются идентификаторами; а последовательности *1a, program 1, программа* идентификаторами не являются.

Создание консольных приложений

Первые программы в среде Delphi удобно создавать как консольные приложения. Консоль — это монитор и клавиатура, рассматриваемые как единое устройство. Консольное приложение — программа, предназначенная для работы в операционной системе MS-DOS (или в окне DOS), для которой устройством ввода является клавиатура, а устройством вывода — монитор, работающий в режиме отображения символьной информации. Для создания консольного приложения нужно выполнить команду меню **File → New → Other….** На вкладке **New** окна диалога выбрать пиктограмму **Console Application.** В результате появится следующий шаблон приложения:

```
program Project1;
{$APPTYPE CONSOLE}
uses  SysUtils;
begin
    {TODO -oUser-cConsole Main : Insert code here}
end.
```

Структура программы

Программа состоит из заголовка, директив компилятора, строки **uses** и блока программы.

Заголовок программы имеет вид:

program *имя программы*;

например, **program** Project1;

Блок состоит из раздела описаний и раздела операторов. Раздел описаний состоит из описаний меток, констант, типов, переменных, процедур и функций. Раздел операторов имеет вид:

begin

операторы

end.

Раздел операторов программы на языке Object Pascal представляет собой последовательность инструкций (операторов), заключенную в операторные скобки **begin end**. Одна инструкция от другой отделяется точкой с запятой. Комментарии к программе пишутся либо в фигурных скобках {}, либо внутри пар символов (* и *). Если комментарий однострочный, то перед комментарием можно поставить две наклонные черты //.

Примеры комментариев:

{*заголовок программы*}

(* *ввод данных* *)

// *вывод результата*

В консольном приложении после строки заголовка следует строка {$APPTYPE CONSOLE}, которая, хотя и похожа на комментарий, таковой не является, так как сразу за открывающей скобкой следует знак $. Эта директива предназначена для компилятора. Следуя ее указаниям, компилятор генерирует исполняемую программу как консольное приложение.

В следующей строке после служебного слова **uses** указано имя подключаемого к программе модуля.

uses SysUtils;

Этот модуль обеспечивает возможность использования в коде программы стандартных идентификаторов.

Для ввода данных в консольных приложениях используются операторы r**ead, readln**, а для вывода — **write, writeln**. Оператор **readln;** (без аргументов) используется для задержки (до нажатия клавиши Enter) DOS-окна с сообщениями консольного приложения. В консольном приложении при выводе сообщений используются буквы лишь латинского алфавита.

2. Типы данных

Любая программа обрабатывает некоторые данные. Данные могут храниться в виде констант и переменных. Каждая константа с именем и переменная в программе должна быть объявлена. Объявить переменную означает указать ее тип. Самый простой способ использования констант – это указание их значения. Способ объявления констант с именем описан в следующем разделе. По **типу данных** компилятор определяет множество допустимых значений этих данных, объем памяти, отведенный под их хранение, а также совокупность операций над ними.

В языке Object Pascal используется следующая классификация основных типов данных: простые, строковые, структурированные, процедурные и типы-указатели. Их можно разделить на предопределенные в языке типы (стандартные) и типы, определяемые пользователем (нестандартные). К предопределенным относятся целые, вещественные, символьные, логические, строковые типы, текстовый файл (textfile) и тип-указатель (pointer). Определяемые пользователем типы могут быть использованы либо непосредственно в описании переменной или типизированной константы, либо их следует описать в разделе type. Раздел описания типов имеет вид:

type

имя типа1 = тип1;

имя типа2 = тип2; и т.д

Кроме того, типы данных подразделяются на *фундаментальные* и *генерируемые.* Особенность генерируемых типов в том, что выделяемая для данных память зависит от типа процессора и операционной среды. Выбор конкретной реализации производится автоматически, причем так, чтобы обеспечить оптимальную эффективность вычислений.

Простые типы подразделяются на порядковые и вещественные, порядковые, в свою очередь, - на логические, целые, символьные, перечисляемые и диапазонные.

Все возможные значения любого порядкового типа образуют упорядоченное множество, с каждым из них можно сопоставить некоторое целое число – порядковый номер значения. Первое значение любого порядкового типа, за исключением целого, имеет порядковый номер – 0, следующее – 1 и т.д. Для величин любого порядкового типа определены двуместные операции сравнения: равно (=), не равно (<>), меньше (<), больше (>), меньше или равно (<=), больше или равно (>=), результат которых вычисляется в соответствии с порядковыми номерами операндов.

К **логическим типам** данных относятся: Boolean, ByteBool, WordBool и LongBool, которые различаются только длиной (1, 1, 2, 4 байт соответственно).

Данное логического типа может принимать одно из двух значений: *true* (истина) или *false* (ложь). Порядковый номер *false* равен 0, а *true* - 1. Поэтому, *false< true*. *False* и *true* – это две логические константы, которые можно использовать в языке Object Pascal. К данным логического типа применимы логические операции: not (не), and (и), or (или), xor (исключающее или) и операции сравнения.

Таблица истинности операции not имеет вид:

X	not X
false	*true*
true	*false*

Таким образом, результатом операции *not false* является *true*, а *not true* – *false*.

Таблица истинности операций and, or, xor:

X	Y	X and Y	X or Y	X xor Y
false	*false*	*false*	*false*	*false*
false	*true*	*false*	*true*	*true*
true	*false*	*false*	*true*	*true*
true	*true*	*true*	*true*	*false*

Например: для *true* or *false* результат – *true*, а для *true* and *false* – *false*.

Целые типы используются для представления целых чисел. Существует несколько целых типов данных. Они различаются диапазоном допустимых значений и размером отводимой памяти. Фундаментальные целые типы данных языка Object Pascal приведены в следующей таблице:

Тип	**Диапазон**	**Размер в байтах**
ShortInt	-128…+127	1
SmallInt	-32 768... +32 767	2
LongInt	-2 147 483 648 ... +2 147 483 647	4
Int64	$-2^{63}\ldots +2^{63}-1$	8
Byte	0…255	1
Word	0…65 535	2
LongWord	0 … 4 294 967 295	4

Два генерируемых целых типа Integer и Cardinal при достаточных системных ресурсах рассматриваются 32-разрядным компилятором как данные формата 32 бита, со знаком и без него соответственно. При худших условиях их формат может быть уменьшен, так что привычный в языке Turbo Pascal целый тип Integer нужно использовать с осторожностью.

Константа целого типа, задаваемая значением, записывается в виде последовательности цифр, которой может предшествовать знак + или -: 25, -175 и т.д.

Целые константы можно записывать как в десятичной, так и шестнадцатеричной системе счисления. Перед шестнадцатеричной константой ставится знак $. Например: -35; $FA8; 0.

Порядковым номером значения целого типа является само это значение.

Над данными целого типа могут выполняться арифметические операции и операции отношения. Арифметические операции: изменение знака (+ или -), сложение (+), вычитание (-), умножение (*), деление (/), целочисленное деление (div), остаток от деления (mod). Результаты перечисленных арифметических операций, кроме операции деления (/), являются значениями целого типа. Результат операции деления (/) двух целых величин всегда - значение вещественного типа. Например:

4*6 результат 24 (целого типа)

5 mod 2 результат 1 (целого типа)

5 div 2 результат 2 (целого типа)

4 div 2 результат 2 (целого типа)

4/2 результат 2 (вещественного типа)

Операции отношения предназначены для сравнения величин, их результат имеет логический тип. Операции сравнения: равно (=), не равно (<>), меньше (<), больше (>), меньше или равно (<=), больше или равно (>=).

Например: 7< 3 результат *false* (ложь)

90<>45 результат *true* (истина)

Вещественные типы данных предназначены для чисел с плавающей точкой. Вещественные типы различаются между собой диапазоном допустимых значений, количеством значащих цифр и количеством байтов, необходимых для хранения данных в памяти компьютера. К фундаментальным вещественным типам относятся:

Тип	**Диапазон**	**Значащих цифр**	**Байт**
Single	$1.5 \times 10^{-45} \ldots 3.4 \times 10^{38}$	7-8	4
Real48	$2.9 \times 10^{-39} \ldots 1.7 \times 10^{38}$	11-12	6
Double	$5.0 \times 10^{-324} \ldots 1.7 \times 10^{308}$	15-16	8
Extended	$3.6 \times 10^{-4951} \ldots 1.1 \times 10^{4932}$	19-20	10
Comp	$-2^{63}+1 \ldots +2^{63}-1$	19-20	8
Currency	-922 337 203 685 477.5808 … 922 337 203 685 477.5807	19-20	8

Вещественный тип Currency предназначен для использования в финансовых вычислениях. Переменные этого типа хранятся не как вещественные, а как целочисленные типа Int64, а при использовании их в выражениях автоматически делятся на 10000. Таким образом, значения переменных типа Currency являются числами не с плавающей, а с фиксированной точкой. Генерируемый вещественный тип Real имеет длину в 8 байт и диапазон значений такой же, как тип Double.

Вещественные константы записываются с использованием десятичной точки и/или экспоненциальной части. Экспоненциальная часть начинается символом E, за которым могут следовать знак "+" или "–" и десятичный порядок. Например: -6.8; 5.0; 4.56E2; -7.9E-15. Здесь 4.56E2 означает число 4.56, умноженное на 10 в степени 2, т.е. 456, а -7.9E-15 означает число минус 7.9, умноженное на 10 в степени минус 15.

Над данными вещественного типа могут выполняться арифметические операции (+, -, *, /) и операции отношения (=, <>, <, >, <=, >=). Результат арифметических операций будет вещественного типа, если хотя бы один операнд вещественный.

Язык Object Pascal поддерживает два фундаментальных **символьных типа** AnsiChar и WideChar. Тип AnsiChar предназначен для символов в кодировке ANSI длиной 1 байт, а тип WideChar - для символов в кодировке Unicode длиной 2 байта. Символьный тип Char в Object Pascal стал генерируемым, он эквивалентен типу AnsiChar. Для типов Char и AnsiChar символы кодируются от 0 до 255, для типа WideChar от 0 до 65535.

Константа символьного типа, задаваемая значением, записывается в виде некоторого символа алфавита, заключенного в апострофы. Например: ‘A’, ‘+’. Над данными символьного типа выполняются операции отношения. Сравнение символов осуществляется по их кодам. Например: результат операции ‘C’<‘A’ равен *false*, т.к. код символа ‘C’ - 67, код символа ‘A’ – 65 и результат сравнения 67<65 будет *false*; а результат операции ‘3’>‘1’ равен *true,* так как код символа ‘3’ - 51, код ‘1’ - 49 и результат сравнения 51>49 будет *true.*

Object Pascal поддерживает три фундаментальных **строковых типа данных**: ShortString, AnsiString и WideString.

Строковая константа – последовательность символов, заключенная в апострофы. Например: ‘текст’, ‘123’.

Переменной, описанной с использованием типа String, назначается тип ShortString, если задана директива компилятора {$H-}, или как AnsiString, если {$H+}. Под эту переменную, значениями которой являются строки из 8-битных символов, отводится в первом случае до 256 байт памяти, а во втором - до 2 Гигабайт. Строка типа WideString состоит из символов, под каждый из которых отводится 16 бит (стандарт UNICODE), ее длина до 2 Гб.

К **структурированным типам** данных относятся массивы, записи, множества, файлы и классы. **Процедурные типы** предназначены для использования процедур и функций в качестве параметров подпрограмм. **Тип-указатель** используется для управления динамической памятью.

3. Виды данных

Константы

Константа – это величина, значение которой не может быть изменено в процессе выполнения программы. Константу можно задать, указав непосредственно ее значение или имя. Задание констант именами (идентификаторами) осуществляется в разделе описания констант const.

Константы, задаваемые именами, подразделяются на обычные константы, тип которых определяется их значением, и типизированные константы, для которых тип указывается явно. Обычные константы могут быть целого, вещественного, символьного, логического и строкового типа, типизированные константы – любого типа, кроме файлового. Обычные константы описываются следующим образом:

const

имя константы = значение константы;

Примеры описания обычных констант:

const

a = -15; b =3.4E-6; x = 18.4E+2; eps = 1E-06;

c = ‘*’; h = true; p =’no’; hex = $12FA;

Типизированные константы описываются следующим образом:

const имя константы: тип константы = значение константы;

Например: **const** maxint: integer = 9999;

r: real = -0.5;

Типизированные константы могут использоваться как переменные, в том числе и в левой части оператора присваивания.

Переменные

Переменная - это величина, значение которой может изменяться в процессе выполнения программы.

Описание переменных имеет вид:

var

имя переменной: тип переменной;

Если несколько переменных имеет один и тот же тип, то их имена можно перечислить через запятую и указать их общий тип.

Например:

var

a, b : integer;

c: real; d: boolean; f: char;

Выражения

Выражение состоит из констант, переменных, функций, знаков операций и круглых скобок. Круглые скобки используются для задания порядка выполнения операций в сложных выражениях. При отсутствии скобок значение выражения вычисляется в соответствии с приоритетом входящих в него операций:

- not, @, знак числа (унарные + или –)
- *, /, div, mod, and
- +, -, or, xor

- =, <>, <, >, <=, >=, in

Здесь операции, расположенные в одной строчке, имеют одинаковый приоритет, т.е. выполняются слева направо, если же две операции записаны в разных строчках, то более высоким приоритетом обладает операция, записанная выше. Так, операции not, @, унарные + или – - имеют наивысший приоритет, а операции сравнения и in - самый низкий приоритет.

Значением арифметического выражения является число, строкового выражения - последовательность символов, логического выражения – значения true или false. Например:

5 +7 div 2 - арифметическое выражение

(x+1>0) and (2*x-3<=5) – логическое выражение

‘a’ + ‘b’ - строковое выражение

Константные выражения – это выражения, которые могут быть вычислены на стадии компиляции. Они могут состоять из констант, знаков операций, круглых скобок и некоторых стандартных функций (abs, chr, length, odd, ord, pi, pred, round, succ, trunc и т.д.). Константные выражения записываются в разделах описания констант и типов.

Например:

```
const
  n=5;
  m=2*n+succ(n-3);  //Здесь 2*n+succ(n-3) – константное выражение.
```

Стандартные функции

Как отмечалось выше, в выражениях могут быть использованы функции. В языках программирования различают функции пользователя и стандартные функции. Функции пользователя должны быть описаны в разрабатываемом приложении. Однако наиболее употребительные функции описаны в модуле System, который является составной частью среды программирования Delphi. Такие функции называются стандартными и используются в коде программы без описания.

Наиболее часто используемые стандартные функции представлены в следующей таблице:

Функция	**Назначение**	**Тип аргумента**	**Тип результата**
abs(X)	Абсолютное значение X	целый, вещественный	совпадает с типом X
arctan(X)	Арктангенс X	целый, вещественный	вещественный
chr(X)	Символ, код которого равен X	целый	символьный
cos(X)	Косинус X (X в радианах)	целый, вещественный	вещественный
exp(X)	Экспонента X (e^X)	целый, вещественный	вещественный
frac(X)	Дробная часть X	вещественный	вещественный
int(X)	Целая часть X	вещественный	вещественный
ln(X)	Натуральный логарифм X	целый, вещественный	вещественный
odd(X)	Проверка X на нечетность $\begin{cases} \text{true}, \text{если X – нечетное} \\ \text{false}, \text{если X – четное} \end{cases}$	целый	логический
ord(X)	Порядковый номер X	порядковый	целый
pi	$\pi = 3.14159265358979324$		вещественный
pred(X)	Предыдущее значение X	порядковый	совпадает с типом X
round(X)	Округление X до ближайшего целого	вещественный	целый
sin(X)	Синус X (X в радианах)	целый, вещественный	вещественный
sqr(X)	Квадрат X	целый, вещественный	совпадает с типом X
sqrt(X)	Корень квадратный из X	целый, вещественный	вещественный
succ(X)	Следующее значение X	порядковый	совпадает с типом X
trunc(X)	Целая часть X	вещественный	целый

Замечание: $x^y = e^{\ln x^y} = e^{y \ln x}$, где x>0. Поэтому вычисление значения x^y программируется с помощью выражения $\exp(y * \ln(x))$

Примеры:

1) Вычислим значения функций:

abs(-3)=3	sqr(2)=4	sqrt(25)=5.0	odd(7)=true
sin(pi/2)=1.0	exp(1)=2.718282	round(5.6)=6	trunc(5.6)=5
frac(5.6)=0.6	int(5.6)=5.0	ord(-5)= -5	ord(false)=0
ord(true)=1	ord('1')=49	chr(49)='1'	pred(5)=4
pred('5')='4'	pred(true)=false	succ(5)=6	succ('5')='6'
succ(false)=true			

2) Запишем выражения на языке Object Pascal:

$ax^2 + bx + c \rightarrow a * sqr(x) + b * x + c$

$\sqrt{\frac{3x^2 - y}{|x + y|}} \rightarrow sqrt((3 * sqr(x) - y) / abs(x + y))$

$tg2x + \sin^2 x - \cos\frac{\pi}{7} \rightarrow \sin(2 * x) / \cos(2 * x) + sqr(\sin(x)) - \cos(pi / 7)$

$e^{x+3} - \frac{\ln 2y}{x^3 - 1} \rightarrow \exp(x + 3) - \ln(2 * y) / (sqr(x) * x - 1)$

$\sqrt[5]{z^2 + 1} + \sqrt[7]{t} \rightarrow \exp(1/5 * \ln(sqr(z) + 1)) + \exp(1/7 * \ln(t))$

4. Программирование линейных алгоритмов

Для программирования линейных алгоритмов используются операторы присваивания, ввода и вывода, вызова подпрограмм.

С помощью *оператора присваивания* переменной, записанной в левой части, присваивается значение выражения, записанного в правой части. Оператор присваивания имеет вид:

имя переменной :=выражение;

Типы выражения и переменной должны быть совместимы для присваивания. Пример:

```
var  a,b:real; s:integer;
     c:char; t:string;
```

```
begin
    {Примеры разрешенных присваиваний}
    s:=5; c:='+'; b:=2*s-1.6;  a:=s;  t:=c;
    {Примеры запрещенных присваиваний}
    s:=b; { переменной целого типа нельзя присвоить вещественное значение}
    c:=t; {символьной переменной нельзя присвоить строковое значение}
end.
```

Для ввода исходных данных с клавиатуры используются процедуры:

read(список ввода);

readln(список ввода);

Список ввода - это перечисленные через запятую имена переменных, которым присваиваются введенные с клавиатуры значения. Переменные могут быть целого, вещественного, символьного и строкового типов.

При вводе с клавиатуры информация представляет собой строку символов, т. е. имеет строковый тип. В процессе ввода происходит автоматическое преобразование строковых значений к типу переменных, входящих в список ввода. При наборе информации с клавиатуры данные сначала помещаются в виде строки в буфер ввода. Преобразование и передача значений из буфера ввода в переменные списка возможно только, если в буфер ввода введен признак конца строки (#13), этот символ вводится в буфер ввода клавишей Enter. При вводе значений из буфера используется указатель считывания.

Процедура read ожидает ввода с клавиатуры значений переменных из списка. Значения числовых переменных разделяются в строке символами пробел (#32), Tab (#9) или признаком конца строки (#13). Если в списке ввода указаны имена символьных переменных, то их значения вводятся подряд без разделителей. Признаком конца значения типа string является признак конца строки (#13).

Примеры:

1)
```
var  a,b:integer;
begin read(a,b); end.
```

Если переменным a и b следует присвоить значения 5 и –76 соответственно, то ввести их с клавиатуры можно следующим образом:

5|_|-76Enter или 5 Enter
-76 Enter

2) var c,d:char;

begin read(c,d); {c должно получить значение ’+’, d - значение ’z’}

end.

Ввод с клавиатуры следует осуществить следующим образом:

+z Enter

3) var c:char; d:integer;

begin read(d,c); { d=23, c=’*’} **end.**

Если ввести: 23|_| * Enter

то d=23, c=’|_|’

Таким образом, если в списке ввода оператора read необходимо указать имена как числовых, так и символьных переменных, то список имен символьных переменных должен предшествовать списку имен числовых переменных. Ввод данных в иной последовательности с использованием одного оператора read невозможен.

read(c,d); Ввести: *23 Enter

4) var s:string ;

begin read(s); { s=’8 марта’} **end.**

Ввести: 8 марта Enter

5) var a,b:string ;

begin read(a); read(b) **end.**

Если ввести:

да Enter ,

то a=’да’ и b=’’.

Ввод значений двух переменных, первая из которых имеет тип string, процедурами read невозможен, необходимо использовать процедуры readln, причем в списке ввода должно быть имя только одной переменной типа string. В отли-

чие от процедуры read процедура readln после ввода значений всех переменных списка очищает буфер ввода. Этот оператор может быть использован с пустым списком ввода в виде readln; для приостановки работы программы, т.е. организации паузы до нажатия клавиши Enter.

var a,b:string ;

begin readln(a); readln(b) {a=’мама’, b=’папа’}**end.**

Ввести: мама Enter

папа Enter

Для вывода на экран используются процедуры:

write(список вывода);

writeln(список вывода);

Список вывода содержит выражения, значения которых могут быть целого, вещественного, символьного, логического или строкового типов. Вещественные значения выводятся с использованием экспоненциальной части.

Процедура **write(список вывода)** выводит значения из списка подряд и устанавливает курсор в следующую позицию после последнего символа.

Процедура **writeln(список вывода)** выводит значения из списка подряд, но после вывода последнего значения переводит курсор в первую позицию следующей строки экрана.

Процедура **writeln;** переводит курсор в первую позицию следующей строки экрана.

Для вывода значений целого, символьного, логического и строкового типов можно использовать формат вывода: **выражение:m**, где m- длина поля, в котором размещается выводимое значение, оно выводится прижатым к правому краю поля. Для вывода значений вещественного типа используется формат вывода: **выражение:m: n**, где m – общая длина поля, n – количество позиций, отводимых под дробную часть.

Если указанной длины поля недостаточно для размещения значения, то поле автоматически раздвигается до нужной длины.

Примеры: На экране:

```
write(15);                  15 _
writeln('a=',-12);          a=-12
                            _
writeln(-15.6:7:2);         -15.60
                            _
write(-15.6);               -1.56000000000000E+0001 _
```

Задача 4.1. Даны длины катетов прямоугольного треугольника. Вычислить длину гипотенузы и площадь треугольника.

Введем обозначения:

a,b – длины катетов треугольника; c – длина гипотенузы; s – площадь.

Входные данные: a,b.

Выходные данные: c,s.

Вводятся a и b. Длина гипотенузы вычисляется по формуле: $c = \sqrt{a^2 + b^2}$, площадь – по формуле: $s = \frac{ab}{2}$. Результат выводится на экран.

```
program Project4_1;
{$apptype console}
uses  SysUtils;
var
    a,b,c,s:real ;                  {описание переменных}
begin
        write('a=');                {вывод на экран сообщения a=}
        readln(a);                  {ввод a}
        write('b=');                {вывод на экран сообщения b =}
        readln(b);                  {ввод b }
        c:=sqrt(sqr(a)+sqr(b));     {вычисление гипотенузы}
        s:=a*b/2;                   {вычисление площади}
        writeln('c=',c:6:1,' s=',s:6:1);   {вывод результатов}
        readln
end.
```

Задача 4.2. Дано действительное число x. Вычислить: значение выражения $y = \frac{\sqrt{|3x+2|}}{x^4+1} + \sin\frac{x}{7}$.

Введем обозначения:

x –заданное действительное число; y – результат.

Входные данные: x.

Выходные данные: y.

```
program Project4_2;
{$apptype console}
uses  SysUtils;
var
    x,y:real ;                    {описание переменных}
begin
      write('x=');                {вывод на экран сообщения x=}
      readln(x);                  {ввод x}
      y:= sqrt(abs(3*x+2))/(sqr(sqr(x))+1)+sin(x/7);
                  {вычисление y }
      writeln('y=',y:8:2);        {вывод y }
      readln
end.
```

5. Программирование разветвляющихся алгоритмов

Для программирования ветвлений в разветвляющихся алгоритмах используются условный оператор, операторы выбора и безусловного перехода.

Условный оператор предназначен для выполнения тех или иных действий в зависимости от истинности или ложности некоторого условия. Условие задается логическим выражением. ***Условный оператор*** имеет полную и сокращенную форму записи. Полная форма условного оператора имеет вид:

if *логическое выражение* **then** *оператор_1* **else** *оператор_2*;

Вычисляется значение *логического выражения*; если оно равно true, то выполняется *оператор_1*, если же - false - *оператор_2*.

Сокращенная форма условного оператора имеет вид:

if *логическое выражение* **then** *оператор*;

Вычисляется значение *логического выражения*; если оно равно true, то выполняется *оператор*, записанный после then, если же - false, то осуществляется переход к следующей команде.

Пример: **if** x>y **then** z:=sqr(x) **else** z:=y;

Здесь в результате выполнения условного оператора переменная z в любом случае получает новое значение.

if (x>=2) and(x<4) **then** b:=sin(x);

Здесь, например, при x = 5 переменная b сохраняет то значение, которое она имела до выполнения условного оператора.

В условном операторе после служебных слов then или else записывается лишь один оператор; если необходимо выполнить несколько действий, то соответствующие операторы объединяются в составной оператор, который имеет вид:

begin

оператор_1;

оператор_2;

…;

оператор_n

end;

Пример: **if** x>y **then begin** min:=y; max:=x **end**

else begin min:=x; max:=y **end;**

В качестве операторов после служебных слов then и else в условном операторе можно использовать и условные операторы. Такой оператор (один условный оператор, вложенный в другой) называется вложенной конструкцией условного оператора. При вложенных конструкциях условного оператора могут возникнуть неоднозначности в понимании того, к какой из вложенных конст-

рукций условного оператора относится else. Компилятор всегда считает, что else относится к последней из конструкций if, в которой не было раздела else.

В условном операторе вида:

if A **then**

if B **then** оператор1 **else** оператор2**;**

вычисляется значение логического выражения A, если оно равно true, то выполняется условный оператор в полной форме: **if** B **then** оператор1 **else** оператор2, если оно равно false, то этот условный оператор не выполняется.

Если в условном операторе в полной форме после служебного слова then нужен условный оператор в сокращенной форме, необходимо записать:

if A **then**

begin if B **then** оператор1 **end**

else оператор2**;**

где A,B – логические выражения.

С помощью ***оператора выбора*** можно выбрать один из нескольких вариантов. Оператор выбора имеет вид:

case *выражение* **of**

константа_1:оператор_1;

константа_2:оператор_2;

…….

константа_n:оператор_n;

else *оператор*

end;

Сначала вычисляется значение *выражения* (имеющего порядковый тип), затем среди *констант* отыскивается константа, равная вычисленному значению. Выполняется *оператор*, записанный после найденной константы, и оператор выбора завершает работу. Если в списке выбора не будет найдена указанная константа, то выполняется *оператор*, стоящий за словом **else**. Если же часть ***else оператор*** отсутствует, и в списке выбора нет нужной константы, то выполнение оператора выбора завершается.

Пример: **case** m **of**

12,1,2:writeln(‘зима’);

3,4,5:writeln(‘весна’);

6,7,8:writeln(‘лето‘);

9,10,11:writeln(‘осень’)

else writeln(‘ошибка в данных’)

end;

Оператор безусловного перехода позволяет перейти к нужному оператору, при этом нарушается естественный порядок выполнения операторов. Оператор имеет вид: **goto** *метка*; в качестве метки используется идентификатор или целое число без знака. Метка описывается в разделе label:

label *метка_*1,*метка_*2, *метка_n*;

Например: **label** 1,ab;

Одной меткой можно пометить только один оператор. Метка отделяется от помеченного оператора двоеточием: *метка:оператор*;

Пример:

label t;

var x,y:real;

begin … **goto** t; …..

t: y: =sqr(x); …..

end.

Задача 5.1. Дано действительное число x. Вычислить значение y:

$$y = \begin{cases} 4x^2 + 17x + 31, \textit{если } x < 1 \\ \ln 2x + 1, \textit{если} \quad x \geq 1 \end{cases}$$

Входные данные: x.

Выходные данные: y.

Условия, записанные в формулировке задачи, являются взаимно противоположными. Это означает, что если при некотором значении x выполняется одно из них, то другое не выполняется, и наоборот, если одно из них не выполняется, то другое обязательно выполнится. Поэтому при программировании дос-

таточно одного условного оператора в полной форме. Алгоритм имеет следующий вид. Вводится действительное число x. Проверяется условие x<1, если это условие выполняется, то y вычисляется по формуле $y = 4x^2 + 17x + 31$, в противном случае (т.е. если выполняется противоположное условие x≥1), то y вычисляется по формуле $y = \ln 2x + 1$. Результат выводится на экран.

```
program Project5_1;
{$apptype console}
uses  SysUtils;
var   x, y:real ;
begin
      write('x='');    readln(x);
      if  x<1 then y:=4*sqr(x)+17*x+31 else y:= ln(2*x)+1;
      writeln('y='',y:8:2);readln
end.
```

Задача 5.2. Даны действительные числа a, b, c. Решить квадратное уравнение $ax^2 + bx + c = 0$.

Введем обозначения:

a, b, c – коэффициенты квадратного уравнения, d – его дискриминант; x1, x2 – действительные корни уравнения.

Входные данные: a, b, c.

Выходные данные: x1, x2.

Коэффициенты уравнения a, b, c вводятся с клавиатуры, после этого вычисляется дискриминант ($d = b^2 - 4ac$). Если дискриминант больше или равен нулю, то уравнение имеет два действительных корня, которые вычисляются по формулам: $x_1 = \frac{-b + \sqrt{d}}{2a}$ и $x_2 = \frac{-b - \sqrt{d}}{2a}$. Если дискриминант отрицательный, то уравнение не имеет действительных корней, можно вывести сообщение “no”.

```
program Project5_2;
{$apptype console}
```

```
uses  SysUtils;
var    a,b,c,d,x1,x2:real ;
begin
        write('a='');  readln(a);
        write('b='); readln(b);
        write('c='); readln(c);
        d:=sqr(b)-4*a*c;
        if d>=0 then
                    begin
                        x1:=(-b+sqrt(d))/(2*a);
                        x2:=(-b-sqrt(d))/(2*a);
                        writeln('x1='',x1:8:2,' x2='',x2:8:2);
                    end
                else   writeln('no');
        readln
end.
```

Если выражение d>=0 истинно (равно true), необходимо выполнить три оператора (вычисление x1, x2 и вывод на экран). Однако формат условного оператора после служебного слова then требует записи лишь одного оператора, поэтому необходимо объединить три оператора в один составной оператор с помощью операторных скобок **begin end**.

6. Программирование циклических алгоритмов

Для программирования циклических алгоритмов используются операторы цикла. В языке Pascal различают три вида операторов цикла: for (цикл с параметром), while (цикл с предусловием), repeat (цикл с постусловием).

Оператор цикла for имеет вид:

for *параметр цикла*:= *выражение_1* **to** *выражение_2* **do** *оператор*;

или

for *параметр цикла*:=*выражение_1* **downto** *выражение_2* **do** *оператор;*

где *параметр цикла* - переменная порядкового типа;

выражение_1- начальное значение параметра цикла;

выражение_2 - конечное значение параметра цикла;

оператор- тело цикла.

Сначала вычисляются и запоминаются значения *выражения_1* и *выражения_2*. Далее проверяется: значение *выражения_1* меньше или равно (для **downto** - больше или равно) значению *выражения_2*. Если нет, то выполнение оператора цикла завершается, если же - да, то *параметр цикла* получает значение *выражения_1*, выполняется *тело цикла* и *параметр цикла* получает следующее по порядку значение (для **downto** – предыдущее значение). Затем проверяется: *параметр цикла* меньше или равен (для **downto** - больше или равен) значению *выражения_2*. Если да, то снова выполняется *тело цикла* и *параметр цикла* получает новое значение и т.д., если же - нет, то выполнение оператора цикла завершается.

Пример:

Оператор цикла **for** i:=1 **to** 10 **do** writeln(‘i=’,i); выводит на экран целые числа от 1 до 10 в порядке возрастания, а оператор **for** i:= 10 **downto** 1 **do** writeln(‘i=’,i); - в порядке убывания.

Оператор цикла while имеет вид:

while *логическое выражение* **do** *оператор*;

Пока значение логического выражения true, выполняется *оператор*, записанный после служебного слова do, являющийся телом цикла; как только значение станет false, оператор цикла завершит свою работу. Если значение *выражения* с самого начала false, то тело цикла не выполнится ни разу.

Пример: Вывод на экран целых чисел 1, 2, 3, …, 10 при помощи цикла **while** можно реализовать следующим образом:

```
i:=1; while i<=10 do
      begin    writeln(‘i=’,i);
               i:=i+1
      end;
```

Оператор цикла repeat имеет вид:

repeat

операторы

until *логическое выражение*;

В этом случае тело цикла составляют *операторы*, записанные между служебными словами repeat и until. Они выполняются до тех пор, пока значение *логического выражения* не станет true. Поэтому, независимо от значения *логического выражения*, тело цикла repeat всегда выполняется по крайней мере один раз.

Пример: Вывод на экран целых чисел 1, 2, 3, …, 10 при помощи цикла **repeat** можно записать следующим образом:

```
i:=1;repeat
        writeln('i=',i);
        i:=i+1
   until  i >10;
```

Задача 6.1. Дано натуральное число n. Вычислить: $s = \sum_{i=1}^{n} \frac{1}{i^2}$

$$s = \sum_{i=1}^{n} \frac{1}{i^2} = 1 + \frac{1}{2^2} + \frac{1}{3^2} + ... + \frac{1}{n^2}$$

при i=1 $s_1 = \frac{1}{1^2} = 1$

при i= 2 $s_2 = 1 + \frac{1}{2^2}$

при i= 3 $s_3 = 1 + \frac{1}{2^2} + \frac{1}{3^2}$

… …..

при i= n-1 $s_{n-1} = 1 + \frac{1}{2^2} + \frac{1}{3^2} + ... + \frac{1}{(n-1)^2}$

при i= n $s_n = 1 + \frac{1}{2^2} + \frac{1}{3^2} + ... + \frac{1}{(n-1)^2} + \frac{1}{n^2}$

Таким образом, $s_i = s_{i-1} + a_i$ (i=1,2,3,…,n).

s_i - сумма i слагаемых;

s_{i-1} - сумма (i -1)-го слагаемого;

a_i - i-е слагаемое суммы ($a_i = \frac{1}{i^2}$);

Искомой является сумма n слагаемых, и значений сумм одного, двух, трех и т.д. слагаемых ($s_1, s_2, .., s_{n-1}$) запоминать не требуется. В таком случае вместо s_i и s_{i-1} можно использовать простую переменную s, значение которой будет изменяться каждый раз при прибавлении очередного слагаемого a:

s:=s+a

При этом s справа и s слева имеют разные значения: s справа – уже вычисленное известное (s_{i-1}) значение суммы, s слева – новое, вычисляемое (s_i) значение суммы. Первоначально сумма равна нулю, не просуммировано ни одно слагаемое, т.е. s:=0.

Входные данные: n.

Выходные данные: s.

При решении этой задачи можно применить любой из циклов: цикл с предусловием, цикл с постусловием или цикл с параметром. Обратите внимание, если используется цикл с предусловием или цикл с постусловием, то начальное значение для параметра цикла i:=1 и его изменение i:=i+1 обязательно указываются. В программе Project6_1a использован цикл с предусловием.

```
program Project6_1a;
{$apptype console}
uses  SysUtils;
var    n,i:integer;
       a,s:real ;
begin write('n='); readln(n);
       s:=0;
       i:=1;                  {начальное значение i}
       while i<=n do          {пока i<=n выполнить}
       begin
```

```
        a:=1/sqr(i);   s:=s+a;
        i:=i+1         {следующее значение i}
    end;
    writeln('s=',s:6:1);readln
end.
```

В программе Project6_1b используется цикл с постусловием:

```
program Project6_1b;
{$apptype console}
uses  SysUtils;
var   n,i: integer; a,s: real ;
begin write('n='); readln(n);
    s:=0; i:=1;
    repeat              {выполнить}
        a:=1/sqr(i);   s:=s+a;      i:=i+1
    until i>n;          { до соблюдения условия i>n }
    writeln('s=',s:6:1);readln
end.
```

Использование цикла с параметром (программа Project6_1c):

```
program Project6_1c;
{$apptype console}
uses  SysUtils;
var   n,i:integer; a,s:real ;
begin write('n='); readln(n); s:=0;
    for i:=1 to n do
    begin
        a:=1/sqr(i);
        s:=s+a;
    end;
    writeln('s=',s:6:1);readln
end.
```

ГЛАВА 2. СТАТИЧЕСКИЕ И ДИНАМИЧЕСКИЕ СТРУКТУРЫ ДАННЫХ

7. Простые нестандартные типы данных. Перечисляемый тип

Перечисляемый тип определяется набором идентификаторов, перечисленных в скобках. Этот набор указывается при описании соответствующего типа и является множеством значений данных этого типа. Описание перечисляемого типа осуществляется одним из двух следующих способов:

type

имя типа = (идентификатор1, идентификатор2,…, идентификаторn);

var *имя переменной*: *имя типа*;

или

var *имя переменной*: *(идентификатор1, …, идентификаторn);*

Значения перечисляемого типа считаются перенумерованными начиная с нуля, в порядке их перечисления.

Например:

```
type     colorEn= (black, red, blue, green, white, yellow);
var a:colorEn;
```

Здесь значение black имеет порядковый номер 0, значение red – 1 и т.д.

К перечисляемым типам применимы операции отношения, а также процедуры и функции, определенные для порядковых типов (pred, succ, ord и т.д.). Сравнение осуществляется по порядковым номерам значений перечисляемого типа. Так, в приведенном примере: blue > green результат - *false*, т.к. ord(blue)=2, ord(green)=3 и результат сравнения 2>3 *false*.

Процедуры ввода с клавиатуры (read, readln) и вывода на экран (write, writeln) не могут использоваться для значений перечисляемого типа. Задание значений возможно с помощью оператора присваивания.

Например: a:= red; или a:= colorEn(1); где colorEn – имя типа, 1- порядковый номер значения в списке идентификаторов.

Один и тот же идентификатор можно использовать в программе при определении только одного перечисляемого типа.

Пример ошибочного описания:

```
type    colorEn1= (black, red, blue, green, white, yellow);
        colorEn2= (gray, magenta, cyan, green);      {нельзя}
```

В стандартных компонентах и подпрограммах Delphi широко используются перечисляемые типы.

Диапазонный тип

Для диапазонного (ограниченного) типа существует базовый порядковый тип. При описании диапазонного типа из всего множества значений базового типа берется некоторый диапазон, который задается двумя константами базового типа.

type *имя типа = константа1 .. константа2*;

var *имя переменной*: *имя типа*;

или

var *имя переменной*: *константа1 .. константа2*;

где *константа1* – минимальное значение; *константа2* – максимальное значение данных описываемого типа.

Таким образом, диапазонный тип задается указанием минимального и максимального значений из базового типа, разделенных двумя точками.

Например:

```
type   index=0..99;  {базовый тип - целый}
       letter='a'..'z'; {базовый тип - символьный}
       town=(Moscow,Kazan, Arsk, London,Berlin,Tula);
       tattown= Kazan .. Arsk;     {базовый тип - town}
var    a:index; b:letter;  c:tattown;
       d: -5..10;        {базовый тип целый}
```

К данным ограниченного типа применимы все операции и функции, которые определены над значениями базового типа. Получаемые значения не должны выходить за пределы указанного диапазона.

8. Описание и использование типа массив

Массив – это структура данных, представляющая собой совокупность элементов одного и того же типа. Для массива характерно следующее: 1) к каждому элементу массива имеется прямой доступ; 2) число элементов массива определяется при его описании и в дальнейшем не меняется; 3) в памяти компьютера элементы массива располагаются в соседних ячейках друг за другом.

Все элементы массива имеют общее имя – имя массива, а указание на конкретный элемент массива осуществляется с помощью индексов. Количество индексов в обозначении элемента массива определяет размерность массива. Массив может быть одномерным (один индекс), двумерным (два индекса) и т.д.

Описание массива осуществляется следующим образом:

type *имя типа* = **array**[*тип индекса(ов)*] **of** *тип элементов*;

var *имя массива*: *имя типа*;

или

var *имя массива*: **array**[*тип индекса(ов)*] **of** *тип элементов*;

Тип индекса может быть только порядковым типом мощностью не более 2 Гбайт (то есть любой порядковый кроме типов Integer, LongInt, LongWord, Int64). Тип элементов – любой тип Object Pascal.

Двумерный массив можно описать различными способами, например:

1) type имя типа = **array**[*тип индекса_1*] **of** **array**[*тип индекса_2*] **of** *тип элементов*;

var *имя массива*: *имя типа* ;

2) type

имя типа = **array**[*тип индекса_1*, *тип индекса_2*] **of** *тип элементов;*

var *имя массива*: *имя типа*;

3) var

имя массива:**array**[*тип индекса_1*,*тип индекса_2*]**of** *тип элементов*;

Примеры:

а) Описание одномерного массива $x_1, x_2, .., x_{20}$, элементы которого являются вещественными числами:

type mas = **array**[1..20] **of** real;

var x: mas ;

или **var** x: **array**[1..20] **of** real;

b) Описание символьного двумерного массива b размера n×m:

const n=3; m= 5;

type mas = **array**[1..n,1..m] **of** char;

var b: mas ;

или **const** n=3; m= 5;

var b: **array**[1..n,1..m] **of** char;

Во всех этих примерах в качестве типа индекса используется тип-диапазон.

При задании значений константе-массиву компоненты указываются в круглых скобках и разделяются запятыми: **const** имя: тип = (значения);

Например:

type mas1=**array**[1..3] **of** integer;

const a : mas1 = (7, -5, 8);

Элементы массива получают следующие значения: $a_1 = 7$, $a_2 = -5$, $a_3 = 8$.

Для двумерного массива: 1 2

3 4

5 6

можно задать значения, описав константу - массив:

type massiv = **array**[1..3,1..2] **of** integer;

const x: massiv = ((1,2), (3,4),(5,6));

Доступ к элементам массива осуществляется указанием имени массива, за которым в квадратных скобках помещается значение индекса (индексов) элементы. Каждый индекс элемента может быть задан выражением соответствующего типа.

Примеры: x[i+2], b[succ(i), sqr(j)], x[5], b[2,3].

В языке Pascal одним оператором присваивания можно передать все элементы одного массива другому массиву идентичного типа.

var a,b : **array**[1..5] **of** integer;

```
        c: array[1..5] of integer;
begin …
   a: =b;  {все пять элементов массива a получат те же значения, что и соответствующие элементы массива b }
   c: =a;        {нельзя}
end.
```

Над массивами не определены операции отношения. Сравнивать массивы можно только поэлементно, целиком – нельзя:

```
if a = b then writeln('равны');          {нельзя}
```

Ввод и вывод массивов

Ввод, вывод и обработка массива осуществляются поэлементно. Одномерный массив вводится и выводится в цикле, цикл организуется по порядковому номеру элемента в массиве (индексу элемента).

Например, ввод одномерного массива $x_1, x_2,.., x_n$ можно осуществить так:

```
for i:=1 to n do  read(x[i]);  или   for  i:=1 to n do readln(x[i]);
```

где n – количество элементов в массиве; x – имя массива; i –порядковый номер элемента в массиве; x_i - i–й элемент массива x.

Пример вывода одномерного массива $x_1, x_2,..,x_n$:

```
for i:=1 to n do  write(x[i],' ');   или     for i:=1 to n do  writeln(x[i]);
```

Иногда для облегчения программирования массив описывают как двумерный. Такой массив можно представить себе в виде прямоугольной матрицы. Тогда первый индекс обозначает номер строки, а второй – номер столбца. Ввод и вывод элементов двумерного массива осуществляется с помощью двух вложенных циклов. Матрица вводится либо по строкам, либо по столбцам. Если по строкам, то внешний цикл организуется по номеру строки, а внутренний - по номеру столбца. Если матрица вводится по столбцам, то внешний цикл организуется по номеру столбца, а внутренний – по номеру строки. Вывод матрицы, как правило, осуществляется построчно.

Построчный ввод числовой матрицы **a** размерности n×m:

```
for i:=1 to n do
  for j:=1 to m do  read(a[i,j]);
```

где n –количество строк в матрице; m – количество столбцов в матрице; i – номер строки матрицы; j –номер столбца матрицы; a –имя матрицы; a_{ij} – элемент матрицы a, находящийся в i –й строке, j –ом столбце.

Вывод двумерного массива **a** размерности n×m в виде прямоугольной матрицы (каждая строка матрицы начинается с новой экранной строки):

```
for i:=1 to n do
begin for j:=1 to m do  write(a[i,j],' '); writeln  end;
```

Задача 8.1. Дан одномерный массив $x_1, x_2, .., x_n$. Найти максимальный элемент и его порядковый номер.

Введем обозначения:

n – количество элементов в массиве; i – порядковый номер элемента в массиве; x – имя массива; x_i – i -й элемент массива x;

max – значение максимального элемента массива;

k – порядковый номер максимального элемента.

Входные данные: n,x.

Выходные данные: max, k.

Первоначально max:=x[1], так как просмотр начинается с первого элемента и на первом шаге именно он является максимальным из всех просмотренных (другие еще не просматривались!); запоминаем его порядковый номер k:=1. Далее поочередно просматриваются все элементы массива, начиная со второго, и их значения сравниваются со значением переменной max. Если очередной элемент массива больше чем max, то переменная max меняет свое значение на значение этого элемента и запоминается его порядковый номер (max:=x[i], k:=i). Если очередной элемент массива не превосходит max, то max и k остаются без изменения.

```
program Project8_1;
{$apptype console}
```

```
uses  SysUtils;
var  x: array[1..20] of real;  i,n,k: integer;   max: real;
begin
     write('n='); readln(n);
     for i:=1 to n do readln(x[i]);
       max:=x[1]; k:=1;
       for i:=2 to n do
                if x[i]>max then
                                 begin max:=x[i];  k:=i  end;
      writeln('max =',max:8:2,' k=',k);   readln
end.
```

Задача 8.2. Сформировать массив размера mxm:

a	*b*	*b*	*b*	.	*b*
c	*a*	*b*	*b*	.	*b*
c	*c*	*a*	*b*	.	*b*
c	*c*	*c*	*a*	.	*b*
.	.	.	.	.	.
c	*c*	*c*	*c*	.	*a*

Введем обозначения:

x – имя массива;

m– количество строк и столбцов массива;

i – номер строки массива;

j – номер столбца массива;

x_{ij} – элемент массива x.

Входные данные: m.

Выходные данные: x.

Все элементы массива x, расположенные на главной диагонали, должны получить значение, равное 'a', над диагональю – значение, равное 'b', под диагональю – значение, равное 'c'. Элемент расположен на главной диагонали, если номер его строки совпадает с номером столбца, т.е. i=j. Если номер строки меньше номера столбца, т.е. i<j, то элемент расположен над главной диагона-

лью. Если номер строки больше номера столбца, т.е. i >j, то элемент расположен под главной диагональю.

```
program Project8_1;
{$apptype console}
uses  SysUtils;
var
    x: array[1..10,1..10] of  char;
    i, j, m : integer;
begin
      write('m<=10, m='');  readln(m);
      {формирование массива}
      for i:=1 to m do
       for j:=1 to m do
          if  i=j then x[i,j]:='a'
                  else
                    if  i<j then x[i,j]:='b'
                            else x[i,j]:='c';
      {вывод массива}
       for i:=1 to m do
      begin
          for j:=1 to m do write(x[i,j],' ');
          writeln
      end;  readln
 end.
```

9. Описание и использование данных строкового типа

Строка – это последовательность некоторых символов алфавита. Длиной строки называется количество символов в последовательности. Для работы со строками в языке Pascal предусмотрен строковый тип данных string. Длина строки типа string не может превышать 255.

Значением величины строкового типа является строка. Константы строкового типа записываются в апострофах. Например: ’задача’, ’57+38’, ’program_11b’, ’#1@2%3’, ’a’, ’’ (пустая строка).

Описание данных строкового типа имеет вид:

type имя типа = **string**; *или* имя типа = **string[n]**;

var имя строки: имя типа ;

или

var имя строки: **string**; *или* имя строки: **string[n]**;

где n – длина строки.

Если длина строки не указана, по умолчанию она принимается равной 255.

Примеры: **var** a: **string**; x: **string[10]**;

К любому символу строки можно обратиться непосредственно, указав за именем строки в квадратных скобках номер позиции этого символа. Например: s[i] –i-й символ строки s.

Операции над строками

К строкам применимы операции склеивания (конкатенации) и сравнения. Операция склеивания обозначается с помощью символа “+”, а операции сравнения – символами “=”,“<>”,“>”,“>=”,“<”,“<=”.

Операция склеивания добавляет к первой строке вторую.

Пример: x:=’фор’; y:=’ма’;z:=x + y; { z =’форма’}

Сравнение осуществляется слева направо в соответствии с кодами соответствующих символов. Если длина одной строка меньше другой, то недостающие символы короткой строки заменяются значениями #0.

Примеры:

‘cb’<‘ab’ результат - false

‘9’>‘123’ результат – true

‘ab’>‘a’ результат – true

Для формирования значения строковой переменной существуют два основных способа:

1) Ввод значения с клавиатуры.

Например:

var a:string[2] ; b:string[5] ;

begin read(a,b) **end.**

Если ввести:

информация Enter

то a=’ин’ и b=’форма’.

Ввод значений двух переменных процедурами read возможен, т.к. длина первой из строк меньше 255, при этом значения вводятся друг за другом.

2) Использование оператора присваивания. В левой части оператора присваивания при этом должно быть имя строки. Посимвольное заполнение строки (подобно заполнению массива) в языке Pascal запрещено. Значение отдельных символов строки можно изменять лишь после того, как строка сформирована. Например: x:=’кол’; x[3]:=’м’;{x=’ком’} y:=x+’ок’ {y=’комок’}

Процедуры и функции для работы со строками

Для работы со строковым типом данных в языке Pascal предусмотрены следующие стандартные процедуры и функции:

1. **function** Concat (S1[,S2,…,SN]:string):string;

(квадратные скобки здесь обозначают, что записанная в них часть является необязательной и может отсутствовать) склеивает строки S1,S2,…,SN.

Пример: y:=concat(‘ка’,’ток’); {y=’каток’}

2. **function** Copy (S:string ; Index, Count:integer):string ;

копирует из строки S Count символов, начиная с символа с номером Index.

Пример: s:=’информатика’; a:=copy(s,3,5); {a=’форма’}

3. **function** Length (S:string):integer ;

вычисляет текущую длину строки S.

Пример:

n:=length(‘каникулы’); {n=8}

4. **function** Pos (ST,S:string):integer ;

отыскивает в строке S первое вхождение подстроки ST и возвращает номер позиции, с которой она начинается. Если подстрока не найдена, возвращает ноль.

Примеры: a:=pos(‘рок’,’строка’); {a=3}

b:=pos(‘программа’,’программирование’); {b=0}

5. **procedure** Delete (**var** S:string ; Index,Count:integer);

удаляет Count символов из строки S, начиная с символа с номером Index.

Примеры: x:=’каток’; delete(x, 1, 2); {x=’ток’}

6. **procedure** Insert (ST:string ; **var** S:string ; Index:integer);

вставляет подстроку ST в строку S, начиная с символа с номером Index.

Примеры: s:=’свет’; insert(‘рас’,s,1); {s=’рассвет’}

7. **procedure** Val (S:string ; **var** X: integer; **var** Code:integer);

или **procedure** Val (S:string ; **var** X: real; **var** Code:integer);

преобразует строку символов S во внутреннее представление целой или вещественной переменной X, которое определяется типом этой переменной. Параметр Code содержит ноль, если преобразование прошло успешно, тогда в X помещается результат преобразования. В противном случае Code содержит номер позиции в строке S, в которой обнаружен ошибочный символ, а X состоит из символов, предшествующих символу с номером Code.

Пример:
```
var a:string ; b,c:integer; d:real;
begin
   a:=’58’; val(a,b,c);   {b=58,  c=0}
   a:=’27d5k’; val(a,b,c);   {b=27,  c=3}
   a:=’a27d5k’; val(a,b,c); {b=0,  c=1}
   val(a,d,c);  {d= 5.80000000000000E+0001,  c=0}
end.
```

8. **procedure** Str (X: integer ; **var** S:string);

или **procedure** Str (X: real ; **var** S:string);

преобразует число X любого вещественного или целого типов в строку символов S. В процедуре str можно указать формат вывода, аналогично процедурам write или writeln.

Пример: **var** s:string ; x:integer; y:real;

```
begin x:=123; str(x,s);   {s='123'}
      y:=123; str(y,s); {s=' 1.23000000000000E+0002'}
      y:=-453.68; str(y:6:1,s); {s='-453.7'}
end.
```

9. **function** StrToFloat(S:string):extended;

преобразует строку символов S в вещественное число. Разделителем целой и дробной частей числа должен быть символ, объявленный в Windows. Для русифицированной версии Windows этим символом является запятая. Если строка содержит недопустимые для вещественного типа символы, выдается сообщение об ошибке. Пример: writeln(strtofloat('-15,6')); {-1.56000000000000E+0001}

10. **function** StrToInt(S:string):integer;

преобразует строку символов S в целое число. Если строка содержит недопустимые для целого числа символы, выдается сообщение об ошибке.

Примеры: writeln(strtoint('46')); {46}

11. **function** IntToStr(X:integer):string;

преобразует целое число X в строку символов.

Пример: writeln(inttostr(56)); {'56'}

12. **function** FloatToStr(X:extended):string;

преобразует вещественное число X в строку символов.

Пример: writeln(floattostr(-15.6)); {'-15,6'}

Последние четыре функции чаще используются при программировании с использованием возможностей визуализации или создания графического интерфейса, которые предоставляет среда Delphi.

Задача 9.1. Дана строка. Удалить из этой строки символ "a".

Введем обозначения:

s – заданная строка;

i – номер символа в строке.

Входные данные: s.

Выходные данные: s.

В строке просматриваются все символы, с первого до последнего, номер последнего символа совпадает с длиной строки. Если i-й символ строки является символом “a”, он удаляется из строки, т.е. в строке s удаляется, начиная с символа с номером i, один символ - delete(s,i,1). Длина строки уменьшается на единицу, при этом символ, находившийся на i +1-й позиции, автоматически окажется на i-й, и именно он должен проверяться следующим. Если же i-й символ не является символом “a”, то проверяется следующий символ строки, для этого i увеличивается на единицу (i:=i+1).

Так как длина строки в этой задаче меняется в процессе обработки, следует использовать для организации цикла операторы while или repeat. Оператор for в данном случае использовать нельзя, так как число повторений тела цикла определяется в процессе его выполнения и заранее неизвестно.

```
program Project9_1;
{$apptype console}
uses  SysUtils;
var  s:string ;
      i:integer;
begin   writeln (‘s= ’);    readln(s);
          i:=1;
          while i<= length(s)  do
                  if  s[i]=’a’ then  delete(s,i,1) else i:=i+1;
          writeln(’s= ’, s); readln
end.
```

10. Описание и использование типа запись

Запись – это структура данных, представляющая собой совокупность компонент, возможно, разных типов. Компоненты записи называются ее полями. Запись описывается следующим образом:

```
type
имя типа записи =record
```

имя поля_1: тип поля_1;

имя поля_2: тип поля_2;

…

имя поля_n: тип поля_n

end;

var имя переменной: имя типа запись;

Имена полей одного и того же типа могут быть перечислены через запятую.

```
type complex=record       {тип комплексных чисел}
   re, im:real;
   end;
   data=record   {тип – дата}
   day:1..31;
   month:1..12;
   year:integer
   end;
var x:complex; dat:data;
```

Переменную типа запись можно описать непосредственно в разделе var:

var *имя переменной*: **record**

*имя поля_1: тип поля*_1;

имя поля_2: тип поля_2;

…

имя поля_n: тип поля_n

end;

Например:

```
var x,y: record
      re, im:real;
    end;
```

Доступ к полям записи осуществляется указанием имени переменной, затем точки и имени поля. Например: x.re:=2.5; x.im:=1.5; y.re:=-x.re; y.im:=-6;

Чтобы упростить доступ к полям записи, используется ***оператор присоединения with***, который имеет вид:

with *имя записи* **do** *оператор*;

Внутри *оператора* можно указывать только имя поля записи.

```
with x do              { x – переменная типа запись}
begin
  re:=2.5; im:=1.5
end;
```

Задача 10.1. В группе n студентов. О каждом студенте известна следующая информация: фамилия, имя студента и оценки по четырем экзаменам. Вывести на экран список студентов, получающих стипендию.

Введем обозначения:

n- количество студентов; i – порядковый номер студента;

a[i] – информация о i-том студенте;

a[i].fam –фамилия i-го студента; a[i].name – имя i-го студента;

j – номер оценки; a[i].oc[j] – j – тая оценка i-го студента;

t – строка, принимающая одно из значений ’yes’ или ’no’.

Входные данные: n, a.

Выходные данные: a[i].fam, a[i].name.

Данное типа student (тип запись) содержит фамилию (поле fam), имя (поле name) и оценки по четырем экзаменам (поля oc[1], oc[2], oc[3], oc[4]). Массив a состоит из n элементов типа student. Например, a[i]- i-й элемент массива, имеет поля: a[i].fam, a[i].name, a[i].oc[1], a[i].oc[2], a[i].oc[3], a[i].oc[4]. При использовании оператора with доступ к полям упрощается, не нужно указывать для каждого поля имя переменной a[i].

Первоначально полагаем, что каждый студент получит стипендию и переменной t присваиваем значение ’yes’. Проверяются все четыре оценки студента; если среди них есть тройка или двойка, то переменная t получает значение ’no’. Фамилия и имя студента, для которого значение t не изменилось и осталось t=’yes’, выводятся на экран.

```
program Project10_1;
{$apptype console}
uses  SysUtils;
const  n=15;
type    student=record
          fam, name:string[30];
          oc:array[1..4] of integer;
           end;
var  a:array[1..n] of student;
      i:integer; t:string;
begin
     for i:=1 to n do
          with a[i] do
          begin readln(fam); readln(name);
                for j:=1 to 4 do readln(oc[j])
          end;
          for i:=1 to n do
          with a[i] do
          begin t:= 'yes';
                for j:=1 to 4 do
                     if (oc[j]=3) or (oc[j]=2) then t:= 'no';
                if t='yes' then writeln(fam, ' ',name)
          end; readln
end.
```

11. Описание и использование множеств

Множество – это структура данных, представляющая собой набор однотипных объектов. Число элементов множества не может быть больше 256, а порядковые номера элементов должны находиться в диапазоне от 0 до 255. Множество описывается следующим образом:

type *имя типа*=**set of** *базовый тип*;

var *имя переменной*: *имя типа*;

или

var *имя переменной*: **set of** *базовый тип*;

Базовый тип множества – любой порядковый тип с не более, чем 256 возможными значениями. Порядковые значения нижнего и верхнего пределов базового типа должны быть в диапазоне от 0 до 255.

Примеры: **type** digit=set of 0..9;

letter=set of 'a'..'z';

symbol= set of char;

var a:digit; b:letter; c:symbol;

Для задания константы типа множество используется конструктор множества: в квадратных скобках через запятую перечисляются выражения базового типа, значения которых являются элементами множества. Допустимо использовать диапазоны элементов.

a:= [1, 3..7, 9]; b:=['x','b'..'n']; c:=['z','x','y','8','+'];

c:=[]; {пустое множество}

Над множествами определены следующие операции:

\+ - объединение двух множеств;

* - пересечение двух множеств;

\- - разность двух множеств;

= - проверка на равенство двух множеств;

<> - проверка двух множеств на неравенство;

<= - проверка, является ли левое множество подмножеством правого множества;

>= - проверка, является ли правое множество подмножеством левого множества;

in - проверка, является ли значение выражения, указанного слева, элементом множества, указанного справа.

Результатом операции объединения, пересечения или разности является соответствующее множество, остальные операции дают результаты логического типа.

Примеры: x:=[0..4,6]; y:=[4,5]; тогда

множество x+y равно [0..6] {элементы обоих множеств}

x*y содержит единственный общий элемент [4]

x-y содержит [0..3,6] {элементы из первого множества, которые не принадлежат второму}

операция x=y дает результат false {если оба множества состоят из одних и тех же элементов, то результат true, иначе - false }

операция x<>y дает результат true {если множества неэквивалентны, результат true, иначе - false }

операция x<=y дает результат false {если первое множество включено во второе, результат true, иначе - false }

операция x>=y дает результат false {если второе множество включено в первое, результат true, иначе - false }

операция 4 in y дает результат true {если элемент принадлежит множеству, результат true, иначе - false }

Таким образом, при работе с данными типа множество нет доступа к отдельному элементу этого множества, нет даже стандартной функции для вычисления количества элементов в данном множестве.

Задача 11.1. Даны две строки символов. Определить, есть ли в них общие символы.

Введем обозначения:

c,d – заданные строки;

a– множество, состоящее из символов строки c;

b – множество, состоящее из символов строки d;

i – порядковый номер символа в строке.

Входные данные: c,d.

Выходные данные: сообщение ‘yes’ или ’no’.

Из символов первой строки (c) формируется множество a, из символов второй строки (d) - множество b. Первоначально каждое множество пустое -[]. Просматриваются все символы c[i] заданной строки и включаются (добавляются) в множество a:=a+[c[i]], аналогично из символов строки d формируется множество b. Если пересечение множеств a и b непусто, то заданные строки имеют общие символы, иначе – не имеют. Программа имеет вид:

```
program Project11_1;
{$apptype console}
uses  SysUtils;
var a,b:set of char; c,d:string; i:integer;
begin
    write('first string '); readln(c); a:=[];
    for i:=1 to length(c) do a:=a+[c[i]];
    write('second string '); readln(d); b:=[];
    for i:=1 to length(d) do b:=b+[d[i]];
    if a*b <>[] then writeln('yes') else writeln ('no');
    readln
end.
```

12. Файловый тип данных. Описание и использование типизированных файлов

В языке программирования Pascal файл – это тип переменной. Значением такой переменной является реальный файл, расположенный на внешнем устройстве. Файловый тип описывается следующим образом:

```
type
имя типа =file of  тип компонент; {типизированный файл}
или
имя типа =textfile;              {текстовый файл}
или
имя типа =file;                  {нетипизированный файл}
```

var *имя переменной: имя типа*;

Типизированный файл содержит компоненты одного и того же объявленного типа. Они могут быть любого типа, кроме файлового. Длина каждой компоненты типизированного файла постоянна, поэтому можно организовать прямой доступ к компоненте по ее порядковому номеру. Компоненты файла нумеруются, начиная с нуля. Число компонентов в файле не объявляется.

Например:
```
type  text1=file of  string;
      student=record
         fam, name:string[30];
         grup:integer
      end;
     univ=file of student;
var  f: univ;   s: text1;
```

Процедуры и функции для типизированных файлов

1. Файлы становятся доступны программе только после связывания переменной типа файл с именем существующего или вновь создаваемого файла, а также указания направления обмена информацией: чтение из файла или запись в него. Файловая переменная связывается с файлом на диске с помощью процедуры: **assignfile(файловая переменная, имя файла);**

имя файла – строковое выражение, значение которого есть имя файла, записанное в соответствии с правилами MS DOS.

Например: assignfile(f, ‘d:\ped\a.121’); assignfile(s,’d:\b.pas’);

2. Процедура **reset(файловая переменная)** открывает существующий файл, с которым связана файловая переменная, и устанавливает указатель текущей компоненты файла на начало файла, то есть на компоненту с порядковым номером 0. После этого файл оказывается подготовленным к чтению или изменению сохраненной в нем информации. Например: reset(f);

3. Процедура **rewrite(файловая переменная)** открывает новый пустой файл, с которым связана файловая переменная, и устанавливает указатель те-

кущей компоненты файла на начало файла. После этого файл оказывается подготовленным к записи информации. Например: rewrite(s);

4. Процедура **closefile(файловая переменная)** закрывает файл на диске, с которым связана файловая переменная. Однако связь файловой переменной с именем файла сохраняется, т.е. этот файл можно повторно открыть без процедуры assignfile, достаточно указать только направление обмена (reset или rewrite). Например: closefile(f); closefile(s).

5. Процедура **read(файловая переменная, список переменных)** читает значения для переменных списка из файла, с которым связана файловая переменная. После чтения каждого значения указатель файла сдвигается к следующей компоненте файла. Переменные в списке и компоненты файла должны быть одного и того же типа. Например: read(f,a); read(s,b,c);

6. Процедура **write(файловая переменная, список выражений)** записывает значения выражений списка в файл, с которым связана файловая переменная. Указатель файла при этом устанавливается в конце очередной записанной компоненты. Значения выражений в списке должны быть того же типа, что и компоненты файла. Например: write(f,a); write(s,’алгоритм’);

7. Процедура **seek(файловая переменная, N)** устанавливает указатель файла, с которым связана файловая переменная, на компоненту с номером N. При этом именно эта компонента становится текущей, то есть именно она считывается при выполнении операции чтения и изменяется при операции записи. Компоненты в файле нумеруются с нуля. Например: seek (f,0); seek(s,4);

8. Процедура **truncate(файловая переменная)** удаляет часть файла, начиная с текущей позиции и до его конца. Например: truncate (f);

9. Процедура **erase(файловая переменная)** уничтожает файл на диске, с которым связана файловая переменная. Перед выполнением этой процедуры необходимо закрыть файл. Например: erase(f);

10. Процедура **rename(файловая переменная, новое имя файла)** переименовывает файл, с которым связана файловая переменная. Новое имя файла – строковое выражение, значением которого является имя файла, записанное в

соответствии с правилами MS DOS. Перед выполнением этой процедуры необходимо закрыть файл. Например: rename(f, ‘d:\ped\b.121’);

11. Функция **eof (файловая переменная)** принимает значение true, если указатель текущей компоненты файла находится в конце файла, и false – в противном случае. Используется, главным образом, при организации цикла для чтения данных из файла: while not(eof(f)) do … или repeat … until eof(f).

12. Функция **filepos (файловая переменная)** возвращает номер текущей компоненты файла, с которым связана файловая переменная. Например: writeln(filepos(f));

13. Функция **filesize (файловая переменная)** возвращает количество компонентов файла, с которым связана файловая переменная. Например: seek(f,filesize(f)); {устанавливает указатель в конец типизированного файла}

Задача 12.1. Создать типизированный файл, содержащий фамилию, имя и номер группы студента. Формирование файла завершить, если введены символы- ‘***’.

Введем обозначения:

f – файловая переменная; a– запись; a.fam – поле, содержащее фамилию студента; a.name - поле, содержащее имя студента; a.grup – поле, содержащее номер группы студента.

Входные данные: a.

Выходные данные: f .

Описывается запись stud с полями для фамилии, имени и номера группы студента. Типизированный файл состоит из компонент типа stud, т.е. из записей. В программе сначала устанавливается связь между файловой переменной f и файлом a.txt корневого каталога диска d, затем открывается новый пустой файл a.txt для записи, при этом указатель текущей компоненты файла указывает на начало файла. С клавиатуры вводится фамилия студента или признак завершения формирования типизированного файла. Пока не введены символы ‘***’, формирование файла продолжается: с клавиатуры вводятся имя и номер группы студента. Вся информация о данном студенте записывается в типизированный

файл с помощью процедуры write(f, a), при этом указатель файла устанавливается в конце очередной записанной компоненты. С клавиатуры снова вводится фамилия студента или признак завершения формирования типизированного файла. При вводе строки символов ‘***’ формирование файла завершается, процедура closefile(f) закрывает файл на диске, с которым связана файловая переменная f.

```
program Project12_1;
{$apptype console}
uses  SysUtils;
  type  stud = record
            fam,name:string[30];   grup:integer;
          end;
  var a:stud; f: file of stud;
begin
          assignfile(f, ‘d:\a.txt’);rewrite(f);
          readln(a.fam);
          while a.fam <>’***’ do
          begin
                readln(a.name); readln(a.grup); write(f, a);
               readln(a.fam)
         end;
          closefile(f); readln
end.
```

В результате выполнения этой программы в корневом каталоге диска d: будет сформирован типизированный файл a.txt. Для того, чтобы просмотреть содержимое этого файла (вывести всю информацию на экран) можно в программу добавить:

```
reset(f);                  {открывается существующий файл a.txt для чтения}
while not eof(f) do      {пока не конец файла, с которым связана переменная f, выполнить}
```

```
begin
    read(f,a);          {читает из файла a.txt запись a}
    writeln(a.fam, ‘ ’,a.name,’ ‘,a.grup); {выводит на экран поля записи a }
end;
```

Другого способа просмотра содержимого типизированного файла нет.

13. Описание и использование текстовых файлов

Текстовый файл представляет собой совокупность строк символов, причем каждая строка заканчивается признаком конца строки. Доступ к каждой строке возможен лишь последовательно, начиная с первой. Текстовый файл описывается следующим образом:

type *имя типа*=**textfile**;

var *имя переменной: имя типа*;

или

var *имя переменной*: **textfile**;

Например: **var** f: textfile;

Процедуры и функции для текстовых файлов

1. Так же, как в случае типизированных файлов, текстовые файлы становятся доступны программе только после связывания файловой переменной с именем существующего или вновь создаваемого файла, а также указания направления обмена информацией: чтение из файла или запись в него. Файловая переменная связывается с файлом на диске с помощью той же процедуры:

assignfile(файловая переменная, имя файла);

имя файла – строковое выражение, значение которого есть имя файла, записанное в соответствии с правилами MS DOS.

Например: assignfile(f, ‘d:\ped\a.txt’);

2. Процедура **reset(файловая переменная)** открывает существующий файл, с которым связана файловая переменная, и устанавливает указатель файла на его начало. Например: reset(f);

3. Процедура **rewrite(файловая переменная)** открывает новый пустой файл, с которым связана файловая переменная, и устанавливает указатель файла на его начало. Например: rewrite(f);

4. Процедура **append(файловая переменная)** открывает существующий файл, с которым связана файловая переменная, и устанавливает указатель в конец файла. Например: append(f);

5. Процедура **closefile(файловая переменная)** закрывает файл на диске, с которым связана файловая переменная. Например: closefile(f);

6. Процедура **read(файловая переменная, список переменных)** читает значения для переменных списка из файла, с которым связана файловая переменная. После чтения каждого значения указатель файла сдвигается к началу следующего. Переменные в списке могут быть целого, вещественного, символьного типа и типа string. Числа в текстовом файле должны отделяться друг от друга пробелом, символом табуляции (Tab) или признаком конца строки (Enter).При вводе целых и вещественных значений автоматически происходит преобразование типа. При вводе символа из файла читается один символ. При вводе значений типа string из файла читается необходимое число символов в пределах одной строки (до признака конца строки), при этом признак конца строки не читается. Например: read(f,a);

7. Процедура **readln(файловая переменная, список переменных)** читает значения для переменных списка, включая признак конца строки, из файла, с которым связана файловая переменная. После чтения каждого значения переменной из списка указатель файла сдвигается к началу следующего значения, а после завершения списка - к началу следующей строки. Например: readln(f,a);

8. Процедура **write(файловая переменная, список выражений)** записывает значения выражений списка в файл, с которым связана файловая переменная. Указатель файла при этом устанавливается на следующей позиции после записанного значения. Выражения в списке могут быть целого, вещественного, логического, символьного типа и типа string. При выводе чисел происходит их автоматическая перекодировка в символьную форму, при выводе значений ло-

гических выражений выводятся слова true или false. Например: write(f, 5); {в файл записывается символ '5'}

9. Процедура **writeln(файловая переменная, список выражений)** записывает значения выражений списка и ставит признак конца очередной строки в файл, с которым связана файловая переменная. Указатель файла при этом устанавливается в первую позицию следующей строки файла.

10. Процедура **erase(файловая переменная)** уничтожает файл, с которым связана файловая переменная. Перед выполнением этой процедуры необходимо закрыть файл. Например: erase(f);

11. Процедура **rename(файловая переменная, новое имя файла)** переименовывает файл, с которым связана файловая переменная. Новое имя файла – строковое выражение, значением которого является имя файла, записанное в соответствии с правилами MS DOS. Перед выполнением этой процедуры необходимо закрыть файл. Например: rename(f,‘d:\ped\b.txt’);

12. Функция **eof (файловая переменная)** возвращает значение true, если указатель файла указывает на признак конца файла, и false – в противном случае. Например: writeln(eof(f));

13. Функция **eoln (файловая переменная)** возвращает значение true, если указатель файла указывает на признак конца строки или признак конца файла, и false – в противном случае.

Задача 13.1. Создать текстовый файл, содержащий фамилию, имя и номер класса ученика. Удалить из файла информацию об учащихся 11 класса.

Для решения этой задачи необходимо сформировать текстовый файл содержащий всю информацию об учениках, затем в промежуточный файл записать только нужную информацию (фамилию, имя, номер класса ученика не 11 класса). Уничтожить файл, содержащий всю исходную информацию, а промежуточный файл переименовать, дав имя этого исходного файла.

Введем обозначения:

f – файловая переменная для создаваемого текстового файла;

s – файловая переменная для промежуточного текстового файла;

n- количество учеников; i – номер ученика; fam – фамилия ученика;

name - имя ученика; class – класс ученика.

Входные данные: fam, name, class.

Выходные данные: f .

С клавиатуры вводится количество учеников. Устанавливается связь между файловой переменной f и файлом d:\a.txt, этот файл открывается для записи. Вводятся с клавиатуры фамилия, имя, номер класса ученика и записываются в текстовый файл, с которым связана файловая переменная f (каждая информация записывается в новую строку файла). Таким образом, в файле d:\a.txt будет информация о n учениках. Далее этот файл открывается для чтения, при этом указатель файла указывает на его начало.

Устанавливается связь между файловой переменной s и файлом d:\b.txt (промежуточный текстовый файл), этот файл открывается для записи. Пока не конец созданного текстового файла d:\a.txt, из него считываются фамилия, имя, номер класса ученика (каждая информация с новой строки файла), если этот ученик не учится в 11 классе, вся информация о нем записывается в новый промежуточный файл d:\ b.txt, с которым связана файловая переменная s (каждый раз в новую строку файла). Таким образом, созданы два файла: в первом (d:\a.txt) находится вся информация об учениках, во втором файле (d:\b.txt) - только информация об учениках не 11 класса. Первый файл d:\a.txt уничтожается, а второй d:\b.txt переименовывается в файл d:\a.txt.

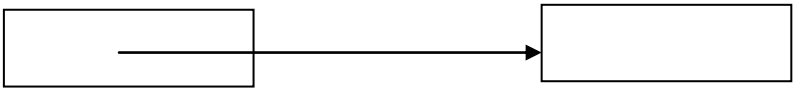
```
program Project13_1;
{$apptype console}
uses  SysUtils;
var fam,name:string; class,n,i :integer;  f,s: textfile;
begin write('n=');readln(n);
assignfile(f, 'd:\a.txt'); rewrite(f);
for i:=1 to n do
begin
        readln(fam); readln(name); readln(class);
```

```
        writeln(f,fam); writeln(f,name); writeln(f,class)
 end;
 reset(f); assignfile(s, 'd:\b.txt'); rewrite(s);
while not eof(f) do
begin
        readln(f,fam); readln(f,name); readln(f,class);
       if class<>11 then
              begin writeln(s,fam); writeln(s,name);
                    writeln(s,class)
               end
 end;
closefile(f);erase(f);
closefile(s); rename(s, 'd:\a.txt'); readln;
end.
```

14.Тип указатель. Динамические переменные

Обычно переменная хранит некоторые данные. Однако помимо обычных, существуют переменные, которые ссылаются на другие переменные. Такие переменные называются переменными типа указатель или просто указателями. Таким образом, в языке Pascal указатель – это и тип данных, и переменная. Указатель-переменная – это переменная, значением которой является адрес другой переменной или структуры данных.

Указатель Обычная переменная



Указатель является величиной, значением которой является адрес байта памяти, где хранятся какие-то данные. Указатели бывают типизированные, указывающие на данные определенного типа, и нетипизированные (типа pointer), которые могут указывать на данные произвольного типа.

Описание типизированного указателя имеет вид:

type

имя типа указателя=^ тип данных;

var *имя переменной: имя типа*;

или

var *имя переменной :^ тип данных*;

При описании указателя записывают тип данных, которые будут размещаться, начиная с байта, на который указывает указатель. Тип данных может быть любым, кроме файлового. Например:

type

Pint=^integer;

var P1,P2:Pint;

Здесь описаны тип Pint как указатель на величину типа integer и две переменные P1 и P2 типа Pint.

Для описания нетипизированного указателя существует стандартный тип **pointer**. Этот тип не связан ни с одним конкретным типом данных, однако совместим с любым типом указатель.

var a: pointer;

В начале работы программы переменная-указатель ни на что не указывает. В языке Pascal имеется предопределенная константа **nil**, которая обычно присваивается указателям, которые в данный момент ни на что не указывают. Идентификатор nil можно использовать в выражениях. Указатели одинакового типа можно сравнивать друг с другом, используя отношения = или <>.

Например: P1:=nil; **if** P1=nil **then** writeln(‘Указатель не инициализирован’);

Указателю можно присвоить значение-адрес переменной соответствующего типа: P:=@i; Для получения значения адреса переменной следует записать оператор @ перед ее именем. Переменная P типа указатель в качестве значения принимает адрес ячейки, в которой размещается значение переменной i.

Указателю можно присвоить значение другого указателя при условии, что они являются указателями на переменную одного и того же типа. Например, если переменные P1и P2 являются указателями типа integer, то в результате вы-

полнения оператора присваивания P2:=P1; переменные P1 и P2 указывают на одну и ту же переменную.

Динамические переменные

Переменные, которые описываются в разделе **var** какого-либо блока программы или подпрограммы, называются **статическими**. Транслятор после анализа этого раздела, т.е. до выполнения программы, отводит каждой переменной в зависимости от ее типа соответствующее число ячеек памяти определенной длины и закрепляет их за данной переменной на все время работы блока. Они не могут использоваться под другие нужды, даже если в процессе работы программы эта переменная становится ненужной. Статические переменные размещаются в непрерывной области памяти, имеющей небольшой объем, большая часть оперативной памяти при этом остается неиспользованной.

Динамическое распределение памяти широко используется для экономии компьютерных ресурсов при обработке данных большого объема, которые могут не уместиться в памяти, отводимой для статических переменных. При этом те переменные или объекты, которые становятся ненужными, уничтожаются, а освобожденное место используется для новых переменных или объектов. Это особенно эффективно в задачах, в которых число необходимых объектов зависит от обрабатываемых данных или от действий пользователя в процессе выполнения программы, т.е. заранее неизвестно. Для динамического распределения выделяется специальная область оперативной памяти – **heap** (куча). Динамическая память (куча) – это свободная часть оперативной памяти, которая доступна для использования в процессе работы программы. Как правило, она значительно больше памяти, отводимой для статических переменных. Динамическое размещение данных происходит непосредственно при выполнении программы. При динамическом размещении заранее неизвестны ни количество, ни тип данных. К ним нельзя обращаться по именам, как к статическим переменным. Для работы с объектами в динамически распределяемой области памяти используются указатели. При этом сами указатели должны описываться в разделе var, т.е. являются статическими переменными.

Динамической переменной называется переменная, память для которой выделяется во время работы программы.

Динамическое распределение памяти в куче может производиться двумя способами: 1) с помощью процедур **new** и **dispose**

2) с помощью процедур **getmem** и **freemem**.

При первом способе выделение памяти для динамической переменной производится процедурой: **procedure** new (имя указателя);

где имя указателя – имя переменной, являющейся типизированным указателем. Этой переменной передается адрес начала выделенной области памяти. Размер выделяемой области определяется размером памяти, необходимым для размещения того типа данных, который указан при объявлении указателя.

Например:

```
var p:^real;
…
new(p);
```

Здесь описывается переменная p, являющаяся указателем на действительное значение. Процедура new выделяет память для переменной типа real, и переменная - указатель p будет содержать адрес первого байта памяти, выделенной для этой переменной. После того, как указатель приобрел некоторое значение, отличное от nil , то есть стал указывать на конкретный байт памяти, в выделенный участок памяти может быть помещено значение соответствующего типа. Чтобы получить доступ к данным, на которые указывает типизированный указатель, надо записать:

имя переменной -указателя ^

Символ ^ после имени указателя означает, что речь идет не о значении переменной-указателя, а о значении того динамического объекта, на который указывает эта переменная. Имя ссылочной переменной с последующим символом ^ называют динамической переменной (переменной с указателем).

Например:

```
var  i: ^integer; a: ^real;
```

```
begin
    new(i); new(a);
    i^:=10; a^:=5.3/i^;
end.
```

Здесь a^ - число 0,53

a – адрес числа 0,53

Нельзя смешивать указатели и данные:

```
a:=sqr(a^)-i^;      {неверно}
a^:=sqr(a);         {неверно}
```

После того, как динамическая переменная стала ненужной, ее можно уничтожить, а освободившиеся байты вернуть в кучу. Освобождение памяти, динамически выделенной процедурой new, осуществляется процедурой dispose:

procedure dispose (имя указателя);

Применение процедуры dispose освобождает память, но не изменяет значение указателя, не делает его равным nil, хотя теперь указатель не указывает ни на что конкретное. Например, если p – указатель на динамическую переменную, память для которой выдела процедурой new(p), то процедура dispose(p) освобождает занимаемую динамической переменной память (байты, содержавшие p^, будут считаться свободными). При этом значение указателя p не изменяется, однако его использование приведет к порче значений других динамических переменных. Поэтому рекомендуется освободившемуся указателю присваивать значение nil: dispose(p); p:= nil;

Пример:

```
var p1, p2, p3:^integer;{указатели на переменные типа integer}
begin
    //выделяется память для динамических переменных типа integer
    new(p1); new(p2);  new(p3);
    //динамические переменные p1^, p2^ и p3^ получают значения
    p1^:=5; p2^:=3;  p3^:=p1^+p2^;
    writeln('сумма чисел равна ',p3^);
```

```
    //освобождается память, занимаемая динамическими переменными
    dispose(p1);  dispose(p2);  dispose(p3);
end.
```

Второй способ динамического выделения памяти связан с применением процедуры Getmem для выделения памяти и Freemem для ее освобождения:

procedure Getmem(имя указателя, объем памяти в байтах);

procedure Freemem(имя указателя, объем памяти в байтах);

В этих процедурах могут использоваться не только типизированные, но и нетипизированные указатели.

Например: **var** a:pointer;

begin Getmem(a,4); … Freemem(a,4); **end.**

Если используется типизированный указатель, то объем необходимой памяти лучше всего определить функцией sizeof, так как размеры памяти, отводимой под тот или иной тип данных, могут изменяться в различных версиях компилятора. Функция sizeof(x) вычисляет длину в байтах для x.

```
Getmem(p,sizeof(real));
Freemem(p,sizeof(real));
```

Чтобы получить доступ к данным, на которые указывает нетипизированный указатель pointer, нельзя писать: имя переменной-указателя ^. Надо сначала привести его к другому типу указателя. Явное приведение типа переменной можно проводить для любых типов, имеющих одинаковый размер, с помощью конструкции: **идентификатор типа (выражение)**.

Например:

```
type Pint=^integer;
var p1,p2:Pint;  p:pointer; i:integer;
```

выражение Pint(p) приведет тип указателя p к объявленному выше типу Pint, после чего его можно записать:

```
Pint(p):=p1;  i:=Pint(p)^+p2^;
```

Пример:

```
type pint = ^integer;
```

```
var p1,p2:pint;p:pointer;
begin
    new(p1); new(p2); getmem(p,sizeof(integer));
    p1^:=5; p2^:=10;              {можно записать p:=p1;p^ уже нельзя}
    pint(p)^:=p1^;
    writeln(pint(p)^+p2^);        {результат 15}
    dispose(p1); dispose(p2); freemem(p,sizeof(integer));
end.
```

Динамическая память используется для организации динамических структур данных. Простейшими динамическими структурами являются стеки, очереди и списки. Элементы этих структур программируются в виде величин типа запись, причем одно из полей является указателем на следующий (предыдущий) элемент структуры. Это позволяет размещать данные в памяти независимо друг от друга, так что удаление элемента из структуры или добавление нового элемента не приводит к необходимости перемещать в памяти другие элементы, как это происходит, например, при использовании массивов.

ГЛАВА 3. СТРУКТУРИРОВАНИЕ ПРОГРАММЫ

15. Описание и использование процедур и функций

Подпрограмма – это часть программы, оформленная в виде отдельной синтаксической конструкции и снабженная именем. Структура подпрограммы аналогична структуре программы. Использование подпрограмм позволяет упростить процесс программирования и логику программы за счет применения методики так называемого нисходящего проектирования, т.е. разбиения исходной задачи на более простые подзадачи. Каждую подзадачу можно оформить в виде подпрограммы. Подпрограмма должна быть описана до того, как она будет использована в вызывающей программе или другой подпрограмме. В языке Pascal имеется два вида подпрограмм - процедуры и функции.

Процедуры

Подпрограмма-процедура предназначена для выполнения некоторой логически законченной последовательности действий. Описание процедуры состоит из заголовка, раздела описаний и раздела операторов. Завершается описание процедуры точкой с запятой. Заголовок процедуры имеет вид:

procedure имя процедуры (список формальных параметров);

В качестве формальных параметров, как правило, указываются входные и выходные параметры процедуры. Список формальных параметров может быть разбит на группы, разделённые точками с запятыми. В группу включаются однотипные параметры. Для каждого формального параметра указывается его имя и тип. Тип формального параметра - идентификатор. Список формальных параметров может отсутствовать.

Примеры:

procedure summa(a:integer; **var** b,c:real); {a,b,c формальные параметры}

procedure t;

procedure a(**var** x:d); {d – имя (идентификатор) нестандартного типа, описанного ранее в разделе **type**}

Оператор вызова процедуры имеет вид:

имя процедуры (список фактических параметров);

В операторе вызова процедуры фактические параметры отделяются друг от друга запятыми. Количество, типы и порядок записи фактических параметров должны соответствовать количеству, типам и порядку записи формальных параметров.

Примеры операторов вызова процедур:

t; {нет списка фактических параметров}

summa(x,y,z); {x,y,z –фактические параметры; x: integer; y, z: real}

a(z); {z – фактический параметр, z:d}

При вызове подпрограммы фактические параметры подставляются вместо соответствующих формальных параметров, затем над ними осуществляются нужные действия.

Функции

Подпрограмма-функция предназначена для вычисления некоторого значения. Описание функции содержит два отличия от описания процедуры.

1) Заголовок функции имеет вид:

function имя функции (список формальных параметров): тип функции;

Тип функции – это тип возвращаемого результата.

2) В разделе операторов имени функции должно быть присвоено значение возвращаемого результата.

Пример: **function** f (x:integer):real;

begin f :=sin(x)+sqr(x) **end;**

В Object Pascal функции могут возвращать значения любого типа, кроме типа объекта старой модели (object) и файловых типов.

Для вызова функции из программы или другой подпрограммы следует указать имя функции со списком фактических параметров в выражении, где необходимо использовать значение функции.

Например: y :=f(a); {a - фактический параметр}

writeln(f(2+b)); {2+b – фактический параметр}

В Object Pascal для каждой функции автоматически определена локальная переменная Result того же типа, что и возвращаемое функцией значение. Эту переменную можно использовать в промежуточных вычислениях. Значение Result при вызове функции не определено. Последнее присвоенное ей значение и вернется как значение функции.

Примеры:

function f (x:integer):real;

begin result :=sin(x)+sqr(x) **end;**

Локальная переменная Result может использоваться и в правой части оператора присваивания, не вызывая рекурсивного вызова функции.

function f : integer;

begin result:=1; result:= result +1 **end;**

Функция f будет возвращать значение 2. А выполнение функции g:

function g: integer;

begin g:=1; g:= g+1 **end;**

приведет к зацикливанию программы.

В Delphi функция может быть вызвана как процедура. При этом возвращаемое значение просто теряется.

Формальные и фактические параметры

Формальные параметры можно разбить на несколько категорий. Рассмотрим две из них:

- параметры-значения;
- параметры-переменные.

Параметры-значения передаются основной программой в подпрограмму в виде их копий и, следовательно, подпрограмма не может изменить собственный параметр программы. Другими словами, формальный параметр-значение при вызове подпрограммы получает свое значение путем копирования соответствующего ему фактического параметра, и при изменении такого формального параметра соответствующий ему фактический параметр не меняется.

Параметр-значение указывается в заголовке подпрограммы своим именем и через двоеточие – типом. Тип параметра-значения может быть любым за исключением файлового. В качестве фактического параметра на месте параметра-значения при вызове подпрограммы может выступать любое выражение совместимого для присваивания типа.

При передаче параметров-переменных в подпрограмму передаются сами переменные, а не их копии (фактически передаются их адреса в порядке, объявленном в заголовке подпрограммы). Следовательно, подпрограмма имеет доступ к этим параметрам и может их изменять. Параметр-переменная используется в том случае, когда значение должно передаваться из подпрограммы вызывающей программе. При вызове подпрограммы на месте параметра-переменной в качестве фактического параметра должна использоваться переменная идентичного типа.

Параметр-переменная указывается в заголовке подпрограммы аналогично параметру-значению, но только перед именем параметра записывается зарезервированное слово var. Действие var распространяется до ближайшей точки с запятой, т.е. в пределах одной группы.

```
procedure summa(a:integer;b:real; var c:real);
begin   c:=a+b
end;     {a, b- параметры-значения; c – параметр-переменная.}
```

Локальные и глобальные параметры

Параметры, объявленные внутри подпрограммы и доступные только ей самой, называются локальными.

В разделе описаний подпрограммы могут встретиться описания подпрограмм низшего уровня, а в них – описания других подпрограмм и т.д. При входе в подпрограмму низшего уровня становятся доступными не только объявленные в ней имена, но и сохраняется доступ ко всем именам верхнего уровня. Подпрограмме доступны только те объекты верхнего уровня, которые описаны до описания данной подпрограммы. Эти объекты называются глобальными по отношению к этой подпрограмме.

Параметры, объявленные до описания данной подпрограммы, доступны этой подпрограмме и являются глобальными параметрами. Одноименные глобальные и локальные параметры – это разные параметры. Любое использование такого имени в подпрограмме трактуется как обращение к локальному параметру, то есть глобальный параметр в этом случае недоступен.

Таким образом, обмен информацией между основной программой и подпрограммой может осуществляться не только с помощью формальных и фактических параметров, но и глобальных параметров.

Примеры:

1)
```
program Project1;
{$apptype console}
uses  SysUtils;
```

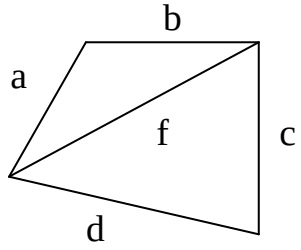
```
var x,y,z:integer;
procedure newval;
var x,y:integer;  {x и y- локальные параметры}
begin  x:=1; y:=1 end; {конец процедуры}
begin x:=0; y:=0;     newval; z:=x+y; writeln(z); readln
end.
```

Результат z=0, так как переменные x и y, используемые в основной программе, остаются равными нулю. Для того, чтобы результат z стал равным 2, нужно параметры x и y использовать как глобальные, для этого в теле процедуры newval следует убрать описание переменных x и y:

```
2) program Project2;
{$apptype console}
uses  SysUtils;
var x,y,z:integer;
procedure newval;
begin x:=1; y:=1 end;
begin x:=0; y:=0;   newval;          z:=x+y;        writeln(z); readln
end.
```

Результат z=2, так как x и y – глобальные параметры.

Задача 15.1. Даны длины сторон и диагонали выпуклого четырехугольника. Вычислить его площадь.



Введем обозначения:

a,b,c,d,f – длины сторон и диагонали четырехугольника;

s – площадь четырехугольника.

Входные данные: a,b,c,d,f.

Выходные данные: s.

Четырехугольник разбит диагональю на два треугольника. Вычислив площади этих треугольников (s1, s2), можно определить искомую площадь четырехугольника (s). Используем подпрограмму для вычисления площади треугольника. Пусть x,y,z – стороны треугольника, s – его площадь. Площадь треугольника можно вычислить по формуле Герона $s = \sqrt{p(p-x)(p-y)(p-z)}$, где p - полупериметр треугольника. В процедуре square будет четыре формальных параметра: x,y,z – параметры-значения (входные величины этой подпрограммы); s – параметр-переменная (выходная величина, результат этой подрограммы). Переменная p – локальный параметр этой процедуры.

```
program Project15_1;
{$apptype console}
uses  SysUtils;
var   a,b,c,d,f,s,s1,s2:real ;
procedure square (x,y,z:real; var s:real);
var p:real;
begin  p:=(x+y+z)/2; s:=sqrt(p*(p-x)*(p-y)*(p-z)) end;
begin  writeln (‘vvod a,b,c,d,f: ’);  readln(a,b,c,d,f);
       square (a,b,f,s1);      {оператор вызова процедуры,
                                a,b,f,s1- фактические параметры}
       square (c,d,f,s2);      {оператор вызова процедуры,
                                c,d,f,s2 - фактические параметры }
       s:=s1+s2;  writeln(‘s=’,s:8:2); readln
 end.
```

Задача 15.2. Даны две строки символов. В первой строке подсчитать количество букв “a”, во второй строке - букв “o”.

Введем обозначения:

s1,s2 – заданные строки;

k1– количество букв “a” в строке s1;

k2 – количество букв “o” в строке s2.

Входные данные: s1,s2.

Выходные данные: k1,k2.

В программе вводятся заданные строки, дважды вызывается подпрограмма, осуществляющая обработку строки, и выводится результат.

В подпрограмме определяется количество k букв, являющихся значением переменной c, в строке символов s. Первоначально количество букв полагается равным нулю. Просматриваются все символы заданной строки - с первого символа до последнего (его номер равен длине строки). Если i-й символ строки s[i] совпадает с разыскиваемой буквой (c), то количество найденных букв увеличивается на единицу, иначе - остается без изменения. В подпрограмме-процедуре три формальных параметра s, c, k: s, c – параметры-значения (исходные данные этой процедуры); k- параметр-переменная (результат процедуры). При вызове процедуры вместо формальных параметров подставляются соответствующие фактические параметры.

```
program Project15_2a;
{$apptype console}
uses  SysUtils;
var  s1,s2:string; k1,k2:integer;
procedure kol (s:string; c:char; var k:integer);
var i:integer;    { i – локальный параметр}
begin   k:=0;
      for i:=1 to length(s) do
          if s[i]= c then k:=k+1
end;
begin
      write(‘first string: ’); readln(s1);
      kol (s1, ‘a’, k1);          {s1,’a’,k1 - фактические параметры}
      writeln(‘k1=’,k1);
      write(‘second string: ’); readln(s2);
```

```
    kol (s2, ‘o’, k2);          {s2,’o’,k2 - фактические параметры}
    writeln(‘k2=’,k2); readln
end.
```

В подпрограмме-функции два формальных параметра-значения s и c. Количество разыскиваемых букв подсчитывает переменная k, ее значение и будет значением функции kol. Программа решения поставленной задачи с использованием процедуры-функции имеет вид:

```
program Project15_2b;
{$apptype console}
uses  SysUtils;
var  s1,s2:string; k1,k2:integer;
function kol (s:string; c:char):integer;   {s, c - формальные параметры}
var i,k:integer;                           { i, k - локальные параметры}
begin   k:=0;
        for i:=1 to length(s) do
                if s[i]= c then k:=k+1;
        kol:=k                {имени функции kol присваивается результат}
end;
begin    write(‘first string:’); readln(s1);
        k1:=kol (s1, ‘a’);    {s1,’a’ - фактические параметры}
        writeln(‘k1=’,k1);
        write(‘second string:’); readln(s2);
        k2:=kol (s2, ‘o’);    {s2,’o’ - фактические параметры}
        writeln(‘k2=’,k2); readln
end.
```

Консольное приложение создается в Windows, а выполняется как программа DOS. В DOS используется кодировка ASCII, а в Windows - ANSI, буквы русского алфавита в этих системах кодирования имеют разные коды. Поэтому в консольном приложении сообщения, которые выводятся на экран, могут содержать буквы только латинского алфавита, что не всегда удобно.

Задача 15.3. Создать функцию перекодировки ANSI строки в строку ASCII для вывода сообщений на русском языке в консольном приложении. В ANSI русские буквы кодируются числами от 192 до 255, в ASCII – от 128 до 175 (А…Я,а..п) и от 224 до 239 (р..я).

```
program Project15_3;
{$apptype console}
uses  SysUtils;
function  rus (s:string):string;
var i:integer;  { i – номер символа в строке s }
begin
    for i:=1 to length(s) do
        case s[i] of
        ‘A’..’ п’ : s[i] := chr(ord(s[i])-64);
        ‘р’..’ я’ : s[i] := chr(ord(s[i])-16);
        end;
        rus :=s
end;
begin
  writeln(rus(‘Вводится любое сообщение’));
  readln;
end.
```

16. Рекурсивные процедуры и функции

Подпрограмма, вызывающая саму себя, называется рекурсивной подпрограммой. При каждом новом обращении к подпрограмме параметры, которые она использует, размещаются в организованной особым образом области памяти, называемой программным стеком, причем параметры предыдущего обращения также сохраняются. Рекурсия часто бывает менее эффективной по времени и может вызвать переполнение стека.

Задача 16.1. Вычислить факториал (n!) данного натурального числа n.

По определению факториала $n!=1 \cdot 2 \cdot 3 \cdot 4 \cdot \ldots \cdot (n-1) \cdot n$,

a (n-1)!= 1 · 2 · 3 · 4· …· (n-1).

Следовательно, n!= n · (n-1)!

Таким образом, факториал числа n равен произведению числа n на факториал числа (n-1). Поэтому вычисление факториала n можно реализовать как функцию, в разделе операторов которой будет вызов функции вычисления факториала числа (n-1), т.е. функция вызывает саму себя. Для завершения рекурсивного процесса можно использовать определение: 1!=1.

Программа имеет вид:

```
program Project16_1;
{$apptype console}
uses  SysUtils;
var  n:integer;
function fact (n:integer):longint;
begin
        if n=1 then fact:=1
                else fact:=fact(n-1)*n
end;
begin
        write('n='); readln(n);
        writeln(fact(n)); readln
end.
```

Исполнение программы. Если ввести n=4, то

fact(4)=fact(3)*4 {функция fact вызывает саму себя}

fact(3)=fact(2)*3

fact(2)=fact(1)*2

fact(1)=**1** {рекурсивный процесс закончен}

Далее последовательно вычисляются:

fact(2)=fact(1)*2 =**1***2=**2**

fact(3)=fact(2)*3 = **2***3=**6**

fact(4)=fact(3)*4 =**6***4=**24**

17. Процедурные типы

В языке программирования Pascal имена процедур и функций можно использовать в качестве фактических параметров. С этой целью вводятся процедурные типы, которые позволяют указать, какой вид подпрограмм можно использовать в качестве параметра и с какими формальными параметрами должны быть эти подпрограммы.

Описание процедурного типа:

type

имя типа = ***procedure(****список формальных параметров);*

имя типа = ***function****(список формальных параметров):тип функции;*

var *имя переменной*: *имя типа*;

Примеры:

```
type      proc = procedure(var x,y:integer);
{процедура с двумя параметрами-переменными целого типа}
    func = function(x:real):real;
{функция вещественного типа с одним параметром-значением}
  var p:proc; f:func;
```

Переменным процедурных типов можно присваивать в качестве значений имена соответствующих подпрограмм.

имя переменной: = имя подпрограммы;

Процедурная переменная и имя подпрограммы должны быть совместимы для присваивания (т.е. должно быть одинаковое число формальных параметров, совпадающих по типам, функции, кроме того, должны иметь идентичный тип).

После такого присваивания имя переменной становится синонимом имени подпрограммы. Переменная процедурного типа используется при вызове подпрограммы: вместо имени подпрограммы указывается имя процедурной переменной: *имя переменной(список фактических параметров)* .

Передаваемые подпрограммы не должны объявляться внутри других процедур и функций.

Пример:

```
type
    proc = procedure(var x,y:integer);
    func = function(a:real):real;
var p:proc; f:func; x,y:integer;
procedure swap(var x,y:integer);      {x↔ y }
 var z:integer;
 begin
     z:=x; x:=y; y:=z
 end;
function tan (a:real):real;                  {tg(a)}
     begin
         tan:=sin(a)/ cos(a)
     end;
begin
    readln(x,y); p:=swap; f:=tan;
    p(x,y); writeln(x,y);
    writeln(f(pi/4)); readln
end.
```

Процедурные переменные позволяют использовать одну и ту же подпрограмму для различной обработки данных.

Задача 17.1. Вывести таблицу сложения и умножения двух целых чисел в диапазоне от 1 до 10.

```
program Project17_1;
{$apptype console}
uses  SysUtils;
type  func=function(x,y:integer):integer;
function add (x,y:integer):integer;
begin
      add:=x+y                {сложение двух целых чисел}
```

```
end;
function mult (x,y:integer):integer;
begin
      mult:=x*y              {умножение двух целых чисел}
end;
procedure table(oper:func);       { oper – переменная процедурного типа}
var i,j:integer;
begin
   for i:=1 to 10 do
      begin
            for j:=1 to 10 do write(oper(i,j):5);       {вывод таблицы}
            writeln
      end;
end;
begin
     table(add); { add – значение процедурной переменной }writeln;
     table(mult); { mult – значение процедурной переменной }readln
end.
```

Здесь одна и таже процедура table позволяет получить как таблицу сложения (с параметром add), так и таблицу умножения (с параметром mult).

18. Модуль пользователя

Для того, чтобы часто используемые типы данных, константы, переменные, процедуры и функции не описывать заново в каждой программе, их описания, оформленные соответсвующим образом, можно сохранить в специальном файле, который называется модулем пользователя, и при необходимости подключать этот файл к программе. Модуль состоит из следующих частей:

- заголовок модуля;
- интерфейсная часть модуля;
- исполняемая часть модуля;

- инициализирующая часть модуля.

Заголовок модуля имеет вид:

unit имя модуля; например: **unit** MyModul;

Модуль должен быть помещен в файл, имя которого совпадает с именем модуля, а расширение - .pas.

Через интерфейсную часть осуществляется взаимодействие основной программы с модулем (или модуля с модулем). Интерфейсная часть является обязательной и начинается со слова **interface**. Далее после слова uses могут быть указаны имена других модулей, используемых данным модулем. После этого записываются описания констант, типов, переменных, процедур и функций, которые могут быть использованы основной программой (модулем) при вызове этого модуля. При описании процедур и функций указываются лишь их заголовки, сами подпрограммы приводятся в исполняемой части модуля.

Исполняемая часть модуля также является обязательной и включает описание всех подпрограмм модуля. Она начинается со слова **implementation**. Затем после слова uses указываются имена модулей, используемых подпрограммами данной исполняемой части. Если какой-то модуль уже указан в интерфейсной части, то в исполняемой части его повторять не следует. Далее могут быть: описание локальных меток, констант, типов, переменных, затем описываются подпрограммы модуля. При описании подпрограмм в исполняемой части допустимо использовать сокращенные заголовки.

В некоторых случаях перед обращением к модулю следует провести его инициализацию. Например, может потребоваться установить связь файловой переменной с внешним файлом на диске, инициализировать (т.е. присвоить определенные начальные значения) какие-то переменные, описанные в модуле, и т.п. Эта часть начинается словом begin, после которого следуют исполняемые операторы, а заканчивается словом end, после которого следует точка.

Например: **begin** assignfile(f,’file.dat’); **end.**

Операторы инициализирующей части выполняются один раз в момент запуска программы, к которой подключен модуль.

Если инициализация модуля не нужна, то в инициализирующей части записывается лишь слово **end**, после которого следует точка.

Для использования констант, типов, переменных, подпрограмм, описанных в интерфейсе модуля, в некоторой программе (или другом модуле) следует записать слово uses и имя модуля. Если в модуле и использующей его программе встречается одно и то же имя для обозначения разных элементов, то при обращении к элементу модуля указывается имя модуля, а затем через точку – имя элемента. Например: y:=x+ MyModul.x; {x – переменная программы;

MyModul.x – переменная модуля}

Задача 18.1. Вычислить сумму двух комплексных чисел.

Введем обозначения:

x,y – заданные комплексные числа;

z – сумма комплексных чисел;

re – вещественная часть комплексного числа;

im – мнимая часть комплексного числа.

Входные данные: x,y.

Выходные данные: z .

Для переменных x, y, и z, которые должны хранить комплексные числа, опишем тип Complex. Значениями переменной типа Complex являются пары вещественных чисел - вещественная и мнимая части комплексного числа. В процедуре Add выполняется сложение двух комплексных чисел x и y, результатом является комплексное число z (суммируются вещественные и мнимые части). Описание типа Complex и процедуры Add поместим в модуль Comp.

Итак, модуль, выполняющий сложение комплексных чисел имеет вид:

```
unit Comp;
interface
type Complex=record
        re,im:real;
   end;
procedure Add(x,y:Complex; var z:Complex);
```

```
implementation
procedure Add;
begin z.re:=x.re+y.re;   z.im:=x.im+y.im end;
end.
```

Сохраним полученный модуль на диске под именем Comp.pas.

В следующем примере к программе My подключены модули SysUtils и Comp. Модуль SysUtils является стандартным модулем Delphi. Модуль Comp позволяет получить доступ к типу Complex и процедуре Add в программе My.

Использование модуля Comp:

```
program My;
{$apptype console}
uses  SysUtils, Comp;
var x,y,z: Complex;
begin x.re:=5; x.im:=6; y.re:=10; y.im:=3;
        Add (x,y,z);
        writeln(z.re,' ',z.im); readln
end.
```

Таким образом, в этой программе тип Complex и процедура Add используются без предварительного описания, как стандартные параметры. Такую возможность предоставляет модуль Comp, заранее составленный, сохраненный на диске и подключенный к программе в строке **uses**.

Литература

1. Аганин А.А., Халитова З.Р., Хисматуллина Н.А. Изучение основ языка программирования Object Pascal. – Казань: ТГГПУ, 2006. - 80 с.

2. Архангельский А.Я. Язык Pascal и основы программирования в Delphi. М.:Бином-Пресс, 2008. - 496 с.

3. Культин Н.Б. Delphi 6. Программирование на Object Pascal. СПб: БХВ - Петербург, 2002.

4. Немнюгин С.А. Turbo Pascal. Программирование на языке высокого уровня. – СПб.:Питер, 2008.

5. Фаронов В. Delphi 6: учебный курс. – СПб.:Питер, 2002. - 512 с.