ЯЗЫК ПРОГРАММИРОВАНИЯ PASCAL

КРАТКИЙ КУРС ЛЕКЦИЙ

СОДЕРЖАНИЕ

Лекция 1. **Общие сведения о языке ПАСКАЛЬ**

Литература

1. Абрамов С.А., Гнездилова Г.Г. Задачи по программированию. М.: Наука, 1988.

2. Быкадоров Ю.А., Кузнецов А.Т. Информатика и вычислительная техника 10–11. – Мн.: Народная асвета,– 1977.

3. Бородич Ю.С., Вальвачев А.Н., Кузьмич А.И. Паскаль для персональных компьютеров. Мн.: Вышэйшая школа, 1991.

4. Гудман С., Хидетниеми С. Введение в разработку и анализ алгоритмов. М.: Мир, 1981.

5. Зима В.С., Абрамов С.А. Начала программирования на языке Паскаль.

6. Офицеров Д.В., Старых В.А. Программирование в интегрированной среде Турбо-Паскаль. Справочное пособие. Мн.: Беларусь, 1992.

7. Офицеров Д.В., Долгий А.Б., Старых В.А. Программирование на персональных компьютерах . Практикум. Мн.: Вышэйшая школа, 1993.

8. Пильщиков В.Н. Сборник упражнений по языку Паскаль. М.: Наука, 1989.

Литература к лабораторным занятиям

1. Вабішчэвіч С.В. Уводзіны ў праграмаванне на мове Паскаль: Лабараторны практыкум.– Мн.:БДПУ імя М.Танка, 2000.

2. Дадатковыя магчымасці мовы Turbo Pascal. Лабараторны практыкум //Аўт.-склад. Л.М. Краўчанка.– Мн.: БДПУ імя М.Танка, 2000.

Язык Паскаль получил свое название в честь великого французского ученого, физика-математика Блеза Паскаля, который в 1642 г. изобрел счетную машину для арифметических операций – Паскалево колесо. История создания языка Паскаль начинается с 1965 года, когда международная федерация по обработке информации IFIP предложила нескольким специалистам в области информатики принять участие в разработке нового языка программирования – приемника АЛГОЛА-60. Среди них был швейцарский ученый, работавший доцентом информатики Стенфордского университета Николаус Вирт. В конце 1968 года Вирт и сотоварищи из швейцарского федерального института технологии в Цюрихе разработали первую версию Паскаля, а спустя 2 года – 1-й вариант компилятора. В 1971 году Вирт выпустил описание своего языка. Создавая Паскаль, Вирт преследовал 2 цели:

во-первых, разработать язык, пригодный для обучения программированию как систематической дисциплине;

во-вторых, реализация языка должна быть эффективной и надежной на существующих вычислительных машинах.

Одним из достоинств языка Паскаль является то, что он воплотил в себе идею структурного программирования, суть которой заключается в том, что с помощью нескольких конструкций можно выразить в принципе любые алгоритмы: линейные, ветвление, циклические конструкции.

При создании и совершенствовании языка Вирт ввел много новшеств, в частности, изобрел синтаксические диаграммы, с помощью которых удобно представлять конструкции языка, первый ввел в алфавит квадратные скобки, высказал идею решения проблемы переносимости программ в виде Пи-системы, которая заключается в том, что написанная на Паскале программа транслируется в Пи-код, в машинный язык некоторой идеальной машины, а затем интерпретируется на реальных машинных языках. Он выявил и ряд недостатков: отсутствие операции возведения в степень, понятие отдельно транслируемого модуля, затрудняющее тем самым создание больших программ и др.

Кроме авторской версии, стали появляться различные его расширения и диалекты. Например: УКСД Паскаль, Паскаль-80, ЭППЛ-Паскаль, ТУРБО-Паскаль, Квик-Паскаль. Особую популярность на микроЭВМ и ПЭВМ в настоящее время получило семейство Паскаль-систем, названное ТУРБО-Паскаль и разработанное BORLAND. Данное семейство, работающее с CP/M, MSXDOS, имеет высокую производительность, т. к. используется ускоренная однопроходная процедура компиляции. Последняя версия ТУРБО-Паскаль 7.0 Паскаль стала прародителем более поздних языков программирования.

Через 10 лет после Паскаля Вирт создал язык МОДУЛА-2, в котором особое внимание уделяется построению программ как набора независимых модулей. На Паскаль опирались все 4 языка , принятые в начале 70-х годов Министерством обороны США для разработки своего универсального высокоуровневого языка, который в дальнейшем получил название АДА.

Алфавит языка Паскаль состоит из букв русского и латинского алфавита, арабских цифр, знаков операций (+,-,*,/,=,<,>,:=), ограничителей (. , : , ; , ',[],(),). Действительные числа изображаются в естественной и полулогарифмической форме (например, 2*E*+5 это 200000). Допустимый диапазон изменения целых и вещественных чисел зависит от конкретной реализации языка.

Простые типы данных. В языке Паскаль типы данных разделяются на простые и структурные.

Простые типы являются базовыми. На их основе строятся более сложные структурные типы данных. Т. к. данные простого типа (константы, переменные, функции, выражения) имеют одно значение, то простые типы также называют скалярными. Скалярный тип определяет упорядоченное множество значений переменных, констант, функций и выражений, принадлежащих к данному типу, форму представления в машине значений этих величин и операций, которые могут выполняться над ними. Скалярный тип может быть предопределенным (стандартным) или задаваться пользователем с помощью перечисления его возможных значений в разделе описания типов. Поэтому в последнем случае тип называют перечисляемым.

Стандартные типы данных. К ним относятся:

1) вещественный (*real*); 2) целочисленный (*integer*); 3) логический (*boolean*); 4) символьный (*char*).

Примеры	Обозначения	Границы	Требуется памяти (байт)
целый	*byte*	0..255	1
	word	0..65535	2
	integer	-32768..32767	2
	shortint	-128..127	1
	longint	-2147483648..2147483647	4
вещественный	*real*	2.9*E*-39..1.7*E*38	6
символьный	*char*	кодовая таблица ПЭВМ	1
логический(булевый)	*Boolean*	true, false	1

Диапазон допустимых значений *real* от *E*-38 до *E*+38 с мантиссой, занимающей 11 двоичных разрядов. Числа этого типа занимают 6 байт памяти. Чтобы некоторая

переменная в программе относилась к вещественному типу, ее имя в разделе описания переменных должно быть описано как *real*:

var c, sk:real;

Тип выражения опередеяется типом входящих в него операндов и видом операций, проводимых над ними. Результат операций +, -, * будет действительным числом (вещественным), если хотя бы один операнд вещественного типа. Результат операции деления – **всегда действительное число**, даже если оба операнда целого типа.

Значениями целочисленного или целого типа являются элементы подмножества целых чисел, допустимых в конкретной ЭВМ. Диапазон допустимых целых чисел в десятичной записи от -32768 до 32767. Определена стандартная константа *MAXINT=32767*. В разделе описания переменных указываются имена:

var c,sp,q:integer;

Арифметическое выражение будет давать целый результат, если все входящие в него операнды относятся к целому типу и к ним применены операции -,+, *, а также *div* или *mod*. Как правило, данные целого типа редко используются в вычислениях. В программировании они применяются для обозначения индексов в массивах, организации счетчиков. Подмножество целых чисел от 0 до 255 обозначают специальным словом *byte*. Кроме этого используется специальное обозначение **отрезочного** типа – две точки. Например, обозначение 5..16 обозначает что переменные, описанные таким образом – это целые числа от 5 до 16.

Для вещественных и целых типов выделяются другие подмножества, которые обозначаются специальными служебными словами. С ними можно ознакомиться в справочниках.

Логические (булевые) значения. Обозначаются стандартными именами *true* и *false*. Установлено, что *false* меньше *true*. Переменная этого типа занимает 1 байт памяти. Логическая переменная – это переменная, принимающая одно из значений, и в разделе описания переменных она должна быть описана так:

var c,sp, q1:boolean;

Значение *true* и *false* получают в результате выполнения операций сравнения <,>,<=,>=,<>,=. Операнды этих операций могут быть вещественного, целого типа. Следует помнить, что к операндам вещественного типа очень осторожно следует применять операцию =, т. к. условие может не выполняться за счет неточного представления действительных чисел в памяти ЭВМ и неизбежных ошибок округления при вычислении выражений вещественного типа. В некоторых случаях необходимо применять вместо записи *A1=A2* запись *ABS(A1-A2)<E*, где *E* – некоторая величина, характеризующая допустимую погрешность округления.

Помимо операций отношения, существуют 3 логические операции, применяемые только к операндам булевого типа. Это *not*, *and*, *or*.

Операция *not* является одноместной. Ее результат – *true*, если значение операнда – *false*. Операции *and* и *or* двухместные.

not		*and*			*or*		
арг	рез	арг	арг	рез	арг	арг	рез
true	*false*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*true*	*false*	*false*	*true*	*false*	*true*
		false	*true*	*false*	*false*	*true*	*true*
		false	*false*	*false*	*false*	*false*	*false*

Логические операции и операции отношения часто встречаются в одном выражении, причем отношения, стоящие слева и справа от логической операции, имеют более высокий приоритет и выполняются в следующем порядке: в первую очередь *not*,

затем *and*, *or* в последнюю очередь. С помощью скобок порядок выполнения может быть изменен.

Например:

1 6 4 2 5 3

(-3>=5) OR NOT (7<9) AND (0<=3). Имеет значение *false*.

Символьный тип задает конечное и упорядоченное множество символов. Значение символьной константы заключается в апострофы. Например: *'к','1','+'*;

var c,k:char;

Все переменные этого типа упорядочены по кодам.

Перечисляемые типы данных. В специальном разделе – разделе описания типов – программист может сам определять некоторые типы данных, перечислить те значения, которые будут принимать переменные этого типа. Переменные и константы скалярного типа, задаваемые перечислением, не могут быть параметрами операторов ввода и вывода в языке Паскаль. Они используются для управления логикой программы в операторах цикла, условных операторах. Они в какой-то мере играют роль комментариев.

Каждое значение типа задается указанием, обозначающим это значение идентификатора. Например:

type day = (sat, sun, mon, tue, wed, thu, fri);

Скалярный тип *day* включает в себя перечисленные значения *day*, *const*. Переменные должны быть описаны в разделе описания переменных:

var paday , st: day;

Переменные *paday* и *st* объявлены как переменные типа *day*, тогда можно записать следующие операции присваивания:

paday := mon; st := tue;

Переменные типа day при выполнении программы могут принимать только одно из 7 указанных значений. Объект, указанный в списке перечисления может присутствовать не более чем в одном описании. Имена объектов, указанных в описании перечисляемого типа являются константами этого типа. Для перечисляемого типа данных существенен порядок указанных объектов. К данным перечисляемого типа применимы операции отношения. Порядковый номер объекта вычисляется с помощью функции *ord*. Примеры:

sun<paday; ord(sat)=0; ord(tue)=3;

Арифметические операции. Функции. Выражения. Над числовыми величинами могут выполняться операции сложения, вычитания, умножения, деления, целочисленного деления (*div*), вычисление остатка от целочисленного деления *mod*. Если операции +,-, * выполняются над целыми числами, результат получается целочисленный. При выполнении операции *div* и *mod* над вещественными числами исходные данные сначала приводятся к целому типу. Результат выполнения этих операций целочисленный. Приоритет операций в порядке убывания:

1– *, /
2– *div*
3– *mod*
4– +, -

Например: *3 div 4+5=5*; *7 mod 2+3=4.*

Арифметические операции

Обозначение	Действие	Тип аргументов	Тип результата
а+в	сложение *а* и *в*	только целый	целый
		только вещественный	вещественный
		один целый, другой вещественный	вещественный
а–в	вычитание *в* из *а*	только целый	целый
		только вещественный	вещественный
		один целый, другой вещественный	вещественный
*а*в*	умножение *а* на *в*	только целый	целый
		только вещественный	вещественный
		один целый, другой вещественный	вещественный
а/в	деление *а* на *в*	только целый	вещественный
		только вещественный	вещественный
		один целый, другой вещественный	вещественный
а mod в	остаток от деления *а* на *в*	только целый	целый
а div в	целочисленное деление *а* на *в*	только целый	целый

Основные математические функции

Обозначение	Действие	Тип аргумента *x*	Тип результата *y*
abs(x)	определяет модуль величины *x*	целый	целый
		вещественный	вещественный
sin(x)	определяет синус *x*	целый	только вещественный
		вещественный	
cos(x)	определяет косинус *x*	целый	только вещественный
		вещественный	
arctan(x)	определяет арктангенс *x*	целый	только вещественный
		вещественный	
exp(x)	*e* возводит в степень *x*	целый	только вещественный
		вещественный	
ln(x)	определяет натуральный логарифм *x* *(x > 0)*	целый	только вещественный
		вещественный	
sqr(x)	возводит *x* в квадрат	целый	целый
		вещественный	вещественный
sqrt(x)	определяет корень из *x* $(x \geq 0)$	целый	только вещественный
		вещественный	

frac(x)	определяет дробную часть *x*	целый вещественный	только вещественный
int(x)	определяет целую часть *x*	целый вещественный	только вещественный
trunc(x)	определяет целую часть *x*	целый вещественный	только целый
round(x)	находит целое, ближайшее к *x*	целый вещественный	только целый
odd(x)	проверяет на нечетность *x*	целый	логический
random(x)	определяет случайное целое значение	целый	целый в пределах $0 \le y < x$
random	определяет случайное вещественное значение	без аргумента	вещественный в пределах $0 \le y < 1$
pi	определяет число π	без аргумента	вещественный

Результат выполнения функции *int(X)*, *trunc(X)*, *frac(X)* для $X < 0$ отличается от привычного значения целой и дробной части. При выполнении *int(x)* и *trunc(x)* цифры после запятой отбрасываются. Например: *int(-3.72)=-3.0 trunc(-3.72)=-3.*

При вычислении функции *frac(x)* отбрасываются все цифры до запятой. Например: *frac(-19.42)=-0.42*.

Результатом функции *round(x)* является ближайшее целое к числу x. Например: *round(1.25)=1*, *round(-12.84)=-13*.

Отметим, что в Паскале нет операции возведения в произвольную степень. Ее можно записать следующим образом: $x^y = x\ x^y = \left(e^{\ln x}\right)^y = e^{y \ln x}$ или x^y *=exp(y*ln(x))*, при этом число $x > 0$.

Если у нас небольшая степень, то лучше выполнить небходимое количество умножений или использавать функцию *sqr(x)*: x^5 *=sqr(sqr(x))*x*.

Структура программы. Программа на языке Паскаль состоит из заголовка, блока и заканчивается точкой. Блок в свою очередь содержит раздел описаний и раздел операторов.

Раздел операторов представляет собой последовательность операторов, разделенных точкой с запятой, и ограничивается служебными словами *begin*, *end*.

```
program имя [(имя файла)];
uses имя_подключаемого_модуля;
label метка, метка;
const имя_const = константа; имя const = константа;
type имя типа = тип;
var имя переменной,имя переменной:тип;
procedure <заголовок>;
  <блок>;
function <заголовок>;
  <блок>;
BEGIN
  <операторы>
END.
```

Раздел констант, раздел переменных, раздел процедур и функций составляют раздел описаний. Раздел описаний и раздел операторов составляют блок программы.

Порядок следования разделов строго определен, однако в некоторых описаниях языка порядок следований *const*, *label*, *type*, *var* произволен. В заголовке программы после служебного слова program указывается имя программы, не имеющее смысла внутри программы. После имени программы в круглых скобках следует список файлов (наборов данных, размещающихся на внешних устройствах), с которыми взаимодействует программа. Обычно используют имена стандартного входного *input* и выходного *output*: *program gons (input, output);*.

Во многих версиях языка имена стандартных файлов могут отсутствовать. Они принимаются по умолчанию: *program gons;*.

Кроме того в последних версиях языка слово *program* также может отсутствовать.

После заголовка программы следует раздел описания меток *label*. Любой оператор в программе можно выделить, поставив перед ним метку (целое число без знака, содержащее не более 4-х цифр). Метка от оператора отделяется двоеточием. Например:

25:read(x,y,z);.

Появление меток в программе дает возможность ссылаться на эти метки в специальных операторах управления (*goto*) и изменять естественный ход выполнения программы. Все метки должны быть описаны в разделе *label*.

```
program cons;
label
  1,25,100;
```

Этот раздел может отсутствовать, если в программе меток нет. За разделом меток следует раздел описания констант *const*.

Константы в программе должны быть представлены именем, тогда в разделе констант этим именам должны быть присвоены некоторые значения.

Например: *const p=3.1415926; m=3; n=4; s='заголовок таблицы';*.

Это позволяет сгруппировать в начале программы постоянные величины, зависящие от машины или характерные для данной задачи. Здесь их легче изменять, не изменяя саму программу.

Раздел описания типов (*type*) служит для определения простых и структурных типов данных, задаваемых пользователем. Каждая величина в программе должна быть сопоставлена с одним и только одним типом.

Тип переменной должен быть описан в специальном разделе *var*. Раздел описаний процедур и функций присутствует в программе, если программист, помимо стандартных процедур и функций, использует свои, являющиеся самостоятельными программным единицами, к которым осуществляется обращение из основной программы. В состав программы могут быть включены комментарии, т. е. тексты, поясняющие программу, но не влияющие на ход выполнения. Комментарии заключаются в специальные скобки. Например:

*goto 20;*выход из цикла (*выход из цикла*) или {выход из цикла}

Операторы в исполнительной части программы определяют, какие действия должны быть выполнены над данными. С каждым элементом данных обычно связывают имя и значение. Имя используется для обозначения элемента данных. В качестве имени в Паскале используется идентификатор – последовательность букв или цифр, начинающаяся с буквы. Хотя длина идентификатора не оговаривается, реально идентификатор не может переноситься со строки на строку. Обычно используют не более 80 символов. Кроме того, в большинстве реализаций Паскаля идентификаторы различаются по первым 6–8 символам. Все идентификаторы должны быть описаны в разделе описаний.

Оператор присваивания записывается в виде:

<имя переменной>:=выражение.

Оператор ввода. Для ввода информации с клавиатуры используется оператор *read*. Формат:

*read (*список переменных*);*

readln(a,b,c);

Переменные перечисляются через запятую. После выполнения программа останавливается и ожидает ввода информации с клавиатуры. Ввод осуществляется набором чисел, разделенных пробелом, и заканчивается нажатием клавиши ввода. Необходимо отметить, что вводимое значение должно соответствовать типам переменных и смыслу программы.

var b:real; a:char; … read(a,b);

Если у нас есть несколько команд *read*, разделенных другими командами, то во время выполнения первого оператора *read* можно ввести все переменные. Для остальных команд программа не останавливается.

В отличие от оператора *read* оператор *readln* после ввода всех указанных в операторе данных осуществляет переход к следующей строке.

Оператор вывода. Оператор вывода *write* служит для вывода информации на экран дисплея или принтера. Вид оператора:

write (список выражений);

Выражения в списке разделяются запятыми. В результате выполнения оператора *write* могут вычисляться значения выражений.

*write ('y= ', 2*y);*

Строки выводятся без всяких изменений, ограничивающие их апострофы не выводятся.

Наряду с оператором *write* используется оператор *writeln*. Отличие состоит в следующем: при выполнении оператора *write* информация выводится на экран и курсор остается в той же строке. Переход на следующую строку выполняется только тогда, когда выводимая строка полностью заполнена. В результате выполнения *writeln* после вывода информации курсор переводится на следующую строку. Результатом выполнения

write (1); writeln; write(-2,3);

будет

1

-2 3

Если не принять специальных мер, то значения вещественных переменных и выражений выводятся в форме с плавающей запятой. Для вывода вещественных чисел в форме с фиксированной точкой указывается формат после числа:

write(число:w:d);

Здесь: w – общая ширина поля числа после вывода; d – количество цифр после запятой (последние цифры округляются).

writeln(- 73.1:6:1);

Результат: (на экране клеток нет) будем считать, что одна клетка таблицы – одно знакоместо на экране

Если число литер в представлении выводимого значения оказывается меньше, чем w, то оно слева дополняется пробелами. Если w опускается, то назначается стандартная длина плоя, зависящая от конкретной ЭВМ.

Примеры операторов вывода:

1. Организация вывода значений переменных a, b, c **целого** типа – *var a,b,c:integer;*

read(a,b,c);
writeln(a,b,c);
Ввод:
12 34 98
Вывод:
123498

writeln(a:4,b:5); write(c);.
Ввод:
12 -34 6
Вывод:
 12 -34
6

writeln('a=',a:4,' b=',b:5);write(c);.
Ввод:
555 -899 -8
Вывод:
a= 555 b= -899
-8

Если количество указанных позиций недостаточно, то происходит автоматическое увеличение поля до необходимых размеров.

2. Организация вывода *вещественных* чисел – *var a,b,c:real;.*
read(a,b,c);
writeln(a,b,c); (длина поля не указана).
Ввод:
222.677777 888.4444444 -6.7
Вывод:
2.2267777700E+02 8.884444440E+0.2-6.7000000000E+00
writeln(a:6:3,b:7:2,c:4:1);.
Ввод:
222.677777 888.4444444 -6.7
Вывод:
222.678 888.44-6.7

Если в операторе вывода указано общее число позиций w и не указано количество позиций после запятой, то числа выводятся в экспоненциальной форме с шириной поля w.

writeln(a:6,b:7,c:4);.
Ввод:
222.677777 888.4444444 -6.7
Вывод:
2.2E+02 8.9E+02-6.7e+00

3. Организация вывода значений **символьного** типа – *var a,b,c:char;.*
read(a,b,c);
writeln(a:6,b:8,c:4);.
Ввод (без пробелов, т. к. пробел – это символ):
pro
Вывод:

p r o

writeln(a,b,c);.
Ввод:
pro
Вывод:
pro
4. При выводе значений **булевского** типа на печать выводится *true* или *false*.

```
program ww;
var a,b,c:integer;
BEGIN
  readln(a,b,c);
  writeln(a<b:7,b>c:4);
END.
```

Ввод:
1 2 3
TRUEFALSE

Пример 1. Рассмотрим программу для вычисления значений выражений: $b = a^3$; $c = \sin(a) + \sqrt{|d|}$.

```
program   umn;
var a,b,c,d: real;
BEGIN
  writeln('ввоð a d');
  readln(a,d);
  b:=a*a*a;  c:=sin(a)+sqrt(abs(d));
  writeln('otwet b= ',b:6:2,' c= ',c:8:1);
END.
```

Основные операторы

Оператор присваивания. Аналогичен оператору присваивания в алгоритмическом языке.

<имя переменной>:=выражение;

При вычислении значения выражения, стоящего в первой части оператора присваивания, учитывается тип данных и операции, которые над ним выполняются. Если в выражение входят данные целого типа, над которыми выполняются операции +,-,*, *div*, *mod*, то результат будет целочисленным. Если хотя бы один элемент данных относится к вещественному типу или встречается операция «деление», то результат будет вещественного типа. При этом будет сообщение об ошибке, указывающее на несоответствие типа.

Оператор условного перехода. Для записи команды ветвления используется команда *if...then...else;*.

если условие	*if* условие
то оператор1	*then* оператор1
иначе оператор2	*else* оператор 2
все	

Пример: *if x>=0 then y:=x else y:=-x ;.*
Перед служебным словом *else* знак «;» не ставится.

Можно сказать, что слова *then* и *else* действуют только на один оператор, т. е. до первого знака «;». Чтобы действие этих слов распространялось на несколько операторов, используют операторные скобки или составной оператор:

begin ..... end (перед *end* знак «;» может не ставиться).

Пример: *if x>0 then begin y:=1; z:=7 end else begin y:=2; z:=9 end;*.

В этом случае команда ветвления заканчивается последним словом *end* и после этого слова ставится знак «;».

Пример:

1)
```
if x>0 then
    y:=1
else begin y:=2; z:=9 end;
```
2)
```
if x>0 then begin y:=1; z=7 end
    else y:=2;
```

В этих примерах только после одного служебного слова используются операторные скобки.

В записи *if x>0 then y:=1 else y:=2;z:=7* значение 7 присвоится переменной *z* и при *x*>0 и при *x*<=0, так как оператор *z*:=7; уже не относится к оператору ветвления.

После служебного слова *if* без скобок записывается только простое условие. При записи составных условий обязательно каждое простое условие заключается в круглые скобки и отделяется пробелом от служебных слов:

if (x>1) and (x<7) then ...

При записи команды ветвления возникает проблема «болтающегося» *else* при оценке нескольких условий. Если не ввести дополнительного соглашения, то в записи

if условие
then if условие
then оператор 1
else оператор 2;

не понятно, к какому условному оператору, внутреннему или внешнему, относится *else*.

Поэтому в Паскале принято соглашение: если в *then* входит оператор *if*, то его лучше заключить в операторную скобку *begin...end*, даже если *if* единственный оператор в *then*-части.

```
if условие then
   begin
    if условие
       then оператор 1
       else оператор 3;
   end
else оператор 2;
```

Сокращенная форма оператора ветвления имеет вид:

```
if условие then оператор;
```

```
if условие                если
  then                    то
    begin                 оператор1
      оператор 1;             оператор2
      оператор 2;             оператор3
      оператор 3;
    end;                  все
```

Оператор выбора варианта. Оператор выбора дает возможность выполнить один или несколько операторов в зависимости от значения условия.

Его вид:
– полная форма:
case вариант *of*
<список меток>:оператор 1;
<список меток>:оператор 2;
.....
<список меток>:оператор *n*
else оператор n+1;
end;
– сокращенная форма:
case вариант *of*
<список меток>:оператор 1;
<список меток>:оператор 2;
.....
<список меток>:оператор *n*;
end;
case, *of*, *end* (выбор, из, конец) – служебные слова.
Вариант – выражение любого скалярного типа, кроме вещественного.
Оператор – любой оператор языка Паскаль.

Список меток – это список разделенных запятыми значений выражения или одно его значение. Эти константы должны иметь тот же тип, что и выражение, и называются метками выбора. Эта метка не обязательно целое число (может быть и символ), и она не описывается в разделе меток *label*, на нее нельзя ссылаться в операторе *goto*.

Оператор выбора исполняет тот оператор, одна из меток которого равна текущему значению условия. По окончании выполнения выборного оператора управление передается на конец.

```
program abs;
var
   x,g,y:real;
   nomer:integer;
BEGIN
   y:=0; x:=2.7; g:=-12.4; readln (nomer);;
   case nomer of
      2: y:=g;
      4: y:=g*x;
      6: y:=g*sqrt(abs(x));
      8: y:=g*sqr(sin(x)+12) end;
   writeln ('y=')
END.
```

Задача. Сколько дней в каждом месяце?

```
program bbb;
type
   mes=(январь, февраль, март, апрель, май, июнь, июль, август,
    сентябрь, октябрь, ноябрь, декабрь);
var
   mm: mes;
   g: 1900..2000;
   p: 28..31;
BEGIN
   case mm of
```

```
                май, январь, март, июль, август, октябрь, декабрь: p:=31;
                апрель, июнь, сентябрь, ноябрь: p:=30;
                февраль: if (g mod 4 = 0) and(g mod 100<>0) or (g mod
400=0)
                        then p:=29
                        else p:=28;
            end;
            writeln('число дней ', p);
        END.
```

Оператор безусловного перехода.

goto <метка>;

Метка – целое число без знака определения в разделе *label*. Оператор *goto* производит передачу управления к оператору, помеченному указанной меткой. Применение операторов безусловного перехода в языке Паскаль является необязательным и нежелательным, т. к. присутствие этого оператора в программе нарушает структурную целостность и наглядность. Такую программу становится трудно читать, отлаживать, модифицировать.

```
program rrr;
label 5;
var a,b:real;
BEGIN
    5: raedln(a);
    if a<0 then goto 5 ;
    b:=a*a;
    writeln(b)
END.
```

Операторы цикла.

В языке Паскаль существуют 3 оператора цикла.

1) Оператор цикла с параметром используется для организации цикла с известным числом повторений цикла (оператор для).

for <имя переменной>:=выр-ние1 *to* выр-ние2 *do*
begin
оператор 1;
оператор 2;
end (здесь значение переменной увеличивается на 1).

```
for i:=1 to 4 do
s:=s+1;
```

Переменная должна быть целого типа. Это оператор в модификации «Вниз к» имеет вид:

for i:=a1 downto a2 do оператор;

```
for i:=4 downto 1 do s:=s+1;
```

В этом случае при каждом новом выполнении оператора значение *a1* уменьшается на 1.

2) Оператор цикла с предусловием (цикл пока). Этот оператор имеет вид:

while условие
do оператор;.

while и *do* – служебные слова, условие – логическое (булево) выражение. Выполняется следующим образом: сначала вычисляется значение булева выражения. Если

это значение истинно (*true*), то выполняется оператор, следующий за служебным словом *do* (операторы могут быть заключены в операторные скобки *begin...end*), и снова происходит возврат к вычислению значения булева выражения. Так повторяется, пока значение булева выражения не станет ложным (*false*). Выполнение оператора, следующего за словом *do*, прекращается, поэтому такой оператор называется оператором с предусловием. Это означает также, что оператор, следующий за служебным словом *do*, может быть и не выполнен ни разу, если при первом же вычислении значение булева выражения будет ложным.

3) Оператор цикла с постусловием.

repeat
  оператор
until условие;

Этот оператор выполняется следующим образом: сначала выполняется оператор, следующий за служебным словом *repeat*, затем вычисляется значение булевого выражения, образующего условие. Если значение булевого выражения ложно, то происходит возврат к выполнению оператора, следующего за служебным словом *repeat*, и снова вычисляется булевое выражение. Так повторяется, пока значение булевого выражения остается ложным. Как только оно станет истинным, выполнение оператора цикла прекращается. В отличие от оператора *while...do* оператор, следующий за служебным словом *repeat* будет обязательно выполнен хотя бы один раз независимо от значения булевого выражения. Если в операторе *while...do* используется составной оператор, то он записывается с использованием служебных слов *begin* и *end*.

```
while x<0 do
   begin
      y:=sqr(x);
      x:=x+1;
   end;
```

Если такой же составной оператор используется в операторе *repeat*, то в его записи служебные слова *begin* и *end* могут быть опущены.

```
repeat
   read (b);
   if b<>0 then m:=m+b;
until b=0;.
```

Чтобы операторы цикла 1 и 2 выполнялись конечное число раз и не произошло зацикливания при построении цикла, необходимо предусмотреть, чтобы среди выполняемых операторов обязательно был оператор, выполнение которого влияло бы на значение булевого выражения. В этих примерах это оператор *x:=x+1*;

Для досрочного выхода из цикла, определяемого одним из операторов *while*, *repeat*, *for*, может быть использован оператор *exit*.

Дано натуральное число *n*. Найти наименьший простой сомножитель.

```
program somn;
var i,n:integer;
BEGIN
  write ('введите число ');
  readln (n);
  for i:=2 to n do
      begin
      if n mod i = 0    then
              begin
                write('наименьший сомножитель',i);
```

```
                    exit
                end;
            end;
        END.
```

Пример. Сумма цифр двузначного числа равна 6. Если к этому числу прибавить 18, то получиться число, записанное теми же цифрами, но в обратном порядке. Найдем это двузначное число:

```
program sls;
 var a,b,n:integer;
BEGIN
 writeln('варыянт 1');
 for a:=1 to 6 do
        for b:=0 to 5 do
                if (a+b=6) and (a*10+b+18=b*10+a) then n:=a*10+b;
 writeln(n);
 writeln('варыянт 2');
 a:=1;
 while a<=6 do
  begin
          b:=0;
           while b<=5 do begin
                if (a+b=6) and (a*10+b+18=b*10+a) then n:=a*10+b;
                b:=b+1
                        end;
 a:=a+1
 end;
 writeln(n);
 writeln('варыянт 3');
 a:=1;
 repeat
        b:=0;
        repeat
                if (a+b=6) and (a*10+b+18=b*10+a) then n:=a*10+b;
                b:=b+1
        until b>5;
        a:=a+1
 until a>6;
 writeln(n)
 END.
```

Пример. Составим программу проверки знаний таблицы умножения:

```
program tablica;
 const k=10;
 var a,b,c,d,i:integer;
BEGIN
 randomize; clrscr;
 for i:=1 to k do
        begin
                a:=random(8)+2; b:=random(8)+2; c:=a*b;
                writeln('Чаму роўна',a,'*',b,' ?');
```

```
            write('Увядзіце адказ'); readln(d);
            if c=d then writeln ('Правільны адказ')
                    else writeln('Памылка');
            writeln;
        end;
END.
```

Лекция 2 **Массивы**

При описании массива (таблица) указывается число его компонент, и это число остается постоянным. Для реализации прямого доступа к любой компоненте массива используются специальные обозначения. Все компоненты массива имеют общее имя – имя массива. Указание на конкретный элемент массива осуществляется с помощью индексов. Обозначение компоненты массива имеет вид:

имя массива [индекс, индекс,..., индекс].

k[1,4,2].

Количество индексов в обозначении компонент массива определяет размерность массива. Массив описывается в разделе описания переменных. При этом описание массива включает описание типа массива и типа индексов. Для описания используется конструкция

имя массива: *array [1..40] of* тип*;*

k: array [1..50] of real;

Одномерный массив *k*. Индексы изменяются от 1 до 50. Элементы массива имеют вещественный тип и будут обозначаться *k[8]*, *k[14]*. Для обозначения типа индекса в этом описании использован отрезочный тип данных. Этот тип не относится к стандартным, может образовываться непосредственно программистом. Описание этого типа задается с помощью констант, разделенных двумя точками: 1..50. Первая константа указывает нижнюю границу отрезка, а вторая – его верхнюю границу. Переменная, относящаяся к этому типу (в нашем случае переменная, обозначающая индекс массива), должна принимать значение того же типа, что и границы отрезка (*integer*). Причем эти значения должны принадлежать указанному отрезку. Если же значение переменной отрезочного типа выходит за пределы указанного отрезка, то возникает ошибка. Границы отрезка в описании переменной отрезочного типа не могут быть вещественного типа. Если компоненты массива имеют более одного индекса, то в описании массива должен быть описан тип каждого индекса.

```
matr:array [1..10,1..4] of real;
mat [1..4];
```

Тот факт, что элементы массива имеют явное обозначение, делает их равнодоступными в любой момент выполнения программы, но это налагает определенные ограничения на использование типа памяти ЭВМ. Обычно для этих целей используется основная память ЭВМ, которая называется памятью с произвольной выборкой, т. к. время выборки и записи информации в такой памяти одинаково для всех ее ячеек. Поэтому, если массив находится в основной памяти, его компоненты одинаково доступны. Однако, если массив так велик, что не вмещается в основной памяти, его хранят в виде отдельного файла на периферийных устройствах. Для обработки же вызывают в основную память ту часть массива, которая может там разместиться.

Пусть одномерный массив описан в разделе описания переменных так:

```
var
vector:array[1..50] of real;

for k:=1 to 50 do read(vector[k]);
```

Приведенный фрагмент программы плох тем, что он годится только для ввода таких массивов, которые содержат ровно 50 компонент Если же программа должна быть написана для работы с массивом, который при каждом новом исполнении программы может иметь другое число компонент, то целесообразно описать этот массив в разделе описаний переменных как массив с максимально возможным кол-вом компонент. Предположим, что кол-во компонент массива не может быть больше 100. Это может быть записано так:

```
program wwod;
const m=100;
var vector:array [1..m] of real;
  k:integer;
BEGIN
  for k:=1 to m do read (vector[k]);
END.
```

Задача. Найти максимальный элемент одномерного массива и его номер. Число элементов массива меньше 100.

```
program maxelem;
const k=100;
var
    x:array [1..k] of real ;
    max:real;
    n,i,jmax:1..k;
BEGIN
    write ('введите размерность массива n ');
    readln (n);
    for i:=1 to n do
        begin
            write('введите x[',i,'] ');
            readln (x[i]);
        end;
    max:=x[1];
    jmax:=1;
    for i:=2 to n do
        if x[i]>max then
            begin
                max:=x[i];
                jmax:=i;
            end;
    writeln('максимальный элемент', max:6:2);
    writeln('индекс ', jmax);
END.
```

Задача. Определим количество элементов одномерного массива, больших среднего арифметического его положительных элементов (обязательно есть такие элементы).

```
program asd;
const n=20;
 var x:array[1..n] of integer;
k,s,i:integer; u:real;
  BEGIN
 randomize;
 for i:=1 to n do
         begin x[i]:=random(40)-15; write(x[i]:7); end;
 writeln;
 s:=0; k:=0; for i:=1 to n do
           if x[i]>0 then begin s:=s+x[i];k:=k+1 end;
 u:=s/k;
 writeln('ср=',u:7:2);
```

```
k:=0; for i:=1 to n do if x[i]>u then k:=k+1;
writeln('ответ ',k);
END.
```

Задача. Найти максимальный элемент матрицы $m \times n$ и его индекс.

```
program maxelement;
const m=5;n=7;
var
  x:array [1..m, 1..n] of real;
  max:real;
  i,j,c,d:integer;
BEGIN
  writeln('размерность матрицы ',m,'x',n);
  for i:=1 to m do
      begin
          writeln('введите через пробел элементы строки ',i);
          for j:=1 to n do read (x[i,j]);
      end;
  max:=x[1,1];
  c:=1; d:=1;
  for i:=1 to m do
      for j:=1 to n do
          if x[i,j]> max then
              begin
                  max:=x[i,j];
                  c:=i; d:=j;
              end;
  writeln('max   строка столбец');
  writeln (max:5:2,c:4,d:7)
END.
```

Дана символьная матрица размера $m \times n$. Требуется получить последовательность $b_1, \text{К}, b_n$ из нулей и единиц, в которой $b_1 = 1$ тогда и только тогда, когда в i -м столбце число символов * не меньше числа "!".

```
program aaa;
const m=3;n=4;
type matr=array[1..m,1..n] of char;
var
    s:matr;
    i,j,k,c:integer;
    b:array[1..n] of char;
BEGIN
    for i:=1 to m do
        for j:=1 to n do
            begin
                writeln ('ww s[',i,' ',j,']');
                readln (s[i,j]);
            end;
    writeln; вывод в виде матрицы на экран
    for i:=1 to m do
        begin
```

```
            for j:=1 to n do write (s[i,j]:3);
            writeln;
        end;
    writeln;
    for j:=1 to n do
        begin
            k:=0; c:=0;
            for i:=1 to m do
                if s[i,j]='*' then k:=k+1
                    else if s[i,j]='!' then c:=c+1;
            if k>=c then b[j]:='1' else b[j]:='0';
        end;
    for j:=1 to n do write (b[j]:7);
    writeln;
END.
```

Лекция 3 Процедуры и функции

В языке Паскаль предусмотрены средства, позволяющие оформить последовательность операторов как подпрограмму. Это бывает необходимо, когда такая последовательность встречается уже более 2-х раз в программе или когда имеется возможность использовать некоторые фрагменты уже написанных ранее программ.

Для оформления подпрограмм, последовательности операторов присваивается имя, и в дальнейшем при необходимости выполнить последовательность в программе упоминают имя этой последовательности. В языке Паскаль такую поименованную последовательность операторов называют процедурой.

Если в результате выполнения поименованной последовательности операторов получается только 1 результат, то имя этой последовательности можно использовать в выражении, и тогда такую последовательность называют функцией. В языке Паскаль имеется целый ряд встроенных в компилятор и распознаваемых без дополнительного описания стандартных функций, например, математические функции sin и cos.

Стандартные процедуры и функции. Для IBM при работе со встроенными процедурами необходимо в самом начале после слова *program* записать служебное слово *uses*, после которого указать имена подключаемых модулей например,

uses crt;

Раздел *uses* описывает список имен, подключаемых стандартных и пользовательских модулей.

crt – содержит средства управления дисплеем и клавиатурой ЭВМ;
printer – открывает доступ к печатающему устройству;
graph – позволяет использовать графику.

Стандартные процедуры или функции:

gotoxy (x,y) – перемещает курсор в строку *x* позиции *y* (*x*,*y* – целые);
clrscr – очистка экрана;
delay (m_s) – задержка программы на m_s миллисекунд (m_s – *integer*);
pred (x) – предыдущее значение *x* (*x* – целые);
succ(x) – последующее число после *x*;
random(x) – случайное число (0–*x*);
keypressed – функция, результат *true* – клавиша нажата, *false* – клавиша не нажата.

Помимо стандартных функций, программист имеет право вводить новые функции. Для этого в программе необходимо задать правило вычисления значения функции, т. е. провести ее описание. Процедуры также должны описываться. Описание процедуры и функции располагается в программе после описания переменных. Сами эти описания не предписывают выполнения действий, как операторы. Чтобы процедура была выполнена, в программе должен быть предусмотрен оператор вызова процедуры. Функция будет выполнена, если ее имя встретится в программе в каком-либо выражении.

Описание процедуры начинается с заголовка. За заголовком следует текст описания, во многом аналогичный тексту программы, т. к. строится по тем же правилам.

procedure имя процедуры (имя аргумента: тип; имя аргумента:тип**;** ***var*** имя результата: тип);

```
var .....
begin
.....
end;.
```

Без служебного слава var перед именем результата значение результата из процедуры не возвращается. Описание процедуры заканчивается служебным словом *end*, за которым следует точка с запятой.

```
procedure swap (var x,y:real);        алг обмен (вещ x,y)
var t:real;                           дано x,y
                                      надо x,y
begin                                 нач вещ t
  t:=x; x:=y; y:=t;                       t:=x; x:=y; y:=t
end;                                  кон
```

Для вызова процедуры в тексте программы указывается ее имя, а в скобках через запятую перечисляются имена переменных или конкретное числовое значение, необходимое для этой процедуры.

Пример.

```
program vv;
var x,y:real;
BEGIN
   x:=5; y:=2.3;
   swap (x,y);
   writeln(x,y)
END.
```

Ответ: x=2.3; y=5.

Процедура может и не содержать параметров. В этом случае заголовок имеет вид:

procedure имя процедуры;

Оператор вызова такой процедуры имеет вид:

```
begin
  оператор;
  ostatok;
  ........
end.
procedure ostatok
begin
  g:=a div b;
  r:=a mod b;
end;
```

В результате ее выполнения получим значение переменных *g* и *r* (частного и остатка от деления целых чисел *a* и *b*).

При этом, конечно, переменные *a*, *b*, *g*, *r* должны быть описаны в программе, содержащей описание этой процедуры, а значения переменных *a* и *b* должны быть заданы до выполнения операторов процедуры. Описание функций аналогично описанию процедуры.

Заголовок описания функции имеет вид:

function имя функции (имя аргумента:тип):тип функции;

```
var ...
begin
...
end;.
```

Например:

```
function summa (var x,y:real):real;,
function summa (x,y:real):real;.
```

В этом заголовке тип, стоящий после круглой скобки, определяет тип переменной имени функции. Отличие от процедуры состоит также в том, что в тексте описания функции ***должен обязательно присутствовать оператор присваивания вида: имя функции:=выражение*** (summa:=s)**.**

```
program sum;
BEGIN
  sum:=x+y
END.
```

Программа вычисления факториала

```
program factorial;
var i,k:integer;
function fact (n:integer):integer;
begin
  if n<=1 then fact:=1
      else fact:=n*fact(n-1);
end;
BEGIN
  write ('ввод символа i '):readln(i);
  for k:=1 to i do
      writeln(fact(k));
END.
```

Задача 1. Оформить в виде процедуры алгоритм вычисления степени $y = x^n$ с натуральным показателем.

```
program ap;
var n:integer; x,y:real;
procedure s1 (k:integer; e:real; var d:real);
var i:integer;
begin
    d:=1;
    for i:=1 to k do d:=d*e;
end;
BEGIN
    writeln ('введи показатель степени');
    readln (n);
    writeln('введи значение x');
    readln(x);
    s1(n,x,y);
    writeln('y=',y)
END.
```

Задача 2. Оформить алгоритм вычисления степени $y = x^n$ в виде процедуры без параметров.

```
program p2;
var n:integer;
  x,y:real;
procedure s2;
var
  i:integer;
begin
  y:=1;
  for i:=1 to n do y:=y*x;
```

```
end;
BEGIN
  writeln ('введите показатель степени');
  readln(n);
  writeln('введите значение x');
  readln(x);
  s2;
  writeln('y=',y)
END.
```

В этом примере *x*, *y* – глобальные переменные по отношению к процедуре *s2*, а переменная *i* там – локальная.

Задача 3. Оформить в виде функции алгоритм вычисления степени $y = x^n$.

```
program p3;
var n:integer;
  x,z:real;
function s3 (n:integer; x:real):real;
var
  i:integer; y:=real;
begin
  y:=1;
  for i:=1 to n do y:=y*x;
  s3:=y;
end;

function s3:real;
var i:integer;
  y:real;
begin
  y:=1;
  for i:=1 to n do
  y:=y*x;
  s3:=y
end;

BEGIN
  writeln ('введи n');   readln(n);
  writeln('введи x');    readln(x);
  z:=s3(n,x);
  z:=s3;
  writeln('y=',z)
END.
```

Задача 3. Определите с помощью функции пользователя периметр восьмиугольника.

```
program tem;
 var i:integer;
    x,y:array[1..8] of real;     p:real;
function dl(x1,y1,x2,y2:real):real;
  begin
    dl:=sqrt(sqr(x2-x1)+sqr(y2-y1));
  end;
```

```
BEGIN
 writeln('Введите координаты 8 точек  ');
 p:=0;
 for i:=1 to 8 do
   begin
    readln(x[i],y[i]);
    if i>1 then  p:=p+dl(x[i-1],y[i-1],x[i],y[i]);
   end;
 p:=p+dl(x[1],y[1],x[8],y[8]);
 writeln;
 writeln('Периметр равен ',p);
END.
```

Главная программа может содержать описание нескольких процедур и функций. Любая из них может быть вызвана в главной программе, или ее вызов может содержаться в теле другой процедуры, описанной в программе. Описание процедуры или функции может быть также приведено в теле другой процедуры или функции, т. е. возможны вложенные процедуры и функции. Главная программа может получать начальные данные, с которыми ей предстоит работать извне, читая их из файла *input*. Процедуры и функции обычно получают такие начальные данные из той программы или процедуры, которая их вызывает. Передача этих данных осуществляется через параметры. В операторе вызова процедуры или при вызове функции указываются фактические параметры, подставляемые вместо формальных параметров и используемые при вычислении. Аналогично результат может быть передан в вызывающую программу через соответствующий параметр. Процедуры и функции могут также получать данные из файлов *input* и выдавать результат в файле *output*. Передача значений между вызывающей программой и процедурой может осуществляться также с помощью глобальных переменных.

Локальные и глобальные переменные. Переменные описываются в программе в разделе описания переменных. Если же программа содержит описание процедуры, то некоторые переменные могут быть описаны в этой процедуре. Переменные, описанные только в главной программе, называются глобальными. Такими переменными можно пользоваться и в главной программе, и в процедуре. Переменные, описанные в процедуре, называются локальными. Этими переменными можно пользоваться только в процедуре. Вне тела процедуры значения таких переменных не определены.

```
program sum;
var p,q,s:integer;
procedure now;
begin
   p:=1;
   q:=1;
end;
BEGIN
   p:=0;
   q:=0;
   now;
   s:=p+q;
   write (s);
```

```
END.
```

В результате выполнения оператора вызова процедуры *now* глобальным переменным *p* и *q* присваивается значение 1. Поэтому оператор *write(s)* выдаст значение 2. *p* и *q* – глобальные.

Рассмотрим пример, где *p* является локальной и *q* – глобальной.

```
program sum ;
var p,q,s:integer;
procedure now;
var p:=integer;
begin
  p:=1;
  q:=1;
end;
BEGIN
  p:=0;
  q:=0;
  now; s:=p+q;
  write (s);
END.
```

Теперь в результате выполнения оператора вызова процедуры значения глобальной переменной *q* и локальной переменной *p* будут равны 1. Но значения локальной переменной *p* теперь недоступно главной программе, т. к. оно считается не определенным вне процедуры. Поэтому при выполнении оператора *s:=p+q* в качестве значения *p* будет взято значение глобальной переменной *p*, которое равно 0, поэтому будет *s=1*.

Если *p* и *q* будут описаны в процедуре, то они будут локальные внутри процедуры и не будут иметь ничего общего с глобальными переменными *p* и *q*, которые не описаны в основной программе. В результате выполнения такой программы будет напечатано 0.

Рассмотренные примеры показывают, что процедуры могут использовать и изменять значение глобальных переменных. Если процедура изменяет значение глобальной переменной, то говорят, что она имеет побочный эффект. Если такой эффект не был предусмотрен программистом, то будет ошибка и ее очень трудно обнаружить. Чтобы избежать непредусмотренных эффектов, то не забывайте описывать вновь введенные переменные и не используйте одинаковых имен переменных в главной программе и процедурах и функциях.

Задача 4. В трех квадратных матрицах одинаковой размерности определить сумму элементов в первом и последнем столбце.

```
program gl;
const n=3;
type mas = array[1..n,1..n] of integer;
var i,j:integer; a,b,c:mas; a1,b1,c1:real;
procedure inpmas(var z:mas);
begin
  for i:=1 to n do
      begin
        for j:=1 to n do
            begin
               z[i,j]:=random(20)-5; write (z[i,j]:9)
            end;
        writeln;
      end;
```

```
  writeln;
end;            {end inpmas}
procedure coun(z:mas;var s:real);
begin
  s:=0;
  for i:=1 to n do
      begin
          s:=s+z[i,1]+z[i,n];
      end;
end;            {end counddig}
BEGIN
  inpmas(a); inpmas(b); inpmas(c);
  coun(a,a1); writeln('sum a=',a1:7:2);
  coun(b,b1);writeln('sum b=',b1:7:2);
  coun(c,c1); writeln('sum c=',c1:7:2);
END.
```

Лекция 4 **Строки**

Строка – это последовательность символов. Количество символов в строке может меняться от 0 до 255. Переменная строкового типа описывается или через описание типа, или непосредственно в разделе описания переменных. Для описания переменной строкового типа используется служебное слово *string*. В квадратных скобках указывается её длина.

```
type slovo = string[50];
var x,y:slovo; var x:string[50];
```

В памяти ЭВМ под строку отводится максимальная длина + 1 байт**.** Этот байт располагается в начале слова и указывает длину строки**.**

В рассмотренных примерах под переменные *x*, *y* отводится по 51 байту**.** Если *m* – текущая длина строки, а *b* – начальный адрес, с которого начинает располагаться строка *k*, то в ячейке *b* будет число *m*, *b*+1 – 1-й символ, *b*+2 – 2-й символ, *b*+*m* – *m*-й символ, *b*+*m*+1 – свободная ячейка, *b*+*k* – свободная ячейка**.**

Над строковыми данными можно выполнять операции:

1) сложение (сцепление, конкатенация);

2) отношение (равно, не равно, <,>, <=, >=)**.** Эти операции выполняются после операции сцепления**.** Операции отношения результатом имеют истинно или ложно**.**

Если длина строкового выражения превышает максимальную длину, отведенную для переменной, то лишние символы отбрасываются**.**

Для присваивания строковой переменной результата строкового выражения используется оператор присваивания**.**

```
A=string[4];
A:='не'+'книга';.
```

Результат: ‘некн’**.**

В выражение могут входить не только строковые, но и символьные величины**.**

```
a,c:string[5];
b,d:char;
a:='лон'; b:='с'; c:=b+a {'слон}';
d:=a;
```

(зависнет)

К отдельному символу можно обратится по номеру:

```
a:='книга';
a:=a[3];.
```

Ответ: ‘и’**.**

```
a:='книга';
a:=a[2]+a[3]+a[1]+a[5];.
```

Ответ: ‘ника’.

В работе со строками можно использовать процедуры:

1) *delete (str, poz, n)* – из строки *str* удалить *n* символов с позиции *poz*: *delete (a,2,1).*

```
program ad;
var a:string[10];
begin
  writeln('вводите a'); readln(a);
  delete(a,3,4); writeln(a);.
```

Ответ: кн.

2) *insert (str1, str2, poz).*

Вставка строки *str1* в строку *str2*, начиная с позиции *poz*.

```
begin
  writeln('введите a'); readln(a);
  writeln('введите c'); readln(c);
```

```
insert(c,a,3); writeln(a);
```

Если в этом фрагменте переменная *a* имеет *string[2]*, то будет ошибка**.** Лучше не писать *read (a,c)* (*a,c:char*), т**.** к**.** при вводе мы должны отделять пробелом, а пробел – строковая величина, и машина не воспринимает *c***.**

3) *str(ib,st)***.** Преобразует числовую величину *ib* в строковую *st***.** Числовая величина *ib* может указываться в формате:

```
str(-40,a);
str(-40:6,a);
a='-40';
a=' -40';
```

4) *val (str,per,cod)*. Строковую переменную *str* переводит в числовую переменную *per* (*cod* – номер позиции; *str* (исходная строковая величина) не должна содержать пробелов в начале и в конце; *per* – переменная типа *real, integer* или *byte*; *cod* – целочисленная переменная.

Если процедура выполнилась, то значение *cod* равно 0, в противном случае это номер позиции, в которой произошла ошибка. При этом значение *per* не определено**.**

```
program cd;
var
  a:string[6]; k,m:integer; d:real;
begin
  val('400',k,m);
  writeln(k-1, ,m);.
```

Ответ: 399 0**.**

```
a:='12.2';
val(a,d,m)val(a,k,m);
writeln(d);          {Здесь будет ошибка: несоответствие}
```

Ответ: 1,22000E+1 типов**.**

```
a:='1 2';
val(k,k,m); writeln(k, , m);.
```

На экране вместо *k* будет любое число, а вместо *m* будет число 2**.**

Функции для работы со строками.

1) *length (str) –* длина строковой переменной. Примеры: *length(a); length('123');.*

Ответ: 3**.**

2) *pos (str1, str2)***.** Обнаруживает первое появление подстроки *str1* в строке *str2***.** Значение – номер позиции, с которой входит *str1***.** Если вхождения нет, то значение функции равно 0**.** Лучше использовать переменную целого типа**.**

Пример:

```
var
   a:string[10]; i:integer;
begin
   a:='wapa'; i:=pos('p',a);
   write(i);.
```

Ответ: 3**.**

3) *copy (str,poz, n)***.** Вырезает из строки *str* подстроку длиной *n* символов, начиная с позиции *poz***.** Следует обратить внимание на то, что тип результата у этой функции *string*.

Задача. Распечатать слово по буквам**.**

```
program buk;
const n=10;
var a:string[n]; i:integer; c:char;
```

```
BEGIN
  writeln('введите a'); readln(a);
  for i:=1 to length(a) do
      begin c:=a[i];
         writeln(c);
      end;
END.
```

Задача. Узнать, сколько раз в слове *b1* встречается 3-я буква *a1*.

```
program ser;
const n=10;
var a1:string[n]; b1:string[n]; i;s:integer; c:string[1];
BEGIN
  writeln ('введите a1');readln(a1);
  writeln('введите b1'); readln(b1);s:=0;
  c:=a1[3];
  for i:=1 to length (b1) do
      if c=copy (b1,i,1) then s:=s+1;
  writeln('s=',s);
END.
```

Пример. В данном слове заменим сочетание букв "ab" многоточием:

```
program slr;
var  s,t:string;   i:integer;
BEGIN
      writeln('Введите предложение'); readln(s);
      t:=''; i:=1;
      while i<=length(s) do
            if copy(s,i,2)='ab' then begin t:=t+'...'; i:=i+2 end
                               else begin t:=t+s[i]; i:=i+1 end;
      writeln(t);
END.
```

Пример. Дан текст до 10 слов, в котором слова (от 1 до 8 символов) могут быть отделены одной или несколькими звездочками. Удалим звездочки в начале и в конце текста и оставим между словами одну звездочку, все слова, содержащие три символа, заменим на слово "три":

```
program gfhd;
const s='****a*bc***efg**k*';
var t,w,b:string;
          i,k,n:integer; sl:array[1..10] of string[8];
BEGIN
t:=s;
writeln('------зыходны тэкст------'); writeln(s);
                  {удаляем все звездочки в начале текста}
while(t[1]='*') and (length(t)>0) do delete(t,1,1);
                 {удаляем все звездочки в конце текста}
while (t[length(t)]='*') do delete(t,length(t),1);
k:=0  ;  {счетчк звездочек между словами}
n:=0;{ счетчик слов}
b:='';
```

```
t:=t+'*'; {добавляем * для получения последнего слова}
{получение массива слов из текста}
for i:=1 to length(t) do
        if t[i]<>'*' then begin b:=b+t[i]; k:=0; end
                        else begin k:=k+1;
                                        if k=1 then begin n:=n+1; sl[n]:=b; b:=''; end;
                                end;
writeln('-----вывод всех слов на экран-----');
for i:=1 to n do writeln(sl[i]);
{замена  слов}
t:='';
for i:=1 to n do
        if length(sl[i])=3 then t:=t+'тры'+'*'
                                else t:=t+sl[i]+'*';
{удаляем звездочку в конце текста}
delete(t,length(t),1);
writeln('------ответ------');
writeln(t);
END.
```